COMPUTING SCIENCE

Quantitative Workflow Resiliency

John C. Mace, Charles Morisset and Aad van Moorsel

TECHNICAL REPORT SERIES

No. CS-TR-1467 May 2015

TECHNICAL REPORT SERIES

No. CS-TR-1467 May, 2015

Quantitative Workflow Resiliency

J. C. Mace, C. Morisset and A. Moorsel

Abstract

A workflow is resilient when the unavailability of some users does not force to choose between a violation of the security policy or an early termination of the workflow. Although checking for the resiliency of a workflow is a well-studied problem, solutions usually only provide a binary answer to the problem, leaving a workflow designer with little help when the workflow is not resilient. We propose in this paper to provide instead a measure of quantitative resiliency, indicating how much a workflow is likely to terminate for a given security policy and a given user availability model. We define this notion by encoding the resiliency problem as a decision problem, reducing the finding of an optimal user-task assignment to that of solving a Markov Decision Process. We illustrate the flexibility of our encoding by considering different measures of resiliency, and we empirically analyse them, showing the existence of a trade-off between multiple aspects such as success rate, expected termination step and computation time, thus providing a toolbox that could help a workflow designer to improve or fix a workflow.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

MACE, J. C; MORISSET, C; MOORSEL, A;
Quantitative Workflow Resiliency
[By] J. C. Mace, C. Morisset, A. Moorsel
Newcastle upon Tyne: Newcastle University: Computing Science, 2015.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1467)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1467

Abstract

A workflow is resilient when the unavailability of some users does not force to choose between a violation of the security policy or an early termination of the workflow. Although checking for the resiliency of a workflow is a well-studied problem, solutions usually only provide a binary answer to the problem, leaving a workflow designer with little help when the workflow is not resilient. We propose in this paper to provide instead a measure of quantitative resiliency, indicating how much a workflow is likely to terminate for a given security policy and a given user availability model. We define this notion by encoding the resiliency problem as a decision problem, reducing the finding of an optimal user-task assignment to that of solving a Markov Decision Process. We illustrate the flexibility of our encoding by considering different measures of resiliency, and we empirically analyse them, showing the existence of a trade-off between multiple aspects such as success rate, expected termination step and computation time, thus providing a toolbox that could help a workflow designer to improve or fix a workflow.

About the authors

John Mace is a research assistant at Newcastle University, working with Charles Morisset and Aad van Moorsel on devising new tools and methodologies to analyse the impact of information security using, in particular security ontology development tools and more formal techniques to quantify security impact on workflows. John completed a BSc (Hons) in Computing Science at Newcastle University in 2010 during which time he received a Scott Logic prize for computer excellence for his year two performance. He was also awarded the School of Computing Science prize for best overall performance in year three. John is due to submit his PhD thesis mid-2015.

Charles Morisset is a Senior Research Associate at Newcastle University, working with Aad van Moorsel on quantitative aspects of security, in particular in the decision making process and in access control mechanisms. Charles received is PhD from Universite Pierre et Marie Curie - Paris VI in France in 2007, on the topic of formalisation of access control systems. He then worked from 2007 to 2009 at the United Nations University, in Macau SAR, China, on formal methods for software engineering, after which he joined the Information Security Group at Royal Holloway, University of London, to work on risk-based access control until 2011. From 2011 to 2013, he worked at the Istituto di Informatica e Telematica in Pisa, Italy, on formal methods and access control, and he joined the Centre for Cybercrime and Computer Security at Newcastle University in 2013.

Aad van Moorsel is a Professor in Distributed Systems and Head of School at the School of Computing Science in Newcastle University. His group conducts research in security, privacy and trust. Almost all of the group's research contains elements of quantification, be it through system measurement, predictive modelling or on-line adaptation. Aad worked in industry from 1996 until 2003, first as a researcher at Bell Labs/Lucent Technologies in Murray Hill and then as a research manager at Hewlett-Packard Labs in Palo Alto, both in the United States. He got his PhD in computer science from Universiteit Twente in The Netherlands (1993) and has a Masters in mathematics from Universiteit Leiden, also in The Netherlands. After finishing his PhD he was a postdoc at the University of Illinois at Urbana-Champaign, Illinois, USA, for two years. Aad became the Head of the School of Computing Science in 2012.

Suggested keywords

SATISFIABILITY PROBLEM
MARKOV DECISION PROCESS
QUANTITATIVE ANALYSIS

Quantitative Workflow Resiliency

John C. Mace, Charles Morisset, and Aad van Moorsel

Centre for Cybercrime & Computer Security,
Newcastle University, Newcastle upon Tyne,
NE1 7RU, United Kingdom
{john.mace,charles.morisset,aad.vanmoorsel}@ncl.ac.uk

Abstract. A workflow is resilient when the unavailability of some users does not force to choose between a violation of the security policy or an early termination of the workflow. Although checking for the resiliency of a workflow is a well-studied problem, solutions usually only provide a binary answer to the problem, leaving a workflow designer with little help when the workflow is not resilient. We propose in this paper to provide instead a measure of *quantitative resiliency*, indicating how much a workflow is likely to terminate for a given security policy and a given user availability model. We define this notion by encoding the resiliency problem as a decision problem, reducing the finding of an optimal user-task assignment to that of solving a Markov Decision Process. We illustrate the flexibility of our encoding by considering different measures of resiliency, and we empirically analyse them, showing the existence of a trade-off between multiple aspects such as success rate, expected termination step and computation time, thus providing a toolbox that could help a workflow designer to improve or fix a workflow.

Keywords: Workflow Satisfiability Problem, Markov Decision Process, Quantitative Analysis

1 Introduction

A workflow is the automation of a business process comprising tasks and predicate conditions defining their partial order [1]. Ensuring all workflow instances complete means assigning each task to a user in accordance with business rules specifying when and by whom workflow data may be accessed and modified. From a security perspective, access management ensures users with the correct clearance and capabilities are matched with appropriate tasks while reducing the threat of collusion and fraud. Each user-task assignment may have to satisfy many different kind of security constraints [6, 7, 14], the three most common being: *i*) the user must be authorised to perform the task; *ii*) if the task is related to another task through a *binding of duty*, then the same user should perform both tasks; *iii*) if the task is related to another task through a *separation of duty*, then the same user cannot perform both tasks.

The Workflow Satisfiability Problem (WSP) [9, 23] therefore consists in finding a user-task assignment ensuring both the termination of all instances and the

non-violation of the security constraints, especially in highly dynamic environments, subject to unpredictable events such as user unavailability. This can be an issue for dynamic workflow management systems whose choices made for early tasks can constrict later assignments [16]. In extreme cases, bad assignments will remove all assignment options for an upcoming task and block a workflow from completing [3, 15]. It is therefore important these workflows can be appraised before enactment via suitable tools and useable metrics that aid workflow configuration and formulation of contingency plans, such as *resiliency* [23], which checks whether the unavailability of some users has an impact on satisfiability.

