Hindawi
Security and Communication Networks
Volume 2021, Article ID 3330923, 15 pages
https://doi.org/10.1155/2021/3330923

Research Article

A Workflow Criticality-Based Approach to Bypass the Workflow Satisfiability Problem

Monsef Boughrous and Hanan El Bakkali

Rabat IT Center, Smart Systems Laboratory (SSL), ENSIAS, Mohammed V University in Rabat, Rabat, Morocco

Correspondence should be addressed to Monsef Boughrous; monsef.boughrous@gmail.com

Received 8 April 2021; Accepted 31 August 2021; Published 25 October 2021

Academic Editor: Chien Ming Chen

Copyright © 2021 Monsef Boughrous and Hanan El Bakkali. 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.

Workflow management systems are very important for any organization to manage and model complex business processes. However, significant work is needed to keep a workflow resilient and secure. Therefore, organizations apply a strict security policy and enforce access control constraints. As a result, the number of available and authorized users for the workflow execution decreases drastically. Thus, in many cases, such a situation leads to a workflow deadlock situation, where there no available authorized user-task assignments for critical tasks to accomplish the workflow execution. In the literature, this problem has gained interest of security researchers in the recent years, and is known as the workflow satisfiability problem (WSP). In this paper, we propose a new approach to bypass the WSP and to ensure workflow resiliency and security. For this purpose, we define workflow criticality, which can be used as a metric during run-time to prevent WSP. We believe that the workflow criticality value will help workflow managers to make decisions and start a mitigation solution in case of a critical workflow. Moreover, we propose a delegation process algorithm (DP) as a mitigation solution that uses workflow instance criticality, delegation, and priority concepts to find authorized and suitable users to perform the critical task with low-security risks.

1. Introduction

Workflow management systems (WFMS) play a major role in all organizations. They allow organizations to automate, analyse, and control their business processes, which helps to increase productivity, achieve their business objectives, and improves the quality of their services. The workflow management coalition (WfMC) defines workflow as the partial or complete automation of a business process that allows organizations to describe and coordinate tasks and activities to achieve business goals following a defined set of rules [1].

Workflows can model any complex business process, and by using WFMS, organizations can execute their workflows, manage the sequences of work activities, and invoke appropriate resources [1]. However, there are significant challenges and problems to solve in order to make workflows more efficient, resilient, and secure.

In the recent years, many research works have proposed new access control models to formalize security policies in the context of workflow systems [2–4]. Mainly, these models are based on RBAC (role-based access control model) [5], which they extend to model the relations between users, tasks, roles, permissions, and other concepts. The use of an access control model that is suitable for workflow systems will allow a formal representation of security policies that reflects better access control constraints like separation of duties and facilitate their enforcement.

Workflow execution depends on several parameters. Thus, it is difficult to satisfy both business objectives [6] and access control constraints to ensure a timely and complete execution of a workflow instance. Moreover, in some cases, early user-task assignments may lead to users' unavailability for future tasks within the current workflow (or even other workflows). This problem is known in the literature as the workflow satisfiability problem (WSP), a problem that has been shown to be NP-hard [7, 8].

The WSP resolution consists of finding user-task assignments that ensure the termination of a workflow

instance in compliance with the business objectives [6] and without the violation of the security constraints [9], especially in environments like workflow systems, where events such as user unavailability can be a challenging issue for the workflow managers.

To bypass WSP and enhance workflow system resiliency, an approach was proposed in [10], which uses delegation and priority concepts to find a suitable and available user to perform the current task instance while satisfying the security constraints. Moreover, it defines workflow criticality and task instance priority to decide which task instance to suspend if necessary. In this paper, we aim to extend and enhance this approach by using a more realistic representation of the users' workload status and by defining new fine-grained metrics that measure task instance priority and workflow criticality.

The rest of this paper is as follows. Section 2 presents the workflow satisfiability problem, while Section 3 defines workflow resiliency and the importance of the delegation mechanism. In Section 4, we give an overview of related works. In Section 5, we present our approach and all the enhancements. A case study example is presented in Section 6 to illustrate the proposed approach. The delegation process is detailed in Section 7, and we conclude our paper in Section 8.

2. Workflow Satisfiability Problem (WSP)

Nowadays, workflow systems are widely used in different areas and often handle critical and sensitive information and/or resources. Therefore, workflow system security is of particular interest to the security community [11]. Protecting workflows means principally protecting workflow resources from unauthorized access. Therefore, organizations use access control models for their workflow authorization systems as a measure that helps to manage access to resources and enforces the security policy [4, 5].

Access control constraints specify the security policy, which defines the set of authorizations and rules to ensure a secure task execution. Moreover, it specifies the security constraints between the different tasks in the workflow to prevent users from performing conflictual tasks or holding conflictual roles [7]. An example of those constraints enforced by the workflow authorization systems is the separation of duties (SoD), which prevents the assignment of the same users to tasks with conflict of interest [11]. In addition, there is the binding of duties (BoD) constraint, which requires that the same user must execute a particular set of tasks [7, 9].

The use of authorization constraints prevents security problems and frauds. Thus, only qualified users can execute tasks and get access to sensitive information needed to complete their duties [12]. On the other hand, workflow execution also depends on workflow specifications (business objectives), which defines the set of tasks and their order of execution that comply with the business objectives [12].

Therefore, the WSP a fundamental problem in workflow systems rise [7]. The WSP asks whether there exists a valid assignment plan of authorized users to workflow tasks that satisfy the workflow specification and the authorization constraints [7, 8]. In other words, the WSP takes a workflow specification as input and returns a valid plan that satisfies the security constraints if one exists and a null value otherwise [6].

Often WSP occurs when workflow specifications are unsatisfiable. In practice, to complete the execution of a given workflow without the WSP situation, the workflow manager has to define a valid plan of user-task assignments that satisfies the workflow specifications and the authorization constraints [6, 9, 12]. However, workflow managers cannot manage to redefine new valid assignment plans in highly dynamic systems such as workflow systems, especially during workflow run-time. Moreover, the trade-off between workflow authorizations and business objectives leaves workflow managers in a hard situation to make decisions about the workflow execution. Therefore, several algorithms have been proposed in the literature to solve the WSP [7, 8, 10, 13, 14].

In an interesting work by Wang and Li [7], the authors used tools from the parameterized complexity to measure the computational complexity of the WSP with multiple input parameters. Studying the computational complexity of the WSP, the authors showed that the WSP is NP-complete. Moreover, they proved that WSP is a fixed parameter tractable (FPT), and efficient algorithms to solve WSP are possible when only inequality and equality relations are used (SOD/BOD). Furthermore, they had shown that WSP is no more FPT if user-defined relations are allowed [7].

On the other hand, Crampton et al. [8] extended the results of Wang and Li [7] by introducing the counting constraints (a generalization of the cardinality constraints). Furthermore, they consider entailment constraints to encode other types of requirements that cannot be represented using counting constraints. Their results showed that counting constraints have no effect on the fixed-parameter tractability of the WSP and suggests that the WSP remains FPT for any constraint whose satisfaction is phrased in terms of the steps that a single user performs [8].

Given the hardness of the problem and the fact that unsatisfiable workflows may never be completed without violating the security policy [6, 10], it is important to check workflow satisfiability before and during its execution. Therefore, applying some appropriate mitigation solutions, such as delegation, increases the flexibility and resiliency of the workflow and decreases the probability that the WSP occurs.

3. Workflow Resiliency and Delegation

3.1. Workflow Resiliency. As described in the previous section, solving the WSP is hard, in general, and proven to be NP-hard [7, 8], which means that naive algorithms for solving the WSP may try all possible combinations of user-task assignments before finding a valid plan that satisfies the workflow specifications and authorizations [15].

However, ensuring a workflow to be satisfiable for a given access control state is not enough because the execution of a workflow designed with critical tasks requires to

guarantee user availability throughout the workflow execution [7, 12].

In practice, users become unavailable because of different reasons some of which may be unexpected (e.g., vacation, sickness, or dynamic access control constraints). As a result, a satisfiable workflow at administration-time may become unsatisfiable during run-time [10, 12].

The workflow resiliency problem extends the WSP by finding a valid plan of user-task assignments that satisfies the workflow specifications and the authorization constraints, under the assumption that some users may become unavailable [7]. In other words, a workflow is resilient if the WFMS can find an available user to execute a critical task in compliance with the workflow specifications and the security policy.

