Assisting the Deployment of Security-Sensitive Workflows by Finding Execution Scenarios*

Daniel R. dos Santos[1,2,3], Silvio Ranise[1], Luca Compagna[2], Serena E. Ponta[2]

[1] Fondazione Bruno Kessler (FBK) [2]SAP Labs France [3]University of Trento

Abstract. To support the re-use of business process models, an emerging trend in Business Process Management, it is crucial to assist customers during deployment. We study how to do this for an important class of business processes, called security-sensitive workflows, in which execution constraints on the tasks are complemented with authorization constraints (e.g., Separation of Duty) and authorization policies (constraining which users can execute which tasks). We identify the capability of solving Scenario Finding Problems (SFPs), i.e. finding concrete execution scenarios, as crucial in supporting the re-use of security-sensitive workflows. Solutions of SFPs provide evidence that the business process model can be successfully executed under the policy adopted by the customer. We present a technique for solving two SFPs and validate it on real-world business process models taken from an on-line library.

1 Introduction

Organizations rely on Business Process Management (BPM) [22] 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* [1], 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).

Example 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 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

* This work has been partly supported by the EU under grant 317387 SECENTIS (FP7-PEOPLE-2012-ITN).

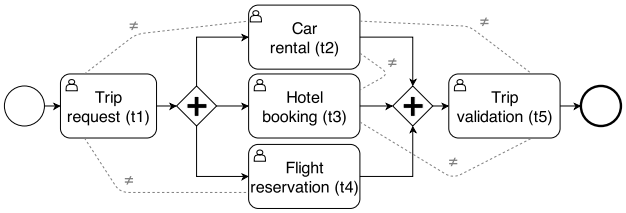

Fig. 1. TRW in (extended) BPM notation

frauds, five authorization constraints—called Separation of Duty (SoD) in the literature; see, e.g., [8]—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 [15] 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 $\neq$. □

One of the most important problems for security-sensitive workflows is the Workflow Satisfiability Problem (WSP) [8], 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., [8–10, 21]) have provided solutions to the WSP, which are becoming less and less satisfactory because of the recent trend in BPM of collecting and re-using large numbers of business process models [19, 11, 23]. For instance, the SAP HANA Workflow[1] is a BPMN-based solution that allows for the creation of business process models (templates) that can be deployed and operated in different contexts. At deployment time, what is crucial for customers re-using 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 scrutinize concrete *execution scenarios* showing 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.

Example 2. A simple situation in which the TRW in Example 1 can be deployed is a tiny organization with a set $U = \{a, b, c\}$ of three users and the following authorization policy $TA = \{(a, t1), (b, t1), (a, t2), (b, t2), (c, t2), (a, t3), (b, t3), (c, t3), (a, t4), (a, t5), (b, t5), (c, t5)\}$, where $(u, t) \in TA$ means that user u is entitled to

[1] http://help.sap.com/saphelp_hana_opint/SAP_OPInt_Developers_Guide.pdf

execute task t. 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: $\eta = t1(b), t3(c), t4(a), t2(a), t5(b)$ where $t(u)$ means that user u has executed task t and the position in the sequence corresponds to the order in which the tasks have been executed (i.e. $t1$ has been executed first, $t5$ last, $t3$ after $t1$ but before $t4$, $t2$, and $t5$, etc). 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). □

Among all possible scenarios permitting a workflow to terminate, customers may be particularly interested in those that can be executed by a smallest possible set of users, called minimal user base in the literature [10]. This knowledge would enable organizations to assess the likelihood of emergencies or extraordinary situations due to, e.g., employee absences.

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 they are not able to exploit the fact that execution and authorization constraints are fixed and only the authorization policy changes at deployment time. (WSP is NP-hard already in the presence of one SoD constraint [21].)

The **main contributions** of this paper are three. First, we give precise statements of two SFPs together with a discussion of their relationships with the WSP (Section 2). Second, we describe techniques to solve them by adapting the technique for the synthesis of run-time monitors for the WSP developed in [5] (Section 3). Third, we validate our solutions on real-world examples from a library of re-usable business process models (Section 4). We use the TRW in Example 1 as a running example to illustrate the main notions in the paper.

