RZ 3812 (# Z1112-001) 12/08/11
Computer Science 35 pages

Research Report

Dynamic Enforcement of Abstract Separation of Duty Constraints

David Basin

Institute for Information Security, ETH Zurich, 8092 Zurich, Switzerland
E-mail: basin@inf.ethz.ch

Samuel J. Burri

ETH Zurich and IBM Research - Zurich
E-mail: samuel.burri@inf.ethz.ch, sbu@zurich.ibm.com

Günter Karjoth

IBM Research – Zurich, 8803 Rüschlikon, Switzerland
E-mail: gka@zurich.ibm.com

Dynamic Enforcement of Abstract Separation of Duty Constraints

David Basin, ETH Zurich
Samuel J. Burri, ETH Zurich and IBM Research – Zurich
Günter Karjoth, IBM Research – Zurich

Abstract

Separation of Duties (SoD) aims at preventing fraud and errors by distributing tasks and associated authorizations among multiple users. Li and Wang proposed an algebra (SoDA) for specifying SoD requirements, which is both expressive in the requirements it formalizes and abstract in that it is not bound to a workflow model. In this article, we bridge the gap between the specification of SoD constraints modeled in SoDA and their enforcement in a dynamic, service-oriented enterprise environment. Using the process algebra CSP, we proceed by generalizing SoDA's semantics to traces, modeling workflow executions that satisfy the respective SoDA terms. We then refine the set of traces induced by a SoDA term to also take a workflow's control-flow and role-based authorizations into account. Our formalization supports the enforcement of SoD on general workflows and handles changing role-assignments during workflow execution, addressing a well-known source for fraud.

The resulting CSP model serves as blueprint for a distributed and loosely-coupled architecture where SoD enforcement is provisioned as a service. This new concept, which we call SoD as a Service, facilitates a separation of concern between business experts and security professionals. As a result, integration and configuration efforts are minimized and enterprises can quickly adapt to organizational, regulatory, and technological changes. We describe an implementation of SoD as a Service, combining commercial components, such as a workflow engine, and newly developed components, such as an SoD-enforcement monitor.

Starting out with a generalization of SoDA's semantics and ending up with a prototype implementation, we go the full distance from theory to practice. To evaluate our design decisions and to demonstrate the feasibility of our approach, we present a case study of a drug dispensation workflow deployed in a hospital.

1 Introduction

Most information-security mechanisms aim at protecting resources from external threats. However, threats often reside within organizations where authorized system users may intentionally or accidentally misuse system resources. Examples of this are non-compliance and include the scandals [Eco01] that led to regulations such as the Sarbanes-Oxley Act [SO02]. These regulations require companies to document their processes, to identify conflicts of interests, to adopt countermeasures, and to audit and control these activities. *Separation of Duties (SoD)* is a well-established extension of access control that aims to ensure data integrity, in particular to prevent fraud and errors [SS75,San88]. The idea behind SoD is to split critical (business) processes into multiple tasks and to ensure that no single user can execute them all. Therefore, at least two users must be involved in a process' execution and fraud requires their collusion.

Existing specification formalisms and enforcement mechanisms for SoD are limited in the kinds of constraints they can handle. Moreover, they are typically bound to specific workflow models. The SoD algebra (SoDA) [LW08] constitutes a notable exception. It allows the modeling of SoD constraints at a high level of abstraction, combining quantification and qualification requirements. As an example, 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 take part in a business process, independent of the details of the process itself. Separating concerns this way facilitates a loose coupling between an application's business logic and its security constraints. 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 this article, we bridge the gap between the abstract specification of SoD constraints, formalized using SoDA, and an architecture for their enforcement in a dynamic and modular enterprise environment. We proceed by constructing formal models of workflows, role-based static authorizations, SoD constraints, and combine them to model an SoD-secure workflow system. We specify these models using the process algebra CSP [Ros97], whose trace-based model is a natural fit for describing workflow executions and whose notion of process synchronization allows us to decompose the system into loosely coupled components.

Our CSP model serves as blueprint for a proof-of-concept implementation. We provision the SoD enforcement functionality as an instance of the software delivery model *Software as a Service (SaaS)* [TBB03], which we call *SoD as a Service*. We show analytically that our approach has an acceptable runtime performance for the workflows used in practice even though deciding the satisfiability of a workflow instance with respect to our CSP model is **NP**-complete in general. We support this analysis with performance measurements from an extensive and realistic case study. In addition, our case study demonstrates the feasibility of SoD as a Service and pinpoints critical design decisions.

Along the way from abstract specification to enforcement, we tackle the following challenges:

1. **Generalization of SoDA semantics:** Due to SoDA's abstract nature, design decisions arise when mapping SoDA terms onto workflow instances. In particular, a link between the satisfaction of terms and the tasks executed in workflow instances is missing. We provide a solution to this problem by generalizing Li and Wang's set-based semantics first to a multiset semantics and second to a trace-based semantics. A correctness proof for the CSP model of the SoD enforcement component establishes that every SoD-constrained workflow instance that successfully terminates satisfies the respective SoDA term with respect to our trace semantics.

2. **Flexible integration:** 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 with our novel concept of SoD as a Service and thereby achieve a loose coupling between a workflow engine that executes the business logic, a user repository that administers users and their 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 the IT infrastructure.

3. **Changing authorizations:** Previous work on SoD enforcement makes the assumption that authorizations do not change during workflow execution. However, organizational changes, triggered by acquisitions, promotions, and job cuts, are among the major sources of fraud [EY09]. By incorporating administrative events, which model authorization changes, in our trace-based SoDA semantics we overcome this limitation. Our formal models and implementation are therefore well-suited to handle the dynamics of today's fast-paced business environments.

The reminder of this article is structured as follows. In Section 2, we provide background on CSP and multisets. In Section 3, we formalize workflows, role-based authorizations, and their composition. Furthermore, we introduce our case study. We then define in Section 4 SoDA's syntax, which we generalize to multisets and traces. Based on our CSP models, we implement in Section 5 SoD as a Service in an industrial workflow environment and present performance measurements for our case study. We evaluate our approach and identify future work in Section 6, present related work in Section 7, and conclude in Section 8. The Appendix provides proofs and summarizes Li and Wang's original SoDA semantics. Overall, this article combines and extends results from our previous papers [BBK09,BBK11a].