Most existing approaches, e.g. [11, 10, 13, 21, 23], address the WSP from a computational point-of-view, by finding the most efficient algorithm to compute a suitable assignment, which either return a correct assignment if it exists, or nothing. In practice finding such an assignment can be demanding and often unmanageable, especially when facing unforeseen and emergency situations. For example, despite the provision of guidelines stipulating many public service staffing levels (e.g. [20]), high sickness rates, budget cuts, staff shortage, increased workloads and unpredictability all contribute to critical workflows often being attempted without enough available users.

Taking a binary approach therefore fails to address a number of real-life issues where the ideal case is not always reachable. Indeed, declaring a workflow to be either resilient or not may be of little practical use to a workflow designer. Of course, a satisfiable workflow is always better than an unsatisfiable one but a workflow where all but one instance can be correctly assigned provides on average, a better service than one where no instance can be assigned. In addition, if both workflows terminate early instead of violating the policy, they are both better than a workflow violating the policy.

In this paper we take the stance to provide a workflow designer with quantitative measures, indicating a degree of satisfaction and/or resiliency for a given workflow, instead of simply returning an assignment if one exists. In order to do so, we propose to model the WSP as a decision problem, in order to benefit from the extensive collections of tools related to the discipline. More precisely, the contributions of this paper are as follows:

- We first encode the workflow satisfaction problem as finding the optimal solution of a Markov Decision Process (MDP);
- We then encode the *decremental resiliency* problem [23] as an extension of the above one, by modelling user availability in the state of the MDP;
- We illustrate the flexibility of our model by showing how a simple change of the reward function can move the focus from the normal termination rate of the workflow to the expected termination step, and we show on a simple use case that both approaches are in general incomparable, and that in general, finding the optimal assignment requires addressing a trade-off between multiple aspects, including computation time.

Is is important to note that the focus of this paper is not to provide a particularly efficient solution to the WSP, but to propose a novel approach of this problem.

We however believe our approach paves the way to defining an efficient solution by using the extensive literature dedicated to the efficient solving of an MDP.

The rest of this paper is as follows. Section 2 gives a brief overview of related work while Section 3 revisits the workflow satisfiability problem and defines it as an MDP. In Section 4 we define quantitative measures for workflow satisfaction and resiliency. An assessment of our approach is given in Section 5 and concluding remarks in Section 6.

2 Related Work

A number of previous studies on workflow resiliency appear in the literature. Wang et al. took a first step in [23] to quantify resiliency by addressing the problem of whether a workflow can still complete in the absence of users and defined a workflow as k resilient to all failures of up to k users across an entire workflow. Lowalekar et al. show in [17] multiple assignments may provide the same level of k resiliency and give a technique for selecting the most favourable using security attributes that minimize the diffusion of business knowledge across all users.

Basin et al. in [3, 4] allow the reallocation of roles to users thus overcoming workflow blocks created by user failure. This is feasible in certain business domains but may have limited application in public service workflows where roles are more specialised; a nurse cannot step in for a doctor for example. Wainer et al. consider in [22] the explicit overriding of security constraints in workflows, by defining a notion of privilege. Similarly, Bakkali [2] suggests introducing resiliency through delegation and the placement of criticality values over workflows. Delegates are chosen on their suitability but may lack competence; this is considered the 'price to pay' for resiliency. As delegation takes place at a task level it is not currently clear whether a workflow can still complete while meeting security constraints.

A more practical approach is presented by Mace et al. [18] and more formally by Watson [24] who discuss the practicalities of assigning workflows, or parts of workflow across multiple cloud-based platforms while ensuring security constraints are met. Discussion is given on the various trade-offs that must be considered including performance and security risks. Current literature does not address the issue of workflows that must execute but may not be satisfiable nor resilient in every instance. Neither does it provide quantitative measures to analyse and optimise workflows in such cases, which is the focus of this paper.

3 The Workflow Satisfiability Problem

In a nutshell, the Workflow Satisfiability Problem (WSP) [23] consists in assigning each user to a task for a given workflow such that all security constraints are met. Whereas existing work mostly considers WSP as a constraint solving problem (for instance as the graph k-colouring problem when the workflow contains separation of duties constraints [23, 4]), we propose to consider it as a

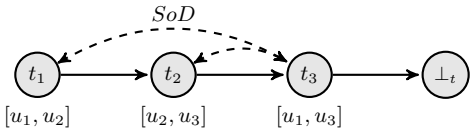

Fig. 1: Running example

	a_1	a_2	a_3	a_4
t_1	u_1	u_2	u_2	u_2
t_2	u_2	u_2	u_2	u_3
t_3	u_3	u_1	u_3	u_1

Table 1: Workflow assignments

decision problem[1], thus modelling the decision made to assign a user to a task in a given context. Hence, we define below an encoding of the WSP as solving a Markov Decision Process (MDP) [5]. We first propose a definition of workflow, which, although driven by the MDP encoding, is general enough to consider complex workflows. After briefly recalling the notion of MDP, we then present the corresponding MDP encoding and the definition of the WSP.

3.1 Workflow

As described in Section 2, there exist several definitions of workflow in the literature. They commonly define a set of users $\mathcal{U}$ and a set of tasks $\mathcal{T}$, structured to indicate which sequences of tasks can be executed, for instance with a partial ordering over tasks. For the sake of generality, we consider a *task manager*, which is a function $\tau : \mathcal{T} \times \mathcal{U} \to \mathcal{P}(\mathcal{T})$[2]. In other words, given a task t and a user u, $\tau(t, u)$ is a probability function over tasks indicating the probability of each task to be the next one.

The set of tasks contains an initial task $t_0 \in \mathcal{T}$, and in order to model the fact that a workflow can finish we consider a special task $\perp_t \in \mathcal{T}$, such that, given any user u, $\tau(\perp_t, u) = \perp_t$[3]

Running example. *As a running example to illustrate the different concepts presented here, we consider the workflow shown in Figure 1 where* $\mathcal{T} = \{t_1, t_2, t_3\}$ *and where the only possible sequence is* $t_1; t_2; t_3$*. Hence, the initial task is* t_1*, and the task manager is defined as, for any user* u*,* $\tau(t_1, u) = t_2$*,* $\tau(t_2, u) = t_3$*, and* $\tau(t_3, u) = \perp_t$*.*

The second common aspect of workflows across existing definitions is to define a set of users $\mathcal{U}$ coming with a security policy over the set of tasks $\mathcal{T}$, including basic user permissions, separations and bindings of duties, expressed as sets of constraints. Hence, we consider security policies of the form $p = (P, S, B)$ where

- $P \subseteq U \times T$ are *user-task permissions*, such that $(u, t) \in P$ if, and only if u is allowed to perform t;

[1] It is worth noting that in the end, both perspectives can join, for instance by solving the decision problem using Linear Programming.

[2] Given a set X, we write $\mathcal{P}(X)$ for the set of functions $f : X \to [0, 1]$, such that $\sum_{x \in X} f(x) = 1$.

