DISS. ETH NO. 20419

MODELING AND ENFORCING WORKFLOW AUTHORIZATIONS

A dissertation submitted to
ETH ZURICH
for the degree of
Doctor of Sciences

presented by

Samuel Jakob Burri

MSc ETH CS
born on 12 October 1981
citizen of Root (LU) and Malters (LU)

accepted on the recommendation of
Prof. Dr. David Basin, examiner
Dr. Günter Karjoth, co-examiner
Prof. Dr. Srdjan Capkun, co-examiner

2012

Was ist bloss mit den Wörtern los?
Ich schüttle Sätze, wie man eine kaputte Uhr schüttelt,
und nehme sie auseinander;
darüber vergeht die Zeit, die sie nicht anzeigt.

Max Frisch, 2010

Abstract

Authorizations are fundamental in protecting information systems. In this dissertation, we study the modeling and enforcement of authorizations for workflows, which are a well-established abstraction of an organization's business processes. We thereby make authorizations sensitive to their business environment. For example, they may be tailored to workflow-specific relations between tasks and may anticipate potential future task executions. A prevalent class of authorizations, whose realization benefits from this additional information, are Separation of Duty (SoD) constraints. They aim at reducing fraud and errors and are therefore commonplace in regulated environments, such as the financial industry. More specifically, we address two main problems.

First, we study the refinement of abstract SoD constraints to concrete workflow models, and how to integrate an enforcement monitor for the resulting refinement into a heterogeneous workflow environment. We thereby bridge the gap between the formalization of high-level, workflow-independent authorization requirements and their enforcement. Our enforcement formalization and its service-oriented implementation account for the dynamics of today's business environments. In particular, we model administrative activities that reflect organizational changes occurring during workflow execution, addressing a well-known source of fraud.

Second, we propose a novel approach to aligning the enforcement of authorizations with their workflow-based business environment. Existing workflow authorization models make strong restrictions on workflows' control-flows; for example, they do not support loops. Our approach, in particular our novel technique for scoping authorizations within workflows, lifts this restriction. We proceed by identifying the notion of an obstruction, which generalizes deadlock caused by authorizations, and we study the construction of obstruction-free enforcement mechanisms. Finally, we introduce the concept of optimal authorizations for a workflow that maximize protection, yet allow for their obstruction-free enforcement. We provide tool support for our approach by extending a modeling platform and by building on algorithms from optimization theory.

Zusammenfassung

Autorisierungen sind ein fundamentaler Bestandteil des Schutzes von Informationssystemen. Im Fokus dieser Dissertation steht das Modellieren und Durchsetzen von Autorisierungen im Kontext von Workflows, einer etablierten Abstraktion von Geschäftsprozessen. Dadurch werden Autorisierungen mit dem Geschäftsumfeld verknüpft und können beispielsweise auf Beziehungen zwischen mehreren Arbeitsschritten zugeschnitten werden und zukünftige Arbeitsschritte antizipieren. Ein weitverbreitetes Autorisierungskonzept, dessen Realisierung von diesen zusätzlichen Informationen profitiert, ist das Mehr-Augen-Prinzip. Dieses bezweckt, Betrug und Fehler zu verhindern, und ist in regulierten Geschäftsfeldern, wie beispielsweise der Finanzindustrie, allgegenwärtig. Wir beschäftigen uns mit zwei Hauptproblemen.

Erstens studieren wir das Verfeinern von abstrakten Bestimmungen, die auf dem Mehr-Augen-Prinzip beruhen, zu konkreten Workflow-Modellen und die Frage, wie die gewonnenen Verfeinerungen in heterogenen Workflow-Umgebungen durchgesetzt werden können. Damit überbrücken wir die Kluft zwischen einer Workflow-unabhängigen Spezifikation von Autorisierungsanforderungen und deren Durchsetzung. Unsere Formalisierung und deren Service-orientierte Umsetzung sind an die Dynamik der heutigen Geschäftswelt angepasst. So modellieren wir beispielsweise administrative Aktivitäten während der Ausführung von Workflows, die zu Autorisierungsveränderungen führen und bei Nichtbeachtung Betrug begünstigen.

Zweitens präsentieren wir einen neuen Ansatz, um die Durchsetzung von Autorisierungen mit ihrem Workflow-basierten Geschäftsumfeld abzustimmen. Existierende Workflow-Autorisierungsmodelle schränken den Kontrollfluss von Workflows stark ein, da sie beispielsweise keine Schleifen erlauben. Unser Ansatz, insbesondere unsere Technik zum Abgrenzen von Autorisierungen, hebt diese Einschränkungen auf. Weiter führen wir den Begriff der Obstruction ein, die eine durch Autorisierungen begründete, teilweise Blockierung einer Workflow-Ausführung beschreibt. Darauf aufbauend studieren wir die Konstruktion von Obstruction-freien Durchsetzungsmechanismen. Schliesslich führen wir

das Konzept von optimalen Autorisierungen für einen Workflow ein, die eine Obstruction-freie Durchsetzung ermöglichen und dabei maximalen Schutz bieten, beziehungsweise Kosten minimieren. Durch die Erweiterung einer bestehenden Modellierungssoftware und unter Anwendung von bekannten Optimierungsalgorithmen stellen wir eine Werkzeugunterstützung für unseren Ansatz bereit.

Acknowledgments

First, I want to thank David Basin and Günter Karjoth for supervising my PhD and for letting me pursue my ideas while acting as a tough but fair sounding board. Their flexibility, generosity, and trust made me feel at home both at IBM Research and at ETH Zurich. Inspired by their curiosity and sense of precision, I learned how to develop and crystallize ideas by undergoing the painful yet rewarding process of putting them into words. Hence, the epigraph. Furthermore, I thank Srdjan Capkun for serving as my co-examiner.

Second, I want to thank my current and former colleagues at IBM and ETH for creating a sociable and stimulating work environment. In particular, I want to mention Patrik Bichsel, Maria Dubovitskaya, Konrad Eriksson, Chris Giblin, Anja Lehmann, Samuel Müller, Franz-Stefan Preiss, Dieter Sommer, and Mario Verdicchio with whom I enjoyed work-related but also many political, social, and geeky discussions. I am grateful to Mohammad Torabi Dashti and Srdjan Marinovic who provided rigorous feedback on multiple drafts of my thesis and thereby repeatedly fueled my ambition to improve my work. Furthermore, I thank my godfather Matthias Burri for proofreading my thesis' Zusammenfassung. Finally, I thank Michael Osborne and Andreas Wespi for their support in dealing with IBM's bureaucracy and politics.

Third, I want to express my gratitude to my family and friends for supporting me wholeheartedly. More specifically, I thank my parents Katharina Burri-Bräm and Balz Burri for giving me an excellent mix of freedom, rules, values, and love throughout my childhood and to the current day. Furthermore, I thank my sister Nina Burri for her often blunt feedback and advice on numerous topics. I am truly blessed to count on the friendship of many great individuals. In particular, the "5 Freunde" gang and their families who have been enriching my life ever since high school. Also, my Unitech friends, in particular, those with whom I had the pleasure to revitalize Unitech's alumni association. Finally, I am deeply indebted to my girlfriend Natalie Surber who not only puts up with me and my undertakings, such as my PhD, but who also gives me a lot of strength and guidance with her good heart, calm, and love.

Contents

Chapter 1

Introduction

Authorizations govern which actions subjects are permitted to perform on protected resources [Saltzer and Schroeder 1975; Harrison *et al.* 1976]. A central problem when specifying authorizations is to strike a balance between protecting a system's resources and empowering its subjects. Tailoring authorization models to the environment in which their respective policies are enforced simplifies this balancing and is thus integral to both the successful operation and protection of today's ubiquitous information systems.

In this dissertation, we present novel results on modeling and enforcing authorizations in the context of workflows, which are a well-established abstraction of an organization's business processes [Curtis *et al.* 1992; Georgakopoulos *et al.* 1995; van der Aalst 1998]. The composition of authorization and workflow models thus enables us to tailor authorizations to their business environment. Yet, the concept of a workflow is abstract enough that our results are applicable across a broad range of domains.

More precisely, a workflow models causal dependencies between a set of tasks, whose execution constitutes a business objective. In this authorization context, subjects are users and the execution of task instances corresponds to performing actions on resources. In the workflow context, authorization decisions are no longer made in isolation. They may depend on previously executed task instances and on the set of potential future executions, derived from the workflow's control-flow. Authorizations can also mitigate threats specific to this additional context. For example, *Separation of Duties (SoD)*, also known as *Four-Eyes-Principle*, aims at reducing fraud and errors by preventing users from executing multiple tasks that result in a conflict of interest [Saltzer and Schroeder 1975; Sandhu 1988]. Similarly, *Binding of Duties (BoD)*, dual to SoD, aims at reusing existing knowledge and preventing widespread dissemination of sensitive information by restricting the execution of related tasks to a small set of users.

A further driver for weaving workflows and authorizations together is the increasing number of regulatory requirements, such as the Sarbanes-Oxley Act (SOX) [2002], and best-practice frameworks like COBIT [ITGI 2005]. They mandate organizations to document their business processes, to identify conflicts of interests, to adopt countermeasures, and to audit and control these activities. These requirements are motivated by the observation that security threats often come from situations where authorized users accidentally or intentionally misuse resources [E&Y 2009]. Examples of such situations are non-compliance and include the scandals as reported by [The Economist 2001]. Hence, as information systems increasingly automate business processes, studying how to apply and extend traditional authorization models to this context becomes inevitable.

1.1 Problem Statement

We first illustrate and motivate the two main problems that we address in this dissertation with an example. Afterward, we describe each problem in more detail.

Example 1.1 (Simple Payment Workflow) Consider the simple payment workflow illustrated on the left of Figure 1.1. First, a user prepares the payment (t_1). Depending on the amount of money involved, afterward the payment is either approved by a peer (t_2) or an executive (t_3). Let an execution of the payment workflow be constrained by the authorizations illustrated as assignments from users to tasks on the right of Figure 1.1. For example, Bob may execute the executive approval (t_3) whereas Alice is not authorized to do so.

Suppose now that an organization that executes the payment workflow has the abstract SoD policy P that at least two different users must participate in the execution of business processes dealing with financial transactions. The first main problem that we address is how to refine such abstract policies to concrete

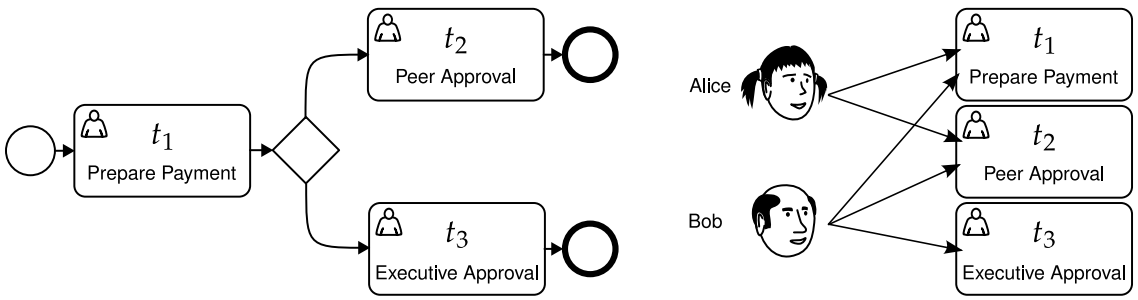


Figure 1.1. Simple payment workflow

workflows, like the payment workflow, and how to combine and integrate such policies with other authorizations, such as a user-task assignment.

When enforcing P and the user-task assignment shown in Figure 1.1, the following problem may occur: After Bob executes t_1, no user is authorized to execute t_3; if Alice were to execute t_3, she would violate the user-task assignment and if Bob were to execute t_3, he would violate P. However, if Alice executes t_2, the workflow can still successfully terminate, but with fewer options to reach the business objective. Thus, the second main problem that we address is how to align the enforcement of authorizations with the business objective represented by a workflow and thereby avoiding situations as sketched above. ★

Refinement and Integration. As illustrated by the policy P in Example 1.1, regulatory requirements and best-practice frameworks are typically specified in an informal and workflow-independent form. An abstract policy model can capture such requirements in their full generality, independent of a specific workflow. However, before the respective policies can be enforced, they must be refined to a concrete workflow. This translation is labor-intensive, error-prone, and therefore inadequate for environments with frequent changes to requirements and business processes. However, such dynamics are common in today's business world due to outsourcing, mergers, acquisitions, *etc.* At the same time, these dynamics create loopholes, which are prone to exploitation by insiders and increase the likelihood of fraud [E&Y 2009]. Hence, in the environments where they are most needed, consistent authorisations and regulations are most difficult to implement.

We address the problem of automating the refinement of abstract authorization requirements to concrete workflows and integrating an enforcement mechanism for these refined policies into an heterogeneous and dynamic workflow environment. We thereby enable a faster adaptation to changes and a reduction in errors and manual work. Furthermore, auditing is simplified due to a formalization of requirements, which is needed for automation, resulting in an increase of precision and transparency.

Alignment and Optimality. We have observed above that modeling and enforcing authorizations requires striking a balance between protection and empowerment. If all task executions were permitted, the underlying system would be operating freely and users would enjoy unlimited empowerment. The protection of the resources' confidentiality and integrity, however, would be minimal. In contrast, if all task executions were prohibited, the resources' protection would be ensured but the system's underlying business objectives could not be achieved.

We thus address the following fundamental problem: How are the conflicting objectives of protecting resources and empowering users best balanced? We distinguish two cases.

First, we consider the case where an authorization policy and a workflow are given. As illustrated in Example 1.1, enforcing authorizations may result in a workflow system that deadlocks or is obstructed in that fewer options are available to achieve the workflow's business objective than were originally designed. We thus analyze the question whether it is possible to enforce authorizations without obstructing workflows, *i.e.* how to align authorizations and business objectives.

Second, we consider the case where a workflow is given, and we are asked to specify an authorization policy that allows an obstruction-free enforcement. A classical answer to the question how to specify authorizations is the principle of *least privilege* [Saltzer and Schroeder 1975], which says that users shall only be authorized to execute the tasks necessary to complete their job. However, in an environment where multiple users may execute different tasks, multiple authorization policies may satisfy least privilege. The choice of an authorization policy may be further influenced by the cost associated with the respective administrative change. Thus, although least privilege is a guiding principle, it does not provide the final answer to the question of what constitutes an optimal authorization policy for a workflow. We revisit this question, building on our results from the first case.

1.2 Contributions

This dissertation makes two main contributions, built upon their respective subcontributions.

1.2.1 Enforcement of Abstract SoD Constraints

To address the problem of refinement and integration, we present a novel approach to bridging the gap between the abstract specification of SoD constraints and their enforcement in a dynamic and heterogeneous enterprise workflow environment. We build on Li and Wang's *SoD Algebra (SoDA)* [2008], which allows the modeling of SoD constraints at a high level of abstraction, independent of a specific workflow, and capturing quantification and qualification requirements. As a consequence, business experts can focus on modeling business processes as workflows and security experts on specifying internal controls. Each of them requires minimal interaction with the other, thereby saving efforts and cost. In

order to go from abstract authorization specifications to their enforcement, we make the following contributions:

- **Generalization of SoDA's Semantics:** Due to SoDA's abstract nature, design decisions arise when mapping SoD policies formalized in SoDA onto workflow instances. We provide a solution to this problem by refining Li and Wang's set-based SoDA semantics first to a multiset semantics and then to a trace-based semantics. A correctness proof for our formal model establishes that every SoD-constrained workflow instance that successfully terminates satisfies the respective SoDA term with respect to our trace semantics.

- **SoD as a Service:** New technologies and methodologies, such as Service-Oriented Architectures (SOAs), facilitate the extension of legacy information systems with new functionality. We build on these advances by provisioning the SoD enforcement functionality as an instance of the software delivery model *Software as a Service (SaaS)* [Turner *et al.* 2003], which we call *SoD as a Service*. We implement SoD as a Service combining off-the-shelf software and newly developed components. Our architecture and implementation enables a loose coupling between a workflow engine that executes the business logic, a user repository that administers users and their workflow-independent authorizations, and the enforcement of abstract SoD constraints. In exchange for a moderate increase in communication, our architecture separates concerns and reduces implementation and configuration costs. At the same time, changing legal requirements and organizational changes can quickly be reflected in our architecture.

- **Accounting for Administrative Changes to Authorizations:** As mentioned in Section 1.1, organizational changes are commonplace and a major source of fraud. However, previous work on SoD enforcement implicitly assumes that authorizations are not administrated, *i.e.* their policies are not edited, during workflow execution. We relax this assumption by modeling administrative activities in our trace-based SoDA semantics. Our formal models and the thereupon building implementation of SoD as a Service are therefore well-suited to handle the dynamics of today's fast-paced business environments.

These results are published in [Basin *et al.* 2009; 2011a] and consolidated in a journal article, which is under submission [Basin *et al.* 2012a].

1.2.2 Alignment of Authorizations and Business Objectives

Orthogonal to the contributions outlined in the previous section, we present a novel approach to enforcing authorizations in alignment with a business objective, modeled by a workflow. To this end, we propose a new class of SoD and BoD constraints, which impose no restrictions on the expressivity of the underlying workflow modeling language. We thereby overcome the weakness that is shared by most existing SoD and BoD models, namely that they cannot cope with workflows containing loops and conditional execution because they model workflows as sequences or partially ordered sets of tasks. Loops and conditional execution, however, are well-established in the business process community [van der Aalst *et al.* 2003] and supported by commercial workflow modeling languages such as the Business Process Modeling Notation (BPMN) [OMG 2011a].

We then study the enforcement of authorization policies which are composed of our SoD and BoD constraints and workflow-independent authorizations, such as a *Role-based Access Control (RBAC)* [Ferraiolo *et al.* 2001] policy stored in a user repository. We make the following contributions:

- **Release Points:** We present a novel approach to scoping authorizations to subsets of task instances. We demarcate scopes with markers, called *release points*, which are inserted into a workflow's control-flow. This approach imposes no restriction on a workflow's control-flow. Release points are, in particular, of interest in the presence of loops. We demonstrate the expressive power of release points by incorporating them into our SoD and BoD constraints and visualize them by extending BPMN. Furthermore, we provide tool support for our modeling approach by extending an existing modeling platform, namely Oryx [2012].

- **Obstruction-Free Authorization Enforcement:** Given an authorization policy and a workflow, we formulate the existence of an obstruction-free enforcement mechanism as a decision problem, which we call the *enforcement process existence (***EPE***)* problem. **EPE** proposes a notion of alignment between authorizations and business objectives, generalizing the deadlock-freedom of processes to also include cases where progress of the workflow execution is possible but with fewer options than specified by the workflow's control-flow. We prove that **EPE** is decidable, but **NP**-hard. Furthermore, we present an approximation algorithms for **EPE** with a polynomial runtime complexity, which has good approximation results when the set of users is large and workflow-independent authorizations are equally distributed among them.

- **Optimal Workflow-Aware Authorizations:** We refine the decision problem of whether an authorization policy allows an obstruction-free enforcement on a given workflow to the notion of an *optimal* authorization policy that satisfies this property. Our approach provides considerable modeling freedom in terms of the optimality notion used. For example, we may aim to minimize the cost associated with a policy change or maximize the protection resulting from the new policy. We thereby facilitate a fine-grained balancing of empowerment and protection with respect to various criteria. Moreover, we prove that finding an optimal role-based authorization policy that enables an obstruction-free enforcement is **NP**-complete.

These results are published in [Basin *et al.* 2011c; 2012c] and a journal paper extending the former paper is under submission [Basin *et al.* 2012b].

1.3 Organization

We first introduce in Section 1.3.1 our classification of authorizations, which we use as roadmap for our work in addition to the problems introduced in Section 1.1 and the contributions presented in Section 1.2. Afterward, we present in Section 1.3.2 the outline of this dissertation.

1.3.1 Authorization Classification

We classify authorizations with respect to three criteria:

- **Modeling Scope:** *Workflow-specific* authorizations are designed in alignment with a given workflow and may be tailored to a workflow's specific properties such as its control-flow. In contrast, *workflow-independent* authorizations are specified without any knowledge of the workflow on which they will be enforced. As such, workflow-independent authorizations are more generic and are enforceable across different workflows, but they do not account for workflow-specific properties.

- **History Dependence:** Authorizations whose evaluation depends on who has, or who has not, executed previous task instances are called *history-dependent*; otherwise, they are called *history-independent*. History-dependent authorizations are more flexible than history-independent authorizations. However, their enforcement and analysis is more complex as it also encompasses an execution history.

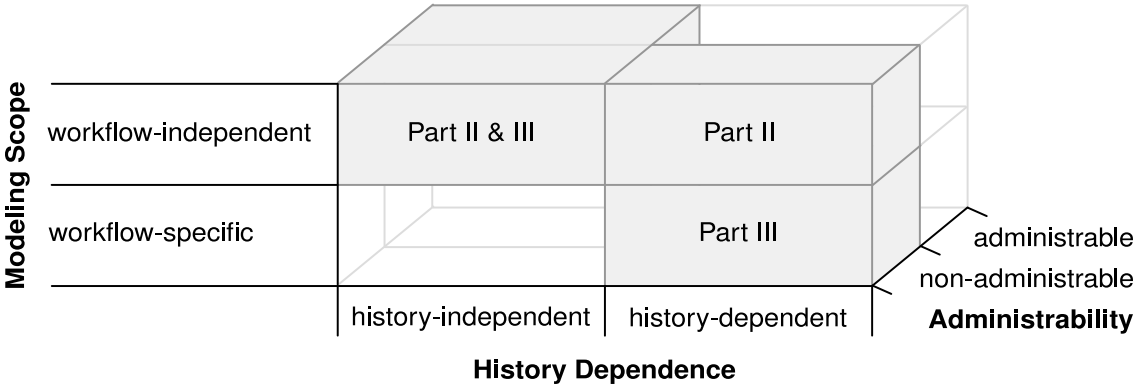


Figure 1.2. Classification of authorizations

- **Administrability:** Users join or leave organizations or are promoted. Authorizations that may be administrated during workflow execution, *i.e.* the respective policy is edited to reflect organizational changes, are called *administrable*. *Non-administrable* authorizations, in contrast, cannot be administrated during workflow execution.

Figure 1.2 shows a cube with all combinations of these criteria. The entries of the matrix refer to the parts of this dissertation where examples of the respective combinations are studied in detail. We comment in Chapter 12, related work, on combinations that are not studied throughout the main parts.

In practice, *e.g.* [IBM 2011], the effective authorization policy that constrains the execution of a workflow is a composition of different (sub-)policies. For example, records of employees and their workflow-independent and history-independent authorizations may be stored in a user repository. Changes to these entries may occur at any time, *i.e.* they are administrable. In addition, a workflow may be augmented by workflow-specific, history-dependent, and non-administrable authorizations. Authorized task executions are then determined by a composition of the authorizations stored in the user repository and those stored in the workflow.

1.3.2 Outline

The structure of this dissertation follows our problem description and the corresponding contributions. Part II addresses the problem of refinement and integration, while Part III addresses the problem of alignment and optimality. These main parts are preceded by Part I, which builds context, and followed by closing observations in Part IV. In more detail:

In Part I, we introduce the theoretical groundwork that we build upon throughout the rest of this dissertation, and we present the regulations and best-practice frameworks that function as our requirements. In particular, we summarize in Chapter 2 the process algebra CSP and the workflow modeling language BPMN, which we use to formalize and visualize authorization-constrained workflows, respectively. We also give a definition of multisets required for our generalization of SoDA's semantics in Part II, graph coloring definitions used in algorithms and complexity reductions in Part III, and integer linear programming employed for solving optimization problems in Part III.

In Part II, we present our workflow-independent approach to enforcing abstract SoD constraints, formalized as SoDA terms, in a dynamic workflow environment. We proceed in Chapter 4 by constructing CSP models of workflows and history-independent authorizations. In Chapter 5, we introduce SoDA and generalize its semantics to multisets and traces. Furthermore, we present a mapping from SoDA terms to CSP processes, which we combine with the previously introduced processes to model an SoD-secure workflow system. These models serve in Chapter 6 as blueprint for our SoD as a Service implementation. We analyze the runtime complexity of this implementation and support our findings with performance measurements from a realistic example workflow. We conclude Part II with an evaluation of our approach in Chapter 7.

In Part III, we present our workflow-specific approach to aligning authorizations and business objectives and to compute an optimal authorization policy for a given workflow. We start in Chapter 8 by refining the CSP-based workflow and authorization models introduced in Part II. In particular, we introduce a novel class of workflow-specific and history-dependent SoD and BoD constraints. To graphically model our SoD and BoD constraints, we propose an extension of BPMN and report on its integration into the modeling platform Oryx [2012]. In Chapter 9, we introduce the enforcement process existence (**EPE**) problem and present algorithms both to solve and approximate **EPE**. In Chapter 10, we drop the previous assumption that workflow-independent authorizations are non-administrable and model the price of changing from one history-independent authorization policy to another one by a cost function. We then reduce the problem of finding a cost-minimizing authorization policy that allows an obstruction-free authorization enforcement to the well-established *integer linear programming* (**ILP**) problem. Finally, we conclude Part III in Chapter 11 with an evaluation of our results.

In Part IV, we round off this dissertation with a presentation of related work in Chapter 12 and conclusions in Chapter 13.

Part I

Context

Chapter 2

Background

This chapter recaptures the underlying definitions and results that we use throughout this dissertation.

We denote by $\mathbb{N}$ and $\mathbb{N}_0$ the set of natural numbers, excluding and including zero, respectively, by $\mathbb{Z}$ the set of integers, and by $\mathbb{R}$ the set of real numbers. Let Z, Z_1, and Z_2 be sets. The *power set* of Z, denoted 2^Z, is the set of all subsets of Z; formally, $2^Z = \{Z' \mid Z' \subseteq Z\}$. We may identify a function $\pi : Z_1 \to Z_2$ with its relation (graph) $\pi \subseteq Z_1 \times Z_2$. For example, for $z_1 \in Z_1$ and $z_2 \in Z_2$, if $\pi(z_1) = z_2$ we may equivalently write $(z_1, z_2) \in \pi$. We refer to the *domain* of a relation π as $\mathsf{dom}(\pi)$, to its *range* as $\mathsf{ran}(\pi)$, and to its *inverse* as π^{-1}.

2.1 Communicating Sequential Processes

We use a subset of Hoare's process algebra CSP [Roscoe 2005] to model the specification and enforcement of authorization constraints on workflows. CSP describes a system as a set of communicating *processes*. A process is referred to by a *name*; let $\mathcal{N}$ be the set of all process names. Processes communicate with each other by concurrently engaging in *events*. Σ is the set of all regular events, which may be structured as tuples. For example, $z_1.z_2.\ldots.z_k$ denotes the event that corresponds to the tuple $(z_1, z_2, \ldots, z_k) \in Z_1 \times Z_2 \times \ldots \times Z_k$, for sets Z_1, Z_2, $\ldots$, Z_k and $k \in \mathbb{N}$. In addition to regular events, there are two special events: τ, a process-internal, hidden event, and $\checkmark$ that communicates successful termination. Let $D \subseteq \Sigma$ be a subset of regular events. We write D^τ for $D \cup \{\tau\}$, $D^\checkmark$ for $D \cup \{\checkmark\}$, and $D^{\tau,\checkmark}$ for $D \cup \{\checkmark, \tau\}$. In particular, $\Sigma^{\tau,\checkmark}$ is the set of all events.

A *trace* is a sequence of regular events, possibly ending with $\checkmark$. $\langle\rangle$ is the empty trace and $\langle\sigma_1, \ldots, \sigma_k\rangle$ is the trace containing the events σ_1 to σ_k, for $k \in \mathbb{N}$. For two traces i_1 and i_2, their concatenation is denoted $i_1 \hat{} i_2$. D^* is the set of all finite traces over D and its superset $D^{*\checkmark} = D^* \cup \{i \hat{} \langle\checkmark\rangle \mid i \in D^*\}$ includes also all traces ending with $\checkmark$. For a trace i, $i \upharpoonright D$ denotes i *restricted* to events

in D. Formally, $\langle\rangle \upharpoonright D = \langle\rangle$ and, for $i = \langle\sigma\rangle\hat{}i'$, $i \upharpoonright D = \langle\sigma\rangle\hat{}(i' \upharpoonright D)$ if $\sigma \in D$ and $i \upharpoonright D = (i' \upharpoonright D)$ if $\sigma \notin D$. We abuse the set-membership operator $\in$ and write $\sigma \in i$ for an event σ and a trace i, if there exist two traces i_1 and i_2 such that $i = i_1\hat{}\langle\sigma\rangle\hat{}i_2$.

For an event $\sigma \in \Sigma$, a set of events $D \subseteq \Sigma$, a name $n \in \mathcal{N}$, and a relation $R \subseteq \Sigma \times \Sigma$, the set of processes $\mathcal{P}$ is inductively defined by the grammar

$$\mathcal{P} ::= \sigma \to \mathcal{P} \mid \mathit{SKIP} \mid \mathit{STOP} \mid n \mid \mathcal{P} \,\Box\, \mathcal{P} \mid \mathcal{P} \sqcap \mathcal{P} \mid \mathcal{P} \underset{D}{\parallel} \mathcal{P} \mid \mathcal{P} \,;\, \mathcal{P} \mid \mathcal{P}[R] \ .$$

There are different approaches to formally describing the behavior of a process. CSP's denotational semantics describes a process P as a prefix-closed set of traces $\mathsf{T}(P) \subseteq \Sigma^{*\checkmark}$, called the *traces model*. The operational semantics describes P as a labelled transition system. The two semantics are compatible. Because we mainly use the traces model in this dissertation, we describe in the following the process composition operators, introduced above, in terms of the denotational semantics. Afterward, we review the subset of CSP's operational semantics that we require in forthcoming proofs.

Denotational Semantics. Let $P, P_1, P_2 \in \mathcal{P}$ be processes. The process $\sigma \to P$ *engages* in the event σ first and behaves like P afterward. Formally, $\mathsf{T}(\sigma \to P) = \{\langle\sigma\rangle\hat{}i \mid i \in \mathsf{T}(P)\} \cup \{\langle\rangle\}$ This notation can be extended. The expression $\sigma : D \to P$ represents a process that engages in a $\sigma \in D$ first and behaves like P afterward. *SKIP* engages in $\checkmark$ and no further event afterward; $\mathsf{T}(\mathit{SKIP}) = \{\langle\rangle, \langle\checkmark\rangle\}$. *STOP* represents the process that does not engage in any event; $\mathsf{T}(\mathit{STOP}) = \{\langle\rangle\}$. In other words, *SKIP* represents successful termination and *STOP* a deadlock. The assignment of P to n is denoted by $n = P$ and can be parametrized. For example, $n(z) = P$ defines a process named n that is parametrized by the variable z and behaves like P. The process $P_1 \,\Box\, P_2$ represents the *external* choice and $P_1 \sqcap P_2$ the *internal* choice between P_1 and P_2. With respect to the traces model, $P_1 \,\Box\, P_2$ and $P_1 \sqcap P_2$ are indistinguishable, namely $\mathsf{T}(P_1 \,\Box\, P_2) = \mathsf{T}(P_1 \sqcap P_2) = \mathsf{T}(P_1) \cup \mathsf{T}(P_2)$. The failures model explained below distinguishes between the two processes. The process $P_1 \underset{D}{\parallel} P_2$ represents the parallel composition of P_1 and P_2 *synchronized* on D. This means, $P_1 \underset{D}{\parallel} P_2$ engages in an event $\sigma_1 \in D$ if P_1 and P_2 synchronously engage in σ_1 and $P_1 \underset{D}{\parallel} P_2$ engages in an event $\sigma_2 \notin D$ if either P_1 or P_2 engages in σ_2. The special event $\checkmark$ is always implicitly contained in the set of synchronization events D, *i.e.* $P_1 \underset{D}{\parallel} P_2$ can only successfully terminate if both P_1 and P_2 can successfully terminate. $P_1 \parallel P_2$ is an alternative notation for the fully synchronized parallel composition $P_1 \underset{\Sigma}{\parallel} P_2$; formally $\mathsf{T}(P_1 \parallel P_2) = \mathsf{T}(P_1) \cap \mathsf{T}(P_2)$.

Similarly, $P_1 \mid\mid\mid P_2$ denotes to the unsynchronized parallel composition of P_1 and P_2, $P_1 \underset{\varnothing}{\parallel} P_2$. The process $P_1 \,;\, P_2$ denotes the sequential composition of P_1 and P_2. It first behaves like P_1. Upon successful termination of P_1, the event $\checkmark$ is hidden, which is denoted by the invisible event τ. Afterward, the process behaves like P_2. Formally, $\mathsf{T}(P_1 \,;\, P_2) = (\mathsf{T}(P_1) \cap \Sigma^*) \cup \{i_1 \hat{} i_2 \mid i_1 \hat{} \langle\checkmark\rangle \in \mathsf{T}(P_1), i_2 \in \mathsf{T}(P_2)\}$. Note that the invisible event τ does not appear in traces, similar to ε-transitions in nondeterministic automata. The process $P[R]$ denotes P *renamed* by R. For every tuple $(\sigma_1, \sigma_2) \in R$, $P[R]$ engages in σ_2 if P engages in σ_1. If $\mathsf{T}(P_1) \subseteq \mathsf{T}(P_2)$, then P_1 is a *trace refinement* of P_2, denoted $P_2 \sqsubseteq_{\mathsf{T}} P_1$. If $P_2 \sqsubseteq_{\mathsf{T}} P_1$ and $P_1 \sqsubseteq_{\mathsf{T}} P_2$, then P_1 and P_2 are *trace equivalent*, denoted $P_1 =_{\mathsf{T}} P_2$.

The traces model is insensitive to nondeterminism. It describes what processes *can* do but not what they *may refuse* to do. The *failures model* F is a refinement of the traces model that overcomes this shortcoming. A *refusal set* of a process P is a set of events all of which P can refuse to engage in and $rs(P) \subseteq 2^{\Sigma^\checkmark}$ is the set of all refusal sets of P. The set of P's *failures* is then $\mathsf{F}(P) = \{(i, D) \mid i \in \mathsf{T}(P), D \in rs(P \setminus i)\}$, where $P \setminus i$ represents the process P after engaging in the events in the trace i. If $\mathsf{F}(P_1) \subseteq \mathsf{F}(P_2)$, then P_1 is a *failure refinement* of P_2, denoted $P_2 \sqsubseteq_{\mathsf{F}} P_1$. Furthermore, P_1 is *failure equivalent* to P_2, written $P_1 =_{\mathsf{F}} P_2$, if $P_1 \sqsubseteq_{\mathsf{F}} P_2$ and $P_2 \sqsubseteq_{\mathsf{F}} P_1$. The transition system interpretation of processes under CSP's operational semantics, presented below, defines failures in terms of non-existing transitions.

Operational Semantics. A *labelled transition system (LTS)* is a quadruple (Q, D, δ, q_0), where Q is a set of states, D is a set of input symbols, $\delta \subseteq Q \times D \times Q$ is a state transition relation, and $q_0 \in Q$ is a start state. For $k \in \mathbb{N}$, $q_0, q_k \in Q$, and a sequence of events $\langle\sigma_1, \ldots, \sigma_k\rangle \in D^*$, we write $q_0 \xrightarrow{\langle\sigma_1,\ldots,\sigma_k\rangle} q_k$ if there exists a sequence of states $\langle q_1, \ldots, q_{k-1}\rangle$ such that $(q_{l-1}, \sigma_l, q_l) \in \delta$ for all $l \in \{1, \ldots, k\}$.

CSP's operational semantics interprets a process as an LTS where the input symbols correspond to the events that the process engages in, *i.e.* $D \subseteq \Sigma^{\tau,\checkmark}$. Let $L = (Q, \Sigma^{\tau,\checkmark}, \delta, q_0)$ be an LTS. For $q_1, q_2 \in Q$ and a trace $i \in \Sigma^{*\checkmark}$, we write $q_1 \overset{i}{\Longrightarrow} q_2$ if there exists a sequence of events $h \in (\Sigma^\tau)^{*\checkmark}$ such that $q_1 \xrightarrow{h} q_2$ and i is equal to h without τ events.

Let $q \in Q$ be a state and $D \subseteq \Sigma^\checkmark$ a set of events. The set D is a *refusal set* of q, written $q \,ref\, D$, if $D \subseteq \{\sigma \in \Sigma^\checkmark \mid \neg\exists q' \in Q, (q, \sigma, q') \in \delta\}$. We say an LTS

$L = (Q, \Sigma^{\tau,\checkmark}, \delta, q_0)$ *corresponds*[1] to a process P, denoted L_P, if

$$\mathsf{F}(P) = \{(i,D) \mid \exists q \in Q, q_0 \stackrel{i}{\Longrightarrow} q, q \textit{ ref } D\} \cup \{(i,D) \mid \exists q \in Q, q_0 \stackrel{i^\frown\langle\checkmark\rangle}{\Longrightarrow} q, D \subseteq \Sigma^\checkmark\}\,.$$

Note that there may be multiple LTSs that correspond to the same process. We call a process P *finite* if there exists an LTS L_P with finitely many states and input symbols.

2.2 Multisets

A *multiset*, or *bag*, is a collection of objects where repetition is allowed [Syropoulos 2000]. Formally, given a set Z, a multiset $\mathbf{Z}$ of Z is a pair (Z, f), where the function $f : Z \to \mathbb{N}_0$ defines how often each element $z \in Z$ occurs in $\mathbf{Z}$. We write $\mathbf{Z}(z)$ as shorthand for $f(z)$ and say that z is an *element* of $\mathbf{Z}$, written $z \in \mathbf{Z}$, if $\mathbf{Z}(z) \geq 1$. We use double curly-brackets to define multisets, *e.g.* $\mathbf{Z} = \{\!\{z_1, z_1\}\!\}$ is the multiset where $\mathbf{Z}(z_1) = 2$ and $\mathbf{Z}(z) = 0$ for all $z \in Z \setminus \{z_1\}$. For a finite multiset $\mathbf{Z}$, $|\mathbf{Z}|$ denotes its *cardinality* and is defined as $\sum_{z \in Z} \mathbf{Z}(z)$. Let $\mathbf{Z}_1$ and $\mathbf{Z}_2$ be two multisets of Z. Their *intersection*, denoted $\mathbf{Z}_1 \cap \mathbf{Z}_2$, is the multiset $\mathbf{Z}$, where for all $z \in Z$, $\mathbf{Z}(z) = \min(\mathbf{Z}_1(z), \mathbf{Z}_2(z))$. Similarly, their *union*, denoted $\mathbf{Z}_1 \cup \mathbf{Z}_2$, is the multiset $\mathbf{Z}$, where for all $z \in Z$, $\mathbf{Z}(z) = \max(\mathbf{Z}_1(z), \mathbf{Z}_2(z))$, and their *sum*, denoted $\mathbf{Z}_1 \uplus \mathbf{Z}_2$, is the multiset $\mathbf{Z}$, where for all $z \in Z$, $\mathbf{Z}(z) = \mathbf{Z}_1(z) + \mathbf{Z}_2(z)$. The *empty multiset* $\varnothing$ of Z is the multiset where $\varnothing(z) = 0$, for all $z \in Z$.

2.3 Graph Coloring

A *graph* G is a tuple (V, E) where V is a set of *vertices* and $E \subseteq \{e \subseteq V \mid 2 = |e|\}$ is a set of 2-element subsets of V called *edges*. The *maximal degree* of a graph G, denoted $\Delta(G)$, is $\max_{v \in V} |\{v' \in V \mid \{v, v'\} \in E\}|$, *i.e.* the maximal number of edges linking a vertex to other vertices.

Definition 2.1 (The *k*-**Coloring** Problem)

Input: *A graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.*

Output: YES *if there exists a function* $\mathsf{col} : V \to \{1, \ldots, k\}$ *such that for every edge* $\{v_1, v_2\} \in E$, $\mathsf{col}(v_1) \neq \mathsf{col}(v_2)$ *or* NO *otherwise.*

Let a graph G and an integer k be given. We call a function col a *k-coloring for* G if col satisfies the condition described in the *k*-**Coloring** problem for G

[1] The CSP-versed reader may have realized that we omit a discussion of divergence. We implicitly assume that the processes that model workflows in the following chapters are divergence-free. Our renaming relations and authorization processes do not introduce divergence.

and k. The k**-Coloring** problem is **NP**-complete [Chartrand and Zhang 2008]. The following problem generalizes k**-Coloring**.

Definition 2.2 (The **ListColoring** Problem)

Input: *A graph* $G = (V, E)$ *and a function* $\mathsf{L} : V \to 2^Z$*, for a set* Z*.*

Output: YES *if there exists a function* $\mathsf{col}_\mathsf{L} : V \to Z$ *such that for every vertex* $v \in V$*,* $\mathsf{col}_\mathsf{L}(v) \in \mathsf{L}(v)$ *and for every edge* $\{v_1, v_2\} \in E$*,* $\mathsf{col}_\mathsf{L}(v_1) \neq \mathsf{col}_\mathsf{L}(v_2)$ *or* NO *otherwise.*

Unlike k**-Coloring**, **ListColoring** does not offer the same set of colors for every vertex; instead, a *color-list function* L defines a "list" of possible colors $\mathsf{L}(v) \subseteq Z$ for each vertex v. Note, for historical reasons, what is called a "list" is actually a set. For consistency with the literature, we stick to the term list. Given a graph G and a color-list function L, we call a function col_L an L*-coloring* for G if col_L satisfies the condition described in Definition 2.2. We call L a *k-color-list function* if $|\mathsf{L}(v)| \geq k$, for all $v \in V$. Given a graph G, the smallest integer k, such that G is L-colorable for all k-color-list functions L, is called G's *list-chromatic number* and is denoted $\chi_l(G)$. The maximal degree of a graph gives us an upper bound for the list-chromatic number.

Lemma 2.1 *For every graph* G*,*

$$\chi_l(G) \leq 1 + \Delta(G)$$

and a greedy algorithm for graph coloring with polynomial runtime complexity finds an L*-coloring for* G *for every* $(1 + \Delta(G))$*-color-list function* L*.*

See [Chartrand and Zhang 2008] for a proof of Lemma 2.1 and the definition of a greedy algorithm for graph coloring.

ListColoring generalizes k**-Coloring** because a k**-Coloring** instance can be translated to a **ListColoring** instance by setting $Z = \{1, \ldots, k\}$ and $\mathsf{L}(v) = Z$, for every $v \in V$. Since a solution to the **ListColoring** problem can be checked in polynomial time, **ListColoring** is also **NP**-complete. Algorithm 1, called LCOL, solves **ListColoring** in exponential time in the size of the input. The following lemma, which we prove in Appendix A.1, states that given a graph G and a color-list function L, LCOL returns an L-coloring for G if and only if there exists an L-coloring for G.

Lemma 2.2 *Let a graph* $G = (V, E)$*, with* $|V| \geq 1$*, and a color-list function* $\mathsf{L} : V \to 2^Z$*, for a set* Z*, be given.*

- *Soundness: If* $\text{LCOL}(V, E, \mathsf{L})$ *returns a function* f*, then* f *is an* L*-coloring for* G*.*
- *Completeness: If there exists an* L*-coloring for* G*, then* $\text{LCOL}(V, E, \mathsf{L})$ *returns a function* f*.*

Algorithm 1: $\textsc{LCol}(V, E, \mathsf{L})$

Input: $|V| \geq 1$, $E \subseteq \{e \subseteq V \mid 2 = |e|\}$, and $\mathsf{L} : V \to 2^Z$, for a set Z
Output: returns an L-coloring for (V, E) if it exists and No otherwise

```
1  if V = {v} then
2      if |L(v)| ≥ 1 then
3          let c ∈ L(v)
4          return {(v,c)}
5      else
6          return No
7  else
8      let v ∈ V
9      foreach c ∈ L(v) do
10         V' ← V \ {v}
11         E' ← {e ∈ E | v ∉ e}
12         L' ← {(v', L(v')) | v' ∈ V', {v,v'} ∉ E} ∪
                  {(v', L(v') \ {c}) | v' ∈ V', {v,v'} ∈ E}
13         r ← LCol(V', E', L')
14         if r ≠ No then
15             return r ∪ {(v,c)}
16     return No
```

2.4 Business Process Modeling Notation

We introduce a subset of the *Business Process Modeling Notation (BPMN)* [OMG 2011a] that we later extend to visually model authorization constraints for workflows. BPMN defines graphical elements for modeling workflows at a high level of abstraction. A workflow model is called a *process* in BPMN. In order to differentiate BPMN processes and CSP processes, we use the term *process* for CSP processes and write explicitly *BPMN process* to refer to the concept of a process in BPMN. In this article we consider BPMN processes that are composed of the six kinds of modeling elements shown in Figure 2.1.

The BPMN standard [OMG 2011a] describes its modeling elements and their relationships in a meta-model using UML class diagrams [OMG 2011b]. Figure 2.2 shows an extract of this meta-model with the classes relevant to our BPMN extension presented in Chapter 8. We only explain the classes depicted in white for now, and return to the gray classes in Section 8.2. Italic class names denote abstract classes. Based on the meta-model, the BPMN standard

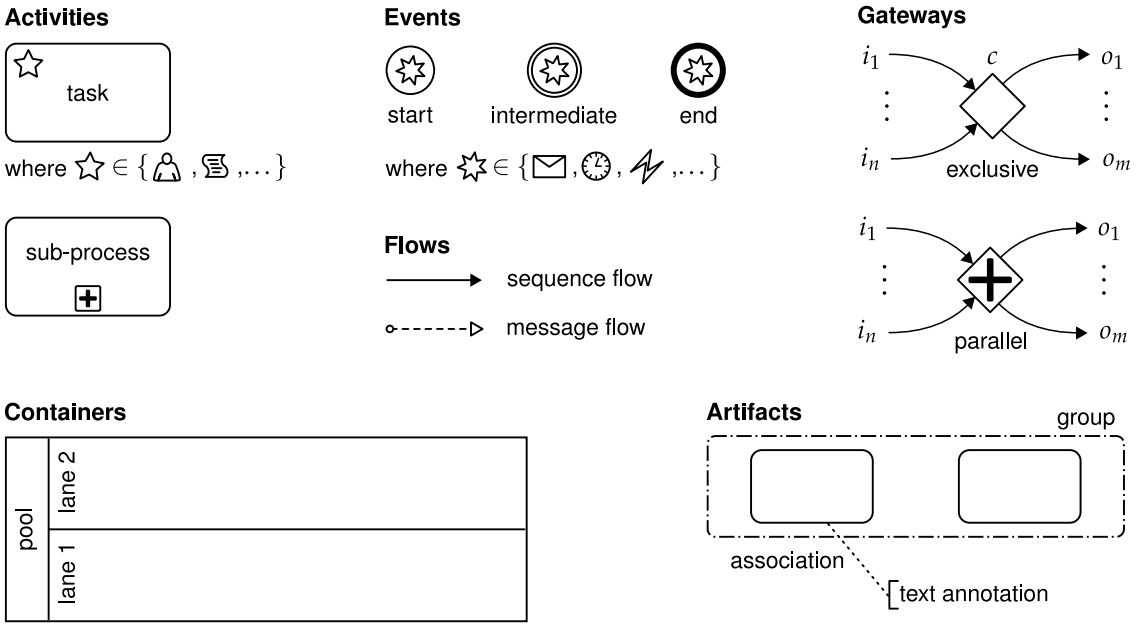


Figure 2.1. BPMN modeling elements

also specifies an XML [W3C 2011] serialization for BPMN processes, which provides software vendors with a tool-independent interchange format for BPMN models. In order not to dive too deeply into XML details, we describe our BPMN extension only in terms of the meta-model. Its mapping to XML Schema [W3C 2010] is straightforward. In the following, we introduce the modeling elements shown in Figure 2.1 and reference the corresponding meta-model classes given in Figure 2.2 in sans-serif font, *e.g.* Event.