2 Background

2.1 CSP

We use a subset of Hoare's process algebra CSP [Ros97] to formalize the enforcement of authorization constraints on workflows. CSP describes a system as a set of communicating *processes* that concurrently engage in *events*. Σ denotes the set of all regular events. In addition, we use the special event $\checkmark$ that communicates successful termination. Let $D \subseteq \Sigma$. We write $D^{\checkmark}$ for $D \cup \{\checkmark\}$. Events can be structured as tuples. For example, $z_1.z_2.\ldots.z_n$ denotes the event that corresponds to the tuple $(z_1, z_2, \ldots, z_n) \in Z_1 \times Z_2 \times \ldots \times Z_n$, for sets Z_1, Z_2, ..., Z_n and $n \geq 1$.

A *trace*, denoted $\langle \sigma_1, ..., \sigma_n \rangle$, is a sequence of events, possibly ending with the special event $\checkmark$. $\langle\rangle$ denotes the *empty* trace and $i_1 \hat{} i_2$ the *concatenation* of two traces i_1 and i_2. Moreover, D^* denotes the set of all finite traces over the set of events D and its superset $D^{*\checkmark} = D^* \cup \{i \hat{} \langle\checkmark\rangle \mid i \in D^*\}$ also includes the traces ending with $\checkmark$. 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 a trace i and $D \subseteq \Sigma^{\checkmark}$, $i \upharpoonright D$ 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 $\sigma \notin D$.

Let $\mathcal{N}$ be the set of *process names* and $n \in \mathcal{N}$. The set of *processes* $\mathcal{P}$ is inductively

defined by the grammar

$$\mathscr{P} ::= \sigma \to \mathscr{P} \mid \mathit{SKIP} \mid \mathit{STOP} \mid n \mid \mathscr{P} \,\Box\, \mathscr{P} \mid \mathscr{P} \sqcap \mathscr{P} \mid \mathscr{P} \underset{D}{\parallel} \mathscr{P} \mid \mathscr{P} \,;\, \mathscr{P} \;,$$

where $\sigma \in \Sigma$. The *assignment* of a process P to n is denoted by $n = P$ and can be *parametrized*. For example $n(z) = P$ defines a process parametrized by the variable z.

There are different approaches to formally describing the behavior of a process. In this article, we use CSP's *traces model* T that describes a process P as a prefix-closed set of finite traces $\mathsf{T}(P) \subseteq \Sigma^{*\checkmark}$. We say P *accepts* i if $i \in \mathsf{T}(P)$.

In the following, let P, P_1, and P_2 be processes. The process $\sigma \to P$ *engages* in the event σ first and behaves like P afterward. Formally, $\mathsf{T}(\sigma \to P) = \{\langle\rangle\} \cup \{\langle\sigma\rangle\hat{}\, i \mid i \in \mathsf{T}(P)\}$. This notation can be extended. For $D \subseteq \Sigma$, the process $\sigma : D \to P$ engages in every $\sigma \in D$ and afterwards behaves like P. The process *SKIP* engages in the special event $\checkmark$ and terminates afterwards, formally $\mathsf{T}(\mathit{SKIP}) = \{\langle\rangle, \langle\checkmark\rangle\}$. The process *STOP* does not engage in any event and therefore represents deadlock, *i.e.* $\mathsf{T}(\mathit{STOP}) = \{\langle\rangle\}$. For an assignment $n = P$, the process n behaves like P. The process $P_1 \,\Box\, P_2$ denotes the *external* choice between P_1 and P_2 in that the environment can choose whether the process behaves like P_1 or P_2. Similarly, $P_1 \sqcap P_2$ denotes the *internal* choice between P_1 and P_2 in that the process can decide whether it behaves like P_1 or P_2. In terms of 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 process $P_1 \underset{D}{\parallel} P_2$ represents the parallel composition of P_1 and P_2 *synchronized* on $D \subseteq \Sigma$. 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 $P_1 \underset{\Sigma}{\parallel} P_2$. Similarly, $P_1 \,|||\, P_2$ 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. Afterwards, the process behaves like P_2. Formally, $\mathsf{T}(P_1 \,;\, P_2) = (\mathsf{T}(P_1) \cap \Sigma^*) \cup \{i_1\hat{}\, i_2 \mid i_1\hat{}\langle\checkmark\rangle \in \mathsf{T}(P_1), i_2 \in \mathsf{T}(P_2)\}$.

2.2 Multisets

We make extensive use of multisets and therefore briefly review their notation. A *multiset*, or *bag*, is a collection of objects where repetition is allowed [Syr00]. 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 $f(z)$. We 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 $z \in Z$.

3 Authorization-constrained Workflows

We call a unit of work a *task*. Because SoD constraints are concerned with human activities, we concentrate on tasks that are executed by humans, either directly, *e.g.* by filling in a form, or indirectly, *e.g.* by executing a program on their behalf. The temporal ordering of tasks and the causal dependencies between them, which together implement a business objective, are called a *workflow*. An alternative name for workflow is *business process*, though we stick in this article to the term workflow.

At *design time*, a workflow is specified using a workflow modeling language. At *run time* it is executed by a *workflow engine*. We call an execution of a workflow a *workflow instance*. A workflow engine may execute multiple instances of the same workflow in parallel. The execution of a task in a workflow instance is called a *task instance*. Standard workflow modeling languages, such as the one we use in Section 3.3, allow the specification of loops, parallel, and conditional execution. Therefore, there can be zero or more instances of the same task in one workflow instance.

3.1 Workflow Formalization

For the rest of this article, let $\mathscr{U}$ be a set of *users* and $\mathscr{T}$ a set of *tasks*. For a task $t \in \mathscr{T}$ and a user $u \in \mathscr{U}$, we call an event of the form $t.u$ a *(task) execution event* and denote by $\mathscr{X} = \{t.u \mid t \in \mathscr{T}, u \in \mathscr{U}\}$ the set of all execution events. An execution event $t.u$ models the execution of the task t by the user u, *i.e.* a task instance together with its associated user.