[3] For the sake of simplicity, we write $f(x) = y$ whenever $f(x, y) = 1$, where $f(x) \in \mathcal{P}(Y)$, for some Y. In this case, we say that $f(x)$ is *deterministic*.

– $S \subseteq \wp(T)$[4] are *separations of duty*, such that $\{t_1, \ldots, t_n\} \in S$ if, and only if each user assigned to t_i is distinct;
– $B \subseteq \wp(T)$ are *bindings of duty*, such that $\{t_1, \ldots, t_n\} \in B$ if, and only if the same user is assigned to all t_i;

Running example. *We now consider a set of users* $U = \{u_1, u_2, u_3, u_4\}$ *and a security policy* $p_1 = (P_1, S_1, B_1)$ *that states:*

– $P_1 = \{(u_1, t_1), (u_2, t_1), (u_2, t_2), (u_3, t_2), (u_1, t_3), (u_3, t_3)\}$
– $S_1 = \{\{t_1, t_3\}, \{t_2, t_3\}\}$
– $B_1 = \emptyset$

Figure 1 illustrates this security policy, where a dotted arrow signifies a constraint given in p *between the tasks* t *and* t'*. A label* $[u_m, ..., u_n]$ *states the users that are authorised by* P *to execute* t*.*

A workflow therefore consists of both a set of tasks, with a task manager and a set of users, with a security policy.

Definition 1 (Workflow). *A workflow is a tuple* $W = (\mathcal{U}, \mathcal{T}, \tau, t_0, p)$*, where* $\mathcal{U}$ *is a set of users,* $\mathcal{T}$ *is a set of tasks,* τ *is a task manager,* t_0 *is the initial task and* p *is a security policy.*

3.2 Workflow assignment

A *user-task assignment* is a relation $UA \subseteq \mathcal{U} \times \mathcal{T}$, associating each t_i with some u_i. Given a policy $p = (P, S, B)$, UA satisfies p, and in this case, we write $UA \vdash p$, if, and only if the three following conditions are met:

$$UA \subseteq P \tag{1}$$

$$\forall s \in S \, \forall u \in \mathcal{U} \quad |\{t \in s \mid (u, t) \in UA\}| \leq 1 \tag{2}$$

$$\forall b \in B \quad |\{u \in \mathcal{U} \mid \exists t \in b \, (u, t) \in UA\}| \leq 1 \tag{3}$$

In our running example, Table 1 provides all workflow assignments satisfying p_1, such that each a_i is represented as a function from task to users. For instance, a_2 assigns t_1 and t_2 to u_2 and t_3 to u_1. An *instance* of a workflow is a sequence of tasks $(t_1, \ldots, t_n)$ such that $\tau(t_i, u, t_{i+1}) \neq 0$, for any i and any user u. Informally, the WSP consists of defining a relation UA such that, for any instance of a workflow, the restriction of UA to tasks in this instance satisfies the policy of the workflow.

In some cases, solving the WSP can be relatively simple. For instance, consider a policy where $S = B = \emptyset$, i.e., where there are no separations or bindings of duty. In this case, it is enough to assign each task t with a user u such that $(u, t) \in P$, and if there is no such user, then the workflow is unsatisfiable. However, the enforcement of separations and bindings of duty might require to keep track of the previous assignments. For instance, in our running example, u_1 can only be assigned to t_3 if it has not been assigned neither to t_2 nor to t_1.

[4] We use $\wp(X)$ to denote the set of finite subsets of X.

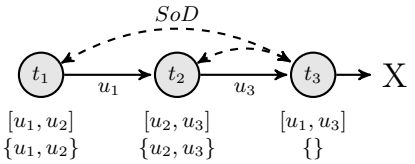

Fig. 2: The task t_3 cannot be executed.

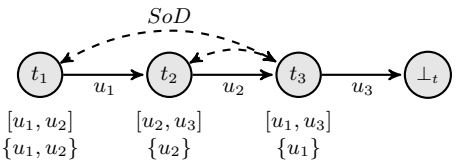

Fig. 3: The workflow terminates correctly.

3.3 Contextual Assignment

As illustrated above, in order to ensure that the security constraints are met, the user-task assignment needs to take into account at least the previous assignments. We call *context* any dynamic information relevant to user-task assignments, such as the security policy, the history of execution, or the user failures. Here again, we aim for generality, and given a workflow $W = (\mathcal{U}, \mathcal{T}, \tau, t_0, p)$, we consider a set $\mathcal{C}$ of contexts, with a *context manager* $\gamma : \mathcal{C} \times \mathcal{T} \times \mathcal{U} \to \mathcal{P}(\mathcal{C})$, such that given a context c, a task t and a user u, $\gamma(c, t, u)$ represents the probability space of the next context. We write $D = (C, \gamma, c_0)$ for a context description, which includes a set of contexts, a context manager and initial context $c_0 \in \mathcal{C}$.

For instance, in order to define the WSP, the context needs to contain all previous assignments, in order to check the validity of separations and bindings of duty at each step. Hence we define $\mathcal{C}_h = \wp(\mathcal{U} \times \mathcal{T})$, such that for any $c \in \mathcal{C}_h$, all $(u, t) \in c$ correspond to previous assignments. We then define the context manager γ_h as, for any context $c \in \mathcal{C}_h$, any task t and any user u, $\gamma_h(c, t, u) = c \cup \{(u, t)\}$. The assignment (u, t) is permitted if it satisfies p when combined with the previous assignments contained in c. Thus, for any context $c \in \mathcal{C}_h$ we write $c, u, t \vdash p$ if, and only if, $c \cup \{(u, t)\} \vdash p$. We write $D_h = (\mathcal{C}_h, \gamma_h, \emptyset)$ for the context description corresponding to previous assignments. In the following, we assume the sets $\mathcal{U}$ and $\mathcal{T}$ to be clear from context when using D_h, unless stated otherwise.

Definition 2 (Assignment). *Given a workflow $W = (\mathcal{U}, \mathcal{T}, \tau, t_0, p)$ and a context description $D = (\mathcal{C}, \gamma, c_0)$, a* contextual assignment *is a function $\delta : \mathcal{C} \times \mathcal{T} \to \mathcal{U}$, such that $\delta(c, t)$ represents the user assigned to t in the context c.*

For instance, with the context description D_h, a simple contextual assignment is to return any user that can execute the task, taking previous assignments into account. We define the set of all permitted users $PU_{c,t} = \{u \mid c, u, t \vdash p\}$. The on-the-fly assignment is then the function $\delta_o(c, t)$ returning any user from $PU_{c,t}$, if it is not empty, and any user otherwise (meaning that no user can execute t in the context c without violating the workflow policy).

As illustrated on Figure 2, where $\{u_m, ..., u_n\}$ denotes $PU_{c,t}$, δ_o might not select the best possible assignment. For instance if u_1 is assigned to t_1 and u_3 is assigned to t_2, which are both correct assignments in their respective context, then the separation of duty constraints make it impossible to assign t_3 to any user. However, as shown on Figure 3, if u_2 is assigned to t_2 instead, then u_3 can

be assigned to t_3. We present in the next section the encoding of the workflow as an MDP, which aims at defining an assignment avoiding the above pitfall.