The notion of resiliency policies in the context of access control systems is formally introduced by Li et al. [16], but there is a major difference with resiliency in the context of workflow systems. Wang and Li [7, 17] studied resiliency in workflow systems and the computational complexity of checking resiliency. They define three levels of resiliency based on user availability. In static resiliency, some users may be absent before a workflow instance execution; in decremental resiliency, users may be absent before or during a workflow instance execution, but absent users do not become available again; and in dynamic resiliency, users may become absent and available again.

Furthermore, Wang and Li showed that for a given access control state checking static workflow resiliency is NP-hard, while checking decremental and dynamic resiliencies is PSPACE-complete. In addition, they observed that other formulations of resiliency are possible and can be of interest [7, 17].

3.2. Delegation Mechanism. In real-world situations, users' availability is an obvious requirement that impacts workflow resiliency and workflow system flexibility. Delaying tasks because of the unavailability of some users violate time constraints on the entire workflow execution. Moreover, it may cause workflow unsatisfiability if it occurs during workflow run-time. There are many reasons for users' unavailability, and some of them are unexpected like sickness, while others are tractable such as workload, leave, and security policy specifications.

In order to handle such exceptions, a delegation mechanism is a suitable approach that ensures alternative scenarios; it provides flexibility in workflow systems and enhances their resiliency and efficiency [18, 19]. In addition, it provides flexibility in access control models allowing access rights delegation across multiple security domains for collaborative activities [9, 18].

Generally, the delegation mechanism describes the process where the delegator (user A) delegates (grants or transfers) his rights to the delegatee (user B). Thus, user B gets all the needed permissions to perform the delegated task on behalf of user A [10, 18].

The authors in [18] presented a taxonomy of delegation. They distinguish two types of delegation: user delegation in which users can delegate their rights without administrator involvement. While in the administrative delegation, the administrator assigns rights to users. Furthermore, they define delegation characteristics such as permanency/temporary delegation and delegation monotonicity that specify if the delegator can use his delegated rights in parallel with the delegatee. In addition, they define delegation multiplicity, which specifies if the same role can be delegated to multiple users at any given time, thence they describe the forced or autorevocation of the delegated privileges.

The delegation of entities may concern elementary permissions (rights), roles, or even performing tasks [18]. However, using the delegation mechanism arbitrarily may cause a breach of security policies and/or a lack of quality in work [10].

4. Related Works

Several authors in the literature studied the workflow satisfiability problem (WSP) and tried different techniques to solve this problem. Among the approaches used to solve this problem are the FPT algorithms, model checking, quantitative approaches, delegation mechanism, policy properties, etc. [20].

Wang and Li [7] studied the computational complexity of the WSP and showed that it is NP-complete. Moreover, they showed that the WSP is NP-hard if the workflow authorization system supports constraints (SOD, BOD, etc.). Moreover, the authors measured the computational complexity of the WSP with multiple input parameters, since the number of tasks is typically smaller than the number of users. Thus, they have proved that the WSP is FPT when only inequality and equality relations are used (SOD/BOD). Therefore, they proposed an algorithm to reduce the WSP to SAT, the reduction is a fixed-parameter reduction, and it allows solving the WSP using the SAT solver [7]. Their research work using the theory of parameterized complexity inspired other researchers to address the WSP using the theory of parameterized complexity [8, 13, 14].

Crampton et al. in [8] reduced the WSP to a max weighted partition problem, which guarantees that an instance of WSP is FPT. Thus, it allows the authors to develop an FPT algorithm to solve the WSP. Moreover, the authors considered preprocessing to solve the WSP more quickly and showed that an instance of WSP in which all constraints are of type 1 (entailment constraints with singleton sets of tasks) and only includes BoD and SoD constraints admits a polynomial kernel. The kernelization allows the transformation of the WSP instance to a smaller instance with the number of users is at most the number of steps k, which will significantly reduce the complexity of solving the WSP instances [8]. However, certain instances of WSP do not have polynomial-size kernels. In such a case, the authors suggested to simply apply the technique they described to reduce the WSP to a max weighted partition.

Cohen et al. in [6] used constraint satisfaction problem (CSP) techniques and the notion of plan-indistinguishability to introduce an algorithm, which is applicable to a wide range of workflow constraints. Thus, the authors proposed a

generic algorithm that builds executions incrementally, discarding partial executions that can never satisfy the constraints. Moreover, they showed that the algorithm could be optimized for a wide class of user-independent constraints. Furthermore, the generic algorithm can deal with unions of different types of constraints. Cohen et al. proved that the algorithm is a fixed-parameter algorithm for equivalent relation constraints and their union with user-independent constraints [6].

In [14], Cohen at al. described the implementations and compared their experimental outcomes to the approach presented in [7]. They demonstrated the practicality of the generic algorithm proposed in [6] by adapting it to the WSP with user-independent constraints and developed its implementation in the case of counting constraints. Thus, the authors compared the performance of the two approaches in a set of computational experiments. The results of the experiments suggest that the generic algorithm has an advantage over SAT4J when solving the WSP, but this advantage does not extend to lightly constrained instances of the problem.

Gutin et al. in [13] studied the kernelization of the WSP and proved that the algorithms described in [6, 8] to solve the WSP for regular and user-independent constraints are probably optimal. Thus, they cannot be solved in time $O(2^{ck})$ and $O(2^{ck\log(k)})$, respectively, for any $c < 1$ unless the strong exponential time hypothesis (SETH) fails [21].

Mace et al. in [22] defined the notion of quantitative resiliency by encoding the workflow satisfaction and resiliency problem as a Markov decision process (MDP). Thus, they have reduced the problem of finding an optimal plan of user-task assignment to that of solving an MDP. Mace et al. had focused their work on the decremental and dynamic resiliency [7, 17], and they suggested to address the resiliency problem from a quantitative viewpoint. Therefore, the authors provided tools and metrics that indicate a degree of satisfaction and/or resiliency for a given workflow and the likelihood of its termination for a given security policy and user availability model. In [23], the same authors extended their approaches by considering workflows with choice, which gives different resiliency values for each execution path. Thus, they define resiliency variance as a metric for workflow failure risk analysis. It indicates overall resiliency variability (volatility) from the resiliency average. The authors claim that the expected resiliency and resiliency variance together are useful for predicting a suitable mitigation strategy [23].

Basin at al. in [24] addressed the problem of how to balance between empowerment and protection by mapping authorization administration to an optimization problem. Therefore, they present an approach to find an authorization policy that maximizes protection, minimizes the cost associated with the administrative change, and empowers users to do their job while satisfying the policy. Moreover, the authors showed that finding an optimal role-based authorization policy that allows a workflow execution is NP-complete.

In [25], Crampton et al. suggested that systems in which the set of available authorized entities is unpredictable over time require delegation mechanisms. Thus, the authors defined the notion of the availability and presented an autodelegation mechanism that can respond automatically to the absence of qualified authorized users.

The approach presented in [10] aims at enhancing workflow system resiliency using the delegation mechanism, the task priority concept, and workflow criticality value to bypass the WSP at run-time. The delegatees are chosen based on their suitability but may lack competence, which is "the price to pay" for resiliency. As delegation takes place at a task level, it is not currently clear whether a workflow can still complete while meeting security constraints [10].

In [26], Lowalekar et al. developed two approaches, namely, exhaustive search and quadratic programming to find user-task assignments that satisfy workflow authorization policy and SoD constraints such that the workflow can be completed even if k users fail. They considered two types of scenarios, purely static, where user-task assignments are fixed before the execution of workflow and thus failure resilience has a fixed value and purely dynamic, where user-task assignment can change at run-time to get more failure resilience and thus failure resilience changes dynamically with a lower bound. Furthermore, they showed that the quadratic programming approach is more efficient.

Table 1 resumes the related works presented in this section based on the used approach for the WSP or its variant, the resiliency problem.

5. Workflow Criticality-Based Approach

As shown above, the WFMS forms an essential part of any organization's information system. However, workflows in some cases are unsatisfiable because of users' unavailability as described in previous sections (cf. Section 2 and 3.1). Often rigid workflow authorization systems and the strict application of the security policy make solving the workflow satisfiability problem (WSP) more difficult. Therefore, many solutions proposed in the literature (cf. Section 4) aims to solve the WSP and improves workflow system resiliency. In this section, we present our approach, which enhances the approach presented in [10] and uses delegation and workflow criticality to bypass WSP and ensure workflow system resiliency.

5.1. Task Priority. We redefine the task representation described in [10], and we propose that each task T is represented as follows (W, P, D, H, C, T-kind), where W is the workflow that contains the task T and P is the set of needed privileges to perform T. While, D is the number of remaining days to complete a task, H is the number of working hours needed to complete a task, and C describes the skills needed to perform a task. T-kind represents a succession of seven digits that identify the task characteristics, 1 if a characteristic holds and 0 if not.