BPMN calls a unit of work an *activity* (Activity). We consider two kinds of activities in this dissertation: *tasks* (Task) and *sub-processes* (SubProcess). Tasks are visualized by rectangles with rounded corners, labelled with the name of the task. A small icon in the upper left corner may specify the task's type. For example, an icon depicting a script visualizes tasks that model the execution of some code. We consider mostly tasks that are executed by humans, called *user tasks* (UserTask), visualized by an icon depicting a person. Sub-processes are visualized by rectangles with rounded corners, a small boxed "+"-symbol at the bottom, and which are labelled by the name of the sub-process. A sub-process models a BPMN process that constitutes part of the parent BPMN process. The refinement of a BPMN process into sub-processes is a powerful means to model workflows at different levels of abstraction.

An *event* (Event) models the occurrence of a condition or an interaction with the environment. Events are circle-shaped and their boundary indicates whether their occurrence triggers a workflow instantiation, called a *start event*, whether

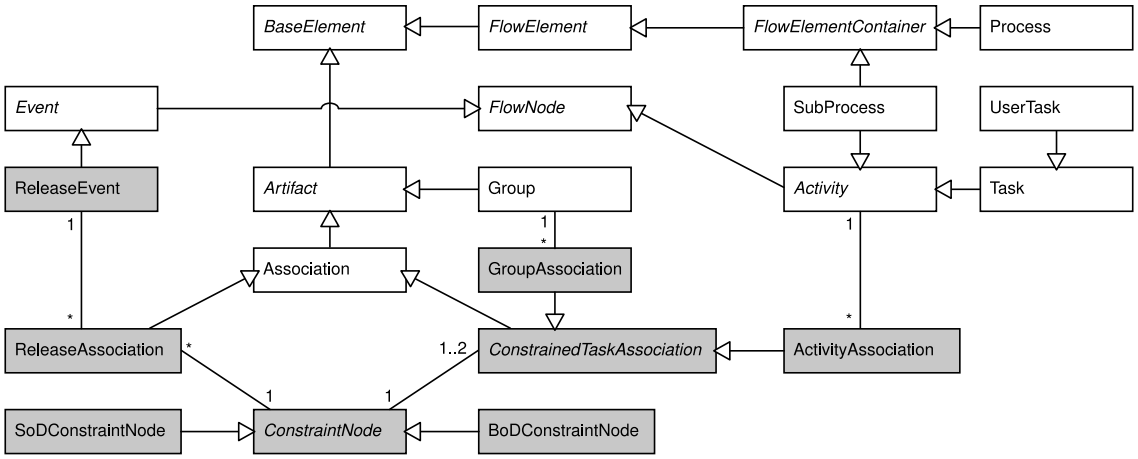


Figure 2.2. Extract of BPMN meta-model in white and our extensions in gray

they occur during the workflow's execution, called an *intermediate event*, or whether their occurrence terminates a workflow instance, called an *end event*. Furthermore, an event's interior may contain an icon, which determines the event's type. Examples are the arrival of a message or the expiration of a deadline, illustrated by an envelope and a clock, respectively.

Flows describe the causal and temporal dependencies between modeling elements. A *sequence flow*, illustrated by a solid line with an arrow, defines the order in which tasks are executed and events occur. Message-based communication is modeled by *message flows*, visualized by a dashed line with a circle at the sender's end and an arrow at the recipient's end. Sequence flows and message flows together determine a workflow's control-flow.

Merging and branching of the control-flow is modeled by *gateways*. A gateway has $n \geq 1$ incoming and $m \geq 1$ outgoing sequence flows. *Exclusive gateways* are depicted by an empty (or with an "x" labeled) diamond. Whenever the control-flow reaches an exclusive gateway on an incoming sequence flow, it passes on the control-flow immediately to exactly one of the m outgoing sequence flows, based on the evaluation of the condition c associated with the gateway. *Parallel gateways* are illustrated by a diamond labeled with the symbol "+". They synchronize the control-flow on the n incoming sequence flows and spawn the concurrent execution on the m outgoing sequence flows.

We distinguish two *containers* for partitioning BPMN processes. *Pools* are typically used to model organizational entities participating in a workflow's execution or they are simply used to demarcate a model's main BPMN process. Pools may be further subdivided into *lanes*, called *swimlanes* in other workflow modeling languages. Control-flow within pools is typically modeled by sequence flows and control-flow across pools may only be defined in terms of message flows. A

BPMN process's environment, in particular the source of input messages and the recipient of its return values, is often modeled by an empty pool [Silver 2009].

Modeling elements for annotations are called *artifacts* (Artifact). Sets of tasks are defined by placing them in a dot-dashed box, called a *group* (Group). Textual annotations as visualized by a dotted line, called an *association* (Association), and a half-open box containing text.

2.5 Integer Programming

Let $m,n \in \mathbb{N}$. We specify by $\mathbf{A} \in \mathbb{R}^{m \times n}$ an m by n *matrix* $\mathbf{A}$ of real numbers. Furthermore, $\mathbf{b} \in \mathbb{R}^m$ is a *(column) vector* composed of m real numbers. Let $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$, and $\mathbf{x} \in \mathbb{Z}^n$. For $i \in \{1,\ldots,m\}$ and $j \in \{1,\ldots,n\}$, we refer to $\mathbf{A}$'s ith row vector as $\mathbf{a}_i$ and a_{ij} is the jth element in $\mathbf{a}_i$. Correspondingly, b_i is $\mathbf{b}$'s ith element. Moreover, $\mathbf{Ax}$ denotes matrix-vector multiplication resulting in a vector $\mathbf{d} \in \mathbb{R}^m$ and $\mathbf{c}^\mathrm{T}\mathbf{x}$ denotes vector multiplication $\sum_{j=1}^{n} c_j x_j$, where $\mathbf{c}^\mathrm{T}$ is $\mathbf{c}$'s *transposed*. For $\mathbf{b},\mathbf{d} \in \mathbb{R}^m$, we write $\mathbf{d} \leq \mathbf{b}$ if for all $i \in \{1,\ldots,m\}$, $d_i \leq b_i$.

We now recall basic definitions from integer linear programming.

Definition 2.3 (The Integer Linear Programming (**ILP**) Problem)

Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, *and* $\mathbf{c} \in \mathbb{R}^n$, *for* $m,n \in \mathbb{N}$.

Output: $\min_{\mathbf{x} \in \mathbb{Z}^n} \{\mathbf{c}^\mathrm{T}\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}\}$ *or* No *if the set is empty.*

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{c} \in \mathbb{R}^n$ be an **ILP** instance, and let $i \in \{1,\ldots,m\}$ and $j \in \{1,\ldots,n\}$. We may refer to the output corresponding to the input $(\mathbf{A},\mathbf{b},\mathbf{c})$ as $\mathbf{ILP}(\mathbf{A},\mathbf{b},\mathbf{c})$. A variable x_j is called a *decision variable* and $\mathbf{c}^\mathrm{T}\mathbf{x}$ is called the *objective function*. Note that $\mathbf{Ax} \leq \mathbf{b}$ can be decomposed into m inequalities of the form $\mathbf{a}_i\mathbf{x} = \sum_{j=1}^{n} a_{ij}x_j \leq b_i$, each called a *constraint*. If $\mathbf{x}$ satisfies $\mathbf{Ax} \leq \mathbf{b}$, *i.e.* all m constraints, it is called a *feasible solution*. If there exists no feasible solution for a given **ILP** instance, then the instance is *infeasible*. A feasible solution that minimizes the objective function with respect to all feasible solutions is an *optimal (feasible) solution*.

It is common practice to use shorthand notation for constraints. For example, the equality $\mathbf{a}_i\mathbf{x} = b_i$ is equivalent to the two constraints $\mathbf{a}_i\mathbf{x} \leq b_i$ and $-\mathbf{a}_i\mathbf{x} \leq -b_i$. If variables are not defined, they are implicitly assumed to be zero. For example, the constraint $a_{i1}x_1 + a_{i2}x_2 + a_{i3}x_3 \leq b_i$ is equivalent to $\mathbf{a}_i\mathbf{x} \leq b_i$ where $a_{i4} = \ldots = a_{in} = 0$.

Integer linear programming is a specialization of linear programming in that decision variables assume only values from $\mathbb{Z}$ and not from $\mathbb{R}$. This is necessary for modeling situations where only a discrete set of states is possible. However,

this restriction has substantial algorithmic implications that are outside the scope of this paper. We simply note that **ILP** is **NP**-complete [Schrijver 1998].

Chapter 3

Requirements

In this chapter, we present requirements for workflow models and their related authorizations originating from regulations, standards, and best-practice frameworks. We thereby embed our work in a legislative, organizational, and business environment and emphasize its practical relevance.

3.1 Workflow Models

Frameworks that provide organizations guidance in the specification and documentation of their business activities vary with respect to the level of detail of their workflow models and differ in how domain-specific they are. Based on these differences, we classify them in three groups, summarized in Table 3.1.

General Business Activity Frameworks	MIT Process Handbook Process Classification Framework (PCF)
Domain-Specific Business Activity Frameworks	Process Reference Model for IT (PRM-IT) IT Infrastructure Library (ITIL) Control Objectives for Information and Related Technology (COBIT) Supply Chain Operations Reference (SCOR)
Workflow Template Frameworks	IBM Tivoli Unified Processes (ITUP) Insurance Application Architecture (IAA) Information FrameWork (IFW)

Table 3.1. Best-practice frameworks for business activities, categorized with respect to their domain-specificity and the level of details of their workflow models.

3.1.1 General Business Activity Frameworks

General business activity frameworks, such as the *MIT Process Handbook* [Herman and Malone 2003] and the *Process Classification Framework (PCF)* [APQC 2009], serve as comprehensive blueprints of an organization's activities and processes. Since such frameworks encompass various kinds of activities, their models are abstract and generic.

For example, the MIT Process Handbook structures business activities using a *part-of* relationship. The activities Evaluate Suppliers, Manage Supplier Policies, and Manage Supplier Relationships, for instance, are part of the activity Manage Suppliers, which in return is part of the activity Buy. The activity Evaluate Suppliers is not further decomposed into sub-activities and the *part-of* relationship abstracts away from control-flow and data-flow.

3.1.2 Domain-Specific Business Activity Frameworks

In contrast to general business activity frameworks, domain-specific frameworks specify workflows at greater level of detail. An example is the *Supply Chain Operations Reference (SCOR)* [SCC 2010], which defines activities to execute and manage supply chains.

Particularly interesting for our work are frameworks that specify activities to design and manage IT systems. They not only specify business activities and related workflows, but also propose protection mechanisms. Examples of these frameworks are the *Process Reference Model for IT (PRM-IT)* [IBM 2007], the *IT Infrastructure Library (ITIL)* [OGC 2010], and *Control Objectives for Information and related Technology (COBIT)* [ITGI 2005]. Some of these models, such as PRM-IT, were developed by companies to support their consulting activities. Others, such as ITIL, are proposed by governmental bodies. Finally, some originate from non-profit organizations. In particular, COBIT, which we present in more detail in Section 3.2.3, has been developed by ISACA, an association for IT governance professionals. The common goal of these frameworks is to standardize a baseline for the specification of IT requirements and thereby improve the quality and lower the cost of IT systems and IT projects.

Even though workflows of domain-specific frameworks are more refined than those of general business activity frameworks, they are still abstract and their implementation requires additional refinement steps. For example, PRM-IT defines the workflow *security management*, which encompasses tasks such as Plan Security Practices, Apply Security Protection Mechanisms, and Operate Security Protection Mechanisms. A workflow's control-flow and data-flow is defined in terms of the dependencies of its tasks' input and output parameters, *e.g.* access

control lists are an output of the task Apply Security Protection Mechanisms and an input of Operate Security Protection Mechanisms.

3.1.3 Workflow Template Frameworks

Workflow template frameworks specify yet more detailed workflows than domain-specific business activity frameworks. Typically sold alongside consulting services that instantiate and implement the respective templates, these frameworks additionally specify data types and organizational roles. They thereby simplify the integration of software and services from different vendors and improve interoperability of systems maintained by different organizations.

Instead of giving an example here, we refer to the collateral evaluation workflow presented in Example 8.1, which is based on a template from the *Information FrameWork (IFW)* [IBM 2010b], a collection of templates used by the financial industry. Further workflow template frameworks are the *Insurance Application Architecture (IAA)* [IBM 2009] and the *IBM Tivoli Unified Processes (ITUP)* [IBM 2010a]. Some workflow template frameworks come with mappings to related domain-specific frameworks, which are used to show how the implementation of the former corresponds to the conformance with the latter. For example, ITUP specifies how its workflow templates correspond to activities of COBIT and PRM-IT.

3.1.4 Concrete Workflows

To show that our results are applicable across multiple domains, we use example workflows from different sources, modeling different business objectives. In addition to the collateral evaluation workflow mentioned above, we present in Example 4.1 a workflow that models the dispensation of drugs in a hospital, which was analyzed throughout the European research project MASTER [Marino *et al.* 2009], and in Example 10.1 a workflow modelling the payment of electronic invoices as sketched in a report on the harmonization of electronic payment processes in the European Union [EGEI 2009].

3.2 Refining Regulations

We illustrate the refinement of regulations to control mechanisms by means of an example, based on SOX compliance. Figure 3.1 visualizes the refinement steps, which we describe in the following.

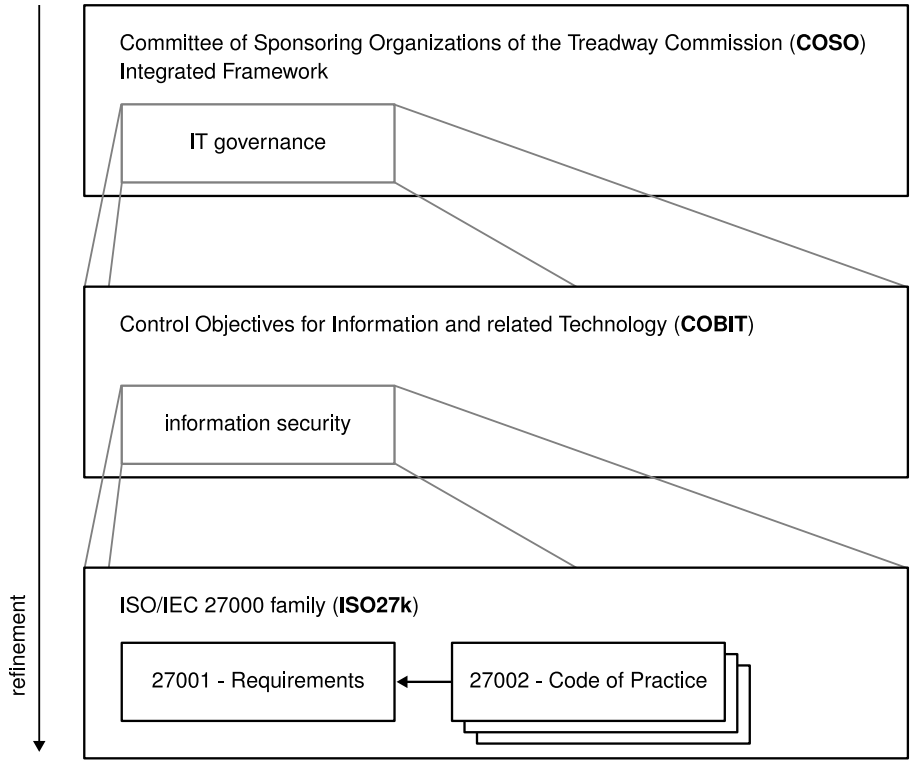


Figure 3.1. Refinement of regulations to control mechanisms

3.2.1 Legislation

In the wake of a series of corporate scandals, *e.g.* as reported by [The Economist 2001], the U.S. Congress enacted the *Sarbanes-Oxley (SOX) Act* [SOX 2002], which mandates publicly traded companies to exercise stricter financial governance. Most interesting to our work is that SOX requires a well-documented and agile financial reporting environment including effective control mechanisms that prevent fraud and errors (SOX sections 404 and 409).

The enactment of SOX had negative economical consequences for effected companies [Zhang 2007], and some authors even claim that SOX led to an increase in delistings [Engel *et al.* 2007]. This emphasizes that SOX compliance is a pressing issue and that a simplification and automation of this process is of great value to many companies.

3.2.2 General Control Framework

The U.S. Securities and Exchange Commission (SEC) supports companies in the interpretation and implementation of SOX. For example, it acknowledges [SEC 2003] the adoption of the *Internal Control – Integrated Framework (ICIF)* of the

Committee of Sponsoring Organizations of the Treadway Commission (COSO) [COSO 2011] as means to comply with SOX. Various domain-specific business activity frameworks (see Section 3.1.2) refine the IT governance activities specified in the ICIF. We focus in the following section on COBIT, which contains a detailed mapping between its activities and those specified in the ICIF.

3.2.3 IT Governance

COBIT [ITGI 2005] provides organizations guidance in planning, building, executing, and monitoring their IT systems. It specifies about three dozen activities whose execution aims at aligning business processes and IT systems, clarifying responsibilities and ownership, identifying critical resources, defining and enforcing protection mechanisms, and establishing controls that enable management to evaluate the effectiveness of these mechanisms. We list in the following three of these activities and link them to results presented in forthcoming chapters.

- **Define the IT Processes, Organization, and Relationships** (activity PO4)**:** This activity connects the modeling of business processes as workflows and the specification of authorizations, the overarching topic of our work. In particular, recommended sub-activities of PO4 are the establishment of roles and responsibilities and the segregation of duties, *i.e.* the specification and enforcement of SoD. We address these requirements with the use of RBAC, SoDA, and the introduction of the novel SoD constraints, presented in Part III.

- **Manage IT Human Resources** (activity PO7)**:** This activity encompasses the timely reflection of job changes and terminations in authorizations policies, while also guaranteeing the continuity of business operations. In Part II, our formalization of authorization-constrained workflows and its implementation account for organizational changes during workflow execution. Furthermore, we present in Chapter 10 algorithms to compute optimal authorization changes that enable an obstruction-free workflow execution.

- **Ensure System Security** (activity DS5)**:** This activity is decomposed into various security measures. Interesting for our work is the requirement to manage users and their authorizations in line with a system's business objectives, the main topic of Part III. COBIT refers to ISO 27k, which we present in the following section, for concrete security mechanisms.

3.2.4 Information Security Standard

ISO 27k refers to a set of standards, which together specify requirements and guidelines for the implementation, operation, and auditing of a so-called *Information Security Management System (ISMS)* [ISO 2009]. ISO 27002 [2005b] lists control mechanisms that an ISMS must implement, depending on the security requirements identified with the processes specified in ISO 27001 [2005a]. ISO 27002 defines concrete protection mechanisms in areas such as network and operation system security. However, with respect to our work, only a few recommendations are more specific than those in COBIT. For example, ISO 27002 says that the activation and deactivation of users should include "checking that the level of access granted is appropriate to the business purpose [...] and is consistent with [the] organizational security policy, *e.g.* it does not compromise segregation of duties" and furthermore "immediately removing or blocking access rights of users who have changed roles or jobs or left the organization". Our contributions listed in Chapter 1 address these requirements directly.

The examples presented in this chapter illustrate that a workflow model and its authorizations are typically the result of a sequence of refinement steps. This observation motivates our work in Part II.

Part II

A Workflow-Independent Approach

Chapter 4

Authorization-Constrained Workflows

We start with an informal introduction to the life cycle of authorization-constrained workflows. Afterward, we use CSP to formalize the underlying concepts. We shall see that CSP's notion of parallel, synchronized process execution facilitates a concise description of workflow systems that are composed from multiple sub-processes, each modeling a separate system aspect. Furthermore, its notion of renaming allows a mapping between different levels of abstraction of a workflow. Many definitions given in this chapter are, with minor adjustments, also used in Part III.

We call an atomic unit of work a *task*. A *workflow* models causal dependencies between a set of tasks, whose execution constitutes a business objective. An alternative name for workflow is *business process*. Although we prefer the term *workflow* as it avoids further overloading the term *process*.

We distinguish two phases in a workflow's life cycle. At *design time*, a business expert designs a workflow using a modeling language such as BPMN and afterward deploys it to a *workflow engine*. At *run time*, the workflow engine executes the workflow. We call a workflow execution a *workflow instance*. A workflow engine may execute multiple instances of the same workflow in parallel. According to the workflow's control-flow and depending on the evaluation of gateway conditions, a workflow engine schedules and instantiates tasks, called *task instances*, during workflow execution. Standard workflow modeling languages, such as BPMN, allow the specification of parallel and conditional execution and loops. Therefore, there may be zero or more instances of the same task in one workflow instance. Depending on a task's type, its instances are executed by humans, by a software program, through the invocation of a web service, *etc.* In this dissertation, we focus on tasks whose instances are executed by humans, either directly, *e.g.* by completing a form, or indirectly, *e.g.* by executing a program on their behalf. An *authorization* specifies whether or not a user is allowed to execute a task instance.

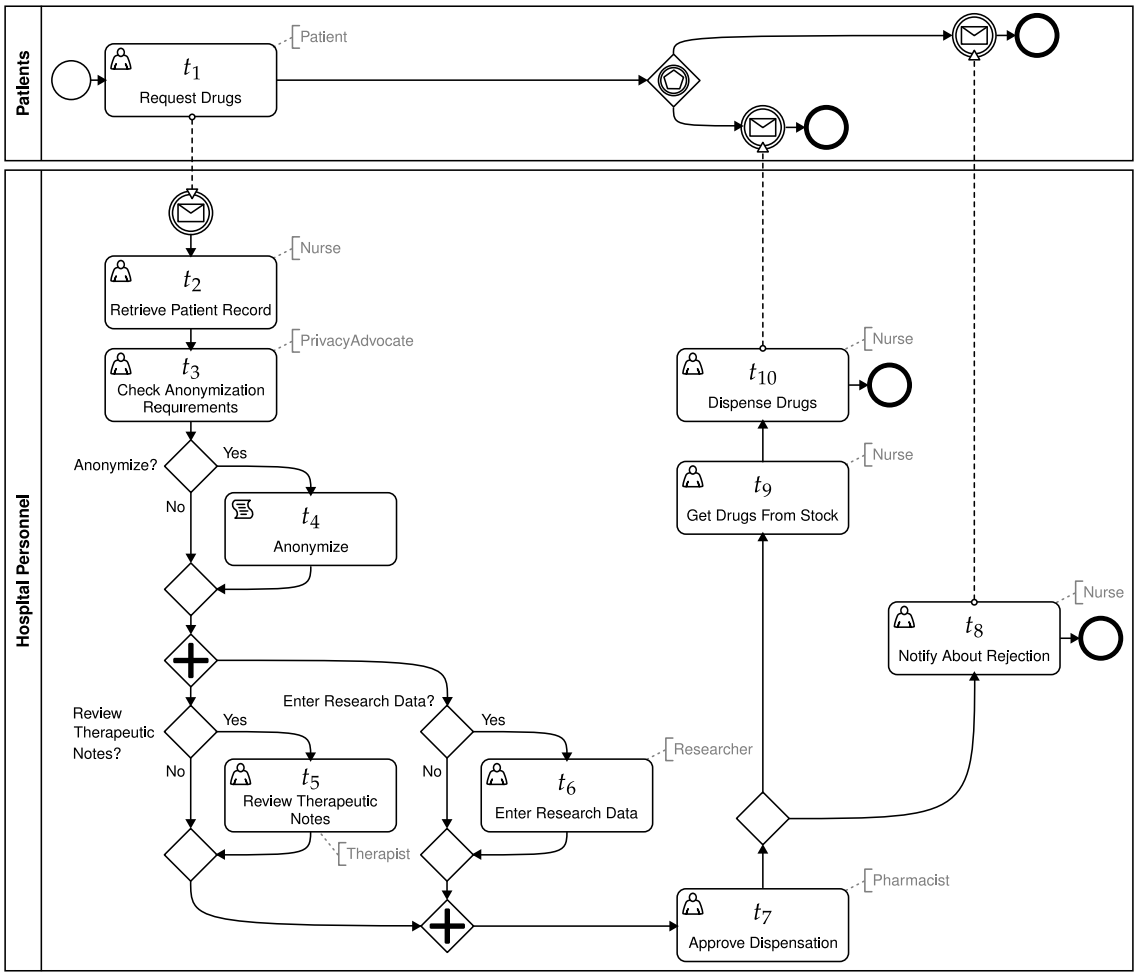


Figure 4.1. BPMN model of drug dispensation workflow

4.1 Workflows

There are numerous translations from BPMN and similar workflow modeling languages to process calculi. We present some of them in Chapter 12. The technical differences are unimportant for our work here and we use a straightforward translation to CSP, illustrated in forthcoming examples.

For the remainder of this dissertation, let $\mathcal{T}$ be a set of *tasks* and $\mathcal{U}$ a set of *users*. For a task t and a user u, the CSP event $t.u$ models the execution of an instance of t by u. We call $t.u$ a *(task) execution event*. Let $\mathcal{X} = \{t.u \mid t \in \mathcal{T}, u \in \mathcal{U}\}$ be the set of all execution events. We then model a workflow as follows.

Definition 4.1 (Workflow Execution Process) *A* workflow execution process *is a process W such that $\mathsf{T}(W) \subseteq \mathcal{X}^{*\checkmark}$.*

In other words, a workflow execution process may engage in an arbitrary number of execution events and finally the event ✓.

Definition 4.2 (Workflow Trace) *Let W be a workflow execution process. A trace $i \in \Sigma^{*\checkmark}$ is a* workflow trace of W *if* $(i \restriction \mathcal{X}^{\checkmark}) \in \mathsf{T}(W)$.

A workflow trace i models a workflow instance. Note that i may not only include execution events. We will subsequently introduce administrative events that model complementary activities taking place during workflow execution. However, in order to be a workflow trace of a workflow execution process W, only the execution events in i must be a trace of W; hence the restriction $i \restriction \mathcal{X}^{\checkmark}$. The workflow instance modeled by i has *successfully terminated* if $\checkmark \in i$.

Given a trace i, the auxiliary function users returns the multiset of users contained in execution events in i.

$$\mathsf{users}(i) = \begin{cases} \varnothing & \text{if } i = \langle\rangle, \\ \{\!\{u\}\!\} \uplus \mathsf{users}(i') & \text{for } i = \langle t.u \rangle^\frown i' \text{ and } t.u \in \mathcal{X}, \\ \mathsf{users}(i') & \text{for } i = \langle \sigma \rangle^\frown i' \text{ and } \sigma \notin \mathcal{X}. \end{cases}$$

We illustrate these definitions with the *drug dispensation workflow* from [Marino *et al.* 2009]. This workflow serves also as a running example for the remainder of Part II.

Example 4.1 (Drug Dispensation Workflow) A BPMN model[1] of the drug dispensation workflow is shown in Figure 4.1. Ignore the gray annotations for the moment. This workflow defines the tasks that must be executed to dispense drugs to patients within a hospital. The drugs dispensed are either in an experimental state or very expensive and therefore require special diligence.

For this example, let $\mathcal{T} = \{t_1, \ldots, t_{10}\}$, where t_1 refers to Request Drugs, t_2 to Retrieve Patient Record, *etc.*, and $\mathcal{U} = \{\mathsf{Alice}, \mathsf{Bob}, \mathsf{Claire}, \mathsf{Dave}, \mathsf{Emma}, \mathsf{Fritz}, \mathsf{Gerda}\}$. An instance of the drug dispensation workflow is started by a user acting as patient who requests drugs by handing his prescription to the hospital. The hospital personnel first retrieves the patient's record from the hospital's database and afterward determines whether the respective data must be anonymized. If anonymization is required, this is done by a computer program. We ignore this task in our forthcoming discussion as we focus on human tasks. Next, the hospital personnel reviews therapeutic notes if contained in the prescription and adds in parallel research-related data to the record if the requested drugs are in an experimental state. Finally, the dispensation is either approved, the drugs

[1]The gateway inside the BPMN process Patients is an exclusive gateway. It passes on the control-flow to the sequence flow that points to the event that occurs first after the control-flow has reached the gateway.

are collected from the stock, and handed to the patient, or the dispensation is denied and the patient is informed accordingly.

We model the drug dispensation workflow in CSP by the workflow execution process

$$
\begin{aligned}
W &= t_1.u_1 : \mathcal{U} \to t_2.u_2 : \mathcal{U} \to t_3.u_3 : \mathcal{U} \to \big((W_1 \mid\mid\mid W_2)\,;\, W_3\big) \\
W_1 &= \mathit{SKIP} \sqcap (t_5.u_5 : \mathcal{U} \to \mathit{SKIP}) \\
W_2 &= \mathit{SKIP} \sqcap (t_6.u_6 : \mathcal{U} \to \mathit{SKIP}) \\
W_3 &= t_7.u_7 : \mathcal{U} \to \big((t_8.u_8 : \mathcal{U} \to \mathit{SKIP}) \sqcap (t_9.u9 : \mathcal{U} \to t_{10}.u_{10} : \mathcal{U} \to \mathit{SKIP})\big)\ .
\end{aligned}
$$

Because we do not model data-flow, we over-approximate gateway decisions, such as whether therapeutical notes must be reviewed, with CSP's internal choice operator $\sqcap$. Note that the tasks in W are fixed, *i.e.* t_1 corresponds to the task Request Drugs, whereas the users are not fixed, *i.e.* for every execution event the respective user can be chosen freely from $\mathcal{U}$. Consider now the traces

$$
i_1 = \langle t_1.\mathsf{Fritz}, t_3.\mathsf{Fritz}, t_5.\mathsf{Bob}, \checkmark \rangle \quad \text{and} \quad i_2 = \langle t_1.\mathsf{Fritz}, t_2.\mathsf{Emma}, t_3.\mathsf{Fritz}, t_5.\mathsf{Bob} \rangle\ .
$$

The trace i_1 is not a workflow trace of W because $(i_1 \restriction \mathcal{X}^{\checkmark}) \notin \mathsf{T}(W)$. However, i_2 is a workflow trace of W because $(i_2 \restriction \mathcal{X}^{\checkmark}) \in \mathsf{T}(W)$ and $\mathsf{users}(i_2) = \{\!\{\mathsf{Bob}, \mathsf{Emma}, \mathsf{Fritz}, \mathsf{Fritz}\}\!\}$ is the multiset of users who are executing task instances in i_2. ⋆

4.2 Authorization Classes and Enforcement Approach

We model three classes of authorization constraints in this dissertation:

- **Basic Authorizations:** This class encompasses all authorizations that restrict the execution of task instances to users with the necessary qualifications and responsibilities in a history-independent and workflow-independent manner. Examples are access control lists (ACLs) [Sandhu and Samarati 1994], the Bell-LaPadula (BLP) model [Bell and LaPadula 1973], and Role-based Access Control (RBAC) [Ferraiolo *et al.* 2001] without sessions. What is often called a *permission* in the context of basic authorizations corresponds here to the right to execute a task.

- **Separation of Duties (SoD):** Authorizations to execute task instances are restricted to ensure that different users execute those instances whose execution results in a conflict of interest. For example, consider two tasks t_1 and t_2 and suppose that their execution by the same user results in a conflict of interest. An SoD constraint is then used to prevent such a conflict

by not authorizing a user from executing an instance of t_2 after executing an instance of t_1 and *vice versa*.

- **Binding of Duties (BoD):** Authorizations to execute task instances are restricted based on who has executed previous task instances to limit the exposure of sensitive data and to reuse knowledge that users have gained from previous task executions. For example, consider two tasks t_1 and t_2, both revealing the same sensitive information. A BoD constraint may force a user to execute all instances of t_2 (and further instances of t_1) after having executed an instance of t_1 and *vice versa*.

As defined here, SoD and BoD constraints are history-dependent. Note that related work on SoD and BoD often uses the term *dynamic* for what we call *history-dependent* and *static* for *history-independent*. We implicitly subsume history-independent SoD and BoD by basic authorizations. A detailed comparison of our terminology and related work follows in Chapter 12.

We formalize authorized executions of task instances in terms of processes. More specifically, for each authorization policy ϕ, we define a process A_ϕ and say that a workflow trace i *satisfies* ϕ if $i \in \mathsf{T}(A_\phi)$. Given a workflow execution process W, we then describe the enforcement of ϕ on W by the parallel execution of A_ϕ and W, synchronized on execution events; formally, $W \underset{\mathcal{X}}{\parallel} A_\phi$.

4.3 Basic Authorizations

In the interest of ecumenical neutrality and supporting numerous authorization models, we first formalize basic authorizations abstractly as a relation $UT \subseteq \mathcal{U} \times \mathcal{T}$, called the *user-task assignment*. Later, we refine user-task assignments using roles. In particular, we use roles to model administrative activities in the remaining chapters of Part II and to model the cost associated with such activities in Chapter 10.

Given a user-task assignment UT, a user u is authorized to execute instances of a task t if $(u,t) \in UT$. Expressed as a process, we have the definition:

Definition 4.3 (Basic Authorization Process) *For a user-task assignment UT, a* basic authorization process *for UT is the process*

$$\begin{aligned} A_{UT} \;=\; & (t.u) : UT^{-1} \to A_{UT} \\ & \Box\, \sigma : (\Sigma \setminus \mathcal{X}) \to A_{UT} \\ & \Box\, \mathit{SKIP}\,. \end{aligned}$$

The process A_{UT} engages in every execution event $t.u$ if the user u is authorized to execute the task t with respect to UT. Furthermore, A_{UT} engages in every event σ that is not an execution event and it can terminate at any time. The history-independent nature of UT is reflected by the fact that A_{UT} behaves again like A_{UT} after engaging in every event (except the final event $\checkmark$).

For the remainder of this dissertation, let $\mathcal{R}$ be a set of *roles*. We now refine user-task assignments using the core idea of RBAC [Ferraiolo *et al.* 2001], namely the decomposition of a user-task assignment into two relations.

Definition 4.4 (RBAC Policy) *An* RBAC policy *is a tuple* (UR, RT)*, where* $UR \subseteq \mathcal{U} \times \mathcal{R}$ *is a* user-role assignment *and* $RT \subseteq \mathcal{R} \times \mathcal{T}$ *is a* role-task assignment .

Given an RBAC policy (UR, RT), we can derive a user-task assignment UT by composing RT and UR with the composition operator $\circ$. Formally, $UT = RT \circ UR = \{(u,t) \mid \exists r \in \mathcal{R}.(u,r) \in UR \text{ and } (r,t) \in RT\}$.

Let (UR, RT) be an RBAC policy and u a user. For a role r, we say that u *acts in the role* r if $(u,r) \in UR$. Furthermore, for a task t we say that u is *authorized to execute* t with respect to (UR, RT) if $(u,t) \in RT \circ UR$.

In contrast to the NIST RBAC standard [Ferraiolo *et al.* 2001], we omit the concept of sessions. This is without loss of generality as the activation and deactivation of roles within a session can be modeled by administrative activities that change RBAC policies as introduced below.

We model changes to an RBAC policy by a set of events $\mathcal{A} \subseteq \Sigma$ that we call the *administrative events*. For a user u and a role r, the administrative event $\mathsf{add}.u.r$ (respectively $\mathsf{rm}.u.r$) models the addition (respectively the removal) of (u,r) from the user-role assignment. We do not consider administrative events that change role-task assignments. This design decision is due to the observation that user-role assignments and the availability of users in general changes much more frequently in practice than workflow and role models.

Having introduced all the kinds of events that we need in Part II, specifically, $\Sigma = \mathcal{X} \cup \mathcal{A}$, we now specify the evolution and enforcement of an RBAC policy in terms of a process.

Definition 4.5 (RBAC Process) *For an RBAC policy* (UR, RT)*, the process*

$$\begin{aligned} RBAC(UR,RT) = {} & (t.u) : (RT \circ UR)^{-1} \to RBAC(UR,RT) \\ & \Box\ \mathsf{add}.u : \mathcal{U}.r : \mathcal{R} \to RBAC(UR \cup \{(u,r)\}, RT) \\ & \Box\ \mathsf{rm}.u : \mathcal{U}.r : \mathcal{R} \to RBAC(UR \setminus \{(u,r)\}, RT) \\ & \Box\ \mathit{SKIP} \end{aligned}$$

is called an RBAC process.

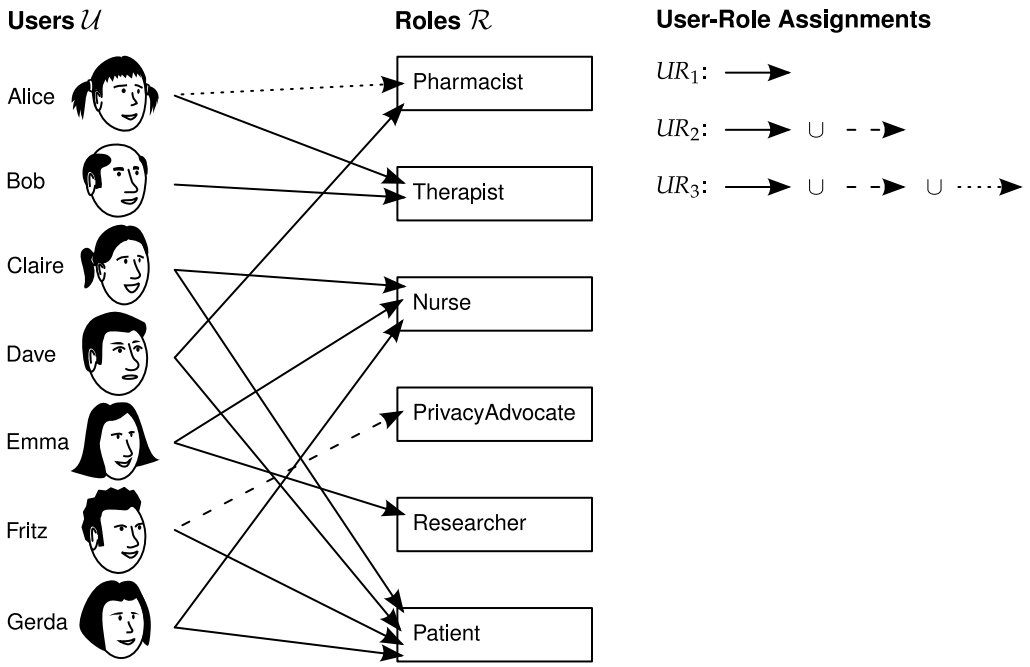


Figure 4.2. User-role assignments for the drug dispensation workflow

An RBAC process is parametrized by an RBAC policy (UR, RT) and engages in every execution event $t.u$ if u is authorized to execute t with respect to (UR, RT). Furthermore, an RBAC process changes its user-role assignment by engaging in administrative events and may terminate at any time. We now compose a workflow execution process with an RBAC process.

Definition 4.6 (Secure Workflow Process) *For a workflow execution process W and an RBAC policy (UR, RT), we call the process*

$$SW(UR, RT) = W \underset{\mathcal{X}}{\parallel} RBAC(UR, RT)$$

a secure (workflow) process.

Like an RBAC process, a secure process $SW(UR, RT)$ is parametrized by an RBAC policy. $SW(UR, RT)$ engages in every execution event $t.u$ if W engages in $t.u$, *i.e.* if the workflow foresees the execution of the respective task instance, and u is authorized to execute t with respect to (UR, RT). By synchronizing only on execution events, arbitrary administrative events can be interleaved with execution events in any order. Thus, the RBAC policy can change during a workflow's execution.

Example 4.2 (Secure Workflow Process) Consider again the drug dispensation workflow presented in Example 4.1. Figure 4.2 shows the set of users $\mathcal{U}$ introduced in Example 4.1 and additionally the set of roles $\mathcal{R}$ that we use throughout

this example. Let (UR_1, RT) be the initial RBAC policy. The user-role assignment UR_1 is depicted in Figure 4.2, ignoring the dashed and dotted lines between users and roles, *e.g.* $(\mathsf{Alice}, \mathsf{Therapist}) \in UR_1$ and $(\mathsf{Alice}, \mathsf{Pharmacist}) \notin UR_1$. We specify the role-task assignment RT by means of BPMN annotations in Figure 4.1. For example, only users acting in the role Nurse are authorized to execute t_2 with respect to (UR_1, RT). We assigned only one role to each task but in general tasks can be annotated with sets of roles.

Let $SW(UR_1, RT)$ be the secure process for W and (UR_1, RT). We concluded in Example 4.1 that $i_2 = \langle t_1.\mathsf{Fritz}, t_2.\mathsf{Emma}, t_3.\mathsf{Fritz}, t_5.\mathsf{Bob}\rangle$ is a workflow trace of W because $(i_2 \restriction \mathcal{X}^{\checkmark}) \in \mathsf{T}(W)$. However, i_2 is not a trace of $SW(UR_1, RT)$ because Fritz is not a PrivacyAdvocate and therefore not authorized to execute t_3 with respect to (UR_1, RT).

Consider now the trace $i_3 = \langle t_1.\mathsf{Fritz}, t_2.\mathsf{Emma}, \mathsf{add.Fritz.PrivacyAdvocate}, t_3.\mathsf{Fritz}, t_5.\mathsf{Bob}\rangle$. This trace is similar to i_2 but includes the administrative event add.Fritz.PrivacyAdvocate. By engaging in this event, the user-role assignment UR_1 becomes $UR_2 = UR_1 \cup \{(\mathsf{Fritz}, \mathsf{PrivacyAdvocate})\}$. Because Fritz is authorized to execute t_3 with respect to (UR_2, RT), i_3 is a trace of $SW(UR_1, RT)$. With respect to execution events, i_3 is equal to i_2, *i.e.* $i_3 \restriction \mathcal{X}^{\checkmark} = i_2 \restriction \mathcal{X}^{\checkmark}$, and therefore i_3 is also a workflow trace of W. ★

Chapter 5

Generalization of SoDA

Our workflow-independent approach to bridging the gap between the abstract specification of SoD constraints and their enforcement in a dynamic enterprise workflow environment builds on Li and Wang's *Separation of Duty Algebra (SoDA)* [2008]. We first present a small motivational example that demonstrates SoDA's quantitative and qualitative expressivity. Afterward, we introduce SoDA's syntax, generalize its original semantics, and finally map SoDA terms to CSP processes.

Consider the SoD policy that requires the involvement of a user other than Bob that acts in the role of a Manager and one or two additional users, acting as an Accountant and a Clerk. Using SoDA, this policy can be modeled by the term

$$(\mathsf{Manager} \sqcap \neg\{\mathsf{Bob}\}) \otimes (\mathsf{Accountant} \odot \mathsf{Clerk}).$$

The term's left side is satisfied by any Manager other than Bob. Under the semantics of the $\odot$-operator, the right side is satisfied by a single user that acts as an Accountant and a Clerk or by two users, provided one of them acts as an Accountant and the other as a Clerk. Finally, the $\otimes$-operator requires that the users in the two parts are disjoint. It thereby separates their duties. As this example shows, SoDA terms specify both the number and kinds of users who must execute task instances in a workflow, though independent of the details of the workflow itself. Separating concerns this way facilitates a loose coupling between an application's business logic and its security constraints and is therefore well-suited to formalize abstract, workflow-independent authorization requirements.

5.1 Syntax

We present below the syntax of SoDA terms.

Definition 5.1 (SoDA Grammar) *A* SoDA grammar $\mathfrak{S}$ *with respect to a set of users* $\mathcal{U} = \{u_1, \ldots, u_n\}$ *and a set of roles* $\mathcal{R} = \{r_1, \ldots, r_m\}$ *is a quadruple* (N, T, P, S) *where:*

- $N = \{S, CT, UT, AT, UR, U, R\}$ *is the set of nonterminal symbols,*
- $T = \{',', (,), \{, \}, \otimes, \odot, \sqcup, \sqcap, {}^{+}, \neg, \mathsf{All}\} \cup \mathcal{U} \cup \mathcal{R}$ *are the terminal symbols,*
- *the set of productions* $P \subseteq (\, N \times (N \cup T)^* \,)$ *is given by:*

$$\begin{aligned} S &::= CT \mid UT \\ CT &::= (CT \sqcup S) \mid (CT \sqcap S) \mid (S \otimes S) \mid (S \odot S) \mid (UT)^+ \\ UT &::= AT \mid (UT \sqcap UT) \mid (UT \sqcup UT) \mid \neg UT \\ AT &::= \{UR\} \mid R \mid \mathsf{All} \\ UR &::= U \mid U, UR \\ U &::= u_1 \mid \ldots \mid u_n \\ R &::= r_1 \mid \ldots \mid r_m \end{aligned}$$

- *and* $S \in N$ *is the start symbol.*

The terminal symbols $\otimes$, $\odot$, $\sqcup$, $\sqcap$, $^{+}$, and $\neg$ are called *operators*. Without loss of generality, we omit the productions $CT ::= (S \sqcap CT)$ and $CT ::= (S \sqcup CT)$. Li and Wang showed in [2008] that $\sqcap$ and $\sqcup$ are commutative with respect to their semantics and this is also the case for our semantics. Therefore, each term that could be constructed with these additional productions can be transformed to a semantically equivalent term constructed without them.

Let $\rightarrow^1_{\mathfrak{S}} \in (N \cup T)^+ \times (N \cup T)^*$ denote one derivation step of $\mathfrak{S}$ and $\rightarrow^*_{\mathfrak{S}}$ the transitive closure of $\rightarrow^1_{\mathfrak{S}}$. We call an element of $\{s \in T^* \mid S \rightarrow^*_{\mathfrak{S}} s\}$ a *term*. Furthermore, we call an element of $\{s \in T^* \mid AT \rightarrow^*_{\mathfrak{S}} s\}$ an *atomic term*. These are either a non-empty set of users, *e.g.* $\{\mathsf{Alice}, \mathsf{Bob}\}$, a single role, *e.g.* Clerk, or the keyword All. We call an element of $\{s \in T^* \mid UT \rightarrow^*_{\mathfrak{S}} s\}$ a *unit term*. These terms do not contain the operators $\otimes$, $\odot$, and $^{+}$. Finally, a *complex term* is an element of $\{s \in T^* \mid CT \rightarrow^*_{\mathfrak{S}} s\}$. In contrast to unit terms, they contain at least one of the operators $\otimes$, $\odot$, or $^{+}$. For a term ϕ, we call a unit term ϕ_{ut} a *maximal unit term of* ϕ, if ϕ_{ut} is a unit term, a subterm of ϕ, and if there is no other unit term ϕ'_{ut} that is also a subterm of ϕ and ϕ_{ut} is a proper subterm of ϕ'_{ut}.

5.2 Multiset Semantics

Li and Wang define in [2008] the satisfaction of SoDA terms for *sets* of users. We refer to their semantics as SoDA$^{\mathsf{S}}$ and summarize it in Appendix B. SoDA$^{\mathsf{S}}$ allows for quantitative constraints where terms define how many different users must execute tasks in a workflow instance. However, SoDA$^{\mathsf{S}}$ does not capture how many tasks each of these users must execute. Consider the policy P that requires Bob to execute two tasks, modeled by the SoDA term $\phi = \{\mathsf{Bob}\} \odot \{\mathsf{Bob}\}$.