3.4 Markov Decision Process

In order to define the optimal contextual assignment, we encode the notion of assignment into a Markov Decision Process (MDP) [5], which is a stochastic process where the transition from one state to another is governed both probabilistically and by a decision made by a policy. Each transition is associated with a reward, and solving an MDP consists in defining a policy maximising the expected reward collected by the process. More precisely, an MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathbf{p}, \mathbf{r})$ where:

- $\mathcal{S}$ is a set of states, describing the possible configurations of the system;
- $\mathcal{A}$ is a set of actions, describing how to go from one state to another;
- $\mathbf{p} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a transition function, such that $\mathbf{p}^a_{ss'}$ describes[5] the probability of reaching s' from s when executing the action a;
- $\mathbf{r} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a reward function, such that $\mathbf{r}^a_{ss'}$ describes the reward associated with execution a from the state s and reaching s'.

A *policy* for an MDP (which should not be confused with the security policy of a workflow) is a function $\delta : \mathcal{S} \to \mathcal{A}$, i.e., associating each state with an action, and the *value* of a policy for an MDP is given as:

$$V^\delta(s) = \sum_{s' \in \mathcal{S}} \mathbf{p}^{\delta(s)}_{ss'} \mathbf{r}^{\delta(s)}_{ss'} + \beta \sum_{s' \in \mathcal{S}} \mathbf{p}^{\delta(s)}_{ss'} V^\delta(s')$$

where $0 \leq \beta < 1$ is a discount factor, giving more or less weight to "future" values. The *optimal* policy is then defined as:

$$\delta^*(s) = arg \max_{a \in \mathcal{A}} \left[\sum_{s' \in \mathcal{S}} \mathbf{p}^a_{ss'} \mathbf{r}^a_{ss'} + \beta \sum_{s' \in \mathcal{S}} \mathbf{p}^a_{ss'} V^*(s') \right] \tag{4}$$

where V^* is the value function of δ^*. Note that since $\beta < 1$, the optimal policy is always defined, even when $s = s'$. It is possible to show that $V^*(s) \geq V^{\delta'}(s)$, for any other policy δ' and any state s, and we refer to [5] for further details about the proof of this property and further details on the notion of MDP.

In order to reduce the WSP to solving an MDP, we combine three different elements: a workflow, a context and a reward function, the latter expressing the metric we are interested in measuring. Note that this encoding is loosely inspired by the MDP encoding of access control mechanisms proposed in [19].

Definition 3 (MDP Encoding). *Given a workflow $W = (\mathcal{T}, \mathcal{U}, \tau, t_0, p)$, a context description $D = (\mathcal{C}, \gamma, c_0)$ and a reward function $\mathbf{r} : (\mathcal{C} \times \mathcal{T}) \times \mathcal{U} \times (\mathcal{C} \times \mathcal{T}) \to \mathbb{R}$, we write $\mathsf{MDP}[W, D, \mathbf{r}]$ for the MDP defined by the tuple $(\mathcal{C} \times \mathcal{T}, \mathcal{U}, \mathbf{p}_{\gamma,\tau}, \mathbf{r})$, where given any pairs $(c, t), (c', t') \in \mathcal{C} \times \mathcal{T}$ and any user $u \in \mathcal{U}$:*

$$\mathbf{p}_{\gamma,\tau}((c, t), u, (c', t')) = \gamma(c, t, u, c') \cdot \tau(t, u, t')$$

[5] For the sake of conciseness, we write $\mathbf{p}^a_{ss'}$ for $\mathbf{p}(s, a, s')$ when no confusion can arise.

A policy for $\mathsf{MDP}[W, D, \mathbf{r}]$ is then a function $\delta : \mathcal{C} \times \mathcal{T} \to \mathcal{U}$, that is, a contextual assignment. In other words, the optimal policy δ^* is the optimal contextual assignment for the workflow, the context description and the reward function. Since we focus on the non-violation of the security policy, we define the reward function associating each violation with $-\infty$ for the context description D_h:

$$\mathbf{r}_p((c,t),u,(c',t')) = \begin{cases} -\infty & \text{if } t \neq \bot_t \text{ and } c,u,t \not\vdash p \\ 0 & \text{otherwise.} \end{cases}$$

where we assume that $-\infty * 0 = 0$, meaning that an $-\infty$ reward on a transition that cannot happen has no effect on the overall value function. Note that by construction, we know that transitions starting from $\bot_t$ can only finish at $\bot_t$, so we do not need to measure such transitions.

Definition 4 (WSP). *Given a workflow $W = (\mathcal{U}, \mathcal{T}, \tau, t_0, p)$, we write V_h^* for the optimal value function of $\mathsf{MDP}[W, D_h, \mathbf{r}_p]$, and we say that W is satisfiable if, and only if, $V_h^*(\emptyset, t_0) = 0$.*

In the following, we usually write δ_c for the optimal policy of $\mathsf{MDP}[W, D_h, \mathbf{r}_p]$, and Table 1 actually presents all possible instance for δ_c in the running example. It is easy to see that this definition of the WSP matches the informal one given above: the optimal policy δ^* avoids any transition reachable from $(\emptyset, t_0)$ with a reward of $-\infty$, i.e., any transition that would violate the policy. So, as long as there is a possible assignment that allows the workflow to reach the task $\bot_t$ without violating the policy, the optimal policy will select it. It is also worth observing that this definition is binary: either $V^*(\emptyset, t_0) = 0$ or $V^*(\emptyset, t_0) = -\infty$. We generalise it in Section 4, since the objective of this model is to go beyond binary satisfaction and resiliency.

3.5 Implementation of the optimal policy

Solving an MDP is in general an intractable problem [5, 8], because the optimal value function must be calculated on every possible state (and the WSP is shown to be NP-complete [23]). Hence, we do not aim here to present an *efficient* solution to solve this problem, and we refer to e.g. [12] for recent work on the complexity of solving WSP.

Calculating all possible future states is equivalent to traversing a tree of all possible assignment paths outgoing from the current state. To help visualise this concept Figure 4 depicts an *assignment tree* where each complete path is equivalent to a valid workflow assignment given in Table 1. A node c in the tree represents a context such that c_i at level j represents a state (c_i, t_j). A leaf node is the workflow finish point $\bot_t$. All outgoing edges from c_i at level j define the set of users PU_{c_i,t_j} from which one is selected. Essentially, δ_c ensures all assignments are made within the bounds of an assignment tree composed of *assignment paths* which all finish with $\bot_t$. It follows that in any state, any user selected from $PU_{c,t}$ will allow the workflow to complete. We present in Section 5 an implementation of the optimal policy using Value Iteration under simplifying assumptions.

