An Intuitive Formal Approach to Dynamic Workflow Modeling and Analysis

Jiacun Wang[1], Daniela Rosca[1], William Tepfenhart[1], Allen Milewski[1], and Michael Stoute[2]

[1] Department of Software Engineering, Monmouth University, West Long Branch, NJ 07762, USA
`{jwang, drosca, btepfenh, amilewsk}@monmouth.edu`

[2] Intellipro, Inc. 255 Old New Brunswick Road, Piscataway, NJ 08854, USA
`jason@intellipro.com`

Abstract. The increasing dynamics and the continuous changes of business processes raise a challenge to the research and implementation of workflows. The significance of applying formal approaches to the modeling and analysis of workflows has been well recognized and many such approaches have been proposed. However, these approaches require users to master considerable knowledge of the particular formalisms, which impacts the application of these approaches on a larger scale. This paper presents a new formal, yet intuitive approach for the modeling and analysis of workflows, which attempts to overcome the above problem. In addition to the abilities of supporting workflow validation and enactment, this new approach possesses the distinguishing feature of allowing users who are not proficient in formal methods to build up and dynamically modify the workflow models that address their business needs.

1 Introduction

Although workflow is an old concept [13] its research and implementation are gaining momentum due to the increasing dynamics and the continuous changes of the market places. The business environment today is undergoing rapid and constant changes. The way companies do business, including the business processes and their underlying business rules, ought to adapt to these changes flexibly with minimum interruption to ongoing operations [3,5]. This flexibility becomes of a paramount importance in applications such as an incident command system (ICS). An ICS would support the activities necessary for the allocation of people, resources and services in the event of a major natural or terrorist incident. An ICS would need to deal with frequent changes of the course of actions dictated by incoming events, a predominantly volunteer-based workforce, the need to integrate various software tools and organizations, a highly distributed workflow management.

Dealing with these issues generates many challenges for a workflow management system. The need of making many ad-hoc changes calls for an on-the-fly verification of the correctness of the modified workflow. This cannot be achieved without an underlying formal approach of the workflow, which does not leave any scope for ambiguity and sets the ground for analysis. Yet, since our main users will be volunteers from various backgrounds, with little computer experience, we need to provide a tool with highly intuitive features for the description and modification of the workflows.

A number of formal modeling techniques have been proposed in the past decades [1, 6, 8, 10, 11, 12]. Van der Aalst [9] identifies three reasons for using Petri Nets in workflow modeling. Firstly, Petri Nets possess formal semantics despite their graphical nature. Secondly, instead of being purely event-based, Petri Nets can explicitly model states, and lastly it is a theoretical proven analysis technique. Other than Petri Nets, techniques such as state charts have also been proposed for modeling WFMS [4]. Although state charts can model the behavior of workflows, they have to be supplemented with logical specification for supporting analysis. Singh et al [7] use event algebra to model the inter-task dependencies and temporal logic. Attia et al [2] have used computational tree logic to model tasks by providing their states together with significant events corresponding to the state transitions (start, commit, rollback etc) that may be forcible, rejectable, or delayable.

As indicated in [10], it is desirable that a business process model can be understood by the various stakeholders involved in an as straightforward manner as possible. Unfortunately, a common major drawback that all the above formal approaches suffer is that only users who have the expertise in these particular formal methods can build their workflows and dynamically change the business rules within the workflows. For example, in order to add a new task to a Petri-net based workflow, one must manipulate the model in terms of transitions, places, arcs and tokens, which can be done correctly and efficiently only by a person with a good understanding of Petri-nets. This significantly affects the application of these approaches on a large scale. This paper attempts to define a new formalism for the modeling and analysis of workflows, which, in addition to the abilities of supporting workflow validation and enactment, possesses the distinguishing feature of allowing users who are not proficient in formal methods to build up and dynamically modify the workflows that address their business needs.

The paper is organized as follows: Section 2 presents the new workflow formalism (WIFA – Workflows Intuitive Formal Approach), its state transition rules and its modeling power. Section 3 introduces well-formed workflows and how to build up a well-formed workflow. Section 4 gives a brief description of our tool for workflows modeling and analysis. Section 5 presents conclusions and ideas for the continuation of this work.

2 The WIFA Workflow Model

In general, a workflow consists of processes and activities, which are represented by well-defined *tasks*. The entities that execute these tasks are humans, application programs or database management systems. These tasks are related and dependent on one another based on business policies and rules [4]. In this section, we introduce the WIFA workflow model which captures tasks and relations among them in a workflow. We also define a set of state transition rules to facilitate the analysis of the dynamic behavior of a workflow.

2.1 WIFA Workflow Model Definitions

The control dependencies among tasks contain the order in which they can execute. Two tasks are said to have *precedence constraints* if they are constrained to execute in some order. As a convention, we use a partial-order relation <, called a *precedence relation*, over the set of tasks to specify the precedence constraints among tasks. A task T_i is a *predecessor* of another task T_j (and T_j a *successor* of T_i) if T_j cannot begin execution until the execution of T_i completes. A short-hand notation for this fact is $T_i < T_j$. T_i is an immediate predecessor of T_j (and T_j an immediate successor of T_i) if $T_i < T_j$ and there is no other task T_k such that $T_i < T_k < T_j$. We denote this fact with notation $p_{ij} = 1$. Naturally, the fact that T_i is *not* an immediate predecessor of T_j is denoted by $p_{ij} = 0$. Two tasks are independent when neither $T_i < T_j$ nor $T_j < T_i$. A classic way to represent the precedence constraints among tasks in a set T is by a directed graph $G = (T, <)$, in which each vertex represents a task in T, and there is a directed edge from vertex T_i to vertex T_j if T_i is an immediate predecessor of T_j. The graph is called a *precedence graph.*