Under SoDA$^{\mathsf{S}}$, ϕ is satisfied by the set $\{\mathsf{Bob}\}$. There is no satisfactory mapping of ϕ to a process that accepts all workflow traces that correspond to satisfying assignments of ϕ. If we define the correspondence between sets and workflow traces in a way that $\{\mathsf{Bob}\}$ maps to the set of traces containing *exactly one* execution event involving Bob, this would not satisfy P. Alternatively, if we map $\{\mathsf{Bob}\}$ to the set of traces containing *arbitrarily many* execution events involving Bob, this set would also include traces that do not satisfy P, for example, the trace containing three execution events involving Bob. The problem is that sets of users are too abstract: users cannot be repeated and hence information is lost on how many tasks a user (here Bob) must execute.

To address this problem, we introduce a novel semantics, SoDA$^{\mathsf{M}}$, that defines term satisfaction based on *multisets* of users. SoDA$^{\mathsf{M}}$ allows us to make finer distinctions concerning repetition (quantification requirements) than in SoDA$^{\mathsf{S}}$. As shown below, under SoDA$^{\mathsf{M}}$, ϕ is only satisfied by the multiset $\{\!\{\mathsf{Bob},\mathsf{Bob}\}\!\}$. Mapping multisets to workflow traces is straightforward. For example, workflow traces corresponding to $\{\!\{\mathsf{Bob},\mathsf{Bob}\}\!\}$ include exactly two execution events involving Bob. In this respect, SoDA$^{\mathsf{M}}$ allows a more precise mapping to workflow traces than SoDA$^{\mathsf{S}}$.

Definition 5.2 (SoDA$^{\mathsf{M}}$) *Let $U \subseteq \mathcal{U}$ be a non-empty set of users and $r \in \mathcal{R}$ a role. For a multiset of users $\mathbf{U}$, a term ϕ, and a user-role assignment UR,* multiset satisfiability *is the smallest ternary relation between multisets of users, user-role assignments, and terms, written $\mathbf{U} \models^{\mathsf{M}}_{UR} \phi$, that is closed under the rules:*

(1) $\dfrac{}{\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \mathsf{All}} \quad u \in \mathsf{dom}(UR)$

(2) $\dfrac{}{\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} r} \quad (u,r) \in UR$

(3) $\dfrac{}{\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} U} \quad u \in U \text{ and } u \in \mathsf{dom}(UR)$

(4) $\dfrac{\{\!\{u\}\!\} \not\models^{\mathsf{M}}_{UR} \phi}{\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \neg\phi}$

(5) $\dfrac{\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi}{\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi^{+}}$

(6) $\dfrac{\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi,\ \mathbf{U} \models^{\mathsf{M}}_{UR} \phi^{+}}{\{\!\{u\}\!\} \uplus \mathbf{U} \models^{\mathsf{M}}_{UR} \phi^{+}}$

(7) $\dfrac{\mathbf{U} \models^{\mathsf{M}}_{UR} \phi}{\mathbf{U} \models^{\mathsf{M}}_{UR} (\phi \sqcup \psi)}$

(8) $\dfrac{\mathbf{U} \models^{\mathsf{M}}_{UR} \psi}{\mathbf{U} \models^{\mathsf{M}}_{UR} (\phi \sqcup \psi)}$

(9) $\dfrac{\mathbf{U} \models^{\mathsf{M}}_{UR} \phi,\ \mathbf{U} \models^{\mathsf{M}}_{UR} \psi}{\mathbf{U} \models^{\mathsf{M}}_{UR} (\phi \sqcap \psi)}$

(10) $\dfrac{\mathbf{U} \models^{\mathsf{M}}_{UR} \phi,\ \mathbf{V} \models^{\mathsf{M}}_{UR} \psi}{\mathbf{U} \uplus \mathbf{V} \models^{\mathsf{M}}_{UR} (\phi \odot \psi)}$

(11) $\dfrac{\mathbf{U} \models^{\mathsf{M}}_{UR} \phi,\ \mathbf{V} \models^{\mathsf{M}}_{UR} \psi}{\mathbf{U} \uplus \mathbf{V} \models^{\mathsf{M}}_{UR} (\phi \otimes \psi)} \quad (\mathbf{U} \cap \mathbf{V}) = \varnothing\,.$

We say that $\mathbf{U}$ *satisfies* ϕ *with respect to* UR if $\mathbf{U} \models^{\mathsf{M}}_{UR} \phi$. Informally, a user u satisfies the term All if there exists a role r such that $(u,r) \in UR$. A user u satisfies a role r if u acts in the role r with respect to UR, and u satisfies a set of users U if u is a member of U and there exists a role r such that $(u,r) \in UR$. A unit term $\neg\phi$ is satisfied by u if u does not satisfy ϕ. A non-empty multiset of users $\mathbf{U}$ satisfies a complex term ϕ^+ if each user $u \in \mathbf{U}$ satisfies the unit term ϕ. A multiset of users $\mathbf{U}$ satisfies a term $\phi \sqcup \psi$ if $\mathbf{U}$ satisfies either ϕ or ψ, and $\mathbf{U}$ satisfies a term $\phi \sqcap \psi$ if $\mathbf{U}$ satisfies both ϕ and ψ. A term $\phi \otimes \psi$ is satisfied by a multiset of users $\mathbf{W}$, if $\mathbf{W}$ can be partitioned into two disjoint multisets $\mathbf{U}$ and $\mathbf{V}$, and $\mathbf{U}$ satisfies ϕ and $\mathbf{V}$ satisfies ψ. Because every user in $\mathbf{W}$ must be in either $\mathbf{U}$ or $\mathbf{V}$, but not both, the $\otimes$-operator separates duties between two multisets of users. In contrast, a term $\phi \odot \psi$ is satisfied by a multiset of users $\mathbf{W}$, if there are two multisets $\mathbf{U}$ and $\mathbf{V}$, which may share users, and $\mathbf{U}$ satisfies ϕ, $\mathbf{V}$ satisfies ψ, and $\mathbf{W}$ is the sum of $\mathbf{U}$ and $\mathbf{V}$. Thus, the $\odot$-operator allows "overlapping" duties in that a user may be contained in both $\mathbf{U}$ and $\mathbf{V}$.

With SoDA$^{\mathsf{M}}$ the significance of maximal unit terms becomes evident. If a multiset of users $\mathbf{U}$ satisfies a term ϕ, every user in $\mathbf{U}$ corresponds to at least one maximal unit term in ϕ. We associate below a user $u \in \mathbf{U}$ with the execution of a task instance by u, *i.e.* an execution event $t.u$, for an arbitrary task t. When mapping terms to processes, the satisfaction of a maximal unit term will therefore correspond to engaging in an execution event.

We now provide two examples of SoDA terms. The first serves as the SoD policy for the drug dispensation workflow and the second illustrates the difference between SoDA$^{\mathsf{M}}$ and SoDA$^{\mathsf{S}}$.

Example 5.1 (SoD Policy for Drug Dispensation Workflow) Fraudulent or erroneous drug dispensations may jeopardize a patients' health, may violate regulations, and could severely impact the hospital's finances and reputation. We therefore assume that a hospital who executes the drug dispensation workflow enforces SoD constraints in order to reduce these risks. Concretely, a Pharmacist may not dispense drugs to himself; *i.e.* he should not act as a Patient and a Pharmacist within the same workflow instance. Similarly, the Nurse who prepares the drugs should not be the same user as the Pharmacist who approves the dispensation. Furthermore, the PrivacyAdvocate must be different from any other user involved in the same workflow instance. Finally, the nurse Claire may not be involved in the dispensation due to her drug abuse history. However, as a Patient she may receive drugs. All these constraints are formalized by the term $\phi = \mathsf{Patient} \otimes ((\neg\{\mathsf{Claire}\})^+ \sqcap (\mathsf{PrivacyAdvocate} \otimes \mathsf{Pharmacist} \otimes (\mathsf{Nurse} \sqcup \mathsf{Researcher} \sqcup \mathsf{Therapist})^+))$.

Consider now the user-role assignment UR_3 shown in Figure 4.2. The multiset $\mathbf{U}_1 = \{\!\{\mathsf{Alice}, \mathsf{Bob}, \mathsf{Dave}, \mathsf{Emma}, \mathsf{Fritz}, \mathsf{Gerda}, \mathsf{Gerda}\}\!\}$ satisfies ϕ with respect to UR_3. However, $\mathbf{U}_2 = \{\!\{\mathsf{Bob}, \mathsf{Emma}, \mathsf{Fritz}, \mathsf{Gerda}, \mathsf{Gerda}\}\!\}$ does not satisfy ϕ with respect to UR_3 because ϕ requires at least one user acting as Pharmacist and $\mathbf{U}_2$ contains no user who acts as Pharmacist with respect to UR_3. ★

Example 5.2 (Difference Between SoDA$^{\mathsf{S}}$ and SoDA$^{\mathsf{M}}$) Under SoDA$^{\mathsf{M}}$, the term $\{\mathsf{Bob}\} \odot \{\mathsf{Bob}\} \odot \{\mathsf{Bob}\}^+$ is satisfied by all multisets that contain Bob three or more times, *i.e.* Bob must execute at least three tasks. Under SoDA$^{\mathsf{S}}$, this term is only satisfied by the set $\{\mathsf{Bob}\}$ and therefore does not define how many tasks Bob must actually execute. ★

We conclude by formally relating SoDA$^{\mathsf{M}}$ and SoDA$^{\mathsf{S}}$. As summarized in Appendix B, under SoDA$^{\mathsf{S}}$, $Y \models^{\mathsf{S}}_{(U,UR)} \phi$ denotes the satisfaction of a term ϕ by a set of users $Y \subseteq \mathcal{U}$ with respect to a tuple (U, UR), where $U \subseteq \mathcal{U}$ and $UR \subseteq U \times \mathcal{R}$. Because tasks can only be executed by users who are assigned to at least one role, we simplify this tuple and extract the available users from UR, as can be seen in Rule (3) of Definition. 5.2. For a user-role assignment UR, the auxiliary function $\mathsf{lwconf}(UR) = (\mathsf{dom}(UR), UR)$ maps UR to the corresponding tuple in SoDA$^{\mathsf{S}}$. Moreover, given a multiset of users $\mathbf{U}$, the function $\mathsf{userset}(\mathbf{U}) = \{u \mid u \in \mathbf{U}\}$ returns the set of users contained in $\mathbf{U}$. The following lemma, which we prove in Appendix A.2, relates SoDA$^{\mathsf{M}}$ to SoDA$^{\mathsf{S}}$.

Lemma 5.1 *For all terms ϕ, all user-role assignments UR, and all multisets of users $\mathbf{U}$, if $\mathbf{U} \models^{\mathsf{M}}_{UR} \phi$, then $\mathsf{userset}(\mathbf{U}) \models^{\mathsf{S}}_{\mathsf{lwconf}(UR)} \phi$.*

5.3 Enforcement Requirements

As shown above, SoDA specifies SoD constraints at a high level of abstraction. However, the enforcement takes place at run time in the context of a workflow instance. Given a term ϕ, we now describe how to construct an enforcement monitor for ϕ. Our construction maps ϕ to a process $SODA_\phi(UR)$, called the *SoDA-enforcement process*, parametrized by a user-role assignment UR. $SODA_\phi(UR)$ accepts all workflow traces corresponding to a multiset that satisfies ϕ with respect to UR. We show later in Chapter 6 how to implement $SODA_\phi(UR)$ as a service and how to provision and integrate this service in an enterprise environment.

In practice, it is critical to allow administrative events during workflow execution. If Bob leaves his company, it should be possible to remove all his assignments to roles, thereby preventing him from subsequently executing task instances. Similarly, if Alice joins a company or changes positions, and is therefore assigned to new roles, she should also be able to execute task instances in

workflow instances that were started prior to the organizational change. Assuming that a user-role assignment does not change during the execution of a workflow instance is therefore overly restrictive. The SoDA-enforcement process defined below accounts for such changes. The function upd ("update") describes how a trace of administrative events changes a user-role assignment.

Definition 5.3 (Administrative Update) *Let $a \in \mathcal{A}^*$ be a trace of administrative events and UR a user-role assignment. The function* upd *is then defined as*

$$\mathsf{upd}(UR, a) = \begin{cases} UR & \text{if } a = \langle\rangle, \\ \mathsf{upd}(UR \cup \{(u,r)\}, a') & \text{if } a = \langle \mathsf{add}.u.r \rangle^\frown a', \\ \mathsf{upd}(UR \setminus \{(u,r)\}, a') & \text{if } a = \langle \mathsf{rm}.u.r \rangle^\frown a', \end{cases}$$

where u ranges over $\mathcal{U}$, r over $\mathcal{R}$, and a' over $\mathcal{A}^*$.

Let ϕ be a term, UR a user-role assignment, and $SODA_\phi(UR)$ the SoDA-enforcement process for ϕ and UR. We postulate the following requirements for $SODA_\phi(UR)$:

(R1) $SODA_\phi(UR)$ must accept every trace of admin events a, and behave like $SODA_\phi(UR')$ afterward, for $UR' = \mathsf{upd}(UR, a)$.

(R2) $SODA_\phi(UR)$ must engage in an execution event $t.u$, if $\{\!\{u\}\!\}$ satisfies at least one maximal unit term of ϕ with respect to UR.

(R3) The semantics of the operators $^+$, $\sqcup$, $\sqcap$, $\odot$, and $\otimes$ with respect to traces must agree with their definition in SoDA$^{\mathsf{M}}$.

Requirement (R1) says that administrative events are always possible and their effects are reflected in the user-role assignment. (R2) formulates agreement with SoDA$^{\mathsf{M}}$, where for a multiset of users $\mathbf{U}$, if $\mathbf{U} \models^{\mathsf{M}}_{UR} \phi$, then each user in $\mathbf{U}$ satisfies at least one maximal unit term of ϕ with respect to UR. Similarly, $SODA_\phi(UR)$ must not engage in an execution event if the corresponding user does not contribute to the satisfaction of ϕ. As for (R3), consider for example the terms $\phi \otimes \psi$ and $\phi \odot \psi$. It must be possible to partition a trace satisfying $\phi \otimes \psi$ or $\phi \odot \psi$ into two subtraces, one satisfying ϕ and the other one satisfying ψ. In the case of $\phi \otimes \psi$, the users who execute task instances in one trace must be disjoint from the users executing task instances in the other trace. In contrast, for $\phi \odot \psi$, the multisets of users need not be disjoint. In particular, (R3) states that if $SODA_\phi(UR)$ accepts a trace i that contains no admin event and reaches a final state, then $\mathsf{users}(i) \models^{\mathsf{M}}_{UR} \phi$.

5.4 Trace Semantics

The following example shows that SoDA$^{\mathsf{M}}$ is not expressive enough to capture the requirements (R1)–(R3).

Example 5.3 (Limitations of SoDA$^{\mathsf{M}}$) Suppose that SoDA$^{\mathsf{M}}$ were expressive enough to capture (R1)–(R3). Consider the policy P that requires one task to be executed by a user acting as a Pharmacist and another task to be executed by a user who is not acting as a Pharmacist. We model P by the term $\phi = \mathsf{Pharmacist} \odot \neg\mathsf{Pharmacist}$ and consider the trace

$$i = \langle \mathsf{add.Alice.Pharmacist},\ t_1.\mathsf{Alice},\ \mathsf{rm.Alice.Pharmacist},\ t_2.\mathsf{Alice} \rangle\ ,$$

for two arbitrary tasks t_1 and t_2. From (R1)–(R3), it follows that $SODA_\phi(\varnothing)$ must accept i. In particular, by (R1), $SODA_\phi(\varnothing)$ engages in add.Alice.Pharmacist and afterward behaves like $SODA_\phi(UR)$, for $UR = \{(\mathsf{Alice}, \mathsf{Pharmacist})\}$. Next, $SODA_\phi(UR)$ engages in t_1.Alice by (R2) and (R3) because Alice acts as a Pharmacist. Again by (R1), $SODA_\phi(UR)$ engages in rm.Alice.Pharmacist and afterward behaves like $SODA_\phi(\varnothing)$. Finally, by (R2) and (R3), $SODA_\phi(\varnothing)$ engages in t_2.Alice because Alice does not act as a Pharmacist. In the end, $SODA_\phi$ engaged in an execution event with a user that acted as a Pharmacist and in another execution event with a user not acting as a Pharmacist, satisfying the policy P. Because of the administrative events it was possible that both tasks were executed by the same user, *i.e.* $\mathsf{users}(i) = \{\!\{\mathsf{Alice}, \mathsf{Alice}\}\!\}$. However, under SoDA$^{\mathsf{M}}$, ϕ can only be satisfied by a multiset of users that contains two different users, which contradicts $\mathsf{users}(i) = \{\!\{\mathsf{Alice}, \mathsf{Alice}\}\!\}$. Hence, SoDA$^{\mathsf{M}}$ is not expressive enough to capture (R1)–(R3). ★

The inability to handle administrative changes motivates the introduction of a third semantics, SoDA$^{\mathsf{T}}$. In SoDA$^{\mathsf{T}}$, subterms correspond to separate traces that may interleave with each other in any order. Administrative events, though, must occur in all traces in the same order. This reflects that SoDA terms do not constrain the order of executed tasks but that the user-role assignment must be consistent across all subterms at any time. We formalize this relation by the *synchronized interleaving* predicate si. For traces i, i_1, and i_2, $\mathsf{si}(i, i_1, i_2)$ holds if and only if i_1 and i_2 "partition" i such that each administrative event in i is contained in both i_1 and i_2, and each execution event is either in i_1 or i_2. More precisely:

Definition 5.4 (Synchronized Interleaving) *Let $i, i_1, i_2 \in (\mathcal{X} \cup \mathcal{A})^*$ be traces. The* synchronized interleaving *predicate* $\mathsf{si}(i, i_1, i_2)$ *is defined as:*

$$\mathsf{si}(i,i_1,i_2) = \begin{cases} \mathit{true} & \mathit{if}\ i = \langle\rangle, i_1 = \langle\rangle\ \mathit{and}\ i_2 = \langle\rangle, \\ \mathsf{si}(i', i_1', i_2') & \mathit{if}\ i = \langle a\rangle\hat{}\, i', i_1 = \langle a\rangle\hat{}\, i_1', \mathit{and}\ i_2 = \langle a\rangle\hat{}\, i_2', \\ \mathsf{si}(i', i_1', i_2)\ \mathit{or}\ \mathsf{si}(i', i_1, i_2') & \mathit{if}\ i = \langle x\rangle\hat{}\, i', i_1 = \langle x\rangle\hat{}\, i_1', \mathit{and}\ i_2 = \langle x\rangle\hat{}\, i_2', \\ \mathsf{si}(i', i_1', i_2) & \mathit{if}\ i = \langle x\rangle\hat{}\, i', i_1 = \langle x\rangle\hat{}\, i_1', \mathit{and}\ i_2 \neq \langle x\rangle\hat{}\, i_2', \\ \mathsf{si}(i', i_1, i_2') & \mathit{if}\ i = \langle x\rangle\hat{}\, i', i_1 \neq \langle x\rangle\hat{}\, i_1', \mathit{and}\ i_2 = \langle x\rangle\hat{}\, i_2', \\ \mathit{false} & \mathit{otherwise}, \end{cases}$$

where a ranges over $\mathcal{A}$, x over $\mathcal{X}$, and i', i_1', and i_2' over $(\mathcal{X} \cup \mathcal{A})^*$.

Note that the Boolean *or* in the third case arises as there are two possible interleavings. The predicate si will hold (evaluate to *true*) if either of the two interleavings hold. We illustrate si with an example.

$$\begin{array}{rcl} i & = & \langle\, x_1, x_2, x_3, a_1, x_4, x_4, a_2, x_5, a_3, x_6, a_4 \,\rangle \\ i_1 & = & \langle\, x_1, \quad x_3, a_1, x_4, \quad a_2, \quad a_3, x_6, a_4 \,\rangle \\ i_2 & = & \langle\, \quad x_2, \quad a_1, \quad x_4, a_2, x_5, a_3, \quad a_4 \,\rangle \end{array}$$

For these three traces, $\mathsf{si}(i, i_1, i_2)$ holds. We now define the satisfaction of SoDA terms by traces.

Definition 5.5 (SoDA$^{\mathsf{T}}$) *Let $u \in \mathcal{U}$ be a user, $t \in \mathcal{T}$ a task, $x \in \mathcal{X}$ an execution event, and $a \in \mathcal{A}$ an administrative event. For a trace $i \in (\mathcal{X} \cup \mathcal{A})^*$, a user-role assignment UR, a term ϕ, and a unit term ϕ_{ut},* trace satisfiability *is the smallest ternary relation between traces, user-role assignments, and terms, written $i \models^{\mathsf{T}}_{UR} \phi$, closed under the rules:*

(1) $\dfrac{\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut}}{\langle t.u\rangle \models^{\mathsf{T}}_{UR} \phi_{ut}}$ (2) $\dfrac{i \models^{\mathsf{T}}_{UR} \phi}{i\hat{}\langle a\rangle \models^{\mathsf{T}}_{UR} \phi}$ (3) $\dfrac{i \models^{\mathsf{T}}_{UR \cup \{(u,r)\}} \phi}{\langle \mathsf{add}.u.r\rangle\hat{}\, i \models^{\mathsf{T}}_{UR} \phi}$

(4) $\dfrac{i \models^{\mathsf{T}}_{UR \setminus \{(u,r)\}} \phi}{\langle \mathsf{rm}.u.r\rangle\hat{}\, i \models^{\mathsf{T}}_{UR} \phi}$ (5) $\dfrac{\langle x\rangle \models^{\mathsf{T}}_{UR} \phi_{ut}}{\langle x\rangle \models^{\mathsf{T}}_{UR} \phi^+_{ut}}$ (6) $\dfrac{\langle x\rangle \models^{\mathsf{T}}_{UR} \phi_{ut}\ ,\ i \models^{\mathsf{T}}_{UR} \phi^+_{ut}}{\langle x\rangle\hat{}\, i \models^{\mathsf{T}}_{UR} \phi^+_{ut}}$

(7) $\dfrac{i \models^{\mathsf{T}}_{UR} \phi}{i \models^{\mathsf{T}}_{UR} \phi \sqcup \psi}$ (8) $\dfrac{i \models^{\mathsf{T}}_{UR} \psi}{i \models^{\mathsf{T}}_{UR} \phi \sqcup \psi}$ (9) $\dfrac{i \models^{\mathsf{T}}_{UR} \phi\ ,\ i \models^{\mathsf{T}}_{UR} \psi}{i \models^{\mathsf{T}}_{UR} \phi \sqcap \psi}$

(10) $\dfrac{i_1 \models^{\mathsf{T}}_{UR} \phi\ ,\ i_2 \models^{\mathsf{T}}_{UR} \psi}{i \models^{\mathsf{T}}_{UR} \phi \odot \psi}$ $\mathsf{si}(i, i_1, i_2)$

(11) $\dfrac{i_1 \models^{\mathsf{T}}_{UR} \phi\ ,\ i_2 \models^{\mathsf{T}}_{UR} \psi}{i \models^{\mathsf{T}}_{UR} \phi \otimes \psi}$ $\mathsf{si}(i, i_1, i_2)$ *and* $\mathsf{users}(i_1) \cap \mathsf{users}(i_2) = \varnothing$.

We say that *i satisfies ϕ with respect to UR*, if $i \models^{\mathsf{T}}_{UR} \phi$. SoDA$^{\mathsf{T}}$ fulfills the requirements of Section 5.3: (R1) follows from rules (2) to (4), (R2) from rule (1), and (R3) from the rules corresponding to the respective operators. That SoDA$^{\mathsf{M}}$ agrees with SoDA$^{\mathsf{T}}$ in the absence of administrative events is shown by the following lemma, which we prove in Appendix A.3.

Lemma 5.2 *For all terms ϕ, all user-role assignments UR, and all traces $i \in \mathcal{X}^*$, if $i \models^{\mathsf{T}}_{UR} \phi$, then* $\mathsf{users}(i) \models^{\mathsf{M}}_{UR} \phi$.

Consider again the trace i and the term ϕ from Example 5.3. It is straightforward to see that i satisfies ϕ with respect to $UR = \varnothing$. Hence, SoDA$^{\mathsf{T}}$ overcomes the limitations of SoDA$^{\mathsf{M}}$ illustrated in Example 5.3.

Summarizing, we first generalized SoDA$^{\mathsf{S}}$ to SoDA$^{\mathsf{M}}$ and thereby solved the problem that SoDA$^{\mathsf{S}}$ does not specify how many tasks each user who contributes to the satisfaction of a term must execute. Second, we introduced administrative events that may change user-role assignments, defined requirements that an SoDA enforcement incorporating these events must satisfy, and showed that SoDA$^{\mathsf{M}}$ does not capture them. Third, we further generalized SoDA$^{\mathsf{M}}$ to SoDA$^{\mathsf{T}}$ and showed that SoDA$^{\mathsf{T}}$ satisfies our requirements. Next, we define a mapping from terms to processes, which model enforcement monitors, and prove the mapping's correctness with respect to SoDA$^{\mathsf{T}}$.

5.5 Mapping Terms to Processes

We first introduce the auxiliary process *END* that engages in an arbitrary number of admin events before it successfully terminates.

$$END = (a : \mathcal{A} \to END) \,\Box\, SKIP$$

Using *END*, we define the mapping $[.]^{U}_{UR}$.

Definition 5.6 (Mapping $[.]^{U}_{UR}$) *Given a set of users U, a user-role assignment UR, and a term ϕ, the* mapping $[\phi]^{U}_{UR}$ *returns a process parametrized by UR. For a unit term ϕ_{ut} and terms ϕ and ψ, the mapping $[.]^{U}_{UR}$ is defined as follows.*

$$\begin{aligned}(1)\quad [\phi_{ut}]^{U}_{UR} \;=\;& t : \mathcal{T}.u : \{u' \in U \mid \{\!\{u'\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut}\,\} \to END \\ & \Box\, \mathsf{add}.u : \mathcal{U}.r : \mathcal{R} \to [\phi_{ut}]^{U}_{UR \cup \{(u,r)\}} \\ & \Box\, \mathsf{rm}.u : \mathcal{U}.r : \mathcal{R} \to [\phi_{ut}]^{U}_{UR \setminus \{(u,r)\}}\end{aligned}$$

$$
\begin{aligned}
(2)\quad [\phi^{+}_{ut}]^{U}_{UR} &= t:\mathcal{T}.u:\{u'\in U \mid \{\!\{u'\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut}\} \rightarrow (END \,\square\, [\phi^{+}_{ut}]^{U}_{UR}) \\
&\quad \square\, \mathsf{add}.u:\mathcal{U}.r:\mathcal{R} \rightarrow [\phi^{+}_{ut}]^{U}_{UR \cup \{(u,r)\}} \\
&\quad \square\, \mathsf{rm}.u:\mathcal{U}.r:\mathcal{R} \rightarrow [\phi^{+}_{ut}]^{U}_{UR \setminus \{(u,r)\}} \\
(3)\quad [\phi \sqcup \psi]^{U}_{UR} &= [\phi]^{U}_{UR} \,\square\, [\psi]^{U}_{UR} \\
(4)\quad [\phi \sqcap \psi]^{U}_{UR} &= [\phi]^{U}_{UR} \parallel [\psi]^{U}_{UR} \\
(5)\quad [\phi \odot \psi]^{U}_{UR} &= [\phi]^{U}_{UR} \underset{\mathcal{A}}{\parallel} [\psi]^{U}_{UR} \\
(6)\quad [\phi \otimes \psi]^{U}_{UR} &= \underset{\{(U_\phi,U_\psi) \mid U_\phi \cup U_\psi = U \wedge U_\phi \cap U_\psi = \varnothing\}}{\square} [\phi]^{U_\phi}_{UR} \underset{\mathcal{A}}{\parallel} [\psi]^{U_\psi}_{UR}
\end{aligned}
$$

Note that the equations (1) and (2) require determining whether $\{\!\{u'\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut}$. This problem is analogous to testing whether a propositional formula is satisfiable under a given assignment and is also decidable in polynomial time.

Definition 5.7 (SoDA-Enforcement Process) *For a term ϕ and a user-role assignment UR, the* SoDA-enforcement process *$SODA_\phi(UR)$ is the process $[\phi]^{\mathcal{U}}_{UR}$.*

Before we show how a SoDA-enforcement process is used together with workflow execution processes and an RBAC process, we define correctness for the mapping $[.]^{U}_{UR}$.

Definition 5.8 (Corectness of $[.]^{U}_{UR}$) *The mapping $[.]^{U}_{UR}$ is* correct *if for all terms ϕ, all user-role assignments UR, and all traces $i \in \Sigma^*$, $i\,\hat{}\,\langle\checkmark\rangle \in \mathsf{T}(SODA_\phi(UR))$ if and only if $i \models^{\mathsf{T}}_{UR} \phi$.*

Informally, the mapping $[.]^{U}_{UR}$ is correct if the following properties hold for all SoDA-enforcement processes $SODA_\phi$: (1) if $SODA_\phi$ accepts a workflow trace that corresponds to a successfully terminated workflow instance, then its prefix excluding $\checkmark$ satisfies ϕ under SoDA$^{\mathsf{T}}$, and (2) if a workflow trace satisfies ϕ under SoDA$^{\mathsf{T}}$, then its extension by $\checkmark$ corresponds to a successfully terminated workflow instance and is accepted by $SODA_\phi$. We prove the following theorem in Appendix A.4.

Theorem 5.1 *The mapping $[.]^{U}_{UR}$ is correct.*

Hence, if the SoDA-enforcement process accepts a successfully terminated workflow instance, then the corresponding SoD constraint is satisfied. We also know that no workflow instance that satisfies the SoD constraint is falsely blocked by the SoDA-enforcement process. The following corollary relates the set of traces of SoDA-enforcement processes without administrative events and

their corresponding multisets of users under the multiset semantics. Its proof follows directly from Theorem 5.1 and Lemma 5.2.

Corollary 5.1 *For all terms ϕ, all user-role assignments UR, and all traces $i \in \mathcal{X}^*$, if $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SODA_\phi(UR))$, then* $\mathsf{users}(i) \models^{\mathsf{M}}_{UR} \phi$.

Given a workflow execution process W, an RBAC policy (UR, RT), and a term ϕ, the *SoDA-secure (workflow) process* SSW_ϕ is the parallel, partially synchronized composition of W, the RBAC process for (UR, RT), and the SoDA-enforcement process $SODA_\phi$, parametrized by an RBAC policy.

$$SSW_\phi(UR, RT) = W \underset{\mathcal{X}}{\parallel} (RBAC(UR, RT) \parallel SODA_\phi(UR))$$

Let $x = t.u$ be an execution event, for a task t and a user u. $SSW_\phi(UR, RT)$ engages in x if W, $RBAC(UR, RT)$, and $SODA_\phi(UR)$ each engage in x. In other words, t must be one of the next tasks according to the workflow specification, the user u must be authorized to execute the task t with respect to (UR, RT), and u must not violate the SoD policy formalized by ϕ, given the previously executed execution events and UR. Furthermore, $RBAC$ and $SODA_\phi$ can synchronously engage in an administrative event at any time and thereby change their RBAC policy. Finally, $SSW_\phi(UR, RT)$ engages in $\checkmark$ if W, $RBAC$, and $SODA_\phi(UR)$ synchronously engage in $\checkmark$.

Example 5.4 (SoDA-Secure Workflow Process) We return to our running example. Consider again the drug dispensation workflow, modeled by the workflow execution process W, the user-role assignments UR_1, UR_2, and UR_3, and the role-task assignment RT, introduced in Section 4.1 and 4.3, respectively. Furthermore, recall the workflow's SoD policy introduced in Example 5.1 and formalized by the term $\phi = \mathsf{Patient} \otimes ((\neg\{\mathsf{Claire}\})^+ \sqcap (\mathsf{PrivacyAdvocate} \otimes \mathsf{Pharmacist} \otimes (\mathsf{Nurse} \sqcup \mathsf{Researcher} \sqcup \mathsf{Therapist})^+))$. We concluded in Example 4.2 that $i_3 = \langle t_1.\mathsf{Fritz},\ t_2.\mathsf{Emma},\ \mathsf{add.Fritz.PrivacyAdvocate},\ t_3.\mathsf{Fritz},\ t_5.\mathsf{Bob} \rangle$ is a trace of $SW(UR_1, RT)$. However, i_3 is not a trace of the SoDA-secure process $SSW_\phi(UR_1, RT)$ because i_3 is not a trace of $SODA_\phi(UR_1)$. In particular, when Fritz executes t_1 in i_3 he acts only in the role Patient and when he later executes t_3 he acts as a Patient and a PrivacyAdvocate. By Definition 5.6 and 5.7, $SODA_\phi(UR_1)$ engages in one execution event where the respective user acts as a Patient and one where the user acts as a PrivacyAdvocate. However, due to the $\otimes$-operator between the corresponding subterms in ϕ, these users must be different, *i.e.* both cannot be Fritz. Of course $SODA_\phi(UR_1)$ engages in further execution events, but Fritz does not satisfy the respective subterms. Hence, $SODA_\phi(UR_1)$ accepts i_3's prefix $\langle t_1.\mathsf{Fritz}, t_2.\mathsf{Emma}, \mathsf{add.Fritz.PrivacyAdvocate} \rangle$ but does not engage in t_3.Fritz afterward.

In contrast, $SODA_\phi(UR_1)$ accepts the trace $i_4 = \langle t_1.\mathsf{Dave}, t_2.\mathsf{Emma}, \mathsf{add.Fritz.PrivacyAdvocate}, t_3.\mathsf{Fritz}, t_5.\mathsf{Bob}, \mathsf{add.Alice.Pharmacist}, t_7.\mathsf{Alice}, t_9.\mathsf{Gerda}, t_{10}.\mathsf{Gerda}, \checkmark \rangle$. By Theorem 5.1 and because $\checkmark \in i_4$ we have that $(i_4 \restriction \Sigma) \models^{\mathsf{T}}_{UR_1} \phi$. In other words, i_4 models a workflow instance that satisfies the drug dispensation workflow's SoD policy. Furthermore, by engaging in the first administrative event in i_4, the user-role assignment changes to UR_2 and after engaging in the second administrative event it becomes UR_3. Thereby, every task execution is authorized and $RBAC(UR_1, RT)$ accepts i_4 as well. Finally, it is easy to see that $(i_4 \restriction \mathcal{X}^\checkmark) \in \mathsf{T}(W)$, *i.e.* i_4 is also a workflow trace of W. As a result, $i_4 \in \mathsf{T}(SSW_\phi(UR_1, RT))$. ★

This example illustrates how the three kinds of processes presented so far interact and how each of them enforces its corresponding specification: W formalizes the workflow model, $RBAC$ a possibly changing workflow-independent RBAC policy, and $SODA_\phi(UR)$ an SoD policy formalized in SoDA, while accounting for changing user-role assignments.

We are now ready to map our processes to a software implementation. We return to the drug dispensation workflow in Section 6.5, when reporting on the performance of our implementation.

Chapter 6
Implementation

In this chapter, we describe an implementation of SoD as a Service. Our goal is to demonstrate the flexibility of this approach, to analyze its scalability, and to identify performance-critical parameters. We use the SoDA-secure process as blueprint for our implementation. Its sub-processes naturally map to components of a SOA as illustrated in Figure 6.1. The components' interfaces can be inferred from the sets of events on which the respective processes synchronize and the processes themselves describe the components' behavior. We proceed by implementing W by a workflow engine, $RBAC$ by a user repository, and $SODA_\phi$ by a program called a *SoDA-enforcement monitor*. Workflow engines and user repositories are well-established concepts and we therefore realize them using off-the-shelf components. The standalone SoDA-enforcement monitor, however, is something fundamentally new. Hence, we implemented it from scratch, indicated by dark gray in Figure 6.1.

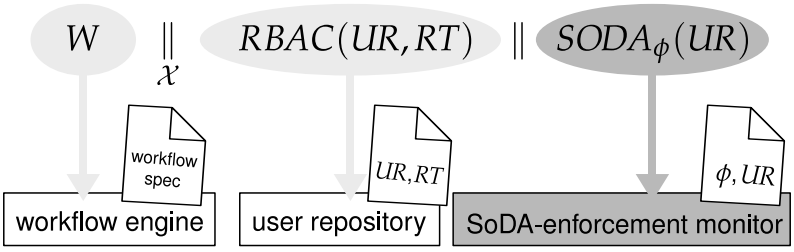


Figure 6.1. From theory to practice: mapping processes to software components.

6.1 Technical Objectives

We aim to realize an effective, practical, and efficient implementation of SoD as a Service. By *effective* we mean that the implementation fulfills its purpose. Namely, it should support the execution of arbitrary workflows, facilitate chang-

ing RBAC policies, and correctly enforce SoD constraints that are specified as SoDA terms.

We understand *practicability* in the sense that the integration and configuration effort is moderate. The main components of our system should be loosely coupled in order to enable a separation of concerns and to allow the integration of preexisting components, such as a legacy workflow system. Furthermore, the system should be configurable using standard means, *e.g.* a workflow definition, an RBAC policy, and an SoD policy, rather than requiring additional, labor-intensive settings.

The performance of our implementation is critical to the success of our approach. We call the running time of a system with a workflow engine and a user repository, but without a SoDA-enforcement monitor, the *running time baseline*. Our objective is to enforce SoD constraints *efficiently*, that is with a low overhead compared to the running time baseline.

6.2 Architecture

As defined in Section 5.5, a SoDA-secure process is the parallel, partially synchronized execution of three sub-processes, each responsible for a specific task. Due to the associativity of CSP's synchronous parallel-composition operator $\|$, these three processes can be grouped in any order. Furthermore, the set of events on which these processes synchronize defines the kinds of events each process engages in. Therefore, any subset of these three processes can be mapped to an enforcement monitor and the set of events synchronized with the remaining processes specifies the monitor's interface. This is of particular interest if a system already provides one of the components we model by our processes. For example, suppose a system comes with a workflow engine and an enforcement monitor for workflow-independent authorizations. In this case, it is sufficient to generate an enforcement monitor for the SoDA-enforcement process and to synchronize all execution and administrative events with the existing components.

Figure 6.1 shows our general approach of mapping W, $RBAC$, and $SODA_\phi$ to individual system components. The concrete software tools we use and their intercommunication are illustrated in Figure 6.2; newly developed components are again indicated in gray. Ignore the arrows and labels for the moment.

- **Workflow Engine:** We use the IBM WebSphere Process Server (WPS) [2011] as a workflow engine. WPS runs on top of the IBM WebSphere Application Server (WAS) [2012b], which is IBM's Java EE application server [Haugland *et al.* 2004].

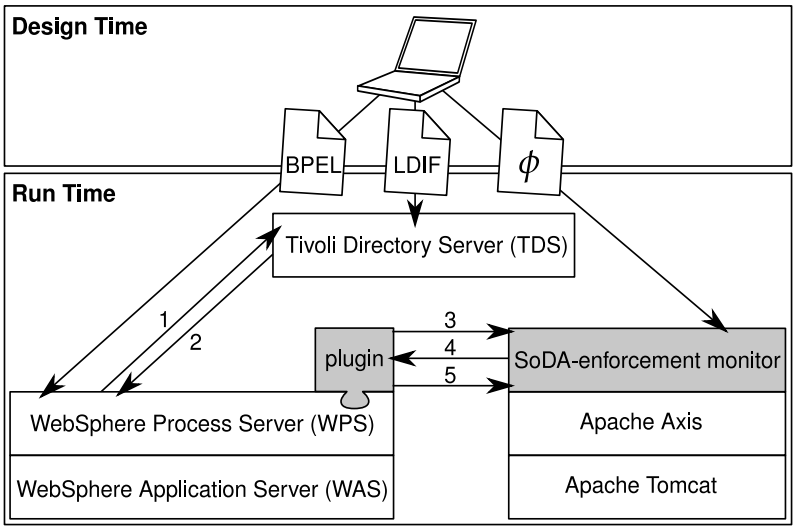


Figure 6.2. SoD as a Service architecture

- **User Repository:** The IBM Tivoli Directory Server (TDS) [2012a] serves as a user repository. TDS is an LDAP server whose LDAP schema we configured to support RBAC policies.

- **SoDA-Enforcement Monitor:** We implemented the SoDA-enforcement monitor in Java and wrapped it as a web service, using Apache Axis [2012a] running on top of Apache Tomcat [2012b].

Along with the various web service standards, many semi-formal business process modeling languages have emerged. Backed by numerous software vendors, the *Web Service Business Process Execution Language (WS-BPEL)* [Alves *et al.* 2007], or *BPEL* for short, is a popular standard for describing workflows at the implementation-level. A BPEL workflow definition can be directly executed by a workflow engine. At design time, we define a workflow in BPEL, possibly generated from a BPMN model, and deploy it to WPS. We use the BPEL extension *BPEL4People* [Agrawal *et al.* 2007] to specify human tasks.

LDAP supports RBAC with the object class `accessRole`. Instances of this class represent a role and store the distinguished name of their members, typically instances of `inetOrgPerson`, in the field `member`. We encode $\mathcal{U}$, $\mathcal{R}$, and *UR* in LDAP's export format LDIF and send it to TDS, or we administer them directly through TDS' web interface.

Using an ASCII version of the SoDA grammar, we encode SoDA terms as character strings and send them to the SoDA-enforcement monitor with a standalone client.

By adopting a service-oriented architecture, we achieve a loose coupling between our three main system components. This allows us to integrate two off-the-shelf components and our newly developed SoDA-enforcement monitor. Hence, we achieve the flexibility described in Section 6.1.

The downside of a SOA approach is the increased communication and serialization overhead. To determine whether a user is authorized to execute a task instance with respect to an SoD constraint, the SoDA-enforcement monitor requires context information, which must be sent across the network. Our design decisions in this regard are explained in Chapter 7 and the performance analysis in Section 6.5 shows that the communication overhead is acceptable. Similar trade-offs between flexible, distributed architectures with an increased communication overhead versus monolithic architectures with a smaller communication overhead have been made in the past. For example, the Hierarchical Resource Profile for XACML [Anderson 2005] proposes sending the hierarchy, based on which an authorization decision is made, to the access control monitor along with each access request. As with our architecture, the access control monitor needs considerable context information to compute an access decision.

6.3 Enforcement

Our prototype system implements a SoDA-secure workflow process SSW_ϕ as follows. The SSW_ϕ process engages in three kinds of events: execution events, administrative events, and the event ✓. The implementation and handling of administrative events and the event ✓ is straightforward. We take therefore a closer look at execution events and explain why every task instance in our system corresponds to an execution event that is accepted by SSW_ϕ. An execution event corresponds to a sequence of steps in our implementation.

Consider the SoDA-secure workflow process

$$SSW_\phi(UR, RT) = W \underset{\mathcal{X}}{\parallel} (RBAC(UR, RT) \parallel SODA_\phi(UR))$$

for a SoDA term ϕ, an RBAC policy (UR, RT), and a workflow execution process W that models a workflow w. Assume that $i \in \mathsf{T}(SSW_\phi(UR, RT))$ corresponds to an unfinished workflow instance of w. Let UR' be the user-role assignment after executing the administrative events in i. Assume that t is the next task of w that is instantiated and executed in the workflow instance corresponding to i. We now look at the steps that our architecture performs, which will finally constitute an execution event $x = t.u$, for a user u. We refer to an arrow labeled with k in Figure 6.2 as (Ak).

1. **Instantiation:** The creation of x is triggered by the termination of the preceding task instance, *i.e.* the rightmost execution event in i, or by the instantiation of the workflow instance corresponding to i.

2. **RBAC Authorization:** In SSW_ϕ, authorization decisions are made by the processes $RBAC$ and $SODA_\phi$, whereas W simply defines the order in which task instances are executed. This is handled differently in most commercial workflow systems, including ours. For example, BPEL4People requires the definition of a query, called a *people link*, for every task. When the workflow engine instantiates the task, it executes the respective query against the user repository. The returned users are candidates for executing the newly created task instance.

   For a user u, the process $RBAC(UR', RT)$ accepts the execution event $t.u$ if u is assigned to one of the roles in $R_t = \{r \mid (r,t) \in RT\}$ with respect to UR'. Therefore, during design time, we specify t's people link in such a way that the user repository returns all users who are assigned to a role in R_t. In other words, the user repository keeps track of the user-role assignment UR and the workflow definition specifies the role-task assignment RT.

   WPS evaluates t's people link after every instantiation of t. Initially, the people link is sent to TDS (A1). Afterward, TDS returns the set of users $U_1 = \{u \mid \exists r \in R_t \,.\, (u,r) \in UR'\}$ to WPS (A2).

3. **Refinement to SoD-Compliant Users:** Next, we select those users from U_1 who are allowed to execute t with respect to ϕ and i. Namely, we compute the set of users $U_2 = \{u \in U_1 \mid i \,\hat{}\, \langle t.u \rangle \in \mathsf{T}(SODA_\phi(UR'))\}$.

   WPS provides a plugin interface that allows one to post-process the sets of users returned by a user repository. We wrote a plugin for this interface that sends U_1, their assignments to roles $UR'_1 = \{(u,r) \in UR' \mid u \in U_1\}$, and the identifiers of w and i to the SoDA-enforcement monitor (A3). We refer to this web service call as a *refinement call*.

   For every workflow, the SoDA-enforcement monitor stores the corresponding SoDA term. Furthermore, it keeps track of the users who execute task instances (see step *Claim*). Together with the above mentioned inputs, the SoDA-enforcement monitor therefore has all the necessary parameters to compute U_2, which it then returns to WPS (A4).

4. **Display:** A user can interact with WPS through a personalized web interface. Once a user has successfully logged into the system, WPS displays a list of task instances that the user is authorized to execute. We call this

list the user's *inbox*. For every user $u \in U_2$, $i \hat{} \langle t.u \rangle \in \mathsf{T}(SSW_\phi(UR, RT))$. Therefore, WPS displays t in the inbox of every user in U_2.

5. **Claim:** In the workflow terminology, if a user requests to execute a task, he is said to *claim* the task. One of the users in U_2 must claim t by clicking on t in his inbox. Assume the user u claims t, which corresponds to the execution event $t.u$. Instantaneously, t is removed from the inboxes of all other users. At this point, we must communicate to the SoDA-enforcement monitor that u is executing t. In addition, we send the roles assigned to u to the monitor (A5). We refer to this web service call as a *claim call*.

6. **Termination:** Afterward, u is prompted with a form whose completion constitutes the work associated with t. The work is completed when the form is submitted. If the instance of t does not terminate the workflow instance, its execution triggers the instantiation of at least one more task.

Summarizing, our system effectively enforces abstract SoD constraints as specified in Section 6.1. Arbitrary workflows, constrained by a possibly changing RBAC policy and an abstract SoD policy, can be executed on WPS. The practicability of our approach is further supported by performance measurements for our running example in Section 6.5. However, we first examine its runtime complexity.

6.4 Complexity

When analyzing the runtime complexity of our SoDA-enforcement monitor implementation, it suffices to consider the complexity of refinement calls. The complexity of claim calls are negligible compared to refinement calls and is therefore not discussed.

In general, the problem of deciding whether a term is satisfied by a set of users is **NP**-complete [Li and Wang 2008]. The SoDA-enforcement monitor must solve this decision problem for every user received through a refinement call. Therefore, it comes as no surprise that refinement calls have a worst-case exponential runtime complexity. However, we can show that the exponent remains small for moderate size workflows.

The parameters of a refinement call are a set of users, U_1, their role assignments, UR_1', and the identifiers of i and w. Using the identifier of w, the monitor retrieves ϕ. With the identifier of i, it retrieves all the users who have executed task instances in i and their role assignments at that time.