There are seven task characteristics: Atomic/Composite, Delay-Sensitive/NonDelay-Sensitive, Delegable/Non-Delegable, Critical/Optional, Human/Automatic, Interruptible/NonInterruptible, and Preemptable/NonPreemptable.

Table 1: Classification of the papers.

Problem	Used approach	Papers
WSP	Initial works	[7, 17]
	FPT algorithms	[6, 8, 13, 14, 21]
	Others	[15, 24]
Resiliency	Static, dynamic	[7, 26]
	Quantitative	[12, 22, 23]
	Delegation	[10, 25]

Thus, the tuple (T, ω, st) represents a task instance "t," which refers to the instantiation of a task T, within a workflow instance "ω," and the status "st" indicates the task instance state, which can be initiated (activated), assigned, started, interrupted, cancelled, failed (aborted), or completed.

We have to mention that task characteristics as atomic/composite and human/automatic do not affect the task priority value; therefore, only the remaining characteristics are used to qualify the priority of a task within a workflow. Moreover, evaluating a task as optional means that the task does not have characteristics that cause a high priority task. Therefore, optional tasks refer to tasks that are by default interruptible (the execution can be paused and resumed later), nondelay-sensitive (the task has no time execution constraints), preemptable (the user who starts the execution can be replaced), and delegable (the task has no confidentiality constraints it can be delegated). Otherwise, T-kind indicates the task characteristics, and in the worst cases, the task priority is critical and reflects the opposite of an optional task.

Additionally, we highlight that the workflow manager specifies the task characteristics during task administration-time. Most of them are unchangeable during the execution of the workflow except for the delay sensitivity characteristic of a task because it is a time-based constraint, and it reflects the execution time of a task and its remaining time to be completed. Thus, the status of a task that is nondelay-sensitive tends to be delay-sensitive when closer a task deadline is.

The algorithm is presented in a manner to cover all the possible combinations of the task characteristics in the T-kind. Thus, the task priority $\Pr(T)$ takes values in $[0, 1]$ interval, since later in the paper this value will be used to calculate the criticality of a task instance $\Pr(t)$. Furthermore, $\Pr(t)$ results from multiplying the $\Pr(T)$ with a probabilistic value that we will present later in the paper. Therefore, we choose a 0.2 step value to cover the interval and fit in all the task characteristics.

To identify the task priority denoted $\Pr(T)$, we propose Algorithm 1, which associates a priority value for a task based on its characteristics.

The algorithm described in [10] gives a value of $\Pr(T) = 1$ to nondelegable tasks, but a nondelegable task can be interruptible or noninterruptible and also can be delay-sensitive or delay-insensitive. In such cases, the algorithm we define gives a value of $\Pr(T) = 1$ to tasks with delay-sensitive, nondelegable, and noninterruptible characteristics. Thus, it gives a value of $\Pr(T) = 0.8$ as priority for tasks with delay-sensitive, nondelegable, and interruptible characteristics.

5.2. User Workload and Availability. In this section, we define the user workload, denoted $\mathrm{L(u)}$, which defines the percentage of user availability for a given day from a workload perspective. This user workload information aims to facilitate the WFMS to dynamically load balance work hours among users who can perform the current task. Thus, more potential delegatees are available for the delegation process from a workload perspective,

$$L(u) = \frac{\sum h_i}{nb} \times 100. \quad (1)$$

The $\sum h_i$ is the daily workload in hours that a user must work in order to complete its assigned tasks, and h is the number of daily work hours to complete a task i, while nb is the daily working hours imposed by the organization. A user who is unavailable because of sickness, leave, or any other reason has an $L(u)$ value of 1.

For example, a user u who is participating in the execution of two workflows $W1$ and $W2$ in a given day d is assigned to tasks $T2(W1, P2, D2, H2, C2, T2\text{-}kind)$ and $T3(W3, P3, D3, H3, C3, T3\text{-}kind)$, respectively. Thus, the value $L(u)$ for day d is $L(u) = (h2 + h3)/nb \times 100$.

We have to mention that h and H are different, since H is the global number of work hours needed to complete the task, while h is the number of work hours per day needed to complete the task in accordance with D the number of remaining days to finish the task. The minimum value for D is $D_{min} = H/nb$, and the maximum value for $\sum h_i = nb$.

In addition to user workload, the availability of a user for a task depends on the amount of time needed for its execution. Thus, we denote $c(T)$ the amount of time needed for the task execution, and $c(T) = H/(D \times nb) \times 100$.

Therefore, a user is available from the workload point of view for the execution of a given task, if the following condition is valid: $100 - L(u) \geq c(T)$.

We use this condition in the function *Load_Status* as described in Algorithm 2. This function takes as parameters a set of users assigned to an authorized role needed for a given task. Thus, it returns, as a result, an ordered set of available users who can perform the task from the workload perspective.

For better accuracy, we have extended the three states of a user's workload (available, loaded, and unavailable) proposed in [10]. Since, a task which requires from a user 50% of its time to be free cannot be delegated to a busy one with 70% of workloads even if it is still considered available with 25% of free time. Measuring the percentage of users' workload availability gives a more accurate view of each user workload state, which allows managing delegations between users in a fair and efficient manner.

5.3. Workflow Criticality. The workflow criticality value denotes that $Cr(W)$ is an important metric that will help the WFMS during the delegation process to bypass the WSP situation [10]. However, using only four static values 0.25 (reduced), 0.5 (medium), 0.75 (important), and 1 (critical) for describing such an important metric does not provide the

```
    //if a Task T is NonDelegable => its NonPreemptable.
    // if a Task T is Delegable => its Preemptable.
(1) if T is Optional then Pr(T) ⟵ 0
(2) else if T is Delay-Sensitive and NonDelegable then
(3)       if T is NonInterruptible then Pr(T) ⟵ 1
(4)       else if T is Interruptible then Pr(T) ⟵ 0.8
(5)       end if
(6) else if T is Delay-Sensitive and Delegable then
(7)       if T is NonInterruptible and NonPreemptable then Pr(T) ⟵ 1
(8)       else if T is Interruptible and NonPreemptable then Pr(T) ⟵ 0.8
(9)       else Pr(T) ⟵ 0.6 /* for NonInterruptible and Preemptable or interruptible and Preemptable. */
(10)      end if
(11) else if T is NonDelay-Sensitive and NonDelegable then
(12)      if T is NonInterruptible then Pr(T) ⟵ 0.8
(13)      else if T is Interruptible then Pr(T) ⟵ 0.6
(14)      end if
(15) else if T is NonDelay-Sensitive and Delegable then
(16)      if T is NonInterruptible and NonPreemptable then Pr(T) ⟵ 0.6
(17)      else if T is Interruptible and NonPreemptable then Pr(T) ⟵ 0.4
(18)      else Pr(T) ⟵ 0.2
(19)      end if
(20) end if
```

Algorithm 1: Task priority algorithm.

```
    //n: the number of users to verify.
    //m: the number of tasks that the user i executes.
(1) while i ≤ n do
(2)    for k ⟵ 1, m do
(3)       UH ⟵ UH + U_k // UH = ∑H_i
(4)    end for
    // L(U) = UH/nb
    // c(T) = H(T)/(nb × D)
(5)    if 1 − L(U) ≥ c(T) then
    //Table of available users ordred by charge.
(6)       Tab[x] ⟵ U_ij // U_ij = Non_Conflict((U(R_i), t), T)
(7)       x ⟵ x + 1
(8)    else i ⟵ i + 1
(9)    end if
(10) end while
```

Algorithm 2: Load_Status algorithm.

WFMS with enough information about workflows in order to bypass the WSP.

Therefore, we proposed a method to evaluate workflow criticality such that $Cr(W) \in]0,1[$. This new metric can be used by workflow managers to make decision, which task instance to suspend or even proceed to workflow reengineering if necessary. Moreover, this metric can be used to analyse workflows by defining a threshold (cf. Section 5.4) for workflow deadlock risk (resiliency problem), the threshold value helps to make decision to start the delegation process or applying another kind of possible mitigations.

