Workflow Modelling and Analysis Based on the Construction of Task Models

Glória Cravo

1 Center for Linear Structures and Combinatorics, University of Lisbon, 1649-003 Lisbon, Portugal
2 Center of Exact Sciences and Engineering, University of Madeira, Funchal, 9020-105 Madeira, Portugal

Correspondence should be addressed to Glória Cravo; gcravo@uma.pt

Received 30 September 2014; Accepted 3 December 2014

Academic Editor: Dar-Li Yang

Copyright © 2015 Glória Cravo. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We describe the structure of a workflow as a graph whose vertices represent tasks and the arcs are associated to workflow transitions in this paper. To each task an input/output logic operator is associated. Furthermore, we associate a Boolean term to each transition present in the workflow. We still identify the structure of workflows and describe their dynamism through the construction of new task models. This construction is very simple and intuitive since it is based on the analysis of all tasks present on the workflow that allows us to describe the dynamism of the workflow very easily. So, our approach has the advantage of being very intuitive, which is an important highlight of our work. We also introduce the concept of logical termination of workflows and provide conditions under which this property is valid. Finally, we provide a counter-example which shows that a conjecture presented in a previous article is false.

1. Introduction

A workflow is an abstraction of a business process that consists on the execution of a set of tasks to complete a process (e.g., hiring process, loan application, and sales order processing). Tasks represent unities of work to be executed that can be processed by a combination of resources, such as a computer program, an external system, or human activity.

Recall that a business process is a collection of interconnected tasks that takes one or more kinds of input and creates an output that is of value to the customers. The construction of process models is very often a difficult accomplishment for humans since their design can be logically incorrect and enclose errors. So the development of tools to support the design of business processes is indispensable and needs to be based on a solid theory. We believe that our technique is appropriate to model workflows.

Workflows have been successfully deployed to various domains, such as bioinformatics, healthcare, the telecommunication industry, the military, insurance, school administration, mobile computing, systems management, multidatabases, Internet, application development, object technology, operating systems, and transaction management.

In the present paper, we use Graph Theory and Propositional Logic to describe the structure of workflows. It is important to point out that a workflow describes all of the tasks needed to achieve each step in a business process. Documents, information, work orders, reports, and so forth are passed from one task to another for action, according to a set of rules defined by the workflow. Employees or automated applications are the entities that carry out the execution of tasks.

In particular we model workflows as trilographic acyclic directed graphs. The use of trilographic graphs to represent workflows was selected since most business process languages support three types of connectors: AND, OR, and XOR. It is important to emphasize that the inclusion of workflows with OR vertices is a significant advantage of our approach.

Furthermore, our approach is based on the execution of all tasks present on the workflow. This analysis has the advantage of being simple and very intuitive. On the other hand, we can create new models based on the existing ones.

Besides our formal framework allows checking the logical termination of workflows. The logical termination is an important property for workflows because it is indispensable to know if a workflow, such as the hiring process, will
eventually finish. Analyzing the termination of workflows is an important assignment since research and commercial products, such as METEOR and TIBCO, have no support for verification. Errors made at design-time are not detected and result in very costly failures at run-time.

The use of Propositional Logic has the advantage of transforming a workflow into a set of Event-Action (EA) models. Specialized EA models can be easily created to represent new advanced workflow patterns. Afterwards, Propositional Logic and Inference can be carried out on the EA models to analyze properties of workflow models.

It is important to point out that, in the last decade, the rapid increase of business process modelling and management through the adoption of Workflow Management Systems has originated the need for frameworks that can be used to provide a formal technique for defining and analyzing workflows [1–3]. Important advancements have been accomplished in the development of theoretical foundations to allow workflow modeling, verification, and analysis. Several formal methods have been proposed, such as State and Activity Charts [4], Event-Condition-Action rules [5, 6], Petri Nets [7–11], Temporal Logic [12], Markov chains [13], Process and Event Algebras [14, 15], and Six Sigma Techniques [16, 17]. Nevertheless more research is required and specially focused on the use of Graph Theory. Based on this need, we develop our formalism that uses a natural combination of Graph Theory and Propositional Logic to model workflows. Besides, our formalism provides a formal framework based on trilogic acrylic directed graphs that facilitate modeling and analyzing workflows. Finally, our formal framework allows checking the logical termination of workflows.

An important highlight of this paper is the emphasis on the tasks present in the workflow, which allows us to identify easily the dynamism present in the workflow. Finally, we describe the logical termination in a very intuitive form and we present conditions under which this property is valid.

We still provide a counter-example which shows that a conjecture presented in a previous article is false.

2. Workflow Modelling and Analysis

This section is devoted to the presentation of our main results. In particular, we start this section by providing the formal definition of a workflow. In other words, we furnish the formal structure of a business process. Notice that this workflow structure can be also found in [18–21]. It is also important to point out that this type of graphs has an input/output logic operator associated with each vertex. Further, we analyze each model present on the workflow and give special emphasis to the execution of all tasks present in a workflow. Besides, we will create new models based on the existing ones. Finally, we will describe conditions under which a workflow logical terminates. In conclusion, our approach allows us to provide a complete description of workflows.

