Verifying the Logical Termination of Workflows

Jorge Cardoso * Glória Cravo †‡

January 9, 2006

Abstract

Systems and infrastructures are currently being developed to support e-commerce activities. Workflow management systems and workflows are one of the strong technological candidates to deploy and support e-commerce applications. E-commerce workflows require a precise modeling to ensure that they perform according to initial specifications. Important advancements have been accomplished in the development of theoretical foundations for workflow modeling, verification, and analysis. Nevertheless, more research is required. It is essential to explore the use of formal methods for the modeling and verification of workflow's properties. In this paper we present a formal framework, based on control flow graphs theory, to verify the correctness of workflows. In our approach, workflows are modeled with tri-logic acyclic directed graphs. The formalism developed allows to verify one important property, the logical termination of workflows.

Keywords: Workflows, Process Modeling, Business Processes, Graphs.

1 Introduction

Organizations operating in modern markets, such as e-commerce, require a systematic design, planning, control, and management of business processes. These requirements can be achieved with the use of Workflow Management

*Departamento de Matemática, Universidade da Madeira, 9000-390 Funchal, Portugal. (jcardoso@uma.pt)

†Departamento de Matemática, Universidade da Madeira, 9000-390 Funchal, Portugal. (gcravo@uma.pt)

‡Work partially supported by the project POSI/EIA/61214/2004 and done within the activities of the LabMAG laboratory.

Systems (WfMSs). WfMSs allow organizations to streamline and automate business processes, reengineer their structure, as well as, increase efficiency and reduce costs.

Workflows have been successfully deployed to various domains, such as bio-informatics [17], healthcare [4], the telecommunication industry [24], the military [20], and school administration [6]. Other areas, such as mobile computing, systems management, multi-databases, the Internet, application development, object technology, operating systems, and transaction management have also beneficed from the use of workflow technology [23].

The complexity, configuration, and structure of workflows depend on the underlying business processes they model. Workflows may involve many distinct, heterogeneous, autonomous, and distributed tasks that are interrelated in complex ways. The complexity of large workflows requires a precise modeling to ensure that they perform according to initial specifications. The development of frameworks and theories to achieve a precise modeling is a difficult undertaking.

In the last decade, important advancements have been accomplished in the implementation of workflow systems (commercial and research systems) and in the development of theoretical foundations to allow workflow modeling, verification, and analysis. Nevertheless, the solutions proposed are still insufficient and more research is required [16].

A number of formal frameworks have been proposed for workflow modeling and include State and Activity Charts [27], Graphs [22], Event-Condition-Action rules [13, 14], Petri Nets [1, 2], Temporal Logic [5], Markov chains [21], and Process and Event Algebras [18, 29]. The use of directed graphs to model the control flow of workflows has been the main formalism used in workflow systems implementation (e.g. METEOR-S [25], TIBCO Workflow [31], and Staffware Process Suite [30]).

Workflow modeling, verification, and analysis take a renewed importance with the development and maturity of infrastructures and solutions that support e-commerce applications, Web services, and Web processes [11, 23]. This is because while in some cases Web services may be utilized in an isolated form, it is natural to expect that Web services will be integrated as part of workflows [15].

In this paper we present a formal framework, based on control flow graphs theory, to check workflow specifications for correctness. In our approach we model workflows with tri-logic acyclic directed graphs [22] and develop a formalism to verify the logical termination of workflows.

This paper is structured as follows. Section 2 briefly discusses business processes and workflows. Section 3 introduces formally the structure of tri-

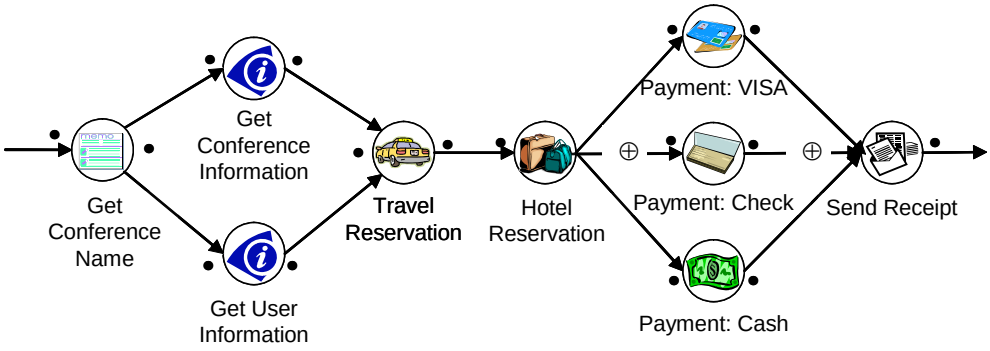

Figure 1: A very simple workflow

logic acyclic directed graphs and presents our approach for checking their logical termination. Section 4 describes two important areas where the theorem for verifying the logical termination of workflows, presented in section 3, can be used. Finally, section 5 contains our conclusions.