For each $u \in U_1$, the SoDA-enforcement monitor computes whether $i \hat{} \langle t.u \rangle \in \mathsf{T}(SODA_\phi(UR_1'))$. This computation is executed $|U_1| = n$ times. Consider the

[.]-mapping. The evaluation of a unit term can be performed in polynomial time in the size of $|\mathcal{U}|$ and $|\mathcal{R}|$; *i.e.* $p(|\mathcal{U}|,|\mathcal{R}|)$ for a polynomial p. In the worst case, $SODA_\phi(UR'_1)$ branches $2^{|\mathcal{U}|}$ times per operator in ϕ. If m is the number of operators, the worst-case runtime complexity is thus in $O(n\,m\,2^{|\mathcal{U}|}\,p(|\mathcal{U}|,|\mathcal{R}|))$.

The exponential factor originates from the $\otimes$-operator, which causes $SODA_\phi(UR'_1)$ to branch for all disjoint subsets of $\mathcal{U}$. Let $U_{i+u} = \mathsf{userset}(\mathsf{users}(i)) \cup \{u\}$, *i.e.* the set of users in execution events in i and u. If we check whether $i \,\hat{}\, \langle t.u \rangle \in \mathsf{T}(SODA_\phi(UR'_1))$, the users in $\mathcal{U} \setminus U_{i+u}$ are not relevant. We therefore need not branch over all partitions of $\mathcal{U}$ but only over those of U_{i+u}. If ϕ does not contain a $^+$-operator, then the maximal number of users in business events in i is $m+1$ and therefore $|U_{i+u}| \leq m+2$. If ϕ does contain a $^+$-operator, then $|U_{i+u}| \leq |\mathcal{U}|$. Our implementation exploits these observations. Hence, its runtime complexity is in $O(n\,m\,2^{|U_{i+u}|}\,p(|\mathcal{U}|,|\mathcal{R}|))$ for $|U_{i+u}|$ as discussed above.

Our experience with business process catalogs, such as the IBM Insurance Application Architecture (IAA) [IBM 2009], is that workflows contain a good dozen human tasks on the average. Furthermore, most workflow modeling languages allow the decomposition of workflows into sub-workflows. Hence, we conclude that the performance penalty imposed by the SoD as a Service approach remains acceptable for most workflows. We provide performance measurements that support this in the following section.

6.5 Performance Measurements

We return to the drug dispensation workflow introduced in Chapter 4, the term ϕ from Example 5.1, and the trace $i_4 = \langle t_1.\mathsf{Dave}, t_2.\mathsf{Emma}, \mathsf{add.Fritz.PrivacyAdvocate}, t_3.\mathsf{Fritz}, t_5.\mathsf{Bob}, \mathsf{add.Alice.Pharmacist}, t_7.\mathsf{Alice}, t_9.\mathsf{Gerda}, t_{10}.\mathsf{Gerda}, \checkmark \rangle$ from Example 5.4.

We modeled the workflow in BPEL, extended by BPEL4People, and deployed it on WPS. We set up the initial user-role assignment UR_1 using TDS' web interface and deployed ϕ, encoded as string, to the SoDA-enforcement monitor. Furthermore, we configured WPS to use our plugin to post-process the sets of users returned when evaluating people links. We then executed instances of the drug dispensation workflow. For example, we logged into WPS as Dave and started a workflow instance by submitting a form that corresponds to t_1 (Request Drugs). Next, we logged into the system as Emma, claimed the newly created instance of the task t_2 (Retrieve Patient Record), and executed it by filling in the corresponding form. Through TDS' web interface we then assigned Fritz to the role PrivacyAdvocate. Thereby, UR_1 evolved to UR_2. Afterward, we executed t_3 (Check Anonymization Requirements) as Fritz. This sequence of activities corre-

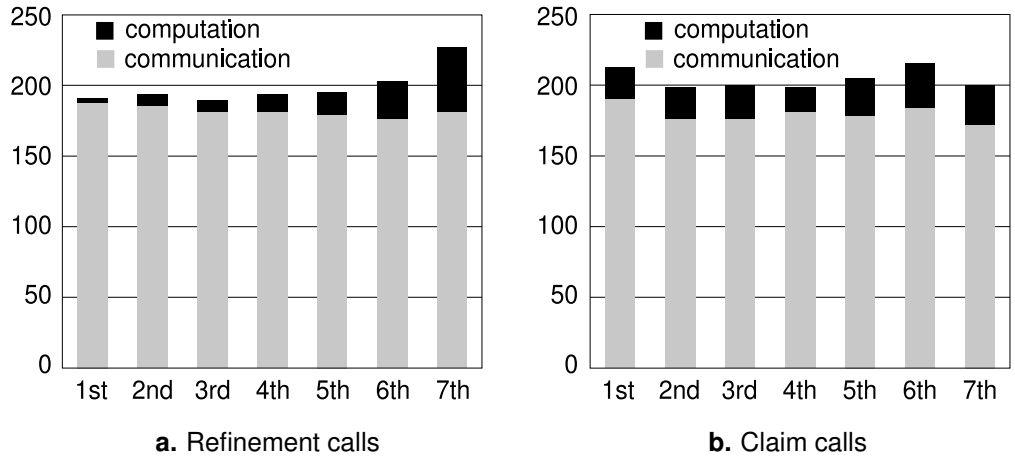


Figure 6.3. Average service call times in milliseconds (ms)

sponds to a prefix of i_4. In the following, we report on the average performance of ten executions of workflow instances corresponding to i_4.

Compared to the running time baseline, the running time of our prototype system is increased by a refinement and a claim call for every task instance. We call the time it takes to call a web service and to retrieve its return values the *total running time* of a web service, which we decompose into two parts: the *communication time* encompasses the time to serialize, transmit, and deserialize the exchanged data and the *computation time* is the time to execute the service's functionality. Figure 6.3 illustrates the averaged communication and computation times in milliseconds (ms) for the ith task instance, for $i \in \{1,\ldots,7\}$.

The communication time depends on various factors including the network throughput, latency, the payload size, and also the time taken to serialize Java objects to SOAP message parameters using the Apache Axis framework. We run the service client and the SoDA-enforcement monitor on two different computers at the same geographical location, connected by a standard enterprise network with an average latency of 1 ms. Both computers have off-the-shelf configurations.[1] The communication time averages between 150 ms and 200 ms per call.

The computation time for claim calls was always around 24 ms. The computation time of refinement calls, however, increased with the number of executed task instances. As derived in Section 6.4, the operators in ϕ cause this time to increase exponentially.

[1]Client: MS Windows XP on Intel Core Duo 2 GHz processor with 3 GB RAM. Server: MS Windows Server 2003 on Intel Xeon 2.9 GHz processor with 4 GB RAM.

Finally, we compare the total running time of these additional calls to the time it takes to execute a task instance in a system without a SoDA-enforcement monitor. The refinement call increases the time between the termination of a preceding task instance and the moment the new task instance is ready to be claimed by a user. The durations for these steps range between 2 and 15 seconds, depending on the load on WPS and the latest patches installed on it. Claiming a new task instance takes only 1–3 seconds. A user clicks on the instance in his inbox and the corresponding form is displayed on his screen. In both cases, the additional running time caused by the SoDA-enforcement monitor calls is an order of magnitude smaller than the running time baseline, which varied between 2 and 5 seconds.

Given the observations made in the previous section and the times reported here, we conclude that the integration of our SoD as a Service implementation into an existing workflow system imposes a performance penalty below 10%. Consequently, we achieved all the objectives described in Section 6.1.

Chapter 7
Evaluation

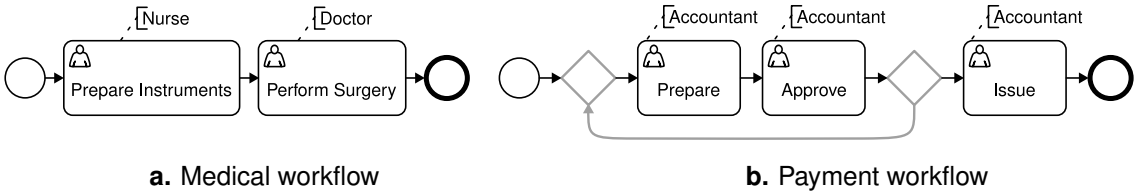
We conclude Part II with an evaluation of our workflow-independent approach to enforcing abstract SoD constraints in a dynamic workflow environment, developed in the previous chapters. Many of the shortcomings presented below serve as motivation for Part III.

7.1 Limitations of an Automated Mapping

As elaborated in Chapter 5, the abstract nature of SoDA has valuable advantages. However, it also poses a challenge when mapping terms onto workflows. In particular, Li and Wang's set-based semantics SoDA$^{\mathsf{S}}$ does not specify a mapping between subterms and tasks. We use the workflows in Figure 7.1 to evaluate our generalization of SoDA$^{\mathsf{S}}$ to SoDA$^{\mathsf{T}}$, our automated mapping to processes, and explore directions for future work.

Consider the medical workflow shown in Figure 7.1a and suppose that we want to enforce the SoDA term $\mathsf{Nurse} \otimes \mathsf{Doctor}$. This term does not specify whether the Doctor performs the surgery and the Nurse prepares the instruments or *vice versa*. We solve this problem by incorporating an RBAC policy into the SoDA-secure process. For this medical workflow, the depicted role-task assignment rules out workflow instances where the Nurse performs the surgery and the Doctor prepares the instruments.

However, incorporating an RBAC policy does not solve all refinement questions related to mapping terms to processes. In particular, when duties are to be separated between tasks assigned to the same role, an RBAC policy is of little help. Consider the payment workflow in Figure 7.1b, ignoring the gray parts for the moment, and the term $\mathsf{Accountant} \otimes \mathsf{Accountant}^{+}$. Implicitly, we would assume that a separation of duties between the tasks Prepare and Approve is intended. However, a trace that corresponds to a workflow instance where one Accountant executes Prepare and Approve and another Accountant executes Issue is also accepted by the SoDA-secure process. A semi-automated approach where



a. Medical workflow **b.** Payment workflow

Figure 7.1. Strengths and weaknesses of role-based refinement

subterms are manually mapped to tasks would solve this problem. However, we studied only fully automated mappings and consider semi-automated solutions outside the scope of this dissertation; left to future work.

An inherent weakness of SoDA that is revealed by an automated mapping is its poor support for loops. Consider the payment workflow in Figure 7.1b, now also including the gray elements, *i.e.* looping over the tasks Prepare and Approve until the payment is approved. SoDA provides no means to specify an SoD constraint for each loop iteration. For example, we cannot specify that each pair of instances of Prepare and Approve must be executed by different users. Only terms containing a $^+$-operator map to SoDA-enforcement processes that accept an arbitrary number of execution events. By SoDA's syntax $\mathfrak{S}$ however, the $^+$-operator only ranges over unit terms. Li and Wang [2008] motivate this design decisions with the *psychological acceptability principle* postulated by Saltzer and Schroeder [1975]. As a consequence, our approach supports only finitely many SoD constraints, *i.e.* $\otimes$-operators, per term and does therefore not support loops in their full generality. Decomposing workflows into sub-workflows that do not contain loops is a possible solution to overcome this limitation. However, it is not fully automated either and also remains as future work. The concept of *release*, which we introduce in Chapter 8, supports a scoping of authorization constraints and is therefore a candidate solution for manually decomposing workflows and refining SoDA's subterm-task mapping.

7.2 Continuous Satisfiability

The original semantics for SoDA, SoDA$^{\mathsf{S}}$, as well as our generalizations SoDA$^{\mathsf{M}}$ and SoDA$^{\mathsf{T}}$ provide a binary decision as to whether a set, multiset, or trace, respectively, satisfy a given term. For example, consider a term ϕ and a trace i. SoDA$^{\mathsf{T}}$ tells us whether i satisfies ϕ but it makes no statement whether there exists a trace i' such that $i \frown i'$ satisfies ϕ. In other words, SoDA's notion of satisfaction does not mean "we can still fulfill all constraints" but rather "all constraints have been fulfilled". As a consequence, a workflow trace corresponding

to a workflow instance that has just been started typically does not satisfy ϕ. Only when engaging in the final event ✓ is ϕ supposed to be satisfied. This is also reflected in Theorem 5.1, which makes only statements about successfully terminated workflow instances and not about their prefixes.

Developing an enforcement monitor that continuously ensures that every accepted prefix can be extended to a workflow trace that satisfies ϕ is therefore another direction for future work. In Chapter 9, we formalize a generalization of this problem by the notion of an *obstruction* and introduce enforcement processes, which ensure an obstruction-free authorization enforcement.

7.3 Communication Versus Statefulness

A SoDA-enforcement process $SODA_\phi(UR)$ is parametrized by the user-assignment relation UR that is modified when the process engages in administrative events. Our SoDA-enforcement monitor, however, does not store all tuples of UR. It receives all relevant tuples as call parameters and stores only those of users who claim a task instance. Although this approach increases the communication overhead between WPS and the SoDA-enforcement monitor, it reduces unnecessary replication. In fact, user repositories of large enterprises may contain thousands of entries and only a few of them may be relevant with respect to a given workflow instance.

Our SoDA-enforcement monitor is stateful because the enforcement of SoD constraints ranges over multiple tasks and may depend on user-role assignments. The service must keep track of the users who execute task instances and the roles they act in at that time. Workflow engines such as WPS keep track of the users who execute task instances but they do not store the history of their assignments to roles. This information is stored in the SoDA-enforcement monitor; the workflow engine and the user repository remain unchanged.

7.4 Abstractions

For simplicity, our SoDA-enforcement monitor does not cope with the abort or suspension of task instances. In practice, however, WPS users can hand back unfinished task instances to the workflow engine or trigger the abortion of a workflow instance. Furthermore, we enforce exactly one term per workflow. This is not a limitation as two or more terms can be combined into a single term with the appropriate SoDA operators; *e.g.* $\sqcap$ for a conjunction or $\sqcup$ for a disjunction. If no SoD constraint must be enforced, the term All^+, which is satisfied by every non-empty multiset of users, can be used.

Part III

A Workflow-Specific Approach

Chapter 8
Scoping Constraints With Release Points

To separate or bind duties between tasks in a history-dependent manner, we must keep track of which users executed previous instances of these tasks in order to determine who is authorized to execute future instances. Thus, we build up associations between task instances and users during workflow execution. Future authorization decisions in turn depend on these associations. In this chapter, we introduce the concept of *release*, which removes such associations and thereby scopes authorizations to subsets of task instances. We annotate workflow models with so-called *release points*. At run time, releasing is triggered when the control-flow passes through a release point, linking the enforcement of authorizations and a workflow's control-flow. Hence, this approach is workflow-specific.

8.1 Formalization

Analogous to Part II, we use CSP to formalize workflows and authorizations that constrain their execution. We build on previous definitions but refine the workflow formalization introduced in Chapter 4. In particular, we change the underlying set of events and model workflows at two levels of abstraction.

8.1.1 Workflows

At the *specification level*, a workflow models the tasks and their causal dependencies that together implement a business objective. The *execution level* refines the specification level and also models who executes which task. The benefit of this distinction becomes evident in Chapter 9. For now, we only make the observation that the specification of a workflow's control-flow, and thereby its business objective, is independent of who is executing tasks. For example, the workflow

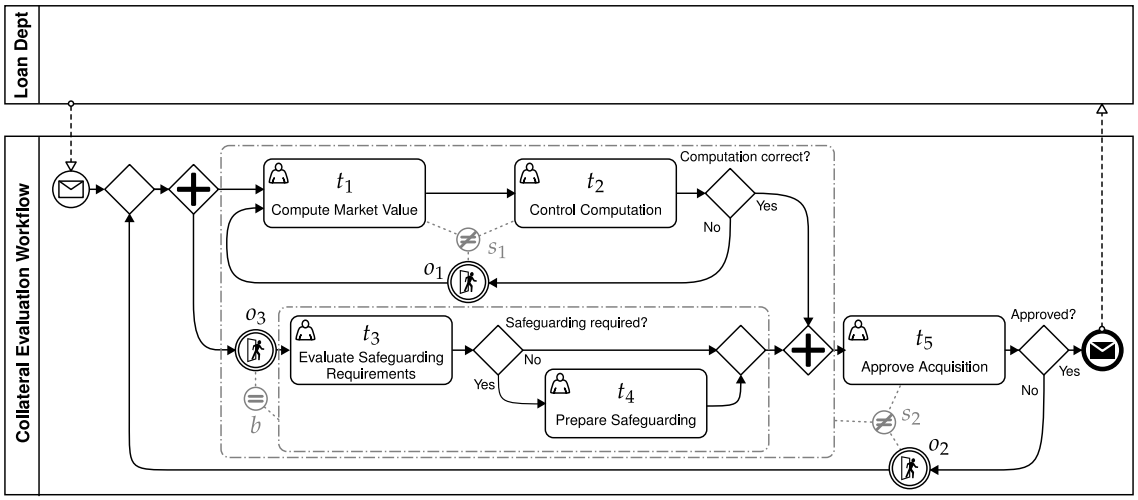


Figure 8.1. BPMN model of the collateral evaluation workflow

execution process modeling the drug dispensation workflow in Example 4.1 allows every task to be executed by any user. Thus, we may just as well abstract away from users when specifying workflows.

For the remainder of this part, let $\mathcal{O}$ be a set of *points*, which we use to model BPMN events. At the specification level, workflows are then formalized as follows.

Definition 8.1 (Workflow Specification Process) *A* workflow specification process *is a process W such that* $\mathsf{T}(W) \subseteq (\mathcal{T} \cup \mathcal{O})^{*\checkmark}$.

In other words, a workflow specification process may engage in tasks, points, and finally the event $\checkmark$. We give below an example workflow, visualized in BPMN, and a corresponding workflow specification process. This workflow serves as running example for this and the next chapter.

Example 8.1 (Collateral Evaluation Workflow) The financial industry distinguishes between secured and unsecured loans. In a secured loan, the borrower pledges some asset, such as a house or a car, as collateral for his debt. If the borrower defaults, the creditor takes possession of the asset to mitigate his financial loss.

Figure 8.1 shows a BPMN model of the *collateral evaluation workflow*, which we adopted from IBM's Information FrameWork [2010b]. Ignore the gray BPMN elements for the moment. This workflow is executed by a financial institution to evaluate, accept, and prepare the safeguarding of the collateral that a borrower pledges in return for a secured loan.

For this example, let $\mathcal{T} = \{t_1, \ldots, t_5\}$ where t_1 refers to Compute Market Value, t_2 to Control Computation, *etc.*, and $\mathcal{O} = \{o_1, o_2, o_3\}$, as shown in Figure 8.1. The

workflow specification process W models the collateral evaluation workflow as follows.

$$\begin{aligned} W &= (W_1 \,|||\, W_2)\,;\,(t_5 \to ((o_2 \to W) \sqcap \mathit{SKIP})) \\ W_1 &= t_1 \to t_2 \to ((o_1 \to W_1) \sqcap \mathit{SKIP}) \\ W_2 &= o_3 \to t_3 \to ((t_4 \to \mathit{SKIP}) \sqcap \mathit{SKIP}) \end{aligned}$$

We do not model data-flow in our example and therefore overapproximate gateway decisions with CSP's internal choice operator $\sqcap$. ★

Next, we model workflows at the execution level. The auxiliary relation $\pi = \{(t.u, t) \mid t \in \mathcal{T}, u \in \mathcal{U}\}$, which maps every execution event $t.u$ to the task t, links the specification level and the execution level. Given a workflow specification process W, the process $W[\pi^{-1}]$ then models W at the execution level. It engages in the execution event $t.u$, for any $u \in \mathcal{U}$, if W engages in the task t. The application of π^{-1} to W has no effect on points and ✓; *i.e.* if W engages in a point or ✓, then so does $W[\pi^{-1}]$. We may abuse the renaming notation to map a trace $i \in \mathsf{T}(W[\pi^{-1}])$ to a trace $i[\pi] \in \mathsf{T}(W)$.

We disregard in Part III administrative events and thus $\Sigma = \mathcal{T} \cup \mathcal{X} \cup \mathcal{O}$. Note that what we call a workflow execution process in Part II corresponds in this part to a workflow specification process W mapped to the execution level, *i.e.* $W[\pi^{-1}]$. Furthermore, we adjust the definition of a workflow trace to also incorporate a workflow's points.

Definition 8.2 (Workflow Trace) *A* workflow trace *is a trace $i \in (\mathcal{X} \cup \mathcal{O})^{*\checkmark}$. In particular, for a workflow specification process W, if $i \in \mathsf{T}(W[\pi^{-1}])$ then i is a workflow trace of W.*

Example 8.2 (Workflow Traces) Let $\mathcal{U} = \{\mathsf{Alice}, \mathsf{Bob}, \mathsf{Claire}, \mathsf{Dave}\}$ for the collateral evaluation workflow. Consider the following workflow traces:

$$\begin{aligned} i_1 &= \langle t_1.\mathsf{Alice},\ t_2.\mathsf{Bob},\ t_4.\mathsf{Claire}\rangle \\ i_2 &= \langle t_1.\mathsf{Alice},\ o_3,\ t_3.\mathsf{Bob},\ t_2.\mathsf{Alice},\ o_1,\ t_1.\mathsf{Bob},\ t_2.\mathsf{Claire},\ t_5.\mathsf{Claire},\ \checkmark\rangle \\ i_3 &= \langle t_1.\mathsf{Alice},\ o_3,\ t_3.\mathsf{Bob},\ t_2.\mathsf{Bob},\ o_1,\ t_1.\mathsf{Alice},\ t_4.\mathsf{Dave},\ t_2.\mathsf{Claire},\ t_5.\mathsf{Claire},\ \checkmark\rangle \\ i_4 &= \langle t_1.\mathsf{Alice},\ o_3,\ t_3.\mathsf{Bob},\ t_2.\mathsf{Bob},\ o_1,\ t_1.\mathsf{Bob},\ t_4.\mathsf{Bob},\ t_2.\mathsf{Claire},\ t_5.\mathsf{Dave},\ \checkmark\rangle \end{aligned}$$

The traces i_2, i_3, and i_4 model successfully terminated workflow instances of the collateral evaluation workflow, where the inner loop was executed twice, *i.e.* $i_2, i_3, i_4 \in \mathsf{T}(W[\pi^{-1}])$. We discuss the differences between these traces in later examples. The trace i_1, however, neither models a successfully terminated workflow instance nor is it a workflow trace of $W[\pi^{-1}]$ because t_4 can only be executed after t_3 has been executed. ★

Example 8.2 supports our observation in Chapter 4 that successfully terminated workflow instances may contain multiple instances of a task. For example,

t_2 and t_4 are part of the collateral evaluation workflow and i_2 contains two execution events involving t_2 but none involving t_4.

8.1.2 SoD Constraints

Let T_1 and T_2 be two non-empty, disjoint sets of tasks , *i.e.* $|T_1| \geq 1$, $|T_2| \geq 1$, and $T_1 \cap T_2 = \varnothing$, and let O be a set of points. An *SoD constraint* is a triple (T_1, T_2, O).

Definition 8.3 (SoD Process) *For an SoD constraint $s = (T_1, T_2, O)$, the* SoD process *for s is the process $A_s(\mathcal{U}, \mathcal{U})$ where*

$$\begin{aligned} A_s(U_{T_1}, U_{T_2}) = \; & t : T_1 . u : U_{T_1} \to A_s(U_{T_1}, U_{T_2} \setminus \{u\}) \\ & \Box\, t : T_2 . u : U_{T_2} \to A_s(U_{T_1} \setminus \{u\}, U_{T_2}) \\ & \Box\, o : O \to A_s(\mathcal{U}, \mathcal{U}) \\ & \Box\, t : \mathcal{T} \setminus (T_1 \cup T_2) . u : \mathcal{U} \to A_s(U_{T_1}, U_{T_2}) \\ & \Box\, o : \mathcal{O} \setminus O \to A_s(U_{T_1}, U_{T_2}) \\ & \Box\, \mathit{SKIP} \; . \end{aligned}$$

An SoD process $A_s(U_{T_1}, U_{T_2})$ offers the (external) choice between six kinds of events. (1) For a task $t \in T_1$ and user $u \in U_{T_1}$, A_s engages in the execution event $t.u$. Afterward, A_s associates u with T_1 by removing u from U_{T_2} and thereby blocking u from executing future instances of tasks in T_2. (2) Symmetrically, A_s associates a user $u \in U_{T_2}$ with T_2 and blocks u from executing future instances of tasks in T_1 after executing an instance of a task in T_2. (3) By engaging in a point $o \in O$, A_s releases all users from their associations with T_1 and T_2. We therefore call a point used in an SoD (or BoD) constraint a *release point*. (4) A_s engages also in every execution event involving tasks other than T_1 and T_2 and (5) points other than O without changing its behavior. (6) Finally, A_s may behave like *SKIP* and terminate at any time.

We may use the following shorthand notation to describe SoD constraints and to avoid cluttering graphical workflow models. Consider the SoD constraint (T_1, T_2, O). If T_1, T_2, or O are singleton sets, we simply use the respective element and omit the set notation. For example, if $T_1 = \{t_1\}$, $T_2 = \{t_2\}$, and $O = \{o\}$, we write (t_1, t_2, o).

To visualize SoD constraints in BPMN, we introduce a novel class of internal (BPMN) events, called *release events*. This facilitates the description of releasing as part of a workflow's control-flow. The release event icon is a user who leaves a door, as shown in Figure 8.1 with o_1, o_2, and o_3. We use the dot-dashed BPMN notation for grouping tasks to specify sets of tasks. For example, Figure 8.1

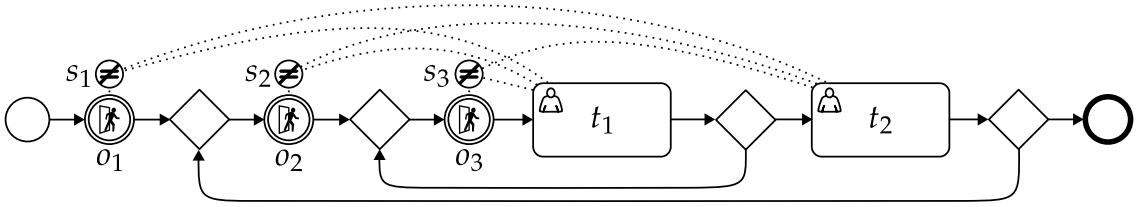


Figure 8.2. Location matters: The placement of a release point within a workflow effects the semantics of the respective SoD constraint.

contains a group denoting the set of tasks $\{t_1, t_2, t_3, t_4\}$. An SoD constraint is graphically described by linking two disjoint, non-empty sets of tasks and a set of release events with a dotted line, joined by a node labeled with the symbol "$\neq$". This notation is an adaptation of BPMN's textual annotation of tasks. If one of the sets of tasks is a singleton set, we may omit the BPMN grouping and directly link the respective task and the $\neq$-node. For example, Figure 8.1 contains the SoD constraint $s_2 = (\{t_1, t_2, t_3, t_4\}, t_5, o_1)$.

The effect of an SoD constraint is only fully defined with respect to a workflow specification process, which defines the order in which task instances are executed and release points are reached. We illustrate the effect of different placements of a release point with an example.

Example 8.3 (Release Point Placement) Figure 8.2 shows a workflow with two tasks and three SoD constraints, $s_i = (t_1, t_2, o_i)$ for $i \in \{1,2,3\}$. Successfully terminated instances of this workflow correspond to workflow traces of the form

$$\begin{aligned}\langle\, & o_1, o_2, o_3, t_1.u_{1,1}, \ldots, o_3, t_1.u_{1,n_1}, t_2.u_{1,n_1+1}, \\ & o_2, o_3, t_1.u_{2,1}, \ldots, o_3, t_1.u_{2,n_2}, t_2.u_{2,n_2+1}, \\ & \ldots \\ & o_2, o_3, t_1.u_{m,1}, \ldots, o_3, t_1.u_{m,n_m}, t_2.u_{m,n_m+1}, \checkmark \,\rangle\end{aligned}$$

for $n_m, m \geq 1$. The only difference between s_1, s_2, and s_3 is the position of the respective release point within the workflow. The SoD constraint s_1 is satisfied if $\{u_{1,1}, u_{1,2}, \ldots, u_{1,n_1}, u_{2,1}, \ldots, u_{m,n_m}\} \cap \{u_{1,n_1+1}, u_{2,n_2+1}, \ldots, u_{m,n_m+1}\} = \varnothing$. In other words, s_1 is satisfied if no user who executes instances of t_1 executes instances of t_2 and *vice versa*. Because o_1 is reached only once and before any task instance is executed, effectively no releasing takes place. Reaching a release point that is placed at the very start or end of a workflow has no effect and, hence, the constraint separates duties over all instances of the respective tasks. This illustrates that our policies are more expressive than existing SoD formalisms that do not distinguish between different instances of the same task.

Let $k \in \{1,2,\ldots,m\}$. The SoD constraint s_2 is satisfied if $u_{k,n_k+1} \notin \{u_{k,1}, u_{k,2}, \ldots, u_{k,n_k}\}$. That is, for every execution of the workflow's outer loop, s_2 separates the duties between users who execute instances of t_1 and those who execute instances of t_2. Finally, s_3 is satisfied if $u_{k,n_k} \neq u_{k,n_k+1}$. Thus, in every execution of the workflow's outer loop, only the user who executes the last instance of t_1 must differ from the user who executes t_2's instance. It follows that a workflow instance that satisfies s_1 also satisfies s_2 and s_3. Moreover, an instance satisfying s_2 also satisfies s_3. ★

8.1.3 BoD Constraints

Assume we want to bind duties between a set of tasks T. At first, every user is authorized to execute an instance of a task in T. Once a user has executed an instance of a task in T, no other user is authorized to execute future instances of tasks in T anymore. Again we use release points to scope BoD constraints to subsets of task instances.

Let T be a non-empty set of tasks, *i.e.* $|T| \geq 1$, and let O be a set of points. A *BoD constraint* is a tuple (T,O).

Definition 8.4 (BoD Process) *For a BoD constraint $b = (T,O)$, the* BoD process *for b is the process $A_b(\mathcal{U})$ where*

$$\begin{aligned} A_b(U) \;=\; & t : T.u : U \to A_b(\{u\}) \\ & \Box\, o : O \to A_b(\mathcal{U}) \\ & \Box\, t : \mathcal{T} \setminus T.u : \mathcal{U} \to A_b(U) \\ & \Box\, o : \mathcal{O} \setminus O \to A_b(U) \\ & \Box\, \mathit{SKIP}\ . \end{aligned}$$

The BoD process $A_b(U)$ offers the external choice between five kinds of events. (1) It engages in every execution event $t.u$ for $t \in T$ and $u \in U$. Initially $U = \mathcal{U}$. Once a user u executes an instance of a task in T, U is updated to $\{u\}$. Only after engaging in one of the release points in O are users other than u authorized to execute instances of tasks in T again. Thus, for $t \in T$, executing $t.u$ "binds" u to T until (2) an $o \in O$ is reached and u is released. In particular, for $|T| = 1$ the respective BoD constraint binds the duties of all instances of a single task. Similar to SoD processes, $A_b(U)$ engages (3) in every execution event involving tasks other than those in T, (4) points other than the ones in O, and (5) may behave like *SKIP* and terminate at any time.

As with SoD constraints, we visualize BoD constraints in BPMN by linking a non-empty set of tasks and a set of release events with a dotted line, joined by

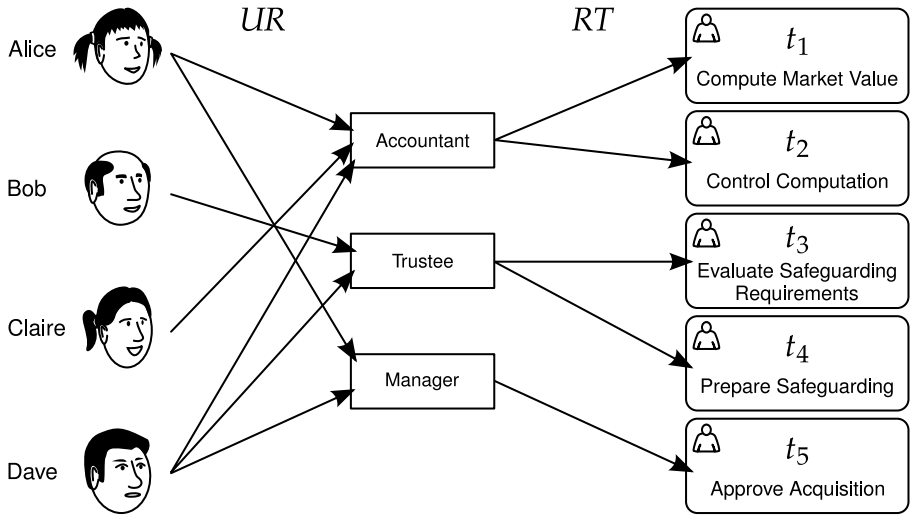


Figure 8.3. RBAC policy for the collateral evaluation workflow

a node labeled with the symbol "=". We may also use the shorthand notation introduced for SoD constraints. For example, Figure 8.1 contains the BoD constraint $b = (\{t_3, t_4\}, o_3)$. Similar to SoD processes, the placement of release points with respect to a workflow specification process effects the semantics of a BoD constraint.

8.1.4 Composition

Let UT be a user-task assignment, S a set of SoD constraints, and B a set of BoD constraints. The triple (UT, S, B), called an *authorization policy*, combines workflow-independent and history-independent authorizations in the form of UT and workflow-specific, history-dependent authorizations in the form of S and B. We define the semantics of authorization policies by composing the respective processes.

Definition 8.5 (Authorization Process) *For an authorization policy $\phi = (UT, S, B)$, the* authorization process *for ϕ is the process*

$$A_\phi = A_{UT} \parallel (\mathop{\Vert}_{s \in S} A_s) \parallel (\mathop{\Vert}_{b \in B} A_b) .$$

By the trace semantics of CSP, a workflow trace i satisfies an authorization policy $\phi = (UT, S, B)$, *i.e.* $i \in \mathsf{T}(A_\phi)$, if and only if i satisfies UT, all SoD constraints in S, and all BoD constraints in B. Given a workflow specification process W, we say ϕ is an *authorization policy for W* if all tasks and points in ϕ appear in W. In the following example, we provide an authorization policy for the collateral evaluation workflow.

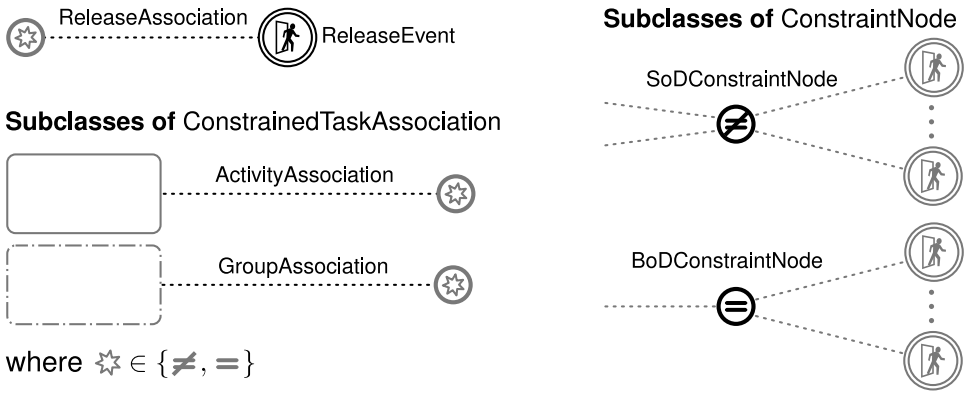


Figure 8.4. BPMN extension for modeling authorization constraints

Example 8.4 (Authorization Policy) Consider the authorization policy $\phi = (UT, S, B)$, where $UT = RT \circ UR$, for the user-role assignment UR and the role-task assignment RT illustrated in Figure 8.3, and $S = \{s_1, s_2\}$ and $B = \{b\}$ illustrated in Figure 8.1. Furthermore, consider the traces i_2, i_3, and i_4 of Example 8.2, which model successfully terminated instances of the collateral evaluation workflow. Trace i_2 does not satisfy ϕ because Alice executed instances of t_1 and t_2 before reaching o_1, thereby violating s_1. Trace i_3 does not satisfy ϕ for several reasons: s_2 is violated because Claire executed an instance of t_2 and an instance of t_5, b is violated because the instances of t_3 and t_4 are not executed by the same user, and Claire is not authorized to execute instances of t_5 with respect to UT. However, i_4 satisfies ϕ. ★

8.2 BPMN Extension and Serialization

We return to BPMN's meta-model introduced in Section 2.4. Figure 2.2 shows BPMN's meta-model classes in white and the new classes that we defined for our extension in gray. Furthermore, Figure 8.4 shows the concrete notation used to visualize the respective new modeling elements. New modeling elements are shown in black and how they are connected to existing or other new elements is illustrated in gray.

Let $s = (T_1, T_2, O)$ be an SoD constraint. Using our BPMN extension, s is modeled by a combination of new and existing modeling elements. The class SoDConstraintNode connects all relevant elements. Each set of tasks is either modeled by an instance of Activity or they are identified by an instance of Group. The respective classes are connected to the SoDConstraintNode by instances of ConstrainedTaskAssociation. Each release point in O is modeled by an instance of class ReleaseEvent and all of them are connected to the SoDConstraintNode by

instances of ReleaseAssociation. A BoD constraint $b = (T, O)$ is modeled analogously. Instead of SoDConstraintNode, the central class is BoDConstraintNode and instead of two sets of tasks only one set of tasks is connected to it.

The BPMN standard provides an extension mechanism [OMG 2011a]. Using the extended meta-model as a blueprint, we specified an XML schema for our BPMN extension. As Figure 2.2 shows, new modeling elements can be easily defined by connecting to, and inheriting from, existing elements. Correspondingly, our new schema file is only a few dozen lines long. We modeled the collateral evaluation workflow in BPMN including our extensions and serialized the model in XML. Afterward, we successfully validated the XML file against the official BPMN XML schema and the XML schema specifying our extension.

8.3 Tool Support

We implemented tool support for our BPMN extension by extending the modeling platform Oryx [2012]. Our objective was to gain modeling experience with our BPMN extension, demonstrate its expressivity in a hands-on fashion, and validate its ease of use.

Oryx is a good choice for our purpose. First, Oryx is designed to be extensible. As a result, our implementation required little programming effort. Second, Oryx's architecture and code is well-documented and mature. In particular, it is the basis for commercial tools such as Signavio's Process Editor [Signavio 2012] and the Activiti BPM Platform [Activiti 2012]. Third, Oryx's source code is freely available under the MIT license [OSI 2012], which gives us full access to all implementation details and does not impede a potential commercial exploitation. Finally, Oryx's web-based architecture is ideal for demonstration purposes because BPMN processes are modeled directly in a web browser and no extra software need be installed.

Oryx adopts a standard three-tier architecture, with a web browser acting as the presentation tier, a J2EE server as the application tier, and a database as the data tier. The implementation provides extension mechanisms in the presentation and application tier. We report on the performance of an extension we made to the application layer in Section 9.4.3. In the following, we describe our extension of the presentation layer to support our BPMN extension.

Oryx groups modeling elements and defines their visualization in so-called *stencil sets* [Polak 2007]. A stencil set may extend existing stencil sets, thereby extending an existing modeling language. We defined a stencil set that specifies the modeling elements of our BPMN extension, as introduced in Section 8.2, extending Oryx's existing BPMN stencil sets. Figure 8.5 shows the user interface

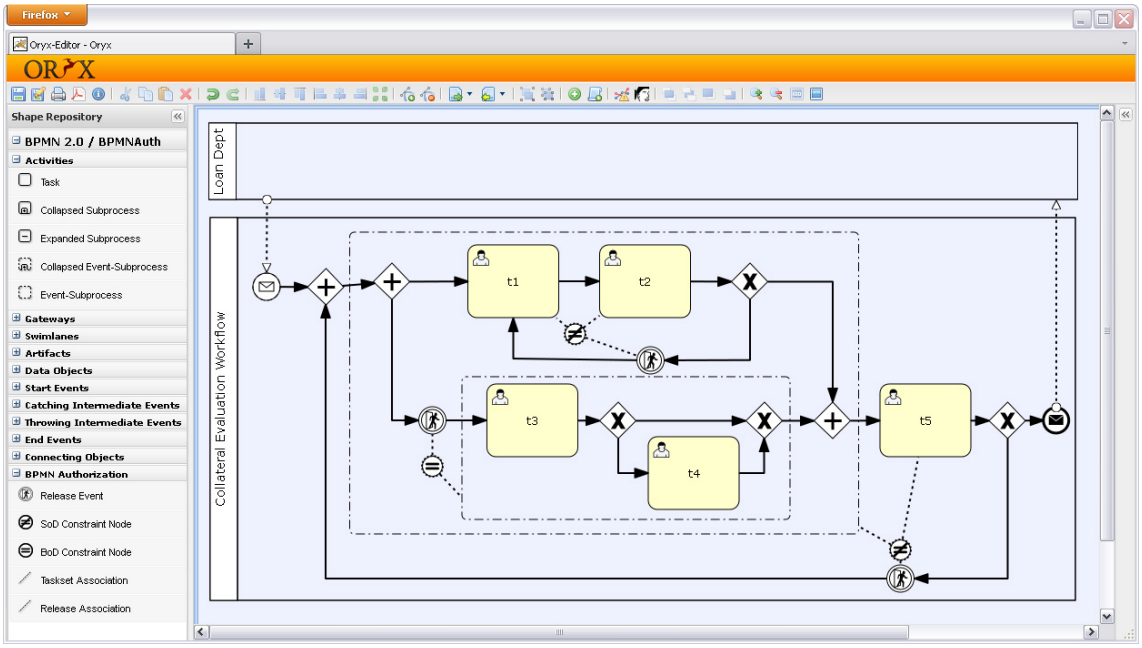


Figure 8.5. Screenshot of Oryx BPMN editor including authorization extension

of Oryx's BPMN editor. Each palette on the left corresponds to a stencil set; BPMN's standard stencil sets are located on the top and our additional stencil set is at the bottom. A BPMN model of the collateral evaluation workflow is shown on the right, combining modeling elements from various stencil sets including our new one.

Chapter 9

Aligning Authorization and Business Objectives

In this chapter, we investigate the question of how to enforce an authorization policy on a workflow without obstructing the workflow's underlying business objectives. To this end, we introduce the notion of an *obstruction*, formalizing the misalignment of authorizations and business objectives. We start with a motivational example.

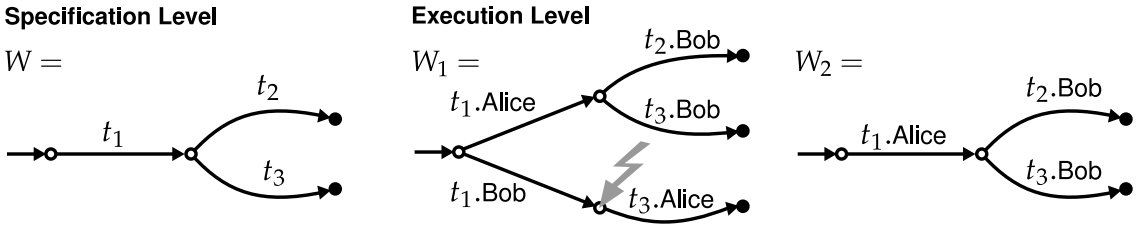


Figure 9.1. Enforcement with and without obstruction

Example 9.1 (Obstruction Motivation) Consider again the simple payment workflow and the abstract SoD policy P introduced in Example 1.1. Let UT be the user-task assignment illustrated in Figure 1.1, let $s = (t_1, \{t_2, t_3\})$ be the SoD constraint that formalizes P, and thus the authorization policy for this example is $\phi = (UT, \{s\}, \varnothing)$. As illustrated in Figure 9.1, we formalize the simple payment workflow as the labelled transition system W and study its refinements W_1 and W_2, both respecting ϕ. In W_1, Bob may execute t_1 but afterward only t_3 is executable without violating ϕ. This, however, corresponds to a restriction of the workflow at the specification level, indicated by the jagged arrow. We call this situation an obstruction. In contrast, W_2 avoids obstructions by being more restrictive than W_1 and not allowing Bob to execute t_1. ★

This example illustrates the tension between authorization and business objectives and suggests that authorization enforcement should be designed in a

way that aligns both objectives. Our underlying assumption is that for achieving business objectives, it does not matter who is executing a task as long as every task can be executed by an authorized user. As illustrated by Example 9.1, we give the preservation of a workflow at the specification level priority over the choice of who can execute a task at the execution level. We proceed by formulating the existence of an obstruction-free authorization enforcement as a decision problem and analyzing its complexity.

9.1 Obstruction

We link the specification and execution level by the notion of an obstruction.

Definition 9.1 (Obstruction) *Let W be a workflow specification process, ϕ an authorization policy, and $i \in \mathsf{T}(W[\pi^{-1}])$ a workflow trace of W. We say that i is* obstructed *if there exists a task t such that $i[\pi]\hat{}t \in \mathsf{T}(W)$ but there does not exist a user u such that $i\hat{}\langle t.u\rangle$ satisfies ϕ.*

An obstruction describes a state of a workflow instance where the enforcement of the authorization policy conflicts with the business objective represented by the workflow. At the specification level, the business objective can be achieved by executing a task t but at the execution level there is no user who is authorized to execute t without violating the authorization policy ϕ.

Example 9.2 (Obstructed Workflow Trace) Consider the workflow specification process W and the authorization policy ϕ introduced in Examples 8.1 and 8.4, respectively. Furthermore, consider the workflow trace $i = \langle t_1.\mathsf{Alice}, t_2.\mathsf{Claire}, t_3.\mathsf{Dave}, t_4.\mathsf{Dave}\rangle$, modeling an instance of the collateral evaluation workflow, *i.e.* $i \in \mathsf{T}(W[\pi^{-1}])$. After executing the workflow instance corresponding to i, an instance of task t_5 can be executed according to the collateral evaluation workflow, *i.e.* $i[\pi]\hat{}t_5 \in \mathsf{T}(W)$. However, the only users who are authorized to execute instances of t_5 with respect to *UT* are Alice and Dave, but neither $i\hat{}\langle t_5.\mathsf{Alice}\rangle$ nor $i\hat{}\langle t_5.\mathsf{Dave}\rangle$ satisfy ϕ. Hence, i is obstructed. In this example, the workflow instance cannot even successfully terminate without violating ϕ. ⋆

9.2 Enforcement Processes

We describe the enforcement of an authorization policy on a workflow specification process W in terms of a process E that executes in parallel with $W[\pi^{-1}]$, formally $W[\pi^{-1}] \parallel E$.

Definition 9.2 (Enforcement Process) *Let a workflow specification process W and an authorization policy ϕ for W be given. An* enforcement process *for ϕ on W, written $E_{\phi,W}$, is a process that satisfies the conditions*

(1) $A_\phi \sqsubseteq_{\mathsf{T}} E_{\phi,W}$ *and*

(2) $(W[\pi^{-1}] \parallel E_{\phi,W})[\pi] =_{\mathsf{F}} W$.

Unlike the authorization process, the enforcement process not only implements the authorization policy ϕ but also takes W into account. Condition (1) states that $E_{\phi,W}$ is at least as restrictive as A_ϕ. The failure equivalence used in condition (2) states that at the specification level W is indistinguishable from W constrained by $E_{\phi,W}$.