Definition 1 (see [18–21]). A workflow is a trilogic acrylic directed graphWG = (T, A, A', M), where T = {t₁, t₂, ..., tₙ} is a finite nonempty set of vertices representing workflow tasks. Each task tᵢ (i.e., a vertex) has assigned an input logic operator (represented by > tᵢ) and an output logic operator (represented by tᵢ < ). An input/output logic operator can be the logical AND (•), the OR (@), or the XOR—exclusive—or—(⊕). The set A = {a₁, a₂, a₃, a₄, ..., aₙ} is a finite nonempty set of arcs representing workflow transitions. The transition aᵢ is the tuple (tᵢ, tⱼ) and transition aⱼ is the tuple (tⱼ, tᵢ), where the symbols ⋃ and ⋂ represent abstract tasks which indicate the entry and ending point of the workflow, respectively. Every transition aᵢ, i ∈ {1, ..., n} corresponds to a tuple of the form (tᵢ, tⱼ), where tᵢ, tⱼ ∈ T.

We use the symbol ' to refer to the label of a transition; that is, aᵢ ' references transition aᵢ, aᵢ ∈ A. The elements aᵢ ' are called Boolean terms and form the set A'.

Given tᵢ ∈ T, the incoming transitions for task tᵢ are the tuples of the form (tᵢ, tⱼ), tⱼ ∈ T, and the outgoing transitions are the tuples of the form (tᵢ, tⱼ), tⱼ ∈ T.

The incoming/outgoing condition of task tᵢ is the Boolean expression aᵢ₁ ⋃ ... ⋃ aᵢₙ, where aᵢ ∈ [•, @, ⊕], aᵢ₁, ..., aᵢₙ ∈ A' and aᵢ₁, ..., aᵢₙ are the incoming/outgoing transitions of task tᵢ. The terms aᵢ₁, ..., aᵢₙ are connected with the logic operator > tᵢ, tⱼ <, respectively. If task tᵢ has only one incoming/outgoing transition we assume that the condition does not have logic operator. An Event-Action (EA) model for task tᵢ is an implication of the form tᵢ : f_E ⊢ f_C, where f_E and f_C are the incoming and outgoing conditions of task tᵢ, respectively. An EA model has the behavior with two distinct modes: when f_E is evaluated to true, f_C is also evaluated to true; when f_E is evaluated to false, f_C is always false. And the EA model tᵢ : f_E ⊢ f_C is true if both f_E, f_C are true; otherwise it is false. We say that the EA model tᵢ : f_E ⊢ f_C is positive if its Boolean value is true; otherwise it is said to be negative.

We denote by M the set of all EA models present in WG.

Task tᵢ is said to be executed if the EA model tᵢ : f_E ⊢ f_C is positive. In this case, task tᵢ has attributed the Boolean value true.

Remark 2. Given an expression whose Boolean value is true (resp., false), we simply can represent this fact by 1, (resp., 0).

Remark 3. Given an EA model tᵢ : f_E ⊢ f_C, if f_E is false, then task tᵢ disables all its outgoing transitions. Consequently f_C is also false.

Notice that the workflow starts its execution by enabling transition aᵢ, that is, by asserting aᵢ ' to be true. In other words, the workflow starts its execution by executing task tᵢ.

Notice that aᵢ ' is true if transition aᵢ is enabled; otherwise aᵢ is false. Transitions can be enabled by a user or by an external event. If the EA model tᵢ : f_E ⊢ f_C is negative, then both f_E, f_C are false. In this case, all the transitions of f_C are disabled.

Example 4. In Figure 1 we present a workflow WG = (T, A, A', M), where T = {t₁, t₂, ..., tₙ}, A = {a₁, a₂, a₃, a₄, ..., aₙ}, A' = {a₁', a₂', a₃', a₄', ..., aₙ'}, and M = {t₁ : a₁' ⊢ a₁, t₂ : a₂' ⊢ a₂, t₃ : a₃' ⊢ a₃, t₄ : a₄' ⊢ a₄, t₅ : a₅' ⊢ a₅, t₆ : a₆' ⊢ a₆, t₇ : a₇' ⊢ a₇, t₈ : a₈' ⊢ a₈, t₉ : a₉' ⊢ a₉, t₁₀ : a₁₀' ⊢ a₁₀, t₁₁ : a₁₁' ⊢ a₁₁}.
The output logic operator of task $t_2$ ($t_2 \prec$) is XOR ($\oplus$), while the input logic operator of task $t_9$ ($\succ t_9$) is an AND ($\otimes$).