2 From the WSP to SFPs

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 $U \times T$. Intuitively, $(u,t) \in TA$ 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., [9]. 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. There are various ways to specify security-sensitive workflows. For instance, [9] introduces the notion of "constrained workflow authorization schema" as a tuple $(T,U,\leq,TA,C)$, where $\leq$ is a partial order over T and the other components are as above. Then, it defines an "execution schedule" as a tuple $(t_1,...,t_k)$ of tasks such that $\{t_1,...,t_k\} = T$ and $t_j \not\leq t_i$ for each $1 \leq i < j \leq k$ and a "valid plan" π as a mapping from T to U such that $(t,\pi(t)) \in TA$ and $(\pi(t_1),\pi(t_2)) \in \rho$ for each constraint (t_1,t_2,ρ) in C. Given an execution schedule and $(t_1,...,t_k)$ and a valid plan π of a constrained workflow authorization schema $(T,U,\leq,TA,C)$, it is easy to derive an authorized and eligible scenario $t_1(\pi(t_1)),...,t_k(\pi(t_k))$ of the security-sensitive workflow $W(T,U)$ according to TA and C.

Definition 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.*

Recall Example 2 in Section 1 for an instance of this problem and a solution.

2.1 Scenario Finding Problems

In the context of business process reuse, it is possible to compute—once and for all—the set E of eligible scenarios associated to a security-sensitive workflow in a library (we will describe how to compute and compactly represent this set in Section 3 below). The problem is then to look for those scenarios in E with some properties when an authorization policy becomes available.

Definition 2 (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. 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 $W(U, T)$, a set C of authorization constraints, an authorization policy TA, and $\eta_e = t(u), t'(u'), ...$ an eligible scenario in E, we can re-use 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 $W(U, T)$, 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 $W(U, T)$, 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 [21] 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 [6]. 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), 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 3.1.

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 [10], 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 (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 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. 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 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 4 (compare the timings for solving SFPs with those for MUB-SFPs in Table 2).

3 From Solving the WSP to Solving SFPs

We now briefly recall the technique in [5] to synthesize run-time monitors solving the WSP. This is important 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 with the specification of an authorization policy, such as the relation TA of Example 2) and consists of two steps.

Off-line step. A symbolic transition system S is automatically derived (in a way similar to that described in [20]) from the task execution constraints of a workflow $W(T, U)$ and the set C of authorization constraints (notice that TA is not yet taken into consideration). S is used to compute a *symbolic reachability graph* RG, i.e. a directed graph whose edges are labeled by task-user pairs in which users are symbolically represented by variables (called *user variables*) and whose nodes are labeled 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). 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 (according to C) execution scenario $\eta_c = \mu(\eta_s) = t_1(\mu(\upsilon_{j_1})), ..., t_n(\mu(\upsilon_{j_n}))$ 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 5. An excerpt of the symbolic reachability graph for the TRW is depicted in Figure 2 where a task-user pair $t(\upsilon_k)$ labeling an edge is abbreviated by $t(k)$ for the sake of compactness. 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 2) represents all those execution scenarios in which a symbolic user identified by υ_3 first performs task $t1$ followed by t_3, 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 scenario $\eta_c = \mu(\eta_s)$ is eligible according to the set C of SoD constraints shown in Figure 1. □

On-line step. A non-recursive Datalog program M with negation is derived from the symbolic reachability graph RG by building a clause of the form $can\_do(\upsilon, t) \leftarrow \beta_v$ where β_v is the formula labeling node v in RG. (For more details on how M is built, the interested reader is pointed to [5]). The formula β_v contains invocations to the binary predicates $auth$ and h. The former is the interface to the authorization policy and such that $auth(u, t)$ holds iff $(u, t) \in TA$ while the latter keeps track of which user has executed which task, i.e. $h(t, u)$

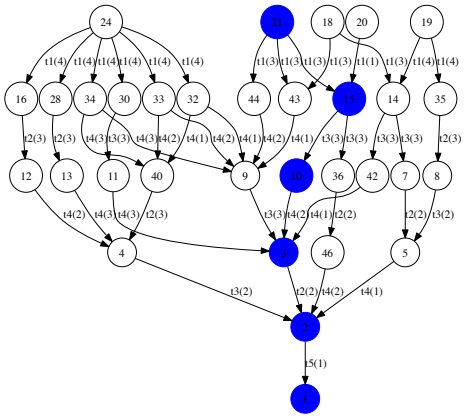

Fig. 2. An excerpt of the symbolic reachability graph for the TRW

#	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

means that t has been executed by u. Following an established tradition (see, e.g., [13]) claiming that (variants of) Datalog are adequate to express a wide range of access control policy idioms, we assume *auth* to be defined by a Datalog program P. Instead, 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.