Suppose $E_{\phi,W}$ is an enforcement process for ϕ on W. By CSP's traces model, if $i \in \mathsf{T}(W[\pi^{-1}] \parallel E_{\phi,W})$ then $i \in \mathsf{T}(W[\pi^{-1}])$ and $i \in \mathsf{T}(E_{\phi,W})$. For a task t, it follows by the failure equivalence of $(W[\pi^{-1}] \parallel E_{\phi,W})[\pi]$ and W, *i.e.* condition (2), that if $i[\pi]\hat{}t \in \mathsf{T}(W)$ then there exists a user u such that $i\hat{}\langle t.u\rangle \in \mathsf{T}(E_{\phi,W})$. By condition (1), it follows that $i\hat{}\langle t.u\rangle$ satisfies ϕ. Hence, i is not obstructed and $E_{\phi,W}$ is an obstruction-free enforcement of ϕ on W.

We now give an example of an enforcement process for the authorization-constrained collateral evaluation workflow.

Example 9.3 (Enforcement Process) Consider W and ϕ from Example 9.2 and the following processes.

$$\begin{aligned} E &= (E_1 \,|||\, E_2)\,;\, (t_5.\mathsf{Dave} \to ((o_2 \to E) \sqcap \mathit{SKIP})) \\ E_1 &= t_1.\mathsf{Alice} \to t_2.\mathsf{Claire} \to ((o_1 \to E_1) \sqcap \mathit{SKIP}) \\ E_2 &= o_3 \to t_3.\mathsf{Bob} \to ((t_4.\mathsf{Bob} \to \mathit{SKIP}) \sqcap \mathit{SKIP}) \end{aligned}$$

All traces of E satisfy ϕ and therefore condition (1) of Definition 9.2 holds. By the laws of CSP and the structure of E, $(W[\pi^{-1}] \parallel E)[\pi] = W[\pi^{-1}][\pi] \parallel E[\pi] = W \parallel W = W$ and therefore condition (2) holds too. Thus, E is an enforcement process for ϕ on W. ★

For illustration purposes, this example is rather simple in that all instances of the same task must be executed by the same user. For example, Alice is the only user who executes instances of t_1. Enforcement processes can, of course, be much more complex and also authorize multiple users to execute instances of the same task.

According to Definition 9.2, an authorization policy is only enforceable if a workflow remains unchanged at the specification level. This is a design decision and other options are possible. For example, one could choose to give authorizations precedence over an obstruction-free enforcement. However, even

if obstructed workflow instances are tolerated, our approach is helpful because it reveals tasks that may not be executed. The workflow can consequently be simplified without reducing the set of possible workflow instances.

9.3 The Enforcement Process Existence Problem

We now formulate the existence of an enforcement process as a decision problem and present complexity bounds.

Definition 9.3 (The Enforcement Process Existence (**EPE**) Problem)

Input: *A workflow specification process W and an authorization policy ϕ for W.*

Output: YES *if there exists an enforcement process for ϕ on W or* NO *otherwise.*

We first show that **EPE** is **NP**-hard by reducing the **NP**-hard ***k*-Coloring** problem, summarized in Section 2.3, to **EPE**.

Lemma 9.1 **EPE** *is* **NP***-hard.*

Proof. Given a ***k*-Coloring** instance consisting of a graph $G = (V, E)$ and an integer k, we describe a polynomial reduction to **EPE**. We construct a workflow specification process W and an authorization policy $\phi = (UT, S, B)$ and show that there exists a k-coloring for G if and only if there exists an enforcement process for ϕ on W. Let $\mathcal{T} = V$, for $V = \{v_1, v_2, \ldots, v_n\}$, and let $\mathcal{U} = \{1, 2, \ldots, k\}$. Now consider $W = v_1 \to v_2 \to \ldots \to v_n \to \mathit{SKIP}$, $UT = \mathcal{U} \times \mathcal{T}$, $B = \varnothing$, and for every edge $\{v_l, v_m\} \in E$ we construct an SoD constraint $(v_l, v_m, \varnothing)$. Figure 9.2 illustrates this construction for a graph with $n = 5$ and $k = 4$.

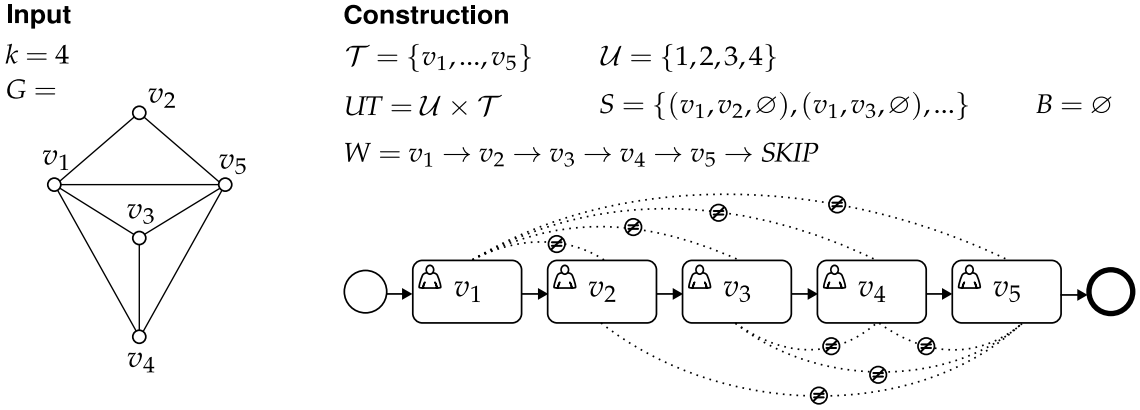


Figure 9.2. Reduction from ***k*-COLORING** to **EPE**

Let $h = \langle v_1, v_2, \ldots, v_n, \checkmark \rangle$. Note that by the construction of W, $h \in \mathsf{T}(W)$. If an algorithm for **EPE** returns YES, then an enforcement process $E_{\phi,W}$ exists by Definition 9.3 and $h \in \mathsf{T}((W[\pi^{-1}] \parallel E_{\phi,W})[\pi])$ by Definition 9.2. It follows that there

exists a workflow trace $i = \langle v_1.u_1, v_2.u_2, \ldots, v_n.u_n, \checkmark \rangle \in \mathsf{T}(W[\pi^{-1}] \parallel E_{\phi,W})$. By our construction, $u_j \in \{1,\ldots,k\}$, for $j \in \{1,\ldots,n\}$. Therefore, every task (*i.e.* node) is executed exactly once and thus associated with one of k users (*i.e.* colors). By Definition 9.2, $i \in \mathsf{T}(A_\phi)$ and therefore i satisfies every constraint in ϕ. In particular, for every SoD constraint $(v_l, v_m, \varnothing)$ in S, the user u_l who executes v_l is different from the user u_m who executes v_m. Hence, i describes a k-coloring for G.

Conversely, let $\mathsf{col} : V \to \{1,\ldots,k\}$ be a k-coloring for G and consider the process $P = v_1.\mathsf{col}(v_1) \to v_2.\mathsf{col}(v_2) \to \ldots \to v_n.\mathsf{col}(v_n) \to \mathit{SKIP}$. Let $i = \langle v_1.\mathsf{col}(v_1), v_2.\mathsf{col}(v_2), \ldots, v_n.\mathsf{col}(v_n), \checkmark \rangle$. By our construction and because col is a k-coloring of G, $i \in \mathsf{T}(A_s)$ for every $s \in S$. Furthermore, by our definition of UT, $i \in \mathsf{T}(A_{UT})$. It follows by Definition 8.5 and $B = \varnothing$ that $i \in \mathsf{T}(A_\phi)$. Because every trace in $\mathsf{T}(P)$ is a prefix of i, $A_\phi \sqsubseteq_\mathsf{T} P$. Furthermore, $P[\pi] =_\mathsf{F} W$ and therefore $(W[\pi^{-1}] \parallel P)[\pi] =_\mathsf{F} W$. Hence, P is an enforcement process for ϕ on W by Definition 9.2.

Because this reduction is in polynomial time, it follows that **EPE** is **NP**-hard. ■

We do not know whether **EPE** is in **NP**. However, it is decidable when $\mathcal{U}$ and W are finite.

Theorem 9.1 **EPE** *is decidable if $\mathcal{U}$ and W are finite.*

We sketch a proof here and give full details in Appendix A.5.

Proof Sketch. If $\mathcal{U}$ and W are finite, it follows by Definitions 4.3, 8.3, 8.4, and 8.5 and the operational semantics of CSP that A_ϕ is finite too. If there is an enforcement process $E_{\phi,W}$, it must satisfy the two conditions of Definition 9.2. Because A_ϕ is finite, for every process P, such that $A_\phi \sqsubseteq_\mathsf{T} P$, there is a finite labelled transition system that corresponds to P. We can therefore construct all processes P that are candidates to be $E_{\phi,W}$ with respect to condition (1). Let P be one of them. Because W and $\mathcal{U}$ are finite, so is $W[\pi^{-1}]$. Furthermore, $(W[\pi^{-1}] \parallel P)[\pi]$ is finite because π and P are finite. Because failure-refinement is decidable for finite processes [Roscoe 1994], we can check if P satisfies condition (2), *i.e.* if $(W[\pi^{-1}] \parallel P)[\pi] =_\mathsf{F} W$. If P satisfies condition (2), then P is an enforcement process for ϕ on W. If none of the finitely many candidate processes P satisfies condition (2), then there exists no enforcement process for ϕ on W. □

The runtime complexity of solving **EPE** as sketched above is as follows. For an SoD constraint s, consider the SoD process A_s. The number of states of a transition system that corresponds to A_s is in $O(2^{|\mathcal{U}|})$ because A_s is parametrized by two subsets of $\mathcal{U}$ and there is a state for every possible subset. The number of

states of a transition system corresponding to A_b, for a BoD constraint b, is linear in the size of $\mathcal{U}$. The number of states of a transition system corresponding to A_{UT}, for an user-task assignment UT, is constant. Let $\phi = (UT, S, B)$. By Definition 8.5 and CSP's operational semantics for the parallel, synchronized composition of two processes (see Definition A.1 in Appendix A.5), it follows that the number of states of a transition system corresponding to A_ϕ is in $O(|\mathcal{U}|^{|B|}2^{|S||\mathcal{U}|})$. The set of input symbols of a transition system corresponding to A_ϕ is $(\mathcal{X} \cup \mathcal{O})^{\checkmark}$. Therefore, the number of transitions is in $O((|\mathcal{O}| + |\mathcal{T}||\mathcal{U}|)|\mathcal{U}|^{2|B|}2^{2|S||\mathcal{U}|})$.

The above described decision procedure checks for each transition system that has a subset of A_ϕ's transitions whether it satisfies condition (2) of Definition 9.2. This requires deciding failure equivalence which is **PSPACE**-complete [Roscoe 1994]. Thus, this approach has a runtime complexity that is double exponential in the number of users and constraints. Hence, it is not practical for workflows with large sets of users. We therefore propose approximation algorithms for **EPE** in the following section.

9.4 Approximations

We now present an approximation algorithm EPEA for **EPE**. EPEA is an approximation in that it may return No even when an enforcement process for the given input exists. However, EPEA makes no approximation error in the opposite case: if there does not exist an enforcement process for ϕ on W, then EPEA's output is always No. EPEA has an exponential runtime complexity. We show in a second step how to change EPEA to approximate **EPE** in polynomial time using bounds from graph-coloring.

9.4.1 Exponential Approximation

EPEA, defined in Algorithm 2, takes an instance of **EPE** as input and returns No or a relation that can be transformed to an enforcement process for the given **EPE** instance. In detail, EPEA is defined as the composition of CGraph and LCol. CGraph, defined in Algorithm 3, transforms the tasks of a workflow specification process W, *i.e.* $T = \{t \in \mathcal{T} \mid \exists i \in \mathsf{T}(W), t \in i\}$, and an authorization policy $\phi = (UT, S, B)$ to an instance of the **ListColoring** problem. CGraph returns either V, E, and L, where (V, E) is a graph and $\mathsf{L} : V \to 2^{\mathcal{U}}$ is a color-list function for (V, E), or No. The vertices in V are sets of tasks of W. Every task of W is contained in one vertex. The BoD constraints B define which sets of tasks form vertices, UT defines L, and the edges correspond to the SoD constraints in S.

Algorithm 2: EPEA(T,ϕ)

Input: T and $\phi = (UT,S,B)$
Output: returns a relation $R \subseteq \pi^{-1}$ or No

```
1 if T = ∅ then
2     return ∅
3 else
4     col_L ← LCol(CGraph(T, φ))
5     if CGraph or LCol return No then
6         return No
7     else
8         return {(t, t.u) | (T, u) ∈ col_L, t ∈ T}
```

CGraph returns No if W contains two tasks t_1 and t_2 whose execution is constrained by an SoD constraint in S and if there is a subset of BoD constraints in B that bind the duties between t_1 and t_2.

Example 9.4 (Graph Returned by CGraph) Figure 9.3 depicts the graph and the color-list function L returned by CGraph for the tasks of the collateral evaluation workflow and our example authorization policy ϕ. ★

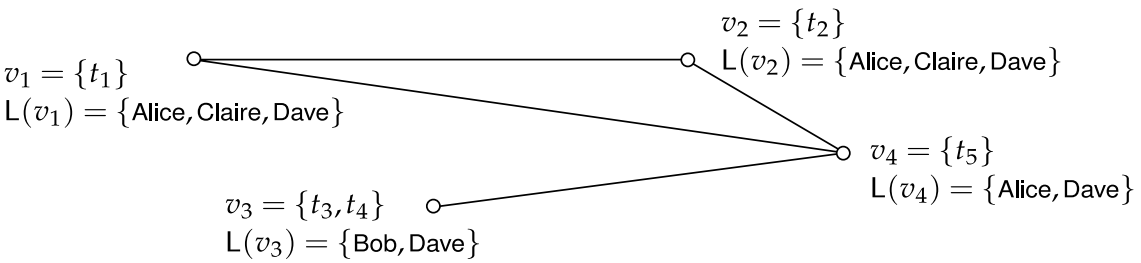


Figure 9.3. Constraint graph of the collateral evaluation workflow

EPEA solves the **ListColoring** instance returned by CGraph using LCol, introduced in Section 2.3. Finally, it transforms the coloring returned by LCol to a relation between tasks and execution events and returns this relation. If CGraph fails to build a graph or LCol does not find a coloring, then EPEA returns No. The following lemma, which we prove in Appendix A.6, shows how a relation returned by EPEA defines an enforcement process.

Lemma 9.2 *Let W be a workflow specification process, $T = \{t \in \mathcal{T} \mid \exists i \in \mathsf{T}(W), t \in i\}$, and ϕ an authorization policy. If* EPEA(T,ϕ) *returns a relation R, then $W[R]$ is an enforcement process for ϕ on W.*

Algorithm 3: CGRAPH(T, ϕ)

Input: T and $\phi = (UT, S, B)$
Output: returns a graph (V, E) and a color-list function $\mathsf{L} : V \to 2^{\mathcal{U}}$ or No

$V, E, \mathsf{L} \leftarrow \varnothing$
foreach $t \in T$ **do**
  $V \leftarrow V \cup \{\{t\}\}$
  $\mathsf{L} \leftarrow \mathsf{L} \cup \{(\{t\}, \{u \mid (u,t) \in UT\})\}$
foreach $(T_1, O) \in B$ **do**
  pick a $t_1 \in T_1$
  let $v_1 \in V$ **s.t.** $t_1 \in v_1$
  foreach $t_2 \in T_1 \setminus \{t\}$ **do**
    let $v_2 \in V$ **s.t.** $t_2 \in v_2$
    $V \leftarrow (V \setminus \{v_1, v_2\}) \cup \{v_1 \cup v_2\}$
    $\mathsf{L} \leftarrow (\mathsf{L} \setminus \{(v_1, \mathsf{L}(v_1)), (v_2, \mathsf{L}(v_2))\}) \cup \{(v_1 \cup v_2, \mathsf{L}(v_1) \cap \mathsf{L}(v_2))\}$
foreach $(T_1, T_2, O) \in S$ **do**
  foreach $t_1 \in T_1$ **do**
    let $v_1 \in V$ **s.t.** $t_1 \in v_1$
    foreach $t_2 \in T_2$ **do**
      let $v_2 \in V$ **s.t.** $t_2 \in v_2$
      if $v_1 \neq v_2$ **then**
        $E \leftarrow E \cup \{\{v_1, v_2\}\}$
      else
        return No
return V, E, L

9.4.2 Polynomial Approximation

We now approximate **EPE** in polynomial time using graph-coloring bounds.

Corollary 9.1 *For a workflow specification process W and an authorization policy ϕ, let $T = \{t \in \mathcal{T} \mid \exists i \in \mathsf{T}(W), t \in i\}$ and $(V, E, \mathsf{L}) = \text{CGRAPH}(T, \phi)$. If*

$$\max_{v \in V} |\{v' \mid \{v, v'\} \in E\}| < \min_{v \in V} |\mathsf{L}(v)|$$

then there exists an enforcement process for ϕ on W.

Proof. Let W be a workflow specification process, ϕ an authorization policy, $T = \{t \in \mathcal{T} \mid \exists i \in \mathsf{T}(W), t \in i\}$, and $(V, E, \mathsf{L}) = \text{CGRAPH}(T, \phi)$. Then $\max_{v \in V} |\{v' \mid$

$\{v,v'\} \in E\}|$ is the maximal degree $\Delta(V,E)$ of (V,E). Furthermore, let $k = \min_{v \in V} |\mathsf{L}(v)|$, *i.e.* L is a k-color-list function for (V,E). Assume that $\Delta(V,E) < k$. By Lemma 2.1 it follows that $\chi_l(V,E) \leq k$. Therefore, there exists an L-coloring for (V,E). Hence, EPEA(T,ϕ) returns a relation R and, by Lemma 9.2, $W[R]$ is an enforcement process for ϕ on W. ■

Informally, Corollary 9.1 tells us the following. If the maximal number of SoD constraints under which a task is constrained is less than the minimal number of users who are authorized to execute a task with respect to the user-task assignment, then there exists an enforcement process. Simplified, there exists an enforcement process if the set of users is large and their workflow-independent authorizations are evenly distributed.

Assume a workflow specification process W and an authorization policy ϕ. The algorithm CGraph computes (V,E,L) in polynomial time or returns No. We can then check if the condition of Corollary 9.1 holds for V, E, and L. If it holds, we only know that an enforcement process for ϕ on W exists but $E_{\phi,W}$ is not constructed yet. However, by Lemma 2.1 and because the condition of Corollary 9.1 holds, a greedy algorithm with polynomial runtime complexity finds an L-coloring for (V,E). We can therefore replace the call to LCol in EPEA by a call to the greedy algorithm. It follows that we can approximate **EPE** in polynomial time.

9.4.3 EPEA Implementation

As presented in Section 8.3, our extension of Oryx's presentation tier enables us to graphically model workflows including SoD and BoD constraints. Additionally, we extended Oryx by a window for specifying user-task assignments. We now describe how we analyze **EPE** instances, which are specified using these extensions in Oryx's presentation tier, with our EPEA implementation in Oryx's application tier.

For performance reasons, we do not use LCol to solve **ListColoring**-instances. Instead, we transform them into Boolean formulae with a variable for each vertex-color combination and clauses encoding the coloring constraints imposed by edges and the requirement that a color must be chosen for every vertex. Such a reduction is standard and we therefore omit a correctness proof. We then use the SAT-solver *sat4j* [Berre and Parrain 2010] to compute satisfying assignments for these formulae. Transforming assignments back into colorings for the initial **ListColoring** instances is straightforward and if a formula is unsatisfiable then no coloring exists.

We decompose the total running time required for solving an **EPE** instance into three parts. The *communication time* is the time required to send the **EPE** instance from Oryx's presentation tier to its application tier and finally to return the result back to the presentation tier. The *transformation time* is the time it takes to transform the **EPE** instance to a Boolean formula and the *solving time* is the time it takes to compute a satisfying assignment for the formula with sat4j.

The communication time depends on various factors such as the network throughput, latency, and the payload size. We run the presentation tier and the application tier on two different computers, which are connected by a standard enterprise network with an average latency of 1 millisecond (ms). Computing an enforcement process for our running example has on the average a communication time of 100 ms, a transformation time of 80 ms, and a solving time of 15 ms, summing up to a total running time of 200 ms. We also tested our implementation with random user-task assignments with up to 50 users and could observe a minor increase in the solving time while the communication and transformation time remained relatively stable. However, with these numbers of users, the communication time still overshadows the solving time.

Chapter 10

Optimal Workflow-Aware Authorizations

In this chapter, we drop the assumption that history-independent authorizations are non-administrable and we model the price of changing from one authorization policy to another one by a cost-function. This function may account for the cost of changing to the new policy, of maintaining it, and the risk associated with it. We consider minimizing risk equivalent to maximizing protection. Based on the **EPE** approximation introduced in Section 9.4, we then investigate the complexity of computing a cost-minimizing authorization policy that can be enforced on a given workflow without obstructions. The resulting authorization policy is *optimal* in the sense that it empowers users to execute job-relevant tasks while maximizing the protection of the workflow's underlying resources.

We first define a generic version of the optimization problem, which models history-independent authorizations as user-task assignments. Afterward, we refine user-task assignments to RBAC policies whose additional structure enables us to map the optimization problem to the Integer Linear Programming (**ILP**) problem. We use the following running example to illustrate our results and to measure the performance of our mapping's implementation.

Example 10.1 (Payment Workflow) Figure 10.1 shows a BPMN model of a payment workflow, which is based on [EGEI 2009] and defines the tasks that a customer (organization) executes to process an invoice received by a supplier. Upon receipt of an invoice, a user checks whether the invoice is correct (t_1). In parallel, a user checks whether the goods corresponding to the invoice have arrived (t_2). If they have not arrived yet and their arrival is not overdue, the user waits for three days and checks again. Otherwise, the workflow proceeds. If inconsistencies have occurred, *i.e.* if the invoice is incorrect or the arrival is overdue, a user sends a dispute case (t_3) to the supplier and the workflow terminates. If no inconsistencies have occurred, a user prepares the payment (t_4). Afterward, the payment is either approved (t_5), executed (t_6) and the workflow terminates, or the payment is not approved (t_5) and the workflow loops back to the start.

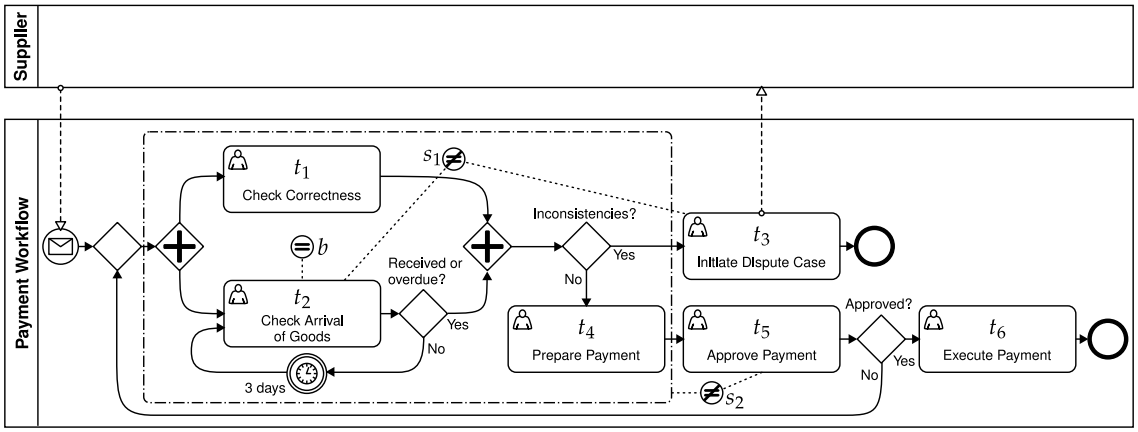


Figure 10.1. Payment workflow modeled in BPMN

We model the payment workflow by the workflow specification process W:

$$\begin{aligned} W &= ((t_1 \to \mathit{SKIP}) \parallel W_1)\,;\,((t_3 \to \mathit{SKIP}) \sqcap W_2) \\ W_1 &= t_2 \to (\mathit{SKIP} \sqcap W_1) \\ W_2 &= t_4 \to t_5 \to (W_1 \sqcap (t_6 \to \mathit{SKIP})) \end{aligned}$$

Since we build in this chapter on the **EPE** approximation presented in Section 9.4, we effectively only use the set of tasks $T = \{t_1, \ldots, t_6\}$, which W engages in, and abstract away from W's control-flow.

Figure 10.1 also shows that the payment workflow is annotated by three history-dependent, workflow-specific authorization constraints. The SoD constraint $s_1 = (t_2, t_3, \varnothing)$ ensures that a user cannot embezzle the received goods and later initiate a dispute case. Similarly, $s_2 = (\{t_1, t_2, t_4\}, t_5, \varnothing)$ ensures that any user who approves a payment did not execute one of the preceding task instances. Therefore, the approval of a fraudulent payment requires the collusion of at least two users. The BoD constraint $b = (t_2, \varnothing)$ requires that only one user checks whether the goods have arrived. This facilitates the reuse of knowledge and thereby increases efficiency if multiple checks are required.

Furthermore, Figure 10.2 shows the payment workflow's initial RBAC policy (UR, RT). We refer to the role Procurement Clerk as r_1, Warehouse Clerk as r_2, Procurement Manager as r_3, and Accountant as r_4. The set of roles is thus $\mathcal{R} = \{r_1, r_2, r_3, r_4\}$ and $UT = RT \circ UR$ is the user-task assignment corresponding to (UR, RT). In summary, we get the authorization policy $\phi = (UT, \{s_1, s_2\}, \{b\})$. ★

We use a user-task assignment UT to specify workflow-independent and history-independent authorizations. Moreover, its domain $\mathsf{dom}(UT)$ also represents the set of *available* users and, conversely, $\mathcal{U} \setminus \mathsf{dom}(UT)$ is the set of *unavailable* users, *e.g.* those users who are not ready to work or not part of the

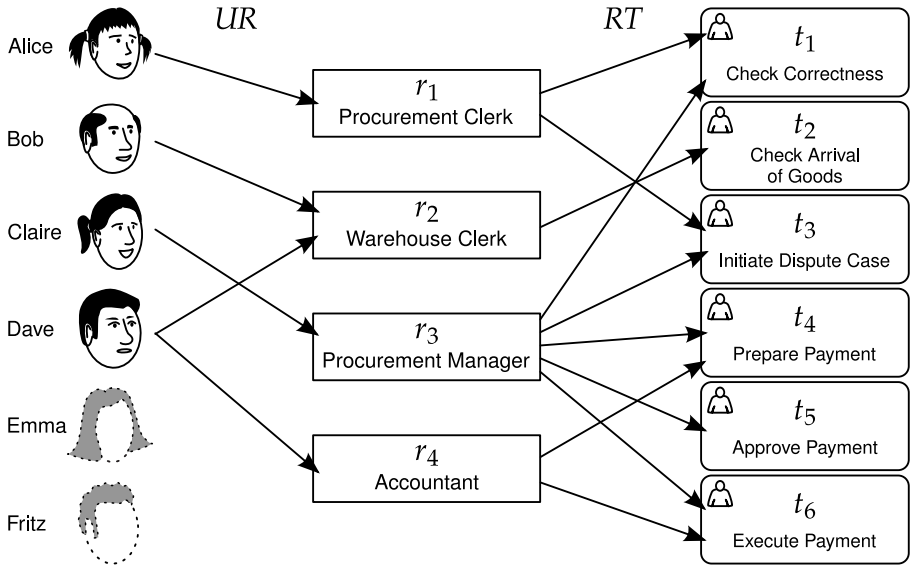


Figure 10.2. Initial RBAC policy for the payment workflow

organization. For our running example, the set of available users is $\mathsf{dom}(UT) = \{\mathsf{Alice}, \mathsf{Bob}, \mathsf{Claire}, \mathsf{Dave}\}$, whereas Emma and Fritz are unavailable (see also Figure 10.2).

10.1 Allocation

In the previous chapter, we approximated **EPE** by first transforming an **EPE** instance to a **ListColoring** instance, then computing a coloring for this **ListColoring** instance, and finally we derived a mapping from users to tasks from this coloring. We now formalize the properties of this mapping as a function, which we call an *allocation*.

Definition 10.1 (Allocation) *Let W be a workflow specification process, $\phi = (UT, S, B)$ an authorization policy for W, and $T = \{t \in \mathcal{T} \mid \exists i \in \mathsf{T}(W) \,.\, t \in i\}$. We call a function* $\mathsf{alloc} : T \to \mathcal{U}$ *an* allocation *for W and ϕ, written* $\mathsf{alloc} \models (W, \phi)$*, if*

(1) for all $t \in T$, $(\mathsf{alloc}(t), t) \in UT$,

(2) for all $(T_1, T_2, O) \in S$, $t_1 \in T_1$, and $t_2 \in T_2$, $\mathsf{alloc}(t_1) \neq \mathsf{alloc}(t_2)$, and

(3) for all $(T', O) \in B$, and $t_1, t_2 \in T'$, $\mathsf{alloc}(t_1) = \mathsf{alloc}(t_2)$.

Given a workflow specification process W and an authorization policy ϕ for W, we write $\models^{\exists} (W, \phi)$ if there exists an allocation for W and T. The following lemma, which we prove in Appendix A.7, formalizes the relation between EPEA's output and the existence of an allocation.

Lemma 10.1 *Let W be a workflow specification process, $\phi = (UT, S, B)$ an authorization policy for W, and $T = \{t \in \mathcal{T} \mid \exists i \in \mathsf{T}(W) \,.\, t \in i\}$.*

(1) If $\mathrm{EPEA}(T, \phi)$ *returns a relation R, then $\{(t,u) \mid (t,t.u) \in R\} \models (W, \phi)$.*

(2) If there exists an allocation alloc *for W and ϕ, then* $\textsc{CGraph}(T, \phi)$ *returns a graph (V, E) and a color-list function* L *and $\{(v,u) \mid v \in V, \exists t \in v \,.\, \mathsf{alloc}(t) = u\}$ is an* L*-coloring for (V, E).*

The following corollary, which follows directly from Algorithm 2, Lemma 9.2, and Lemma 10.1, shows that an allocation for a workflow specification process W and an authorization policy ϕ defines an enforcement process for ϕ on W.

Corollary 10.1 *Let W be a workflow specification process, ϕ an authorization policy, and* alloc *and allocation for W and ϕ. $W[\{(t,t.u) \mid (t,u) \in \mathsf{alloc}\}]$ is an enforcement process for ϕ on W.*

In the remainder of this chapter, we focus on computing optimal authorization policies for which an allocation exists. By Corollary 10.1 such an optimal authorization policy allows an obstruction-free enforcement. We now proceed with an example of an allocation for the payment workflow and then cast the existence of an allocation as a decision problem.

Example 10.2 (Allocation) Consider again the payment workflow. In particular, let W be the workflow specification process and ϕ the authorization policy introduced in Example 10.2. The function $\mathsf{alloc} = \{(t_1, \mathsf{Alice}), (t_2, \mathsf{Bob}), (t_3, \mathsf{Alice}), (t_4, \mathsf{Dave}), (t_5, \mathsf{Claire}), (t_6, \mathsf{Dave})\}$ is an allocation for W and T. ★

Definition 10.2 (The Allocation Existence Problem (**AEP**))

Input: *A workflow specification process W and an authorization policy ϕ.*

Output: YES *if* $\not\models^{\exists} (W, \phi)$ *or* NO *otherwise.*

The following lemma, which we prove in Appendix A.8, states that **AEP** is **NP**-complete. The **NP**-hardness reduction is analogous to the proof of Lemma 9.1 and completeness follows immediately from the fact that an allocation can be verified in polynomial time. Lemma 10.2 gives us a lower bound on the complexity of forthcoming problems that build on **AEP**.

Lemma 10.2 **AEP** *is* **NP***-complete.*

We do not provide an algorithm for **AEP** here. Instead, we show in Section 10.3 how to use algorithms for problems that build on **AEP** to solve instances of **AEP**.

10.2 The General Problem

The concept of an allocation gives us a notion of empowerment that is required for achieving a business objective. We now investigate the counterpart of empowerment, namely protection, and in particular the question of how to maximize protection without sacrificing the achievement of business objectives. Consider the following motivational example.

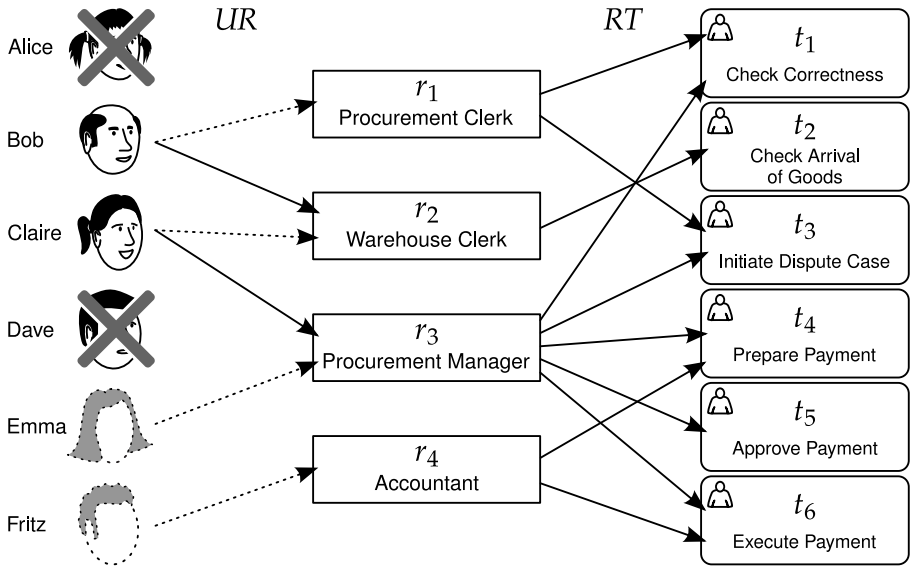


Figure 10.3. Changed RBAC policy

Example 10.3 (No Allocation For Changed RBAC Policy) Let UR_0 be the user-role assignment UR illustrated in Figure 10.2 and let RT be the corresponding role-task assignment. Furthermore, let $\phi_0 = (RT \circ UR_0, \{s_1, s_2\}, \{b\})$. We concluded in Example 10.2 that there exists an allocation for the payment workflow and ϕ_0. Suppose now that Alice and Dave become unavailable, say they went on holiday. The new RBAC policy (UR, RT) is illustrated in Figure 10.3, ignoring the dotted arrows for the moment. As a result, we get the new user-task assignment $UT = RT \circ UR$ and the new authorization policy is $\phi = (UT, \{s_1, s_2\}, \{b\})$.

The only user who may execute t_1 with respect to ϕ is Claire and t_2 may only be executed by Bob. Only Bob and Claire are available with respect to UT but it follows by s_2 that neither Bob nor Claire can be allocated to t_5. Hence, there exists no allocation for W and ϕ. ★

To overcome the situation illustrated in this example, we must change UT by assigning more roles to available users or making previously unavailable users available. However, this change should incur minimal cost. In the following,

we introduce a cost function that models the administrative cost of changing UT to UT' and the associated risks. We use this function to evaluate potential new user-task assignments and to find the optimal assignment UT'.

Definition 10.3 (Cost Function) *For a totally ordered set C, a* cost function *is a partial function* $\mathsf{cost} : 2^{\mathcal{U}\times\mathcal{T}} \times 2^{\mathcal{U}\times\mathcal{T}} \to C$.

We use a cost function for two purposes. For two user-task assignments UT and UT'

1. $\mathsf{cost}(UT, UT')$ is the cost of changing UT to UT' and

2. $\mathsf{dom}(\mathsf{cost})$ defines the feasible changes, *i.e.* it is possible to change from UT to UT' if $(UT, UT') \in \mathsf{dom}(\mathsf{cost})$.

In this general setting, the cost of changing from a user-task assignment UT to a user-task assignment UT' can have many meanings and cost may satisfy different properties accordingly. We give a few examples of potential costs that may be modeled using cost. A concrete example for a role-based cost function follows in the next section.

Risk. By empowering users to execute tasks, a user-task assignment exposes the underlying resources to risks, such as fraud, errors, and data leakage. There exist various methodologies for performing a risk analysis [Molloy *et al.* 2008; Basin *et al.* 2011b]. We consider them outside the scope of this dissertation and simply point out that the expected value computed by a quantitative risk analysis corresponds to a cost [Molloy *et al.* 2008]. If the cost function encodes only risks, the value of $\mathsf{cost}(UT, UT')$ is independent of UT. Additionally, if the risk quantifies only the misuse of authorizations, it is reasonable to assume that $\mathsf{cost}(UT, \varnothing) \leq \mathsf{cost}(UT, UT')$ for all user-task assignments UT and UT'. In other words, empowering no user to execute a task entails the least risk, *i.e.* maximizes protection.

Administrative Cost. The activities associated with changing an authorization policy are typically not for free. For example, recruiting a new employee, assigning her initial authorizations, and training her to use them appropriately may be costly [Casa Mont *et al.* 2010]. Consequently, if cost encodes only administrative costs it is reasonable to assume that $\mathsf{cost}(UT, UT) \leq \mathsf{cost}(UT, UT')$ for all user-task assignments UT and UT'. In other words, it costs the least to make no changes at all.

Maintenance Cost. Maintaining an authorization policy may involve costs such as salaries and license fees required for task executions. Abstractly, a cost function only encoding maintenance costs behaves the same way as a cost function only encoding risk: it is cheapest to maintain an empty user-task assignment.

Using the existence of an allocation as the empowerment condition and a cost function as measure of protection, we now reduce the question of how to balance empowerment and protection to an optimization problem.

Definition 10.4 (The Optimal Workflow-Aware Authorization (**OWA**) Problem)

Input: *A cost function* cost*, a workflow specification process* W*, and an authorization policy* (UT, S, B)*.*

Output: $\min_{UT'}\{\mathsf{cost}(UT, UT') \mid \models^{\exists} (W, (UT', S, B)) \text{ and } (UT, UT') \in \mathsf{dom}(\mathsf{cost})\}$
or No *if the above set is empty.*

The optimal workflow-aware authorization (**OWA**) problem asks for a user-task assignment that allows an obstruction-free enforcement on the given workflow and incurs minimal cost.

Without any assumptions about the structure of the cost function, it is impossible to make statements about **OWA**'s runtime or space complexity. The refined cost function that we propose in the following section allows us to determine these complexities.

10.3 A Role-Based Cost Function

To demonstrate the applicability of **OWA** to a realistic example, we refine **OWA** by decomposing user-task assignments into RBAC policies and let the cost function be role-based. For simplicity, we let the totally ordered set C be $\mathbb{R}$. Specifically, we define the cost function in terms of the following auxiliary functions. For a role $r \in \mathcal{R}$:

- $\mathsf{risk}(r) \in \mathbb{R}$ models the risk associated with the assignment of a user to r,
- $\mathsf{add}(r) \in \mathbb{R}$ models the administrative cost of assigning a user to r,
- $\mathsf{rm}(r) \in \mathbb{R}$ models the administrative cost of removing a user's assignment from r, and
- $\mathsf{ma}(r) \in \mathbb{R}$ models the maintenance cost of having a user assigned to r.

Using these functions, we define the cost of changing a user-role assignment.

Definition 10.5 (Role Cost Function) *Given the auxiliary functions* $\mathsf{risk}, \mathsf{add}, \mathsf{rm}, \mathsf{ma} : \mathcal{R} \to \mathbb{R}$, *a* role cost function *is a function* $\mathsf{costR} : 2^{\mathcal{U}\times\mathcal{R}} \times 2^{\mathcal{U}\times\mathcal{R}} \to \mathbb{R}$, *such that for two user-role assignments* UR *and* UR',

$$\begin{aligned}\mathsf{costR}(UR, UR') = \; & \textstyle\sum_{(u,r)\in UR'}(\mathsf{risk}(r) + \mathsf{ma}(r)) + \\ & \textstyle\sum_{(u,r)\in UR'\setminus UR} \mathsf{add}(r) + \sum_{(u,r)\in UR\setminus UR'} \mathsf{rm}(r)\end{aligned}$$

A role cost function defines the cost of changing from UR to UR' simply as the sum of all the risk and maintenance costs associated with UR' and the administrative cost of adding and removing assignments when changing from UR to UR'. We assume that the auxiliary functions risk, add, rm, and ma are total and hence costR is total too. Instead of using costR's domain to determine feasible user-role assignment changes, we define a *maximal user-role assignment* $UR^{\max} \subseteq \mathcal{U} \times \mathcal{R}$ and assume that every user-role assignment $UR \subseteq UR^{\max}$ is feasible.

Example 10.4 (Role Cost Function) Table 10.1 lists the risk, maintenance, and administrative costs associated with the four roles of the payment workflow. We adopt the elementary approach that roles assigned to a large number of tasks represent more responsibility and are therefore more costly [Han *et al.* 2009]. Let costR be the corresponding role cost function.

	risk	ma	add	rm
Procurement Clerk (r_1)	5	3	2	1
Warehouse Clerk (r_2)	3	3	2	1
Procurement Manager (r_3)	12	5	3	2
Accountant (r_4)	7	4	2	1

Table 10.1. Decomposition of the role cost function

Recall the RBAC policy (UR, RT) shown in Figure 10.3 and let the solid and dotted arrows between users and roles be the maximal user-role assignment $UR^{\max}$ for the payment workflow. For example, Emma is unavailable with respect respect to UT. Because $(\mathsf{Emma}, r_3) \in UR^{\max}$, we may change Emma's availability by assigning her to r_3, resulting in the user-role assignment $UR' = UR \cup \{(\mathsf{Emma}, r_3)\}$. The administrative activity of assigning Emma to r_3 costs 3 and the overall risk and maintenance cost rises by $12 + 5$. Thus, $\mathsf{costR}(UR, UR') - \mathsf{costR}(UR, UR) = 3 + 12 + 5 = 20$. ★

Using costR and $UR^{\max}$, we now refine **OWA** to **ROWA**.

Definition 10.6 (The Role-Based Optimal Workflow-Aware Authorization (**ROWA**) Problem)

Input: *A role cost function* costR, *a maximal user-role assignment* UR^{max}, *a workflow specification process* W, *an RBAC policy* (UR, RT), *a set of SoD constraints* S, *and a set of BoD constraints* B.

Output: $\min_{UR' \subseteq UR^{max}} \{\mathsf{costR}(UR, UR') \mid \models^{\exists}(W, (RT \circ UR', S, B))\}$
or No *if the above set is empty.*

We refer to the output corresponding to the **ROWA** instance *rowa* as **ROWA**(*rowa*). In the following, we define a function ROWAtoILP that transforms a **ROWA** instance to an **ILP** instance. We specify the matrix **A** and the vectors **b**, **c**, and **x** indirectly by defining the respective (**ILP**) constraints and the cost function in terms of sums. Furthermore, we index decision variables with a superscript, which should not be mistaken for an exponent. We thereby simplify the forthcoming proofs. Transforming the constraints and variables to a matrix-vector form is straightforward and therefore not shown in detail.

Definition 10.7 (ROWAtoILP) *Let* $(\mathsf{costR}, UR^{max}, W, (UR, RT), S, B)$ *be a* **ROWA** *instance, let* costR *be composed of the auxiliary functions* risk, add, rm, *and* ma, *let* $T = \{t \in \mathcal{T} \mid \exists i \in \mathsf{T}(W) \,.\, t \in i\}$, *and let* $U = \mathsf{dom}(UR^{max})$ *and* $R = \mathsf{ran}(UR^{max})$. *The function* ROWAtoILP *transforms* $(\mathsf{costR}, UR^{max}, W, (UR, RT), S, B)$ *to an* **ILP** *instance as follows:*

Decision variables: $x^{u,r}, x^{u,t} \in \mathbb{Z}$ *for every* $u \in U, r \in R$, *and* $t \in T$

Objective function:

$$\sum_{(u,r) \in U \times R} x^{u,r}(\mathsf{risk}(r) + \mathsf{ma}(r)) + \sum_{(u,r) \in (U \times R) \setminus UR} x^{u,r}\mathsf{add}(r) + \sum_{(u,r) \in UR} (1 - x^{u,r})\mathsf{rm}(r)$$

Constraints:

$$(1) \quad \forall\, t \in T,\, u \in U \,.\, \textstyle\sum_{\{r \mid (r,t) \in RT\}} x^{u,r} \geq x^{u,t}$$

$$(2) \quad \forall\, t \in T. \textstyle\sum_{u \in U} x^{u,t} = 1$$

$$(3) \quad \forall\, (T_1, T_2, O) \in S, t_1 \in T_1, t_2 \in T_2, u \in U \,.\, x^{u,t_1} + x^{u,t_2} \leq 1$$

$$(4) \quad \forall\, (T', O) \in B, t_1, t_2 \in T', u \in U \,.\, x^{u,t_1} = x^{u,t_2}$$

$$(5) \quad \textstyle\sum_{(u,r) \in (U \times R) \setminus UR^{max}} x^{u,r} = 0$$

$$(6) \quad \forall\, u \in U,\, r \in R \,.\, x^{u,r} \geq 0 \text{ and } x^{u,r} \leq 1$$

$$(7) \quad \forall\, u \in U,\, t \in T \,.\, x^{u,t} \geq 0 \text{ and } x^{u,t} \leq 1$$

Consider a **ROWA** instance composed of costR, $UR^{\max}$, W, (UR, RT), S, and B, and let $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ be the corresponding **ILP** instance returned by ROWAtoILP. We refer to a constraint or a set of constraints i in Definition 10.7 as Ci. Next, we define a relation between feasible solutions of **ILP** instances generated by ROWAtoILP, and user-role assignments and allocations for their corresponding **ROWA** instances. Afterward, we use this relation to explain the constraints C1–C7 and to prove the soundness and completeness of ROWAtoILP.

Note that a feasible solution $\mathbf{x}$ for $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ is composed of the decision variables $x^{u,r}$ and $x^{u,t}$, where u ranges over $\mathsf{dom}(UR^{\max})$, r over $\mathsf{ran}(UR^{\max})$, and t over T. Because $\mathbf{x}$ is a feasible solution, the decision variables satisfy all constraints listed in Definition 10.7, in particular C6 and C7. Therefore, the decision variables assume either the value 0 or 1.

Definition 10.8 (Correspondence Relation) *Let* $(\mathsf{costR}, UR^{max}, W, (UR, RT), S, B)$ *be a* **ROWA** *instance and* $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ *the corresponding* **ILP***-instance returned by* ROWAtoILP. *Furthermore, let* $\mathbf{x}$ *be a feasible solution for* $(\mathbf{A}, \mathbf{b}, \mathbf{c})$, $T = \{t \in \mathcal{T} \mid \exists i \in \mathsf{T}(W) \,.\, t \in i\}$, $U = \mathsf{dom}(UR^{max})$, *and* $R = \mathsf{ran}(UR^{max})$. *For a user-role assignment* UR' *and an allocation* alloc, *we say that* $\mathbf{x}$ corresponds *to* (UR', alloc), *written* $\mathbf{x} \sim (UR', \mathsf{alloc})$, *if*

(1) $UR' = \{(u, r) \in U \times R \mid x^{u,r} = 1\}$ *and*

(2) $\mathsf{alloc} = \{(t, u) \in T \times U \mid x^{u,t} = 1\}$.