In order to compute the workflow criticality $Cr(W)$, we have to compute the criticality of each task instance within the workflow because they affect directly the workflow criticality. One critical task instance may cause the failure of the workflow instance. Therefore, we assume that the criticality of a workflow $Cr(W)$ is equal to the maximum value of criticality calculated for a task instance that belongs to that workflow, and the task instance criticality is denoted as $Cr(\omega(t))$:

$$\begin{aligned} Cr(W) &= max_{t_i \in W} Cr\left(\omega\left(t_i\right)\right) \\ &= \left\{t_i \in W \mid \forall t_j \in W, Cr\left(\omega\left(t_j\right)\right) \leq Cr\left(\omega\left(t_i\right)\right)\right\}. \end{aligned} \tag{2}$$

Computing the criticality of task instance is not easy; for doing so, we have to understand what makes task instances critical. In the context of workflow satisfiability and resiliency, a task instance can be qualified as critical if its assigned

user becomes unavailable (event A) and the WFMS cannot find an available user (the delegatee) to perform the task (event B). Thus, $Cr(\omega(t))$ is given by

$$Cr(\omega(t)) = P(A) \times P(B). \quad (3)$$

It is worth pointing out that understanding users' availability is an obvious requirement to calculate the probability of a user becoming unavailable. Thus, the value of this probability may be deduced from a mixture of operational logs, behavioural analysis, users' tentative absences, and even users' medical records. Describing a method to compute the probability of users' availability is beyond the scope of this paper. Even though, we have to model this probability to calculate the criticality of a workflow instance $Cr(\omega(t))$. Therefore, we use the Poisson distribution to model the probability of users' unavailability, since it describes the distribution of infrequent events, which we can assume for user availability inside organizations. Moreover, the Poisson distribution focuses on the number of discrete events (occurrences over a specified period), and we assume that each event is independent of the other events.

We assume that data logs about users' availability and previous works (executed tasks) are provided to the WFMS in order to compute the probabilities. Formally, computing $P(A)$ is done using Poisson distribution. The value of the parameter λ, which is the average rate of users' unavailability for a specified interval of time (e.g., a month) can be estimated using previous users' logs,

$$Thus, P(A) = P(a \geq 1) = 1 - e^{-\lambda}. \quad (4)$$

Mainly, workflow authorization constraints affect users' availability for a given task. The authors in [9] presented a set of security requirements that are essential to ensure workflow system security. Thus, these requirements we believe are essential for workflow security and may cause users' unavailability. We have separation of duty (SoD), binding of duty (BoD), cardinality (Card), context awareness (CA), least privilege (LP), and privacy preservation (PrP) [9].

Therefore, to compute the probability that a potential delegatee becomes unavailable $P(B)$, we use a Bayesian network (BN) to model the flow of causality between the different security constraints. Thus, each node in the BN represents a security requirement that we believe may be the leading cause for a potential delegatee to become unavailable (Figure 1). The BN reflects how the enforcement of the security requirements in authorization systems could cause workflow users' unavailability. Furthermore, the order and relations between the nodes are chosen based on the definitions of the security constraints and reasonably to reflect their causal relations.

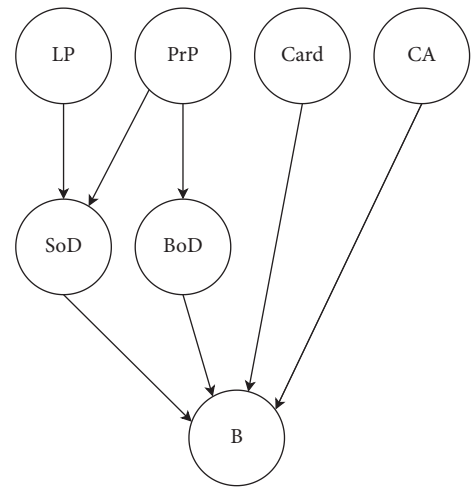

Figure 1: Bayesian network model.

To understand the causal relationship between the nodes, we took the example in Figure 1, where the SoD node has relations with LP and PrP nodes as parents. These relations reflect the fact that the LP principal supports the SoD constraint by definition [9]. Furthermore, the PrP constraint is needed as a parent for SoD, since the latter prevents users from holding exclusive roles and also from executing tasks with conflict of interest from privacy and confidentiality perspectives.

The values in the conditional probability tables of each node in the BN are updated each time we get new data about the security constraints and their likelihood to cause the unavailability of delegatees for a given task.

The joint probability distribution over all variables X_i (security constraints) in a Bayesian network is given in the following equation, where $Parent(X_i)$ is the immediate predecessor of X_i in the BN [27, 28]:

$$P(x_1, \ldots, x_2) = \prod_{i=1}^{n} P(x_i | Parents(X_i)). \quad (5)$$

In order to compute $P(B)$, we have to express the joint probability using equation (5) for the BN in Figure 1. However, to lighten the formula, we chose to take off CA and card constraints from the BN. The reason why we kept only the security constraints LP, PrP, SoD, and BoD is because these constraints are necessary and must be implemented in any workflow authorization system. Therefore, we consider the BN in Figure 2 for $P(B)$ computation using the following equation for joint probability:

$$P(B, SoD, BoD, LP, PrP) = P(B|SoD, BoD)P(SoD|LP, PrP)P(BoD|PrP)P(LP)P(PrP). \quad (6)$$

Using evidential reasoning, we can use the joint probability in equation (5) to express the probability of any desired query in terms of the conditional probabilities specified in the network [27, 28]. Assuming that the desired query Q, the evidence variables $(E_1, \ldots, E_m)$, and the hidden variables $(H_1, \ldots, H_k)$ represent all the variables

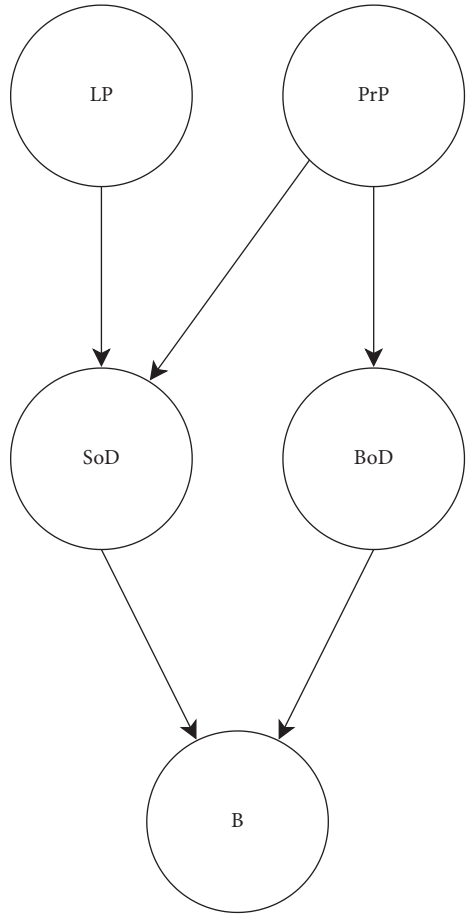

Figure 2: Simplified Bayesian network.

$(X_1, \ldots, X_n)$ in a Bayesian Network. Thus, the probability of Q, given the evidences, is given by

$$P(Q|E_1 = e_1, \ldots, E_m = e_m) = \frac{\sum_{h_1,\ldots,h_k} P(Q, h_1, \ldots, h_k, e_1, \ldots, e_m)}{\sum_{Q,h_1,\ldots,h_k} P(Q, e_1, \ \ldots, e_m)}. \tag{7}$$

Computing the probability of the event B, given the security constraints can be expressed using equation (7). Moreover, we can consider our Bayesian network as a causal network; thus, the parents of each node are their direct causes [28]. Therefore, the probability of event B to be true or false can be expressed over possible observed events (true or false) on the security constraints directly connected to B in the network, namely, SoD and BoD.

For example: $P(B|SoD = True, BoD = false)$ is given by

$$\begin{aligned} P(B|SoD = True, BoD = false) &= \frac{P(B, SoD, BoD)}{P(SoD, BoD)}, \\ Or, P(B, SoD, BoD) &= \sum_{LP,\, PrP} P(B, SoD = True, BoD = false, LP, PrP), \\ And, P(SoD, \ BoD) &= \sum_{B,\, LP,\, PrP} P(SoD = True, BoD = false). \end{aligned} \tag{8}$$

5.4. Criticality Threshold. As we have mentioned earlier in this paper, the workflow criticality value can be used to make decisions about workflow execution or the necessity for applying workflow reengineering or other kinds of mitigations to override the WSP. However, to make such decisions, the workflow manager needs a

threshold value in addition to the workflow criticality value.

Computing the threshold value can be easily done if we have enough data about the executions of a given workflow, but it is harder for a newly designed workflow or if we have no data about the history of the execution of a workflow.

Therefore, we suggest θ as a threshold value for workflow criticality, which represents the higher bound of the confidence interval of the lowest task instance criticality value in a given workflow, with 95% confidence level (different confidence levels can be used).