Definition 1 (preset of a task): The preset of a task T_k, denoted by $*T_k$, is

$$*T_k = \{T_i \mid p_{ik} = 1\}.$$

Definition 2 (postset of a task): The postset of a task T_k, denoted by T_k*, is

$$T_k* = \{T_i \mid p_{ki} = 1\}.$$

Basically, the preset of a task is the set of all tasks that are immediate predecessors of the task, while the postset of a task is the set of all tasks that are immediate successors of the tasks. If $|T_k*| \geq 1$, then the execution of T_k might trigger multiple tasks. Suppose $\{T_i, T_j\} \subseteq T_k*$. There are two possibilities: (1) T_i and T_j can be executed simultaneously, and (2) only one of them can be executed, and the execution of one will disable the other due to the conflict between them. We denote the former case by $c_{ij} = c_{ji} = 0$, and the latter case by $c_{ij} = c_{ji} = 1$.

If $|*T_k| \geq 1$, then based on the aforementioned classic precedence model, the execution of T_k won't start until *all* of its immediate predecessors are executed. This precedence constraint is also called *AND precedence constraint.* An extension to this classic precedence model is to allow a task to be executed when *some* of its immediate predecessors are executed. This loosens the precedence constraints to some extent, and the loosened precedence constraint is also called *OR precedence constraint.* Obviously, the *OR* precedence model provides more flexibility than the classic *AND* precedence model in describing the dependencies among tasks. So in this paper, the *OR* precedence model is adopted. The *AND* precedence model can be viewed as a special case of the *OR* precedence model.

Suppose $*T_k = \{T_{k1}, T_{k2}, \dots T_{kn}\}$, $n \geq 1$. Define $A(T_k) = \{A_1, A_2, \dots A_h\}$, $h \geq 1$ such that

1) $A_i \subseteq *T_k$, $i = 1, 2, \dots, h$, i.e. $A(T_k)$ is a set of subsets of $*T_k$.
2) $A_i \neq A_j$, $\forall i \neq j$, $i, j \in \{1, 2, \dots, h\}$, i.e. these subsets are all different.

3) T_k is executable if and only if all tasks in any $A_i \in A(T_k)$ are executed. In other words, T_k can be triggered by any subset in $A(T_k)$, but only after all tasks in that subset are executed.

The set $A(T_k)$ is used to specify the pre-condition set for T_k to become executable.

The state of a workflow can be described as an array whose elements are the states of all individual tasks in the workflow. Denote by S a state of a workflow, then $S = (S(T_1), S(T_2), \ldots, S(T_m))$.

Now we are ready to formally define our WIFA workflow model.

Definition 3 (workflow): A workflow is $WF = (T, P, C, A, S_0)$, where

1) $T = \{T_1, T_2, \ldots, T_m\}$ is a set of *tasks*, $m \geq 1$.
2) $P = (p_{ij})_{mxm}$ is the *precedence matrix* of the task set. If T_i is the direct predecessor of T_j, then $p_{ij} = 1$; otherwise, $p_{ij} = 0$.
3) $C = (c_{ij})_{mxm}$ is the *conflict matrix* of the task set. $c_{ij} \in \{0, 1\}$ for $i = 1, 2, \ldots m$ and $j = 1, 2, \ldots m$.
4) $A = (A(T_1), A(T_2), \ldots, A(T_m))$ defines *pre-condition set* for each task. $\forall T_k \in T$, $A(T_k)$: $*T_k \rightarrow 2^{*T_k}$. Let set $A' \in A(T_k)$. Then $T_i \in A'$ implies $p_{ik} = 1$.
5) $S_0 \in \{0, 1, 2, 3\}^m$ is the *initial state* of the workflow.

Definition 4 (state values): Denote a state of the WF by $S = (S(T_1), S(T_2), \ldots, S(T_m))$, where $S(T_i) \in \{0, 1, 2, 3\}$.

1) $S(T_i) = 0$ means T_i is *not executable* at state S and *not executed previously*.
2) $S(T_i) = 1$ means T_i is *executable* at state S and *not executed previously*.
3) $S(T_i) = 2$ means T_i is *not executable* at state S and *executed previously*.
4) $S(T_i) = 3$ means T_i is *executable* at state S and *executed previously*.

By the definition of state values, at any state, only those tasks whose values are either 1 or 3 can be selected for execution. Suppose task T_i at state S_a is selected for execution, and the new state resulted from the execution of T_i is S_b, then the execution of T_i is denoted by $S_a(T_i)S_b$.

Now we can have a more accurate explanation on the conflict matrix C and the precondition set A of a task. Let tasks T_i, T_j and $T_k \in T$ with $p_{ki} = p_{kj} = 1$. Suppose there are three states S_a, S_b and S_c such that either $S_a(T_i) = S_a(T_j) = 1$ or $S_a(T_i) = S_a(T_j) = 3$, and $S_a(T_i)S_b$ and $S_a(T_j)S_c$.

1) If $S_a(T_i) = S_a(T_j) = 1$, then $c_{ij} = c_{ji} = 1$ implies $S_b(T_j) = S_c(T_i) = 0$, and $c_{ij} = c_{ji} = 0$ results in $S_b(T_j) = S_c(T_i) = 1$.
2) If $S_a(T_i) = S_a(T_j) = 3$, then $c_{ij} = c_{ji} = 1$ implies $S_b(T_j) = S_c(T_i) = 2$, and $c_{ij} = c_{ji} = 0$ results in $S_b(T_j) = S_c(T_i) = 3$.

On the other hand, suppose $A(T_k) = \{A_1, A_2, \ldots A_h\}$, $h \geq 1$. Then $S_a(T_k) \in \{1, 3\}$ if $\exists A_i \in A(T_k)$ such that $S_a(T_j) = 2$ for $\forall T_j \in A_i$.