Example 6. For the TRW, let us consider again the set of users and the authorization policy discussed in Example 2. The relation *TA* can be specified after the Role Base Access Control (RBAC) model [17] by the Datalog program P:

$$
\begin{array}{l}
ua(a, r1).\ ua(a, r2).\ ua(a, r3).\ ua(b, r2).\ ua(b, r3).\ ua(c, r2). \\
pa(r_3, t1).\ pa(r_2, t2).\ pa(r_2, t3).\ pa(r_1, t4).\ pa(r_2, t5). \\
auth(\upsilon, \tau) \ \leftarrow \ ua(\upsilon, \rho) \wedge pa(\rho, \tau).
\end{array}
$$

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 *auth* is defined as the join of the relations ua and pa (cf. Datalog clause in the last line). Recall the definition of *TA* in Example 2 and notice that $P \vdash auth(u, t)$ 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 2 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 scenario 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 scenario $t1(\upsilon_1), t3(\upsilon_3), t4(\upsilon_2), t2(\upsilon_2), t5(\upsilon_1)$ in the graph of Figure 2 (cf. the path with the blue nodes; see also Example 5) by applying the injective function μ mapping υ_1 to b, υ_2 to a, and υ_3 to c. □

3.1 Solving the SFPs

In order to solve B-SFPs and MUB-SFPs (recall Definitions 2 and 3), 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 problems. For TA, we have already assumed (see paragraph **On-line step** in previous section) the availability of a Datalog program P defining the binary predicate *auth*. 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 through an example.

Example 7. Let us consider the TRW with a set U of 6 users. The graph in Figure 2 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 later. The main idea underlying the algorithm is to adapt a standard Depth-First Search (DFS) algorithm 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 [5]. 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

Algorithm 1. Solving the B-SFP

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

```
 1: for all v ∈ Nodes(RG) do visited[v] ← false;
 2: end for
 3: η ← ϵ; NI ← NoIncoming(RG);
 4: while (v ∈ NI and η = ϵ) do
 5:     η ← DFS(v, ϵ, Γ); NI ← NI \ {v};
 6: end while
 7: return η
 8: function DFS(v, η, H)
 9:     visited[v] ← true; OE ← OutGoing(v);
10:     if OE = ∅ then return η
11:     else
12:         for all v --t(v)--> w ∈ OE do
13:             if (not visited[w] and M, P, H ⊢^{v↦u} can_do(t, v)) then
14:                 return DFS(w, append(η, t(u)), H ∪ {h(t, u)})
15:             end if
16:         end for
17:     end if
18:     return ϵ
19: end function
```

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 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 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.

An interesting extension of the algorithm is provided by considering a set Γ of facts, which 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 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 are either true or false. Again, it is possible to add the facts encoding the truth or falsity of the condition to Γ in order to drive the algorithm and find scenarios with such conditions. The flexibility provided by Γ is illustrated in Section 4 below.

It is possible to modify Algorithm 1 following the idea discussed after Example 4 in order to solve the MUB-SFP. This requires to avoid returning the authorized scenario as soon as we find one (line 10) 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. Also the search of this modified algorithm can be driven by a set Γ of facts as it was the case for the algorithm solving B-SFP.

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 [6]. It is easy to see that we invoke $O(n + m)$ times the Datalog engine (at most) in line 13 for both the

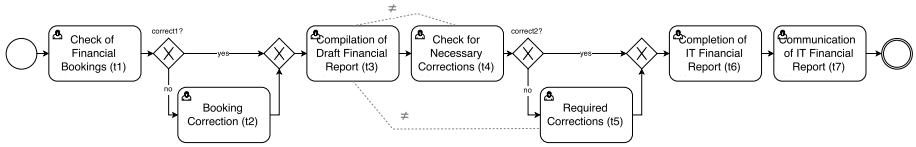

Fig. 3. ITIL 2011—IT Financial Reporting (abbreviated ITIL)