In other words, the decision variables of the form $x^{u,r}$ determine UR' and those of the form $x^{u,t}$ determine alloc. More specifically, if $x^{u,r} = 1$, for a user u and a role r, then u is assigned to r in UR'. Moreover, for a user u and a task t, $x^{u,t} = 1$ implies that alloc maps t to u. Note that the correspondence relation $\sim$ uniquely determines a tuple (UR', alloc) given a vector $\mathbf{x}$ and *vice versa*.

We now informally describe the (**ILP**) constraints created by ROWAtoILP. We expand upon this in the proof of Lemma 10.3. C1 ensures that an allocation assigns a user u only to a task t if u is assigned to a role r that is assigned to t. C2 enforces that an allocation maps every task to exactly one user. C3 and C4 enforce that an allocation satisfies the given SoD and BoD constraints, respectively. Finally, C5 restricts user-role assignments to subsets of the given maximal user-role assignment. C6 and C7 were already explained above.

The following lemma, which we prove in Appendix A.9, establishes that ROWAtoILP is both sound and complete.

Lemma 10.3 *Let* $(\mathsf{costR}, UR^{max}, W, (UR, RT), S, B)$ *be a* **ROWA** *instance and* $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ *the corresponding* **ILP***-instance returned by* ROWAtoILP. *Let* $\mathbf{x}$ *be a vector,* UR' *a user-role assignment, and* alloc *an allocation, such that* $\mathbf{x} \sim (UR', \mathsf{alloc})$.

- Soundness: *If* $\mathbf{x}$ *is a feasible solution for* $(\mathbf{A},\mathbf{b},\mathbf{c})$ *then* $UR' \subseteq UR^{max}$ *and* $\mathsf{alloc} \models (W,(RT \circ UR',S,B))$.
- Completeness: *If* $UR' \subseteq UR^{max}$ *and* $\mathsf{alloc} \models (W,(RT \circ UR',S,B))$ *then* $\mathbf{x}$ *is a feasible solution for* $(\mathbf{A},\mathbf{b},\mathbf{c})$.

Given the soundness and completeness of ROWAtoILP, we now show with Theorem 10.1 that ROWAtoILP and algorithms for **ILP** can be employed to solve **ROWA** instances.

Theorem 10.1 *For every* **ROWA** *instance rowa,*

$$\mathbf{ROWA}(rowa) = \mathbf{ILP}(\mathsf{ROWAtoILP}(rowa)) .$$

Proof. Let $(\mathsf{costR},UR^{\max},W,(UR,RT),S,B)$ be a **ROWA** instance and $(\mathbf{A},\mathbf{b},\mathbf{c})$ be the corresponding **ILP**-instance returned by ROWAtoILP. Let $U = \mathsf{dom}(UR^{\max})$, $R = \mathsf{ran}(UR^{\max})$, and let costR be defined by the auxiliary functions risk, add, rm, and ma. Furthermore, let UR' be a user-role assignment, alloc an allocation, and $\mathbf{x}$ a vector such that $UR' \subseteq UR^{\max}$, $\mathsf{alloc} \models (W,(RT \circ UR'))$ and $\mathbf{x} \sim (UR',\mathsf{alloc})$. From Lemma 10.3 we have that $\mathbf{x}$ is a feasible solution for $(\mathbf{A},\mathbf{b},\mathbf{c})$.

It follows from Definitions 10.5, 10.7, and 10.8 that

$$\begin{aligned}
\mathsf{costR}(UR,UR') &= \textstyle\sum_{(u,r)\in UR'}(\mathsf{risk}(r) + \mathsf{ma}(r)) \\
&\quad + \textstyle\sum_{(u,r)\in UR'\setminus UR} \mathsf{add}(r) + \sum_{(u,r)\in UR\setminus UR'} \mathsf{rm}(r) \\
&= \textstyle\sum_{(u,r)\in UR'} 1\,(\mathsf{risk}(r) + \mathsf{ma}(r)) \\
&\quad + \textstyle\sum_{(u,r)\in (U\times R)\setminus UR'} 0\,(\mathsf{risk}(r) + \mathsf{ma}(r)) \\
&\quad + \textstyle\sum_{(u,r)\in UR'\setminus UR} 1\,\mathsf{add}(r) + \sum_{(u,r)\in ((U\times R)\setminus UR')\setminus UR} 0\,\mathsf{add}(r) \\
&\quad + \textstyle\sum_{(u,r)\in UR\setminus UR'} 1\,\mathsf{rm}(r) + \sum_{(u,r)\in UR\cap UR'} 0\,\mathsf{rm}(r) \\
&= \textstyle\sum_{(u,r)\in U\times R} x^{u,r}(\mathsf{risk}(r) + \mathsf{ma}(r)) \\
&\quad + \textstyle\sum_{(u,r)\in (U\times R)\setminus UR} x^{u,r}\mathsf{add}(r) + \sum_{(u,r)\in UR}(1 - x^{u,r})\mathsf{rm}(r) \\
&= \mathbf{c}^{\mathrm{T}}\mathbf{x} .
\end{aligned}$$

Assume that UR' minimizes costR with respect to all user-role assignments $UR'' \subseteq UR^{\max}$ such that $\not\models^{\exists} (W,(RT \circ UR'',S,B))$, *i.e.* $\mathsf{costR}(UR,UR') = \mathbf{ROWA}(\mathsf{costR},UR^{\max},W,(UR,RT),S,B)$. To derive a contradiction, assume that $\mathbf{ILP}(\mathbf{A},\mathbf{b},\mathbf{c}) \neq \mathsf{costR}(UR,UR')$. Because $\mathbf{x}$ is a feasible solution for $(\mathbf{A},\mathbf{b},\mathbf{c})$ and $\mathsf{costR}(UR,UR') = \mathbf{c}^{\mathrm{T}}\mathbf{x}$, there exists a feasible solution $\mathbf{y}$ for $(\mathbf{A},\mathbf{b},\mathbf{c})$ such that

$\mathsf{costR}(UR, UR') > \mathbf{c}^{\mathrm{T}}\mathbf{y}$. Let UR'' be a user-role assignment and alloc' an allocation such that $\mathbf{y} \sim (UR'', \mathsf{alloc}')$. It follows by Lemma 10.3 that $UR'' \subseteq UR^{\max}$ and $\mathsf{alloc}' \models (W, (RT \circ UR'', S, B))$. As reasoned before, we have $\mathsf{costR}(UR, UR'') = \mathbf{c}^{\mathrm{T}}\mathbf{y}$ and therefore $\mathsf{costR}(UR, UR') > \mathsf{costR}(UR, UR'')$. However, this violates our minimality assumption for $\mathsf{costR}(UR, UR')$. Hence, $\mathbf{x}$ is an optimal solution for $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ and the two outputs are equal. ■

We now establish the space and runtime complexity of ROWAtoILP. Let $(\mathsf{costR}, UR^{\max}, W, (UR, RT), S, B)$ again be a **ROWA** instance and $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ the corresponding **ILP** instance returned by ROWAtoILP. Furthermore, let $U = \mathsf{dom}(UR^{\max})$, $R = \mathsf{ran}(UR^{\max})$, and $T = \{t \in \mathcal{T} \mid \exists i \in \mathsf{T}(W) \,.\, t \in i\}$. The **ILP** instance $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ ranges over $|U||R| + |U||T|$ decision variables, which corresponds to the same number of columns of the matrix $\mathbf{A}$. There are $|T||U|$ constraints of kind (1), $|T|$ constraints of kinds (2), $O(|S||T|^2|U|)$ constraints of kind (3), $O(|B||T|^2|U|)$ constraints of kind (4), there is one constraint of kind (5), $|U||R|$ constraints of kind (6), and $|U||T|$ constraints of kind (7). Thus, the total number of constraints is in $O(|U|(|T|^2(|S| + |B|) + |R| + |T|))$, corresponding to the same number of rows of $\mathbf{A}$. Hence, ROWAtoILP is a polynomial reduction from **ROWA** to **ILP**.

Solving **ROWA** requires solving **AEP**, which is **NP**-complete by Lemma 10.2. Therefore, the following corollary is a direct consequence of Theorem 10.1 and the observation that ROWAtoILP is a polynomial reduction from **ROWA** to the **NP**-complete problem **ILP**.

Corollary 10.2 **ROWA** *is* **NP***-complete.*

We have thereby shown that finding an optimal authorization policy that allows an obstruction-free enforcement on given workflow is in the same complexity class as deciding whether a given authorization policy allows an obstruction-free enforcement. Furthermore, the polynomial reduction from **ROWA** to **ILP** enables us to solve **ROWA** instances using well-established algorithms for **ILP**. An example follows in the next section.

Note that ROWAtoILP and an algorithm for **ILP** can also be used to solve **AEP**. Let $(W, (UT, S, B))$ be an **AEP** instance. Using a set of roles R, we decompose UT into an RBAC policy (UR, RT) such that $RT \circ UR = UT$. Furthermore, let $UR^{\max} = UR$, and costR be the role cost function composed of the auxiliary functions $\mathsf{risk}(r) = \mathsf{ma}(r) = 0$ and $\mathsf{add}(r) = \mathsf{rm}(r) = 1$, for all $r \in R$. $\mathbf{ROWA}(\mathsf{costR}, UR^{\max}, W, (UR, RT), S, B) = 0$ if and only if $\models^{\exists} (W, (UT, S, B))$. This follows from the observation that the minimal value of costR is 0, which is only possible for $\mathsf{costR}(UR, UR) = 0$, implying that $\models^{\exists} (W, (RT \circ UR, S, B))$.

10.4 Experimental Results

We return to our running example and demonstrate how off-the-shelf software can be used to solve **ROWA** instances using our reduction to **ILP**. We implemented ROWAtoILP using the numerical software MATLAB [The MathWorks 2012].

Example 10.5 (Optimal Authorization Policy for the Payment Workflow) Recall the RBAC policy (UR, RT) shown in Figure 10.3 and our observation in Example 10.3 that there exists no allocation for the workflow specification process W and $(UR \circ RT, S, B)$. Furthermore, recall the role cost function costR and the maximal user-role assignment $UR^{\max}$ presented in Example 10.4.

Using our ROWAtoILP-implementation, we transform the **ROWA** instance $(\mathsf{costR}, UR^{\max}, W, (UR, RT), S, B)$ to an **ILP** instance $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ and compute an optimal solution $\mathbf{x}$, which corresponds by Definition 10.8 to the user-role assignment $UR' = \{(\mathsf{Bob}, r_2), (\mathsf{Claire}, r_3), (\mathsf{Emma}, r_3)\}$ and the allocation $\mathsf{alloc} = \{(t_1, \mathsf{Emma}), (t_2, \mathsf{Bob}), (t_3, \mathsf{Claire}), (t_4, \mathsf{Emma}), (t_5, \mathsf{Claire}), (t_6, \mathsf{Claire})\}$. The cost of changing from UR to UR' is $\mathsf{costR}(UR, UR') = 43$. Hence the optimal administrative change with respect to costR is to extend UR by assigning Emma to the role Procurement Manager (r_3). This empowers the users to execute the payment workflow without obstructions.

Suppose now that the risk exposure changes in that the risk associated with an assignment to role r_3 increases by 3 to 15. The other numbers in Table 10.1 remain unchanged. By running our program again, we see that this small change of cost results in a different optimal solution. The optimal user-role assignment is now $UR'' = \{(\mathsf{Bob}, r_2), (\mathsf{Bob}, r_2), (\mathsf{Claire}, r_3), (\mathsf{Fritz}, r_4)\}$, the respective allocation is $\mathsf{alloc}' = \{(t_1, \mathsf{Bob}), (t_2, \mathsf{Bob}), (t_3, \mathsf{Claire}), (t_4, \mathsf{Fritz}), (t_5, \mathsf{Claire}), (t_6, \mathsf{Claire})\}$, and $\mathsf{costR}(UR, UR'') = 46$. Because the risk associated with r_3 increased, it is now cheaper, *i.e.* less risky, to assign Bob additionally to the role Procurement Clerk (r_1) and Fritz to Accountant (r_4) instead of assigning Emma to the role Procurement Manager (r_3). ★

Computing optimal solutions for **ILP** instances, such as the ones presented in the example above, takes about 100 milliseconds on a standard PC configuration.[1] We also experimented with larger, randomly generated maximal user-role assignments. On our test system, we observed an exponential increase of the running time in the size of the input, which is consistent with our complexity analysis of ROWAtoILP and Corollary 10.2. However, we did not investigate optimizations of our prototype implementation.

[1] Mac OS X, 2.5 GHz Intel Core 2 Duo, 2 GB RAM.

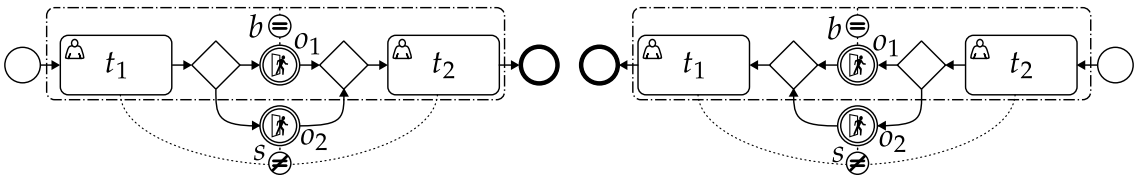
Chapter 11
Evaluation

We conclude Part III with an evaluation of our workflow-specific approach to aligning the enforcement of authorizations with business objectives.

11.1 Allocations Versus Enforcement Processes

Most related work formalizes the existence of a workflow execution that satisfies a set of authorizations similar to our notion of an allocation, see *e.g.* [Crampton 2005; Li *et al.* 2009]. As established by Lemma 10.1 and Corollary 10.1, an allocation corresponds to our graph-based approximation of enforcement processes, presented in Section 9.4. We now illustrate by means of examples the limitations of this approximation. At the same time, these examples illustrate the expressivity of enforcement processes and properties of the **EPE** problem in general.

Example 11.1 (Release Points Matter) Consider the workflow shown in Figure 11.1a, which we formalize by the workflow specification process $W_a = t_1 \to ((o_1 \to t_2 \to \mathit{SKIP}) \sqcap (o_2 \to t_2 \to \mathit{SKIP}))$. Furthermore, let $b = (\{t_1, t_2\}, o_1)$ and $s = (t_1, t_2, o_2)$ be the BoD and SoD constraints shown in Figure 11.1a and let $UT = \{(\mathsf{Alice}, t_1), (\mathsf{Alice}, t_2), (\mathsf{Bob}, t_2)\}$ be a user-task assignment. Given $T = \{t_1, t_2\}$ and $\phi = (UT, \{s\}, \{b\})$, CGraph fails to produce an instance of **ListColoring** because t_1 and t_2 are both bound and separated by b and s, respectively. Correspondingly, the output corresponding to the **AEP** instance (W, ϕ) is No. However,



a. Enforcement process but no allocation **b.** Inverted control-flow

Figure 11.1. Limitations of **EPE** approximations

no matter how the workflow is executed, when t_2 is instantiated either b's or s's history has been released and the respective constraint is irrelevant at this point. Thus, $E = t_1.\mathsf{Alice} \to ((o_1 \to t_2.\mathsf{Bob} \to \mathit{SKIP}) \sqcap (o_2 \to t_2.\mathsf{Alice} \to \mathit{SKIP}))$ is an enforcement process for ϕ on W_a. ★

This example shows that by abstracting away from release points, our approximation misses information needed to determine whether an enforcement process exists. Moreover, our approximation not only abstracts away release points but also control-flow. Example 11.2 illustrates how the existence of an enforcement process may only depend on control-flow.

Example 11.2 (Control-Flow Matters) Consider the **EPE** instance shown in Figure 11.1b, which is equivalent to the **EPE** instance examined in Example 11.1, except that the workflow's control-flow is inverted. Let $W_b = t_2 \to ((o_1 \to t_1 \to \mathit{SKIP}) \sqcap (o_2 \to t_1 \to \mathit{SKIP}))$ be the corresponding workflow specification process. There exists no enforcement process for ϕ on W_b because when t_2 is instantiated one cannot know whether the execution later passes through o_1 or o_2 and whether Alice or Bob should therefore execute t_2's instance. Hence, an algorithm for **EPE** that disregards control-flow can only approximate **EPE**. ★

11.2 When Users Become Unavailable

Dropping the assumption that authorizations are non-administrable leads to the question what to do if users become unavailable during workflow execution. Clearly, when no user is available, obstructions, in fact even deadlock, is unavoidable. If only some users become unavailable, the existence of an obstruction-free enforcement depends on which task instances these users have executed in the past and what history-independent authorizations the remaining, *i.e.* available, users have. The analysis of administrable authorizations furthermore reveals a fundamental difference between SoD and BoD constraints, illustrated by the following example.

Example 11.3 (Recovering From Unavailable Users) Consider the workflow shown in Figure 11.2, ignoring the release point o for the moment, and let $W = W_1 \,;\, (t_2 \to (W \sqcap \mathit{SKIP}))$, for $W_1 = t_1 \to (W \sqcap \mathit{SKIP})$, be the respective workflow specification process. Furthermore, let $s = (t_1, t_2, \varnothing)$ be an SoD constraint, $b = (t_1, \varnothing)$ a BoD constraint, $\mathit{UT} = \{(\mathsf{Alice}, t_1), (\mathsf{Bob}, t_2)\}$ a user-task assignment, and $\phi = (\mathit{UT}, \{s\}, \{b\})$ the corresponding authorization policy. It is easy to see that $i = \langle t_1.\mathsf{Alice}, t_2.\mathsf{Bob}\rangle$ is a workflow trace of W, satisfying ϕ.

Suppose now that after the workflow trace corresponding to i has been executed, Alice becomes unavailable, resulting in the new authorization policy

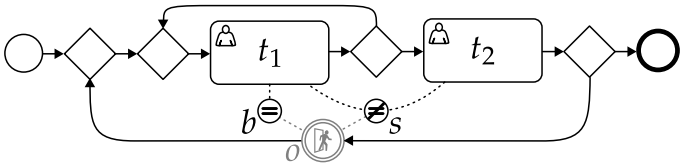


Figure 11.2. Recovering from unavailable users

$\phi' = (\{(t_2, \mathsf{Bob})\}, \{s\}, \{b\})$. If the workflow instance loops back, at least one more instance of t_1 and t_2 need to be executed before the workflow instance can successfully terminate. However, there is no user who is authorized to execute t_1 with respect to ϕ'. When evaluating possible changes to ϕ' that allow an obstruction-free enforcement, more specifically when analyzing which user to assign to t_1, it turns out that in order to satisfy s, every user except Bob is fine. Though, there is only one assignment that satisfies b, namely to again assign Alice to t_1. ★

The observation made in Example 11.3 shows the dual nature of SoD and BoD. Generalizing, if a user becomes unavailable and he has previously executed a task instance that is bound by a BoD constraint to a task that might still be executed, obstructions can only be avoided if *the very same user* is again made available. In contrast, if a user becomes unavailable and she has previously executed a task instance that is separated from another task that might still be executed, obstructions can be avoided if *arbitrary though enough users* are assigned to the appropriate tasks.

One option to lift the restrictions imposed by previous task executions is the augmentation of the workflow by release points. To give an example, consider again the workflow shown in Figure 11.2, now including the release point o, and recall the workflow trace i presented in Example 11.3. If the workflow corresponding to i loops back to the beginning, it passes through o and previous task executions become thereby irrelevant with respect to the SoD constraint $s' = (t_1, t_2, o)$ and the BoD constraint $b' = (t_1, o)$. As a result, if Alice becomes unavailable, any other user who is assigned to t_1 may execute t_1 without violating b'. Thus, performing administrative activities when a workflow passes through release points lowers the risk of obstructions.

Some real world events that trigger administrative activities, such as accidents, cannot be planned and are therefore not in sync with release points. In such cases a violation of authorizations may be unavoidable. This leads to the idea of providing some sort of exception mechanism. Authorizations may for example be associated with compensational activities, which must be executed if the respective authorizations are violated. The work on break-glass autho-

rization models [Povey 2000; Marinovic *et al.* 2011], which allow an override of default authorizations if certain conditions are met, is a promising starting point for exploring this direction of future work. Additionally, our optimization approach may be extended to identify the least risky authorization overrides that allow an obstruction-free workflow execution.

11.3 Optimizing for Partially Executed Workflows

Our notion of an allocation does not account for previously executed task instances. Consequently, **OWA** and **ROWA**, which build on the definition of an allocation, describe just the problem of finding an optimal authorization policy for a workflow (instance) that has not been executed yet. However, as modeled by administrative events in Part II, administrative changes may not only happen before a workflow is being executed but also during its execution.

In fact, in our paper [Basin *et al.* 2012c], upon which Chapter 10 is based, we do account for previously executed task instances. However, we decided to disregard partially executed workflows in Chapter 10 to formally relate obstruction-freedom and the notion of an optimal, workflow-aware authorization policy; more specifically, to relate allocations to our **EPE** approximation presented in Section 9.4. This reduction of expressivity has no impact on the complexity of **AEP** and **ROWA**. In other words, **AEP** and **ROWA** are both **NP**-complete no matter whether we do account for partially executed workflow instances or not.

Part IV

Closing

Chapter 12
Related Work

While Chapter 3 identifies related requirements arising from the business context of our work, in this chapter, we compare our work to related approaches from the scientific literature.

12.1 Authorizations

The specification of authorizations is a prerequisite for the protection of a computer system's resources [Saltzer and Schroeder 1975]. Various models and languages for capturing authorizations have been proposed. Two of the most classical models are *mandatory access control (MAC)* and *discretionary access control (DAC)*, *e.g.* formalized by Sandhu in [1993] and Harrison *et al.* in [1976], respectively. The main difference between the two models is who specifies the respective policies. In MAC, a central authority assigns security labels to both subjects and resources, which then determine authorized actions. In contrast, DAC allows every subject to alter a part of the so-called *access matrix*, which represents the ternary relation between subjects, actions, and resources. This ternary relation is often decomposed into a binary relation between subjects and *permissions*, where permissions represent action-resource pairs, see *e.g.* [Ferraiolo *et al.* 2001]. In our work, we consider only one kind of action, the execution of a task, and thus by assigning a user to a task we implicitly assign her the permission to execute the respective task.

Different decompositions of access matrices have been proposed. For example, *capability lists* correspond to their decomposition by subject, whereas *access control lists (ACLs)* correspond to their decomposition by resource, see *e.g.* [Sandhu and Samarati 1994]. A very popular decomposition is the *Role-based Access Control (RBAC)* model [Ferraiolo *et al.* 2001], which we introduced already in Chapter 4.

Various extensions to the classical subject-resource-action triple have been published to account for further security-critical elements. For example, Berti-

no *et al.* propose *temporal RBAC (TRBAC)* [2001a], an extension of RBAC by conditions constraining the temporal availability of roles. A related extension is *history-based access control* [Fong 2004] where authorizations depend on events that happened in the past. In contrast to the authorization models introduced above, TRBAC and history-based access control are history-dependent, may in fact encode a workflow instance, and are as such similar to our authorization processes. However, they do not account for potential future task executions.

These different decomposition and extension options lead to numerous design decisions. As a result, some authors propose generic, comprehensive authorization languages, such as the *extensible access control markup language (XACML)* [Parducci *et al.* 2009] and the *flexible authorization framework* [Jajodia *et al.* 2001], supporting various authorization models, whereas other authors investigate domain-specific languages, *e.g.* authorizations for XML documents [Bertino *et al.* 2001b]. Our authorization policies combine generic, history-independent authorizations, which may be described with different authorization models, and concrete, history-dependent authorizations — SoDA terms in Part II, and SoD and BoD constraints in Part III.

12.1.1 Constraints

The term *constraint* is often used for properties that an authorization policy must satisfy, in particular in the context of RBAC [Jaeger 1999; Ahn and Sandhu 2000; Ferraiolo *et al.* 2001; Bertino *et al.* 2001a; Li *et al.* 2007; Jha *et al.* 2008]. For example, Li *et al.* propose in [2007] SoD constraints of the form $(\{p_1,\ldots,p_n\},k)$, for a set of permissions $\{p_1,\ldots,p_n\}$ and $n,k \in \mathbb{N}$, which satisfy an RBAC policy if there is no set of subjects smaller than k that is together assigned to all n permissions. This example illustrates that SoD is realizable in a history-independent manner.

Related work on SoD, *e.g.* [Simon and Zurko 1997; Gligor *et al.* 1998], uses often the term *static* for what we call *history-independent* and *dynamic* for *history-dependent*. Because we distinguish authorizations both with respect to their dependency on a workflow's execution history and their administrability, the term *dynamic* is not sufficiently refined. Hence, we avoid it. We present related work on history-dependent SoD and BoD constraints in Section 12.3.

12.1.2 Enforcement

The fundamental ideas pursued in Part II are in line with the notion of *model-driven security (MDS)* [Basin *et al.* 2006] that aims at synthesizing a system's implementation from the composition of abstract specifications of its business and security requirements. In particular, Basin *et al.* [2003] generate implementations

of workflow systems that include authorization requirements. In contrast, we focus specifically on SoD constraints and our SoD as a Service implementation integrates with heterogeneous software components as opposed to being an automatically generated, monolithic software application.

Another approach to enforcing security policies is to execute an enforcement monitor in parallel to an insecure system, checking whether commands are authorized prior to their execution and blocking the insecure system if they are not. Schneider formalized this concept as a *security automaton* [2000]. To the best of our knowledge, Basin *et al.* [2007] were the first to use CSP to formalize security automata. Like them, we encode what is known in other authorization architectures as *policy decision point (PDP)* as a CSP process and the synchronous process composition constitutes the *policy enforcement point (PEP)*.

Security automata are limited in that preventing unauthorized commands either causes the target system to terminate or requires exception handling to be part of the security automaton as well as the target system. To overcome this limitation, several extensions to security automata, such as *edit automata* [Ligatti *et al.* 2005], have been proposed. Our notion of an enforcement process follows a different direction. By incorporating knowledge about the system's control-flow, modeled by a workflow, into the enforcement monitor, we avoid obstructions and thereby undesired termination.

12.1.3 Administration and Delegation

Given the large number of authorization models, there is consequently also a large number of models for administrating them. Harrison, Ruzzo, and Ullman's access matrix [1976], and in particular the commands that allow subjects to administrate parts of the matrix, is one of the first formal authorization-administration models, often called the *HRU model*. Examples of models for administrating RBAC policies are *ARBAC* [Sandhu *et al.* 1999], Crampton and Loizou's *SARBAC* [2003], and Li and Mao's *UARBAC* [2007]. All of them could be employed to refine the set of administrative events in Part II and to constrain the set of possible policy changes in Chapter 10.

A question that naturally arises from the design of an authorization-administration model is formalized by the *safety problem* [Harrison *et al.* 1976]. This problem asks whether a sequence of administrative activities can transform an initial authorization policy to another policy that satisfies a particular property, *e.g.* that a subject is given permissions he should not have. The safety problem is not decidable for the HRU model [Harrison *et al.* 1976]. However, it is decidable, more precisely **PSPACE**-complete, for ARBAC [Jha *et al.* 2008].

Delegation allows a subject to transfer authorizations to other subjects, who may then perform activities on behalf of the former subject [Barka and Sandhu 2000; Li *et al.* 2003]. Delegations are a standard approach to overcome situations where not enough subjects are available to perform an activity in a timely way. They also serve as building block for authorization languages, such as *DKAL* [2008], that are tailored to distributed and dynamic environments. We revisit delegations in the context of workflows in Section 12.3.

12.1.4 Risk

The notion of risk has been introduced into authorization models to adapt authorization policies to changing conditions. Methods to measure and quantify risks are given in [Cheng *et al.* 2007; Molloy *et al.* 2008]. Aziz *et al.* [2006] use a risk semantics to transform policies with respect to operational, combinatorial, and conflict of interest risks. In contrast to our approach presented in Chapter 10, they transform role-task assignments, leaving user-role assignments, where changes occur in practice more frequently, untouched.

To quantify the risk of delegations, Han *et al.* [2009] consider the position of the delegated role within the role hierarchy and the number of permissions gained. Associating risk and benefit vectors with every read and update transactions, Zhang *et al.* [2006] study the optimization of an allowed transaction graph with respect to a given accessibility graph that defines the underlying communication system. The cost drivers for authorization management identified by Casassa Mont *et al.* [2010] provide further metrics for defining role cost functions.

12.2 Workflows

Georgakopoulos *et al.* [1995] are among the first to propose a comprehensive workflow management terminology, which they then use to compare commercial workflow systems. Van der Aalst predicts in [1998] that workflow functionality will become an essential building block required by a large number of applications, comparable to the importance of databases. It is debatable whether workflow functionality has meanwhile reached this importance. However, analysts see *business process management (PBM)* as a core discipline for business success [Dixon and Jones 2011], the BPM market is expected to grow at double-digit rates annually during the next years [Fleming and Silverstein 2011], and the alignment of business activities and IT functionality by means of business

processes was repeatedly identified as a top priority of IT executives [McDonald and Aron 2012].

12.2.1 Patterns and Modeling Languages

There are many languages for modeling workflows, for example the *XML Process Definition Language (XPDL)* [WFMC 2008], *UML Activity Diagrams* [OMG 2011b], and *Yet Another Workflow Language (YAWL)* [YAWL 2011]. We used BPMN [OMG 2011a] for visualizing workflows and BPEL [Alves *et al.* 2007], including its human task extension BPEL4People [Agrawal *et al.* 2007], for their implementation in Part II.

To compare different workflow modeling languages, van der Aalst *et al.* identify a set of *workflow patterns* [2003] and analyze to what extent some given workflow systems support them. Later overview articles compare modeling languages not only with respect to their control-flow expressivity, *i.e.* supported workflow patterns, but also with respect to other modeling aspects such as whether the languages support visualization [Mendling *et al.* 2004; List and Korherr 2006]. Our choice of BPMN and BPEL is motivated by their expressivity, standardization, vendor-independence, and the fact that they are both well-established.

12.2.2 Formalizations and Properties

Petri Nets [Peterson 1977] is the predominant theory used by the business process community for formalizing workflows. Process calculi lack the graphical appeal of Petri Nets but their notion of parallel, synchronous process execution is well-suited to formally decompose a workflow system into different subcomponents, a technique that we use intensively. We give a few examples of workflow formalizations with process calculi: Puhlmann and Weske map generic workflow patterns to π-calculus [2005], Wong and Gibbons formalize BPMN in CSP [2008], and Cámara *et al.* use CCS to formalize BPEL [2006]. Yet another approach to formalizing workflows is pursued by Börger and Sörensen who formalize BPMN by abstract state machines [2011].

The question of what constitutes a well-formed workflow model has been extensively investigated in the business process community. For example, van der Aalst calls a workflow *sound* if it has no dead transitions and it does not deadlock before completing its final task [1996]. Obstruction-freedom is a complementary property; combined with soundness it guarantees that the workflow will always successfully terminate and thus will achieve its business objectives.

12.3 Authorizations in the Context of Workflows

In his seminal paper [1988], Sandhu introduces *transaction control expressions* for specifying history-dependent SoD constraints on data objects. A workflow, associated with a data object, is defined by a list of tasks, each with one or more attached roles. A user is authorized to execute a task if she acts in one of these roles. By default, all tasks must be executed by different users.

Bertino *et al.* [1999] propose a framework, often called the *BFA model*, for specifying (first-order) predicates on the execution of a workflow's tasks. Knorr and Stormer [2002] map history-dependent SoD constraints along with basic workflow models to Prolog clauses in order to compute all workflow instances that satisfy the specified constraints.

In Nash and Poland's *object-based SoD* [1990], each data object keeps track of the users who have executed actions on it. If a user requests to execute an action on an object, this is only granted if he has not executed an action on this object before. By associating a SoDA-enforcement process with every data object, this functionality could also be modeled with our formalisms.

Crampton *et al.* develop an algebraic formalism to constrain a workflow's task executions [Crampton 2003; Tan *et al.* 2004; Crampton 2005]. Their constraints have the form $(U,(t_1,t_2),\pi)$, for a set of users U, a pair of tasks (t_1,t_2), and a binary relation $\pi \subseteq \mathcal{U} \times \mathcal{U}$ on users. If a user $u_1 \in U$ executes t_1, then a user u_2 who executes t_2 must satisfy $(u_1,u_2) \in \pi$. The relation π may encode SoD or BoD constraints but also other properties such seniority. In this respect, Crampton's model is similar to our model proposed in Part III but it is more expressive in terms of the relations it supports between users.

Wolter *et al.* [2007; 2008; 2010] propose an extension to BPMN to visually model *entailment constraints* that allow the specification of lower and upper bounds on the number of different users that must execute a set of tasks. To enforce their constraints, the authors propose a translation to XACML [2007].

None of the above models supports a scoping of constraints to task instances, such as our concept of release. In the following, we discuss what kind of workflow patterns these models support, what analysis algorithms the respective authors propose, and relate the models to the authorization classification introduced in Chapter 1.

12.3.1 Workflow Abstraction

Early work on authorizations in the context of workflows, *e.g.* Sandhu's transaction control expressions [1988], model workflows only as part of the constraints. The BFA model is the first to model workflows explicitly, formalizing them as se-

quences of tasks [Bertino *et al.* 1999]. Later, Crampton refines workflow models to partially ordered sets of tasks [2005].

These workflow formalizations and most other work on authorizations in the context of workflows do not support loops and conditional execution. A notable exception is Solworth [2006], who models a workflow as a directed graph. However, authorizations in the presence of loops are restricted such that the first task must always be executed by the same person. Given a sufficient number of users per task, this restriction ensures that a workflow instance can always successfully terminate if there are no conflicts between SoD and BoD constraints. The graph transformation used in CGraph is inspired by Solworth's conflict graph [2006].

Our CSP-based formalization of workflows imposes no restriction on a workflow's control-flow. By introducing release points into workflows, we support a fine-grained scoping of authorizations to subsets of task instances. This feature is particularly powerful in the presence of loops.

12.3.2 Analysis

Crampton was the first to analyze the computational complexity of deciding for a given workflow whether an allocation of users to tasks exists such that an authorization policy is satisfied [2005]. In [2010], Wang and Li call this decision problem the *workflow satisfiability problem* and prove it to be **NP**-complete for their authorization model. **AEP** is an adaptation of the workflow satisfiability problem to our SoD and BoD constraints and serves as a building block for the definition of **OWA** and **ROWA**.

As illustrated in Chapter 11, enforcement processes are more expressive in terms of the authorization policies they support than an allocation of users to tasks. However, this additional expressivity comes at the cost of **EPE**'s double exponential complexity, which led us to investigate **EPE** approximations. Solworth's notion of *scheduled approvability* [2006], requiring that every workflow instance can be extended to a final state no matter which path is taken, lies in terms of expressivity between allocations and enforcement processes.

Schaad *et al.* employ a symbolic model checker to automate the analysis of whether delegations allow a circumvention of SoD constraints [2006]. Wolter *et al.* [2009] use the model checker SPIN [Holzmann 2003] to find inconsistencies in their BPMN-based authorization model. To solve instances of the workflow satisfiability problem, Wang and Li use SAT solvers [2010]. We experimented with FDR [FSE 2005] on small instances of the **EPE** problem. However, a thorough investigation of this automation approach remains future work.

To the best of our knowledge, we are the first to employ integer programming to compute optimal, workflow-aware authorization policies. However,

planning algorithms developed in the operations research community, *e.g.* [Blum and Furst 1997], solve the related problem of finding an optimal allocation of resources that satisfies a set of constraints. In particular, Senkul *et al.* study the planning problem in the context of workflows [2002].

12.3.3 Modeling Scope

Most authorization models for workflows describe constraints between two or more explicit tasks and are therefore tightly coupled with a workflow model [Sandhu 1988; Bertino *et al.* 1999; Kandala and Sandhu 2002; Crampton 2005; Wang and Li 2010; Wolter and Meinel 2010]. However, there are exceptions, such as Li and Wang's SoDA [2008], that are workflow-independent. This has the advantages presented in Chapter 5 but poses the challenges elaborated in Chapter 7.

All of the above mentioned workflow authorization models are history-dependent. With respect to history-dependence and modeling scope, the only combination that we did not consider is history-independent, workflow-specific authorizations. An example of such authorizations are lanes, which may be used to group tasks with respect to the organizational unit that is responsible for their execution [Silver 2009; OMG 2011b]. Lanes are sometimes directly interpreted as workflow-specific roles [OMG 2011b]. However, if tasks can be executed by multiple roles, a workflow may have exponentially many lanes in the number of roles, which leads to large representations.

12.3.4 Administrability

Most work on authorization-constrained workflows implicitly assumes that authorizations are non-administrable, *i.e.* the respective policies are not edited during workflow execution. One exception is the work on delegation in workflow systems. Building on and extending the seminal work of Atluri *et al.* [2003], different delegation models for workflows have been proposed [Atluri and Warner 2005; Wainer *et al.* 2007]. Crampton and Khambhammettu [2008] were the first to check if permitting a delegation request prevents the completion of a workflow instance.

Another exception is the *workflow resiliency problem* introduced by Wang and Li [2010], which asks whether a workflow can be executed successfully if a given number of users is unavailable. However, none of these works consider optimality of authorization policies.

12.3.5 Workflow Language Extensions and Tool Support

Different authors use BPMN's extension mechanism to augment workflow models with security requirements [Wolter and Schaad 2007; Rodríguez *et al.* 2007]. We see this approach as an instance of model-driven security [Basin *et al.* 2006]. Our BPMN and Oryx extension builds on Rüegger's master thesis [2011]. Wolter *et al.*[2009] also extend Oryx to provide tool support for their workflow authorization model. In particular, they invoked the model checker SPIN in Oryx's application tier to reason about their constraint models.

Although not fully specified, the query language for people links in BPEL4People allows one to specify basic SoD constraints [Agrawal *et al.* 2007; Mendling *et al.* 2008]. By using SoDA, SoD as a Service supports more expressive constraints. Paci *et al.* [2008] propose another authorization extension for BPEL, which is based on Crampton's model [2005]. Authorizations, including SoD constraints, are enforced by a web service, which pools all information that is relevant for enforcement: the history of workflow instances, an RBAC policy, and SoD constraints. The underlying workflow model [Crampton 2005], however, does not support loops, which is in conflict with the workflow patterns supported by BPEL.

Chapter 13
Conclusion

We conclude this dissertation with a summary of our results and an outlook on open problems and future work.

13.1 Summary

We presented two orthogonal approaches, each addressing a separate problem in the area of authorization-constrained workflows.

Refinement and Integration. In Part II, we studied a workflow-independent approach to refining abstract SoD constraints to concrete workflow models and to integrating an enforcement monitor for the resulting refinement into a dynamic and heterogeneous workflow environment. One of our central assumptions was that a workflow-independent, abstract language for SoD constraints facilitates a separation of concerns between business and security personnel. This separation enables a higher degree of flexibility with respect to integration and enforcement.

Starting out with a generalization of SoDA's semantics and ending up with a prototype implementation, we went the full distance from theory to practice. The key ideas were, first, to generalize SoDA's semantics to traces and to describe a mapping from terms to CSP processes that accept the respective traces; second, to use our CSP formalization as blueprint for the architecture of SoD as a Service, which enables the dynamic integration and configuration of SoD enforcement in a SOA; and third, to make our formalization and implementation account for administrative activities that change authorizations during workflow execution. The choice of software components for our proof-of-concept implementation illustrates how SoD as a Service enables the integration of new internal controls into existing workflow environments. Our approach allows enterprises to quickly adapt to organizational, regulatory, and technological changes.

Alignment and Optimality. In Part III, we presented a workflow-specific approach to aligning the enforcement of authorizations with their workflow-based business environment. We then studied the concept of an optimal authorization policy that allows such an alignment. Our notion of optimality comes with considerable modeling freedom. For example, the objective function can model the risk associated with an authorization policy and, hence, the optimal policy minimizes risk, which corresponds to maximizing protection.

These results serve to strike a balance between the protection of a system's resources and the empowerment of its users. The key ideas here were to first augment workflow models with release points to scope authorizations to subsets of task instances without imposing restrictions on a workflow's control-flow; second, to model workflows at two levels of abstraction, to link these levels by the notion of an obstruction, and to formalize an obstruction-free enforcement in terms of a process; and third, to model the cost associated with changes to authorization policies by a function and to cast the problem of finding a cost-minimizing authorization policy as a well-known optimization problem.

We experimentally validated both of our approaches with a respective implementation. Furthermore, we supported our complexity analyses with performance measurements from realistic example workflows.

13.2 Outlook

Our work gives rise to many interesting follow-up questions. We present four of them in more detail.

Combining Our Two Approaches. An obvious direction for future work is to study further refinements of our SoDA-based approach presented in Part II using our results on alignment developed in Part III. More specifically, to investigate how a SoDA-enforcement process could be refined to an enforcement process that avoids obstructions.

Conversely, the semantics of SoDA points us at potential refinements of enforcement processes. We designed enforcement processes for the authorization model presented in Chapter 8, which has the property that every prefix of a trace that satisfies an authorization policy ϕ, also satisfies ϕ. Thus, every trace that is accepted by an enforcement process satisfies the respective authorization policy. As observed in Section 7.2, SoDA$^{\mathsf{T}}$ does not have the same prefix property, in other words, a prefix of a trace that satisfies a SoDA term ϕ does not necessarily satisfy ϕ, too. Hence, refining enforcement processes such that they would

guarantee that every accepted prefix i can be extended by a trace i' such that $i \hat{} i'$ satisfies the respective SoDA term ϕ, is another direction of future work.

Extended Study of Enforcement Processes. As shown in Chapter 11, enforcement processes are more expressive in terms of the authorizations they can enforce without obstructions than allocations. Moreover, the concept of an enforcement process is neither tied to CSP nor to our authorization model. It is therefore worthwhile to explore the use of enforcement processes for other authorization models and to study the construction of enforcement processes in more detail. For example, given a workflow specification process W and an authorization policy ϕ, many processes may meet the conditions of an enforcement process for ϕ on W as required by Definition 9.2. This raises the question of what constitutes a "good" enforcement process. One possibility is to search for an enforcement process $E_{\phi,W}$ such that $\mathsf{T}(A_\phi) \setminus \mathsf{T}(E_{\phi,W})$ is minimal. In other words, one that maximizes the number of authorized execution events and thereby minimizes the restrictions enforced at the execution level.

We would also like to sharpen our complexity analysis for **EPE**, ideally finding upper-bounds that match the lower-bounds we have given.

Refinement of Optimization. The generality of our optimization approach gives rise to many design decisions and consequently to various directions for future work. For example, other workflow authorization models provide different features than our model introduced in Chapter 8, *e.g.* support for delegation [Crampton and Khambhammettu 2008]. Similarly, user-task assignments can be refined based on different authorization models and our role-based cost function could be further refined to account for additional properties such as role hierarchies.

Meaningful risk metrics for authorization policies are a precondition for the effective use of our optimization approach. We pointed in Section 12.1.4 to various methods for quantifying the risk associated with authorization policies. However, finding such metrics is challenging. This does not, of course, reduce the importance of such metrics and we see our results as providing additional evidence for their usefulness.

Combination of Obstruction-Freedom and Safety Problem. Our authorization administration models are abstract and basic. We simply allow the assignment and the removal of users to roles and restrict in Chapter 10 additionally the set of possible assignments by maximal user-role assignments. As summarized in

Section 12.1.3, there exist various administration models that may be used to describe possible authorization changes in more detail.

With a more concrete administration model comes the question whether certain sequences of administrative activities may lead to undesired authorization policies. As introduced in Section 12.1.3, related work calls this question the safety problem. To study the safety problem of administrative models in the context of workflows leads to the question how safety relates to obstructions. For example, the safety problem could be generalized to not only range over the transitions described by the administrative model but also encompass the transitions corresponding to potential task executions. Thus, understanding the relation between obstruction-freedom and the safety problem promises to disentangle the interdependency between the execution of workflows and the administration of related authorizations.

As this list of topics for future work and the evaluation of our results in Chapters 7 and 11 show, many questions in the intersection of security and business process management remain to be studied. Additionally, we expect that this research field will gain in importance as the number of security threats and regulations keeps growing and the level of automation and distribution of work further increases.

Bibliography

Activiti 2012. Activiti BPM Platform. `www.activiti.org`.

Agrawal, A., Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., König, D., Leymann, F., Müller, R., Pfau, G., Plösser, K., Rangaswamy, R., Rickayzen, A., Rowley, M., Schmidt, P., Trickovic, I., Yiu, A., and Zeller, M. 2007. *WS-BPEL extension for people (BPEL4People)*, v. 1.0.

Ahn, G.-J. and Sandhu, R. S. 2000. Role-based authorization constraints specification. *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 207–226. ACM Press, New York, NY, USA.

Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guizar, A., Kartha, N., Liu, C. K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., and Yiu, A. 2007. *Web services business process execution language (WS-BPEL)*, v. 2.0. OASIS Standard, OASIS, Burlington, MA, USA.

Anderson, A. 2005. *Hierarchical resource profile of XACML*, v. 2.0. OASIS Standard, OASIS, Burlington, MA, USA.

APQC 2009. *Process classification framework (PCF)*. American Productivity and Quality Center (APQC), Houston, TX, USA.

ASF 2012a. Apache Axis2, `http://ws.apache.org/axis2`. The Apache Software Foundation (ASF), Forest Hill, MD, USA.

ASF 2012b. Apache Tomcat, `http://tomcat.apache.org`. The Apache Software Foundation (ASF), Forest Hill, MD, USA.

Atluri, V., Bertino, E., Ferrari, E., and Mazzoleni, P. 2003. Supporting delegation in secure workflow management systems. In *Proceedings of the 17th Annual Working Conference on Data and Application Security*, pp. 190–202. Springer, Berlin, Germany.

Atluri, V. and Warner, J. 2005. Supporting conditional delegation in secure workflow management systems. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT '05)*, pp. 49–58. ACM Press, New York, NY, USA.

Aziz, B., Foley, S. N., Herbert, J., and Swart, G. 2006. Reconfiguring role based access control policies using risk semantics. *Journal of High Speed Networks*, vol. 15, no. 3, pp. 261–273. IOS Press, Amsterdam, The Netherlands.

Barka, E. and Sandhu, R. S. 2000. Framework for role-based delegation models. In *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC '00)*. pp. 168–176. IEEE Computer Society Press, Los Alamitos, CA, USA.

Basin, D., Burri, S. J., and Karjoth, G. 2011a. Separation of duties as a service. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS '11)*, pp. 423–429. ACM Press, New York, NY, USA.

Basin, D., Burri, S. J., and Karjoth, G. 2012a (**accepted for publication**). Dynamic enforcement of abstract separation of duty constraints. *ACM Transactions on Information and System Security (TISSEC)*. ACM Press, New York, NY, USA.

Basin, D., Burri, S. J., and Karjoth, G. 2012b (**under submission**). Obstruction-free authorization enforcement: Aligning security and business objectives. *Journal of Computer Security (JCS)*. IOS Press, Amsterdam, The Netherlands.

Basin, D., Burri, S. J., and Karjoth, G. 2012c. Optimal workflow-aware authorizations. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies (SACMAT '12)*, pp. 93–102. ACM Press, New York, NY, USA.

Basin, D., Doser, J., and Lodderstedt, T. 2003. Model driven security for process-oriented systems. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT '03)*, pp. 100–109. ACM, New York, NY, USA.

Basin, D., Doser, J., and Lodderstedt, T. 2006. Model driven security: From UML models to access control infrastructures. *ACM Transactions on Software Engineering Methodologies (TOSEM)*, vol. 15, no. 1, pp. 39–91. ACM, New York, NY, USA.