The incoming transition for task $t_2$ is $a_1 = (t_1, t_2)$ and its outgoing transitions are $a_3 = (t_2, t_4) \oplus a_4 = (t_2, t_5)$. Hence the incoming condition for task $t_2$ is $a_1$, while its outgoing condition is $a_3 \oplus a_4$.

Task $t_2$ is executed if the EA model $t_2 : a_1' \otimes a_3' \oplus a_4'$ is positive, that is, if $a_1'$ is true and only one of the Boolean terms $a_3', a_4'$ is true.

Notice that the workflow from Figure 1 corresponds to the following real situation. Indeed, it can represent the tasks necessary to be executed for a person driving a new car. Let us assume that tasks $t_i$, $i \in \{1, \ldots, 9\}$ have the following meanings:

$t_1$: deciding to purchase a new car to own use;
$t_2$: payment of the car;
$t_3$: getting the drivers license;
$t_4$: deciding to pay by credit;
$t_5$: deciding to pay without credit;
$t_6$: getting rental credit;
$t_7$: getting bank credit;
$t_8$: purchasing of the car;
$t_9$: driving of the new car.

Clearly, the decision of purchasing a new car to own use implies to pay the car and to get the drivers license. The payment of the car implies the execution of only one of the situations to pay by credit or to pay without credit. And to get credit implies to get a rental credit or a bank credit. It is clear that the purchase of the car depends on the execution of only one of the tasks: decide to pay without credit, get rental credit, and get bank credit.

Hence, the possibility to drive a new car depends on the purchase of the car and in obtaining the drivers license. In other words, the execution of task $t_9$ depends on the execution of both tasks $t_5$, $t_8$.

Many other examples can be given. Indeed, too many situations in our life can be described by workflows. For example, the request for a credit card or a loan application is simple examples of workflows.

**Proposition 5.** Let $WG = (T, A, A', M)$ be a workflow. Let $a_i = (t_i, t_j) \in A$, $t_i, t_j \in T$. If $a_i'$ is true, then $t_i$ is necessarily executed.

**Proof.** Let us assume that $a_i'$ is true. Let $t_i : f_{E_i} \otimes f_{C_i}$ be the EA model associated to task $t_i$. If task $t_i$ is not executed, then the EA model $t_i : f_{E_i} \otimes f_{C_i}$ is negative. Since the EA model is negative, all outgoing transitions of task $t_i$ are disabled; in particular, $a_i$ is disabled, that is, $a_i'$ is false, which is a contradiction. Hence task $t_i$ is executed. \qed

**Remark 6.** The condition of Proposition 5 is not sufficient. For example in the workflow from Figure 1, if task $t_2$ is executed, then the EA model $t_2 : a_1' \otimes a_3' \oplus a_4'$ is positive. For $a_1' = true, a_3' = true, a_4' = false$, and $a_4 = (t_2, t_5)$, $t_2$ is executed, but $a_4'$ is false.

**Remark 7.** Let us consider the Boolean term $a_i'$ where $a_i = (t_i, t_j) \in A$, $t_i, t_j \in T$. If $a_i'$ is true, task $t_i$ is not necessarily executed. For example, in the workflow from Figure 2, let us assume that $a_4' = true$, $a_4 = true$, $a_2' = false$, $a_3' = true$, $a_7' = false$, $a_6' = true$, $a_8 = (t_8, t_7)$ and $a_9' = true$. Hence, for this assignment the EA model $t_7 : a_6' \oplus a_9' \otimes a_7'$ is negative, which means that task $t_7$ is not executed. Nevertheless, $a_8 = (t_8, t_7)$ and $a_9'$ is true.

Next we introduce the concept of logical termination. This is a very important structural property, since its analysis will allow to verify if a workflow will eventually finish, according to the initial specifications.

**Definition 8.** Let $WG = (T, A, A', M)$ be a workflow. One says that $WG$ logically terminates if task $t_n$ is executed whenever task $t_1$ is executed.

In the following result we establish a necessary and sufficient condition for the logical termination.

**Theorem 9.** Let $WG = (T, A, A', M)$ be a workflow. Then $WG$ logically terminates if and only if $a_i'$ is true whenever $a_i'$ is true.
Proof. Let us assume that WG logically terminates; that is, task \( t_1 \) is executed whenever task \( t_1 \) is executed. This means that the EA model \( t_1 : a_1' \bowtie f_{C_1} \) is positive whenever the EA model \( t_1 : a_1' \bowtie f_{C_1} \) is positive. Bearing in mind that WG starts its execution by executing task \( t_1 \), then the EA model \( t_1 : a_1' \bowtie f_{C_1} \) is positive. Hence the EA model \( t_n : f_{E_n} \bowtie a'_1 \) is also positive. Consequently, \( a_1', f_{C_1}, f_{E_n'1}, \) and \( a'_1 \) are true. Thus, \( a'_1 \) is true whenever \( a'_1 \) is true.