2 Business Processes and Workflows

A workflow is an abstraction of a business process that consists of one or more tasks to be executed to perform the business process. A task represents a unit of work to be executed, which will be processed by a combination of resources. A resource may be a simple fragment of code, a computer program, an external system, or a human activity.

Workflows can be modeled using graphs as shown in Figure 1. Graphs are a formal notation for representing business processes. Tasks are represented with vertices and the partial ordering of tasks is modeled with arcs, known as transitions. For example, the task *Get Conference Information* is associated with a computer program that checks the information of a conference automatically. A task may also be associated with a human activity, requiring a specific person, or a person with a particular role, to carry out the execution of the task manually. For example, the task *Get Conference Name* is executed manually and consists in entering a conference's name into a workflow application form.

The workflow illustrated in Figure 1 is formally described using a trilogic acyclic directed graph. This is the formal method that we will use to model workflows. This formal method has the desired degree of intuitiveness

and simplicity, which is appropriate to be used by business process analysts. These workflows are called tri-logic because a logic operator, an and ($\bullet$), an or ($\otimes$), or an exclusive-or ($\oplus$), can be associated with input/output transitions of each task (vertex). For example, the task *Get Conference Name* has associated with its output transitions an and. The task *Travel Reservation* has also associated with its input transitions an and. The task *Hotel Reservation* has associated with its input transition an and, and has associated with its output transitions an exclusive-or.

3 Workflow Termination

In our approach we model workflows with tri-logic acyclic directed graphs. This type of graphs has an input/output logic operator associated with each vertex of the graph. We start by giving a formal definition of a workflow structure. The semantics of these vertices are well-known and have been widely used [25][31][30].

Definition 1 *A workflow is a tri-logic acyclic direct graph* $WG = (T, A)$, *where* $T = \{t_1, t_2, \ldots, t_n\}$ *is a finite nonempty set of vertices representing workflow tasks. Each task* t_i *(i.e., a vertex) has an input logic operator (represented by* $\succ t_i$*) and an output logic operator (represented by* $t_i \prec$*). An input/output logic operator can be the logical and (*$\bullet$*), the or (*$\otimes$*), or the exclusive-or (*$\oplus$*). The set* $A = \{a_{\sqcup}, a_{\sqcap}, a_1, a_2, \ldots, a_m\}$ *is a finite nonempty set of arcs representing workflow transitions. Each transition* a_i, $i \in \{1, \ldots, m\}$, *is a tuple* (t_k, t_l) *where* t_k, $t_l \in T$. *The transition* $a_{\sqcup}$ *is a tuple of the form* $(\sqcup, t_1)$ *and transition* $a_{\sqcap}$ *is a tuple of the form* $(t_n, \sqcap)$. *The symbols* $\sqcup$ *and* $\sqcap$ *represent abstract tasks which indicate the entry and ending point of the workflow, respectively. We use the symbol* $'$ *to reference the label of a transition, i.e.* a_i' *references transition* a_i, $a_i \in A$. *The elements* a_i' *are called Boolean terms and form the set* A'.

Example 1 *Figure 2 shows a workflow* $WG = (T, A)$, *where* $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$, $A = \{a_{\sqcup}, a_{\sqcap}, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\}$ *and* $A' = \{a_{\sqcup}', a_{\sqcap}', a_1', a_2', a_3', a_4', a_5', a_6', a_7', a_8'\}$. *The tuple* $a_2 = (t_1, t_3)$ *is an example of a transition. In task* t_3, $\oplus$ *is the input logic operator (*$\succ t_3$*) and* $\bullet$ *is the output logic operator (*$t_3 \prec$*).*

Definition 2 *The incoming transitions for task* $t_i \in T$ *are the tuples of the form* $a_j = (x, t_i)$, $x \in T, a_j \in A$, *and the outgoing transitions for task* t_i *are the tuples of the form* $a_l = (t_i, y)$, $y \in T, a_l \in A$.

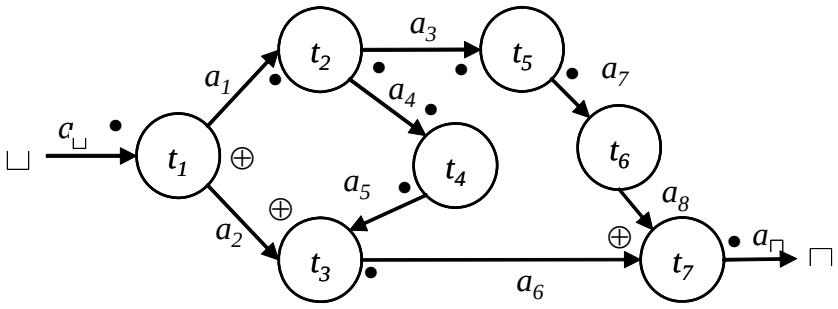

Figure 2: Example of a tri-logic acyclic direct graph

Example 2 *In figure 2, the incoming transition for task* t_2 *is* $a_1 = (t_1, t_2)$ *and the outgoing transitions are* $a_4 = (t_2, t_4)$ *and* $a_3 = (t_2, t_5)$.