We now formalize workflows using CSP as follows.

Definition 1 *A* workflow process *is a process W such that $\mathsf{T}(W) \subseteq \mathscr{X}^{*\checkmark}$*

Let a workflow process W be given. We call a trace $i \in \Sigma^{*\checkmark}$ a *workflow trace* of W, if $(i \upharpoonright \mathscr{X}^{\checkmark}) \in \mathsf{T}(W)$. A workflow trace i models a workflow instance. Note that i may not only include execution events and we will subsequently introduce administrative events that model complementary activities taking place during workflow execution. However, in order to be a workflow trace of W, only the execution events in i must be a trace of W; hence the restriction $i \upharpoonright \mathscr{X}^{\checkmark}$. We say 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.

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

3.2 Composing Workflows and Access Control

We describe authorized task executions in terms of a process A. Given a workflow process W, we then describe the enforcement of the authorization constraints encoded in A by executing A in parallel with W, synchronized on $\mathscr{X}$, formally $W \underset{\mathscr{X}}{\parallel} A$.

We use *Role-based Access Control (RBAC)* [FSG+01] for specifying workflow-independent authorizations and only make use of RBAC's core feature, the decomposition of the user-permission assignment into a user-role and a role-permission assignment. For the reminder of this article, let $\mathscr{R}$ be a set of *roles*.

Definition 2 *An* RBAC configuration *is a tuple* (UA, PA)*, where* $UA \subseteq \mathscr{U} \times \mathscr{R}$ *is a* user-assignment relation *and* $PA \subseteq \mathscr{R} \times \mathscr{T}$ *is a* permission-assignment relation*.*

In this article, a permission corresponds to the right to execute a task. We therefore assign roles directly to tasks. Assume an RBAC configuration (UA, PA) and a user u. For a role r, we say that u *acts in the role* r if $(u,r) \in UA$. Furthermore, for a task t we say that u is *authorized to execute* t with respect to (UA, PA) if there is a role r such that $(u,r) \in UA$ and $(r,t) \in PA$.

In contrast to the NIST RBAC standard [FSG+01], 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 changing RBAC configurations as discussed below.

We model changes to the RBAC configuration by a set of events $\mathscr{A} \subseteq \Sigma$ 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-assignment relation. We do not consider administrative events that change permission-assignment relations. 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.

We now specify the evolution of an RBAC configuration and authorized task executions as a process.

Definition 3 *For an RBAC configuration* (UA, PA) *we call the process*

$$\begin{aligned} RBAC(UA,PA) \quad = \quad & \big((t.u) : \{t.u \mid \exists r \in \mathscr{R}.\ (u,r) \in UA, (r,t) \in PA\} \to RBAC(UA,PA)\big) \\ & \Box\ \big(\mathsf{add}.u : \mathscr{U}.r : \mathscr{R} \to RBAC(UA \cup \{(u,r)\}, PA\big) \\ & \Box\ \big(\mathsf{rm}.u : \mathscr{U}.r : \mathscr{R} \to RBAC(UA \setminus \{(u,r)\}, PA)\big) \\ & \Box\ SKIP \end{aligned}$$

an RBAC process*.*

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

Definition 4 *For a workflow process* W *and an RBAC configuration* (UA, PA)*, we call the process*

$$SW(UA,PA) = W \underset{\mathscr{X}}{\parallel} RBAC(UA,PA)$$

a secure (workflow) process*.*

Like an RBAC process, a secure process $SW(UA, PA)$ is parametrized by an RBAC configuration. $SW(UA, PA)$ 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 (UA, PA). By synchronizing only on execution events, arbitrary administrative events can be interleaved with execution events in any order. Thus, the RBAC configuration can change during a workflow's execution.

Having introduced all the kinds of events that we need, specifically, $\Sigma = \mathscr{X} \cup \mathscr{A}$, we now introduce the case study that accompanies this article.

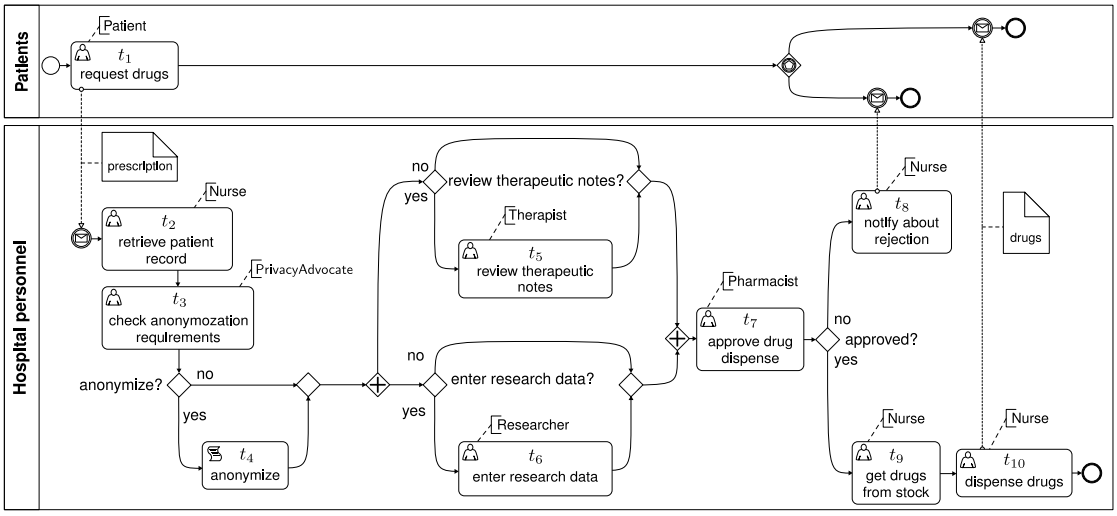


Figure 1: Drug dispensation process modeled in BPMN

3.3 Case Study

We illustrate the notions introduced above with a drug dispensation workflow from [MPH+09]. 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 are very expensive and therefore require special diligence.