Conversely, let us assume that \( a'_1 \) is true whenever \( a'_1 \) is true. Let us assume that task \( t_1 \) is executed. This means that the EA model \( t_1 : a'_1 \bowtie f_{C_1} \) is positive. Bearing in mind that \( a'_1 \) is true, according to the behavior of the EA models, necessarily \( f_{E_n} \) is true. Hence the EA model \( t_n : f_{E_n} \bowtie a'_1 \) is positive, which means that task \( t_n \) is executed. So we can conclude that task \( t_n \) is executed whenever task \( t_1 \) is executed, which means that WG logically terminates.

Example 10. It is not hard to check that, in the workflow from Figure 1, \( a'_1 \) is true whenever \( a'_1 \) is true. Thus, the workflow logically terminates.

Next we address our study on the dynamism present in a workflow. Obviously the dynamism is associated with the sequential execution of its tasks. In the workflow from Figure 1 the execution of task \( t_1 \) implies the execution of both tasks \( t_2, t_3 \); the execution of task \( t_2 \) implies the execution of only one of the tasks \( t_4, t_5 \); the execution of task \( t_5 \) implies the execution of only one of the tasks \( t_6, t_7 \); the execution of only one of the tasks \( t_5, t_6 \), and \( t_7 \) implies the execution of task \( t_8 \). Finally, the execution of both tasks \( t_5, t_6 \) implies the execution of task \( t_9 \). Hence, we can state the execution of task \( t_1 \) implies the execution of \( t_2 \circ t_3 \); the execution of task \( t_2 \) implies the execution of \( t_4 \circ t_5 \); the execution of task \( t_5 \) implies the execution of \( t_6 \circ t_7 \); the execution of \( t_6 \circ t_7 \) implies the execution of \( t_8 \). Notice that when we consider \( t_2 \circ t_3 \), the operator \( \circ \) is the output logic operator of task \( t_1 \), while when we consider \( t_5 \circ t_6 \circ t_7 \), \( \circ \) is the input logic operator of task \( t_8 \).

These remarks led us to introduce the following concept.

Definition 11. Let WG = (\( T, A, A', M \)) be a workflow. The compound tasks of WG are the elements of the following form: \( t_i, q_t, p_t, q_t, p_t, t_i, t_i, \ldots, t_i \in T \), \( \varphi \in \{\circ, \circ, \circ\} \). The set of all compound tasks of WG is denoted by \( T' \); that is,

\[
T' = \{ t_i, q_t, p_t, q_t, p_t, t_i, t_i, \ldots, t_i \in T \}, \varphi \in \{\circ, \circ, \circ\} \}.
\]

Example 12. In the workflow from Figure 1, \( T' = \{ t_2 \circ t_3, t_4 \circ t_5, t_6 \circ t_7, t_2 \circ t_3, t_5 \circ t_6 \circ t_7 \} \).

Remark 13. Since every task \( t_i \) has associated a Boolean value, according to its execution, it is also natural to attribute a Boolean value to the compound tasks of WG. The natural attribution is the following. Given any compound task of WG, \( t_i, q_t, p_t, q_t, p_t, t_i \in \{\circ, \circ, \circ\} \).

If \( \varphi = \circ \), then the Boolean value of \( t_i, q_t, p_t, q_t, p_t, t_i \) is 1 if and only if the Boolean value of all \( t_i, t_i, \ldots, t_i \) is equal to 1.

If \( \varphi = \circ \), then the Boolean value of \( t_i, q_t, p_t, q_t, p_t, t_i \) is 1 if and only if there exists at least one of the tasks \( t_i, t_i, \ldots, t_i \) whose Boolean value is equal to 1.

If \( \varphi = \circ \), then the Boolean value of \( t_i, q_t, p_t, q_t, p_t, t_i \) is 1 if and only if there exists only one of the tasks \( t_i, t_i, \ldots, t_i \) with Boolean value equal to 1.

 Naturally, we can state that a compound task \( t_i, q_t, p_t, q_t, p_t, t_i \) is executed if and only if its Boolean value is equal to 1, which means that the compound task \( t_i, q_t, p_t, q_t, p_t, t_i \) is positive. In other words, \( t_i, q_t, p_t, q_t, p_t, t_i \) is executed if and only if, one of the following cases holds:

If \( \varphi = \circ \), all tasks \( t_i, t_i, \ldots, t_i \) are executed.

If \( \varphi = \circ \), at least one of the tasks \( t_i, t_i, \ldots, t_i \) is executed.

If \( \varphi = \circ \), only one of the tasks \( t_i, t_i, \ldots, t_i \) is executed.

In what follows, we introduce a new type of task models that we designate by compound task models.

Definition 14. Let WG = (\( T, A, A', M \)) be a workflow. Let \( t_i, t_j, t_i, t_j, t_i, t_j, \ldots, t_i \in T \), \( \varphi \in \{\circ, \circ, \circ\} \). A compound task model is an implication with one of the following forms:

\[
(1) \ t_i \leftarrow t_j, \ t_j, \ t_j, \ t_j, \ t_j, \ t_j \in T, \varphi \in \{\circ, \circ, \circ\}.
\]

(2) \( t_i, q_t, p_t, q_t, \ t_j \).