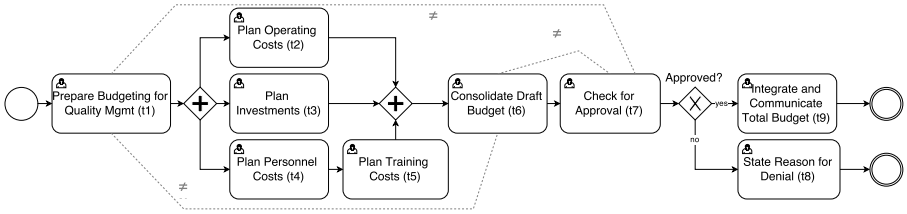

Fig. 4. ISO9000—Budgeting for Quality Management (abbreviated ISO)

Algorithm 1 and its modified version described above for solving the MUB-SFP. This is much better than the upper bounds discussed in Section 3.1. To see this, consider the situation in Example 7: $\ell_{max} = 5$ 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 Validation of the Technique

We consider two real-world examples, shown in Figures 3 and 4, derived from business processes available in an on-line library provided by Signavio,[2] which contains more than 120 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 quality requirements. 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

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

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 3 and 4) 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 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 $U = \{u1, ..., u9\}$ of users. For the TRW, we consider two policies P_0 and P_1: the former is that in Example 6 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 executing our techniques for solving 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 3) from [5]. For the TRW, the symbolic reachability graph is computed in around a second and contains 46 nodes with 81 edges. For the ITIL, the graph is computed in around 3.5 seconds and has 78 nodes with 72 edges. For the 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 Algorithm 1 for solving B-SFPs and its modification for solving MUB-SFPs (described towards the end of Section 3.1) in Python v2.7.9. The invocation to the Datalog engine at line 13 in Algorithm 1 is implemented with the Datalog engine pyDatalog v0.15.2. Table 2 shows the findings of our experiments. 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) 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).

Discussion. Our experiments indicate that the SFPs introduced in this paper together with Algorithm 1 (and its modification for the MUB-SFP) fit well with emerging BPM practices for re-use. Whenever a customer wants to deploy a business process by re-using a workflow template, some SFP is solved (if possible) to provide him/her with an authorized scenario showing that a template business process can be successfully instantiated by his/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 re-used with every authorization policy. In this way, multiple changes to a policy, which are well-known to be costly [14], become much less problematic to

Table 2. Experiments

#	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	$\{c1, \textbf{not } c2\}$	$t1(u3), t3(u3), t4(u7), t5(u8), t6(u3), t7(u7)$	4.454
3	ITIL+P_2	$\{\textbf{not } c1, c2\}$	$t1(u3), t2(u1), t3(u9), t4(u8), t6(u9), t7(u9)$	4.374
4	ITIL+P_2	$\{\textbf{not } c1, \textbf{not } c2\}$	$t1(u3), t2(u1), t3(u3), t4(u7),$ $t5(u8), t6(u3), t7(u7)$	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	$\{\textbf{not } appr\}$	$t1(u3), t4(u7), t5(u8), t2(u3), t3(u7),$ $t6(u7), t7(u8), t8(u6)$	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	$\{c1, \textbf{not } c2\}$	$t1(u3), t3(u3), t4(u7), t5(u7), t6(u3), t7(u3)$	116.525
14	ITIL+P_2	$\{\textbf{not } c1, c2\}$	$t1(u1), t2(u1), t3(u1), t4(u7), t6(u1), t7(u1)$	108.827
15	ITIL+P_2	$\{\textbf{not } c1, \textbf{not } c2\}$	$t1(u3), t2(u3), t3(u3), t4(u7),$ $t5(u7), t6(u3), t7(u3)$	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	$\{\textbf{not } appr\}$	$t1(u5), t3(u5), t2(u5), t4(u5), t5(u5),$ $t6(u9), t7(u6), t8(u9)$	166.644

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 2.1 (after Example 3) 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 [5] 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.