Figure 1 shows a visualization of the drug dispensation workflow in the *Business Process Modeling Notation (BPMN)* [BPMN2]. For our case study, let $\mathscr{T} = \{t_1, \ldots, t_{10}\}$, where t_1 refers to Request Drugs, t_2 to Retrieve Patient Record, etc., as illustrated in Figure 1. The set of users $\mathscr{U}$ and the set of roles $\mathscr{R}$ are shown in Figure 2. Let (UA_1, PA) be the initial RBAC configuration of our case study. The user-assignment relation UA_1 is depicted in Figure 2, ignoring the dashed and dotted lines between users and roles, *e.g.* $(\mathsf{Alice}, \mathsf{Therapist}) \in UA_1$ and $(\mathsf{Alice}, \mathsf{Pharmacist}) \notin UA_1$. The permission-assignment relation PA is illustrated in Figure 1 by means of BPMN annotations. For example, only users acting in the role Nurse are authorized to execute t_2 with respect to (UA_1, PA). We assigned only one role to each task but in general tasks can be annotated with sets of roles.

An instance of the drug dispensation workflow is started by a Patient who requests drugs by handing his prescription to a Nurse. The Nurse retrieves the patient's record from the hospital's database and forwards all data to a PrivacyAdvocate who checks whether the patient's 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. If therapeutic notes are contained in the prescription, they are reviewed by a Therapist. In parallel, research-related data is added by a Researcher if the requested drugs are in an experimental state. Finally, a Pharmacist either approves the dispensation and a Nurse collects the drugs from the stock and gives them to the patient, or he denies the dispensation and a Nurse informs the Patient accordingly.

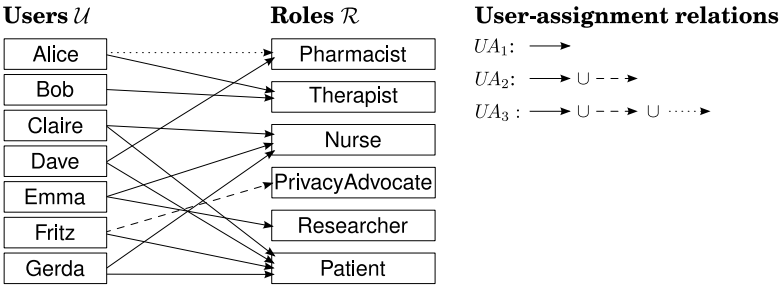


Figure 2: User-assignment relations

We model the drug dispensation workflow in CSP as the workflow process

$$
\begin{aligned}
W &= t_1.u_1 : \mathscr{U} \to t_2.u_2 : \mathscr{U} \to t_3.u_3 : \mathscr{U} \to \big((W_1 \,|||\, W_2)\,;\, W_3\big) \\
W_1 &= \mathit{SKIP} \sqcap (t_5.u_5 : \mathscr{U} \to \mathit{SKIP}) \\
W_2 &= \mathit{SKIP} \sqcap (t_6.u_6 : \mathscr{U} \to \mathit{SKIP}) \\
W_3 &= t_7.u_7 : \mathscr{U} \to \big((t_8.u_8 : \mathscr{U} \to \mathit{SKIP}) \sqcap (t_9.u_9 : \mathscr{U} \to t_{10}.u_{10} : \mathscr{U} \to \mathit{SKIP})\big)\ .
\end{aligned}
$$

Because we do not model data-flow, we over-approximate gateway decisions, such as whether therapeutical notes must be reviewed, with CSP's operator $\sqcap$ (internal choice). Let $SW(UA_1, PA)$ be the secure process for W and (UA_1, PA). The trace $i_1 = \langle t_1.\mathsf{Fritz}, t_2.\mathsf{Emma}, t_3.\mathsf{Fritz}, t_5.\mathsf{Bob}\rangle$ is a workflow trace of W because $(i_1 \upharpoonright \mathscr{X}^{\checkmark}) \in \mathsf{T}(W)$. However, i_1 is not a trace of $SW(UA_1, PA)$ because Fritz is not a PrivacyAdvocate and therefore not authorized to execute t_3 with respect to (UA_1, PA).

Consider now the trace $i_2 = \langle t_1.\mathsf{Fritz}, t_2.\mathsf{Emma}, \mathsf{add.Fritz.PrivacyAdvocate}, t_3.\mathsf{Fritz}, t_5.\mathsf{Bob}\rangle$. This trace is similar to i_1 but includes the administrative event add.Fritz.PrivacyAdvocate. By engaging in this administrative event, the user-assignment relation UA_1 becomes $UA_2 = UA_1 \cup \{(\mathsf{Fritz}, \mathsf{PrivacyAdvocate})\}$. Because Fritz is authorized to execute t_3 with respect to (UA_2, PA), i_2 is a trace of $SW(UA_1, PA)$. With respect to execution events, i_2 is equal to i_1, *i.e.* $i_2 \upharpoonright \mathscr{X}^{\checkmark} = i_1 \upharpoonright \mathscr{X}^{\checkmark}$, and therefore i_2 is also a workflow trace of W.

4 Generalizing Abstract Separation of Duty Constraints to Traces

4.1 SoDA Syntax

Our work builds on Li and Wang's *Separation of Duty Algebra (SoDA)* [LW08]. We present below the syntax of SoDA terms.

Definition 5 *A* SoDA grammar $\mathfrak{S}$ *with respect to a set of users* $\mathscr{U} = \{u_1, \ldots, u_n\}$ *and a set of roles* $\mathscr{R} = \{r_1, \ldots, r_m\}$ *is a quadruple* (N, T, P, S) *where:*

- $N = \{S, CT, UT, AT, US, UR, U, R\}$ *is the set of nonterminal symbols,*
- $T = \{',', (,), \{, \}, \otimes, \odot, \sqcup, \sqcap, {}^{+}, \neg, \mathsf{All}\} \cup \mathscr{U} \cup \mathscr{R}$ *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 [LW08] 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}.