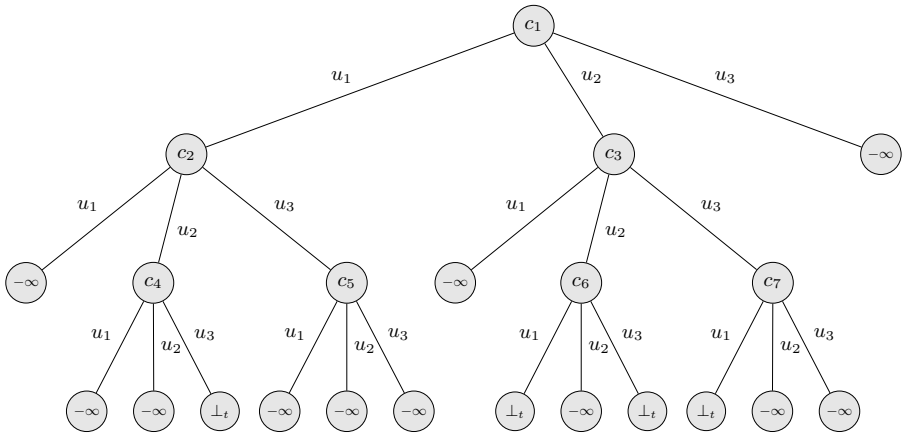

Fig. 4: Workflow assignment tree

4 Quantitative Analyses

4.1 Quantitative Satisfaction

In general, the fact alone that a workflow is unsatisfiable is, as such, of little help for a system designer. Consider for instance a workflow where all but one instance can be correctly assigned, and another where no instance can be correctly assigned. Both workflows are unsatisfiable, however, on average, the first one provides a better service than the second one. In addition, if both workflows terminate early instead of violating the policy, they are both better than a workflow violating the policy. Of course, a satisfiable workflow is always better than an unsatisfiable one, but as said in the Introduction, we aim at providing tools and quantitative measures for concrete situations, where the ideal case is not always reachable.

In order to model the early termination of a workflow, given a workflow $W = (\mathcal{U}, \mathcal{T}, \tau, t_0, p)$ with the context D_h, we introduce a special user $\perp_u \in \mathcal{U}$, such that $\tau(t, \perp_u) = \perp_t$, for any task t. In order to reward successful termination we provide a positive reward for such successful completion, and we associate a null reward for the early termination:

$$\mathbf{r}_s((c,t),u,(c',t')) = \begin{cases} -\infty & \text{if } t \neq \perp_t,\ u \neq \perp_u \text{ and } c,u,t \not\models p \\ 1 & \text{if } t \neq \perp_t,\ u \neq \perp_u \text{ and } t' = \perp_t \\ 0 & \text{otherwise.} \end{cases}$$

We are then able to provide a probabilistic statement about satisfiability.

Definition 5. *Given a workflow $W = (\mathcal{U}, \mathcal{T}, \tau, t_0, p)$ we define the quantitative satisfaction of W by $V_s^*(\emptyset, t_0)$, where V_s^* is the optimal value function of* $\mathsf{MDP}[W, D_h, \mathbf{r}_s]$.

The quantitative satisfaction of a workflow is either $-\infty$, if the workflow is not satisfiable, or a number between 0 and 1, indicating the probability of the workflow to finish, based on the probabilistic task manager. In particular, it is easy to prove the following proposition, following a similar reasoning to that matching Definition 4 with the informal description of the WSP.

Proposition 1. *Given a workflow W, W is satisfiable if, and only if its quantitative satisfaction is equivalent to 1.*

Note that we cannot define the quantitative satisfaction to be *equal* to 1, which is only possible if $\beta = 1$, which is forbidden, by definition. In practice, if there is no infinite loop in the MDP (as it is the case in Section 5), this factor can be equal to 1. The proof of Proposition 1 is quite straight-forward: in order to obtain $V_s^*(\emptyset, t_0) = 1$, the optimal policy must be able to assign each task to a user without violating the policy nor terminating early.

4.2 Quantitative Resiliency

Wang and Li define in [23] resiliency as a "property of those system configurations that can satisfy the workflow even with absence of some users". As described in the Introduction, there are indeed multiple scenarios where users can fail at some point, thus not being able to execute an assigned task. Hence, a user-task assignment might need to take into account such failures.

Several levels of resiliency are introduced in [23]: static resiliency, where users can only fail before the start of the workflow; decremental resiliency, where users can fail during the workflow, and cannot become available again; and dynamic resiliency, where users can fail and later become available during the workflow.

Let us first observe that the notion of static resiliency does not require any special encoding, since checking for it can be done by directly checking the WSP for the workflow without the failing users. We now focus on the decremental and dynamic resiliency. We consider the set of contexts $\mathcal{C}_{f,N} = \{f \subseteq \mathcal{U} \mid |f| \leq N\}$, where each context $c \in \mathcal{C}_{f,N}$ corresponds to a set of at most N users not available. For any user u and task t, an assignment (u, t) satisfies p if u is available, hence for any context $c \in \mathcal{C}_{f,N}$, we write $c, u, t \vdash p$ if, and only if, $u \notin c.f$. For the sake of simplicity, we assume that each user has the same probability of failing (although this could clearly be easily generalised), and given a context c, the set of all possible next contexts is defined as:

$$nf_N(c) = \{c \cup f \mid f \subseteq (\mathcal{U} \setminus c) \wedge |c \cup f| \leq N\}$$

In particular, when the size of c is already equal to N, then $nf_N(c) = \{c\}$. We can then define the probability of reaching a new context c' from a context c from the user failure perspective:

$$\gamma_{f,N}(c, t, u, c') = \begin{cases} |nf_N(c)|^{-1} & \text{if } c' \in nf_N(c), \\ 0 & \text{otherwise.} \end{cases}$$

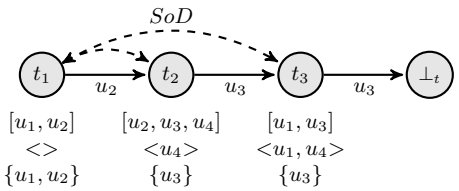

Fig. 5: δ_s - optimal resiliency

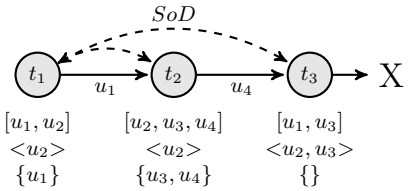

Fig. 6: δ_s - sub-optimal distance

We write $D_{f,N} = (\mathcal{C}_{f,N}, \gamma_{f,N}, \emptyset)$ for the context description corresponding to the decremental and equiprobable failure of up to N users, and we write $D_{h,f,N} = D_h \times D_{f,N}$ for the cartesian product of this context description and the one modelling previous assignments, where the context manager is defined in a pointwise way, and reward functions applies to the relevant components of a tuple[6]. For any context $c \in \mathcal{C}_{h,f,N}$, user u and task t, an assignment (u, t) satisfies p, and we write $c, u, t \vdash p$, if, and only if, $c \cup \{(u, t)\} \vdash p \wedge u \notin c.f$.