5 Conclusions

We have introduced two SFPs, discussed their relationships with the WSP, and argued that solving them supports the deployment of business processes in the activity of model reuse. We have also described algorithms to solve two SFPs, based on a previously proposed technique [5] for the WSP. An experimental evaluation on two real-world examples shows 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 added to a library and re-use it for any possible authorization policy.
Related work. Bertino et al. [4] were the first to present, among many other contributions, a method capable of computing execution scenarios by using logic programming techniques. The practical feasibility of the approach is not assessed as we do for our technique in Section 4. Kohler and Schaad [12] introduces the notion of policy deadlocks (corresponding to situations in which the WSP is unsolvable) and propose a graph-based technique to compute minimal user bases to help policy designers avoid such situations. There are some similarities with our approach (e.g., the use of symbolic users) but our work is not limited to RBAC policies as theirs and focuses on business process reuse, which is not considered in [12]. Solworth [18] uses an approvability graph to describe sequences of actions defining the termination of a workflow. His technique focus on linear workflows whereas we support constructs for parallel executions and conditionals. Many works [8, 21, 16, 2, 3, 9, 7, 10, 14] study the WSP. As discussed above, most of them 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 [2, 7] separate between an off-line and on-line phase as done in [5] and here but do not exploit it for business process reuse as we do.
Future work. We intend to study the notion of resiliency [21] in SFPs and how to automatically suggest changes to authorization policies so that solutions of an SFP are optimal with respect to some criteria, e.g., least privilege.

References

1. A. Armando and S. E. Ponta. Model Checking of Security-sensitive Business Processes. In *Proc. of FAST*, 2009.
2. D. Basin, S. J. Burri, and G. Karjoth. Obstruction-free authorization enforcement: Aligning security with business objectives. In *CSF'11*, pages 99–113. IEEE, 2011.
3. D. Basin, S. J. Burri, and G. Karjoth. Optimal workflow-aware authorizations. In *Proc. of SACMAT '12*, pages 93–102, New York, NY, 2012. ACM.
4. E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *TISSeC*, 2:65–104, 1999.
5. C. Bertolissi, D. R. dos Santos, and S. Ranise. Automated synthesis of run-time monitors to enforce authorization policies in business processes. In *ASIACCS 2015*, USA, 2015. ACM.
6. 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.
7. D. Cohen, J. Crampton, A. Gagarin, G. Gutin, and M. Jones. Iterative plan construction for the workflow satisfiability problem. *JAIR*, 51:555–577, 2014.
8. J. Crampton. A reference monitor for workflow systems with constrained task execution. In *10th ACM SACMAT*, pages 38–47. ACM, 2005.
9. J. Crampton, G. Gutin, and A. Yeo. On the parameterized complexity of the workflow satisfiability problem. In *CCS'12*, pages 857–868. ACM, 2012.
10. J. Crampton, M. Huth, and J. Kuo. Authorized workflow schemas: deciding realizability through LTL(F) model checking. *STTT*, 16(1):31–48, 2014.
11. 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.
12. M. Kohler and A. Schaad. Avoiding policy-based deadlocks in business processes. In *ARES'08*, pages 709–716. IEEE, 2008.
13. N. Li and J. C. Mitchell. Datalog with constraints: a foundation for trust management languages. In *PADL'03*, pages 58–73, 2003.
14. H. Lu, Y. Hong, Y. Yang, Y. Fang, and L. Duan. Dynamic workflow adjustment with security constraints. In *DBSeC*, volume 8566 of *LNCS*, pages 211–226. 2014.
15. OMG. Business Process Model and Notation (BPMN), Version 2.0. Technical report, Object Management Group, 2011.
16. I. Ray P. Yang, X. Xie and S. Lu. Satisfiability analysis of workflows with control-flow patterns and authorization constraints. *IEEE TSC*, 99, 2013.
17. R. Sandhu, E. Coyne, H. Feinstein, and C. Youmann. Role-Based Access Control Models. *IEEE Computer*, 2(29):38–47, 1996.
18. J. A. Solworth. Approvability. In *Proc. of ASIACCS 2006*, pages 231–242, New York, NY, USA, 2006. ACM.
19. W. M. P. van der Aalst. Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, 2013, 2013.
20. W.M.P. van der Aalst and A. H. M. Ter Hofstede. Yawl: Yet another workflow language. *Inf. Systems*, 30:245–275, 2003.
21. Q. Wang and N. Li. Satisfiability and resiliency in workflow authorization systems. *TISSeC*, 13:40:1–40:35, December 2010.
22. M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
23. N. Zaaboub Haddar, L. Makni, and H. Ben Abdallah. Literature review of reuse in business process modeling. *Software & Systems Modeling*, 13(3):975–989, 2014.