4.2 SoDA Multiset Semantics

Li and Wang define the satisfaction of SoDA terms for *sets* of users [LW08]. We refer to their semantics as SoDA$^{\mathsf{S}}$. It allows for quantitative constraints where terms define how many different users must execute tasks in a workflow instance. However, SoDA$^{\mathsf{S}}$ does not express how many tasks each of these users must execute. Consider the policy P that requires Bob to execute two task, 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 traces that correspond to satisfying assignments of ϕ. If we define the correspondence between sets and 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 new 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, ϕ is only satisfied under SoDA$^{\mathsf{M}}$ by the multiset $\{\!\{\mathsf{Bob}, \mathsf{Bob}\}\!\}$. Mapping multisets to traces is straightforward. For

example, 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 traces than SoDA$^{\mathsf{S}}$.

Definition 6 *Let $U \subseteq \mathscr{U}$ be a non-empty set of users and $r \in \mathscr{R}$ a role. For a multiset of users* $\mathbf{U}$*, a term ϕ, and a user-assignment relation UA,* multiset satisfiability *is the smallest ternary relation between multisets of users, user-assignment relations, and terms, written $\mathbf{U} \models^{\mathsf{M}}_{UA} \phi$, that is closed under the rules:*

(1) $\dfrac{}{\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \mathsf{All}} \quad \exists r \in \mathscr{R}\,.\,(u,r) \in UA$

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

(3) $\dfrac{}{\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} U} \quad u \in U \text{ and } \exists r \in \mathscr{R}\,.\,(u,r) \in UA$

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

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

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

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

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

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

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

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

We say that $\mathbf{U}$ *satisfies ϕ with respect to UA* if $\mathbf{U} \models^{\mathsf{M}}_{UA} \phi$. Informally, a user u satisfies the term All if u is in the domain of UA. A user u satisfies a role r if there is a role assignment (u,r) in UA, and u satisfies a set of users U if u is member of U and is in the domain of UA. 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 where a user is 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 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 our case study and the second illustrates the difference between SoDA$^{\mathsf{M}}$ and SoDA$^{\mathsf{S}}$.

Example 1 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 encoded 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-assignment relation UA_3 shown in Figure 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 UA_3. However, $\mathbf{U}_2 = \{\!\{\mathsf{Bob}, \mathsf{Emma}, \mathsf{Fritz}, \mathsf{Gerda}, \mathsf{Gerda}\}\!\}$ does not satisfy ϕ with respect to UA_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 UA_3. ◇

Example 2 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. This example illustrates again how SoDA$^{\mathsf{M}}$ refines SoDA$^{\mathsf{S}}$. ◇

We conclude by formally relating SoDA$^{\mathsf{M}}$ and SoDA$^{\mathsf{S}}$. Under SoDA$^{\mathsf{S}}$, $X \models^{\mathsf{S}}_{(U,UR)} \phi$ denotes the satisfaction of a term ϕ by a set of users $X \subseteq \mathscr{U}$ with respect to a tuple (U, UR), where $U \subseteq \mathscr{U}$ and $UR \subseteq U \times \mathscr{R}$. 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 UA, as can be seen in Rule (3) of Definition. 6. For a user-assignment relation UA, the function $\mathsf{lwconf}(UA) = (dom(UA), UA)$ maps UA 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 shows how SoDA$^{\mathsf{M}}$ generalizes SoDA$^{\mathsf{S}}$. We prove Lemma 1 in Appendix A.2.1.

Lemma 1 *For all terms ϕ, all user-assignment relations UA, and all multisets of users $\mathbf{U}$, if $\mathbf{U} \models^{\mathsf{M}}_{UA} \phi$, then $\mathsf{userset}(\mathbf{U}) \models^{\mathsf{S}}_{\mathsf{lwconf}(UA)} \phi$.*

4.3 Enforcement Approach and Requirements

As shown above, SoDA specifies SoD constraints at a high level of abstraction. However, the enforcement takes place at runtime 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 $SOD_\phi(UA)$, called the *SoD-enforcement process*, parametrized by a user-assignment relation UA. $SOD_\phi(UA)$ accepts all traces corresponding to a multiset that satisfies ϕ with respect to UA. We show later in Section 5 how to implement $SOD_\phi(UA)$ 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 role assignments, thereby preventing him from subsequently executing tasks. Similarly, if Alice joins a company or changes positions,

and is therefore assigned new roles, she should also be able to execute tasks in workflow instances that were started prior to the organizational change. Assuming that a user-assignment relation does not change during the execution of a workflow instance is therefore overly restrictive. The SoD-enforcement process defined below accounts for such changes. The function upd ("update") describes how a trace of administrative events changes a user-assignment relation.

Definition 7 *Let $a \in \mathscr{A}^*$ be a trace of administrative events and UA a user-assignment relation. The function* upd *is then defined as*

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

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

Let ϕ be a term, UA a user-assignment relation, and $SOD_\phi(UA)$ the SoD-enforcement process for ϕ and UA. We postulate the following requirements for $SOD_\phi(UA)$:

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

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

(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-assignment relation. (R2) formulates agreement with SoDA$^{\mathsf{M}}$, where for a multiset of users $\mathbf{U}$, if $\mathbf{U} \models^{\mathsf{M}}_{UA} \phi$, then each user in $\mathbf{U}$ satisfies at least one maximal unit term of ϕ with respect to UA. Similarly, $SOD_\phi(UA)$ 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 $SOD_\phi(UA)$ accepts a trace i that contains no admin event and reaches a final state, then $\mathsf{users}(i) \models^{\mathsf{M}}_{UA} \phi$.

4.4 SoDA Trace Semantics

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

Example 3 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 $\phi =$ Pharmacist $\odot \neg$Pharmacist and consider the trace

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

for two arbitrary tasks t_1 and t_2. From (R1)–(R3), it follows that $SOD_\phi(\varnothing)$ must accept i. By (R1), $SOD_\phi(\varnothing)$ engages in add.Alice.Pharmacist and afterwards behaves like $SOD_\phi(UA)$, for $UA = \{(\mathsf{Alice}, \mathsf{Pharmacist})\}$. Next, $SOD_\phi(UA)$ engages in t_1.Alice by (R2) and (R3) because Alice acts as a Pharmacist. Again by (R1), $SOD_\phi(UA)$ engages in rm.Alice.Pharmacist and afterwards behaves like $SOD_\phi(\varnothing)$. Finally, by (R2) and (R3), $SOD_\phi(\varnothing)$ engages in t_2.Alice because Alice does not act as a Pharmacist. In the end, SOD_ϕ 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-assignment relation 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 8 *Let $i, i_1, i_2 \in (\mathscr{X} \cup \mathscr{A})^*$ 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 $\mathscr{A}$, x over $\mathscr{X}$, and i', i'_1, and i'_2 over $(\mathscr{X} \cup \mathscr{A})^*$.