Proposition 2. *A workflow is decrementally resilient up to N users if and only if $\sum_{c \in \mathcal{C}_{f,N}} V^*_{f,N}((\emptyset, c), t_0) \simeq 1$, where $V^*_{f,N}$ is the optimal value function of $\mathsf{MDP}[W, D_{h,f,N}, \mathbf{r}_s]$.*

In the following, we usually write δ_s for the optimal policy of $\mathsf{MDP}[W, D_{h,f,N}, \mathbf{r}_s]$. Note that we sum over all possible contexts, because the notion of decremental resiliency also considers that users can fail before the start of the execution of the workflow. In addition, dynamic resiliency can be encoded similarly by defining $nf_N(c) = \mathcal{C}_{f,N}$, and we therefore focus only in the rest of the paper on decremental resiliency, unless specified otherwise.

Running example. *Consider now a different security policy $p_2 = (P_2, S_2, B_2)$, where $P_2 = \{(u_1, t_1), (u_2, t_1), (u_2, t_2), (u_3, t_2), (u_4, t_2), (u_1, t_3), (u_3, t_3)\}$, $B_2 = \emptyset$ and $S_2 = \{\{t_1, t_2\}, \{t_1, t_3\}\}$. A label <$u_i$> denotes u_i has failed. Figure 5 illustrates how δ_s can maximise the quantitative resiliency of the running example for two failed users. Assigning t_1 to u_2 generates two assignment options for t_2. In turn, selecting u_3 for the assignment of t_2 ensures two assignment options for t_3. Despite the failure of u_4 at t_2 and u_1 at t_3, the workflow can terminate.*

Figure 6 illustrates how the workflow can still fail under δ_s due to an unavoidable block caused by two failed users. User u_1 must be assigned t_1 following the failure of u_2. Two assignment options are available at t_2 from which either can be chosen but due to the failure of u_3 at t_3, the workflow is blocked.

4.3 Expected Distance

The quantitative satisfaction of a workflow denotes the probability of a workflow to terminate, taking the context into account. This metric therefore does not

[6] In this paper, the only examples of reward functions we consider are defined either on the violation of the policy or the (early) termination. However, in general, we could have more complex reward functions, depending for instance on user availability.

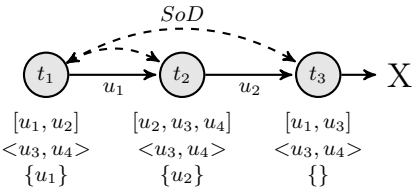

Fig. 7: δ_d - optimal distance

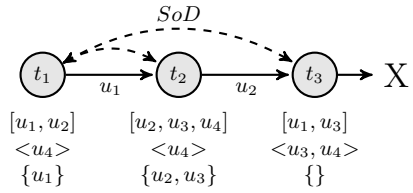

Fig. 8: δ_d - sub-optimal resiliency

differentiate between an instance terminating at the first task and an instance terminating at the penultimate task. In order to illustrate the flexibility of our model, we define the expected distance of the workflow, i.e., the number of tasks performed before terminating, using the following reward function, defined over the context description D_h:

$$\mathbf{r}_d((c,t),u,(c',t')) = \begin{cases} -\infty & \text{if } t \neq \bot_t,\ u \neq \bot_u \text{ and } c,u,t \not\models p \\ 1 & \text{if } t \neq \bot_t \text{ and } u \neq \bot_u, \\ 0 & \text{otherwise.} \end{cases}$$

The expected distance of the workflow can therefore be calculated with $V_d^*(\emptyset, t_0)$, where V_d^* is the optimal value function of $\mathsf{MDP}[W, D_h, \mathbf{r}_d]$. We can also redefine the notion of resiliency to measure the expected distance instead of the success rate with the optimal value function of $\mathsf{MDP}[W, D_{h,f,N}, \mathbf{r}_d]$, and we usually write δ_d for the optimal policy of this model. Interestingly, an assignment optimal for the notion of resiliency as defined in Section 4.2 is not necessarily optimal for this notion of resiliency, as illustrated in Section 5. This reinforces our motivation for building the model presented here, which can provide several metrics and thus help a system designer to improve or fix a workflow, rather than a simple boolean indicating whether the workflow is satisfiable or not.

Running example. *Figure 7 illustrates how δ_d can optimize the expected distance of our running example for two failed users. Following the failure of u_3 and u_4 at t_1, u_1 is assigned t_1 to generate the assignment option u_2 at t_2. Task t_2 can be assigned before the workflow blocks at t_3. In contrast, δ_s would always finish at t_1 while δ_o and δ_c would assign either u_1 or u_2 at t_1 resulting in the workflow finishing at either t_2 or t_3. Figure 8 illustrates how δ_d can lower the expected distance for two failed users. The failure of u_4 means t_1 must be assigned to u_1 to optimise the distance-resiliency at that point. The failure of u_3 at t_3 means the workflow is now blocked. However, under δ_s, t_1 would be assigned to t_2 which reduces the assignment options for t_2 to just u_3 but increases the options for the final task t_3 to u_1 and u_3. It follows that the workflow would complete with this particular failure under δ_s.*

Algorithm 1 Value Iteration for the optimal value function, where $c.h$ refers to the history of previous assignments in the context c.

function $Q^*(c, t, u, \beta, \overline{R})$
  if $t = \bot_t$ **then** R_S
  else if $c, u, t \not\vdash p$ **then** R_V
  else
    $t' \leftarrow \tau(t, u)$
    if $u = \bot_u$ **then** 0
    else $R_T + \beta * \max_{u \in \mathcal{U}} \left[|nf_N(c)|^{-1} * \sum_{i=1}^{|nf_N(c)|} \{Q^*(c_i, t', u, \beta, \overline{R}) \mid c_i \in nf_N(c)\} \right]$
    end if
  end if
end function

5 Assessment

In this section, we give an empirical assessment of the different policies introduced in Sections 3 and 4. More precisely, given a uniform distribution of user failure, we are able to generate resiliency metrics for a workflow and show the average success rate appears higher using the resiliency assignments δ_s and δ_d.

Implementation We solve the MDP defined in Section 3 using value iteration [5] and implement a simplified version of the optimal policy function δ^* under the assumption that user failures are equiprobable and workflow behaviour is linear, i.e., no loops or branches exist. Given a context c, a task t, a discount factor β and a reward vector $\overline{R} = (R_S, R_T, R_V)$, corresponding to the atomic rewards for successfully terminating, doing one step and violating the policy, respectively, the optimal policy is given as:

$$\delta^*(c, t, \beta, \overline{R}) = \arg\max_{u \in \mathcal{U}} Q^*(c, t, u, \beta, \overline{R})$$

where Q^* is defined in Algorithm 1. We are now in position to define the policies introduced in Section 3 and Section 4. Given $c \in \mathcal{C}_h$, $c' \in \mathcal{C}_{h,f,N}$ and $t \in \mathcal{T}$

$$\delta_o(c, t) = \delta^*(c, t, 0, 1, 0, -\infty) \qquad \delta_c(c, t) = \delta^*(c, t, 1, 1, 0, -\infty)$$
$$\delta_s(c', t) = \delta^*(c', t, 1, 1, 0, -\infty) \qquad \delta_d(c', t) = \delta^*(c', t, 1, 0, 1, -\infty)$$

Note that since we assume the workflow is linear, we can safely assign 1 to β. As δ_c and δ_s concern themselves with optimising workflow satisfaction it is possible to carry out a degree of pre-processing before runtime. Assignments with reward $-\infty$ can be removed offline as they clearly do not contribute to the satisfiability of a workflow. All correct assignments are therefore cached in a tree data structure. Any unavailable users are removed before selecting a user for the current assignment. If multiple users are found to be optimal by any of the four policies, one is simply selected at random.

