Automatically Finding Execution Scenarios to Deploy Security-Sensitive Workflows [1]

Daniel Ricardo dos Santos [a,b,c,*], Silvio Ranise [a], Luca Compagna [b] and Serena Elisa Ponta [b]
[a] *Fondazione Bruno Kessler*
[b] *SAP Labs France*
[c] *University of Trento*

Abstract. We introduce a new class of analysis problems, called Scenario Finding Problems (SFPs), for security-sensitive business processes that—besides execution constraints on tasks—define access control policies (constraining which users can execute which tasks) and authorization constraints (such as Separation of Duty). The solutions to SFPs are concrete execution scenarios that assist customers in the reuse and deployment of security-sensitive workflows. We study the relationship of SFPs to well-known properties of security-sensitive processes such as Workflow Satisfiability and Resiliency together with their complexity. Finally, we present a symbolic approach to solving SFPs and describe our experience with a prototype implementation on real-world business process models taken from an on-line library.

Keywords: Workflow Satisfiability, Run-time monitor, Business Process

1. Introduction

Organizations rely on Business Process Management (BPM) [38] to achieve certain business objectives by orchestrating workflows, which are collections of sequences of tasks executed by human or software agents. An increasingly important class of workflows is that of *security-sensitive workflows* [2], in which task execution constraints are complemented with an authorization policy (defining which users can execute which tasks) and a set of authorization constraints (further restricting which users can execute some sub-sets of the tasks). One of the most important problems for security-sensitive workflows is the Workflow Satisfiability Problem (WSP) [10], which consists of checking if there exists an assignment of users to tasks such that at least one task execution sequence in a workflow successfully terminates while satisfying all authorization constraints and the authorization policy.

Several papers (see, e.g., [10,15,16,37]) have provided solutions to the WSP, which are becoming less and less satisfactory because of the recent trend in BPM of collecting and reusing large numbers of business process models [36,18,39]. For instance, SAP HANA Workflow[1] is a BPMN-based solution

[1]This work has been partly supported by the EU under grant 317387 SECENTIS (FP7-PEOPLE-2012-ITN).
[*]Corresponding author. E-mail: dossantos@fbk.eu.
[1]https://help.sap.com/hana-opint

that allows for the creation of template business process models that can be deployed and operated in different contexts. At deployment time, what is crucial for customers reusing a template from the library is to understand whether it can be successfully instantiated with the authorization policy adopted by their organization. This means that customers want to solve the *problem of finding execution scenarios* that show the termination of the instantiated business process model by giving evidence that some of the employees can successfully execute the various tasks in the workflow. Variants of this basic problem may be of interest to assist customers in deploying security-sensitive workflows under specific operational conditions. A first refinement is to focus on those scenarios that can be executed by a given "small" set of users, called *minimal user base* in the literature [16]. This would enable organizations to assess the likelihood of emergencies or extraordinary situations due to, e.g., employee absences. A second variant of the problem of finding execution scenarios is to discover those that are *resilient* to the absence of users, i.e. whether the termination of the workflow is guaranteed even if a certain number of users (regardless of which concrete users) is unavailable to execute tasks for a given execution [37]. A refinement of all previous problems consists of considering only those scenarios satisfying some additional conditions on the execution of a workflow, such as only a given user can execute a task, the Boolean value of a conditional is true (or false), the order of execution of a set of parallel tasks is fixed, or arbitrary combinations of such constraints. In the following, we call *Scenario Finding Problems (SFPs)* this kind of problems.

Techniques for solving the WSP can also be used to solve SFPs; unfortunately, this has a very high computational cost because of two reasons. First, WSP is NP-hard already in the presence of one SoD constraint [37]. Second, techniques for the WSP are not able to exploit the fact that execution and authorization constraints are fixed and only the authorization policy changes at deployment time. To overcome these limitations, the paper makes the following contributions.

- We give precise statements of four SFPs together with a discussion of their relationships with the WSP (Section 3).
- We describe methods to automatically solve the four SFPs by adapting the technique for the synthesis of run-time monitors for the WSP developed in [6] (Section 4).
- We validate our solutions on real-world examples from a library of reusable business process models (Section 5).

The idea underlying our approach can be summarized as follows. Model checking is a technique for determining whether a (formal) model of a system satisfies a given property. If the property is false in the model, model checkers typically produce a counterexample scenario, which is then used by developers to fix bugs in the design of the system. To solve a SFP, we use the capability of model checkers to return counterexamples as follows. We formally represent security-sensitive workflows as symbolic transition systems following [6]. The symbolic model checker is then asked to find a counterexample to the property that the system is not terminating. Indeed, the returned counterexample (if any) is precisely an execution scenario solving the SFP. Since we are interested in finding all execution scenarios, we modify the model checker in order to compute all counterexamples, not just one. We represent a set of counterexample scenarios by using a scenario graph, which allows us to compactly encode all possible interleavings of tasks in a workflow.

The crux of our approach is that the model of the security-sensitive workflow in input to the symbolic model checker only contains the constraints on the control flow and the authorization constraints while it abstracts away from the authorization policy. In this way, the scenario graphs computed by the model checker can then be refined with respect to an authorization policy associated to a particular deployment context. The refinement is performed by a depth-first search of the scenario graph to prune those scenarios

that do not satisfy the authorization policy used in the deployment context under consideration. When considering resiliency, this is combined with a (heuristic) method to generate subsets of users not containing k users (by adapting a result from [24]) in order to find scenarios guaranteeing the termination of a workflow despite the absence of k users.

This paper is an extended version of [19], with the main additions being the definition of two new SFPs (involving resiliency and constrained scenarios) in Sections 3.3 and 3.4; an expanded description of the solution to the WSP in Section 4.2; new experiments in Section 5; an extensive comparison with related work in Section 6.1; and a description of future work in Section 6.2.

2. Preliminaries

Let T be a finite set of tasks and U a finite set of users. An *execution scenario* (or, simply, a *scenario*) is a finite sequence of pairs of the form (t, u), written as $t(u)$, where $t \in T$ and $u \in U$. The intuitive meaning of a scenario $\eta = t_1(u_1), ..., t_n(u_n)$ is that task t_i is executed before task t_j for $1 \leq i < j \leq n$ and that task t_k is executed by user u_k for $k = 1, ..., n$. A *workflow* $W(T, U)$ is a set of scenarios. Among the scenarios in a workflow, we are interested in those that describe successfully terminating executions in which users execute tasks satisfying the authorization constraints and the authorization policy. Since the notion of successful termination depends on the definition of the workflow (e.g., in case of a conditional choice, we will have two acceptable execution sequences according to the Boolean value of the condition), in the following we focus only on the authorization policy and the authorization constraints while assuming that all the scenarios in the workflow characterize successfully terminating behaviors.

Given a workflow $W(T, U)$, an *authorization relation* TA is a sub-set of $U \times T$ where $(u, t) \in TA$ means that u is authorized to execute task t. We say that a scenario η of a workflow $W(T, U)$ is *authorized* according to TA iff (u, t) is in TA for each $t(u)$ in η. An *authorization constraint* over a workflow $W(T, U)$ is a tuple (t_1, t_2, ρ) where $t_1, t_2 \in T$ and ρ is a sub-set of $U \times U$. (It is possible to generalize authorization constraints to the form (T_1, T_2, ρ) where T_1, T_2 are sets of tasks as done in, e.g., [15]. We do not do this here for the sake of simplicity.) For instance, a SoD constraint between tasks t and t' can be formalized as $(t, t', \neq)$ with $\neq$ being the relation $\{(u, u') | u, u' \in U \text{ and } u \neq u'\}$. A scenario η of $W(T, U)$ *satisfies* the authorization constraint (t_1, t_2, ρ) over $W(T, U)$ iff there exist $t_1(u_1)$ and $t_2(u_2)$ in η such that $(u_1, u_2) \in \rho$. Let C be a (finite) set of authorization constraints, a scenario η satisfies C iff η satisfies c, for each c in C. A scenario η of a workflow $W(T, U)$ is *eligible according to a set C of authorization constraints* iff η satisfies C. A workflow $W(T, U)$ is *security-sensitive* according to an authorization relation TA and a (finite) set C of authorization constraints iff every scenario η in $W(T, U)$ is both authorized and eligible[2].

There are various ways to specify security-sensitive workflows; we illustrate one of the most used methods by means of an example.

Example 2.1 (Trip Request Workflow (TRW))**.** *A typical example of a security-sensitive workflow has the goal of requesting trips for employees in an organization. It is composed of five tasks: Trip request ($t1$), Car rental ($t2$), Hotel booking ($t3$), Flight reservation ($t4$), and Trip validation ($t5$). The execution of the tasks is constrained as follows: $t1$ must be executed first, then $t2$, $t3$ and $t4$ can be executed in any order, and when all have been performed, $t5$ can be executed. Overall, there are six possible task*

[2]The notions of *eligible* and *authorized* scenarios were adapted from [15].

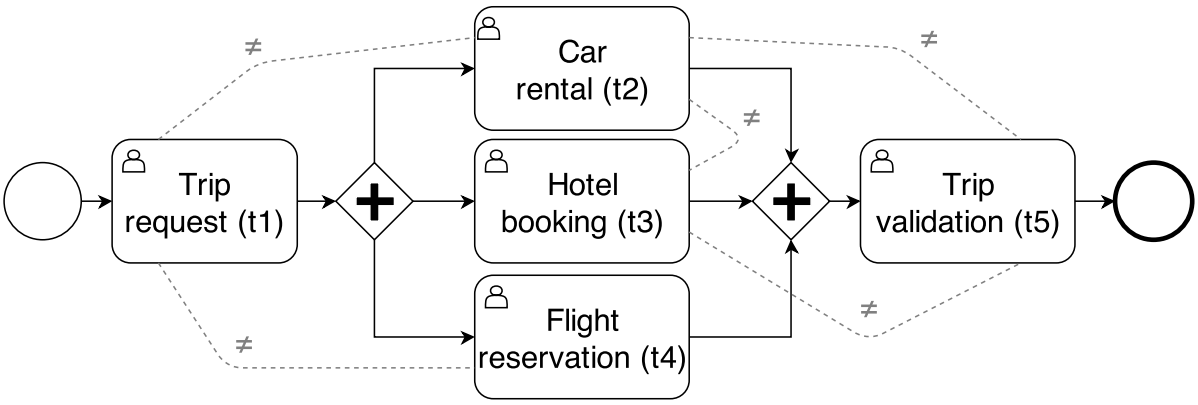


Fig. 1. TRW in (extended) BPM notation

execution sequences in which the first is always task $t1$, the last is always task $t5$, and—in between—there is any one of the six permutations of $t2$, $t3$ and $t4$.

It is also required that each task is executed under the responsibility of a user who has the right to execute it according to some authorization policy. To prevent frauds, five authorization constraints—called Separation of Duty (SoD) in the literature; see, e.g., [10]—must also be enforced: each one of the following pairs of tasks must be executed by distinct users in any sequence of task executions of the workflow: $(t1, t2)$, $(t1, t4)$, $(t2, t3)$, $(t2, t5)$, and $(t3, t5)$.

This workflow can be modeled in a graphical notation such as BPMN [30] as shown in Figure 1: the circle on the left represents the start event (triggering the execution of the workflow), whereas that on the right the end event (terminating the execution of the workflow), tasks are depicted by labeled boxes, the constraints on the execution of tasks are shown as solid arrows (for sequence flows) and diamonds labeled by $+$ (for parallel flows), the fact that a task must be executed under the responsibility of a user is indicated by the man icon inside a box, and the SoD constraints as dashed lines labeled by $\neq$. □

We use the TRW introduced in the example above to illustrate the main notions in the paper.

Definition 2.1 (Workflow Satisfiability Problem (WSP))**.** *Given a workflow $W(T, U)$, an authorization relation TA, and a set C of authorization constraints, return (if possible) a scenario η which is authorized according to TA and eligible according to C.*

We use Example 2.2 to show an instance of this problem and the solution.