(3) \( t_i, q_t, p_t, q_t, t_i \leftarrow t_j, t_j \).

Usually we represent a compound task model by \( t_i \leftarrow t_j \), where \( t_i \) is the incoming task and \( t_j \) is the outgoing task. We say that a compound task model \( t_i \leftarrow t_j \) is positive if both incoming and outgoing tasks are positive, that is, if both tasks \( t_i, t_j \) are executed.

In particular, the implication of the form \( t_i \leftarrow t_j \) is called a simple task model. Clearly, it is positive if both tasks \( t_i, t_j \) are executed.

The set of all simple and compound task models present in WG is called the set of task models of WG and is denoted by \( TM \).

The task models have the behavior with two distinct modes: if its incoming task is true, necessarily its outgoing task is true; if the incoming task is false, the outgoing task is false. In other words, if \( t_i \leftarrow t_j \) is a compound task model, then \( t_j \) is executed if and only if \( t_i \) is executed.

Notice that, in a compound task model \( t_i \leftarrow t_j \), at least one of the tasks \( t_i, t_j \) is compound.

Example 15. In the workflow from Figure 1, the set of its task models is \( TM = \{ t_1 \leftarrow t_2, t_1 \leftarrow t_2 \leftarrow t_3 \circ t_4 \leftarrow t_5 \circ t_4 \leftarrow t_6 \circ t_7, t_5 \circ t_6 \circ t_7 \leftarrow t_8, t_6 \circ t_8 \leftarrow t_9 \} \).

From now on, we use the symbol \( \leftarrow \) with the following meaning: \( X \leftarrow Y \) means that the compound statements \( X \) and \( Y \) are logically equivalent.

According to simple rules of Logic and taking into account the behavior of the task models, we can infer the following result. And the establishment of this result allows us to identify new task models present in the workflow.

In what follows we establish some properties that will allow us to create new task models based on the existing ones.
Proposition 16. Let \( \text{WG} = (T, A, A', M) \) be a workflow. Suppose that the task models \( t_i \rightarrow t_O \) and \( t_O \rightarrow t_j \) belong to TM. Then the model \( t_i \rightarrow t_O \) still holds in WG.

Proof. The proof is trivial.

Theorem 17. Let \( \text{WG} = (T, A, A', M) \) be a workflow.

(a) If both task models

\[
\begin{align*}
  t_i &\rightarrow t_O, \\
  t_j &\rightarrow t_O
\end{align*}
\]

belong to TM, where \( t_O \rightarrow t_j \), then the model \( t_i \rightarrow t_O \) still holds in WG.

(b) If both task models \( t_i \rightarrow t_O \) and \( t_j \rightarrow t_O \) belong to TM, where \( t_O \rightarrow t_Lq_i t_j, \phi \in \{\cdot, \otimes, \oplus\} \), then the compound task model \( t_i \rightarrow t_Lq_i t_O \) still holds in WG.

(c) If both task models \( t_i \rightarrow t_O \) and \( t_O \rightarrow t_j \) belong to TM, where \( t_i \rightarrow t_Lq_i t_j, \phi \in \{\cdot, \otimes, \oplus\} \), then the compound task model \( t_i \rightarrow t_O \) still holds in WG.

(d) If both task models \( t_i \rightarrow t_O \) and \( t_j \rightarrow t_O \) belong to TM, where \( t_i \rightarrow t_Lq_i t_j, \phi \in \{\cdot, \otimes, \oplus\} \), then the compound task model \( t_iq_i t_O \rightarrow t_O \) still holds in WG.

(e) If both task models \( t_i \rightarrow t_O \) and \( t_O \rightarrow t_j \) belong to TM, where \( t_i \rightarrow t_Lq_i t_j, \phi \in \{\cdot, \otimes, \oplus\} \), then the compound task model \( t_iq_i t_O \rightarrow t_O \) still holds in WG.

Proof. (a) Let us assume that both task models (2) and (3) belong to TM. Notice that if either (2) or (3) is negative, since \( t_O \rightarrow t_j \), then necessarily both task models (2) and (3) are negative. So, we just need to verify the result when both task models (2) and (3) are positive.

Let us assume that both tasks models (2) and (3) are positive. Since (2) is positive, necessarily both \( t_i, t_O \) are true. In other words, some of the tasks from \( t_i, t_O \) are executed allowing that \( t_L, t_O \) are executed. Bearing in mind that \( t_O \rightarrow t_j \), then \( t_i \) is true. So according to the behavior of the task models, necessarily \( t_O \) is true. Hence we can state that \( t_O \) is executed, whenever \( t_i \) is executed. Therefore the model \( t_i \rightarrow t_O \) still holds in WG.