The computation of the threshold value θ is as follows:

$$\theta = \frac{e^{\varphi}}{1 + e^{\varphi}}, \quad or, \varphi = \widehat{\varphi} + 1.96\sqrt{\frac{1}{I(\widehat{\varphi})}}. \tag{9}$$

We can find $\widehat{\varphi}$, the maximum likelihood estimator of φ, using the logit transformation to β as described in equation (10), and β is the $min_{t_i \in W} Cr(\omega(t_i))$ described in equation (12). Thus, we use equation (11) to compute the fisher information for $\widehat{\varphi}$.

$$\widehat{\varphi} = \log\frac{\beta}{1-\beta}, \tag{10}$$

$$and, I(\widehat{\varphi}) = \frac{I(\beta)}{\left[\varphi'(\beta)\right]^2} = n\beta(1-\beta), \tag{11}$$

$$\beta = min_{t_i \in W} Cr\left(\omega\left(t_i\right)\right) = \left\{t_i \in W | \forall t_j \in W, Cr\left(\omega\left(t_i\right)\right) \leq Cr\left(\omega\left(t_j\right)\right)\right\}. \tag{12}$$

The workflow manager can make decisions easily by comparing workflow criticality $Cr(W)$ and the criticality threshold value θ. Especially, if the workflow criticality value is higher than the threshold value, which means it is highly likely that WSP occurs. Thus, it would be better to suspend the workflow or the critical tasks within the workflow in order to apply the delegation process or any other possible mitigation solutions.

5.5. Task Instance Priority. Task instance priority denoted $\Pr(t)$ is calculated for each task T in workflow instance ω using the task priority $\Pr(T)$ and the criticality of the task instance $Cr(\omega(t))$. The value of the task instance priority is used in the delegation process through the function *Priority_DP* described in the algorithm (5.3).

Thus, the task instance priority is given by

$$\Pr(t) = \Pr(T) \times Cr(\omega(t)). \tag{13}$$

The delegation process (cf. Section 7) calls the function *Priority_DP* presented in Algorithm 3 to check the possibility to delegate the critical task for a user who does not create any conflict of interest, security issue, or causes another critical task after the delegation.

Therefore, the function *Priority_DP* checks if the assigned task to the potential delegatee can be interrupted and does not have any time constraints. Thus, the Priority_DP function returns as value, a user, from an ordered set of users *Non_Conflict* or *Non_Conflict_CT* that are supposed to be WSP free.

Furthermore, the function verifies if the task that will be interrupted has a low task instance priority $\Pr(t')$, and also if the task instance criticality $Cr(\omega(t'))$ is lower than the workflow threshold value.

6. Case Study

In order to illustrate the usefulness of the proposed approach to measure workflow criticality and the threshold value, we performed a case study using a sample workflow with BoD (==) and SoD (=/=) as security constraints between some of the interconnected tasks. We considered the following workflow presented in Figure 3.

The workflow is decomposed into seven tasks $W = \{T1, T2, \ldots, T7\}$ and a set of five users $U = \{U1, U2, \ldots, U5\}$ with the relevant skills to execute these tasks. Thus, we assume that assignments of roles and tasks respect the predefined security requirements. The user-task assignments and task priorities are given in Table 2.

We consider dynamic user availability, which means that any user who becomes unavailable for a task may become available again for assignment at any step later in the workflow execution. As described in [23], understanding when users will and will not be available may be deduced from a mixture of operational logs, behavioural analysis, and user tentative absences. In our case, we will assume that user availability can be described using Poisson distribution with λ^* the average rate of users unavailability in a given period (e.g. month).

Therefore, the probability $P(A)$ as described in section (5.3) can be computed using equation (4). Thus, for $\lambda^* = 1, P(A) = 1 - e^{-1} = 0.632120559$.

The next step in order to compute the criticality of each task instance in the workflow we have to compute the probability $P(B)$ for those tasks as described in section (5.3) using equation (7). Figure 4 illustrates the probability tables of each node in the Bayesian network (BN) and the queries on B given all possible observations on the security constraints.

The hardest part for computing workflow criticality is the computation of the probabilities of events A and B. Using the values of $P(A)$ and $P(B)$ in equation (3) for each user-task assignment allows the computation of the criticality of each task instance $Cr(\omega(t))$. The accuracy of the computation of the probabilities $P(A)$ and $P(B)$ depends on the quality and the quantity of the data we have about user availability and the security constraints.

Table 3 gives the computation results of $Cr(\omega(t))$ for an average rate of user unavailability $\lambda^* = 1$, which means $P(A) = 0.632$. In addition to the task instance priority values $Pr(t)$ computed for each task using equation (13) described in section (5.5).

```
(1) for each user U_k in "Non_Conflict" or "Non_Conclict_CT" do
(2)   if t' ← U_k in ω' and st(t') ← ("Assigned" or "Started") then
(3)     if T' is "Optional" then
(4)       st(t') ← "Cancelled"
(5)       Return U_k
(6)     else if T' is "Interruptible" then
(7)       if T' is "NonDelay_Sensitive" or [T' is "Delay_Sensitive" and (H + H' < D × nb)] then
(8)         if U_k ∈ "Non_Conflict" then
(9)           if Pr(t') < Pr(t) and Cr(ω(t')) ≤ Theta(ω) then
(10)            st(t') ← "Interrupted"
(11)            Return U_k
(12)          else
(13)            Return "null"
(14)          end if
(15)        else if U_k "Non_Conflict_CT" then
(16)          if Cr(ω(t')) ≤ Theta(ω') then
(17)            st(t') ← "Interrupted"
(18)            Return U_k
(19)          else
(20)            Return "null"
(21)          end if
(22)        end if
(23)      else
(24)        Return "null"
(25)      end if
(26)    else
(27)      Return "null"
(28)    end if
(29)  else
(30)    Return U_k
(31)  end if
(32) end for
```

Algorithm 3: Priority_DP algorithm.

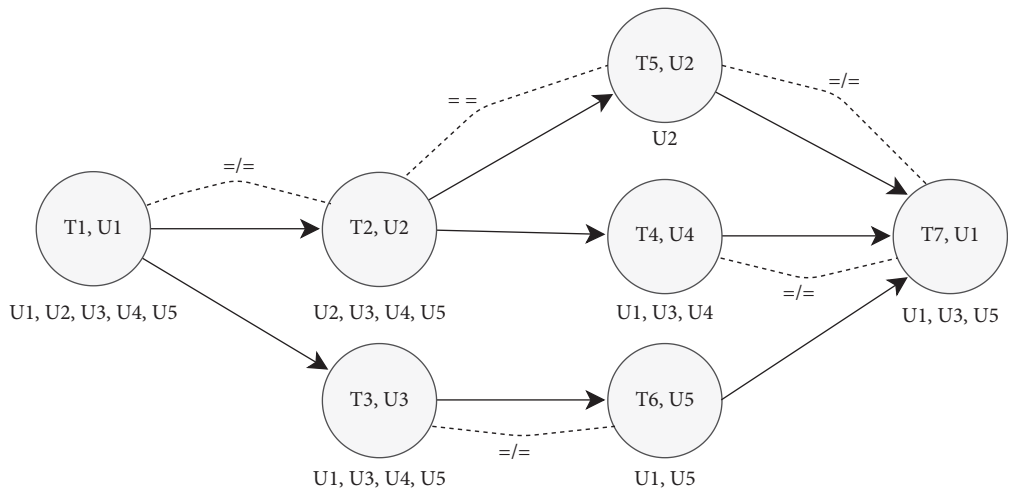

Figure 3: Running example workflow.

Finally, we use equation (2) to get the criticality of the workflow $Cr(W)$. We follow the process described in section (5.4) to get the threshold value θ with n the number of the tasks in the workflow. Thus, $\theta = 0.586$ in our example.

The curves presented in Figures 5 and 6 illustrate the evolution of the task instance criticality throughout the workflow execution. We can easily notice that the task instance criticality increases more and more throughout the execution of the workflow. This is a logical result

Table 2: User-task assignments and task priority.