Example 2.2. *A simple situation in which the TRW in Example 2.1 can be deployed is a tiny organization with a set $U = \{a, b, c\}$ of three users and the following authorization policy $TA = \{(a, t1), (b, t1), (a, t2), (b, t2), (c, t2), (a, t3), (b, t3), (c, t3), (a, t4), (a, t5), (b, t5), (c, t5)\}$. The organization would then like to know if there is an execution scenario that allows the process to terminate according to TA. Indeed, this is the case as shown by the following sequence of task-user pairs: $\eta = t1(b), t3(c), t4(a), t2(a), t5(b)$. It is easy to check that the tasks in η are executed so that the ordering constraints on task execution are satisfied, each user u in each pair $t(u)$ of η is authorized to execute t since $(u, t) \in TA$, and each SoD constraint is satisfied (e.g., tasks $t1$ and $t2$ are executed by the distinct users b and a, respectively).* □

3. Scenario Finding Problems

Reuse in Business Process Management has been an important topic of research and industrial application; see, e.g., [22,17]. The reuse of business process models increases the productivity of designers and the quality of the models. A crucial step in reusing process templates is their deployment. Helping customers to assess whether the process instance is suitable to their needs is indeed fundamental. For example, checking the existence of execution scenarios for a deployed security-sensitive workflow—i.e. after adding a particular authorization policy—can reassure customers about the business continuity of the process. Automating the identification of such scenarios is of particular importance when a library of process templates are available and are to be re-used in different organizations adopting different authorization policies. As mentioned above, it is possible to pre-compute—once and for all—the set E of eligible scenarios, i.e. those scenarios satisfying the authorization constraints, associated to a security-sensitive workflow in a library (we will describe how to compute and compactly represent this set in Section 4 below). We are then left with the problem of finding those scenarios in E that are still computable as soon as an authorization policy becomes available (and possibly satisfying some additional properties).

3.1. Basic scenarios

We begin by defining a basic version of our scenario finding problems, whose definition (and solution) will help us in understanding (and solving) more complex problems.

Definition 3.1 (Basic Scenario Finding Problem (B-SFP))**.** *Given the finite set E of eligible scenarios according to a set C of authorization constraints in a workflow $W(T, U)$, return (if possible) a scenario $\eta \in E$ which is authorized according to a given authorization relation TA.*

Example 3.1. *Let us consider the TRW. If $U = \{Alice, Bob, Charlie, Dave, Erin, Frank\}$ is the set of users, then the set E of eligible scenarios contains, among many others, the following elements:*

$$\eta_1 = t_1(Alice), t_2(Bob), t_3(Charlie), t_4(Dave), t_5(Erin)$$

$$\eta_2 = t_1(Bob), t_2(Alice), t_3(Charlie), t_4(Alice), t_5(Bob)$$

$$\eta_3 = t_1(Bob), t_4(Charlie), t_2(Alice), t_3(Dave), t_5(Bob)$$

Now, let $TA = \{(Alice, t_1), (Bob, t_1), (Alice, t_2), (Bob, t_2), (Charlie, t_3), (Alice, t_4), (Dave, t_4), (Bob, t_5), (Erin, t_5)\}$ be the authorization relation, then η_1 and η_2 are solutions to the B-SFP, while η_3 is not because $(Dave, t_3) \notin TA$. □

A scenario η solving the B-SFP is also a solution of the WSP and vice versa. So, in principle, to solve the B-SFP for a workflow $W(T, U)$, a set C of authorization constraints, an authorization policy TA, and $\eta_e = t(u), t'(u'), ...$ an eligible scenario in E, we can reuse an algorithm $\mathcal{A}$ returning answers to the WSP as follows. Initially, we consider the task-user pair $t(u)$ in η_e and create a new authorization relation $TA_1 = TA|_{(u,t)}$ derived from TA by deleting all pairs $(x, t) \in TA$ with $x \neq u$. We invoke $\mathcal{A}$ on the WSP for $W(T, U)$, C, and TA_1: if $\mathcal{A}$ returns a scenario, this must have the form $t(u), \eta$ where η is some sequence of task-user pairs (notice that $t(u), \eta$ and η_e are guaranteed to have only $t(u)$ as a common prefix). Afterwards, we move to the task-user pair $t'(u')$ in η_e and run $\mathcal{A}$ on the WSP for $W(T, U)$, C, and $TA_2 = TA_1|_{(u',t')}$. If $\mathcal{A}$ returns a scenario, this must have the form $t(u), t'(u'), \eta'$ where η' is some sequence of task-user pairs (notice that $t(u), t'(u'), \eta'$ and η_e are guaranteed to have only $t(u), t'(u')$ as a common prefix). By repeating this process for each η_e in E, until all tasks in η_e are executed, we can

check if it is also authorized according to TA (besides being eligible as η_e is in E). Overall, there are at most $O(\ell_{max} \cdot |E|)$ invocations to $\mathcal{A}$, where ℓ_{max} is the longest (in terms of number of task-user pairs occurring in it) scenario of E. Indeed, this is very expensive from a computational point of view since the WSP is NP-hard already in presence of one SoD constraint [37] and, most importantly, we do not exploit the fact that the scenarios in E are eligible.

A better approach to solve the B-SFP is to consider each eligible scenario η_e in E and check if all task-user pairs in η_e are authorized according to TA. This means that there are at most $O(\ell_{max} \cdot |E|)$ invocations to the algorithm for checking membership of a user-task pair to TA. The complexity of such an algorithm depends on how TA has been specified. Policy languages are designed to make such a check very efficient (e.g., linear or polynomial); this is in sharp contrast to the heavy computational cost of running $\mathcal{A}$. Below, we assume authorization policies to be specified in Datalog so that checking membership to TA is equivalent to answering a Datalog query, which is well-known to have polynomial-time (data) complexity [7]. Even though checking for membership to TA is efficiently performed, the overall computational complexity may be problematic since such a check must be repeated $O(\ell_{max} \cdot |E|)$ and $|E|$ may be very large. For instance, as we will show below, already for the simple TRW with $|U| = 6$ (as in Example 3.1), the cardinality of E is $19,080$. Intuitively, the larger the set U of users, the higher the cardinality of E. It is thus important to design a suitable data structure to represent the available set E of eligible scenarios which permits to design an efficient strategy to search through all scenarios and identify one that is authorized. We will see this in Section 4.2.

3.2. Minimal user-base scenarios

A refinement of B-SFP is to search for (eligible and) authorized scenarios in which a "minimal" set of users occurs. Formally, let η be a scenario in a workflow $W(T, U)$, the set of users occurring in η is $usr(\eta) = \{u | t(u) \in \eta\}$. Following [16], we define a *minimal user base* of a workflow $W(T, U)$ to be a sub-set U' of the set U of users such that there exists a scenario η in $W(T, U)$ in which $usr(\eta) = U'$ and there is no scenario η' in $W(T, U)$ in which $usr(\eta')$ is a strict sub-set of U'.

Definition 3.2 (Minimal user-base scenario finding problem (MUB-SFP))**.** *Given the set E of eligible scenarios according to a set C of authorization constraints in a workflow $W(T, U)$, return (if possible) a scenario $\eta \in E$ which is authorized according to a given relation TA and such that the set $usr(\eta)$ of users occurring in η is a minimal user base.*

Example 3.2. *Let us consider again the TRW together with the set U of users, the set E of eligible scenarios, and the authorization relation TA of Example 3.1. A solution to the MUB-SFP is $\eta_M = t_1(Bob), t_2(Alice), t_3(Charlie), t_4(Alice), t_5(Bob)$ and a minimal user base is $usr(\eta_M) = \{Alice, Bob, Charlie\}$.* □

An approach derived from that of solving the B-SFP can also solve the MUB-SFP. We consider each eligible scenario η_e in E and check if all task-user pairs in η_e are authorized according to TA. We also maintain a variable η_M storing an eligible scenario in E such that η_M is authorized (according to TA) and $usr(\eta_M)$ is a candidate minimal user base. Initially, η_M is set to the empty sequence ϵ. If the eligible scenario η_e under consideration is authorized and $\eta_M \neq \epsilon$, then we compare the cardinalities of $usr(\eta_e)$ and $usr(\eta_M)$: if $|usr(\eta_e)| < |usr(\eta_M)|$, then $\eta_M \leftarrow \eta_e$; otherwise η_M is left unchanged. When $\eta_M = \epsilon$, we do not perform the comparison between the cardinalities of $usr(\eta_e)$ and $usr(\eta_M)$ and simply set η_M to η_e. Indeed, when all eligible scenarios in E have been considered, $usr(\eta_M)$ stores a minimal user base. This process requires that there are $O(\ell_{max} \cdot |E|)$ invocations to the algorithm for checking membership

of a user-task pair to TA. Although the complexity bounds of solving the B-SFP and the MUB-SFP are identical, the bound for the latter is tighter than the former. This is so because we always need to consider all eligible scenarios in E for the MUB-SFP whereas we can stop as soon as we find an authorized scenario for the B-SFP. This is confirmed by our experimental evaluation in Section 5 (compare the timings for solving SFPs with those for MUB-SFPs in Table 2).

3.3. Resilient scenarios

Resiliency in workflow systems concerns the question of whether a workflow is still satisfiable even if a number of users is absent for an instance execution. It is an important property of workflows and authorization relations, since it indicates the likelihood of the workflow terminating even in adverse conditions.

There are several possible definitions of resiliency. The most obvious one is to fix a set U_a of absent users and check if the workflow can terminate. This can be easily reduced to a B-SFP problem by eliminating all those execution scenarios in $W(T, U)$ in which at least one task is executed by an absent user and by removing all pairs containing an absent user in U_a from the authorization policy TA. If the resulting B-SFP has a solution, then it can be considered as a witness of the fact that the workflow $W(T, U)$ is resilient to the absence of the users in U_a. Another, more interesting, notion of resiliency has been proposed in [37] and amounts to checking whether a workflow is resilient to the absence of *all* subsets of users of a fixed size k. This means that no matter which concrete users are absent, as long as no more than a given number is absent (at any given time), the workflow remains satisfiable. We adapt this idea of resiliency to our context by identifying resilient sets of execution scenarios as follows.

Definition 3.3 (Statically k-resilient set of scenarios)**.** *Given a workflow $W(T, U)$, an authorization relation TA, and an integer $k > 0$, a set of scenarios H is statically resilient up to k absent users if and only if for every subset U' of U of size $t = |U| - k$, there is (at least) one scenario $\eta \in H$ that satisfies $W(T, U)$ under $TA|_{U'}$.*

Notice that resiliency is defined *up to* k since a k-resilient set of scenarios is clearly also $(k-1)$-resilient. Notice that it is not possible to limit resiliency to a single scenario because the second notion of resiliency discussed above must hold for all subsets of users of size t and this implies that if any one of the users u in a scenario η is removed, then η can no more be executed until the end. Instead, by considering a set H of execution scenarios, when we consider a certain subset U' of absent users of size t, we can hope to find a scenario in H containing no user in U' that can still be executed until the end.

We are now ready to introduce the problem of finding sets of resilient scenarios.

Definition 3.4 (Statically k-Resilient Scenario Finding Problem (SkR-SFP))**.** *Given the set E of eligible scenarios according to a set C of authorization constraints in a workflow $W(T, U)$ and an integer k, return (if possible) a set H of scenarios η which are authorized according to a given relation TA and statically resilient up to k absent users.*

Example 3.3. *Let us consider again the TRW together with the set U of users, the set E of eligible scenarios, and the authorization relation TA of Example 3.1. In that case, there is no solution to the SkR-SFP, even with $k = 1$, because if Charlie is removed from U, there is no user authorized to perform $t3$ and if Alice or Bob are removed the same user has to perform $t1$ and $t2$, which violates the SoD constraint. Now consider the same workflow and set of users, but with*

$TA' = TA \cup \{(Charlie, t1), (Charlie, t2), (Alice, t3)\}$. *In this case, a solution to the SkR-SFP, with* $k = 1$ *is* $H = \{\eta_1, \eta_2, \eta_3, \eta_4, \eta_5\}$ *where*

$$\eta_1 = t1(Charlie), t2(Bob), t3(Charlie), t4(Dave), t5(Erin)$$
$$\eta_2 = t1(Charlie), t2(Alice), t3(Charlie), t4(Dave), t5(Erin)$$
$$\eta_3 = t1(Alice), t2(Bob), t3(Alice), t4(Dave), t5(Erin)$$
$$\eta_4 = t1(Bob), t2(Alice), t3(Charlie), t4(Alice), t5(Erin)$$
$$\eta_5 = t1(Charlie), t2(Alice), t3(Charlie), t4(Dave), t5(Bob)\,.$$

Notice that η_1 *is satisfiable when* $U' = \{Alice\}$, η_2 *is satisfiable when* $U' = \{Bob\}$, η_3 *is satisfiable when* $U' = \{Charlie\}$, η_4 *is satisfiable when* $U' = \{Dave\}$, *and* η_5 *is satisfiable when* $U' = \{Erin\}$. *All scenarios are also satisfiable when* $U' = \{Frank\}$, *because Frank is not used in any of them. Even with the new authorization relation* TA' *the workflow is not 2-resilient, because if, e.g., Alice and Charlie are removed from the set of users, again there is no user capable of executing* $t3$. □

To solve the SkR-SFP, an obvious strategy is to initialize the set H of k-resilient execution scenarios to the empty set, generate all subsets of users with size $t = n - k$ (for n the total number of users), update the authorization policy TA by removing the k absent users, and then compute a solution of the resulting B-SFP. If such a scenario exists, we add it to the set H. Indeed, enumerating all subsets of size t of a set of n elements is equal to the binomial coefficient $\binom{n}{t}$. Thus, we would need $O(n!)$ calls to a procedure solving the B-SFP in the worst case (or close to n^t when t is small compared with n [24]). It is possible to reduce the number of sets that must be considered by observing (as done in [24]) that some sets of users can be ignored as they are "dominated" by others with respect to the set of tasks that they can execute. We will make this precise in Section 4.3 below.