Table 2: Test results for Example 1

	δ_o	δ_c	δ_s	δ_d
Success rate	0.136	0.769	0.803	0.793
Expected distance (tasks)	5.20	8.83	8.67	9.29
Computation time (μs)	579.06	139.20	1.35×10^5	2.94×10^6

To model user failure, all possible user failures are generated up to N as a set of failure vectors. The test program takes as input parameters a sequence of tasks, a security policy, a set of users, a single failure vector and an assignment policy. Before assigning each task, the failure vector is checked and users removed as appropriate. A call is made to the given assignment policy for each task in turn and the result logged in an assignment history. The program terminates if the end of the workflow is reached or no assignment can be made and outputs the assignment history and assignment time which is the aggregation of computation time captured through a benchmarking library. For instance, we define Example 1 to contain 10 tasks, 6 users and a security policy consisting 10 separation and 2 binding of duty constraints. We consider 1 user failure per run giving 61 failure vectors in all, each run 10 times. The testing was done on a computing platform incorporating a 2.40Ghz i5 Intel processor and 4GB RAM.

Results A total of 610 runs were recorded per policy. It should be noted that 150 correct assignments exist for the workflow instance in Example 1. The recorded test data has been analysed and the primary results presented in Table 2. To aid understanding, the sample data is also presented in graphical form. Figure 9 shows the probability of assignment for each task in Example 1. For example, the probability of the workflow to execute until at least t_3 using δ_o is 0.49 while under δ_c the probability is 0.95. The workflow success rate is equivalent to the probability of reaching t_{10}. Figure 10 gives average computation time (excluding pre-computed data, such as the assignment tree). For example, the average time to compute assignments up to and including t_4 under δ_o is 111μs while under δ_c the time is 17.2μs.

Discussion Table 3 summarises the different characteristics of the different policies. As expected, δ_s generates assignments providing the highest success rate, and δ_d generates assignments giving the highest expected distance. Intuitively δ_s reserves users for the final task assignments and δ_d reserves users to ensure every task has the highest possibility of being assigned. Our results indicate choosing δ_s to optimise success rate does not mean optimal expected distance will follow: since δ_s is only concerned with reaching the finish point, it terminates once it knows failure is guaranteed, thus lowering expected distance achieved by other strategies. This phenomenon is seen in Figure 9 which gives the appearance of δ_s striving straight to the finish point while δ_c and δ_d prioritise assigning tasks along the way.

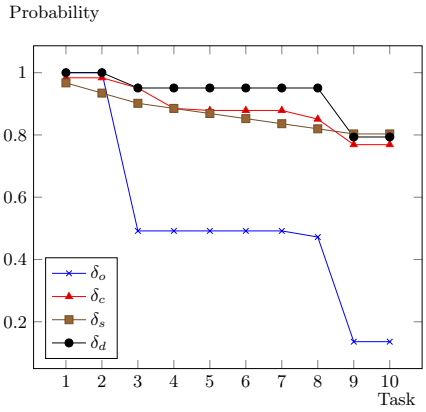

Fig. 9: Example 1 - expected distance

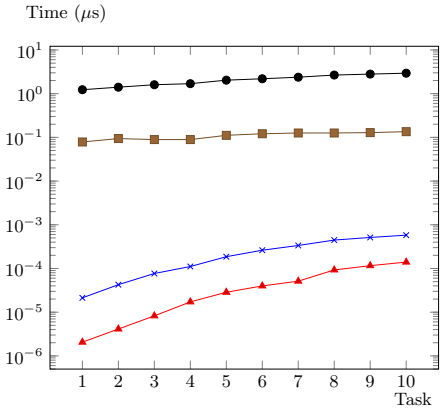

Fig. 10: Example 1 - computation time

It is noticeable that average task assignment probabilities are raised when made within the bounds of an assignment tree used in δ_c and δ_s. However an observed side-effect of removing known bad assignments (with reward $-\infty$) is to lower initial task assignment probabilities below that achieved by δ_o and δ_d. Figure 9 shows the probability to assign t_2 under δ_s is 0.9 and δ_c as 0.98, yet equals 1 under δ_o and δ_d. It follows that basing decisions solely on initial task assignments would indicate δ_o, δ_c and δ_d are better then δ_s, yet δ_s is more likely to finish. Caution should be taken of this somewhat false impression, especially with δ_o as performance can clearly drop suddenly. The behaviour of δ_o can be attributed to bad assignments made with or without user failure, and a greedy nature of using up critical users for early tasks leaving more and more bad assignment options for later ones. If no bad assignments or user failures exist, δ_o can expect to match the performance of δ_c.

The fastest computation time is achieved by δ_c which is expected due to the least amount of runtime processing it must perform. Note that δ_c uses the assignment tree data structure so does not need to calculate correct task assignments at runtime, nor does it calculate any aspect of resiliency; only the current failed users must be removed. This runtime performance does come with the cost of calculating all correct assignments offline and generating the data tree structure (17.39s for Example 1); the time for this grows exponentially with the number of tasks and users. It follows that timings for each strategy increase with the amount of runtime processing performed. As expected, the slowest strategy δ_d has the heaviest runtime workload, i.e. calculating correct assignments and calculating a resiliency value for each potential task assignment.

To summarise, none of the four policies we have introduced guarantee a full success rate and we do not suggest one is the outright best in terms of optimising workflow satisfaction and resiliency. Optimising success can lower distance while optimising time can lower resiliency for example. These tensions are heavily dependent on the nature of the workflow, the security policy and which users

Table 3: Assignment strategy comparison

Characteristic	δ_o	δ_c	δ_s	δ_d
User selection from $PU_{c,t}$	random	random	optimal	optimal
Caches assignment options	×	✓	✓	×
Calculates resiliency	×	×	✓	✓
Optimal success rate	random	random	✓	random
Optimal expected distance	random	random	random	✓
Computation	2^{nd}	1^{st}	3^{rd}	4^{th}

fail and at what point they fail. The results we have presented give a first step in the generation of useable metrics indicating the success rate, expected distance, computation time and task-assignment probabilities of workflow assignments. These values give a more meaningful measure than previous work and make it easier to compare workflow assignments in terms of how much satisfiability and resiliency they can give.

Clearly the establishment of an acceptable workflow is a case of security and business trade-offs. Favouring one measure over another will depend on workflow priorities, i.e., whether the only concern is to finish or instead be confident that a certain point will be reached, or does computation time outweigh the need for resiliency? This decision can become crucial due to tension we have shown existing between these aspects. Providing suitable metrics and tools for workflow designers would facilitate more informed decisions regarding these concerns.