Basin, D., Schaller, P., and Schläpfer, M. 2011b. *Applied Information Security*. Springer, Berlin, Germany.

Basin, D., Burri, S. J., and Karjoth, G. 2009. Dynamic enforcement of abstract separation of duty constraints. In *Proceedings of the 14th European Symposium on Research in Computer Security (ESORICS '09)*, pp. 250–267. Springer, Berlin, Germany.

Basin, D., Burri, S. J., and Karjoth, G. 2011c. Obstruction-free authorization enforcement: Aligning security with business objectives. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium (CSF '11)*, pp. 99–113. IEEE Computer Society Press, Los Alamitos, CA, USA.

Basin, D., Olderog, E.-R., and Sevinç, P. E. 2007. Specifying and analyzing security automata using CSP-OZ. In *Proceedings of the 2nd ACM Symposium on Information,*

Computer and Communications Security (ASIACCS '07), pp. 70–81. ACM, New York, NY, USA.

Bell, D. E. and LaPadula, L. J. 1973. *Secure computer systems: Mathematical foundations*. Technical Report MTR-2547, The Mitre Corporation, Bedford, MA, USA.

Berre, D. L. and Parrain, A. 2010. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, vol. 7, pp. 59–64. IOS Press, Amsterdam, The Netherlands.

Bertino, E., Bonatti, P. A., and Ferrari, E. 2001a. TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 191–233. ACM, New York, NY, USA.

Bertino, E., Castano, S., and Ferrari, E. 2001b. Securing XML documents with Author-X. *IEEE Internet Computing*, vol. 5, no. 3, pp. 21–31. IEEE Computer Society Press, Los Alamitos, CA, USA.

Bertino, E., Ferrari, E., and Atluri, V. 1999. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security (TISSEC)*, vol. 2, no. 1, pp. 65–104. ACM, New York, NY, USA.

Blum, A. and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence*, vol. 90, no. 1–2, pp. 281–300. Elsevier, Amsterdam, The Netherlands.

Börger, E. and Sörensen, O. 2011. BPMN core modeling concepts: Inheritance-based execution semantics. In *Handbook of conceptual modeling*, ch. 9, D. W. Embley and B. Thalheim, Eds. Springer, Berlin, Germany.

Cámara, J., Canal, C., Cubo, J., and Vallecillo, A. 2006. Formalizing WSBPEL business processes using process algebra. *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 154, no. 1, pp. 159–173. Elsevier, Amsterdam, The Netherlands.

Casa Mont, M., Beresnevichiene, Y., Pym, D., and Shiu, S. 2010. Economics of identity and access management: Providing decision support for investments. In *Network Operations and Management Symposium Workshops (NOMS Wksps)*, pp. 134 –141. IEEE Computer Society Press, Los Alamitos, CA, USA.

Chartrand, G. and Zhang, P. 2008. *Chromatic graph theory*. Chapman & Hall/CRC Press, Boca Raton, FL, USA.

Cheng, P.-C., Rohatgi, P., Keser, C., Karger, P. A., Wagner, G. M., and Reninger, A. S. 2007. Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P '07)*, pp. 222–230. IEEE Computer Society Press, Los Alamitos, CA, USA.

COSO 2011. *Internal control - integrated framework*, Public exposure draft. The Committee of Sponsoring Organizations of the Treadway Commission (COSO), New York, NY, USA.

Crampton, J. 2003. Specifying and enforcing constraints in role-based access control. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT '03)*, pp. 43–50. ACM, New York, NY, USA.

Crampton, J. 2005. A reference monitor for workflow systems with constrained task execution. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT '05)*, pp. 38–47. ACM, New York, NY, USA.

Crampton, J. and Khambhammettu, H. 2008. Delegation and satisfiability in workflow systems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT '08)*, pp. 31–40. ACM, New York, NY, USA.

Crampton, J. and Loizou, G. 2003. Administrative scope: A foundation for role-based administrative models. *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 2, pp. 201–231. ACM, New York, NY, USA.

Curtis, B., Kellner, M. I., and Over, J. 1992. Process modeling. *Communications of the ACM*, vol. 35, no. 9, pp. 75–90. ACM, New York, NY, USA.

Dixon, J. and Jones, T. 2011. *Hype cycle for business process management, 2011*. Gartner Report G00214214, Gartner, Stamford, CT, USA.

E&Y 2009. *European fraud survey 2009 – is integrity a casualty of the downturn?* Technical Report, Ernest & Young, London, UK.

EGEI 2009. *Final report of the expert group on e-invoicing*. Expert Group on e-Invoicing (EGEI), European Commission, Brussels, Belgium.

Engel, E., Hayes, R. M., and Wang, X. 2007. The The Sarbanes-Oxley Act and firms' going-private decisions. *Journal of Accounting and Economics*, vol. 44, no. 1–2, pp. 116–145. Elsevier, Amsterdam, The Netherlands.

Ferraiolo, D. F., Sandhu, R. S., Gavrila, S. I., Kuhn, D. R., and Chandramouli, R. 2001. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274. ACM, New York, NY, USA.

Fleming, M. and Silverstein, K. 2011. *Woldwide business process management software 2011–2015 forecast*. IDC Market Analysis 229788, International Data Corporation (IDC), Framingham, MA, USA.

Fong, P. W. L. 2004. Access control by tracking shallow execution history. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P '04)*, pp. 43–55. IEEE Computer Society Press, Los Alamitos, CA, USA.

Frisch, M. 2010. *Entwürfe zu einem dritten Tagebuch*. Suhrkamp Verlag, Berlin, Germany.

FSE 2005. *Failures-divergence refinement, FDR2 user manual*. Formal Systems Europe (FSE), Oxford, UK.

Georgakopoulos, D., Hornick, M. F., and Sheth, A. P. 1995. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, vol. 3, no. 2, pp. 119–153. Kluwer Academic Publishers, Boston, MA, USA.

Gligor, V. D., Gavrila, S. I., and Ferraiolo, D. 1998. On the formal definition of separation-of-duty policies and their composition. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P '98)*, pp. 172–183. IEEE Computer Society Press, Los Alamitos, CA, USA.

Gurevich, Y. and Neeman, I. 2008. DKAL: Distributed-knowledge authorization language. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF '08)*, pp. 149–162. IEEE Computer Society Press, Los Alamitos, CA, USA.

Han, W., Ni, Q., and Chen, H. 2009. Apply measurable risk to strengthen security of a role-based delegation supporting workflow system. In *In Proceedings of the IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY '09)*, pp. 45–52. IEEE Computer Society Press, Los Alamitos, CA, USA.

Harrison, M. A., Ruzzo, W. L., and Ullman, J. D. 1976. Protection in operating systems. *Communications of the ACM*, vol. 19, no. 8, pp. 461–471. ACM, New York, NY, USA.

Haugland, S., Cade, M., and Orapallo, A. 2004. *J2EE 1.4: The big picture*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Herman, G. A. and Malone, T. W. 2003. What is in the process handbook? In *Organizing business knowledge: The MIT process handbook*, ch. 8, pp. 221–258. The MIT Press, Cambridge, MA, USA.

Holzmann, G. J. 2003. *The SPIN model checker: Primer and reference manual*. Addison-Wesley, Boston, MA, USA.

IBM 2007. *Sequencing the DNA of IT management: IBM process reference model for IT (PRM-IT)*. IBM Corporation, Armonk, NY, USA.

IBM 2009. *IBM industry models for insurance, the insurance process and service models, general information manual*. IBM Corporation, Armonk, NY, USA.

IBM 2010a. IBM Tivoli Unified Process (ITUP). IBM Corporation, Armonk, NY, USA.

IBM 2010b. Information Framework (IFW). IBM Corporation, Armonk, NY, USA.

IBM 2011. WebSphere Process Server (WSP), v. 6.2. IBM Corporation, Armonk, NY, USA.

IBM 2012a. Tivoli Directory Server (TDS), v. 6. IBM Corporation, Armonk, NY, USA.

IBM 2012b. WebSphere Application Server (WAS), v. 6.1. IBM Corporation, Armonk, NY, USA.

ISO 2005a. *ISO Standard 27001-2005: Information technology – Security techniques – Information security management systems – Requirements*. International Standardization Organization (ISO), Geneva, Switzerland.

ISO 2005b. *ISO Standard 27002-2005: Information technology – Security techniques – Code of practice for information security management*. International Standardization Organization (ISO), Geneva, Switzerland.

ISO 2009. *ISO Standard 27000-2009: Information technology – Security techniques – Information security management systems – Overview and vocabulary*. International Standardization Organization (ISO), Geneva, Switzerland.

ITGI 2005. *Control objectives for information and related technology (COBIT) 4.1*. The IT Governance Institute (ITGI), Rolling Meadows, IL, USA.

Jaeger, T. 1999. On the increasing importance of constraints. In *Proceedings of the 4th ACM workshop on Role-based access control (RBAC '99)*, pp. 33–42. ACM, New York, NY, USA.

Jajodia, S., Samarati, P., Sapino, M. L., and Subrahmanian, V. S. 2001. Flexible support for multiple access control policies. *ACM Transactions on Database Systems (TODS)*, vol. 26, no. 2, pp. 214–260. ACM, New York, NY, USA.

Jha, S., Li, N., Tripunitara, M. V., Wang, Q., and Winsborough, W. H. 2008. Towards formal verification of role-based access control policies. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 5, no. 4, pp. 242–255. IEEE Computer Society Press, Los Alamitos, CA, USA.

Kandala, S. and Sandhu, R. S. 2002. Secure role-based workflow models. *Proceedings of the 15th annual working conference on Database and application security (Das '01)*, pp. 45–58. Kluwer Academic Publishers, Boston, MA, USA.

Li, N., Grosof, B. N., and Feigenbaum, J. 2003. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 1, pp. 128–171. ACM, New York, NY, USA.

Li, N. and Mao, Z. 2007. Administration in role-based access control. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS '07)*, pp. 127–138. ACM, New York, NY, USA.

Li, N., Tripunitara, M. V., and Bizri, Z. 2007. On mutually exclusive roles and separation-of-duty. *ACM Transactions on Information and System Security (TISSEC)*, vol. 10, no. 2, pp. 1–36. ACM, New York, NY, USA.

Li, N. and Wang, Q. 2008. Beyond separation of duty: An algebra for specifying high-level security policies. *Journal of the ACM (JACM)*, vol. 55, no. 3, pp. 12.1–46. ACM, New York, NY, USA.

Li, N., Wang, Q., and Tripunitara, M. V. 2009. Resiliency policies in access control. *ACM Transactions on Information and System Security (TISSEC)*, vol. 12, no. 4, pp. 20.1–34. ACM, New York, NY, USA.

Ligatti, J., Bauer, L., and Walker, D. 2005. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security (IJIS)*, vol. 4, no. 1-2, pp. 2–16. Springer, Berlin, Germany.

List, B. and Korherr, B. 2006. An evaluation of conceptual business process modelling languages. In *Proceedings of the ACM Symposium on Applied Computing (SAC '06)*, pp. 1532–1539. ACM, New York, NY, USA.

Marino, D., Potral, J. J., Hall, M., Rodriguez, C. B., Rodriguez, P. S., Sobota, J., Jiri, M., and Asnar, Y. D. W. 2009. *D1.2.1: MASTER scenarios*. FP7 EU project MASTER.

Marinovic, S., Craven, R., Ma, J., and Dulay, N. 2011. Rumpole: a flexible break-glass access control model. In *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies (SACMAT '11)*, pp. 73–82. ACM, New York, NY, USA.

McDonald, M. P. and Aron, D. 2012. *Amplifying the enterprise: The 2012 CIO agenda*. Gartner Report G00230430, Gartner, Stamford, CT, USA.

Mendling, J., Neumann, G., and Nüttgens, M. 2004. A comparison of XML interchange formats for business process modelling. In *Beiträge des Workshops der GI-Fachgruppe Informationssysteme im E-Business und E-Government (EMISA)*, pp. 129–140. Gesellschaft für Informatik, Bonn, Germany.

Mendling, J., Ploesser, K., and Strembeck, M. 2008. Specifying separation of duty constraints in BPEL4People processes. In *Proceedings of the 11th International Conference on Business Information Systems (BIS '08)*, pp. 273–284. Springer, Berlin, Germany.

Molloy, I., Cheng, P.-C., and Rohatgi, P. 2008. Trading in risk: Using markets to improve access control. In *Proceedings of the New Security Paradigms Workshop (NSPW '08)*, pp. 107–125. ACM, New York, NY, USA.

Nash, M. J. and Poland, K. R. 1990. Some conundrums concerning separation of duty. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P '90)*, pp. 201–207. IEEE Computer Society Press, Los Alamitos, CA, USA.

OGC 2010. Information technology infrastructure library (ITIL). Office of Government Commerce (OGC), Norwich, UK.

OMG 2011a. *Business process model and notation (BPMN)*, v. 2.0. OMG Standard, Object Management Group (OMG), Needham, MA, USA.

OMG 2011b. *OMG unified modeling language (OMG UML), superstructure*, v. 2.4.1. OMG Standard, Object Management Group (OMG), Needham, MA, USA.

Oryx 2012. The Oryx project. `http://bpt.hpi.uni-potsdam.de/Oryx`. Business Process Technology group, Hasso-Plattner-Institute, Potsdam, Germany.

OSI 2012. The MIT license. Open Source Initiative (OSI), San Francisco, CA, USA.

Paci, F., Bertino, E., and Crampton, J. 2008. An access-control framework for WS-BPEL. *International Journal of Web Services Research*, vol. 5, no. 3, pp. 20–43. IGI Publishing, Hershey, PA, USA.

Parducci, B., Lockhart, H., and Rissanen, E. 2009. *Extensible access control markup language (XACML)*, v. 3.0. OASIS Standard, OASIS, Burlington, MA, USA.

Peterson, J. L. 1977. Petri nets. *ACM Computing Surveys*, vol. 9, no. 3, pp. 223–252. ACM Press, New York, NY, USA.

Polak, D. 2007. *Oryx – BPMN stencil set implementation*. Bachelor Thesis, Hasso-Plattner-Institute, Potsdam, Germany.

Povey, D. 2000. Optimistic security: a new access control paradigm. In *Proceedings of New Security Paradigms Workflow (NSPW '99)*, pp. 40–45. ACM, New York, NY, USA.

Puhlmann, F. and Weske, M. 2005. Using the π-calculus for formalizing workflow patterns. In *Proceedings of the 3rd International Conference on Business Process Management (BPM '05)*, pp. 153–168. Springer, Berlin, Germany.

Rodríguez, A., Fernández-Medina, E., and Piattini, M. 2007. A BPMN extension for the modeling of security requirements in business processes. *Transactions on Information and Systems*, vol. E90-D, no. 4, pp. 745–752. The Institute of Electronics, Information and Communication Engineers (IEICE), Tokyo, Japan.

Roscoe, A. W. 1994. Model-checking CSP. In *A classical mind: Essays in honour of C. A. R. Hoare*, pp. 353–378. Prentice Hall, Upper Saddle River, NJ, USA.

Roscoe, A. W. 2005. *The theory and practice of concurrency*. Pearson, Upper Saddle River, NJ, USA.

Rüegger, D. 2011. *Tool support for authorization-constrained workflows*. Master's thesis ETH Zurich, Switzerland.

Saltzer, J. and Schroeder, M. 1975. The protection of information in computer systems. *Proceeding of the IEEE*, vol. 63, no. 9, pp. 1278–1308. IEEE Computer Society Press, Los Alamitos, CA, USA.

Sandhu, R. S. and Samarati, P. 1994. Access control: Principle and practice. *IEEE Communications Magazine*, vol. 32, no. 9, pp. 40–48. IEEE Computer Society Press, Los Alamitos, CA, USA.

Sandhu, R. S. 1988. Transaction control expressions for separation of duties. In *Proceedings of the 4th IEEE Aerospace Computer Security Applications Conference*, pp. 282–286. IEEE Computer Society Press, Los Alamitos, CA, USA.

Sandhu, R. S. 1993. Lattice-based access control models. *IEEE Computer*, vol. 26, no. 11, pp. 9–19. IEEE Computer Society Press, Los Alamitos, CA, USA.

Sandhu, R. S., Bhamidipati, V., and Munawer, Q. 1999. The ARBAC97 model for role-based administration of roles. *ACM Transactions on Information and System Security (TISSEC)*, vol. 2, no. 1, pp. 105–135. ACM Press, New York, NY, USA.

SCC 2010. Supply chain operations reference (SCOR) model, v. 10. Supply Chain Council (SCC), Cypress, TX, USA.

Schaad, A., Lotz, V., and Sohr, K. 2006. A model-checking approach to analysing organisational controls in a loan origination process. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT '06)*, pp. 139–149. ACM Press, New York, NY, USA.

Schneider, F. B. 2000. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 1, pp. 30–50. ACM Press, New York, NY, USA.

Schrijver, A. 1998. *Theory of linear and integer programming*. Wiley, Hoboken, NJ, USA.

SEC 2003. *Final rule: Management's report on internal control over financial reporting and certification of disclosure in exchange act periodic reports*. Securities and Exchange Commission (SEC), Washington, DC, USA.

Senkul, P., Kifer, M., and Toroslu, I. H. 2002. A logical framework for scheduling workflows under resource allocation constraints. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB '02)*, pp. 694–705. Very Large Data Base Endowment.

Signavio 2012. Signavio Process Editor. www.signavio.com.

Silver, B. 2009. *BPMN method & style*. Cody-Cassidy Press, Aptos, CA, USA.

Simon, R. and Zurko, M. E. 1997. Separation of duty in role-based environments. In *Proceedings of the 10th IEEE Workshop on Computer Security Foundations (CSFW '97)*, pp. 183–194. IEEE Computer Society Press, Los Alamitos, CA, USA.

Solworth, J. A. 2006. Approvability. In *Proceedings of the 1st ACM Symposium on Information, Computer and Communications Security (ASIACCS '06)*, pp. 231–242. ACM Press, New York, NY, USA.

SOX 2002. *Sarbanes-Oxley Act of 2002*. United States Government Printing Office, Washington, DC, USA.

Syropoulos, A. 2000. Mathematics of multisets. In *Proceedings of the Workshop on Multiset Processing (WMP '00)*, pp. 347–358. Springer, Berlin, Germany.

Tan, K., Crampton, J., and Gunter, C. A. 2004. The consistency of task-based authorization constraints in workflow systems. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW '04)*, pp. 155–169. IEEE Computer Society Press, Los Alamitos, CA, USA.

The Economist 2001. Enron, see you in court. *The Economist*, Nov. 15, London, UK.

The MathWorks 2012. Matlab r2011b. The MathWorks, Natick, MA, USA.

Turner, M., Budgen, D., and Brereton, P. 2003. Turning software into a service. *Computer*, vol. 36, pp. 38–44. IEEE Computer Society Press, Los Alamitos, CA, USA.

van der Aalst, W. M. P. 1996. *Structural characterization of sound workflow nets*. Computing Science Reports 96123, Eindhoven University of Technology, The Netherlands.

van der Aalst, W. M. P. 1998. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers (JCSC)*, vol. 8, no. 1, pp. 21–66. World Scientific Publishing, Singapore.

van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., and Barros, A. P. 2003. Workflow patterns. *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51. Springer, Berlin, Germany.

W3C 2010. *XML schema*. World Wide Web Consortium (W3C), Sophia Antipolis, France.

W3C 2011. *Extensible markup language (XML)*. World Wide Web Consortium (W3C), Sophia Antipolis, France.

Wainer, J., Kumar, A., and Barthelmess, P. 2007. DW-RBAC: A formal security model of delegation and revocation in workflow systems. *Information Systems*, vol. 32, no. 3, pp. 365–384. Elsevier, Amsterdam, The Netherlands.

Wang, Q. and Li, N. 2010. Satisfiability and resiliency in workflow authorization systems. *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 4, pp. 40.1–35. ACM Press, New York, NY, USA.

WFMC 2008. *Process definition interface – XML process definition language*, v. 2.1a. Standard WFMC-TC-1025, The Workflow Management Coalition (WFMC), Hingham, MA, USA.

Wolter, C. and Meinel, C. 2010. An approach to capture authorisation requirements in business processes. *Requirements Engineering*, vol. 15, no. 4, pp. 359–373. Springer, Berlin, Germany.

Wolter, C., Miseldine, P., and Meinel, C. 2009. Verification of business process entailment constraints using SPIN. In *Proceedings of the First International Symposium on Engineering Secure Software and Systems (ESSoS '09)*, pp. 1–15. Springer, Berlin, Germany.

Wolter, C. and Schaad, A. 2007. Modeling of task-based authorization constraints in BPMN. In *Proceedings of the 5th International Conference on Business Process Management (BPM '07)*, pp. 64–79. Springer, Berlin, Germany.

Wolter, C., Schaad, A., and Meinel, C. 2007. Deriving XACML policies from business process models. In *Proceedings of the 2007 International Workshop on Web Information Systems Engineering (WISE '07)*. pp. 142–153. Springer, Berlin, Germany.

Wolter, C., Schaad, A., and Meinel, C. 2008. Task-based entailment constraints for basic workflow patterns. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT '08)*, pp. 51–60. ACM Press, New York, NY, USA.

Wong, P. Y. H. and Gibbons, J. 2008. A process semantics for BPMN. In *Proceedings of the 10th International Conference on Formal Engineering Methods (ICFEM '08)*, pp. 355–374. Springer, Berlin, Germany.

YAWL 2011. *YAWL user manual*. The YAWL Foundation, `www.yawlfoundation.org`.

Zhang, I. X. 2007. Economic consequences of the Sarbanes-Oxley Act of 2002. *Journal of Accounting and Economics*, vol. 44, no. 1–2, pp. 74–115. Elsevier, Amsterdam, The Netherlands.

Zhang, L., Brodsky, A., and Jajodia, S. 2006. Toward information sharing: Benefit and risk access control (BARAC). In *In Proceedings of the 7th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '06)*, pp. 45–53. IEEE Computer Society Press, Los Alamitos, CA, USA.

Curriculum Vitae

Personal Information

Name	Samuel Jakob Burri
Date of birth	12 October 1981
Place of origin	Root (LU) and Malters (LU)
Nationality	Swiss

Education

April 2008 – May 2012	Doctoral studies at ETH Zurich, Switzerland
October 2002 – January 2008	Studies in computer science at ETH Zurich, Switzerland (MSc ETH CS)
February 2006 – June 2006	Exchange semester at TU Delft, The Netherlands
August 1996 – January 2001	Kantonsschule Romanshorn, Switzerland

Professional Experience

April 2008 – May 2012	Predoctoral researcher at IBM Research – Zurich, Switzerland
October 2006 – April 2007	Intern at Siemens R&D, Beijing, China
June 2002 – March 2008	Software engineer (part-time) at panta rhei pr, Amriswil, Switzerland
June 2002 – January 2006	Software engineer (part-time) at apfelX MedienDesign, Konstanz, Germany
October 2001 – May 2002	Web designer at euro style advertising, San Diego, CA, USA

Appendix A

Proofs

When proving the equivalence of two statements, we may refer to the left-hand side as LHS, to the right-hand side as RHS, and make a case distinction between $LHS \Rightarrow RHS$ and $LHS \Leftarrow RHS$.

A.1 Proof of Lemma 2.2

Proof. Let $G = (V,E)$ be a graph, with $|V| \geq 1$, and $\mathsf{L} : V \to 2^Z$ a color-list function, for a set Z. We refer to a line i of Algorithm 1 as LCi. We prove by induction over V first the soundness property and afterward the completeness property.

Soundness: For the base case, let $V = \{v\}$. Furthermore, let $\mathsf{f} = \{(v,z)\} = \textsc{Col}(\{v\},E,\mathsf{L})$, for $z \in Z$. Because $|\mathsf{f}| = 1$, Algorithm 1 returns at LC4. Therefore, $|\mathsf{L}(v)| \geq 1$ by LC2 and it follows by LC3 that $\mathsf{f}(v) \in \mathsf{L}(v)$. Because $|V| = 1$, we have $E = \varnothing$ by the definition of a graph. Hence, f is an L-coloring of G.

For the step case, let $|V| \geq 2$ and $v \in V$. Furthermore, let $G' = (V',E')$, for $V' = V \setminus \{v\}$, $E' = \{e \in E \mid v \notin e\}$, and $\mathsf{L}' : V' \to 2^Z$. Induction hypothesis: If $\textsc{Col}(V',E',\mathsf{L}')$ returns a function f', then f' is an L'-coloring for G'. Let $\mathsf{f} = \textsc{Col}(V,E,\mathsf{L})$. Because $|V| \geq 2$, Algorithm 1 returns at LC15. Let $\mathsf{f} = r \cup \{(v,z)\}$. By LC13, LC14, and the induction hypothesis, r is an L'-coloring for $G' = (V',E')$, for V', E', and L' as defined in LC10, LC11, and LC12. Therefore, $\mathsf{f}(v') \in \mathsf{L}'(v')$ for all $v' \in V'$ and $\mathsf{f}(v_1) \neq \mathsf{f}(v_2)$ for all $\{v_1,v_2\} \in E'$. Let $E'' = E \setminus E'$. It follows by LC11 that for every $\{v_1,v_2\} \in E''$ either $v_1 = v$ or $v_2 = v$. Without loss of generality let $v_1 = v$. By LC10, $v_2 \in V'$. By LC12, $\mathsf{f}(v_2) \neq z$ and therefore $\mathsf{f}(v_1) \neq \mathsf{f}(v_2)$. Furthermore, $\mathsf{f}(v) \in \mathsf{L}(v)$ by LC9. Hence, f is an L-coloring of G and the soundness property of Lemma 2.2 follows.

Completeness: Let col_L be an L-coloring for G. For the base case, let $V = \{v\}$. By the definition of an L-coloring, $\mathsf{col}_\mathsf{L}(v) \in \mathsf{L}(v)$ and therefore, $|\mathsf{L}(v)| \geq 1$. It follows from LC1 and LC2 that $\textsc{Col}(\{v\},E,\mathsf{L})$ returns a function at LC4.

For the step case, let $|V| \geq 2$ and $v \in V$. Furthermore, let $G' = (V', E')$, for $V' = V \setminus \{v\}$ and $E' = \{e \in E \mid v \notin e\}$, and let $\mathsf{L}' : V' \to 2^{\mathbb{Z}}$. Induction hypothesis: If there exists an L'-coloring for G', then $\textsc{LCol}(V', E', \mathsf{L}')$ returns a function. Because $|V| \geq 2$, $\textsc{LCol}(V, E, L)$ passes through LC8; let v be the vertex chosen in LC8 and $z = \mathsf{col}_{\mathsf{L}}(v)$. Algorithm 1 cannot return No before z is chosen in LC9. Let V', E', and L' as defined in LC10, LC11, and LC12. The coloring $\mathsf{col}_{\mathsf{L}'} = \mathsf{col}_{\mathsf{L}} \setminus \{(v, z)\}$ is an L'-coloring for (V', E'). Therefore, $\textsc{LCol}(V', E', \mathsf{L}')$ returns a function f' at LC13 by the induction hypothesis. Hence, $\textsc{LCol}(V, E, \mathsf{L})$ returns the function $\mathsf{f}' \cup \{(v, c)\}$ at LC15 and the completeness property of Lemma 2.2 follows. ■

A.2 Proof of Lemma 5.1

We refer to rule i of Definition 5.2 as MUi and to rule j of Definition B.1 as SEj. We first establish two auxiliary propositions and then prove Lemma 5.1.

Proposition A.1 *Under* $\textsc{SoDA}^{\mathsf{M}}$*, unit terms are only satisfied by multisets of users that contain exactly one element.*

Proof. The only rules of Definition 5.2 that allow for the derivation of a unit term are $MU1$–$MU4$ and $MU7$–$MU9$. The rules $MU1$–$MU4$ have a conclusion with a multiset that contains exactly one user. The remaining rules, $MU7$–$MU9$, have only multisets in their conclusions that are also in their premises. Hence, every unit term is only satisfied by a multiset of users that contains one element. ■

Proposition A.2 *For all unit terms ϕ_{ut}, all user-role assignments UR, and all users $u \in \mathcal{U}$, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut}$ if and only if $\{u\} \models^{\mathsf{S}}_{\mathsf{lwconf}(UR)} \phi_{ut}$.*

Proof. We reason by induction on the structure of ϕ_{ut}. The only rules of Definition 5.2 that allow for the derivation of a unit term are $MU1$–$MU4$ and $MU7$–$MU9$. All other rules can be safely ignored. Let a user-role assignment UR and a user u be given. Furthermore, let $(U, UR) = \mathsf{lwconf}(UR)$.

$LHS \Rightarrow RHS$:

Base Cases: Consider the term All and let $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \mathsf{All}$. By $MU1$, there exists an $r \in \mathcal{R}$ such that $(u, r) \in UR$. Therefore, $u \in U$ by the definition of lwconf. From $SE1$ it follows that $\{u\} \models^{\mathsf{S}}_{(U,UR)} \mathsf{All}$.

Consider a term of the form r, for $r \in \mathcal{R}$, and let $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} r$. From $MU2$ it follows that $(u, r) \in UR$ and therefore, $\{u\} \models^{\mathsf{S}}_{(U,UR)} r$ by $SE2$.

Consider a term of the form S, for $S \subseteq \mathcal{U}$, and let $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} S$. By $MU3$, $u \in S$ and there exists an $r \in \mathcal{R}$ such that $(u,r) \in UR$. By the definition of lwconf, $u \in U$ and therefore $u \in U \cap S$. From $SE3$ it follows that $\{u\} \models^{\mathsf{S}}_{(U,UR)} S$.

Step Cases: Assume that Proposition A.2 holds for two unit terms ϕ_{ut} and ψ_{ut}. Consider now the term $\neg\phi_{ut}$ and let $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \neg\phi_{ut}$. By $MU4$, $\{u\} \not\models^{\mathsf{M}}_{UR} \phi_{ut}$. From the induction hypothesis, it follows that $\{u\} \not\models^{\mathsf{S}}_{(U,UR)} \phi_{ut}$. Therefore, $\{u\} \models^{\mathsf{S}}_{(U,UR)} \neg\phi_{ut}$ by $SE4$.

Consider the term $\phi_{ut} \sqcup \psi_{ut}$ and let $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut} \sqcup \psi_{ut}$. By $MU7$ and $MU8$, either $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut}$ or $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \psi_{ut}$. In the first case, by the induction hypothesis, $\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi_{ut}$ and therefore $\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi_{ut} \sqcup \psi_{ut}$ by $SE7$. The second case is analogous. Hence, $\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi_{ut} \sqcup \psi_{ut}$.

Consider the term $\phi_{ut} \sqcap \psi_{ut}$ and let $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut} \sqcap \psi_{ut}$. By $MU9$, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut}$ and $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \psi_{ut}$. By the induction hypothesis, $\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi_{ut}$ and $\{u\} \models^{\mathsf{S}}_{(U,UR)} \psi_{ut}$. Therefore, $\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi_{ut} \sqcap \psi_{ut}$ by $SE9$.

$LHS \Leftarrow RHS$:

Base Cases: Consider the term All and let $\{u\} \models^{\mathsf{S}}_{(U,UR)} \mathsf{All}$. By $SE1$, $u \in U$ and therefore, there exists an $r \in \mathcal{R}$ such that $(u,r) \in UR$, by the definition of lwconf. From $MU1$ it follows that $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \mathsf{All}$.

Consider a term of the form r, for $r \in \mathcal{R}$, and let $\{u\} \models^{\mathsf{S}}_{(U,UR)} r$. From $SE2$ it follows that $(u,r) \in UR$ and therefore $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} r$ by $MU2$.

Consider a term of the form S, for $S \subseteq \mathcal{U}$, and let $\{u\} \models^{\mathsf{S}}_{(U,UR)} S$. By $SE3$, $u \in U \cap S$ and therefore, $u \in U$ and $u \in S$. From the definition of lwconf, it follows that there exists an $r \in \mathcal{R}$ such that $(u,r) \in UR$. By $MU3$, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} S$.

Step Cases: Assume that Proposition A.2 holds for two unit terms ϕ_{ut} and ψ_{ut}. Consider now the term $\neg\phi_{ut}$ and let $\{u\} \models^{\mathsf{S}}_{(U,UR)} \neg\phi_{ut}$. By $SE4$, $\{u\} \not\models^{\mathsf{S}}_{(U,UR)} \phi_{ut}$. From the induction hypothesis, it follows that $\{u\} \not\models^{\mathsf{M}}_{UR} \phi_{ut}$. Therefore, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \neg\phi_{ut}$ by $MU4$.

Consider the term $\phi_{ut} \sqcup \psi_{ut}$ and let $\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi_{ut} \sqcup \psi_{ut}$. By $SE7$ and $SE8$, either $\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi_{ut}$ or $\{u\} \models^{\mathsf{S}}_{(U,UR)} \psi_{ut}$. In the first case, by the induction hypothesis, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut}$ and therefore $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut} \sqcup \psi_{ut}$ by $MU7$. The second case is analogous. Hence, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut} \sqcup \psi_{ut}$.

Consider the term $\phi_{ut} \sqcap \psi_{ut}$ and let $\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi_{ut} \sqcap \psi_{ut}$. By $SE9$, $\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi_{ut}$ and $\{u\} \models^{\mathsf{S}}_{(U,UR)} \psi_{ut}$. By the induction hypothesis, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut}$ and $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \psi_{ut}$. Therefore, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut} \sqcap \psi_{ut}$ by $MU9$.

■

Proof of Lemma 5.1. Let UR be a user-role assignment and $\mathbf{U}$ a multiset of users. Furthermore, let $(U, UR) = \mathsf{lwconf}(UR)$. We reason inductively over the structure of SoDA terms.

Base Case: Consider a unit term ϕ_{ut} and let $\mathbf{U} \models^{\mathsf{M}}_{UR} \phi_{ut}$. By Proposition A.1, $\mathbf{U} = \{\!\{u\}\!\}$, for a user $u \in \mathcal{U}$. From Proposition A.2 it follows that $\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi_{ut}$. By the definition of userset, $\{u\} = \mathsf{userset}(\mathbf{U})$ and therefore $\mathsf{userset}(\mathbf{U}) \models^{\mathsf{S}}_{(U,UR)} \phi_{ut}$.

Step Cases: Assume that Lemma 5.1 holds for two terms ϕ and ψ. Consider now the term ϕ^+, let $\mathbf{U} \models^{\mathsf{M}}_{UR} \phi^+$ and $Y = \mathsf{userset}(\mathbf{U})$. From $MU5$ and $MU6$ follows that for every user $u \in \mathbf{U}$, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi$. By the induction hypothesis and the definition of userset follows that for every user $u \in Y$, $\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi$. From $SE5$ and $SE6$ it follows that $Y \models^{\mathsf{S}}_{(U,UR)} \phi^+$.

Consider the term $\phi \sqcup \psi$ and let $\mathbf{U} \models^{\mathsf{M}}_{UR} \phi \sqcup \psi$. By $MU7$ and $MU8$, either $\mathbf{U} \models^{\mathsf{M}}_{UR} \phi$ or $\mathbf{U} \models^{\mathsf{M}}_{UR} \psi$. In the first case, by the induction hypothesis, $\mathsf{userset}(\mathbf{U}) \models^{\mathsf{S}}_{(U,UR)} \phi$ and therefore $\mathsf{userset}(\mathbf{U}) \models^{\mathsf{S}}_{(U,UR)} \phi \sqcup \psi$ by $SE7$. The second case is analogous. Hence, $\mathsf{userset}(\mathbf{U}) \models^{\mathsf{S}}_{(U,UR)} \phi \sqcup \psi$.

Consider the term $\phi \sqcap \psi$ and let $\mathbf{U} \models^{\mathsf{M}}_{UR} \phi \sqcap \psi$. By $MU9$, $\mathbf{U} \models^{\mathsf{M}}_{UR} \phi$ and $\mathbf{U} \models^{\mathsf{M}}_{UR} \psi$. By the induction hypothesis, $\mathsf{userset}(\mathbf{U}) \models^{\mathsf{S}}_{(U,UR)} \phi$ and $\mathsf{userset}(\mathbf{U}) \models^{\mathsf{S}}_{(U,UR)} \psi$. Therefore, $\mathsf{userset}(\mathbf{U}) \models^{\mathsf{S}}_{(U,UR)} \phi \sqcap \psi$ by $SE9$.

Consider the term $\phi \odot \psi$ and let $\mathbf{U} \models^{\mathsf{M}}_{UR} \phi \odot \psi$. By $MU10$, there are two multisets of users $\mathbf{V}$ and $\mathbf{W}$ such that $\mathbf{V} \models^{\mathsf{M}}_{UR} \phi$ and $\mathbf{W} \models^{\mathsf{M}}_{UR} \psi$. By the induction hypothesis, $\mathsf{userset}(\mathbf{V}) \models^{\mathsf{S}}_{(U,UR)} \phi$ and $\mathsf{userset}(\mathbf{W}) \models^{\mathsf{S}}_{(U,UR)} \psi$. By the definition of userset, $\mathsf{userset}(\mathbf{U}) = \mathsf{userset}(\mathbf{V}) \cup \mathsf{userset}(\mathbf{W})$. From $SE10$ it follows that $\mathsf{userset}(\mathbf{U}) \models^{\mathsf{S}}_{(U,UR)} \psi \odot \psi$.

Consider the term $\phi \otimes \psi$ and let $\mathbf{U} \models^{\mathsf{M}}_{UR} \phi \otimes \psi$. By $MU11$, there are two multisets of users $\mathbf{V}$ and $\mathbf{W}$ such that $\mathbf{V} \models^{\mathsf{M}}_{UR} \phi$, $\mathbf{W} \models^{\mathsf{M}}_{UR} \psi$, and $\mathbf{V} \cap \mathbf{W} = \varnothing$. By the induction hypothesis, $\mathsf{userset}(\mathbf{V}) \models^{\mathsf{S}}_{(U,UR)} \phi$ and $\mathsf{userset}(\mathbf{W}) \models^{\mathsf{S}}_{(U,UR)} \psi$. By the definition of userset, $\mathsf{userset}(\mathbf{U}) = \mathsf{userset}(\mathbf{V}) \cup \mathsf{userset}(\mathbf{W})$. Furthermore, if $\mathbf{V}$ and $\mathbf{W}$ are disjoint, then $\mathsf{userset}(\mathbf{V})$ and $\mathsf{userset}(\mathbf{W})$ are disjoint too. Therefore, by $SE11$ $\mathsf{userset}(\mathbf{U}) \models^{\mathsf{S}}_{(U,UR)} \psi \otimes \psi$. ■

A.3 Proof of Lemma 5.2

We refer to rule i of Definition 5.2 as MUi and to rule j of Definition 5.5 as TRj. We first establish two auxiliary propositions and then prove Lemma 5.2.

Proposition A.3 *For* $i, i_1, i_2 \in \Sigma^*$, *if* $\mathsf{si}(i, i_1, i_2)$ *then* $\mathsf{users}(i) = \mathsf{users}(i_1) \uplus \mathsf{users}(i_2)$.

Proof. By Definition 5.4, each execution event in i is either in i_1 or i_2, but not in both. Therefore, $\mathsf{users}(i) = \mathsf{users}(i_1) \uplus \mathsf{users}(i_2)$ since the function users returns the multiset of users that are contained in the business events of its argument. ■

Proposition A.4 *For $i \in \mathcal{X}^*$, $i_1, i_2 \in \Sigma^*$, if $\mathsf{si}(i, i_1, i_2)$ then $i_1 \in \mathcal{X}^*$ and $i_2 \in \mathcal{X}^*$.*

Proof. By Definition 5.4, each event that is in i_1 or i_2 is also in i. Since $i \in \mathcal{X}^*$, we therefore have that i_1 and i_2 contain only execution events. ■

Proof of Lemma 5.2. Let UR be a user-role assignment. We reason inductively over the structure of SoDA terms.

Base Cases: Consider a unit term ϕ_{ut} and a trace $i \in \mathcal{X}^*$. Let $i \models^{\mathsf{T}}_{UR} \phi_{ut}$. The only rules of Definition 5.5 that may have a unit term in their conclusion are *TR*1–*TR*4 and *TR*7–*TR*9. Because i contains no admin events, only *TR*1 and *TR*7–*TR*9 are applicable. In the case of *TR*7–*TR*9 i is already contained in at least one premise. In a derivation tree for $i \models^{\mathsf{T}}_{UR} \phi_{ut}$, i must therefore be in the conclusion of rule *TR*1 by the structure of terms, *i.e.* Definition 5.1. Therefore, i must be of the form $\langle t.u \rangle$, for an execution event $t.u$. By *TR*1, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR} \phi_{ut}$, which is equivalent to $\mathsf{users}(i) \models^{\mathsf{M}}_{UR} \phi_{ut}$ by the definition of users.

Consider a term of the form ϕ^+_{ut} and a trace $i \in \mathcal{X}^*$. Let $i \models^{\mathsf{T}}_{UR} \phi^+_{ut}$. The only rules of Definition 5.5 that have a term of the form ϕ^+_{ut} in the conclusion are *TR*5 and *TR*6. For both rules, the trace i must contain at least one execution event x such that $\langle x \rangle \models^{\mathsf{T}}_{UR} \phi_{ut}$. As derived before, $\mathsf{users}(\langle x \rangle) \models^{\mathsf{M}}_{UR} \phi_{ut}$ and therefore, by *MU*5, $\mathsf{users}(\langle x \rangle) \models^{\mathsf{M}}_{UR} \phi^+_{ut}$. By induction over the length of i, with $\langle x \rangle$ as the induction basis, it follows that $\mathsf{users}(i) \models^{\mathsf{M}}_{UR} \phi^+_{ut}$ from *TR*6 and *MU*6.

Step Cases: Assume that Lemma 5.2 holds for two terms ϕ and ψ. Consider now the term $\phi \sqcup \psi$ and a trace $i \in \mathcal{X}^*$. Let $i \models^{\mathsf{T}}_{UR} \phi \sqcup \psi$. By *TR*7 and *TR*8, either $i \models^{\mathsf{T}}_{UR} \phi$ or $i \models^{\mathsf{T}}_{UR} \psi$. In the first case, by the induction hypothesis, $\mathsf{users}(i) \models^{\mathsf{M}}_{UR} \phi$ and therefore $\mathsf{users}(i) \models^{\mathsf{M}}_{UR} \phi \sqcup \psi$ by *MU*7. The second case is analogous. Hence, $\mathsf{users}(i) \models^{\mathsf{M}}_{UR} \phi \sqcup \psi$.

Consider the term $\phi \sqcap \psi$ and a trace $i \in \mathcal{X}^*$. Let $i \models^{\mathsf{T}}_{UR} \phi \sqcap \psi$. By *TR*9, $i \models^{\mathsf{T}}_{UR} \phi$ and $i \models^{\mathsf{T}}_{UR} \psi$. From the induction hypothesis, $\mathsf{users}(i) \models^{\mathsf{M}}_{UR} \phi$ and $\mathsf{users}(i) \models^{\mathsf{M}}_{UR} \psi$. Therefore, $\mathsf{users}(i) \models^{\mathsf{M}}_{UR} \phi \sqcap \psi$ by *MU*9.

Consider the term $\phi \odot \psi$ and a trace $i \in \mathcal{X}^*$. Let $i \models^{\mathsf{T}}_{UR} \phi \odot \psi$. By *TR*10, there exist two traces i_1 and i_2 such that $\mathsf{si}(i, i_1, i_2)$, $i_1 \models^{\mathsf{T}}_{UR} \phi$, and $i_2 \models^{\mathsf{T}}_{UR} \psi$. By Proposition A.4, i_1 and i_2 consist only of admin events because $i \in \mathcal{X}^*$. Therefore, from the induction hypothesis, $\mathsf{users}(i_1) \models^{\mathsf{M}}_{UR} \phi$ and $\mathsf{users}(i_2) \models^{\mathsf{M}}_{UR} \psi$. Moreover, by Proposition A.3, $\mathsf{users}(i) = \mathsf{users}(i_1) \uplus \mathsf{users}(i_2)$. Hence, $\mathsf{users}(i) \models^{\mathsf{M}}_{UR} \phi \odot \psi$ by *MU*10.

Finally, consider the term $\phi \otimes \psi$ and a trace $i \in \mathcal{X}^*$. Let $i \models^{\mathsf{T}}_{UR} \phi \otimes \psi$. By *TR*11, there exist two traces i_1 and i_2 such that $\mathsf{si}(i, i_1, i_2)$, $\mathsf{users}(i_1) \cap \mathsf{users}(i_2) = \varnothing$, $i_1 \models^{\mathsf{T}}_{UR} \phi$, and $i_2 \models^{\mathsf{T}}_{UR} \psi$. By Proposition A.4, i_1 and i_2 consist only of admin events because $i \in \mathcal{X}^*$. Therefore, from the induction hypothesis, $\mathsf{users}(i_1) \models^{\mathsf{M}}_{UR} \phi$ and $\mathsf{users}(i_2) \models^{\mathsf{M}}_{UR} \psi$. Furthermore, by Proposition A.3, $\mathsf{users}(i) = \mathsf{users}(i_1) \uplus \mathsf{users}(i_2)$. Hence, $\mathsf{users}(i) \models^{\mathsf{M}}_{UR} \phi \otimes \psi$ by *MU*11. ■

A.4 Proof of Theorem 5.1

We refer to rule i of Definition 5.5 as *TR*i and to rule j of Definition 5.6 as *MA*j. We first establish four auxiliary propositions and then prove Theorem 5.1. Recall Definition 5.8. We prove that for all terms ϕ, all user-role assignments UR, and all traces $i \in \Sigma^*$, $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SODA_\phi(UR))$ if and only if $i \models^{\mathsf{T}}_{UR} \phi$.

Proposition A.5 *For a term ϕ, a trace $i \in \Sigma^*$, a trace of admin events $a \in \mathcal{A}^*$, a user-role assignment UR, and $UR' = \mathsf{upd}(UR, a)$, $i \models^{\mathsf{T}}_{UR'} \phi$ if and only if $a \hat{} i \models^{\mathsf{T}}_{UR} \phi$.*

Proof Sketch. Proposition A.5 follows by induction on a directly from *TR*3, *TR*4, and Definition 5.3. □

Proposition A.6 *For a user-role assignment UR, a term ϕ, a trace $i \in \Sigma^*$, and a trace of admin events $a \in \mathcal{A}^*$, $i \models^{\mathsf{T}}_{UR} \phi$ if and only if $i \hat{} a \models^{\mathsf{T}}_{UR} \phi$.*

Proof Sketch. Proposition A.6 follows directly by applying *TR*2 for each admin event in a to $i \models^{\mathsf{T}}_{UR} \phi$. □

Proposition A.7 *For a user-role assignment UR, a term ϕ, and a set of users U, the process $[\phi]^U_{UR}$ engages only in an execution event $t.u$, for a task t and a user u, if $u \in U$.*

Proof. Let UR be a user-role assignment, ϕ a term, U a set of users, and $t.u$ an execution event, for a task t and a user u. We reason inductively on the structure of ϕ. Terms of the form ϕ_{ut} and ϕ^+_{ut} are the base cases. By *MA*1 and *MA*2, $[\phi]^U_{UR}$ and $[\phi^+]^U_{UR}$ only engage in $t.u$, if $u \in U$. For two terms ϕ and ψ, assume that Proposition A.7 holds. By *MA*3, *MA*4, and *MA*5, the processes $[\phi \sqcup \psi]^U_{UR}$, $[\phi \sqcap \psi]^U_{UR}$, and $[\phi \odot \psi]^U_{UR}$ only engage in $t.u$ if either $[\phi]^U_{UR}$ or $[\psi]^U_{UR}$ engage in $t.u$. By *MA*6, the process $[\phi \otimes \psi]^U_{UR}$ only engages in $t.u$ if either $[\phi]^{U'}_{UR}$ or $[\psi]^{U''}_{UR}$ engage in $t.u$, for $U', U'' \subseteq U$. From the induction hypothesis, it follows that all processes only engage in $t.u$ if $u \in U$. ■

Proposition A.8 *For the traces $i, i_1, i_2 \in \Sigma^*$ and the processes $P, Q \in \mathcal{P}$, if $i_1 \in \mathsf{T}(P)$, $i_2 \in \mathsf{T}(Q)$ and $\mathsf{si}(i, i_1, i_2)$, then $i \in \mathsf{T}(P \underset{\mathcal{A}}{\parallel} Q)$.*

Proof Sketch. The proof is by induction over i. Proposition A.8 follows by the definition of the $||$-operator under the denotational semantics of CSP and by Definition 5.4. The implicit synchronization on ✓ can be ignored because i, i_1, and i_2 do not contain ✓. □

Proof of Theorem 5.1. We prove that for all terms ϕ, all user-role assignments UR, and all traces $i \in \Sigma^*$, $LHS \Rightarrow RHS$ and $LHS \Leftarrow RHS$. In both cases we reason by induction on the structure of ϕ. Let UR be given.