Another interesting problem is to find the maximal value k_{max} such that there is a set H of scenarios that are authorized according to a relation TA and resilient up to k_{max}. A possible way to attack this problem is to use a procedure solving SkR-SFP to find a set H of resilient scenarios for a value $k^* = 1$. If there is a solution, we increase the value of k^* by one and invoke again the procedure to solve the Sk^*R-SFP. We keep increasing the value of k until no more solutions are found and set the value of k_{max} to $k^* - 1$. Since there are at most $|U|$ users, the search for k_{max} eventually terminates. In case the set U of users is large, the search may go through several iterations, each requiring the invocation of the procedure solving the SkR-SFP. A better solution is to first find an upper bound to k_{max} by preliminarily finding a solution m to the MUB-SFP. Then, an upper bound on k_{max} is given by $k^*_{max} = |U| - m$ as it is not possible to complete the workflow with less than m users. Afterwards, we invoke the procedure to solve the SkR-SFP with $k = k^*_{max}$. If the problem is solvable, then we return such a value as k^*_{max}; otherwise, we decrease k by one and invoke again the procedure solving the SkR-SFP. In the worst case (i.e. when the workflow is 0-resilient), the procedure to solve the SkR-SFP will be invoked m times; this indeed implies termination.

Another problem extension is how to achieve k-resiliency while minimally changing the authorization policy, i.e. adding as few (u, t) pairs as possible to TA. Adding (u, t) pairs can be done with, e.g., delegation or an administrative policy such as ARBAC [32]. A solution to this problem is left as future work.

3.4. Constrained scenarios

The three problems (B-SFP, MUB-SFP, and SkR-SFP) introduced above accept as solutions unconstrained scenarios, i.e. scenarios where any authorized user can execute any task and any allowed control path can be taken (e.g., any interleaving of parallel tasks is possible and any branch of a conditional is equally likely to be executed). Sometimes, policy designers may be interested in finding scenarios satisfying certain properties, e.g., scenarios in which only certain authorized users can execute certain tasks or some control-flows can be executed. For instance, the designer may want to investigate scenarios in which the test of a conditional evaluates to true or in which only certain users perform some tasks. We use constraints to specify the properties that scenarios must additionally satisfy in the constrained versions of the three problems above. The constraints specified by the designer are represented as predicates and are collected in a set Γ.

We are now in the position to generalize the B-SFP problem by defining its constrained version as follows.

Definition 3.5 (Constrained Scenario Finding Problem (C-SFP))**.** *Given the set E of eligible scenarios according to a set C of authorization constraints in a workflow $W(T, U)$ and a set Γ of constraints, return (if possible) a scenario $\eta \in E$ which is authorized according to a given relation TA and such that all the predicates in Γ evaluate to true.*

Example 3.4. *Let us consider again the TRW together with the set U of users, the set E of eligible scenarios, and the authorization relation TA of Example 3.1. A solution to the C-SFP with $\Gamma = \{t2(Bob)\}$ (requiring that Bob executes task $t2$ in any scenario) is the scenario $\eta = t_1(Alice), t_2(Bob), t_3(Charlie), t_4(Dave), t_5(Erin)$.* □

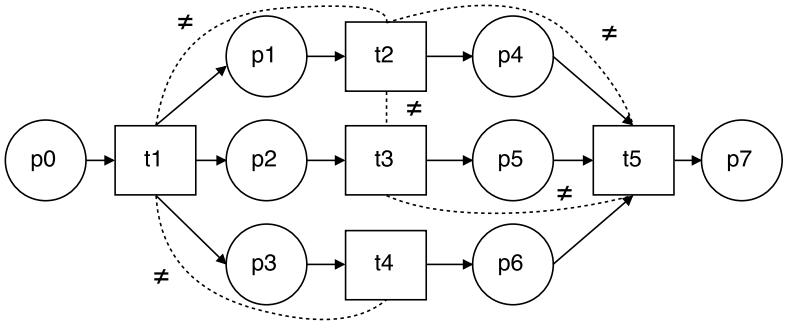
Similar generalizations of the MUB-SFP and the SkR-SFP are possible, thereby obtaining their constrained versions that we call C-MUB-SFP and C-SkR-SFP, respectively. More precisely, a solution to the C-MUB-SFP is a scenario that uses a minimal number of users and makes all the predicates in the set of constraints evaluate to true. A solution to the C-SkR-SFP is a k-resilient set H of scenarios such that each scenario in H makes all the predicates in the set of constraints evaluate to true.

As we will see in Section 4.2, the procedures used to solve the unconstrained versions of the scenario finding problems can be lifted to solve also their constrained version.

4. From Solving the WSP to Solving SFPs

We now explain our approach to solve the three problems (B-SFP, MUB-SFP, and SkR-SFP) introduced in Section 3. The procedure to solve the B-SFP (Section 4.2) is used as a sub-procedure in those tackling the MUB-SFP (see again Section 4.2) and the SkR-SFP (Section 4.3). Preliminarily (Section 4.1), we recall the technique in [6] to synthesize run-time monitors for the WSP. Besides creating monitors for the WSP, this procedure is based on a (symbolic) model checking algorithm capable of generating a compact data structure representing scenario graphs (recall the discussion in the Introduction) which allow us to compactly encode all possible interleavings of tasks in a workflow. Both the run-time monitors and the data structure representing scenario graphs are used by the procedure to solve the B-SFP.

The procedures to solve the constrained versions of the three scenario finding problems are illustrated in Section 4.4.



id	enabled		action	
	CF	Auth	CF	Auth
$t1(u)$	$p0 \wedge \neg d_{t1}$	$a_{t1}(u)$	$p0, p1, p2, p3, d_{t1} := F, T, T, T, T$	$h_{t1}(u) := T$
$t2(u)$	$p1 \wedge \neg d_{t2}$	$a_{t2}(u) \ \wedge \ \neg h_{t3}(u) \wedge \neg h_{t1}(u)$	$p1, p4, d_{t2} := F, T, T$	$h_{t2}(u) := T$
$t3(u)$	$p2 \wedge \neg d_{t3}$	$a_{t3}(u) \wedge \neg h_{t2}(u)$	$p2, p5, d_{t3} := F, T, T$	$h_{t3}(u) := T$
$t4(u)$	$p3 \wedge \neg d_{t4}$	$a_{t4}(u) \wedge \neg h_{t1}(u)$	$p3, p6, d_{t4} := F, T, T$	$h_{t4}(u) := T$
$t5(u)$	$p4 \wedge p5 \wedge p6 \wedge \neg d_{t5}$	$a_{t5}(u) \ \wedge \ \neg h_{t3}(u) \wedge \ \neg h_{t2}(u)$	$p4, p5, p6, p7, d_{t5} := F, F, F, T, T$	$h_{t5}(u) := T$

Fig. 2. TRW as an extended Petri net (top) and as a transition system (bottom)

4.1. Synthesizing monitors for the WSP

The technique in [6] is key to our approach as it provides us with a compact data structure to represent the set of all eligible scenarios in a workflow, which is crucial for the design of an efficient solution to SFPs. It takes as input the specification of a security-sensitive workflow (e.g., the BPMN in Figure 1 for the TRW) together with the specification of an authorization policy (such as the relation TA of Example 2.2) and consists of an *off-line* and an *on-line* step. In the former, the specification of a security-sensitive workflow with a finite set T of tasks, a finite set U of users, and a finite set C of authorization constraints is translated to a symbolic transition system $S = (V, Tr)$ [34] where V is a finite set of state variables and Tr is a finite set of transitions. The authorization policy TA is not taken into consideration in the off-line step as the technique is capable of synthesizing a monitor for the WSP which is specialized to the particular TA only in the on-line step. For concreteness, we illustrate how to translate the TRW in BPM notation of Figure 1 into a symbolic transition system.

Example 4.1. *Figure 2 shows the (extended) Petri net corresponding to the BPM Figure 1 and the corresponding derived transition system $S = (V, Tr)$. The semantics of the BPM notation has been given in terms of Petri nets and it is well-known how to translate these into symbolic transition systems (for more details and further references on these transformations, the interested reader is pointed to [6]). The set V of state variables contains the (Boolean) control-flow variables p_i and d_{ti}, where p_i represents the existence of a token in the place pi of the Petri net at the top of the same figure and d_{ti} represents the fact that transition ti of the Petri net at the top of the same figure has been executed. Additionally, V contains the authorization variables a_{ti} and h_{ti} that are (Boolean) arrays such that $a_{ti}(u)$ means that user u is entitled to execute task ti and $h_{ti}(u)$ means that user u has executed task ti.*

The transitions in Tr are listed in the table at the bottom of Figure 2 and are composed of three parts: an id(entifier), an enabling condition, and an effect. To illustrate, consider the second line of the table: the id indicates that user u executes task $t2$, the enabling condition is composed of two parts CF, which

stands for control-flow, and $Auth$, which stands for authorization. The enabling condition CF is the conjunction of predicates $p1$ and $\neg d_{t2}$ indicating that, for this event to be enabled, there must be a token in place $p1$ of the Petri net (at the top of the same figure) and task $t2$ has not been executed yet. The enabling condition $Auth$ is the conjunction of predicates $a_{t2}(u)$, indicating that the user requesting to execute this task must be authorized to do so by the authorization policy (i.e. $(u,t) \in TA$), $\neg h_{t3}(u)$, indicating that the user requesting to execute this task should not have executed task $t3$ (notice that $t2$ and $t3$ can be executed in parallel, due to the gateway), and $\neg h_{t1}(u)$, indicating that the user requesting to execute this task should not have executed task $t1$ (notice that the SoD constraint between $t2$ and $t5$ is not present in $t2(u)$ because $t5$ is always executed after $t2$). The effect is also divided in a CF and an $Auth$ part. The effect of executing $t2$ at the control flow level (CF) is to remove a token from place $p1$ and put a token in place $p4$ (formally, this is done by setting $p1$ to $False$ and $p4$ to $True$, as well as recording the execution of $t2$ by setting d_{t2} to $True$). The effect of executing $t2$ at the authorization level ($Auth$) is to update the history function h_{t2} to record the fact that $t2$ has been executed by user u (formally, $h_{t2}(u) := T$). □

The core of the off-line step of the technique in [6] is the computation of the data structure representing scenario graphs. The symbolic transition system S is taken as input to a model checking procedure which generates a *(symbolic) reachability graph* RG which represents all eligible execution scenarios. Formally, $RG = (N, \lambda, E)$ is a directed graph whose edges (in the set E) are labeled by task-user pairs in which users are symbolically represented by variables (called *user variables*) and whose nodes (in the set N) are labeled (according to the labeling function λ) by a symbolic representation (namely, a formula of first-order logic) of the set of states from which it is possible to reach a state in which the workflow successfully terminates (for the TRW, this is the set of states in which all five tasks have been executed).