When a transition is enabled its Boolean term is true and when a transition is disabled its Boolean term is false.

Example 3 *Let us consider again figure 2. If transition* a_1 *is enable/disable then* a'_1 *is* $true/false$, *respectively.*

Definition 3 *The incoming condition for task* $t_i \in T$ *is a Boolean expression with terms* $a' \in A'$, *where* a *is an incoming transition of task* t_i. *The terms* a' *are connected with the logic operator* $\succ t_i$.

Example 4 *In figure 2, the incoming condition for task* t_3 *is* $a'_2 \oplus a'_5$.

Definition 4 *The outgoing condition for task* $t_i \in T$ *is a Boolean expression with terms* $a' \in A'$, *where* a *is an outgoing transition of task* t_i. *The terms* a' *are connected with the logic operator* $t_i \prec$.

Example 5 *In figure 2, the outgoing condition for task* t_2 *is* $a'_3 \bullet a'_4$.

In order to verify the logical termination of a workflow, we need to introduce the concept of Event-Action (EA) model. EA models describe which conditions need to be verified for a task to be executed and the consequences of the execution of a task.

Definition 5 *Given a workflow* $WG = (T, A)$, *an Event-Action (EA) model for a task* $t_i \in T$ *is an implication of the form* $t_i : f_E \rightsquigarrow f_C$, *where*

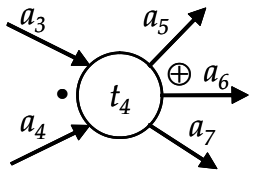

Figure 3: An Event-Action model for task t_4

Table 1: EA model behavior

f_E	f_C	$f_E \rightsquigarrow f_C$
0	0	0
1	0	0
1	1	1

f_E and f_C are the incoming and outgoing conditions of task t_i, respectively. The condition f_E is called the event condition and the condition f_C is called the action condition.

Event-Action models can be constructed for any given task of a workflow. The model expresses that the Boolean value of the condition f_E is propagated to f_C. For a particular task t, the value of the condition f_E is evaluated to *true* or $false$ depending on the Boolean terms associates with the incoming transitions for task t.

An EA model has a behavior with two distinct modes: when f_E is evaluated to *true* and when f_E is evaluated to $false$. In the first situation f_C can be evaluated to *true* or $false$ according to the values of the outgoing transitions; in the other case, f_C is always $false$. The behavior of an EA model is described in table 1.

Only when the event expression f_E and the action expression f_C are 1, the model $f_E \rightsquigarrow f_C$ is 1. In all the other cases the model has the value 0. Event-Action models can be seen as logic gates. The gate is open when its two composing elements, f_E and f_C, are 1. Otherwise, when f_E or f_C is 0, the gate is closed.

Example 6 *As an example, let us consider task t_4 illustrated in Figure 3. Task t_4 has the following Event-Action model, $t_4 : a'_3 \bullet a'_4 \rightsquigarrow a'_5 \oplus a'_6 \oplus a'_7$, where $f_E = a'_3 \bullet a'_4$ and $f_C = a'_5 \oplus a'_6 \oplus a'_7$. This model expresses that when both incoming transitions of task t_4 (a_3 and a_4) are enables (i.e., the Boolean*

terms a'_3 and a'_4 are true) the Boolean condition f_E is evaluated to true. If only one of the outgoing transitions of task t_4 (i.e., a_5, a_6 or a_7) become enable (i.e., only one of the Boolean terms a'_5, a'_6 or a'_7 is true) the Boolean condition f_C is evaluated to true. Consequently, the model $f_E \rightsquigarrow f_C$ is true if and only if both terms a'_3 and a'_4 are true and only one of the terms a'_5, a'_6 or a'_7 is true.

Definition 6 *We say that the EA model $f_E \rightsquigarrow f_C$ is positive if its value is 1, otherwise we say that the model is negative.*

A workflow starts its execution when transition $a_\sqcup$ is enabled. The transition can be enabled explicitly by a user or implicitly by an external event.

Note that the outgoing conditions are enabled only after the incoming conditions are enabled.

A workflow is started by enabling its entry point transition ($a_\sqcup$). When the workflow is correctly designed, it terminates by enabling the ending transition ($a_\sqcap$).