Now, in order to prove (b) and (c), we start with the following argument. Bearing in mind that \( t_O \rightarrow t_Lq_i t_j \), according to the behavior of the task models, we can consider that \( t_O \rightarrow t_Lq_i t_j \) still holds in WG. Since \( t_i \rightarrow t_O \) and \( t_O \rightarrow t_Lq_i t_j \) holds in WG, according to Proposition 16 we can conclude that

\[
t_i \rightarrow t_Lq_i t_j
\]

still holds in WG.

(b) Taking into account that \( t_j \rightarrow t_O \) belong to TM, and consequently \( t_j, t_O \) have the same Boolean value, we can replace \( t_j \) by \( t_O \) in (4), obtaining the new model \( t_i \rightarrow t_Lq_i t_O \), which means that \( t_i \rightarrow t_O \) still holds in WG.

(c) Taking into account that \( t_O \rightarrow t_j \) belong to TM, and consequently \( t_O, t_j \) have the same Boolean value, we can replace \( t_j \) by \( t_O \) in (4), obtaining the new model \( t_i \rightarrow t_Lq_i t_O \), which means that \( t_i \rightarrow t_O \) still holds in WG.

Analogously, in order to prove (b) and (c) we start by making the next statement. Taking into account that \( t_i \rightarrow t_Lq_i t_j \), according to the behavior of the compound task models, we can consider that \( t_iq_i t_j \rightarrow t_i \) holds in WG. Since \( t_iq_i t_j \rightarrow t_i \) and \( t_i \rightarrow t_O \) hold in WG, according to Proposition 16 we can conclude that

\[
t_iq_i t_j \rightarrow t_O
\]

still holds in WG.

(d) Bearing in mind that \( t_j \rightarrow t_O \) holds in WG, and consequently \( t_j, t_O \) have the same Boolean value, we can replace \( t_j \) by \( t_O \) in (4), obtaining the new model \( t_i \rightarrow t_Lq_i t_O \), which means that \( t_i \rightarrow t_O \) still holds in WG.

(e) Bearing in mind that \( t_O \rightarrow t_j \) holds in WG, and consequently \( t_O, t_j \) have the same Boolean value, we can replace \( t_j \) by \( t_O \) in (4), obtaining the new model \( t_iq_i t_O \rightarrow t_O \), which means that \( t_iq_i t_O \rightarrow t_O \) still holds in WG.

The previous results allow us to identify new task models based on the existing ones, as it is described below.

Definition 18. Let \( \text{WG} = (T, A, A', M) \) be a workflow. An extended task model is a model obtained by applying a finite sequence of some of the properties established in Proposition 16 and Theorem 17. One adopts the notation \( \text{TM}' \) to represent the set of all extended task models of WG.

Example 19. In the workflow from Figure 1, bearing in mind that \( t_1 \rightarrow t_2 \cdot t_3, t_2 \rightarrow t_4 \cdot t_5 \in \text{WG} \), according to Theorem 17 we can conclude that the model \( t_1 \rightarrow t_2 \cdot t_3, t_4 \cdot t_5 \) still holds in WG. Therefore, we can state that \( t_1 \rightarrow t_2 \cdot t_3, t_4 \cdot t_5 \) is an extended task model of WG.

Notice we adopt the same notation of the task models to represent the extended task models. Furthermore, the extended task models verify the same properties of the task models. In particular, given an extended task model \( B \rightarrow C \), necessarily both \( B, C \) have the same Boolean value.

Naturally when we consider the set of all task models and extended task models presented in WG, we obtain all possible models that can be generated by the task models of the workflow. This set of task models will be designated by the closure of TM. In certain sense, we can state that the set of all task models from WG is a set of generators of all possible models of WG.

Definition 20. Let \( \text{WG} = (T, A, A', M) \) be a workflow. One defines the closure of TM as the set of all task models and extended task models in WG. This set is denoted by \( \text{TM}' \). In other words, \( \text{TM}' = \text{TM} \cup \text{TM}' \).
Example 21. As we saw in Example 15 in the workflow from Figure 1, TM = \{t_1 \rightarrow t_2 \cdot t_3, t_2 \rightarrow t_4 \oplus t_5, t_4 \rightarrow t_6 \oplus t_7, t_5 \rightarrow t_8 \oplus t_9 \rightarrow t_3\}. Since \(t_1 \rightarrow t_2 \cdot t_3, t_2 \rightarrow t_4 \oplus t_5 \rightarrow t_6 \oplus t_7 \rightarrow t_8 \cdot t_9 \rightarrow t_3\), we can conclude that \(t_1 \rightarrow t_2 \cdot t_3, t_2 \rightarrow t_4 \oplus t_5 \rightarrow t_6 \oplus t_7 \rightarrow t_8 \cdot t_9 \rightarrow t_3\) in TM. Now bearing in mind that \(t_4 \rightarrow t_6 \oplus t_7 \rightarrow t_8 \cdot t_9 \rightarrow t_2\), we can state that \(t_1 \rightarrow (t_2 \cdot t_3 \oplus t_4) \cdot t_5 \rightarrow t_6 \oplus t_7 \rightarrow t_8 \cdot t_9 \rightarrow t_2\). Bearing in mind that \(t_5 \rightarrow t_6 \oplus t_7 \rightarrow t_8 \cdot t_9\), applying once more Theorem 17 we infer that \(t_1 \rightarrow t_2 \cdot t_3 \rightarrow t_5 \rightarrow t_2\). As \(t_8 \cdot t_9 \rightarrow t_3 \rightarrow t_1\), applying again Theorem 17 we conclude that \(t_5 \rightarrow t_9 \rightarrow t_1\) in TM.