The details of the construction of RG are given in [6]. Briefly, the procedure works as follows. Initially, a node is created, labeled with a formula describing the set of final states in the execution of S, and added to a set of nodes to be visited TBV. Then the procedure picks a node $\nu \in TBV$ and tries to apply each possible transition τ to the formula ϕ labeling ν; if the application is successful, the procedure creates a new node ν', labels ν' with ϕ' (the formula resulting from the application of τ), and connects ν' to ν using a new edge labeled by τ and a fresh user variable. The procedure stops when it reaches a fixpoint, i.e. when the application of new transitions leads only to states that have already been visited.

A sequence $\eta_s = t_1(\upsilon_{j_1}), ..., t_n(\upsilon_{j_n})$ of task-user pairs is a *symbolic execution scenario* where υ_{j_i} is a user variable with $1 \leq j_i \leq n$ and $i = 1, ..., n$. A *well-formed* path in RG is a path starting with a node without an incoming edge and ending with a node without an outgoing edge.

The crucial property of RG is that the symbolic execution scenario $\eta_s = t_1(\upsilon_{j_1}), ..., t_n(\upsilon_{j_n})$ collected while traversing one of its well-formed paths corresponds to an eligible (i.e. not violating any constraint in C) execution scenario $\eta_c = t_1(\mu(\upsilon_{j_1})), ..., t_n(\mu(\upsilon_{j_n}))$ for μ an injective function from the set $\Upsilon = \{\upsilon_{j_1}, ..., \upsilon_{j_n}\}$ of user variables (also called *symbolic users*) to the given set U of users of $W(T, U)$.

Three observations are in order. First, μ is extended to symbolic execution scenarios in the obvious way, i.e. by applying it to each user variable occurring in them. Second, since j_i can be equal to $j_{i'}$ for $1 \leq i \neq i' \leq n$, the cardinality of Υ is at most equal to the number n of tasks in the symbolic execution scenario. Third, since μ is injective, distinct user variables are never mapped to the same user.

Example 4.2. *An excerpt of the symbolic reachability graph for the TRW is depicted in Figure 3. The graph is not complete and the formulae labeling the nodes are not shown in the figure for the sake of simplicity.*

#	History	Query	Answer
0	$\emptyset$	$can\_do(a, t1)$	deny
1	-	$can\_do(b, t1)$	grant
2	$h(t1, b)$	$can\_do(c, t3)$	grant
3	$h(t3, c)$	$can\_do(a, t4)$	grant
4	$h(t4, a)$	$can\_do(b, t2)$	deny
5	-	$can\_do(a, t2)$	grant
6	$h(t2, a)$	$can\_do(b, t5)$	grant
7	$h(t5, b)$	-	-

Table 1
A run of the monitor program for the TRW

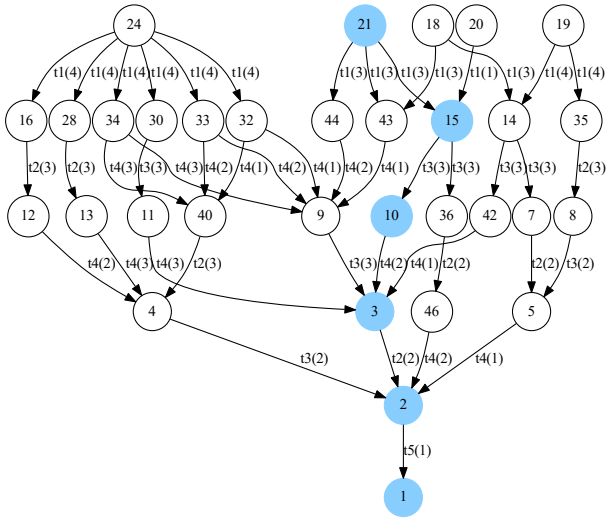


Fig. 3.
An excerpt of the symbolic reachability graph for the TRW

The graph in the Figure is constructed as follows. First, the procedure creates node 1*, labels it with the formula*

$$\beta_1 := p7 \bigwedge_{i=0,...,6} \neg pi \bigwedge_{i=1,...,5} d_{ti}$$

representing the final state in the execution of the TRW (when there is just one token in $p7$ *and all the tasks have been executed), and adds* 1 *to the set* TBV*. Then, the procedure picks node* 1 *from* TBV *(it is the only node in the set) and tries to apply transitions* $t1, ..., t5$*. Only* $t5$ *can be applied (due to the execution constraints) and the result of its application is the formula*

$$\beta_2 := \left(\begin{array}{l} \neg p0 \wedge \neg p1 \wedge \neg p2 \wedge \neg p3 \wedge p4 \wedge p5 \wedge p6 \wedge d_{t1} \wedge d_{t2} \wedge d_{t3} \wedge d_{t4} \wedge \neg d_{t5} \wedge \\ a_{t5}(\upsilon 1) \wedge \neg h_{t2}(\upsilon 1) \wedge \neg h_{t3}(\upsilon 1) \end{array} \right)$$

describing the set of states from which it is possible to execute $t5$*: there should not be tokens in* $p0, ..., p3$*, there should be tokens in* $p4, ..., p6$*, tasks* $t1, ..., t4$ *should have been executed, but not task* $t5$ *(first line), there should be a user* $\upsilon 1$ *authorized to execute* $t5$ *and who has not executed neither* $t2$ *nor* $t3$ *(second line). The procedure then creates node* 2*, labels it with* β_2*, and connects node* 2 *to node* 1 *by creating an edge labeled by* $t5(1)$*. Node* 1 *is removed from* TBV *and node* 2 *is added to the same set. In the next step, the procedure picks node* 2 *from* TBV *and tries to apply transitions* $t1, ..., t5$ *to* β_2*. As shown in Figure 3 transitions* $t2$*,* $t3$*, and* $t4$ *can be applied, which leads to the creation of nodes* 3*,* 4*,* 5*, and* 46*. There are two nodes for* $t4$ *because it can be executed by the same symbolic user who executes* $t5$ *or a fresh one, since there are no constraints between* $t4$ *and* $t5$*;* $t2$ *and* $t3$*, on the other hand, must be executed by a different user because of the SoD constraints. To continue the example, we show the formula*

$$\beta_3 := \left(\begin{array}{l} \neg p0 \wedge p1 \wedge \neg p2 \wedge \neg p3 \wedge \neg p4 \wedge p5 \wedge p6 \wedge d_{t1} \wedge \neg d_{t2} \wedge d_{t3} \wedge d_{t4} \wedge \neg d_{t5} \wedge \\ a_{t2}(\upsilon 2) \wedge \neg h_{t1}(\upsilon 2) \wedge \neg h_{t3}(\upsilon 2) \wedge a_{t5}(\upsilon 1) \wedge \neg h_{t3}(\upsilon 1) \wedge \neg h_{t2}(\upsilon 1) \wedge \upsilon 1 \neq \upsilon 2 \end{array} \right)$$

attached to node 3 of the graph, describing the set of states from which it is possible to execute $t2$. Intuitively, β_3 expresses the conditions for a user $\upsilon 2$ to be allowed to execute $t2$: the workflow (instance) must be in a state where there are tokens in places $p1$, $p5$, and $p6$ while there are no tokens in places $p0$, $p2$, $p3$, and $p4$ (first line), tasks $t1$, $t3$, and $t4$ have been already executed while tasks $t2$ and $t5$ have not been executed (first line), user $\upsilon 2$ should be authorized to perform $t2$ and should not have executed neither $t1$ nor $t3$ (second line), and there should exist a user $\upsilon 1$ (distinct from $\upsilon 2$) authorized to execute $t5$ who should have executed neither $t1$ nor $t3$ (second line). The procedure continues until a fixpoint is reached when trying to apply transitions in the leaf nodes 18, 19, 20, 21, and 24, which represent the initial states in the execution.

Concerning the labels on the edges of the symbolic reachability graph, in Figure 3, a task-user pair $t(\upsilon_k)$ labeling an edge is abbreviated by $t(k)$ for the sake of compactness. So, for instance, the symbolic execution scenario $\eta_s = t1(\upsilon_3), t3(\upsilon_3), t4(\upsilon_2), t2(\upsilon_2), t5(\upsilon_1)$ (cf. the well-formed path identified by the blue nodes in Figure 3) represents all those executions in which a symbolic user identified by υ_3 first performs task $t1$ followed by $t3$, then a symbolic user identified by υ_2 performs $t4$ and $t2$ in this order, and finally a symbolic user identified by υ_1 executes $t5$. If we apply an injective function μ from the set $\Upsilon = \{\upsilon_1, \upsilon_2, \upsilon_3\}$ of user variables to any finite set U of users (of cardinality at least three), the corresponding execution $\eta_c = \mu(\eta_s)$ is eligible according to the set C of SoD constraints shown in Figure 1. □

To conclude the off-line step, the technique derives a non-recursive Datalog program M (with negation) from the symbolic reachability graph RG by generating a clause of the form $can\_do(\upsilon, t) \leftarrow \beta_v$ for each node v in the graph RG.

Example 4.3. *To illustrate, consider again node 3 in the graph as done in Example 4.2. The Datalog program M will contain the following clause:*

$$can\_do(t2, \upsilon 2) \leftarrow \neg p0, p1, \neg p2, \neg p3, \neg p4, p5, p6, d_{t1}, \neg d_{t2}, d_{t3}, d_{t4}, \neg d_{t5}, \\ a_{t2}(\upsilon 2), \neg h_{t1}(\upsilon 2), \neg h_{t3}(\upsilon 2), a_{t5}(\upsilon 1), \neg h_{t3}(\upsilon 1), \neg h_{t2}(\upsilon 1), \upsilon 1 \neq \upsilon 2.$$

where the comma stands for logical conjunction. □

We now describe the on-line step which consists of building a run-time monitor for the WSP by taking into account the authorization policy TA. Technically, this is done by combining the Datalog program M obtained in the off-line step with the Datalog program P specifying the authorization policy TA. The combination of M and P is possible because of the following observation. As shown in Example 4.3, the formula β_v contains invocations to the binary predicates a_{ti} and h_{ti}. The former is the interface to the authorization policy and it is such that $a_t(u)$ holds iff $(u, t) \in TA$ while the latter keeps track of which user has executed which task, i.e. $h(t, u)$ means that t has been executed by u. Following an established tradition (see, e.g., [23]) claiming that (variants of) Datalog are adequate to express a wide range of access control policy idioms, we assume the predicates a_t's to be defined by the Datalog program P. The predicate h is dynamic and defined by a set H of (ground) facts which is updated after each task execution. Thus, if the query $can\_do(u, t)$ can be derived from M, P, H (in symbols, $M, P, H \vdash can\_do(u, t)$), then user u can execute task t and the workflow can terminate while satisfying the authorization policy and the authorization constraints. We illustrate these ideas on the TRW.

Example 4.4. *Figure 4 depicts the monitor program M for the TRW (obtained in the off-line step) combined (in the on-line step) with an authorization policy P specified in Datalog (from a user-task*

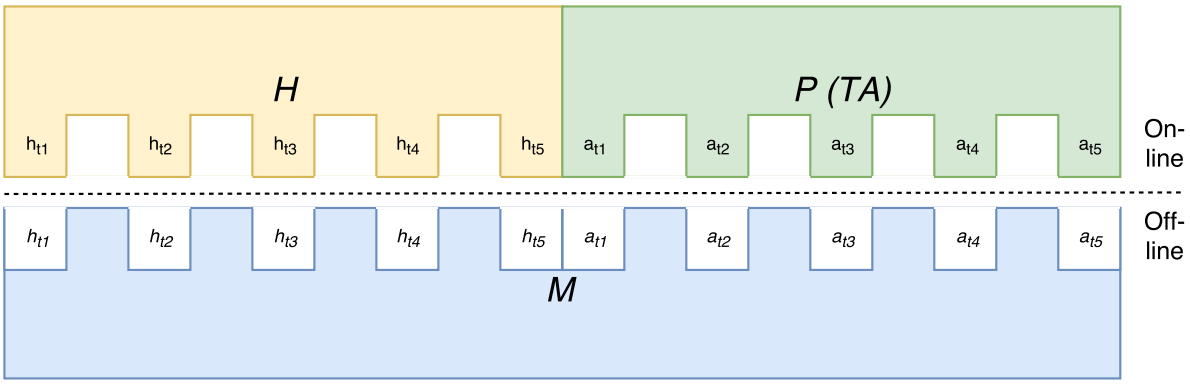


Fig. 4. Monitor for the TRW combined with an authorization policy and an execution history

relation TA) and an execution history H. The "holes" in M ($h_{t1}, \ldots, h_{t5}$ and $a_{t1}, \ldots, a_{t5}$) are "filled" by the definitions in P and H. Thus, the combination M, P, H is used to answer the queries $can\_do(u, t)$.