Definition 7 *A simple EA model is an EA model $f_E \rightsquigarrow f_C$, where $f_E = a'_j$ and $f_C = a'_l$, for $j, l \in \{\sqcup, \sqcap, 1, \ldots, m\}$, with $j \neq l$.*

Definition 8 *A compound EA model is a nonsimple EA model $f_E \rightsquigarrow f_C$, i.e., f_E or f_C are boolean expressions with an and ($\bullet$), an or ($\otimes$), or an exclusive-or ($\oplus$).*

Example 7 *In Figure 2, $a'_7 \rightsquigarrow a'_8$ is a simple EA model. The models $a'_1 \rightsquigarrow a'_3 \bullet a'_4$ and $a'_2 \oplus a'_5 \rightsquigarrow a'_6$ are compound EA models.*

Remark 1 *The set of all elements $i \in \{1, \ldots, n\}$ such that $t_i : f_{E_i} \rightsquigarrow f_{C_i}, t_i \in T$ is a compound EA model is denoted by Γ, i.e., $\Gamma = \{i \in \{1, \ldots, n\}$ such that $t_i : f_{E_i} \rightsquigarrow f_{C_i}$ is a compound EA model$\}$.*

Definition 9 *Logical Workflow Representation. The logical representation of a workflow $WG = (T, A)$ is a Boolean expression $b = \wedge (f_{E_i} \rightsquigarrow f_{C_i})$, $t_i : f_{E_i} \rightsquigarrow f_{C_i}, t_i \in T$, for all $i \in \Gamma$, i.e., the conjunction of all compound EA models of WG.*

Definition 10 *A workflow WG is a contradiction if its logical workflow representation b is a Boolean contradiction.*

Definition 11 *A materialized workflow instance of a workflow* $WG = (T, A)$ *is a workflow* WG_i *for which the Boolean terms* $a'_j, a'_j \in A'$*, have been set to true or* $false$ *according to table 1.*

Definition 12 *A materialized workflow instance* WG_i *is true if its logical workflow representation* b *is true and a materialized workflow instance* WG_j *is* $false$ *if its logical workflow representation* b *is* $false$*.*

Definition 13 *A materialized workflow instance* WG_j *of a workflow* $WG = (T, A)$ *logically terminates if transition* $a_{\sqcap}$ *is enabled at some point in time after transition* $a_{\sqcup}$ *has been enabled.*

Reaching the ending transition ($a_{\sqcap}$) indicates the (logical) termination of processing; i.e. indicates that no further processing should proceed. Once $a_{\sqcup}$ is enabled, tasks of the workflow start their execution. The processing of the workflow stops when one of the following cases occurs: (a) the workflow finishes by enabling transition $a_{\sqcap}$, (b) the processing stops at some task because of a workflow design problem. Note that when situation (b) holds, it means that there exists a task t for which the respectively compound EA model is negative. So both f_E and f_C are false, or f_E is true but f_C is false.

Definition 14 *A workflow* $WG = (T, A)$ *logically terminates if all its materialized workflow instances* WG_j *logically terminate.*

Remark 2 *A workflow for which all EA models are simple has the following structure,* $\sqcup \xrightarrow{a_{\sqcup}} t_1 \xrightarrow{a_1} t_2 \xrightarrow{a_2} t_3 \ldots t_{n-1} \xrightarrow{a_n} t_n \xrightarrow{a_{\sqcap}} \sqcap$*. The set of all compound EA models is* $\varnothing$ *and therefore* b *does not exist. However, this situation is not a problem, since this type of workflow always logically terminates. In fact, this is a trivial case of logical workflow termination. Note that all of its EA models are positive.*

From now on, we will consider nontrivial situations, i.e., we consider workflows with compound EA models. Our aim is to find necessary and sufficient conditions for logical workflow termination.

Theorem 1 *If a workflow* WG *is a contradiction then it does not logically terminates.*

Proof 1 *Suppose that the workflow* WG *is a contradiction. According to Definition* 10 *its logical workflow representation* b *is a Boolean contradiction. As* $b = \wedge(f_{E_i} \rightsquigarrow f_{C_i})$, $t_i : f_{E_i} \rightsquigarrow f_{C_i}$, $t_i \in T$*, for all* $i \in \Gamma$*, then there exists* $i \in \Gamma$ *such that* $f_{E_i} \rightsquigarrow f_{C_i}$ *is negative.*

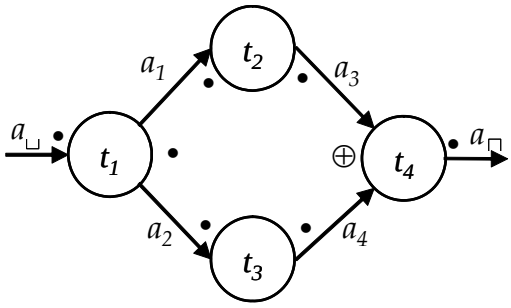

Figure 4: A workflow contradiction

Note that the execution of the workflow starts by enabling transition $a_{\sqcup}$, *i.e.* $a'_{\sqcup}$ *is true.*

On the other side, we still have the following equalities:

$b = (f_{E_1} \rightsquigarrow f_{C_1}) \wedge (\wedge(f_{E_i} \rightsquigarrow f_{C_i})) \wedge (f_{E_n} \rightsquigarrow f_{C_n}) = (a'_{\sqcup} \rightsquigarrow f_{C_1}) \wedge (\wedge(f_{E_i} \rightsquigarrow f_{C_i})) \wedge (f_{E_n} \rightsquigarrow a'_{\sqcap}), i \in \Gamma \backslash \{1, n\}$. *Bearing in mind these equalities and the fact of* b *being a contradiction, then one of the following cases must occurs:*

(1) $a'_{\sqcup} \rightsquigarrow f_{C_1}$ *is negative;* (2) *there exists* $i \in \Gamma \backslash \{1, n\}$ *such that* $f_{E_i} \rightsquigarrow f_{C_i}$ *is negative;* (3) $f_{E_n} \rightsquigarrow a'_{\sqcap}$ *is negative.*

Case 1. *Suppose that* (1) *holds. As* $a'_{\sqcup}$ *is true then* f_{C_1} *must be false and so the workflow stops its execution in task* t_1. *Therefore* $a_{\sqcap}$ *is not enabled.*

Case 2. *Suppose that* (2) *holds. Then either both* f_{E_i} *and* f_{C_i} *are false or* f_{E_i} *is true but* f_{C_i} *is false. In both situations the workflow stops its execution in task* t_i *and consequently* $a_{\sqcap}$ *is not enabled.*

Case 3. *Suppose that* (3) *holds. In this situation either both* f_{E_n} *and* $a'_{\sqcap}$ *are false or* f_{E_n} *is true and* $a'_{\sqcap}$ *is false. In both situations* $a'_{\sqcap}$ *is false and therefore* $a_{\sqcap}$ *is not enabled.*

Conclusion, in any of the previous cases $a_{\sqcap}$ *is not enabled, which means that the workflow does not logically terminates.*

Example 8 *The EA models of the workflow shown in figure 4 are:* $t_1 : a'_{\sqcup} \rightsquigarrow a'_1 \bullet a'_2, t_2 : a'_1 \rightsquigarrow a'_3, t_3 : a'_2 \rightsquigarrow a'_4, t_4 : a'_3 \oplus a'_4 \rightsquigarrow a'_{\sqcap}$. *Its logical workflow representation is* $b = (a'_{\sqcup} \rightsquigarrow a'_1 \bullet a'_2) \wedge (a'_3 \oplus a'_4 \rightsquigarrow a'_{\sqcap})$. *It can be easily proved using table 1 that all materialized workflow instances are* $false$. *Therefore, the workflow is a contradiction and does not logically terminate.*

Theorem 2 *A materialized workflow instance* WG_i *logically terminates if its logical workflow representation* b *is true.*

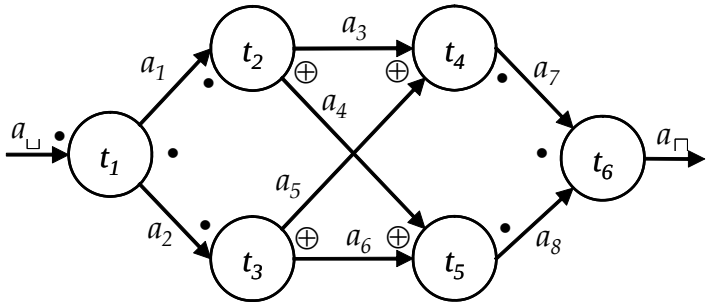

Figure 5: A workflow with a partial termination

Proof 2 *As* $b = \wedge(f_{E_i} \rightsquigarrow f_{C_i})$, $t_i : f_{E_i} \rightsquigarrow f_{C_i}$, $t_i \in T$, *for all* $i \in \Gamma$, *is true, then every* $f_{E_i} \rightsquigarrow f_{C_i}$ *is true, for* $i \in \Gamma$. *Consequently, for all* $i \in \Gamma$, f_{E_i} *and* f_{C_i} *are true. In particular* $a'_{\sqcup}$ *and* $a'_{\sqcap}$ *are true, which means that both* $a_{\sqcup}$ *and* $a_{\sqcap}$ *are enabled. As the workflow starts its execution by enabling* $a_{\sqcup}$, *then* $a_{\sqcap}$ *is enabled in some point in time after* $a_{\sqcup}$ *has been enabled. Therefore the workflow instance* WG_i *logically terminates.*

Theorem 3 *A materialized workflow instance* WG_i *does not logically terminates if its logical workflow representation* b *is* $false$.