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

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

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

$$(7)\quad \frac{i \models^{\mathsf{T}}_{UA} \phi}{i \models^{\mathsf{T}}_{UA} \phi \sqcup \psi} \qquad (8)\quad \frac{i \models^{\mathsf{T}}_{UA} \psi}{i \models^{\mathsf{T}}_{UA} \phi \sqcup \psi} \qquad (9)\quad \frac{i \models^{\mathsf{T}}_{UA} \phi\ ,\ i \models^{\mathsf{T}}_{UA} \psi}{i \models^{\mathsf{T}}_{UA} \phi \sqcap \psi}$$

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

$$(11)\quad \frac{i_1 \models^{\mathsf{T}}_{UA} \phi\ ,\ i_2 \models^{\mathsf{T}}_{UA} \psi}{i \models^{\mathsf{T}}_{UA} \phi \otimes \psi}\ \mathsf{si}(i,i_1,i_2)\ \textit{and}\ \mathsf{users}(i_1)\cap\mathsf{users}(i_2)=\varnothing\ .$$

We say that *i satisfies* ϕ *with respect to UA*, if $i \models^{\mathsf{T}}_{UA} \phi$. SoDA$^{\mathsf{T}}$ fulfills the requirements of Section 4.3: (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.2.2.

Lemma 2 *For all terms* ϕ*, all user-assignment relations UA, and all traces* $i \in \mathscr{X}^*$*, if* $i \models^{\mathsf{T}}_{UA} \phi$*, then* $\mathsf{users}(i) \models^{\mathsf{M}}_{UA} \phi$.

Consider again the trace i and the term ϕ from Example 3. It is straightforward to see that i satisfies ϕ with respect to $UA = \varnothing$. Hence, SoDA$^{\mathsf{T}}$ overcomes the limitations of SoDA$^{\mathsf{M}}$ illustrated in Example 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-assignment relations, 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 its correctness with respect to SoDA$^{\mathsf{T}}$.

4.5 Mapping Terms to Processes

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

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

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

Definition 10 *Given a set of users U, a user-assignment relation UA, and a term* ϕ*, the* mapping $[\![\phi]\!]^{U}_{UA}$ *returns a process parametrized by UA. For a unit term* ϕ_{ut} *and terms* ϕ *and* ψ*, the*

mapping $[\![.]\!]^U_{UA}$ *is defined as follows.*

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

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

Definition 11 *For a term* ϕ *and a user-assignment relation UA, the* SoD-enforcement process $SOD_\phi(UA)$ *is the process* $[\![\phi]\!]^{\mathscr{U}}_{UA}$.

Before we show how an SoD-enforcement process is used together with workflow processes and the RBAC process, we define correctness for the mapping $[\![.]\!]^U_{UA}$.

Definition 12 *The mapping* $[\![.]\!]^U_{UA}$ *is* correct *if for all terms* ϕ*, all user-assignment relations UA, and all traces* $i \in \Sigma^*$*,* $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_\phi(UA))$ *if and only if* $i \models^{\mathsf{T}}_{UA} \phi$.

Informally, the mapping $[\![.]\!]^U_{UA}$ is correct if the following properties hold for all SoD-enforcement processes SOD_ϕ: (1) if SOD_ϕ accepts a workflow trace that corresponds to a successfully terminated workflow instance, then its prefix excluding ✓ satisfies ϕ under SoDA$^{\mathsf{T}}$, and (2) if a workflow trace satisfies ϕ under SoDA$^{\mathsf{T}}$, then its extension by ✓ corresponds to a successfully terminated workflow instance and is accepted by SOD_ϕ. We prove Theorem 1 in Appendix A.2.3.

Theorem 1 *The mapping* $[\![.]\!]^U_{UA}$ *is correct.*

Hence, if the SoD-enforcement process accepts a successfully terminated workflow instance, then the corresponding SoD constraint is satisfied. We also know that no compliant workflow instance is falsely blocked by the SoD-enforcement process. The following corollary relates the set of traces of SoD-enforcement processes without administrative events and their corresponding multisets of users under the multiset semantics. Its proof follows directly from Theorem 1 and Lemma 2.

Corollary 1 *For all terms ϕ, all user-assignment relations UA, and all traces $i \in \mathscr{X}^*$, if $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_\phi(UA))$, then* $\mathsf{users}(i) \models^{\mathsf{M}}_{UA} \phi$.

Given a workflow process W and a term ϕ that models an SoD policy, the *SoD-secure (workflow) process* SSW_ϕ is the parallel, partially synchronized composition of W, the RBAC process, and the SoD-enforcement process SOD_ϕ, parametrized by a user-assignment relation UA and a permission-assignment relation PA.

$$SSW_\phi(UA,PA) = W \underset{\mathscr{X}}{\parallel} (RBAC(UA,PA) \parallel SOD_\phi(UA))$$

Let $x = t.u$ be an execution event, for a task t and a user u. $SSW_\phi(UA,PA)$ engages x if W, $RBAC(UA,PA)$, and $SOD_\phi(UA)$ 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 according to the RBAC configuration (UA,PA), and u must not violate the SoD policy ϕ, given the previously executed execution events and UA. Furthermore, $RBAC$ and SOD_ϕ can synchronously engage in an administrative event at any time. Finally, $SSW_\phi(UA,PA)$ engages in $\checkmark$ if W, $RBAC$, and $SOD_\phi(UA)$ synchronously engage in $\checkmark$.