Running Example In addition, two sets of results for the running example used throughout this paper are given in Table 4. The first, Example 2 are generated using policy p_1 defined in Section 3.1, and second, Example 3 using policy p_2 defined in Section 4.2. A maximum of 2 user failures is considered totalling 67 equiprobable failures, each run 10 times per assignment policy. The testing for Example 2 was carried out on a computing platform incorporating a 2.40Ghz i5 Intel processor, and for Example 3, a 2.3GHz Intel duo-core processor, both with 4GB RAM.

6 Conclusion

We have presented in this paper a Markov Decision Process (MDP) encoding of the workflow satisfaction and resiliency problem. We have therefore reduced the problem of finding optimal user-task assignment to that of solving an MDP, which is a well studied problem. One of the main strengths of our approach is to provide a very flexible approach, where a simple modification of the context or the reward function provides a new metric to analyse a workflow. We believe that by addressing the workflow satisfaction and resiliency problem from a quantitative viewpoint rather than from a binary one, we provide tools and metrics that can be helpful for a workflow designer to analyse all those cases that are neither satisfiable nor resilient ideally, but need to work nevertheless.

Table 4: Test results

	Example 2				Example 3			
	δ_o	δ_c	δ_s	δ_d	δ_o	δ_c	δ_s	δ_d
Success rate (%)	20.27	26.49	43.24	43.24	62.99	63.43	74.62	73.13
Expected distance (tasks)	2.04	1.93	1.81	2.27	2.57	2.59	2.52	2.67
Computation time (μs)	4.56	0.77	123.27	299.62	12.03	5.10	379.35	678.91

We have illustrated that the analysis of a workflow is multi-dimensional, and that there is a trade-off to be established, among others, between computation time, success rate and expected distance. Clearly, other dimensions can be taken into account, such that the possibility to dynamically modify the security policy [4], or perhaps the possibility to override the security constraints [22].

For future work, an interesting point is to develop the tools to help the system designer *fix* a given workflow, using different metrics. For instance, a set of workflow modifications proven to be monotonic with the quantitative satisfaction or with the decremental resiliency could be a very helpful tool, especially in the context of structured workflow design, e.g., with business processes. Another lead is the study of sub-optimal policies. Indeed, calculating a sub-optimal solution might be more tractable [8], at the cost of a loss of accuracy. In this case, it could be worth understanding the impact on the WSP of using a sub-optimal solution.

References

1. Workflow handbook 1997. chapter The Workflow Reference Model, pages 243–293. John Wiley & Sons, Inc., New York, NY, USA, 1997.
2. H. E. Bakkali. Enhancing workflow systems resiliency by using delegation and priority concepts. *Journal of Digital Information Management*, 11(4):267 – 276, 2013.
3. D. Basin, S. J. Burri, and G. Karjoth. Obstruction-free authorization enforcement: Aligning security with business objectives. In *Proceedings of the 2011 IEEE 24th Computer Security Foundations Symposium*, CSF '11, pages 99–113, Washington, DC, USA, 2011. IEEE Computer Society.
4. D. Basin, S. J. Burri, and G. Karjoth. Optimal workflow-aware authorizations. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, SACMAT '12, pages 93–102, New York, NY, USA, 2012. ACM.
5. R. Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.
6. E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur.*, 2(1):65–104, Feb. 1999.
7. R. Botha and J. H. P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682, 2001.
8. A. R. Cassandra. Optimal policies for partially observable markov decision processes. Technical report, Brown University, Providence, RI, USA, 1994.

9. J. Crampton. A reference monitor for workflow systems with constrained task execution. In *Proceedings of the tenth ACM symposium on Access control models and technologies*, SACMAT '05, pages 38–47, New York, NY, USA, 2005. ACM.
10. J. Crampton and G. Gutin. Constraint expressions and workflow satisfiability. In *Proceedings of the 18th ACM Symposium on Access Control Models and Technologies*, SACMAT '13, pages 73–84, New York, NY, USA, 2013. ACM.
11. J. Crampton, G. Gutin, and A. Yeo. On the parameterized complexity of the workflow satisfiability problem. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 857–868, New York, NY, USA, 2012. ACM.
12. J. Crampton, G. Gutin, and A. Yeo. On the parameterized complexity and kernelization of the workflow satisfiability problem. *ACM Trans. Inf. Syst. Secur.*, 16(1):4, 2013.
13. A. Khan and P. Fong. Satisfiability and feasibility in a relationship-based workflow authorization model. In S. Foresti, M. Yung, and F. Martinelli, editors, *Computer Security ESORICS 2012*, volume 7459 of *Lecture Notes in Computer Science*, pages 109–126. Springer Berlin Heidelberg, 2012.
14. M. Kohler, C. Liesegang, and A. Schaad. Classification model for access control constraints. In *Performance, Computing, and Communications Conference, 2007. IPCCC 2007. IEEE Internationa*, pages 410–417, April 2007.
15. M. Kohler and A. Schaad. Avoiding policy-based deadlocks in business processes. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 709–716, 2008.
16. A. Kumar, W. M. P. van der Aalst, and E. M. W. Verbeek. Dynamic work distribution in workflow management systems: How to balance quality and performance. *J. Manage. Inf. Syst.*, 18(3):157–193, Jan. 2002.
17. M. Lowalekar, R. Tiwari, and K. Karlapalem. Security policy satisfiability and failure resilience in workflows. In V. Maty, S. Fischer-Hbner, D. Cvrek, and P. venda, editors, *The Future of Identity in the Information Society*, volume 298 of *IFIP Advances in Information and Communication Technology*, pages 197–210. Springer Berlin Heidelberg, 2009.
18. J. Mace, A. van Moorsel, and P. Watson. The case for dynamic security solutions in public cloud workflow deployments. In *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, pages 111–116, June 2011.
19. F. Martinelli and C. Morisset. Quantitative access control with partially-observable markov decision processes. In *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, CODASPY '12, pages 169–180, New York, NY, USA, 2012. ACM.
20. National Quality Board. How to ensure the right people, with the right skills, are in the right place at the right time @ONLINE, 2013.
21. K. Tan, J. Crampton, and C. Gunter. The consistency of task-based authorization constraints in workflow. In *Computer Security Foundations Workshop, 2004. Proceedings. 17th IEEE*, pages 155–169, June 2004.
22. J. Wainer, P. Barthelmess, and A. Kumar. W-rbac - a workflow security model incorporating controlled overriding of constraints. *International Journal of Cooperative Information Systems*, 12:2003, 2003.
23. Q. Wang and N. Li. Satisfiability and resiliency in workflow authorization systems. *ACM Trans. Inf. Syst. Secur.*, 13(4):40:1–40:35, Dec. 2010.
24. P. Watson. A multi-level security model for partitioning workflows over federated clouds. *Journal of Cloud Computing*, 1(1):1–15, 2012.