Task (T)	1	2	3	4	5	6	7
Pr (T)	0, 4	0, 6	0, 4	0, 6	0, 8	0, 6	0, 4
Users	$U1$	$U2$	$U3$	$U4$	$U2$	$U5$	$U1$

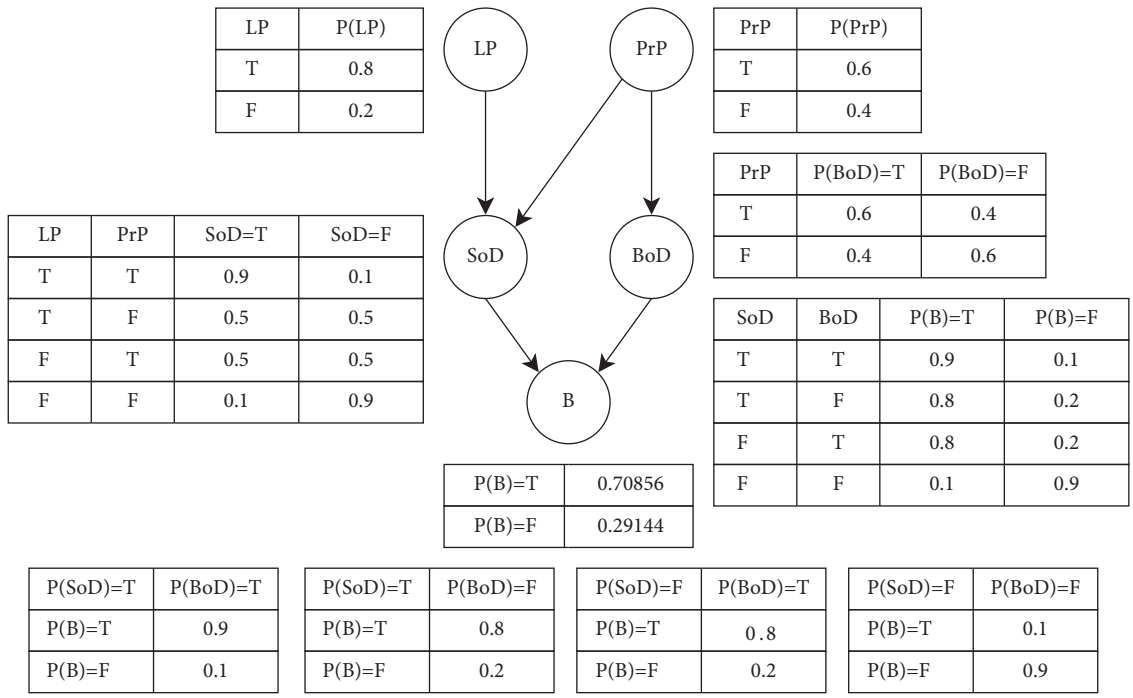

Figure 4: Bayesian network of the running example.

Table 3: Task instance criticality Cr (w (t)) and task instance priority Pr (t).

Tasks (T)	Users	Pr (T)	P (B)	Cr (w (t))	Pr (t)
1	**U1**	0, 4	0, 1	0, 06321206	0, 02528482
2	**U2**	0, 6	0, 8	0, 50569645	0, 30341787
3	**U3**	0, 4	0, 70856	0, 44789534	0, 17915814
4	**U4**	0, 6	0, 70856	0, 44789534	0, 26873721
5	**U2**	0, 8	0, 86969	0, 54974893	0, 43979914
6	**U5**	0, 6	0, 8	0, 50569645	0, 30341787
7	**U1**	0, 4	0, 8	0, 50569645	0, 20227858

because of the security constraints that must be respected for every user-task assignment, as well as the workflow specifications and other constraints (task time sensitivity, user's availability, etc.). Moreover, the strict application of the security policy decreases the number of available and authorized users for each task in the workflow.

All these observations reflect the fact that the risk of occurrence of the workflow satisfiability problem increases during the run-time of the workflow, especially for the last tasks in the workflow. Therefore, we proposed to use $Cr(\omega)$, we defined earlier (cf. Section 5.3), in addition to the threshold value theta (cf. Section 5.4) as a horizontal barrier for the workflow criticality value $Cr(\omega)$, to trigger a mitigation solution like the delegation process algorithm presented in this paper (cf. Section 7). In other words, the threshold value theta represents the limit of criticality for a given workflow, and it is used to control $Cr(\omega)$, the workflow criticality value, if it reaches or crosses the theta barrier.

Two different situations are presented in Figures 5 and 6, where the workflow criticality value $Cr(W)$ corresponds to the criticality value of the fifth task instance $Cr(\omega(t_5))$. In Figure 5, the average rate of users unavailability is $\lambda^* = 1$. Thus, the probability $P(A)$ that a user participating in the workflow execution or a potential delegatee becomes unavailable is 0, 632. Even though the probability value $P(A)$ is significant, we cannot make decisions about the workflow using only $P(A)$. Since it does not reflect the actual criticality of the workflow, therefore, to make decisions, we propose to

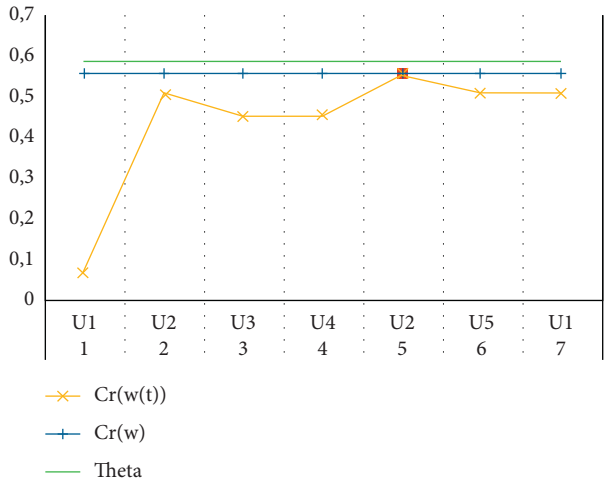

Figure 5: Workflow criticality for $\lambda^* = 1$.

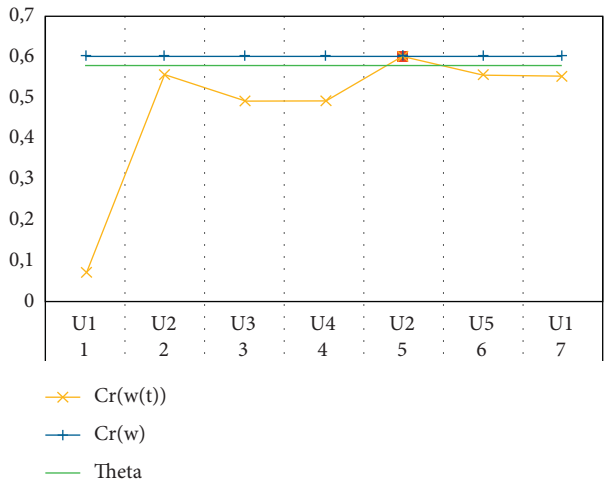

Figure 6: Workflow criticality for $\lambda^* = 1.2$.

use $Cr(W)$ the workflow criticality (cf. Section 5.3) and the threshold theta we derive from a prediction interval as described in Section 5.4.

Accordingly, the workflow criticality in Figure 5 has a value of $Cr(W) = 0,549$, less than the threshold theta, which is equal to $\theta = 0,586$. Thus, the workflow manager can decide to start or continue the execution of the workflow with a risk of 5% that the workflow satisfiability problem occurs since the theta barrier represents a confidence level of 95%. In the second case, the curve in Figure 5 has an average rate of user unavailability of $\lambda^* = 1,2$, and the workflow criticality has a value of $Cr(W) = 0,607$, which is higher than the horizontal barrier $\theta = 0,578$. This situation illustrates a case where a small increase in the average rate of users unavailability may be harmful to the workflow execution, especially during the workflow run-time. As a recommendation to bypass probable WSP occurrence in such a situation is to start the delegation process described in section 7 or any other mitigation solutions because it is highly risky to start or continue the execution of a workflow with such critical values.

7. Delegation Process Algorithm

In our approach, we consider a simple form of workflow definition, which represents a workflow as a set of ordered atomic tasks, and each task is assigned to an authorized user [10]. Moreover, in addition to role engineering, the proposed approach is based on some complementary steps.

7.1. Role-Task Association. The first complementary step is role-task association, which is denoted as R(T) and represents a set of suitable roles to perform a task T. In fact, the first element of R(T), which is denoted as R-type(T), is the most suitable role to perform a task T from security and business efficiency viewpoints.

The remnant roles in R(T) are secondary options to perform T if the delegation process (cf. Section 7.3) cannot find an authorized user to be assigned to the R-type(T). Moreover, roles in R(T) are ordered on the basis of their suitability to the workflow specifications for the task needed skills, but they may not have all needed privileges to perform task T. In such cases, where no authorized user is available for the most suitable role, a break-glass procedure is launched in the form of privilege escalation for the next suitable role in R(T) (defining privilege escalation is beyond the scope of this paper). The privilege escalation is applied under some constraints to keep the system secure and only if a user who can perform the critical task T is available, in order to override the WSP. The role-task association is supported in most RBAC-based WFMS [10].