Example 4 We return to our case study. Consider again the drug dispensation workflow, modeled by the workflow process W, the user-assignment relations UA_1, UA_2, and UA_3, and the permission-assignment relation PA, introduced in Section 3.3. Furthermore, recall the workflow's SoD policy introduced in Example 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 Section 3.3 that $i_2 = \langle t_1.\mathsf{Fritz}, t_2.\mathsf{Emma}, \mathsf{add.Fritz.PrivacyAdvocate}, t_3.\mathsf{Fritz}, t_5.\mathsf{Bob}\rangle$ is a trace of $SW(UA_1,PA)$. However, i_2 is not a trace of the SoD-secure process $SSW_\phi(UA_1,PA)$ because i_2 is not a trace of $SOD_\phi(UA_1)$. In particular, when Fritz executes t_1 in i_2 he acts only in the role Patient and when he later executes t_3 he acts as a Patient and a PrivacyAdvocate. By Definition 10 and 11, $SOD_\phi(UA_1)$ 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 $SOD_\phi(UA_1)$ engages in further execution events, but Fritz does not satisfy the respective subterms. Hence, $SOD_\phi(UA_1)$ accepts i_2's prefix $\langle t_1.\mathsf{Fritz}, t_2.\mathsf{Emma}, \mathsf{add.Fritz.PrivacyAdvocate}\rangle$ but does not engage in t_3.Fritz afterwards.

In contrast, $SOD_\phi(UA_1)$ accepts $i_3 = \langle t_1.\mathsf{Dave},\ t_2.\mathsf{Emma},\ \mathsf{add.Fritz.PrivacyAdvocate},\ t_3.\mathsf{Fritz},\ t_5.\mathsf{Bob}, \mathsf{add.Alice.Pharmacist}, t_7.\mathsf{Alice}, t_9.\mathsf{Gerda}, t_{10}.\mathsf{Gerda}, \checkmark\rangle$. By Theorem 1 and because $\checkmark \in i_3$ we have that $(i_3 \restriction \Sigma) \models^{\mathsf{T}}_{UA_1} \phi$. In other words, i_3 models a workflow instance that satisfies the drug dispensation workflow's SoD policy. Furthermore, by engaging in the first administrative event in i_3, the user-assignment relation changes to UA_2 and after engaging in the second administrative event it becomes UA_3. Thereby, every task execution is authorized and $RBAC(UA_1,PA)$ accepts i_3 as well. Finally, it is easy to see that $i_3 \restriction \mathscr{X} \in \mathsf{T}(W)$, *i.e.* i_3 is also a workflow trace of W. As a result, $i_3 \in \mathsf{T}(SSW_\phi(UA_1,PA))$. ◇

This example illustrates how the three kinds of processes presented in this article interact and how each of them enforces its corresponding specification: W formalizes the workflow model, $RBAC$ formalizes a possibly changing access control policy, and $SOD_\phi(UA)$ formalizes the SoD policy, while accounting for changing role assignments.

We have now completed our formal models and are ready to map them to a software implementation in the next section. We return to the case study in Section 5.5, when we measure the performance of our implementation.

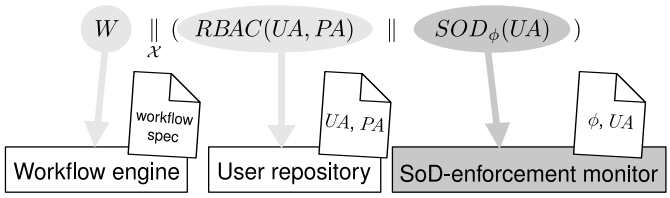


Figure 3: From theory to practice: Mapping processes to software components

5 Implementation

In the following, 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 SoD-secure process as blueprint for our implementation. Its subprocesses naturally map to components of a SOA as illustrated in Figure 3. 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 SOD_ϕ by a program called an *SoD-enforcement monitor*. Workflow engines and user repositories are well-established concepts and we therefore realize them using off-the-shelf components. The standalone SoD-enforcement monitor, however, is something fundamentally new. Hence, we implemented it from scratch. In Figure 3 and further illustrations we use dark gray to identify newly developed components.

5.1 Technical Objectives

We aim at realizing 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 changing RBAC configurations, 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 pre-existing components, such as a legacy workflow system. Furthermore, the system should be configurable using standard means, *e.g.* a workflow definition, an RBAC configuration, 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 runtime of a system with a workflow engine and a user repository, but without an SoD-enforcement monitor, the *runtime baseline*. Our objective is to enforce SoD constraints efficiently, that is with a low overhead compared to the runtime baseline.

5.2 Architecture

As defined in Section 4.5, an SoD-secure process is the parallel, partially synchronized execution of three subprocesses, 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

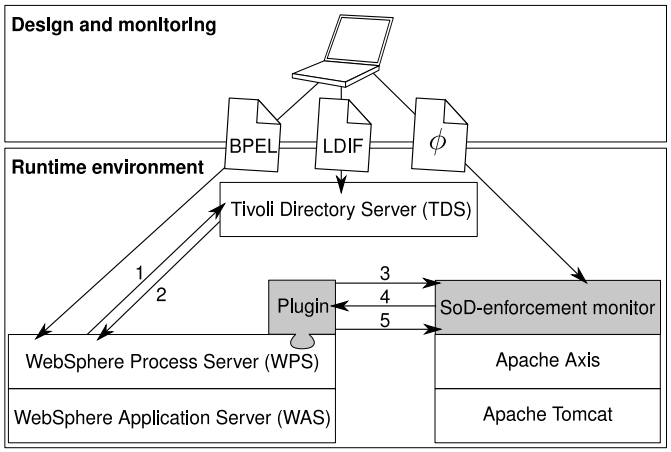


Figure 4: Architecture

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, assume a system comes with a workflow engine and an access control enforcement monitor. In this case, it is sufficient to generate an enforcement monitor for the SoD-enforcement process and to synchronize all business and administrative events with the existing components.

Figure 3 shows our general approach of mapping W, $RBAC$, and SOD_ϕ to individual system components. The concrete software tools we use and their intercommunication are illustrated in Figure 4. Ignore the arrows and labels for the moment.