$LHS \Rightarrow RHS$:

Base Cases: Consider a unit term ϕ_{ut} and let $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SODA_{\phi_{ut}}(UR))$. By $MA1$ and the denotational semantics of CSP, i is of the form $a_1 \hat{} \langle t.u \rangle \hat{} a_2$, for $a_1, a_2 \in \mathcal{A}^*$, a task t, and a user u. Let $UR' = \mathsf{upd}(UR, a_1)$. Because $[\phi_{ut}]^{\mathcal{U}}_{UR'}$ engages in $t.u$, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR'} \phi_{ut}$ by $MA1$. From $TR1$ it follows that $\langle t.u \rangle \models^{\mathsf{T}}_{UR'} \phi_{ut}$. Therefore, by Proposition A.5, $a_1 \hat{} \langle t.u \rangle \models^{\mathsf{T}}_{UR} \phi_{ut}$ and by Proposition A.6 $a_1 \hat{} \langle t.u \rangle \hat{} a_2 \models^{\mathsf{T}}_{UR} \phi_{ut}$. Hence, $i \models^{\mathsf{T}}_{UR} \phi_{ut}$.

Consider a term of the form ϕ^+_{ut} and let $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SODA_{\phi^+_{ut}}(UR))$. By $MA2$ and the denotational semantics of CSP, i is of the form $a_1 \hat{} \langle t_1.u_1 \rangle \hat{} \ldots \hat{} a_n \hat{} \langle t_n.u_n \rangle \hat{} a_{n+1}$, for $a_i \in \mathcal{A}^*$, $a_{n+1} \in \mathcal{A}^*$, $t_i \in \mathcal{T}$, $u_i \in \mathcal{U}$, $i \in \{1 \ldots n\}$, and $n \in \mathbb{N}$. We reason inductively over n. Assume $n = 1$ and let $UR' = \mathsf{upd}(UR, a_1)$. Analogous to the previous case, it follows that $a_1 \hat{} \langle t_1.u_1 \rangle \hat{} a_2 \models^{\mathsf{T}}_{UR} \phi_{ut}$. By $TR5$, $a_1 \hat{} \langle t_1.u_1 \rangle \hat{} a_2 \models^{\mathsf{T}}_{UR} \phi^+_{ut}$. We now assume $n > 1$ and $a_2 \hat{} \langle t_2.u_2 \rangle \hat{} a_3 \hat{} \ldots \hat{} a_n \hat{} \langle t_n.u_n \rangle \hat{} a_{n+1} \models^{\mathsf{T}}_{UR'} \phi^+_{ut}$, for $UR' = \mathsf{upd}(UR, a_1)$. Because $[\phi_{ut}]^{\mathcal{U}}_{UR'}$ engages in $t_1.u_1$, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR'} \phi_{ut}$ by $MA2$. From $TR1$ it follows that $\langle t_1.u_1 \rangle \models^{\mathsf{T}}_{UR'} \phi_{ut}$ and from $TR6$ that $\langle t_1.u_1 \rangle \hat{} a_2 \hat{} \langle t_2.u_2 \rangle \hat{} a_3 \hat{} \ldots \hat{} a_n \hat{} \langle t_n.u_n \rangle \hat{} a_{n+1} \models^{\mathsf{T}}_{UR'} \phi^+_{ut}$. By Proposition A.5, $a_1 \hat{} \langle t_1.u_1 \rangle \hat{} a_2 \hat{} \langle t_2.u_2 \rangle \hat{} a_3 \hat{} \ldots \hat{} a_n \hat{} \langle t_n.u_n \rangle \hat{} a_{n+1} \models^{\mathsf{T}}_{UR} \phi^+_{ut}$ and hence $i \models^{\mathsf{T}}_{UR} \phi^+_{ut}$.

Step Cases: For two terms ϕ and ψ, assume that $LHS \Rightarrow RHS$ holds. Consider now the term $\phi \sqcup \psi$ and let $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SODA_{\phi \sqcup \psi}(UR))$. By $MA3$, $SODA_{\phi \sqcup \psi}(UR) = SODA_\phi(UR) \,\Box\, SODA_\psi(UR)$. From the denotational semantics of CSP it follows that either $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SODA_\phi(UR))$ or $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SODA_\psi(UR))$. Consider the first case. From the induction hypothesis $i \models^{\mathsf{T}}_{UR} \phi$. By $TR7$, $i \models^{\mathsf{T}}_{UR} \phi \sqcup \psi$. The second case follows analogously by $TR8$. Hence, $i \models^{\mathsf{T}}_{UR} \phi \sqcup \psi$.

Consider the term $\phi \sqcap \psi$ and let $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SODA_{\phi \sqcap \psi}(UR))$. By $MA4$, $SODA_{\phi \sqcap \psi}(UR) = SODA_\phi(UR) \parallel SODA_\psi(UR)$. From the denotational semantics of CSP it follows that $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SODA_\phi(UR))$ and $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SODA_\psi(UR))$. By the induction hypothesis, $i \models^{\mathsf{T}}_{UR} \phi$ and $i \models^{\mathsf{T}}_{UR} \psi$. Therefore, $i \models^{\mathsf{T}}_{UR} \phi \sqcap \psi$ follows by $TR9$.

Consider the term $\phi \odot \psi$ and let $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SODA_{\phi \odot \psi}(UR))$. By $MA5$, $SODA_{\phi \odot \psi}(UR) = SODA_\phi(UR) \underset{\mathcal{A}}{\parallel} SODA_\psi(UR)$. From the denotational seman-

tics of CSP it follows that there are two traces $i_\phi, i_\psi \in \Sigma^*$ such that $i_\phi \hat{} \langle\checkmark\rangle \in \mathsf{T}(SODA_\phi(UR))$ and $i_\psi \hat{} \langle\checkmark\rangle \in \mathsf{T}(SODA_\psi(UR))$. From the induction hypothesis, it follows that $i_\phi \models^{\mathsf{T}}_{UR} \phi$ and $i_\psi \models^{\mathsf{T}}_{UR} \psi$. Moreover, because $SODA_\phi(UR)$ and $SODA_\psi(UR)$ synchronize on $\mathcal{A}$ but not on $\mathcal{X}$, $\mathsf{si}(i, i_\phi, i_\psi)$. By *TR*10, $i \models^{\mathsf{T}}_{UR} \phi \odot \psi$.

Finally, consider the term $\phi \otimes \psi$ and let $i \hat{} \langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi\otimes\psi}(UR))$. By *MA*6, $SODA_{\phi\otimes\psi}(UR) = ([\phi]^{U_\phi}_{UR} \underset{\mathcal{A}}{\parallel} [\psi]^{U_\psi}_{UR}) \,\square\, \ldots$. From *MA*6 and the denotational semantics of CSP it follows that there are two disjoint sets of users U_ϕ and U_ψ such that $i \hat{} \langle\checkmark\rangle \in \mathsf{T}([\phi]^{U_\phi}_{UR} \underset{\mathcal{A}}{\parallel} [\psi]^{U_\psi}_{UR})$. Analogous to the previous case, there are two traces $i_\phi, i_\psi \in \Sigma^*$ such that $i_\phi \models^{\mathsf{T}}_{UR} \phi$, $i_\psi \models^{\mathsf{T}}_{UR} \psi$, and $\mathsf{si}(i, i_\phi, i_\psi)$. By Proposition A.7, users in $\mathsf{users}(i_\phi)$ are in U_ϕ and users in $\mathsf{users}(i_\psi)$ are in U_ψ. Because $U_\phi \cap U_\psi = \varnothing$, it follows that $\mathsf{users}(i_\phi) \cap \mathsf{users}(i_\psi) = \varnothing$. Therefore, by *TR*11, $i \models^{\mathsf{T}}_{UR} \phi \otimes \psi$.

LHS $\Leftarrow$ *RHS:*

Base Cases: Consider a unit term ϕ_{ut} and let i be a trace such that $i \models^{\mathsf{T}}_{UR} \phi_{ut}$. The only rules of Definition 5.5 that allow for the derivation of ϕ_{ut} are *TR*1–*TR*4 and *TR*7–*TR*9. We can safely ignore *TR*7–*TR*9 because all these rules do not change the trace that is derived. *I.e.* the same trace that is contained in the conclusion is already contained in a premise. Out of *TR*1–*TR*4, only *TR*1 does not have a trace in its premises. Therefore, *TR*1 is at the leaves of every derivation tree of $i \models^{\mathsf{T}}_{UR} \phi_{ut}$ and, thus, i contains an execution event $t.u$ for a task t and a user u. By iteratively applying the rules *TR*2–*TR*4, one can add admin events before and after $t.u$ but no additional execution event (otherwise ϕ_{ut} would not be a unit term). It follows that i is of the form $a_1 \hat{} \langle t.u \rangle \hat{} a_2$, for $a_1, a_2 \in \mathcal{A}^*$. Let $UR' = \mathsf{upd}(UR, a_1)$. From Proposition A.5 it follows that $\langle t.u \rangle \hat{} a_2 \models^{\mathsf{T}}_{UR'} \phi_{ut}$ and therefore $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR'} \phi_{ut}$ by *TR*1. By *MA*1, $SODA_{\phi_{ut}}(UR)$ accepts a_1 and behaves like $SODA_{\phi_{ut}}(UR')$ afterward. Because $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR'} \phi_{ut}$, $SODA_{\phi_{ut}}(UR')$ engages in $t.u$ and behaves like *END* afterward. From *END*'s definition, it follows that *END* accepts $a_2 \hat{} \langle\checkmark\rangle$. Hence, $i \hat{} \langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi_{ut}}(UR))$.

Consider a term ϕ^+_{ut} and let i be a trace such that $i \models^{\mathsf{T}}_{UR} \phi^+_{ut}$. The only rules of Definition 5.5 that allow for the derivation of ϕ^+_{ut} are *TR*2–*TR*6. Out of these, only *TR*5 does not have a trace that satisfies ϕ^+_{ut} in its premises. Therefore, every derivation of $i \models^{\mathsf{T}}_{UR} \phi^+_{ut}$ contains one application of *TR*5 and, thus, i contains at least one execution event $t.u$, for a task t and a user u. By *TR*2–*TR*4 and *TR*6, it follows that i is of the form $a_1 \hat{} \langle t_1.u_1 \rangle \hat{} \ldots \hat{} a_n \hat{} \langle t_n.u_n \rangle \hat{} a_{n+1}$ for $a_i \in \mathcal{A}^*$, $a_{n+1} \in \mathcal{A}^*$, $t_i \in \mathcal{T}$, $u_i \in \mathcal{U}$, $i \in \{1, \ldots, n\}$, and $n \in \mathbb{N}_0$. Because there is at least one execution event in i, $n \geq 1$. We reason inductively over n. For $n = 1$, it follows analogous to the unit term case that $a_1 \hat{} \langle t_1.u_1 \rangle \hat{} a_2 \hat{} \langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi^+_{ut}}(UR))$ by *MA*2. For $n > 1$, assume $a_2 \hat{} \langle t_2.u_2 \rangle \hat{} \ldots \hat{} a_n \hat{} \langle t_n.u_n \rangle \hat{} a_{n+1} \hat{} \langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi^+_{ut}}(UR'))$, for $UR' =$

$\mathsf{upd}(UR, a_1)$. Because $a_1\hat{}\langle t_1.u_1\rangle\hat{}\ldots\hat{}a_n\hat{}\langle t_n.u_n\rangle\hat{}a_{n+1} \models^{\mathsf{T}}_{UR} \phi^+_{ut}$, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR'} \phi_{ut}$ by *TR*1, *TR*6, and Proposition A.5. By *MA*2, $SODA_{\phi^+_{ut}}(UR)$ accepts a_1 and behaves like $SODA_{\phi^+_{ut}}(UR')$ afterward. Because $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UR'} \phi_{ut}$, $SODA_{\phi^+_{ut}}(UR')$ engages in $t_1.u_1$ and behaves like the external choice between $SODA_{\phi^+_{ut}}(UR')$ and END afterward. We can therefore decide that the process behaves like $SODA_{\phi^+_{ut}}(UR')$. Therefore, by the induction hypothesis, $a_1\hat{}\langle t_1.u_1\rangle\hat{}a_2\ \hat{}\langle t_2.u_2\rangle\hat{}\ldots\hat{}a_n\hat{}\langle t_n.u_n\rangle\hat{}a_{n+1}\ \hat{}\langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi^+_{ut}}(UR))$. Hence, $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi^+_{ut}}(UR))$.

Step Cases: For two terms ϕ and ψ, assume that $LHS \Leftarrow RHS$ holds. Consider now a term of the form $\phi \sqcup \psi$ and let $i \models^{\mathsf{T}}_{UR} \phi \sqcup \psi$. By *TR*7 and *TR*8, either $i \models^{\mathsf{T}}_{UR} \phi$ or $i \models^{\mathsf{T}}_{UR} \psi$. Consider the first case. By *MA*3 and the denotational semantics of CSP, $\mathsf{T}(SODA_{\phi\sqcup\psi}(UR)) = \mathsf{T}(SODA_{\phi}(UR)) \cup \mathsf{T}(SODA_{\psi}(UR))$. From the induction hypothesis it follows that $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi}(UR))$ and therefore, $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi\sqcup\psi}(UR))$. The second case is analogous. Hence, $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi\sqcup\psi}(UR))$.

Consider the term $\phi \sqcap \psi$ and let $i \models^{\mathsf{T}}_{UR} \phi \sqcap \psi$. By *TR*9, $i \models^{\mathsf{T}}_{UR} \phi$ and $i \models^{\mathsf{T}}_{UR} \psi$. By *MA*4 and the denotational semantics of CSP, $\mathsf{T}(SODA_{\phi\sqcap\psi}(UR)) = \mathsf{T}(SODA_{\phi}(UR)) \cap \mathsf{T}(SODA_{\psi}(UR))$. From the induction hypothesis it follows that $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi}(UR))$ and $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SODA_{\psi}(UR))$ and therefore, $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi\sqcap\psi}(UR))$.

Consider the term $\phi \odot \psi$ and let $i \models^{\mathsf{T}}_{UR} \phi \odot \psi$. By *TR*10, there exist two traces $i_\phi, i_\psi \in \Sigma^*$ such that $i_\phi \models^{\mathsf{T}}_{UR} \phi$, $i_\psi \models^{\mathsf{T}}_{UR} \psi$, and $\mathsf{si}(i, i_\phi, i_\psi)$. By the induction hypothesis, $i_\phi\ \hat{}\langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi}(UR))$ and $i_\psi\ \hat{}\langle\checkmark\rangle \in \mathsf{T}(SODA_{\psi}(UR))$, and therefore also $i_\phi \in \mathsf{T}(SODA_{\phi}(UR))$ and $i_\psi \in \mathsf{T}(SODA_{\psi}(UR))$. From *MA*5 and Proposition A.8 it follows that $i \in \mathsf{T}(SODA_{\phi\odot\psi}(UR))$. Moreover, by *MA*5, because $SODA_{\phi}(UR)$ and $SODA_{\psi}(UR)$ both engage in $\checkmark$ after having accepted i_ϕ and i_ψ respectively, $SODA_{\phi\odot\psi}(UR)$ engages in $\checkmark$ too, after having accepted i. Hence, $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi\odot\psi}(UR))$.

Finally, consider a term of the form $\phi \otimes \psi$ and let $i \models^{\mathsf{T}}_{UR} \phi \otimes \psi$. By *TR*11, there exist two traces i_ϕ and i_ψ such that $i_\phi \models^{\mathsf{T}}_{UR} \phi$, $i_\psi \models^{\mathsf{T}}_{UR} \psi$, $\mathsf{si}(i, i_\phi, i_\psi)$, and $\mathsf{users}(i_\phi) \cap \mathsf{users}(i_\psi) = \varnothing$. Because $\mathsf{users}(i_\phi) \cap \mathsf{users}(i_\psi) = \varnothing$, there exist two sets of users $U_\phi, U_\psi \subseteq \mathcal{U}$ such that $U_\phi \cup U_\psi = \mathcal{U}$, $U_\phi \cap U_\psi = \varnothing$, $\mathsf{userset}(\mathsf{users}(i_\phi)) \subseteq U_\phi$, and $\mathsf{userset}(\mathsf{users}(i_\psi)) \subseteq U_\psi$. By the induction hypothesis, $i_\phi\ \hat{}\langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi}(UR))$ and $i_\psi\ \hat{}\langle\checkmark\rangle \in \mathsf{T}(SODA_{\psi}(UR))$, and therefore also $i_\phi \in \mathsf{T}(SODA_{\phi}(UR))$ and $i_\psi \in \mathsf{T}(SODA_{\psi}(UR))$. From Proposition A.7, $[\phi]^{U_\phi}_{UR}$ therefore accepts i_ϕ and $[\psi]^{U_\psi}_{UR}$ accepts i_ψ. By *MA*6 and the denotational semantics of CSP, $SODA_{\phi\otimes\psi}(UR)$ also behaves like $[\phi]^{U_\phi}_{UR} \underset{\mathcal{A}}{\parallel} [\psi]^{U_\psi}_{UR}$. Analogous to the previous case, it follows that $i \in \mathsf{T}(SODA_{\phi\otimes\psi}(UR))$ and $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SODA_{\phi\otimes\psi}(UR))$. ■

A.5 Proof of Theorem 9.1

The proof of Theorem 9.1 requires a formal definition of the parallel, (fully-) synchronized composition of two processes in terms of the operational semantics of CSP, which is a standard parallel composition of two LTSs. Without loss of generality, we let the set of input symbols to an LTS that correspond to a process be the set of all events $\Sigma^{\tau,\checkmark}$.

Definition A.1 (Operational Semantics of the Parallel, Synchronized Process Composition) *Let P_1 and P_2 be two processes and let $L_{P_1} = (Q^{P_1}, \Sigma^{\tau,\checkmark}, \delta^{P_1}, q_0^{P_1})$ and $L_{P_2} = (Q^{P_2}, \Sigma^{\tau,\checkmark}, \delta^{P_2}, q_0^{P_2})$. An LTS $L_{P_1 \parallel P_2} = (Q^{P_{12}}, \Sigma^{\tau,\checkmark}, \delta^{P_{12}}, q_0^{P_{12}})$ corresponding to the process $P_1 \parallel P_2$ can be constructed as follows:*

- $Q^{P_{12}} = Q^{P_1} \times Q^{P_2}$
- $\delta^{P_{12}} = \{((q_1^{P_1}, q_1^{P_2}), \sigma, (q_2^{P_1}, q_2^{P_2})) \mid (q_1^{P_1}, \sigma, q_2^{P_1}) \in \delta^{P_1},\ (q_1^{P_2}, \sigma, q_2^{P_2}) \in \delta^{P_2},\ \sigma \in \Sigma^{\checkmark}\}$
  $\cup \{((q_1^{P_1}, q^{P_2}), \tau, (q_2^{P_1}, q^{P_2})) \mid (q_1^{P_1}, \tau, q_2^{P_1}) \in \delta^{P_1},\ q^{P_2} \in Q^{P_2}\}$
  $\cup \{((q^{P_1}, q_1^{P_2}), \tau, (q^{P_1}, q_2^{P_2})) \mid (q_1^{P_2}, \tau, q_2^{P_2}) \in \delta^{P_2}, q^{P_1} \in Q^{P_1}\}$
- $q_0^{P_{12}} = (q_0^{P_1}, q_0^{P_2})$

Proof of Theorem 9.1. Assume $\mathcal{U}$ is finite. Let $\phi = (UT, S, B)$ be an authorization policy, W be a finite workflow specification process, and $L_W = (Q^W, \Sigma^{\tau,\checkmark}, \delta^W, q_0^W)$. Because $\mathcal{U}$ is finite, π^{-1} maps the finite number of tasks $\mathcal{T}$ of W to a finite number of execution events. We construct a finite LTS $L_{W[\pi^{-1}]} = (Q^{W[\pi^{-1}]}, \Sigma^{\tau,\checkmark}, \delta^{W[\pi^{-1}]}, q_0^{W[\pi^{-1}]})$ as follows: $Q^{W[\pi^{-1}]} = Q^W$, $\delta^{W[\pi^{-1}]} = \{(q_1, t.u, q_2) \mid (q_1, t, q_2) \in \delta^W, t \in \mathcal{T}, u \in \mathcal{U}\} \cup \{(q_1, \sigma, q_2) \mid (q_1, \sigma, q_2) \in \delta^W, \sigma \in (\Sigma^{\tau,\checkmark} \setminus \mathcal{T})\}$, and $q_0^{W[\pi^{-1}]} = q_0^W$. In other words, $L_{W[\pi^{-1}]}$ is the same LTS as L_W except for every transition $q_1 \xrightarrow{\langle t \rangle} q_2$ in L_W, for a task t, there is a set of transitions $q_1 \xrightarrow{\langle t.u \rangle} q_2$ in $L_{W[\pi^{-1}]}$, for every user $u \in \mathcal{U}$.

Because $\mathcal{T}$ and $\mathcal{U}$ are finite, the basic authorization process A_{UT} is finite by Definition 4.3, every SoD process A_s, for $s \in S$, is finite by Definition 8.3, and every BoD process A_b, for $b \in B$, is finite by Definition 8.4. By Definition 8.5, A_ϕ is the parallel, synchronized composition of A_{UT}, every A_s, for $s \in S$, and every A_b, for $b \in B$. From Definition A.1, it follows that A_ϕ is finite too. Let $L_{A_\phi} = (Q^{A_\phi}, \Sigma^{\tau,\checkmark}, \delta^{A_\phi}, q_0^{A_\phi})$.

By condition (1) of Definition 9.2, an enforcement process $E_{\phi,W}$ for ϕ on W must trace refine A_ϕ, *i.e.* $A_\phi \sqsubseteq_{\mathsf{T}} E_{\phi,W}$. Therefore, if $E_{\phi,W}$ exists, there exists an LTS $L_{E_{\phi,W}} = (Q^{E_{\phi,W}}, \Sigma^{\tau,\checkmark}, \delta^{E_{\phi,W}}, q_0^{E_{\phi,W}})$ for $Q^{E_{\phi,W}} = Q^{A_\phi}$, $\delta^{E_{\phi,W}} \subseteq \delta^{A_\phi}$, and $q_0^{E_{\phi,W}} = q_0^{A_\phi}$.

Because A_ϕ is finite, so is δ^{A_ϕ} and there are finitely many LTSs that are candidates to be $L^{E_{\phi,W}}$. It is straightforward to construct a process from an LTS. Because there are finitely many LTSs, there are also finitely many corresponding processes. For each such process P, we can check if $(W[\pi^{-1}] \parallel P)[\pi] =_{\mathsf{F}} W$. Failure equivalence of finite processes is decidable [Roscoe 1994], for example using the CSP model-checker FDR [FSE 2005]. If none of the candidate processes P satisfies the above check, *i.e.* condition (2) of Definition 9.2, there exists no enforcement process for ϕ on W. ■

A.6 Proof of Lemma 9.2

We refer to a line i of CGraph as CGi and to line j of EPEA as EAj.

Proof. Assume a workflow specification process W, let $T = \{t \in \mathcal{T} \mid \exists i \in \mathsf{T}(W), t \in i\}$, and $\phi = (UT, S, B)$ be an authorization policy. Assume EPEA(T, ϕ) returns a relation R.

If $T = \varnothing$, then W does not engage in any task and $R = \varnothing$ by EA2. Because ϕ is an authorization policy for W and W contains no tasks, $UT = \varnothing$, $S = \varnothing$, and $B = \varnothing$. It follows that $A_\phi = A_{UT}$. Therefore, A_ϕ engages in every point and ✓, by Definition 4.3. It follows that $A_{UT} \sqsubseteq_{\mathsf{T}} W$, *i.e.* condition (1) of Definition 9.2 holds. By the trace semantics of CSP and because W does not engage in tasks, $(W[\pi^{-1}] \parallel W[\varnothing])[\pi] =_{\mathsf{F}} (W \parallel W)[\pi] =_{\mathsf{F}} W[\pi] =_{\mathsf{F}} W$, *i.e.* condition (2) of Definition 9.2 holds.

Assume $T \neq \varnothing$. Because EPEA returns a relation and $T \neq \varnothing$, CGraph(T, ϕ) returns a graph (V, E) and a function L by EA1, EA4, and EA5. Furthermore, LCol(V, E, L) returns a coloring $\mathsf{col}_{\mathsf{L}}$ by EA4 and EA5. Because $T \neq \varnothing$ and by CG2, CG3, and CG10, $V \geq 1$. It follows from Lemma 2.2 (soundness and completeness of LCol) that $\mathsf{col}_{\mathsf{L}}$ is an L-coloring for (V, E). Let $t \in T$. By CG2, CG3, and CG10, there is exactly one vertex $v \in V$ such that $t \in v$. Therefore, there is exactly one tuple $(t, t.u) \in R$ by EA8, for a user u.

Let $i \in \mathsf{T}(W[R])$. In the following, we show for every constraint $c \in (\{UT\} \cup S \cup B)$ that $i^\frown\langle t.u\rangle \in \mathsf{T}(A_c)$. By Definitions 4.3, 8.3, and 8.4, also $i^\frown\langle o\rangle \in \mathsf{T}(A_c)$, for $o \in \mathcal{O}$, and $i^\frown\langle ✓\rangle \in \mathsf{T}(A_c)$. It follows by Definition 8.5 that $A_\phi \sqsubseteq_{\mathsf{T}} W[R]$, *i.e.* condition (1) of Definition 9.2 holds.

Case UT: Let $v \in V$ such that $t \in v$. By EA8, $u = \mathsf{col}_{\mathsf{L}}(v)$. By the definition of L-coloring, $\mathsf{col}_{\mathsf{L}}(v) \in \mathsf{L}(v)$. By CG4 and CG11, $\mathsf{L}(v) \subseteq \{u' \mid (u', t) \in UT\}$. Hence, $(u, t) \in UT$ and $i^\frown\langle t.u\rangle \in \mathsf{T}(A_{UT})$ by Definition 4.3.

Case $s \in S$: Let $s = (T_1, T_2, O)$. If $t \notin (T_1 \cup T_2)$ then $i^\frown\langle t.u\rangle \in \mathsf{T}(A_s)$ by Definition 8.3. Consider the case $t \in (T_1 \cup T_2)$. $(T_1 \cap T_2) = \varnothing$ by the definition

of SoD constraints. Without loss of generality, let $t \in T_1$ and $t \notin T_2$. Furthermore, let $t_2 \in T_2$ and $(t_2, t_2.u_2) \in R$, for a user u_2, and let $v_1, v_2 \in V$ such that $t \in v_1$ and $t_2 \in v_2$. By CG12–CG18, $\{v_1, v_2\} \in E$. By the definition of L-coloring, $\mathsf{col}_\mathsf{L}(v_1) \neq \mathsf{col}_\mathsf{L}(v_2)$ and therefore $u \neq u_2$ by EA8. Because there is only one execution event in R for every task, $t_2.u \notin i$ and therefore $i \hat{} \langle t.u \rangle \in \mathsf{T}(A_s)$ by Definition 8.3.

Case $b \in B$: Let $b = (T_1, O)$. If $t \notin T_1$ then $i \hat{} \langle t.u \rangle \in \mathsf{T}(A_b)$ by Definition 8.4. Consider the case $t \in T_1$. Let $t_2 \in T_1$ and $(t_2, t_2.u_2) \in R$ for a user u_2. Let $v \in V$ such that $t \in v$. By CG5–CG11 it holds that $t_2 \in v$. By EA8 it follows that $u = u_2$. Therefore, no matter whether $t_2.u_2 \in i$ or $t_2.u_2 \notin i$, $i \hat{} \langle t.u \rangle \in \mathsf{T}(A_b)$ by Definition 8.4.

It remains to be shown that $W[R]$ satisfies condition (2) of Definition 9.2. By CSP's traces model and because $R \subseteq \pi^{-1}$, $(W[\pi^{-1}] \parallel W[R])[\pi] =_\mathsf{F} W[R][\pi] =_\mathsf{F} W$. ■

A.7 Proof of Lemma 10.1

We refer to a line i of CGraph as CGi and to line j of EPEA as EAj.

Proof of Lemma 10.1. Let W be a workflow specification process, $\phi = (UT, S, B)$ an authorization policy, and $T = \{t \in \mathcal{T} \mid \exists i \in \mathsf{T}(W) \,.\, t \in i\}$.

Property (1): Assume $\mathrm{EPEA}(T, \phi)$ returns the relation R and let $\mathsf{alloc} = \{(t, u) \mid (t, t.u) \in R\}$. We first show that alloc is a function that maps every task in T to a user and afterward that $\mathsf{alloc} \models (W, \phi)$. Because $\mathrm{EPEA}(T, \phi)$ returns a relation, it follows by EA4, EA5, and EA8 that $\textsc{CGraph}(T, \phi)$ returns a graph (V, E) and a color-list function L. Furthermore, $\textsc{LCol}(V, E, \mathsf{L})$ returns an L-coloring col_L for (V, E). By CG1, CG3, and CG10, for every $t \in T$ there is exactly one $v \in V$ such that $t \in v$. It follows by the definition of an L-coloring, EA8, and the definition of alloc that alloc maps every $t \in T$ to a user.

We now show that the three conditions of Definition 10.1 hold for alloc. Condition (1): Let $t \in T$ and $v \in V$ such that $t \in v$. By the definition of alloc and EA8, $\mathsf{alloc}(t) = \mathsf{col}_\mathsf{L}(v)$. Furthermore, by the definition of an L-coloring, $\mathsf{col}_\mathsf{L}(v) \in \mathsf{L}(v)$ and thus $\mathsf{alloc}(t) \in \mathsf{L}(v)$. By CG1, CG4, and CG11 it follows that $\mathsf{L}(v) \subseteq \{u \mid (u, t) \in UT\}$ and therefore $(\mathsf{alloc}(t), t) \in UT$.

Condition (2): Let $(T_1, T_2, O) \in S$ be an SoD constraint and $t_1 \in T_1$ and $t_2 \in T_2$. Furthermore, let $v_1, v_2 \in V$ such that $t_1 \in v_1$ and $t_2 \in v_2$. It follows by CG12–CG18 that $\{v_1, v_2\} \in E$. By the definition of L-coloring, $\mathsf{col}_\mathsf{L}(v_1) \neq \mathsf{col}_\mathsf{L}(v_2)$ and therefore $\mathsf{alloc}(t_1) \neq \mathsf{alloc}(t_2)$ by EA8 and the definition of alloc.

Condition (3): Let $(T',O) \in B$ be a BoD constraint and $t_1,t_2 \in T'$. By CG5–CG10, there is a $v \in V$ such that $t_1 \in v$ and $t_2 \in v$. It follows by EA8 and the definition of alloc that $\mathsf{alloc}(t_1) = \mathsf{alloc}(t_2)$.

Property (2): Assume alloc is an allocation for W and ϕ. We prove by contradiction that $\textsc{CGraph}(T,\phi)$ returns a graph and a color list function. Assume that $\textsc{CGraph}(T,\phi)$ returns No. By CG20, and CG12–CG17, there exists an SoD constraint $(T_1,T_2,O) \in S$ with $t_1 \in T_1$ and $t_2 \in T_2$, such that between CG1 and CG11 a vertex v was created with $t_1 \in v$ and $t_2 \in v$. It follows by CG1, CG3, and CG5–CG10 that there is a set of BoD constraints $\{(T_3,O_3),\ldots,(T_n,O_n)\} \subseteq B$ such that for all $i,j \in \{3,\ldots,n\}$, $T_i \cap T_j \neq \varnothing$ and $\{t_1,t_2\} \subseteq T_3 \cup \ldots \cup T_n$. Because $(T_1,T_2,O) \in S$ it follows by condition (2) of Definition 10.1 that $\mathsf{alloc}(t_1) \neq \mathsf{alloc}(t_2)$. However, by repeatedly applying condition (3) of Definition 10.1, for (T_3,O_3) to (T_n,O_n), we also have $\mathsf{alloc}(t_1) = \mathsf{alloc}(t_2)$, contradicting $\mathsf{alloc}(t_1) \neq \mathsf{alloc}(t_2)$. Hence, $\textsc{CGraph}(T,\phi)$ returns a graph (V,E) and a color list function L.

Next, we show that for every $v \in V$ and $t_1,t_2 \in v$, $\mathsf{alloc}(t_1) = \mathsf{alloc}(t_2)$. Let $v \in V$ and $t_1,t_2 \in v$. If $t_1 = t_2$, then the property trivially holds. Otherwise, it again follows by CG1, CG3, and CG5–CG10 that there is a set of BoD constraints $\{(T_3,O_3),\ldots,(T_n,O_n)\} \subseteq B$ such that for all $i,j \in \{3,\ldots,n\}$, $T_i \cap T_j \neq \varnothing$ and $\{t_1,t_2\} \subseteq T_3 \cup \ldots \cup T_n$. By repeatedly applying condition (3) of Definition 10.1, we get $\mathsf{alloc}(t_1) = \mathsf{alloc}(t_2)$. Therefore, $\mathsf{col}_\mathsf{L} = \{(v,u) \mid v \in V, \exists t \in v \,.\, \mathsf{alloc}(t) = u\}$ is a function that maps every vertex in V to a user.

Finally, we shown that col_L is an L-coloring for (V,E). Let $v \in V$ be a vertex. Because $\mathsf{alloc}(t_1) = \mathsf{alloc}(t_2)$, for all $t_1,t_2 \in v$, $\mathsf{col}_\mathsf{L}(v) \in \cap_{t \in v}\{u \mid (u,t) \in UT\}$ by condition (1) of Definition 10.1 and the definition of col_L. By CG1, CG4, and CG11, $\mathsf{L}(v) = \cap_{t \in v}\{u \mid (u,t) \in UT\}$, and therefore $\mathsf{col}_\mathsf{L}(v) \in \mathsf{L}(v)$. Let $\{v_1,v_2\} \in E$ be an edge. By CG12–CG18 there is an SoD constraint $(T_1,T_2,O) \in S$ such that $t_1 \in v_1$ and $t_2 \in v_2$. It follows by condition (2) of Definition 10.1 that $\mathsf{alloc}(t_1) \neq \mathsf{alloc}(t_2)$. By the definition of col_L therefore $\mathsf{col}_\mathsf{L}(v_1) \neq \mathsf{col}_\mathsf{L}(v_2)$. Hence, col_L is an L-coloring for (V,E). ■

A.8 Proof of Lemma 10.2

Proof. Let a graph (V,E) and an integer k be an instance of the **NP**-complete ***k*-Coloring** problem. In the following, we present a polynomial reduction to **AEP**. Let $\mathcal{T} = V$ and, for $V = \{v_1,\ldots,v_n\}$, let $W = v_1 \to \ldots \to v_n \to \mathit{SKIP}$ be a workflow specification process. Furthermore, let $\mathcal{U} = \{1,\ldots,k\}$ and $UT = \mathcal{U} \times T$. For every $\{v_1,v_2\} \in E$, we add an SoD constraint $(v_1,v_2,\varnothing)$ to the set of SoD constraints S, let $B = \varnothing$, and thus we get the authorization policy $\phi = (UT,S,\varnothing)$.

Suppose an algorithm for **AEP** finds an allocation alloc such that $\mathsf{alloc} \models (W,\phi)$. We show that alloc is a k-coloring for (V,E). By our construction and Definition 10.1, $\mathsf{alloc} : V \to \{1,\ldots,k\}$, *i.e.* alloc has the domain and range of a k-coloring for (V,E). Consider an edge $\{v_1,v_2\} \in E$ and let $(v_1,v_2,\varnothing)$ be the corresponding SoD constraint S. By condition (2) of Definition 10.1, it follows that $\mathsf{alloc}(v_1) \neq \mathsf{alloc}(v_2)$. Hence, alloc is a k-coloring for (V,E).

Conversely, let $\mathsf{col} : V \to \{1,\ldots,k\}$ be a k-coloring for (V,E). Because $UT = \{1,\ldots,k\} \times V$, col satisfies condition (1) of Definition 10.1. By our construction and because col is a k-coloring for (V,E), for every SoD constraint $(v_1,v_2,\varnothing) \in S$, $\mathsf{col}(v_1) \neq \mathsf{col}(v_2)$, *i.e.* col satisfies condition (2) of Definition 10.1. Because $B = \varnothing$, col trivially satisfies condition (3) of Definition 10.1, too. Hence, $\mathsf{col} \models (W,\phi)$ and **AEP** is **NP**-hard.

Given an instance (W,ϕ) of **AEP** and a function $\mathsf{alloc} : \mathcal{T} \to \mathcal{U}$, one can check in polynomial time whether $\mathsf{alloc} \models (W,\phi)$ by verifying that alloc satisfies all three conditions of Definition 10.1. Hence, **AEP** is in **NP** and thereby **NP**-complete. ■

A.9 Proof of Lemma 10.3

We refer to a constraint or a set of constraints i in Definition 10.7 as Ci.

Proof. Let $(\mathsf{costR},UR^{\max},W,(UR,RT),S,B)$ be a **ROWA** instance and $(\mathbf{A},\mathbf{b},\mathbf{c})$ the corresponding **ILP**-instance returned by ROWAtoILP. Furthermore, let $U = \mathsf{dom}(UR^{\max})$, $R = \mathsf{ran}(UR^{\max})$, and $T = \{t \in \mathcal{T} \mid \exists i \in \mathsf{T}(W) \,.\, t \in i\}$.

Soundness: Let $\mathbf{x}$ be a feasible solution for $(\mathbf{A},\mathbf{b},\mathbf{c})$, UR' a user-role assignment, and alloc an allocation, such that $\mathbf{x} \sim (UR',\mathsf{alloc})$. We first show that $UR' \subseteq UR^{\max}$. To derive a contradiction, assume $(u,r) \in UR' \setminus UR^{\max}$, for a user u and a role r. By Definition 10.8, it follows that $x^{u,r} = 1$. However, this contradicts C5, which forces $x^{u,r}$ to be 0 (Remember, decision variables of a feasible solution only assume the values 0 and 1). Hence, $UR' \setminus UR^{\max} = \varnothing$ and therefore $UR' \subseteq UR^{\max}$.

By C2, alloc maps every task to exactly one user and is therefore a total function. We now show that $\mathsf{alloc} \models (W,(RT \circ UR',S,B))$. Let $(t,u) \in \mathsf{alloc}$, for a task t and a user u. By Definition 10.8, $x^{u,t} = 1$. It follows by C1 that there exists an r such that $x^{u,r} = 1$ and $(r,t) \in RT$. By Definition 10.8, $(u,r) \in UR'$ and therefore $(u,t) \in RT \circ UR'$. Hence, condition (1) of Definition 10.1 holds.

To show that condition (2) of Definition 10.1 holds, consider an SoD constraint $(T_1,T_2,O) \in S$. Let $u \in U$, $t_1 \in T_1$, and $t_2 \in T_2$. If $\mathsf{alloc}(t_1) = u$, then $x^{u,t_1} = 1$. It follows by C3 that $x^{u,t_2} = 0$ and therefore $\mathsf{alloc}(t_1) \neq \mathsf{alloc}(t_2)$. The case where $\mathsf{alloc}(t_2) = u$ is analogous.

For condition (3) of Definition 10.1, consider a BoD constraint $(T',O) \in B$. Let $u \in U$, $t_1 \in T'$, and $t_2 \in T'$. If $\mathsf{alloc}(t_1) = u$, then $x^{u,t_1} = 1$. It follows by C4 that $x^{u,t_2} = 1$ and therefore $\mathsf{alloc}(t_1) = \mathsf{alloc}(t_2)$. The case where $\mathsf{alloc}(t_1) \neq u$ is analogous.

Completeness: Let UR' be a user-role assignment, alloc an allocation, and $\mathbf{x}$ a vector such that $UR' \subseteq UR^{\max}$, $\mathsf{alloc} \models (W,(RT \circ UR',S,B))$, and $\mathbf{x} \sim (UR',\mathsf{alloc})$. In the following, we show that $\mathbf{x}$ is a feasible solution for $(\mathbf{A},\mathbf{b},\mathbf{c})$ by showing that every constraint of $(\mathbf{A},\mathbf{b},\mathbf{c})$ is satisfied.

C1: Let t be a task, u a user, and consider the constraint $\sum_{\{r|(r,t)\in RT\}} x^{u,r} \geq x^{u,t}$. If $\mathsf{alloc}(t) \neq u$, then $x^{u,t} = 0$ by Definition 10.8 and the constraint is trivially satisfied. If $\mathsf{alloc}(t) = u$, it follows by condition (1) of Definition 10.1 that $(u,t) \in RT \circ UR'$. Therefore, there exists an $r \in R$ such that $(u,r) \in UR'$ and $(r,t) \in RT$. By Definition 10.8, $x^{u,r} = 1$ and the constraint is therefore satisfied.

C2: Let t be a task. By Definition 10.1, alloc is a total function and therefore maps t to one user $u \in U$. By Definition 10.8 we have $x^{u,t} = 1$. Hence, $\sum_{u\in U} x^{u,t} = 1$.

C3: Let $(T_1,T_2,O) \in S$ be an SoD constraint, $t_1 \in T_1$, $t_2 \in T_2$, and $u \in U$. Consider the constraint $x^{u,t_1} + x^{u,t_2} \leq 1$. If $\mathsf{alloc}(t_1) \neq u$, then $x^{u,t_1} = 0$ by Definition 10.8 and the constraint is trivially satisfied. If $\mathsf{alloc}(t_1) = u$, then $x^{u,t_1} = 1$ by Definition 10.8. To derive a contradiction, assume that $x^{u,t_2} = 1$. It follows by Definition 10.8 that $\mathsf{alloc}(t_2) = u$. However, this contradicts condition (2) of Definition 10.1, which must hold because $\mathsf{alloc} \models (W,(RT \circ UR',S,B))$. Hence $x^{u,t_2} = 0$ and the constraint is satisfied.

C4: Let $(T',O) \in B$ be a BoD constraint, $t_1,t_2 \in T'$, and $u \in U$. Consider the constraint $x^{u,t_1} = x^{u,t_2}$ and let $x^{u,t_1} = 1$. It follows from Definition 10.8 that $\mathsf{alloc}(t_1) = u$. To derive a contradiction, assume $x^{u,t_2} = 0$. By C2, there exists an $u_2 \in U$ such that $u \neq u_2$ and $x^{u_2,t_2} = 1$. By Definition 10.8, then $\mathsf{alloc}(t_2) = u_2$. However, this contradicts condition (3) of Definition 10.1, which must hold because $\mathsf{alloc} \models (W,(RT \circ UR',S,B))$. Hence $x^{u,t_2} = 1$ and the constraint is satisfied. The case where $x^{u,t_1} = 0$ and we derive a contradiction for $x^{u,t_2} = 1$ is analogous.

C5: $\sum_{(u,r)\in(U\times R)\setminus UR^{\max}} x^{u,r} = 0$ follows directly from $UR' \subseteq UR^{\max}$ and Definition 10.8.

C6 and C7: The satisfaction of these constraints follows directly from Definition 10.8. ■

Appendix B
SoDA$^{\mathsf{S}}$

We summarize SoDA$^{\mathsf{S}}$, the semantics for SoDA terms that was originally introduced by Li and Wang [2008]. They define satisfaction with respect to a tuple (U, UR), where $U \subseteq \mathcal{U}$ and $UR \subseteq U \times \mathcal{R}$. In contrast, we defined multiset satisfaction SoDA$^{\mathsf{M}}$ simply with respect to a user-role assignment UR.

Definition B.1 (SoDA$^{\mathsf{S}}$) *Let S be a non-empty set of users and $r \in \mathcal{R}$ a role. Furthermore, let U be a set of users and $UR \subseteq U \times \mathcal{R}$ a user-role assignment. For two sets of users Y and Z, and a term ϕ,* set satisfiability *is the smallest relation between two sets of users, user-role assignments, and terms, written $Y \models^{\mathsf{S}}_{(U,UR)} \phi$, closed under the following rules:*

(1) $\dfrac{}{\{u\} \models^{\mathsf{S}}_{(U,UR)} \mathsf{All}}\ u \in U$

(2) $\dfrac{}{\{u\} \models^{\mathsf{S}}_{(U,UR)} r}\ (u, r) \in UR$

(3) $\dfrac{}{\{u\} \models^{\mathsf{S}}_{(U,UR)} S}\ u \in (S \cap U)$

(4) $\dfrac{\{u\} \not\models^{\mathsf{S}}_{(U,UR)} \phi}{\{u\} \models^{\mathsf{S}}_{(U,UR)} \neg\phi}$

(5) $\dfrac{\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi}{\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi^{+}}$

(6) $\dfrac{\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi,\ Y \models^{\mathsf{S}}_{(U,UR)} \phi^{+}}{(\{u\} \cup Y) \models^{\mathsf{S}}_{(U,UR)} \phi^{+}}$

(7) $\dfrac{Y \models^{\mathsf{S}}_{(U,UR)} \phi}{Y \models^{\mathsf{S}}_{(U,UR)} (\phi \sqcup \psi)}$

(8) $\dfrac{Y \models^{\mathsf{S}}_{(U,UR)} \psi}{Y \models^{\mathsf{S}}_{(U,UR)} (\phi \sqcup \psi)}$

(9) $\dfrac{Y \models^{\mathsf{S}}_{(U,UR)} \phi,\ Y \models^{\mathsf{S}}_{(U,UR)} \psi}{Y \models^{\mathsf{S}}_{(U,UR)} (\phi \sqcap \psi)}$

(10) $\dfrac{Y \models^{\mathsf{S}}_{(U,UR)} \phi,\ Z \models^{\mathsf{S}}_{(U,UR)} \psi}{(Y \cup Z) \models^{\mathsf{S}}_{(U,UR)} (\phi \odot \psi)}$

(11) $\dfrac{Y \models^{\mathsf{S}}_{(U,UR)} \phi,\ Z \models^{\mathsf{S}}_{(U,UR)} \psi}{(Y \cup Z) \models^{\mathsf{S}}_{(U,UR)} (\phi \otimes \psi)}\ (Y \cap Z) = \varnothing\,.$