Definition 5: (initial state) At the initial state S_0, for any task $T_i \in T$, if there is no T_j such that $p_{ji} = 1$, then $S_0(T_i) = 1$; otherwise $S_0(T_i) = 0$.

Note that tasks that have no predecessor do not need to wait for any other task to execute first. In other words, these tasks are executable immediately. We assume that there is always such kind of tasks in a workflow. They are the initial triggers or "starting" tasks of workflows. In Definition 1 there is no restriction on the preset and postsets of tasks. Therefore, there may be multiple tasks whose presets are empty, and there may be multiple tasks whose postsets are empty. In other words, this formalism supports multiple "starting" tasks and "ending" tasks in a workflow.

2.2 State Transition Rules

The dynamics of a workflow can be captured by state transitions. Of course, state transitions should be guided by a set of state transition rules. In this subsection, we define the rules.

Definition 6: (state transition rules) If $S_a(T_i)S_b$, then $\forall\ T_j \in T$,

1) If $T_j = T_i$ then $S_b(T_j) = 2$;
2) If $T_j \neq T_i$ then the state value of T_j at new state S_b depends on its state value at state S_a. We consider four cases:

   Case A – $S_a(T_j) = 0$:
   If $p_{ij} = 1$ and $\exists A' \in A(T_j)$ such that $S_b(T_k) = 2$ for any $T_k \in A'$, then $S_b(T_j) = 1$; otherwise $S_b(T_j) = 0$.

   Case B – $S_a(T_j) = 1$
   If $c_{ij} = 0$ then $S_b(T_j) = 1$; otherwise $S_b(T_j) = 0$.

   Case C – $S_a(T_j) = 2$
   If $p_{ij} = 1$ and $\exists A' \in A(T_j)$ such that $S_b(T_k) = 2$ for any $T_k \in A'$, then $S_b(T_j) = 3$; otherwise $S_b(T_j) = 2$.

   Case D – $S_a(T_j) = 3$
   If $c_{ij} = 0$ then $S_b(T_j) = 3$; otherwise $S_b(T_j) = 2$.

According to the above state transition rules, a task's state value at a given state other than the initial state is 0 iff one of the following holds:

1) Its state value is 0 in the previous state, and it is not the successor of the task which is just executed.
2) Its state value is 0 in the previous state, and it is the successor of the task which is just executed, but for each of its precondition sets there is at least one task that is not executed.
3) Its state value is 1 in the previous state but it conflicts with the task which is just executed.

A task's state value at a given state other than the initial state is 1 iff one of the following holds:

1) Its state value is 0 in the previous state, it is the successor of the task which is just executed, and in at least one of its precondition sets all tasks are executed.
2) Its state value is 1 in the previous state and it does not conflict with the task which is just executed.

A task's state value at a given state other than the initial state is 2 if and only if one of the following holds:

1) It is just executed.
2) Its state value is 2 in the previous state, and it is not the successor of the task which is just executed.
3) Its state value is 2 in the previous state, and it is the successor of the task which is just executed, but for each of its precondition sets there is at least one task that is not executed.
4) Its state value is 3 in the previous state but it conflicts with the task which is just executed.

A task's state value at a given state other than the initial state is 3 if and only if one of the following holds:

1) Its state value is 2 in the previous state, it is the successor of the task which is just executed, and there is at least one of its precondition sets in which every task is executed.
2) Its state value is 3 in the previous state and it does not conflict with the task which is just executed.

Note that a state value can increment from 0 to 1, from 1 to 2 or from 2 to 3; it can also decrement from 1 to 0 or from 3 to 2. But it cannot decrement from 2 to 1. Fig. 1 illustrates possible state value changes for a given task when a workflow changes from one state to another state due to the execution of some task.

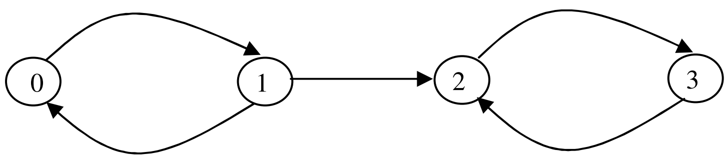

$$\begin{tikzcd} 0 \arrow[r, bend left] & 1 \arrow[l, bend left] \arrow[r] & 2 \arrow[r, bend left] & 3 \arrow[l, bend left] \end{tikzcd}$$

Fig. 1. State transition of an individual task

2.3 Example

We now illustrate how to apply the WIFA approach to workflow modeling and analysis through an example. Assume that we have a workflow with eight tasks, namely T_1, T_2, … T_8. Its specification is as follows:

- T_1 is the direct predecessor of T_2 and T_3, T_2 is the immediate predecessor of T_4, T_4 is the immediate predecessor of T_5, T_5 is the immediate predecessor of T_6 and T_7, T_6 is the second immediate predecessor of T_2, T_3 is the immediate predecessor of T_7, and T_7 is the second immediate predecessor of T_8. See Fig. 2.
- T_6 and T_7 conflict with each other. In other words, after T_5 is executed, if T_6 is selected for execution, then the execution of T_6 will make T_7 not executable and vice versa.