Proof 3 *The proof follows immediately from Theorem 1.*

Theorem 4 *If a workflow* WG *is not a contradiction and is not a tautology then* WG *terminates for some, but not all, materialized workflow instances* WG_i. *In this situation we say that the workflow partially terminates.*

Proof 4 *If a workflow* WG *is not a contradiction then some materialized workflow instances* WG_t *are true. Also, if a workflow* WG *is not a tautology then some materialized workflow instances* WG_f *are false. Therefore, according to theorems 2 and 3, the materialized workflow instances* WG_t, *which are true, terminate and the materialized workflow instances* WG_f, *which are false, do not terminate.*

Example 9 *The logical workflow representation of the workflow shown in figure 5 is* $b = (a'_{\sqcup} \rightsquigarrow a'_1 \bullet a'_2) \wedge (a'_1 \rightsquigarrow a'_3 \oplus a'_4) \wedge (a'_2 \rightsquigarrow a'_5 \oplus a'_6) \wedge (a'_3 \oplus a'_5 \rightsquigarrow$

$a'_7) \wedge (a'_4 \oplus a'_6 \rightsquigarrow a'_8) \wedge (a'_7 \bullet a'_8 \rightsquigarrow a'_{\sqcap})$. *Consider the following materialized workflow instances: (1)* $a'_{\sqcup} = true; a'_1 = true; a'_2 = true; a'_3 = true; a'_4 = false; a'_5 = false; a'_6 = true; a'_7 = true; a'_8 = true; a'_{\sqcap} = true$*; (2)* $a'_{\sqcup} = true; a'_1 = true; a'_2 = true; a'_3 = false; a'_4 = true; a'_5 = true; a'_6 = false; a'_7 = true; a'_8 = true; a'_{\sqcap} = true$*. In both cases the materialized workflow instances are true. Now consider the following materialized workflow instance:* $a'_{\sqcup} = true; a'_1 = true; a'_2 = true; a'_3 = true; a'_4 = false; a'_5 = true; a'_6 = true; a'_7 = false; a'_8 = false; a'_{\sqcap} = false$*. In this case the materialized workflow instance does not logically terminate. Note that in this situation the materialized workflow instance stops its execution in task* t_3*, because its EA model is negative. Consequently the workflow does not logically terminate, but partially terminates.*

Once again it is important to state that for a task t, in the correspondent EA model $f_E \rightsquigarrow f_C$, the outgoing condition is enabled after the incoming condition is enabled, i.e. f_C can be true only if f_E is true. Consequently we cannot atribute values arbitrarily to the transitions a'_j, where $a'_j \in A'$. We allways must atribute values to a'_j, according to table 1.

Theorem 5 *Logical Workflow Termination. A workflow WG logically terminates if and only if b is a tautology.*

Proof 5 *Suppose that the workflow logically terminates. Then, for all materialized workflow instances* WG_i*,* $a_{\sqcap}$ *is enabled at some point in time after transition* $a_{\sqcup}$ *has been enabled. Let* $b = \wedge(f_{E_i} \rightsquigarrow f_{C_i})$*,* $t_i : f_{E_i} \rightsquigarrow f_{C_i}$*,* $t_i \in T$*, for all* $i \in \Gamma$*, be the logical representation of the workflow. We assume, by contradiction, that there exists* $i \in \Gamma$ *such that* $f_{E_i} \rightsquigarrow f_{C_i}$ *is negative. Then, both* f_{E_i} *and* f_{C_i} *are false, or* f_{E_i} *is true but* f_{C_i} *is false. As a consequence, the workflow stops its execution in task* t_i*. Therefore* $a'_{\sqcap}$ *is false and so* $a_{\sqcap}$ *is not enabled, which is a contradiction. Consequently, every compound EA model in WG is positive and b is a tautology.*

Conversely, if b is a tautology then every compound EA model in WG is positive. Therefore, for every $i \in \Gamma$*,* f_{E_i} *and* f_{C_i} *are true. In particular* $a'_{\sqcup}$ *and* $a'_{\sqcap}$ *are true, which means that both* $a_{\sqcup}$ *and* $a_{\sqcap}$ *are enabled. As the workflow starts its execution by enabling* $a_{\sqcup}$*, then* $a_{\sqcap}$ *is enabled at some point in time after* $a_{\sqcup}$ *has been enabled. Then the workflow logically terminates.*

Example 10 *It can be easily seen that the logical representation of the workflow shown in figure 1 is a tautology. Therefore, the workflow logically terminates.*

4 Using the Logical Workflow Termination Theorem

In this section we describe two applications areas for which the use of the logical workflow termination is indispensable.

4.1 Workflow Complexity Analysis

In a competitive e-commerce and e-business market, workflows can span both between enterprises and within the enterprise [28]. While organizations want their workflows to be simple, modular, easy to understand, easy to maintain and easy to re-engineer, in cross-organizational settings these processes have an inherent complexity. Nevertheless, in some cases, workflows' design can be highly complex, due, for example, to the vast number of transactions carried out in global markets. High complexity in a process has several undesirable drawbacks, it may result in bad understandability, more errors, defects, and exceptions leading processes to need more time to develop, test, and maintain. Therefore, excessive complexity should be avoided.

To achieve an effective process management, one fundamental area of research that needs to be explored is the complexity analysis of workflows [7]. Studies indicate that 38% of process management solutions will be applied to redesign enterprise-wide processes (source Delphi Group 2002). Workflow complexity can be defined as the degree to which a process is difficult to analyze, understand or explain. It may be characterized by the number and intricacy of tasks' interfaces, transitions, conditional and parallel branches, the existence of loops, roles, activity categories, the types of data structures, and other process characteristics [8].