Let us consider again the set of users and the authorization policy discussed in Example 2.2. The relation TA can be specified after the Role Based Access Control (RBAC) model [33] by the Datalog program P:

$$ua(a, r1).\ ua(a, r2).\ ua(a, r3).\ ua(b, r2).\ ua(b, r3).\ ua(c, r2).$$
$$pa(r_3, t1).\ pa(r_2, t2).\ pa(r_2, t3).\ pa(r_1, t4).\ pa(r_2, t5).$$
$$a_{t1}(\upsilon) \leftarrow ua(\upsilon, \rho),\ pa(\rho, t1).\ a_{t2}(\upsilon) \leftarrow ua(\upsilon, \rho),\ pa(\rho, t2).\ a_{t3}(\upsilon) \leftarrow ua(\upsilon, \rho),\ pa(\rho, t3).$$

where $r1$, $r2$, and $r3$ are roles, ua is the user-role assignment (cf. first line of facts), pa is the role-task assignment (cf. second line of facts), υ is a user variable, τ is a variable ranging over tasks, and a_τ is defined as the join of the relations ua and pa (cf. the Datalog clauses in the last line) for each $\tau \in \{t1, t2, t3\}$. Recall the definition of TA in Example 2.2 and notice that $P \vdash a_t(u)$ iff $(u, t) \in TA$ for user u and task t.

An example run of the monitor derived from the symbolic reachability graph in Figure 3 combined with the RBAC policy above is shown in Table 1: column 'History' shows which facts are added to the set H and column 'Answer' reports grant (deny, respectively) when the query in column 'Query' can (cannot, respectively) be derived from M, P, H. For instance, there are two denied requests: in line 0, user a requests to execute task $t1$ but this is not possible since a is the only user authorized to execute $t4$, and if a executes $t1$, he/she will no more be allowed to execute $t4$ because of the SoD constraint between $t1$ and $t4$ (see Figure 1); in line 4, user b requests to execute task $t2$ but again this is not possible since b has already executed task $t1$ and this would violate the SoD constraint between $t1$ and $t2$. All other requests are granted, as they violate neither task execution nor authorization constraints. The execution resulting from this run of the monitor is $t1(b), t3(c), t4(a), t2(a), t5(b)$, which is derived from the symbolic execution $t1(\upsilon_1), t3(\upsilon_3), t4(\upsilon_2), t2(\upsilon_2), t5(\upsilon_1)$ in the graph of Figure 3 (cf. the path with the blue nodes; see also Example 4.2) by applying the injective function μ mapping υ_1 to b, υ_2 to a, and υ_3 to c. □

4.2. Solving the B-SFP and the MUB-SFP

Preliminarily, we need to decide how the set E of eligible scenarios and the authorization policy TA are specified as input to the algorithm solving the scenario finding problems. For TA, we have already assumed

(see the discussion at the end of the previous section when describing the on-line step) the availability of a Datalog program P defining the binary predicates a_t's for each task t in the workflow. For E, we define the *set $E(RG, U)$ of eligible scenarios induced by a symbolic reachability graph RG and a set U of users* as the collection of all the scenarios of the form $\mu(t_1(\upsilon_{j_1}), ..., t_n(\upsilon_{j_n}))$ where $v_0 \xrightarrow{t_1(\upsilon_{j_1})} \cdots \xrightarrow{t_n(\upsilon_{j_n})} v_{n+1}$ is a well-formed path in RG and μ is an injective function from $\Upsilon = \{\upsilon_{j_1}, ..., \upsilon_{j_n}\}$ to U.

Two observations are important. First, there are several different sets $E(RG^*, U)$ induced by a fixed symbolic reachability graph RG^* and a varying set U of users. Second, a symbolic reachability graph—once a set of users is fixed—provides an implicit and compact representation of the set of eligible scenarios. This is due to two reasons: one is the sharing of common sub-sequences of task-user pairs in execution scenarios and the other is the symbolic representation of several execution scenarios by means of a single symbolic execution scenario. This is best illustrated by an example.

Example 4.5. *Let us consider the TRW with a set U of 6 users. The graph in Figure 3 is, for the sake of readability, an excerpt of the full symbolic reachability graph showing only a small sub-set of all well-formed paths. The full graph has 46 nodes, 81 edges, and 61 well-formed paths of which 21, 34, and 6 contain 3, 4, and 5, respectively, symbolic users. For instance, notice how the sub-sequence $t3(\upsilon_2), t5(\upsilon_1)$ is shared by 6 distinct (symbolic) execution scenarios induced by the well-formed paths whose initial node is 24 (left of figure). Additionally, observe that, from the definition of $E(RG, U)$ above, in order to establish the number of all eligible paths when $|U| = n$, we just need to calculate how many injective functions there are from a set of cardinality k to a set of cardinality n—which is known to be $J(n, k) = n(n-1)(n-2)\cdots(n-k+1)$—for $n = 6$, $k = 3, 4, 5$, and take their sum. Thus, the set of all eligible paths in our case is $21 \cdot J(6, 3) + 34 \cdot J(6, 4) + 6 \cdot J(6, 5) = 19,080$. Compare this, with the number of well-formed paths in the symbolic reachability graph which is only 61: the blow-up factor is more than 300. Indeed, the increase is even more dramatic for larger sets of users.* □

We are now ready to describe our technique, depicted in Algorithm 1, to solve the B-SFP. For the time being, let us ignore the additional input set Γ (by setting it to $\emptyset$); it will be explained in Section 4.4 below. The main idea underlying Algorithm 1 is to adapt a standard Depth-First Search (DFS) search to explore all well-formed paths in the reachability graph RG while checking that the scenario associated to the path is indeed authorized by using the run-time monitor, synthesized in the on-line phase of the technique in [6] (recall Section 4.1). Lines 1–2 are the standard initialization phase of a DFS algorithm in which all nodes in RG (returned by the function Nodes) are marked as not yet visited. Lines 3–6 invoke the (modified) DFS algorithm on each node without an incoming edge in RG (returned by the function NoIncoming) until either all such nodes have been considered (this allows us to consider all well-formed paths) or an authorized scenario (if any) has been found (line 7). Lines 8–19 show the (modified) DFS recursive function which takes as input a node v and extends a sequence η of task-user pairs to an authorized execution scenario (if possible). Line 9 marks as visited the node v under consideration and computes its set OE of outgoing edges (returned by the function OutGoing). Line 10 checks whether the set of outgoing edges of v is empty: if this is the case, then we have considered all task-user pairs in a well-formed path and the sequence η containing them is an authorized execution scenario. If this is not the case, we have not yet considered all task-user pairs in a well-formed path of RG and thus we need to consider the possible continuations in OE. This is done in the loop at lines 12-16: an edge $v \xrightarrow{t(v)} w$ in OE is selected (line 12), it is checked if the node w is not yet visited and if the run-time monitor combined with the authorization policy P can find a user u capable of executing the task t in label of the edge in OE under consideration (line 13). The second check (namely, $M, P, H \vdash^{v \mapsto u} can\_do(t, v)$) is done by asking a Datalog engine

to find a user u in U to which the user variable v can be mapped (cf. superscript $v \mapsto u$ of $\vdash$) without violating the execution and the authorization constraints together with the authorization policy specified by P. If the test at line 13 is successful, line 14 is executed whereby a recursive call to the DFS function is

Algorithm 1. Solving the B-SFP

Input: RG symbolic reachability graph, U set of users, P Datalog program defining a_t's, Γ set of facts
Output: η authorized execution scenario

1: **for all** $v \in \mathsf{Nodes}(RG)$ **do** $visited[v] \leftarrow \texttt{false}$;
2: **end for**
3: $\eta \leftarrow \epsilon$; $NI \leftarrow \mathsf{NoIncoming}(RG)$;
4: **while** ($v \in NI$ **and** $\eta = \epsilon$) **do**
5: $\quad \eta \leftarrow \mathrm{DFS}(v, \epsilon, \Gamma)$; $NI \leftarrow NI \setminus \{v\}$;
6: **end while**
7: **return** η
8: **function** $\mathrm{DFS}(v, \eta, H)$
9: $\quad visited[v] \leftarrow \texttt{true}$; $OE \leftarrow \mathsf{OutGoing}(v)$;
10: $\quad$ **if** $OE = \emptyset$ **then return** η
11: $\quad$ **else**
12: $\quad\quad$ **for all** $v \xrightarrow{t(v)} w \in OE$ **do**
13: $\quad\quad\quad$ **if** (**not** $visited[w]$ **and** $M, P, H \vdash^{v \mapsto u} can\_do(t, v)$) **then**
14: $\quad\quad\quad\quad$ **return** $\mathrm{DFS}(w, \mathsf{append}(\eta, t(u)), H \cup \{h(t, u)\})$
15: $\quad\quad\quad$ **end if**
16: $\quad\quad$ **end for**
17: $\quad$ **end if**
18: $\quad$ **return** ϵ
19: **end function**

Algorithm 2. Solving the MUB-SFP

Input: RG symbolic reachability graph, U set of users, P Datalog program defining a_t's, Γ set of facts
Output: η authorized execution scenario

1: **for all** $v \in \mathsf{Nodes}(RG)$ **do** $visited[v] \leftarrow \texttt{false}$;
2: **end for**
3: $\eta \leftarrow \epsilon$; $\eta_M \leftarrow \epsilon$; $NI \leftarrow \mathsf{NoIncoming}(RG)$;
4: **while** $v \in NI$ **do**
5: $\quad \eta \leftarrow \mathrm{DFS}(v, \epsilon, \Gamma)$; $NI \leftarrow NI \setminus \{v\}$;
6: $\quad$ **if** ($\eta \neq \epsilon$ **and** $\eta_M = \epsilon$) **then**
7: $\quad\quad \eta_M \leftarrow \eta$;
8: $\quad$ **else if** ($\eta \neq \epsilon$ **and** $|usr(\eta)| < |usr(\eta_M)|$) **then**
9: $\quad\quad \eta_M \leftarrow \eta$;
10: $\quad$ **end if**
11: **end while**
12: **return** η_M

performed in which the new node to consider is w, the sequence η of task-user pairs is extended with $t(u)$ (by invoking the function append), and the set H of facts keeping track of the tasks executed so far is also extended by $h(t, u)$. In case all edges in OE have been considered but none of them makes the check at line 13 successful, the empty sequence is returned (line 18). Notice that at line 13, instead of enumerating all suitable users in U to which v can be mapped, we exploit the capability of the Datalog engine to find the right user. This permits us to exploit well-engineered implementations of Datalog engines instead of designing and implementing new heuristics to reduce the time taken to enumerate the users in U. This concludes the description of the algorithm solving the B-SFP.

It is possible to modify Algorithm 1 following the idea discussed after Example 3.2 in order to *solve the MUB-SFP*. This requires to avoid returning the authorized scenario as soon as we find one (removing the condition $\eta = \epsilon$ in line 4) so that all well-formed paths in RG are considered. Moreover, a global variable η_M is maintained in which a candidate scenario with a minimal user base is stored and updated according to the strategy discussed above comparing the users occurring in η_M and those in the currently considered scenario. As a result of these modifications, we obtain Algorithm 2 that is capable of solving the MUB-SFP. The DFS function used in this algorithm is the same as the DFS function in Algorithm 1).

Complexity of Algorithms 1 and 2. The complexity of both algorithms can be derived from that of the standard DFS algorithm, which is $O(n + m)$ for n the number of nodes and m the number of edges, when using an adjacency list to represent the graph. Notice that the most computationally intensive operation is the invocation of the Datalog engine at line 13, which takes polynomial time as the only part that changes over time is the set H of facts whereas the Datalog programs M and P are fixed; cf. the results on data complexity of Datalog programs in [7]. It is easy to see that we invoke (at most) $O(n + m)$ times the Datalog engine for both algorithms (line 13 in Algorithm 1 and hidden in the DFS function in Algorithm 2). This is much better than the upper bounds discussed in Section 3. To see this, consider the situation in Example 4.5: $\ell_{max} = 5$ and $|E| = 19,080$ so that the check for authorization (modulo constant factors) is invoked at most $95,400$ times whereas in Algorithm 1 (or its modified version for the MUB-SFP) the same check is invoked at most $n + m = 46 + 81 = 127$ times.