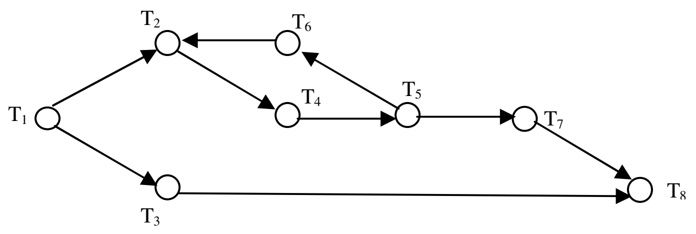


Fig. 2. Precedence graph of an eight-task workflow

- T_1 is executable when the workflow is started. T_2 and T_3 become executable when T_1 is executed. T_2 also becomes executable when T_6 is executed. T_4 becomes executable when T_2 is executed. T_5 becomes executable when T_4 is executed. T_6 and T_7 become executable when T_5 is executed. T_4 becomes executable when T_2 is executed. T_5 becomes executable when T_4 is executed. And T_8 becomes executable when both T_3 and T_7 are executed.

This workflow is formulated in the WIFA framework as:

$T = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8\}$,

$$P = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$A(T_1) = \emptyset, A(T_2) = \{\{T_1\}, \{T_6\}\}, A(T_3) = \{\{T_1\}\},$

$A(T_4) = \{\{T_2\}\}, A(T_5) = \{\{T_4\}\},$

$A(T_6) = A(T_7) = \{\{T_5\}\}, A(T_8) = \{\{T_3, T_7\}\}.$

$S_0 = (1, 0, 0, 0, 0, 0, 0, 0).$

Now let us examine the execution of this workflow. At S_0, T_1 is the only executable task. Let $S_0(T_1)S_1$, then based on the state transition rule, we have

$S_1(T_1) = 2$ (Rule 1)

$S_1(T_2) = S_1(T_3) = 1$ (Rule 2A)

$S_1(T_4) = S_1(T_5) = S_1(T_6) = S_1(T_7) = S_1(T_8) = 0$ (Rule 2A)

So $S_1 = (2, 1, 1, 0, 0, 0, 0, 0)$.

At S_1, T_2, T_3 are executable, because their state values are 1. Let $S_1(T_2)S_2$, then based on the state transition rule, we have

$S_2(T_1) = 2$ (Rule 2C)

$S_2(T_2) = 2$ (Rule 1)

$S_2(T_3) = 1$ (Rule 2B)

$S_2(T_4) = 1$ (Rule 2A)

$S_2(T_5) = S_2(T_6) = S_2(T_7) = S_2(T_8) = 0$ (Rule 2A)

So $S_2 = (2, 2, 1, 1, 0, 0, 0, 0)$.

At S_2, T_3 and T_4 are executable, because their state values are 1. Let $S_2(T_3)S_3$, then based on the state transition rule, we have

$S_3(T_1) = S_3(T_2) = 2$ (Rule 2C)

$S_3(T_3) = 2$ (Rule 1)

$S_3(T_4) = 1$ (Rule 2B)

$S_2(T_5) = S_2(T_6) = S_2(T_7) = S_2(T_8) = 0$ (Rule 2A)

So $S_3 = (2, 2, 2, 1, 0, 0, 0, 0)$. Notice that T_6 is not executable now because neither T_4 nor T_5 is executed.

At S_3, only T_4 is executable, because it is the only task with state value 1 or 3. Let $S_3(T_4)S_4$, then it follows from the state transition rules that $S_4 = (2, 2, 2, 2, 1, 0, 0, 0)$. At S_4, only T_5 is executable. Let $S_4(T_5)S_5$, then it follows from the state transition rules that $S_5 = (2, 2, 2, 2, 2, 1, 1, 0)$.

At S_5, T_6 and T_7 are executable, because their state values are 1. The execution T_6 causes the workflow to proceed along the T_2-T_4-T_5-T_6-T_2 loop. Let $S_5(T_6)S_6$, then based on the state transition rule, we have

$S_6(T_1) = S_5(T_3) = S_5(T_4) = S_5(T_5) = 2$ (Rule 2C)

$S_6(T_2) = 3$ (Rule 2C)

$S_6(T_6) = 2$ (Rule 1)

$S_6(T_7) = S_6(T_8) = 0$ (Rule 2A)

So $S_6 = (2, 3, 2, 2, 2, 2, 0, 0)$. Notice that T_7 becomes not executable now because T_6 and T_7 are in conflict.

Task T_2 will execute at S_6, which results in $S_7 = (2, 2, 2, 3, 2, 2, 0, 0)$. Then task T_4 will execute at S_7, which results in $S_8 = (2, 2, 2, 2, 3, 2, 0, 0)$. Then task T_5 will execute at S_8, which results in $S_9 = (2, 2, 2, 2, 2, 3, 1, 0)$. Let $S_9(T_7)S_{10}$, then based on the state transition rule, we have

$S_{10}(T_1) = S_5(T_2)= S_5(T_3) = S_5(T_4) = S_5(T_5) = 2$ (Rule 2C)

$S_{10}(T_6) = 0$ (Rule 2A)

$S_{10}(T_7) = 2$ (Rule 1)

$S_{10}(T_8) = 1$ (Rule 2A)

So S_{10} = (2, 2, 2, 2, 2, 2, 2, 1). Notice that T_8 is executable now because both T_3 and T_7 are executed. The execution of T_8 results in S_{11} = (2, 2, 2, 2, 2, 2, 2, 2). At this state, no more tasks are executable.

The above analysis only traces one execution path. The entire state transition graph of this workflow is depicted in Fig. 3, which contains 22 states in total. The workflow may either stop at state (2, 2, 2, 2, 2, 2, 2, 2) if the workflow is looped or at state (2, 2, 2, 2, 2, 0, 2, 2) if it doesn't go through the loop.

Discussion: This example shows that our formal workflow model can be directly formulated from the users' specification of the workflow. The importance of this fact lies in that the proposed approach supports automated formulation from users' workflow description to a formal model of the workflow. More discussion will be provided in next section.