Workflow complexity metrics can be used during the development of workflows to improve their quality and maintainability. But, it does not make sense to complexity analysis to workflows that are incorrectly designed and that do not logically terminate. Therefore, the logical workflow termination theorem has an important role in complexity analysis.

4.2 Semi-automatic design of workflows

A wide spectrum of workflow system architectures has been developed to support various types of business processes. Cardoso, Bostrom et al. [10] report that more than 200 workflow products are available in the market. Most of the systems provide a set of tools which include a graphical appli-

cation to design workflows and an engine or enactment system to manage the execution of workflow instances.

Although major research has been carried out to enhance workflow systems [22][12][26][3][19], the work on workflow application development lifecycles and methodologies is practically inexistent. The development of adequate frameworks is of importance to guarantee that workflows are constructed according to initial specifications. Furthermore, it would be advantageous for workflow analysts to have tools to support – automatically or semi-automatically – the design of workflows. Unfortunately, it is recognized that despite the diffusion of workflow systems, methodologies and frameworks to support the development of workflow applications are still missing.

To surpass this lack of tools to support the design of workflows, the Poseidon framework [9] has been developed. Poseidon framework helps analysts during their interviews with administrative staff, managers, and employees in general to design processes. The framework includes a set of procedures that guide the workflow analyst during his interviews and supply methods to semi-automatically design workflows. As a result, workflows can be developed and implemented more rapidly and accurately. The semi-automatic design of workflows can generate workflows which are incorrect and that do not logically terminate. Especially, since a semi-automatic design requires human involvement which can design inconsistent workflows. The use of the logical workflow termination theorem guarantees that the workflows semi-automatically generated terminate and that the subsequent alterations made by designers to workflows are consistent.

5 Conclusions

Workflow management systems are capable of hosting e-commerce applications by integrating business functionalities in a short time and with a low cost. This is of significant importance for global and competitive markets. Workflows describing e-commerce applications require a precise modeling, verification, and analysis to ensure that they perform according to initial specifications. The development of frameworks and theories to achieve an accurate modeling is a difficult task and the solutions proposed are still insufficient and more research is required.

To guarantee that workflows are successfully executed at runtime, it is necessary to verify their properties at design time. In this paper we present a formal framework, based on control flow graphs theory, to check workflow

specifications for correctness. In our approach we model workflows with trilogic acyclic directed graphs and develop a formalism to verify one important property: the logical termination of workflows.

The contribution of our work will enable the development of tools that will support and allow business process analysts to verify the correctness of their workflows at design time.

References

[1] W. M. P. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[2] Wil M. P. van der Aalst. 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, pages 161–183. Springer-Verlag, Berlin, 2000.

[3] G. Alonso, C. Mohan, R. Guenthoer, D. Agrawal, A. El Abbadi, and M. Kamath. Exotica/fmqm: A persistent message-based architecture for distributed workflow management. In *IFIP WG8.1 Working Conference on Information Systems for Decentralized Organizations*, Trondheim, Norway, 1994.

[4] Kemafor Anyanwu, Amit Sheth, Jorge Cardoso, John A. Miller, and Krys J. Kochut. Healthcare enterprise process development and integration. *Journal of Research and Practice in Information Technology, Special Issue in Health Knowledge Management*, 35(2):83–98, 2003.

[5] P. Attie, M. Singh., A. Sheth., and M. Rusinkiewicz. Specifying and enforcing intertask dependencies. In *Proceedings 19th Intlernational Conference on Very Large Data Bases*, pages 134–145, Dublin, Ireland, 1993. Morgan Kaufman.

[6] CAPA. Course approval process automation (capa). Technical report, LSDIS Lab, Department of Computer Science, University of Georgia, July 1, 1996 - June 30, 1997 1997.

[7] Jorge Cardoso. About the complexity of teamwork and collaboration processes. In Wojciech Cellary Esaki and Hiroshi, editors, *IEEE International Symposium on Applications and the Internet (SAINT 2005),*

Workshop - Teamware: supporting scalable virtual teams in multi-organizational settings, pages 218–221, Trento, Italy, 2005. IEEE Computer Society.

[8] Jorge Cardoso. Evaluating workflows and web process complexity. In Layna Fischer, editor, *Workflow Handbook 2005*, page 284. Future Strategies Inc., Lighthouse Point, FL, USA, 2005.

[9] Jorge Cardoso. Poseidon: A framework to assist web process design based on business cases. *International Journal of Cooperative Information Systems (IJCIS)*, (accepted for publication), 2005.

[10] Jorge Cardoso, Robert P. Bostrom, and Amit Sheth. Workflow management systems and erp systems: Differences, commonalities, and applications. *Information Technology and Management Journal. Special issue on Workflow and E-Business (Kluwer Academic Publishers)*, 5(3-4):319–338, 2004.