Notice the workflow from Figure 1 logically terminates and \(t_1 \rightarrow t_9 \rightarrow t_1\) in TM. Furthermore, we studied many other examples of workflows that logically terminates and simultaneously \(t_1 \rightarrow t_9 \rightarrow t_1\) in TM. The analysis of these different cases led us to formulate the following conjecture.

Conjecture 22 (see [21]). Given a workflow \(WG = (T, A, A', M)\), then \(WG\) logically terminates if and only if \(t_1 \rightarrow t_n \rightarrow TM^*\).

After the analysis of many cases, we start believing that the conjecture was true. Nevertheless it is false, as the following example proves. Let us consider the following workflow.

Example 23. It is not hard to check that the workflow from Figure 3 logically terminates; nevertheless the condition \(t_1 \rightarrow t_n \rightarrow TM^*\) is not valid.

Indeed the condition of the Conjecture 22 is not necessary. However it is sufficient, as we prove in the following result.

Theorem 24. Let \(WG = (T, A, A', M)\) be a workflow. If \(t_1 \rightarrow t_n \rightarrow TM^*\), then \(WG\) logically terminates.

Proof. Let us assume that \(t_1 \rightarrow t_n \rightarrow TM^*\).

Case 1. Suppose that \(t_1 \rightarrow t_n \rightarrow TM\). This means that WG has the following structure:

\[
\leftarrow a_i \rightarrow t_1 \rightarrow a_n \rightarrow t_7, \quad (6)
\]

In this case, \(t_1 = t_n\). Since WG starts its execution by executing task \(t_1\), we can conclude that task \(t_n\) is executed whenever task \(t_1\) is executed, which means that WG logically terminates.

Case 2. Suppose that \(t_1 \rightarrow t_n \rightarrow TM^* \setminus TM = TM^*. \) So, \(t_1 \rightarrow t_n\) was obtained by applying some of the results established in Proposition 16 and Theorem 17. Since \(t_1 \rightarrow t_n \rightarrow TM^*\) and the workflow starts its execution by executing task \(t_1\), that is, by asserting \(t_1\) to be true, according to the behavior of the extended models, necessarily \(t_n\) is still asserted to be true. This means that task \(t_n\) is executed. Hence, task \(t_n\) is executed whenever task \(t_1\) is executed. Thus \(WG\) logically terminates. \(\square\)

3. Conclusions

In this paper we develop a formalism to describe and analyse the structure of workflows, based on graphs and Propositional Logic. Indeed, we describe the structure of a workflow as a graph whose vertices represent tasks and the arcs are associated to workflow transitions. To each task an input/output logic operator is associated and this logic operator can be the logical AND (\(\cdot\)), the OR (\(\oplus\)), or the XOR—exclusive-or (\(\oplus\)). Furthermore, we associate a Boolean term to each transition present in the workflow.

It is important to point out that our main emphasis is the analysis of a workflow through the study of its task models which allows us to describe the dynamism of the workflow in a very simple and intuitive way.

Another relevant aspect of our approach is the introduction of the concept of compound tasks. This concept allows us to identify new task models based on the existing ones. Through these new task models we are able to describe the dynamism present in a workflow in a very simple way. Clearly, the study of the dynamism of a workflow is equivalent to analyse the sequential execution of its tasks.

We still analyze the concept of logical termination and we provide necessary and sufficient conditions under which this property is valid.

Finally, given a workflow \(WG\) we provide a counter-example which shows that the conjecture of \(t_1 \rightarrow t_n \rightarrow TM^*\) being a necessary and sufficient condition under which WG logically terminates is false. In fact, the condition is necessary; nevertheless it is not sufficient.

Conflict of Interests

The author declares that there is no conflict of interests regarding the publication of this paper.

References

[1] J. Cao, C. Chan, and K. Chan, “Workflow analysis for web publishing using a stage-activity process model,” Journal of Systems and Software, vol. 76, no. 3, pp. 221–235, 2005.

[2] M. Hu, J. Zhang, and X. Chen, “Grid workflow for decision resource scheduling,” in Proceedings of the 5th WSEAS International Conference on Applied Computer Science, pp. 533–539, Hangzhou, China, April 2006.
[3] J. L. Kmetz, *Mapping Workflows and Managing Knowledge: Simply, Sensibly, Flexibly, and Without Software*, 2010.