4.3. Solving the SkR-SFP

As discussed in Section 3.3, a naive solution to the SkR-SFP requires the enumeration of all sub-sets of the set of users of a given size. Indeed, this is very expensive from a computational point of view. To alleviate this, it is possible to adapt an heuristic proposed in [24] and reduce the number of sets of a given size to be considered. The idea is based on the notion of "dominance" which is defined in terms of which tasks a given set of users can execute. If a set A_1 of users can execute at least the same set of tasks that a second set A_2 of users can execute, then we say that A_1 dominates A_2 and the latter can be ignored without loss of generality. This implies that if an instance of the SkR-SFP can tolerate the removal of users in a set of absent users A_1, then the problem instance can tolerate the removal of users in the second set A_2. This is why we can consider only the set A_1 and safely disregard A_2. Below, the set of users in the authorization policy TA is $\{u \mid (u, t) \in TA\}$ and the set $Perms(u)$ of permissions associated to a user u is $\{t \mid (u, t) \in TA\}$.

Definition 4.1 (Absent set domination)**.** *A user u_1 in TA dominates a user u_2 in TA iff $Perms(u_1) \supseteq Perms(u_2)$. A set of (absent) users A_1 dominates a set A_2 of (absent) users iff there exists a bijection f from A_1 to A_2 such that, for each user u in A_1, u dominates $f(u)$ in A_2.*

We are now in the position to adapt the result (Lemma 7) in [24] to avoid the enumeration of all sub-sets of a given size while solving the SkR-SFP.

Lemma 4.1. *Let E be a set of eligible scenarios according to a set C of authorization constraints containing only SoD (i.e. ρ is $\neq$ for each $(t, t', \rho) \in C$) in a workflow $W(T, U)$ and an integer k. If $A_1, A_2 \subseteq U$, A_1 dominates A_2, and the Sk**R**-SFP for $W(T, U \setminus A_1)$ is solvable, then the Sk**R**-SFP for $W(T, U \setminus A_2)$ is also solvable.*

Proof. (Sketch) Assume that A_1 dominates A_2 and the SkR-SFP for $W(T, U \setminus A_1)$ is solvable. The former is equivalent to

$$Perms(u) \supseteq Perms(f(u)) \quad \text{for each } u \text{ in } U \setminus A_1 \tag{1}$$

while the latter implies (by recalling the definition of solution to the WSP) that

$$\forall t \in T, \exists u \in U \setminus A_1.\ (u, t) \in TA \tag{2}$$

$$\forall (t_1, t_2, \rho) \in C, \exists u_1, u_2 \in U \setminus A_1.\ (u_1, u_2) \in \rho, \tag{3}$$

where (2) states that each task in a scenario solving the WSP is performed by an authorized user and (3) states that each authorization constraint is satisfied.

By (1), it is possible to show that if (2) holds (i.e. the problem is solvable after the removal of A_1), it also holds when replacing A_2 with A_1 (i.e. the problem is solvable after the removal of A_2), because A_2 contains the same number of users of A_1 (since f is a bijection) and each user in A_2 has the same permissions or less of the corresponding user (via the bijection f) in A_1 (a more detailed proof of this property proceeds along the same line of that in Lemma 7 in [24]).

Unfortunately, it is not possible to show for an arbitrary set C of constraints that (3) implies (3) with A_2 in place of A_1. This is so because we can imagine a scenario where the users removed in A_2, which are different from those in A_1, are necessary to fulfill some of the constraints in C. A somewhat contrived example is a constraint requiring a task $t2$ to be executed by a user more senior, given a seniority relation $<$ inside an organization [11], than that who executes $t1$, and a less senior user $u1$ with more permissions than a user $u2$, so that $u1 < u2$, but $Perms(u_1) \supseteq Perms(u_2)$. Although $u1$ dominates $u2$, there may be no user able to execute $t2$ due to the constraint. Fortunately, when C contains only SoD constraints, one can observe that the bijection f from A_1 to A_2 maps distinct users into distinct users. As a consequence, from an execution scenario solving the SkR-SFP for $W(T, U \setminus A_1)$, it is always possible to find an execution scenario solving the SkR-SFP for $W(T, U \setminus A_2)$. □

Lemma 4.1 allows us to avoid enumerating all possible sub-sets of a given size when solving the SkR-SFP by exploiting the dominance relation among sets of absent users in presence of SoD constraints only. (Finding a generalization of the notion of dominance for arbitrary authorization constraints is left as future work.) How this is done is shown in Algorithm 3. In line 1, the variable H which stores the scenarios to be returned is initialized to the empty set. In line 2, the function NextSubset returns a new, i.e. not previously generated, sub-set of size $t = |U| - k$ of the set U of users and s stores the returned sub-set. The NextSubset function is implemented as an iterator that returns only the next sub-set at each call, so that it is not necessary to generate all subsets up front, which is very expensive from a computational point of view. Most importantly, NextSubset implements the smart enumeration of sub-sets that are not dominated by others according to Lemma 4.1 when the set C contains only SoD constraints. In line 3, the Datalog program defining the a_t's is updated by removing all assertions that include users not in the current subset S ($P|_S$), and in line 4 the SFP function (implemented as shown in Algorithm 1) is called to find an execution scenario. If no scenario is found (line 5), an empty set is returned (line 6) and execution

halts. Notice that the execution must stop because the SkR-SFP is defined as finding a scenario for *every* sub-set of size t in U. If a scenario is found (line 7), it is added to the set H (line 8); after this is done for all sub-sets, NextSubset returns an empty set and the final H is returned (line 11).

4.4. Solving the C-SFP

So far, we have assumed that the set Γ of constraints taken as input of Algorithm 1 is empty. Here, we consider the situation in which this is no more the case and show how Algorithms 1, 2, and 3 also solve the constrained versions of the B-SFP, MUB-SFP, and the SkR-SFP.

Let us consider Algorithm 1 and how it solves a C-SFP by taking as input a non-empty set Γ of facts, that can be used to drive the search for a scenario with particular characteristics. For instance, one can be interested in authorized scenarios in which a certain user only, say u^*, executes a given task, say t^*. It is possible to steer the search towards such scenarios by setting Γ to be the singleton containing the fact $h(t^*, u^*)$. Another use of Γ is guiding the search towards scenarios in which the tests of certain conditionals of the workflows are either true or false. Again, it is possible to add the facts encoding that particular control conditions are true or false to Γ in order to force Algorithm 1 to find scenarios taking only particular execution branches.

A more interesting approach is to develop an interactive algorithm where after each step in the search for a solution, the user of the algorithm is asked to enter his or her preferences to which user executes which task and which is the next task to be executed. If he or she does not specify anything, the algorithm picks an arbitrary user and task; otherwise the algorithm checks if the corresponding instance of the WSP is solvable and, if the case, would allow the user to proceed with the exploration.

Since Algorithms 2 and 3 (i.e. the procedures to find the solutions to the MUB-SFP and to the SkR-SFP) use Algorithm 1 as a sub-procedure, they straightforwardly inherit the capability to solve the constrained versions of B-SFP and MUB-SFP.

Algorithm 3. Solving the SkR-SFP

Input: RG symbolic reachability graph, U set of users, P Datalog program defining a_t's, Γ set of facts, C set of authorization constraints, $k \in \mathbb{N}$ desired resiliency
Output: H set of authorized execution scenarios

1: $H \leftarrow \emptyset$;
2: **while** ($S \leftarrow \mathsf{NextSubset(k, U)}$ **and** $S \neq \emptyset$) **do**
3: $\quad P' \leftarrow P|_S$;
4: $\quad \eta \leftarrow \text{SFP}(RG, U, P', \Gamma)$;
5: $\quad$ **if** $\eta = \epsilon$ **then**
6: $\quad\quad$ **return** $\emptyset$;
7: $\quad$ **else**
8: $\quad\quad H \leftarrow H \cup \{\eta\}$;
9: $\quad$ **end if**
10: **end while**
11: **return** H;

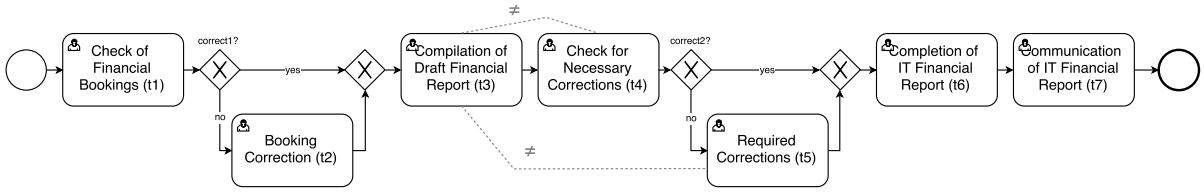


Fig. 5. ITIL 2011—IT Financial Reporting (abbreviated ITIL)

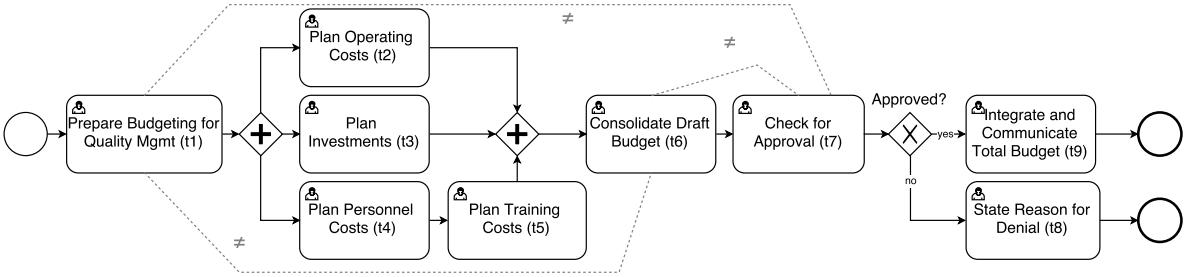


Fig. 6. ISO9000—Budgeting for Quality Management (abbreviated ISO)

5. Validation of the Technique

We consider two real-world examples—called ITIL and ISO, shown in Figures 5 and 6—derived from business processes available in an on-line library provided by Signavio,[3] which contains models inspired by the ISO9000 standard for quality management and the ITIL 2011 set of best practices for IT service management.

ITIL. The goal of this workflow is to report costs and revenues of an IT Service. It is composed of 7 tasks and 2 SoD constraints. Tasks $t1$, $t2$, $t3$, $t6$ and $t7$ are for the checking and correction of bookings, compilation of the financial report, and its communication; tasks $t4$ and $t5$ are for checking and defining corrections. The execution of tasks $t2$ and $t5$ depends on the conditions associated to two exclusive gateways: *correct1?* (abbreviated with $c1$) and *correct2?* (abbreviated with $c2$), respectively. The SoD constraints forbid that the same user compiles a draft report and checks for errors $(t3, t4, \neq)$ or compiles the draft and defines the corrections $(t3, t5, \neq)$.

ISO. The goal of this workflow is to plan for enough financial resources to fulfill some quality requirement. It is composed of 9 tasks and 3 SoD constraints. Tasks $t1$, $t2$, ..., $t6$ involve the detailed preparation and consolidation of a draft budget, whereas tasks $t7$, $t8$ and $t9$ are for the approval of the previous activities, the integration into the total budget, and the communication of the results. The execution of tasks $t8$ or $t9$ depends on the exclusive gateway *approved?* (abbreviated with $appr$). The SoD constraints forbid that the same user prepare and consolidate a budget $(t1, t6, \neq)$, prepare and approve a budget $(t1, t7, \neq)$, or consolidate and approve a budget $(t6, t7, \neq)$.