7.2. Potential Role Delegatees. The second complementary step is the potential role delegates, which is denoted as Deleg(R, T). It represents the set of users who can be assigned a given role R and get all needed permissions to perform the task T. The potential role delegatees represent an ordered set of users based on their level of qualification (competence) to perform task T (measuring qualification is beyond the scope of this paper). In emergent situation, where no qualified user is available, lack of qualification is the price to pay in order to have resilient and flexible workflows [10].

7.3. Delegation Process. Before describing the delegation process, we assume that the role engineering is already done at administration time with respect to the access control constraints. Thus, we assume that the WFMS is able to identify the conflicting entities (R, T, U, W) at both administration and run-time. Moreover, we assume that the WFMS logs workflow changes and the execution of each workflow instance, and also the authorization constraints applied within the workflow and the data about workflow users.

As described in [10], we define the local and global contexts. The local context denoted $ctl(\omega)$ represents the set of users who have already participated in ω by executing a task instance t. While the global context denoted CT concerns the whole organization and all its workflows, CT represents the set of $ctl(\omega)$ of all executed or under

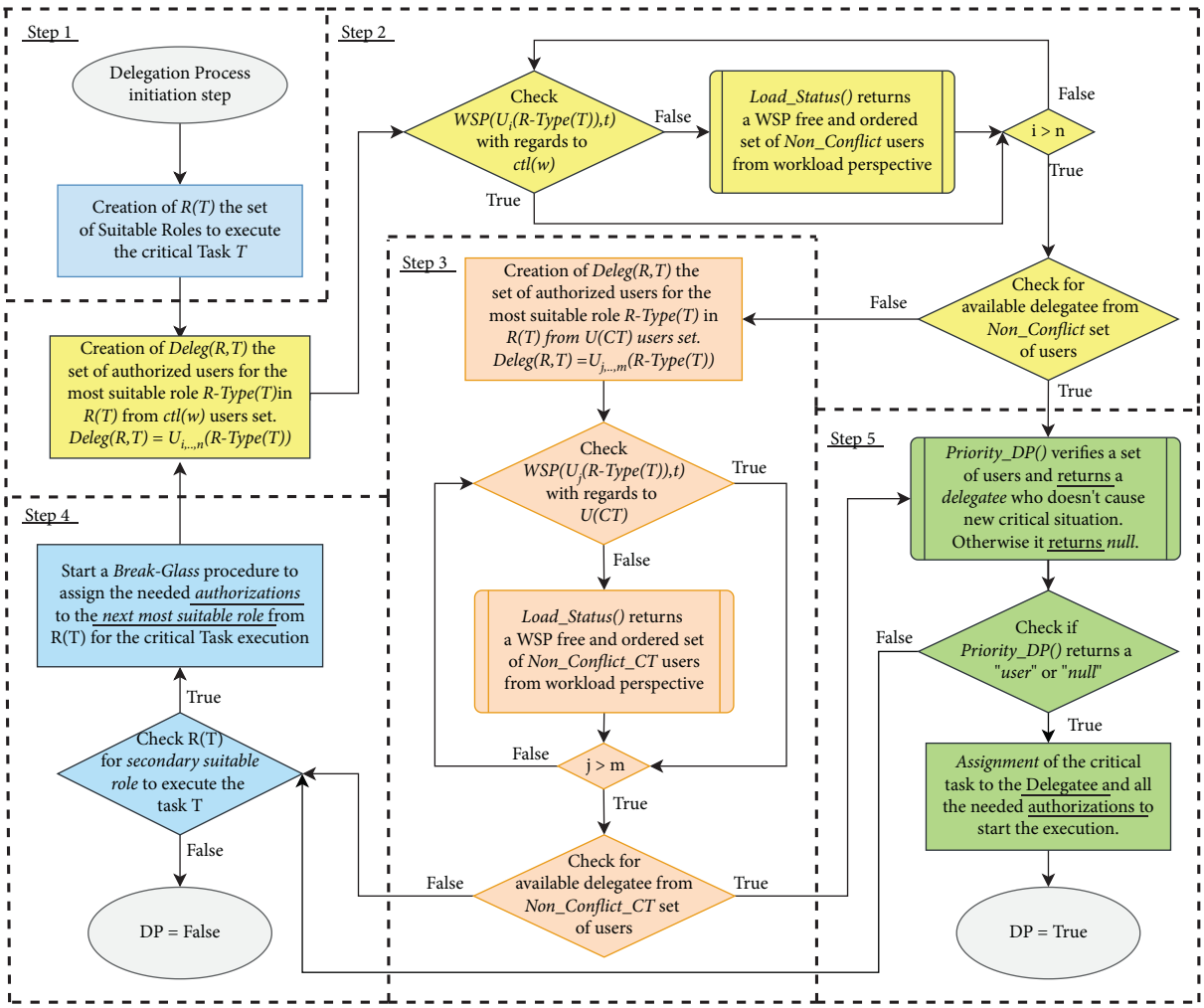

Figure 7: Delegation process algorithm.

execution workflows. Moreover, we denote $U(CT)$ as the set of users who are executing task instances in workflows, which are currently under execution.

The delegation process algorithm is presented in Figure 7.

The flowchart in Figure 7 presents the delegation process algorithm. The algorithm goes through different steps to find a delegatee for the critical task, therefore bypassing the WSP situation.

Step 1: The algorithm starts with initiations where the Boolean variable DP gets false as value, afterward the creation of $R(T)$, the set of suitable roles to execute the critical task T from a security point of view. Creating $R(T)$ is a necessary step for the delegation process that we call *Role-Task association* (cf. Section 7.1). The most suitable role for the critical task T denoted *R-Type*(T) is the first role in $R(T)$.

Step 2: The second step starts with the *Potential-Role delegatees* (cf. Section 7.2) in which the algorithm creates $Dleg(R,T) = U_{i,\ldots,n}(R\text{-}Type(T))$, the set of authorized users for the most suitable role from $ctl(\omega)$ user set. Subsequently, the algorithm verifies the users in $U_{i,\ldots,n}(R\text{-}Type(T))$, if they do not create any conflict, and the workflow will be WSP-free after the user-task assignment for the critical task T with regard to the local context $ctl(\omega)$. After that, the function *Load_Status* described in section 5.2 returns from the WSP-free users set an ordered set of users that are available from a workload perspective. At the end of the second step, the algorithm creates an ordered set of users denoted *Non_Conflict*. The next step depends on the *Non_Conflict* set of users; in case it is empty, the algorithm proceeds to step 3 else it starts step 5.

Step 3: The third step repeats the steps in step 2, but the main difference is that the third step takes into consideration the global context CT (cf. Section 7.3). Therefore, it starts with the *Potential-Role delegatees* to create $Dleg(R,T) = U_{j,\ldots,m}(R\text{-}Type(T))$, the set of authorized users for the most suitable role from $U(CT)$ user set. Thereafter, the algorithm verifies the users in $U_{j,\ldots,m}(R\text{-}Type(T))$, if they do not create any conflict, and the workflow will be WSP-free after the user-task assignment for the critical Task T with regard to the global context $U(CT)$. After that, the function *Load_Status* returns from the WSP-free users set an

ordered set of users that are available from a workload perspective. At the end of the third step, the algorithm creates an ordered set of users denoted as *Non_Conflict_CT*. The next step depends on the *Non_Conflict_CT* set of users; in case it is empty, the algorithm proceeds to step 4 else it starts step 5.

Step 4: The fourth step checks if there is any other roles in the set of suitable roles $R(T)$ to execute the critical task T. In case there is no secondary role in $R(T)$, the algorithm returns false, which means that the WSP situation is not resolved. Otherwise, in case there is a secondary role in $R(T)$, a break-glace procedure is launched to assign the needed authorizations to the secondary role from $R(T)$ (cf. Section 7.1). After that, the algorithm goes to step 2.

Step 5: The fifth and final step in the algorithm begins with the execution of the function *Priority_DP*. This function takes as input *Non_Conflict* or *Non_-Conflict_CT* set of users and returns a delegatee who does not cause a new critical situation, or else the function returns null (more details in Section 5.5). Subsequently, if *Priority_DP* returns null, the algorithm goes to step 4. Otherwise, if it returns a user (the delegatee), the algorithm starts the assignment of the critical task to the delegatee and all the needed authorizations to start the execution. In the end, the algorithm returns True as value for DP, which means that the WSP situation is bypassed successfully.