[4] P. Muth, D. Wodtke, J. Weissenfels, G. Weikum, and K. Dittrich, “Enterprise-wide workflow management based on state and activity charts,” in *Workflow Management Systems and Interoperability*, vol. 164 of NATO ASI Series, pp. 281–303, Springer, Berlin, Germany, 1998.

[5] U. Dayal, M. Hsu, and R. Ladin, “Organizing long-running activities with triggers and transactions,” in *Proceedings of the ACM SIGMOD International Conference on Management of Data Table of Contents*, pp. 204–214, ACM Press, Atlantic City, NJ, USA, 1990.

[6] J. Eder, H. Groiss, and H. Nekvasil, “A workflow system based on active databases,” in *Proceedings of the 9th Austrian-Informatics Conference on Workflow Management: Challenges, Paradigms and Products (CON ´94)*, G. Chroust and A. Benczur, Eds., pp. 249–265, Linz, Austria, 1994.

[7] W. M. P. van der Aalst, “The application of petri nets to workflow management,” *Journal of Circuits, Systems and Computers*, vol. 8, no. 1, pp. 21–66, 1998.

[8] W. M. P. van der Aalst, “Workflow verification: finding control-flow errors using petri-net-based techniques,” in *Business Process Management*, W. M. P. van der Aalst, J. Desel, and A. Oberweis, Eds., vol. 1806 of *Lecture Notes in Computer Science*, pp. 161–183, Springer, Berlin, Germany, 2000.

[9] F. Gottschalk, W. M. P. van der Aalst, M. H. Jansen-Vullers, and M. La Rosa, “Configurable workflow models,” *International Journal of Cooperative Information Systems*, vol. 17, no. 2, pp. 177–221, 2008.

[10] F. Gottschalk, W. M. P. van der Aalst, M. H. Jansen-Vullers, and H. M. W. Verbeek, “Protos2CPN: using colored Petri nets for configuring and testing business processes,” *International Journal on Software Tools for Technology Transfer*, vol. 10, no. 1, pp. 95–110, 2008.

[11] H. M. W. Verbeek, W. M. P. van der Aalst, and A. H. M. Hofstede, “Verifying workflows with cancellation regions and or-joins: an approach based on relaxed soundness and invariants,” *Computer Journal*, vol. 50, no. 3, pp. 294–314, 2007.

[12] P. Attie, M. Singh, A. Sheth, and M. Rusinkiewicz, “Specifying and enforcing intertask dependencies,” in *Proceedings of the 19th International Conference on Very Large Data Bases*, pp. 134–145, Morgan Kaufman, Dublin, Ireland, 1993.

[13] J. Klingemann, J. Wasch, and K. Aberer, “Deriving service models in cross-organizational workflows,” in *Proceedings of the 9th International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises (RIDE-VE ´99)*, pp. 100–107, Sydney, Australia, March 1999.

[14] A. H. M. ter Hofstede and E. R. Nieuwland, “Task structure semantics through process algebra,” *Software Engineering Journal*, vol. 8, no. 1, pp. 14–20, 1993.

[15] M. P. Singh and K. van Hee, “Semantical considerations on workflows: an algebra for intertask dependencies,” in *Proceedings of the 5th International Workshop on Database Programming Languages (DBLP ´95)*, Springer, Umbria, Italy, 1995.

[16] B. B. Manish, A. Bhardwaj, and A. P. S. Rathore, “Six sigma methodology utilization in telecom sector for quality improvement—a DMAIC process,” *International Journal of Engineering Science and Technology*, vol. 2, no. 12, pp. 7653–7659, 2010.

[17] S. H. Han, M. J. Chae, K. S. Im, and H. D. Ryu, “Six sigma-based approach to improve performance in construction operations,” *Journal of Management in Engineering*, vol. 24, no. 1, pp. 21–31, 2008.

[18] G. Cravo, “Applications of propositional logic to workflow analysis,” *Applied Mathematics Letters*, vol. 23, no. 3, pp. 272–276, 2010.

[19] G. Cravo, “Logical termination of workflows: an interdisciplinary approach,” *Journal of Numerical Analysis, Industrial and Applied Mathematics*, vol. 5, no. 3–4, pp. 153–161, 2011.

[20] G. Cravo and J. Cardoso, “Termination of workflows: a snapshot-based approach,” *Mathematica Balkanica*, vol. 21, no. 3–4, pp. 233–243, 2007.

[21] G. Cravo, “Workflow analysis—a task model approach,” in *Proceedings of the 2014 International Conference om Pure Mathematics, Applied Mathematics, Computational Methods (PMAMCM ’14)*, E. Nikos, E. Mastorakis, M. Panos, R. P. Agarwal, and L. Kocinac, Eds., Mathematics and Computers in Science and Engineering Series 29, pp. 2227–4588, July 2014.