Although none of the workflows comes with an authorization policy, swimlanes (not shown in Figures 5 and 6) suggest that a controlling manager executes tasks $t1$, $t2$, $t3$, $t6$ and $t7$ while a financial manager executes tasks $t4$ and $t5$ for ITIL and that a quality manager executes tasks $t1$, ..., $t6$ and a controlling

[3] Available at `http://www.signavio.com/reference-models/`

manager executes tasks $t7$, $t8$ and $t9$ for ISO. These indications are taken into consideration for designing the authorization policies (based on the RBAC model and encoded in Datalog) in various scenarios with a fixed set U of 9 users when solving the B-SFP, the MUB-SFP, and their constrained versions. For TRW, we consider two policies P_0 and P_1: the former is that in Example 4.4 and the latter is derived from the former in such a way that no user is authorized to execute $t1$ (thus no authorized scenario should be found). For ITIL, we have policies P_2 and P_3, each one with 3 users as financial managers, 3 users as controlling managers, and 3 with both roles; P_3 is derived from P_2 by preventing users to be able to execute task $t6$. For ISO, we consider policies P_4 and P_5, each one with 3 users assigned to the role of quality manager, 3 users as controlling managers, and 3 users assigned to both roles; P_5 is derived from P_4 by preventing users to be able to execute task $t3$.

Before invoking our algorithms for solving the SFPs, we need to build the symbolic reachability graph (and the run-time monitor) for each example. We did this by running the implementation of the off-line step (described in Section 4) from [6]. For TRW, the symbolic reachability graph is computed in around a second and contains 46 nodes with 81 edges. For ITIL, the graph is computed in around 3.5 seconds and has 78 nodes with 72 edges. For ISO, graph building takes around 10.5 seconds and has 171 nodes with 669 edges. These timings, as well as all those that follow below, have been obtained by using a MacBook Air 2014 with Mac OS X v10.10.2. The time for deriving the monitor M from the symbolic reachability graph of each example is negligible and thus omitted.

We have implemented Algorithms 1, 2, and 3 (described in Section 4.2) in Python v2.7. The invocation to the Datalog engine at line 13 in Algorithm 1 (and reused in the other algorithms) is implemented with the Datalog engine pyDatalog v0.15.2.

Table 2 shows the findings of our experiments for the B-SFP, MUB-SFP, and C-SFP. Each entry in column 'Instance,' describing the input to Algorithm 1 (or its modification to solve the MUB-SFP), is of the form $W + P_i$ where W is the identifier of one of the three security-sensitive workflows and P_i is one of the authorization policies described above. Column 'Γ' shows the facts in the set Γ that can be used to drive the search of execution scenarios with particular properties. For instance, ITIL contains two exclusive gateways labeled with conditions $c1$ and $c2$: we may be interested in those scenarios in which $c1$ and $c2$ take some particular truth values (see lines 1–4 and 12–15 of the table). Another use of the set Γ is shown at line 8: we are interested in finding authorized scenarios of TRW under the authorization policy P_0 in which task $t2$ is always executed by user b. There is no such scenario (the 'Solution Scenario' column reports the empty sequence ϵ) since when b performs $t2$, a must perform $t1$—because of the SoD constraint $(t1, t2, \neq)$—but if a performs $t1$, no user can perform $t4$—because of the other SoD constraint $(t1, t4, \neq)$. Column 'Time' reports the running time (in seconds) taken to find a scenario (if any).

We report the performance of the implementation of Algorithm 3, to solve instances of the SkR-SFP on the TRW, ITIL, and ISO workflows in Table 3. Column 'Instance' shows the workflows used in the experiments. Each one of the columns 'Time' reports the time taken (in seconds) to find a solution for $k = 2$, $k = 4$, $k = 6$, and $k = 8$. The experiments have been run with three policy configurations, containing $n = 9$, $n = 18$, and $n = 27$ users in U (indicated with $|U| = n$ in the table). The performance results are for the version of the algorithm using the sub-set enumeration optimization (discussed in Section 4.3), since all the workflows used in the experiments contain only SoD constraints. We do not report the performance without the optimization because it is unsuitable.

Discussion. Our experiments indicate that the SFPs, their constrained versions, and the algorithms for solving them introduced in this paper fit well with emerging practices for BPM reuse. Whenever a customer wants to deploy a business process by reusing a workflow template, some SFP is solved (if possible) to provide him or her with an authorized scenario showing that a template business process can

Table 2
Experiments for the B-SFP, MUB-SFP, and C-SFP

#	Instance	Γ	Solution Scenario	Time
B-SFP				
0	TRW+P_0	$\emptyset$	$t1(b), t2(a), t4(a), t3(c), t5(b)$	0.288
1	ITIL+P_2	$\{c1, c2\}$	$t1(u3), t3(u9), t4(u8), t6(u9), t7(u9)$	4.267
2	ITIL+P_2	$\{c1, \mathbf{not}\ c2\}$	$t1(u3), t3(u3), t4(u7), t5(u8), t6(u3), t7(u7)$	4.454
3	ITIL+P_2	$\{\mathbf{not}\ c1, c2\}$	$t1(u3), t2(u1), t3(u9), t4(u8), t6(u9), t7(u9)$	4.374
4	ITIL+P_2	$\{\mathbf{not}\ c1, \mathbf{not}\ c2\}$	$t1(u3), t2(u1), t3(u3), t4(u7),$ $t5(u8), t6(u3), t7(u7)$	4.561
5	ISO+P_4	$\{appr\}$	$t1(u3), t4(u7), t5(u8), t2(u3), t3(u7),$ $t6(u9), t7(u7), t9(u8)$	6.581
6	ISO+P_4	$\{\mathbf{not}\ appr\}$	$t1(u3), t4(u7), t5(u8), t2(u3), t3(u7),$ $t6(u7), t7(u8), t8(u6)$	6.637
7	TRW+P_1	$\emptyset$	ϵ	0.407
8	TRW+P_0	$\{t2(b)\}$	ϵ	1.554
9	ITIL+P_3	$\emptyset$	ϵ	9.562
10	ISO+P_5	$\emptyset$	ϵ	44.076
MUB-SFP				
11	TRW+P_0	$\emptyset$	$t1(b), t2(c), t3(b), t4(a), t5(a)$	2.385
12	ITIL+P_2	$\{c1, c2\}$	$t1(u1), t3(u1), t4(u7), t6(u1), t7(u1)$	108.819
13	ITIL+P_2	$\{c1, \mathbf{not}\ c2\}$	$t1(u3), t3(u3), t4(u7), t5(u7), t6(u3), t7(u3)$	116.525
14	ITIL+P_2	$\{\mathbf{not}\ c1, c2\}$	$t1(u1), t2(u1), t3(u1), t4(u7), t6(u1), t7(u1)$	108.827
15	ITIL+P_2	$\{\mathbf{not}\ c1, \mathbf{not}\ c2\}$	$t1(u3), t2(u3), t3(u3), t4(u7),$ $t5(u7), t6(u3), t7(u3)$	116.533
16	ISO+P_4	$\{appr\}$	$t1(u5), t3(u5), t2(u5), t4(u5), t5(u5),$ $t6(u3), t7(u7), t9(u7)$	166.632
17	ISO+P_4	$\{\mathbf{not}\ appr\}$	$t1(u5), t3(u5), t2(u5), t4(u5), t5(u5),$ $t6(u9), t7(u6), t8(u9)$	166.644

Table 3
Experiments for the SkR-SFP

#	Instance	Time ($k=2$)	Time ($k=4$)	Time ($k=6$)	Time ($k=8$)
$\lvert U\rvert = 9$					
0	TRW	2.612	3.209	3.990	0.504
1	ITIL	3.606	6.012	7.416	1.308
2	ISO	10.815	12.034	13.820	26.254
$\lvert U\rvert = 18$					
3	TRW	91.606	95.892	109.895	226.074
4	ITIL	114.128	159.105	161.935	165.059
5	ISO	107.864	135.725	197.564	207.941
$\lvert U\rvert = 27$					
6	TRW	680.767	712.977	815.842	841.445
7	ITIL	1252.796	1370.184	1410.400	1637.548
8	ISO	702.154	1179.483	1557.616	2185.207

be successfully instantiated by his or her authorization policy. The efficiency of the proposed approach exploits the fact that the eligible scenarios (resulting from execution and authorization constraints) can be computed once and reused with every authorization policy. In this way, multiple changes to a policy, which are well-known to be costly [25], become much less problematic to handle and customers can even explore and evaluate the suitability of variants of a policy. This is in sharp contrast to the approach discussed in Section 3 (after Example 3.1) that consists of re-invoking an available algorithm for solving the WSP on every task-user pair in a scenario. To illustrate, consider the instance at line 4 of Table 2. Recall that the off-line step for ITIL takes around 3.5 seconds and observe that this is computed once and for all. If, instead, we use the technique to solve the WSP in [6] as a black-box (i.e. without being able to retrieve the symbolic reachability graph computed during the off-line phase), which is common to (almost) all techniques available in the literature, solving the same B-SFP would require almost 30 seconds resulting from re-computing 7 times (corresponding to the 7 task-user pairs in the returned scenario) the same symbolic reachability graph (compare this with the timing of 4.561 seconds reported in the table). This is a significant performance gain despite the small size of the example.

For each workflow, the times shown in Table 3 grow with the number of users ($|U|$) and desired resiliency (k) for two reasons. First, there are more absent subsets to be considered. Second, finding a scenario for each subset takes longer, since the time to answer Datalog queries depends on the number of users. For a combination of many users and a large resiliency, the time taken may be unacceptable, which is inherent to the complexity of the problem. In any case, the observation above about reusing the pre-computed symbolic reachability graph is even more important for the SkR-SFP. If the graph was not reused, it would have to be computed from scratch for each user-task pair of each absent subset. For the SkR-SFP, the optimization based on absent set domination is crucial for acceptable performance. As an example, in line 2, column '$k = 2$', the reported time (10.815) is obtained considering only 3 dominant subsets of users, out of a total of $\binom{9}{2} = 36$. If the 36 subsets are considered, the total time is 107.724. This difference is even larger for instances with more users or a larger k. Another advantage of Algorithm 3 is that it stops as soon as a concrete scenario is not found for a subset (line 6 in the algorithm). Notice that the times in column '$k = 8$' of lines 0 and 1 are less than the time in columns '$k = 2$', '$k = 4$', and '$k = 6$' of the same lines, since those instances are not 8-resilient. The instance in column '$k = 8$' of line 3 is also not resilient, but the algorithm took much longer to encounter a subset with no concrete scenario.

6. Discussion

We have introduced four SFPs, discussed their relationships with the WSP, and argued that solving them supports the deployment of business processes in the activity of model reuse. Then, we have described algorithms to solve the four SFPs, based on a previously proposed technique [6] for the synthesis of monitors for the WSP. An experimental evaluation on real-world examples has shown that our techniques can be used in practice at deployment time since they perform the computationally heaviest part (namely, computing the set of eligible scenarios) once and for all when the workflow is designed (and possibly added to a library of business processes) and reuse it with arbitrary authorization policies.

6.1. Related work

WSP and Scenario Finding. Bertino et al. [5] were the first to present a method capable of computing execution scenarios by using logic programming techniques. The practical feasibility of the approach was not assessed as we did for our technique in Section 5. Kohler and Schaad [21] introduced the notion of

policy deadlocks (corresponding to situations in which the WSP is unsolvable) and proposed a graph-based technique to compute minimal user bases to help policy designers to avoid such situations. There are some similarities with our approach such as the use of symbolic users, but our work is not limited to RBAC policies as theirs and focuses on business process reuse, which was not considered in [21]. Solworth [35] used an approvability graph to describe sequences of actions defining the termination of a workflow. His technique focused on linear workflows, whereas we support constructs for parallel executions and conditionals.

Many other works provided solutions to the WSP. Crampton [10] considered workflows as partial orders and showed an algorithm to determine whether there is an execution scenario. Wang and Li [37] reduced the WSP to SAT and showed that it is NP-hard even in the presence of simple constraints. Crampton et al. [15] improved the complexity bounds for the WSP and generalized the constraints supported by their solution. Crampton et al. [16] used model checking on a fragment of linear temporal logic to compute execution scenarios, minimal user bases, and a safe bound on resiliency. Yang et al. [31] studied the complexity of the WSP with different control-flow patterns and showed that, in general, the problem is intractable. Basin et al. [4] defined and solved the problem of choosing authorization policies that allow a successful workflow execution and an optimal balance between system protection and user empowerment. As discussed above, most of these solutions cannot be used to solve the SFPs without an unacceptable decrease in performances because they are not able to pre-compute the set of eligible scenarios.