[11] Jorge Cardoso, Christoph Bussler, and Amit Sheth. Tutorial: Semantic web services and processes: Semantic composition and quality of service. In *International Federated Conferences: DOA/ODBASE/CooPIS 2002*, Irvine, CA, 2002.

[12] S. Ceri, P. Grefen, and G. Sanchez. Wide-a distributed architecture for workflow management. In *Proceedings of the 7th International Workshop on Research Issues in Data Engineering*, pages 76–79, Birmingham, UK, 1997.

[13] Umeshwar Dayal, Meichun Hsu, and Rivka Ladin. Organizing long-running activities with triggers and transactions. In *ACM SIGMOD international conference on Management of data table of contents*, pages 204–214, Atlantic City, New Jersey, 1990. ACM Press, New York, NY, USA.

[14] J. Eder, H. Groiss, and H. Nekvasil. A workflow system based on active databases. In G. Chroust and A. Benczur, editors, *Proceedings of CON '94, Workflow Management: Challenges, Paradigms and Products*, pages 249–265, Linz, Austria, 1994.

[15] D. Fensel and C. Bussler. The web service modeling framework, 2002.

[16] Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to infras-

tructure for automation. *Distributed and Parallel Databases, An International Journal*, 3(2):119–153, 1995.

[17] R. David Hall, John A. Miller, Jonathan Arnold, Krys J. Kochut, Amit P. Sheth, and Michael J. Weise. Using workflow to build an information management system for a geographically distributed genome sequence initiative. In R. A. Prade and H.J. Bohnert, editors, *Genomics of Plants and Fungi*, pages 359–371. Marcel Dekker, Inc., New York, NY, 2003.

[18] A.H.M. ter Hofstede and E.R. Nieuwland. Task structure semantics through process algebra. *Software Engineering Journal*, 8(1):14–20, 1993.

[19] N. R. Jennings, P. Faratin, T. J. Norman, M. P. O'Brien, E. Wiegand, C. Voudouris, J. L. Alty, T. Miah, and E. H. Mamdani. Adept: Managing business processes using intelligent agents. In *Proc. BCS Expert Systems 96 Conference*, pages 5–23, Cambridge, UK, 1996.

[20] M. H. Kang, J. N. Froscher, A. P. Sheth, K. J. Kochut, and J. A. Miller. A multilevel secure workflow management system. In Matthias Jarke and Andreas Oberweis, editors, *Proceedings of the 11th Conference on Advanced Information Systems Engineering*, Lecture Notes in Computer Science, pages 271–285, Heidelberg, Germany, 1999. Springer-Verlag.

[21] J. Klingemann, J. Wäsch, and K. Aberer. Deriving service models in cross-organizational workflows. In *Proceedings of RIDE - Information Technology for Virtual Enterprises (RIDE-VE '99)*, pages 100–107, Sydney, Australia, 1999.

[22] Krys J. Kochut, Amit P. Sheth, and John A. Miller. Orbwork: A corba-based fully distributed, scalable and dynamic workflow enactment service for meteor. Technical report, Large Scale Distributed Information Systems Lab, Department of Computer Science, University of Georgia, 1999.

[23] F. Leymann, D. Roller, and M.T. Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2):198–211, 2002.

[24] Zongwei Luo. *Knowledge Sharing, Coordinated Exception Handling, and Intelligent Problem Solving to Support Cross-Organizational Business Processes.* Ph.d. dissertation, University of Georgia, 2000.

[25] METEOR. Meteor (managing end-to-end operations) project home page, 2004.

[26] John A. Miller, D. Palaniswami, Amit P. Sheth, Krys J. Kochut, and H. Singh. Webwork: Meteor2's web-based workflow management system. *Journal of Intelligence Information Management Systems: Integrating Artificial Intelligence and Database Technologies (JIIS)*, 10(2):185–215, 1998.

[27] P. Muth, D. Wodtke, J. Weissenfels, G. Weikum, and A. Kotz Dittrich. Enterprise-wide workflow management based on state and activity charts. In A. Dogac, L. Kalinichenko, T. Ozsu, and A. Sheth, editors, *Proceedings NATO Advanced Study Institute on Workflow Management Systems and Interoperability*. Springer Verlag, 1998.

[28] Amit P. Sheth, Wil van der Aalst, and Ismailcem B. Arpinar. Processes driving the networked economy. *IEEE Concurrency*, 7(3):18–31, 1999.

[29] M.P. Singh. Semantical considerations on workflows: An algebra for intertask dependencies. In Paolo Atzeni and Val Tannen, editors, *Fifth International Workshop on Database Programming Languages*, Electronic Workshops in Computing, Umbria, Italy, 1995. Springer.

[30] Staffware. Staffware web site, 2005.

[31] TIBCO. Tibco inconcert, 2005.