2.4 WIFA Modeling Power

The characteristics exhibited by the task executions of workflows such as concurrency, decision making, synchronization and loops are modeled very effectively with the WIFA model. These characteristics are represented using a set of simple constructs:

1) *Sequential execution*: In the example, tasks T_1 and T_2 are executed sequentially. This relationship is specified by $p_{12} = 1$ in the precedence matrix. Such precedence constraints are typical of execution tasks in a workflow. Also, this construct models the causal relationships among activities.
2) *Conflict*: In the example, tasks T_6 and T_7 conflict with each other. This is specified by $c_{67} = c_{76} = 1$ in the conflict matrix. Such a situation will arise, for example, when a user has to choose among multiple possible actions.
3) *Concurrency*: In example, tasks T_2 and T_3 are concurrent. Concurrency is an important attribute of a workflow. A sufficient condition for two tasks to be concurrent is that they are successors of some other task, and they are not in conflict.
4) *Synchronization*: Oftentimes, a task in a workflow has to wait for execution results of two or more other tasks before it can be executed. The resulting synchronization of tasks can be captured by the pre-condition set of a task. In the example, $A(T_8) = \{T_3, T_7\}$,means T_3 is synchronized with T_7 for T_8.
5) *Loop*: Loop is a common characteristic within a workflow structure where some tasks are executed repeatedly. As an example shown in Fig. 2, tasks T_2, T_4, T_5 and T_6 could be executed again and again.
6) *Mutual exclusion*: Mutual exclusion is defined as following.

Definition 7 (mutual exclusion) Two tasks T_i and T_j are said to be mutual exclusive based on the following recursive definition:

1) T_i and T_j are mutual exclusive if $c_{ij} = 1$.
2) If T_i and T_j are mutual exclusive and $^*T_k = \{T_i\}$, then so are T_k and T_j.

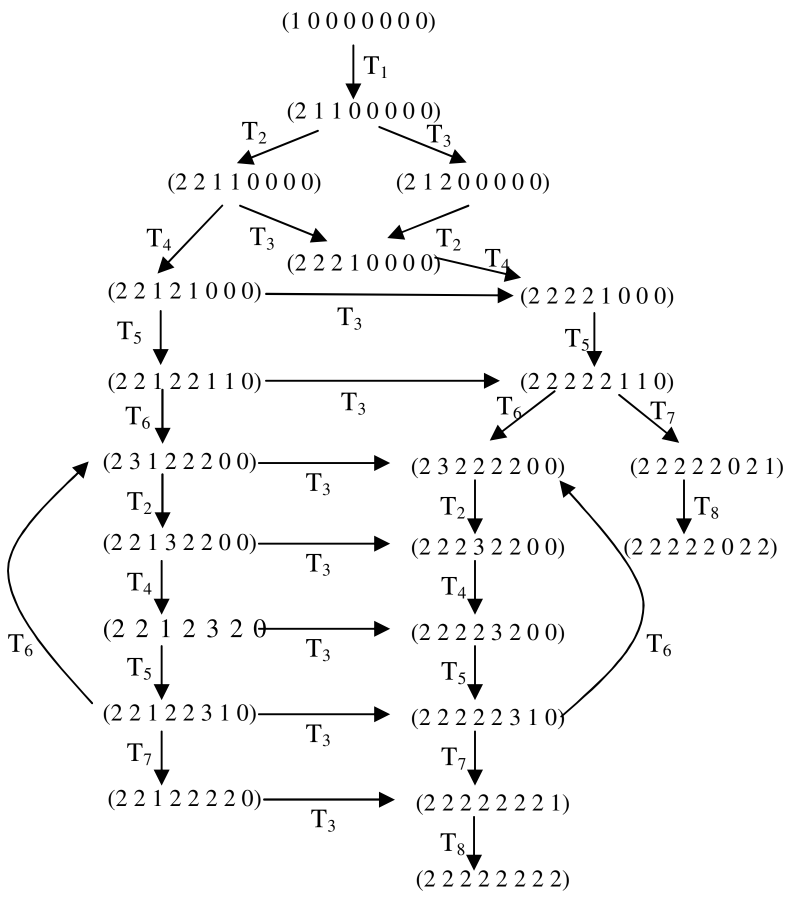


Fig. 3. State transition graph of the example workflow

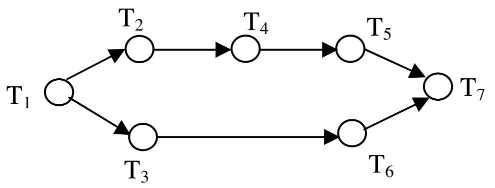


Fig. 4. A precedence graph where T_2 and T_3 are in conflict

According to this definition, any two mutual exclusive tasks are rooted from two conflicting tasks. For example, assume that tasks T_2 and T_3 in Fig.4 are in conflict. Then any task from set $\{T_2, T_4, T_5\}$ and any task from set $\{T_3, T_6\}$ are mutual exclusive. So, when T_2 and T_3 are triggered by T_1, either the branch T_2-T_4-T_5 or the branch T_3-T_6 will be chosen to execute.

3 Well-Formed Workflows

In this section, we introduce *well-formed workflows* which have no dangling tasks and are guaranteed to finish. We particularly discuss *confusion-free workflows*, which are a class of well-formed workflows and have some distinguishing properties. We introduce how to build confusion-free workflows, and how to ensure a workflow remains confusion-free when it needs to be changed.

3.1 Well-Formed Workflow Definitions

Definition 8 (reachable set): A state S of a workflow is *reachable* from the initial state if and only if there is a sequence of tasks that are executable sequentially from the initial state and the execution of these tasks leads the workflow to state S. The set of all reachable states, including the initial state, is called the reachable set. It is denoted by $\mathcal{R}$.