The works in [3,8] separated between an off-line and on-line phase as done in this work and [6] but it was not exploited for business process reuse as we did.

Resiliency. Li et al. [24] introduced the notion of resiliency policies in the context of access control systems, i.e. policies that require the access control system to be resilient to the absence of users. Their work was inspired by the need of forming teams that, even in emergency situations, are able to access all resources in a common pool. They also defined the Resiliency Checking Problem (RCP), which amounts to checking if an access control state satisfies a given resiliency policy. Wang and Li [37] then studied resiliency in workflow systems by identifying three types: *static*, when users can become absent only before the beginning of the execution of a workflow and stay available during execution; *decremental*, when users may become absent before or during the workflow instance execution but they cannot become available until the workflow execution is terminated; and *dynamic*, when users can become absent and available at any given time before and during the execution of the workflow. In [37], Wang and Li studied the relationship between these notions of resiliency and the WSP and showed that checking static workflow resiliency is in NP, whereas checking decremental and dynamic resiliency is in PSPACE. The authors of [37] observed that there are other possible formulations of resiliency that can be of interest. For instance, Mace et al. [26] defined *quantitative* workflow resiliency, whereby a user wants to know how likely a workflow instance is to terminate. They used specific user availability models based on Markov Decision Processes (MDPs) in which probabilities measure the likelihood that users are available or absent. Mace et al. proposed a method to solve the problem of finding optimal execution scenarios guaranteeing the termination of workflows. Additionally, in [29], the same authors discussed the effects of alternative execution paths on resiliency. This leads to the possibility of a different resiliency value for each path. They defined resiliency variance as a metric to indicate volatility from the resiliency average, claiming that a higher variance increases the likelihood of workflow failure. User availability models were discussed in more details in [27], categorized into non-deterministic, probabilistic and bounded and with several encodings for the PRISM (probabilistic) model checker.[4] The same authors also studied the impact

[4] `http://www.prismmodelchecker.org/`

of policy design on workflow resiliency computation time in [28], observed that adding or removing authorization constraints has a substantial impact, and described a way to compute the set of security constraints that can be added to a policy in order to reduce the computation time while maintaining resiliency. Crampton et al. [13] studied the Bi-Objective WSP, which is the problem of minimizing two weight functions associated to a valid execution scenario. In particular, they considered the following pair of weight functions: one representing the violation of constraints and one representing the violation of the authorization policy. This problem does not have a single solution, but a set of solutions known as a Pareto front, which cannot be compared, and users are left with the burden of choosing the most suitable. Crampton et al. also related this problem to workflow resiliency, claiming that Mace et al.'s translation to MDP is not necessary since the same metrics can be computed by constructing a graph where the nodes are partial valid execution scenarios, and the edges, connecting successive scenarios, are labeled with the probability of a user being available to execute the next task (checking every possible partial scenarios has exponential-time complexity). The Bi-Objective WSP is a generalization of the Valued WSP [12], which has as single objective minimizing the sum of both weights.

Crampton et al. [14] reduced the RCP to the WSP and showed how to solve it using a fixed-parameter tractable algorithm for the latter. The RCP differs from the problem of workflow resiliency by considering more parameters (see [37] for details). On the other hand, the RCP is always static, whereas workflow resiliency can be static, decremental, or dynamic. The basic solution in the original paper about RCP [24] is to enumerate all subsets of s users and check for satisfiability (using a solution for $s = 0$ as a black-box), but the authors presented a pruning strategy based on set domination, to have a more efficient solution. Crampton et al. [14] solved the same RCP problem, using the same pruning strategy and then using their FPT algorithm as this black-box to decide satisfiability (by translating the resources in P to a workflow). The solution of Section 4 is similar, with the difference that we invoke the available monitor for solving the WSP as a black-box algorithm. The authors of [14] also mentioned that this basic reduction cannot be applied directly to decremental or dynamic workflow resiliency, but they point to their work on Valued WSP [12] as a possibility to do it, by using weights to represent the availability of users.

6.2. Future work

As a first step, we intend to study how to support other forms of resiliency (namely, decremental and dynamic) by extending our approach. We also intend to design new heuristics to support other classes of constraints efficiently in the solution to the SkR-SFP. Afterwards, we identify the following ideas to stimulate further work on the topic of scenario finding techniques to support the deployment of security-sensitive workflows.

Tool integration. Currently, the implementation used to produce the experimental results reported in Section 5 uses a command-line interface where users can run each of the problems on a given workflow specification and policy file. We have developed a tool to generate enforcement monitors for workflow management systems and store them in a repository (along with the reachability graph). The tool was integrated with one of SAP's platforms for BPM [9]. We would like to integrate the solution to the SFPs into this tool by providing a graphical user interface allowing users to retrieve the workflow and associated reachability graph from the repository, specify a policy, select the desired problems and visualize the solutions in an intuitive way.

Policy synthesis. Another interesting problem to consider is how to automatically synthesize, or suggest changes to existing, authorization policies so that solutions of an SFP are optimal with respect to some criteria, e.g., least privilege [20] or cost/risk [4]. This is related to the Bi-Objective WSP introduced

in [13] and briefly discussed above. Using our approach, weights could be associated with each edge in the reachability graph and then instantiate symbolic users in all possible ways (i.e. according to injective functions) so that standard graph search algorithms for finding minimum weight paths can compute optimal solutions. Unfortunately, this naive approach does not scale to medium or large number n of users as the injective functions to be considered grow as the factorial of n. It would thus be interesting to study situations in which the graph can be pruned to achieve sub-optimal solutions.

Run-time adaptation. Two assumptions are very common in the workflow satisfiability and resiliency literature. First, that the control-flow and authorization constraints do not change at run-time. Second, that only humans can be absent during the execution of a workflow, whereas external applications for automated tasks are always available. These are abstractions from real-world executions and we intend to investigate how to solve the SFPs on weaker versions of these assumptions. Considering arbitrary changes to control-flow at run-time is not feasible, but we would like to allow "controlled" deviations, as in [1] and pre-compute eligible scenarios for each of the allowed deviations. User availability models [27] could be adapted to model the availability of external applications and consider resiliency to other kinds of failures.

References

[1] A. Adriansyah, B. van Dongen, and N. Zannone. Controlling break-the-glass through alignment. *ASE Science Journal*, 2(4):198–212, 2013.

[2] A. Armando and S.E. Ponta. Model Checking of Security-sensitive Business Processes. In *Proc. of FAST*. Springer, 2009.

[3] D. Basin, S.J. Burri, and G. Karjoth. Obstruction-free authorization enforcement: Aligning security with business objectives. In *Proc. of CSF*. IEEE, 2011.

[4] D. Basin, S.J. Burri, and G. Karjoth. Optimal workflow-aware authorizations. In *Proc. of SACMAT*. ACM, 2012.

[5] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *TISSEC*, 2:65–104, 1999.

[6] C. Bertolissi, D.R. dos Santos, and S. Ranise. Automated synthesis of run-time monitors to enforce authorization policies in business processes. In *Proc. of ASIACCS*. ACM, 2015.

[7] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE TKDE*, 1(1):146–166, 1989.

[8] D. Cohen, J. Crampton, A. Gagarin, G. Gutin, and M. Jones. Iterative plan construction for the workflow satisfiability problem. *JAIR*, 51:555–577, 2014.

[9] L. Compagna, D.R. dos Santos, S.E. Ponta, and S. Ranise. Cerberus: Automated synthesis of enforcement mechanisms for security-sensitive business processes. In *Proc. of TACAS*. Springer, 2016.

[10] J. Crampton. A reference monitor for workflow systems with constrained task execution. In *Proc. of SACMAT*. ACM, 2005.

[11] J. Crampton, R. Crowston, G. Gutin, M. Jones, and M.S. Ramanujan. Fixed-parameter tractability of workflow satisfiability in the presence of seniority constraints. In *Proc. of FAW-AAIM*. Springer, 2013.

[12] J. Crampton, G. Gutin, and D. Karapetyan. Valued workflow satisfiability problem. In *Proc. of SACMAT*. ACM, 2015.

[13] J. Crampton, G. Gutin, D. Karapetyan, and R. Watrigant. The bi-objective workflow satisfiability problem and workflow resiliency. *JCS*, Preprint, 2016.

[14] J. Crampton, G. Gutin, and R. Watrigant. Resiliency policies in access control revisited. In *Proc. of SACMAT*. ACM, 2016.

[15] J. Crampton, G. Gutin, and A. Yeo. On the parameterized complexity of the workflow satisfiability problem. In *Proc. of CCS*. ACM, 2012.

[16] J. Crampton, M. Huth, and J. Kuo. Authorized workflow schemas: deciding realizability through LTL(F) model checking. *STTT*, 16(1):31–48, 2014.

[17] J. de Freitas. Model business processes for flexibility and re-use: A component-oriented approach. Technical report, IBM, 2009.

[18] R. Dijkman, M. La Rosa, and H.A. Reijers. Editorial: Managing large collections of business process models-current techniques and challenges. *CI*, 63(2):91–97, 2012.

[19] D.R. dos Santos, S. Ranise, L. Compagna, and S.E. Ponta. Assisting the deployment of security-sensitive workflows by finding execution scenarios. In *Proc. of DBSec*. Springer, 2015.

[20] H. Huang, F. Shang, J. Liu, and H. Du. Handling least privilege problem and role mining in rbac. *Journal of Combinatorial Optimization*, 30(1):63–86, 2013.

[21] M. Kohler and A. Schaad. Avoiding policy-based deadlocks in business processes. In *Proc. of ARES*. IEEE, 2008.

[22] A. Koschmider, M. Fellmann, A. Schoknecht, and A. Oberweis. Analysis of process model reuse: Where are we now, where should we go from here? *Decision Support Systems*, 66(0):9–19, 2014.
[23] N. Li and J.C. Mitchell. Datalog with constraints: a foundation for trust management languages. In *Proc. of PADL*, 2003.
[24] N. Li, Q. Wang, and M. Tripunitara. Resiliency policies in access control. *TISSEC*, 12(4):20:1–20:34, April 2009.
[25] H. Lu, Y. Hong, Y. Yang, Y. Fang, and L. Duan. Dynamic workflow adjustment with security constraints. In *Proc. of DBSec*. Springer, 2014.
[26] J.C. Mace, C. Morisset, and A. van Moorsel. Quantitative workflow resiliency. In *Proc. of ESORICS*. Springer, 2014.
[27] J.C. Mace, C. Morisset, and A. van Moorsel. Modelling user availability in workflow resiliency analysis. In *Proc. of HotSoS*. ACM, 2015.
[28] J.C. Mace, C. Morisset, and A. van Moorsel. Proc. of qest. In *Impact of Policy Design on Workflow Resiliency Computation Time*. Springer, 2015.
[29] J.C. Mace, C. Morisset, and A. van Moorsel. Resiliency variance in workflows with choice. In *Proc. of SERENE*. Springer, 2015.
[30] OMG. Business Process Model and Notation (BPMN), Version 2.0. Technical report, Object Management Group, 2011.
[31] I. Ray P. Yang, X. Xie and S. Lu. Satisfiability analysis of workflows with control-flow patterns and authorization constraints. *IEEE TSC*, 99, 2013.
[32] R. Sandhu, V. Bhamidipati, and Q. Munawer. The arbac97 model for role-based administration of roles. *TISSEC*, 2(1):105–135, February 1999.
[33] R. Sandhu, E. Coyne, H. Feinstein, and C. Youmann. Role-Based Access Control Models. *IEEE Computer*, 2(29):38–47, 1996.
[34] A.U. Shankar. An Introduction to Assertional Reasoning for Concurrent Systems. *ACM Comput. Surv.*, 25(3):225–262, September 1993.
[35] J.A. Solworth. Approvability. In *Proc. of ASIACCS*. ACM, 2006.
[36] W.M.P. van der Aalst. Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, 2013, 2013.
[37] Q. Wang and N. Li. Satisfiability and resiliency in workflow authorization systems. *TISSEC*, 13:40:1–40:35, December 2010.
[38] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
[39] N.H. Zaaboub, L. Makni, and H.B. Abdallah. Literature review of reuse in business process modeling. *Software & Systems Modeling*, 13(3):975–989, 2014.