The presented algorithm has a complexity of order $O(k * n * m)$, where k is the number of authorized users for the role *R-Type(T)* and which does not create WSP when assigned to T with regard to ctl (w). While n is the number of available users for the execution of the critical task, m is the number of tasks in the workflow. In fact, the proposed algorithm considers all the available users in the organization and selects the most suitable and authorized users to execute the critical task. Hence, we refine the delegatee selection process through the algorithm steps using Bayesian network and a thresholding method, which allows us to consider security requirement on tasks. Thus, we select only users, whom if assigned to the critical task makes the workflow WSP free.

The first step has a complexity of $O(q)$, and q is the number of roles. The second and third steps have a complexity of the order $O(k * n * m)$. While the fourth step has a complexity of $O(q)$, the fifth and final steps have a complexity of the order $O(n * m)$. In general, the values of k, m, and q in a workflow are small in comparison to n the number of users, which is in most cases much greater, thus generally $n > m > q > k$. As a result, the total complexity of the proposed algorithm will be of the order $O(2q + 2 * k * n * m + n * m)$, which is approximately equal to $O(k * n * m)$.

8. Conclusion and Future Work

In this paper, we have presented a new approach, which is based on workflow criticality as a metric to prevent the WSP. Thus, we reused the result in [10], which uses the delegation mechanism as a mitigation solution to the WSP; also we redesigned the main concepts in the delegation process with new fine-grained ones. The workflow criticality metric we introduced aims to predict the risk of failure of a workflow before its implementation and to track probable failures during the execution. This approach helps workflow managers to design flexible and resilient workflows with low risks that WSP occurs during the execution, especially if rigid security constraints are included. We introduced an algorithm to bypass the WSP and/or search for a potential delegatee with lower security risks, which replays to the needs of the critical task. The DP algorithm combines all the presented concepts (workflow criticality, priority concepts, users' availability, etc.).

To our knowledge, the solution we proposed is new, given the very few works in the literature [22, 23], which proposes to develop new metrics as solutions to enhance workflow flexibility and resiliency in order to solve the WSP without neglecting security constraints. As a future work, we aim to enhance the presented approach by improving the proposed algorithm and introducing new metrics, in order to give managers good visibility on the designed workflows during the administration and execution times. Finally, we aim to integrate our proposed approach to bypass the WSP in a WFMS in order to test it on some real case scenarios and evaluate the benefits in terms of resiliency and security.

Data Availability

No data were used to support this study.

Conflicts of Interest

The author (s) declare (s) that there are no conflicts of interest regarding the publication of this paper.

References

[1] D. Hollingsworth, "Workflow management coalition the workflow reference model," *The Workflow Management Coalition Specification*, vol. TC00-1003, pp. 1–55, 1995.

[2] B. Mitra, S. Sural, J. Vaidya, and V. Atluri, "Migrating from RBAC to temporal RBAC," *IET Information Security*, vol. 11, no. 5, pp. 294–300, 2017.

[3] Q. M. Rajpoot, C. D. Jensen, and R. Krishnan, "Attributes enhanced role-based access control model," *Trust, Privacy and Security in Digital Business*, in *Proceedings of 12th International Conference, TrustBus*, pp. 3–17, Valencia, Spain, September 2015.

[4] W. W. Smari, P. Clemente, and J.-F. Lalande, "An extended attribute based access control model with trust and privacy: application to a collaborative crisis management system," *Future Generation Computer Systems*, vol. 31, pp. 147–168, 2014.

[5] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[6] D. Cohen, J. Crampton, A. Gagarin, G. Gutin, and M. Jones, "Iterative plan construction for the workflow satisfiability

problem," *Journal of Artificial Intelligence Research*, vol. 51, no. 1, pp. 555–577, 2014.

[7] Q. Wang and N. Li, "Satisfiability and resiliency in workflow authorization systems," *ACM Transactions on Information and System Security*, vol. 13, no. 4, pp. 40:1–40:35, 2010.

[8] J. Crampton, G. Gutin, and A. Yeo, "On the parameterized complexity and kernelization of the workflow satisfiability problem," *ACM Transactions on Information and System Security*, vol. 16, no. 1, pp. 4:1–4:31, 2013.

[9] M. Boughrous and H. El Bakkali, "A comparative study on access control models and security requirements in workflow systems," *Advances in Intelligent Systems and Computing*, Springer Cham, vol. 735, Basel, Switzerland, 2018.

[10] H. El Bakkali, "Enhancing workflow systems resiliency by using delegation and priority concepts," *Journal of Digital Information Management (JDIM)*, vol. 11, no. 4, pp. 267–276, 2013.

[11] J. D. Ultra and S. Pancho-Festin, "A simple model of separation of duty for access control models," *Computers & Security*, vol. 68, pp. 69–80, 2017.

[12] J. C. Mace, C. Morisset, and A. V. Moorsel, "Modelling user availability in workflow resiliency analysis," in *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security (HotSoS '15)*, pp. 1–7, ACM, Pittsburgh, Pennsylvania, 2015.

[13] G. G. M. Wahlström, "Tight lower bounds for the workflow satisfiability problem based on the strong exponential time hypothesis," *Information Processing Letters*, vol. 116, no. 3, pp. 223–226, 2016.

[14] D. Cohen, J. Crampton, A. Gagarin, G. Gutin, and M. Jones, "Algorithms for the workflow satisfiability problem engineered for counting constraints," *Journal of Combinatorial Optimization*, vol. 32, no. 1, pp. 3–24, 2016.

[15] P. Yang, X. Xie, I. Ray, and S. Lu, "Satisfiability analysis of workflows with control-flow patterns and authorization constraints," *IEEE Transactions on Services Computing*, vol. 7, no. 2, pp. 237–251, 2014.

[16] N. Li and Q. Wang, "Mahesh tripunitara, "resiliency policies in access control"," *ACM Transactions on Information and System Security*, vol. 12, no. 4, pp. 20:1–20:34, 2009.

[17] Q. Wang and N. Li, "Satisfiability and resiliency in workflow systems," *Computer Security - ESORICS 2007*, in *Proceedings of Computer Security - ESORICS 2007: 12th European Symposium On Research In Computer Security*, pp. 90–105, Dresden, Germany, September 2007.

[18] A. Ali, U. Habiba, and M. A. Shibli, "Taxonomy of delegation model," in *Proceedings of the 12th International Conference on Information Technology - New Generations*, pp. 218–223, Las Vegas, NV, USA, 2015.

[19] K. Gaaloul and F. Charoy, "Task delegation based access control models for workflow systems," in *Proceedings of the The 9th IFIP Conference on e-Business, e-Services, and e-Society*, September 2009.

[20] D. R. Dos Santos and S. Ranise, "A Survey on Workflow Satisfiability, Resiliency, and Related Problems," CoRR, 2017.

[21] G. Gutin, S. Kratsch, and M. Wahlström, "Polynomial kernels and user reductions for the workflow satisfiability problem," *Algorithmica*, vol. 75, no. 2, pp. 383–402, 2016.

[22] J. C. Mace, C. Morisset, and A. van Moorsel, "Quantitative workflow resiliency," *Computer Security - ESORICS 2014*, in *Proceedings of 19th European Symposium on Research in Computer Security*, pp. 344–361, Wroclaw, Poland, September 2014.

[23] J. C. Mace, C. Morisset, and A. van Moorsel, "Resiliency variance in workflows with choice," *Lecture Notes in Computer Science*, in *Proceedings of Software Engineering for Resilient Systems: 7th International Workshop, SERENE 2015*, pp. 128–143, Paris, France, September 2015.

[24] D. Basin, S. J. Burri, and G. Karjoth, "Optimal workflow-aware authorizations," in *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, pp. 93–102, Newark, New Jersey, USA, June 2012.

[25] J. Crampton and C. Morisset, "An auto-delegation mechanism for access control systems," *Security and Trust Management*, in *Proceedings of Security and Trust Management: 7th International Workshop, STM 2011*, pp. 1–16, Copenhagen, Denmark, June 2011.

[26] M. Lowalekar, R. K. Tiwari, and K. Karlapalem, "Security policy satisfiability and failure resilience in workflows," *The Future of Identity in the Information Society*, Springer, Berlin Heidelberg, Germany, pp. 197–210, 2009.

[27] C. M. Bishop, "Model-based machine learning," *Philosophical Transactions of the Royal Society A: Mathematical, Physical & Engineering Sciences*, vol. 371, Article ID 20120222, 2013.

[28] V. j. Finn and D. N. Thomas, *Bayesian Networks and Decision Graphs: Causal and Bayesian Networks. Information Science And Statistics*, Springer, Basel, Switzerland, 2007.