Definition 9 (well-formed workflow): A workflow is *well-formed* if and only if the following two *behavior conditions* are met:

1) $\forall T_i \in T$, $\exists\ S \in \mathcal{R}$ such that $S(T_i) = 1$. (i.e. there is no dangling task.)
2) $\exists\ S \in \mathcal{R}$ such that $S(T_i) \in \{0, 2\}$ for $\forall T_i \in T$. (i.e. there is at least one ending state.)

The example workflow given in Section 2.3 is well-formed, because every task in this workflow is executable, and there are two ending states. In general, the validation of a workflow being well-formed requires the reachability analysis of the workflow. Below we introduce *confusion-free* workflows, which are a class of well-formed workflows with some restrictions imposed on their structure.

Definition 10 (confusion-free workflow): A well-formed workflow is *confusion-free* if and only if the following two *structural conditions* are met:

1) $\forall T_k \in T$ with $|T_k^*| \geq 3$, if $\exists\ T_i, T_j \in T_k^*$ such that $c_{ij} = 1$ (or $c_{ij} = 0$), then for $\forall T_a, T_b \in T_k^*$ $c_{ab} = 1$ (or $c_{ab} = 0$) (i.e., either all tasks triggered by the same task are in conflict, or no pair of them are in conflict.)
2) $\forall T_k \in T$ with ${}^*T_k = \{T_{k1}, T_{k2}, \ldots, T_{kn}\}$, $n \geq 2$, either

$$A(T_k) = \{\{ T_{k1}, T_{k2}, \ldots, T_{kn}\}\}, \tag{1}$$

or

$$A(T_k) = \{\{T_{k1}\}, \{T_{k2}\}, \ldots, \{T_{kn}\}\} \tag{2}$$

(i.e., T_k becomes executable either when all of its predecessor tasks are executed, or when any one of them is executed.)

Based on this definition, the example workflow in Section 2.3 is also confusion-free. As will be described next in Theorem 1, it is easy to construct and validate a confusion-free workflow.

From the perspective of triggering condition and relation among triggered tasks, tasks in a confusion-free well-formed workflow can be classified into four types:

1) *And-In-Parallel-Out* A task belongs to this class iff it is not executable until all its direct predecessor tasks are executed, and after it is executed, all its direct successor tasks can be executed in parallel.
2) *And-In-Conflict-Out* A task belongs to this class iff it is not executable until all its direct predecessor tasks are executed, and after it is executed, only one of its direct successor tasks can be executed.
3) *Or-In-Parallel-Out* A task belongs to this class iff it is executable as long as one of its direct predecessor tasks is executed, and after it is executed, all its direct successor tasks can be executed in parallel.
4) *Or-In-Conflict-Out* A task belongs to this class iff it is executable as long as one of its direct predecessor tasks is executed, and after it is executed, only one of its direct successor tasks can be executed.

Without loss of generality, a task with only one direct predecessor is treated as an "And-In" task, and a task with only one direct successor treated as a "Parallel-Out" task. Denote by set T_{AP} for all And-In-Parallel-Out tasks, T_{AC} for all And-In-Conflict-Out tasks, T_{OP} for all Or-In-Parallel-Out tasks, and T_{OC} for all Or-In-Conflict-Out tasks. Then in the example workflow, we have $T_{AP} = \{T_1, T_3, T_4, T_6, T_7, T_8\}$, $T_{AC} = \{T_5\}$, $T_{OP} = \{T_2\}$, and $T_{OC} = \varnothing$.

3.2 Build a Well-Formed Workflow

Theorem 1: Given a confusion-free, well-formed workflow $WF = (T, P, C, A, S_0)$, by adding a new task T_k to it, the obtained new workflow is denoted by $WF' = (T', P', C', A', S_0')$. Then WF' is also a confusion-free workflow if it matches one of the following cases:

1) $*T_k = T_k* = \varnothing$, i.e., $p'_{ki} = p'_{ik} = 0$ for all $T_i \in T' \setminus \{T_k\}$.
2) $*T_k = \varnothing$, $T_k* \neq \varnothing$, and $\forall T_i \in T_k*$, if $A(T_i)$ is defined in the form of (1) in Definition 10, then $A'(T_i)$ is also defined in the form of (1) by adding T_k to the only set. If $A(T_i)$ is defined in the form of (2), then $A'(T_i)$ is also defined in the form of (2) by adding $\{T_k\}$ to $A(T_i)$.
3) $*T_k \neq \varnothing$, $T_k* = \varnothing$. If $A(T_k)$ is defined in the form of (1) in Definition 10, then there exists a S_a in WF such that all tasks in $*T_k$ have state value of 2; If $A(T_k)$ is defined in the form of (2) in Definition 10, then there exists a S_a in WF such that at least one task in $*T_k$ has state value of 2. In addition, $\exists T_i \in *T_k$, if T_i triggers two or more conflicting tasks, then T_k conflicts with each of these tasks, otherwise, $c_{kj} = 0$ for any $T_j \in T_i*$.
4) $*T_k \neq \varnothing$, $T_k* \neq \varnothing$, with all other conditions appear in 2) and 3). Besides, $\forall T_i \in T_k*$, if T_i is also a predecessor of T_k (i.e., T_k introduces a loop), then $A(T_i)$ can only be in the form of (2) in Definition 10.

Proof:

Case 1): T_k is an isolated task. Based on Definition 3, T_k will not be in any other task's pre-condition set, so it has no impact to the original workflow *WF*, and the two structural conditions of confusion-free workflows are all met in *WF'*. Because T_k has no predecessors, so it is executable in S'_0. Since *WF* is well-formed, there must be an ending state $S_q \in \mathrm{R}(WF)$, then state $S'_q = S_q \cup \{S(T_k) = 2\}$ is an ending state of *WF'*. Therefore, *WF'* is confusion-free.

Case 2) In this case, T_k has no predecessors, so it is executable in S'_0. We need to make sure that all tasks that are successors to T_k are still executable after adding in T_k. $\forall T_i \in T_k{*}$, if $A'(T_i)$ is defined in the form of (1) by adding T_k to the only set, then that *WF* is confusion-free indicates that there is a state S_a in *WF* such that all tasks in ${*}T_i$ have state value of 2. Because T_k is unconditionally executable, so there must be a corresponding state S_a' in *WF'* such that $S_a' = S_a \cup \{S_a'(T_i) = 2\}$. Thus T_i is still executable in *WF'*. If $A'(T_i)$ is defined in the form of (2) by adding $\{T_k\}$ to $A'(T_i)$, then the execution of any task in ${*}T_i$ in *WF'* can still trigger T_i as it does in *WF*, and T_k is just an additional task to trigger T_i. Thus T_i is still executable in *WF'*. Since *WF* is well-formed, there must be an ending state $S_q \in \mathrm{R}(WF)$, then state $S_q' = S_q \cup \{S(T_k) = 2\}$ is an ending state of *WF'*. In addition, $A'(T_i)$ is defined in one of the two desired forms. Therefore, *WF'* is also confusion-free.

Case 3) In this case, T_k has no successors. The other conditions guarantee already that task T_k is executable, and the two structural conditions of confusion-free workflows are also met. We only need to prove that the introduction of T_k won't cause other tasks to become non-executable. It is easy to understand that the state transition behavior of *WF'* from any state *S'* in which $S'(T_k) = 0$ is not affected due to the introduction of T_k. Suppose that at state S_a' we have $S_a'(T_k) = 1$ and T_k is triggered by T_i ($T_i \in {*}T_k$). If all tasks triggered by T_i are able to execute in parallel with T_k ($c_{kj} = 0$ for any $T_i \in T_i{*}$), then T_k has no impact to the execution of other triggered tasks. The other possibility is that T_k is in conflict with any other task triggered by T_i. In this case, if T_k is not chosen for execution, the state transition behavior from *S'* will be just like the case in state $S = S' \setminus \{ S'(T_k) = 1\}$ of *WF*. All these suggests that *WF'* is also a confusion-free workflow.

Case 4) This case is a combination of *Case* 2 and *Case* 3. The *WF'* can be proved confusion-free by jointly applying the reasoning for these two cases if T_k does not introduce a loop to the workflow, In case T_k introduces a loop, since we already restrict that $\forall T_i \in T_k{*}$, if T_i is also a predecessor of T_k, then $A(T_i)$ can only be in the form of (2) in Definition 10, T_i can be triggered as it is without T_k in place. Adding T_k simply introduces one more trigger to T_i. So the loop does not cause any task unexecutable.

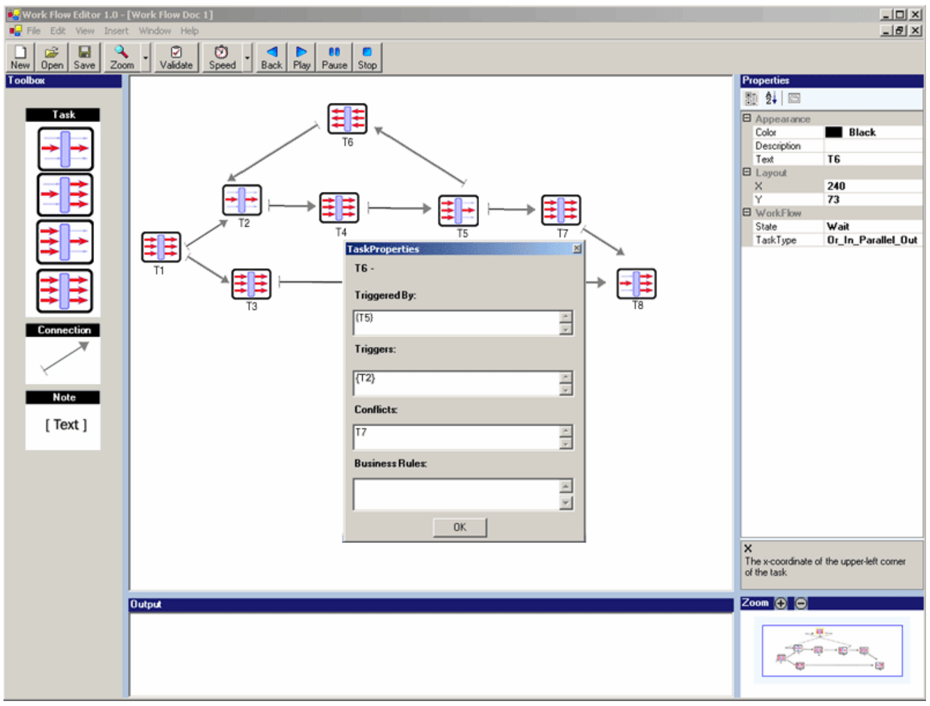
The theorem is proved. □

Theorem 1 can serve as a rule in building a confusion-free workflow. At the beginning, the task set is empty. When the first task is introduced, the workflow is well-formed, because this single task has no predecessors and successors and it is executable. Then we add a second task. This second task can either be an isolated one (Case 1 of Theorem 1), or be a successor of the first task (Case 2 of Theorem 1), or be a predecessor of the first task (Case 3 of Theorem 1), or even be both a predecessor and successor to the first task (Case 4 of Theorem 1). Since the first task is the only possible successor or predecessor to the second task, the new workflow (with these two tasks) is still confusion-free. When we continue to introduce more tasks to the workflow, as long as we make sure each new task is added in such a way that it satisfies the conditions defined in one of the four cases, then the new workflow is guaranteed to be confusion-free.

4 Tool Support

We are currently in the process of developing a visual tool to automate the workflow editing and enactment. In this section we briefly introduce the tool.

The tool has three components: an editor, a simulator and a validator. The editor enables users to create a workflow with an easy to use drag and drop interface. As shown in Fig. 5, the editor has a Tool Box which contains all the objects available for dragging and dropping into the Working area, with each of the four types of tasks represented by a unique icon. Connections add the directional flow from one task to the next. Every connection must have one start task and one end task. When a user is


Work Flow Editor 1.0 - [Work Flow Doc 1]
File Edit View Insert Window Help
New Open Save Zoom Validate Speed Back Play Pause Stop
Toolbox
Task
Connection
Note
[Text]
T1
T2
T3
T4
T5
T6
T7
T8
TaskProperties
Properties
Output
Zoom

T6 -

Triggered By:
{T5}

Triggers:
{T2}

Conflicts:
T7

Business Rules:

OK

Appearance
Color Black
Description
Text T6
Layout
X 240
Y 73
WorkFlow
State Wait
TaskType Or_In_Parallel_Out

X
The x-coordinate of the upper-left corner of the task

Fig. 5. Screenshot of the workflow editor

adding a connection, the working area will change into “connection start mode”. The user will then select the start task by clicking on an existing task in the working area. Once a start task has been selected, the working area will change to “connection end mode”. A phantom connection will follow the user until an end task is selected or the connection adding has been cancelled. Once the connection has been established, the connection is drawn and the working area returns to “normal mode”.

Each object in the working area has general properties such as position, text, description among others. The general Properties area allows the user to see at a glance and change the properties of an object. This Properties area will be populated with the currently active object. Specially, the properties of each task can be seen by right-clicking a task and selecting the Task Properties. The task properties will show the tasks that the current task is triggered by, tasks that this task triggers, any conflicts with this task, and any business rules associated with the task.

A complete workflow can be saved as either an XML file or an image.

The simulator allows users to simulate the execution of a workflow. The users can set the simulation speed with the Speed command, and have options on Play, Back, Pause, and Stop.

The validator allows users to verify if their workflows are well-formed. The users can perform the validation at any stage of workflow construction.

5 Concluding Remarks

In this paper we presented a new formal, yet intuitive, approach for the modeling and analysis of workflows. We introduced our representation of tasks, relations among tasks, state transition rules, and the expressive power of this framework that enables the creation and enactment of workflows. We have showed our definition of well-formed workflows and how to build them, such that whenever a new task is added, it will not alter the well-formedness property of the workflow.

We are currently developing theorems on deleting a task from a well-formed workflow and changing some business rules in a well-formed workflow such that the modified workflow is still well-formed. Meanwhile, we are designing and implementing a visual tool to automate the workflow editing and enactment. The tool will allow the recording of an audit log that will permit the analysis and improvement of current workflows. We will also be working on extending our approach to the inter-organizational workflow modeling and analysis, to be able to represent the interactions between different people and organizations that need to work together for achieving different business goals.

References

1. N. R. Adam, V. Atluri and W. Huang, “Modeling and Analysis of Workflows Using Petri Nets”, *Journal of Intelligent Information Systems*, pp. 131-158, March 1998.
2. P. C. Attie, M. P. Singh, A. Sheth and M. Rusibkiewicz, “Specifying Interdatabase Dependencies,” *Proceedings 19th International Conference on Very Large Database*, pp.134-145, 1993.

3. P. Dourish, “ Process Descriptions as Organizational Accounting Devices: The Dual use of Workflow Technologies”, Paper presented at GROUP'01, (ACM), Sept. 30-Oct. 3, 2001, Boulder, Colorado, USA
4. P. Lawrence, editor, “Workflow Handbook 1997, Workflow Management Coalition”, John Wiley and Sons, New York, 1997.
5. D.C. Marinescu, Internet-Based Workflow Management: Towards a Semantic Web, Wiley Series on Parallel and Distributed Computing, vol. 40, Wiley-Interscience, NY, 2002
6. D. Rosca, S. Greenspan, C. Wild, “Enterprise Modeling and Decision-Support for Automating the Business Rules Lifecycle”, *Automated Software Engineering Journal,* Kluwer Academic Publishers, vol.9, pp.361-404, 2002.
7. M.P. Singh, G. Meredith, C. Tomlinson, and P.C. Attie, “An Event Algebra for Specifying and Scheduling Workflows,” *Proceedings 4th International Conference on Database System for Advance Application*, pp. 53-60, 1995.
8. W.M.P. van der Aalst, “Verification of Workflow Nets”, *Proceedings of Application and Theory of Petri Nets*, Volume 1248 of Lecture Notes in Computer Science, pp. 407-426, 1997.
9. W.M.P. van der Aalst, “Three Good Reasons for Using a Petri Net-Based Workflow Management System”, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises* (IPIC’96), pp. 179–201, Nov 1996.
10. W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, “Business Process Management: A Survey.” *International Conference on Business Process Management* (*BPM 2003*), volume 2678 of *Lecture Notes in Computer Science*, pages 1-12. Springer-Verlag, Berlin, 2003.
11. J. Wang, Timed Petri Nets: Theory and Application, Kluwer Academic Publishers, 1998, ISBN: 0-7923-8270-6.
12. D. Wodtke and G. Weikum, “A Formal Foundation for Distributed Workflow Execution Based State Charts,” *Proceedings 18th International Conference on Database theory*, 1997.
13. M.D. Zisman, “Representation, Specification and Automation of Office Procedures”, *PhD thesis*, University of Pennsylvania, Warton School of Business, 1977.