Workflow engine We use the IBM WebSphere Process Server (WPS) [WPS] as a workflow engine. WPS runs on top of the IBM WebSphere Application Server (WAS) [WAS], which is IBM's Java EE application server.

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

SoD enforcement monitor We implemented the SOD-enforcement monitor in Java and wrapped it as a web service, using Apache Axis [AA09] running on top of Apache Tomcat.

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) [AAA+07], or BPEL for short, is a popular standard for describing business processes at the implementation-level. A BPEL process 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 [AAD+07] 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 $\mathscr{U}$, $\mathscr{R}$, and UA 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. We send them to the SoD-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 SoD-enforcement monitor. Hence, we achieve the flexibility described in Section 5.1.

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 SoD-enforcement monitor requires context information, which must be sent across the network. Our design decisions in this regard are explained in Section 6.3 and the performance analysis in Section 5.5 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 [And05] proposes sending the hierarchy, based on which an access control 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.

5.3 Enforcement

Our prototype system implements an SoD-secure workflow process SSW_ϕ as follows. The SSW_ϕ process engages in three kinds of events: execution events, administrative events, and the event $\checkmark$. The implementation and handling of administrative events and the event $\checkmark$ 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 SoD-secure workflow process

$$SSW_\phi(UA, PA) = (W \underset{\mathscr{X}}{\|} RBAC(UA, PA)) \underset{\Sigma}{\|} SOD_\phi(UA),$$

for a SoDA term ϕ, an RBAC configuration (UA, PA), and a workflow process W that models a workflow w. Assume that $i \in \mathsf{T}(SSW_\phi(UA, PA))$ corresponds to an unfinished workflow instance of w. Let UA' be the user-assignment relation after executing the administrative events in i. Assume that t is the next task of w that is 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 n in Figure 4 as An.

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 creation of i itself.

2. **RBAC Authorization:** In SSW_ϕ, authorization decisions are made by the *RBAC* and the SOD_ϕ process and W simply defines the order in which tasks must be executed. This is handled differently in most commercial workflow systems, including ours. For example, BPEL4People requires the definition of a query, called *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 $RBAC(UA', PA)$ accepts the execution event $t.u$ if u is assigned to one of the roles in $R_t = \{r \mid (r,t) \in PA\}$ according to UA'. 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-assignment relation UA and the workflow definition specifies the permission-assignment relation PA. Implicitly, we assume a one-to-one relation between permissions and tasks.

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

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

   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 $UA'_1 = \{(u,r) \in UA' \mid u \in U_1\}$, and the identifiers of w and i to the SoD-enforcement monitor (A3). We refer to this web service call as a *refinement call*.

   For every workflow, the SoD-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 SoD-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 $u \in U_2$, $i\hat{}\langle t.u\rangle \in \mathsf{T}(SSW_\phi(UA, PA))$. Therefore, WPS displays x 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 x by clicking on x in his inbox. Assume the user u claims x, *i.e.* in CSP x corresponds to the execution event $t.u$. Instantaneously, x is removed from the inboxes of all other users. At this point, we must communicate to the SoD-enforcement monitor that u is executing x. 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:** Afterwards, u is prompted with a form whose completion constitutes the work associated with x. The work is completed when the form is submitted. If x is not a task instance that terminates the workflow instance, its termination triggers the instantiation of at least one other task.

Summarizing, our system effectively enforces abstract SoD constraints as specified in Section 5.1. Arbitrary workflows, constrained by a possibly changing RBAC configuration and an abstract SoD policy, can be executed on WPS. The practicability of our approach is further supported by performance measurements for our case study in Section 5.5. However, we first examine its runtime complexity.

5.4 Complexity

When analyzing the runtime complexity of our SoD-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 [LW08]. The SoD-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, UA'_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 SoD-enforcement monitor computes whether $i \hat{} \langle t.u \rangle \in \mathsf{T}(SOD_\phi(UA'_1))$. 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 $|\mathscr{U}|$ and $|\mathscr{R}|$; *i.e.* $p(|\mathscr{U}|, |\mathscr{R}|)$ for a polynomial p. In the worst case, $SOD_\phi(UA'_1)$ branches $2^{|\mathscr{U}|}$ times per operator in ϕ. If m is the number of operators, the worst-case runtime is therefore in $O(nm2^{|\mathscr{U}|}\, p(|\mathscr{U}|, |\mathscr{R}|))$.

The exponential factor originates from the $\otimes$-operator, which causes $SOD_\phi(UA'_1)$ to branch for all disjoint subsets of $\mathscr{U}$. 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 $i \hat{} \langle t.u \rangle \in \mathsf{T}(SOD_\phi(UA'_1))$, the users in $\mathscr{U} \setminus U_{i+u}$ are not relevant. We therefore need not branch over all partitions of $\mathscr{U}$ 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 $|U_{i+u}| \leq |\mathscr{U}|$. Our implementation exploits these observations. Hence, its runtime complexity is in $O(nm2^{|U_{i+u}|}\, p(|\mathscr{U}|, |\mathscr{R}|))$ for $|U_{i+u}|$ as discussed above.

Our experience with business process catalogs, such as the IBM Insurance Application Architecture (IAA) [IAA], is that workflows contain a good dozen human tasks on the average. Furthermore, most workflow 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.

5.5 Performance Measurements

We return to the drug dispensation workflow introduced in Section 3.3, the term ϕ from Example 1, and the trace $i_3 = \langle t_1.\mathsf{Dave}, t_2.\mathsf{Emma}, \mathsf{add}.\mathsf{Fritz}.\mathsf{PrivacyAdvocate},\ t_3.\mathsf{Fritz}, t_5.\mathsf{Bob}, \mathsf{add}.\mathsf{Alice}.\mathsf{Pharmacist}, t_7.\mathsf{Alice}, t_9.\mathsf{Gerda}, t_{10}.\mathsf{Gerda}, \checkmark \rangle$ from Example 4.

We modeled the workflow in BPEL, extended by BPEL4People, and deployed it on WPS. We set up the initial user-assignment relation UA_1 using TDS' web interface and deployed ϕ, encoded as string, to the SoD-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

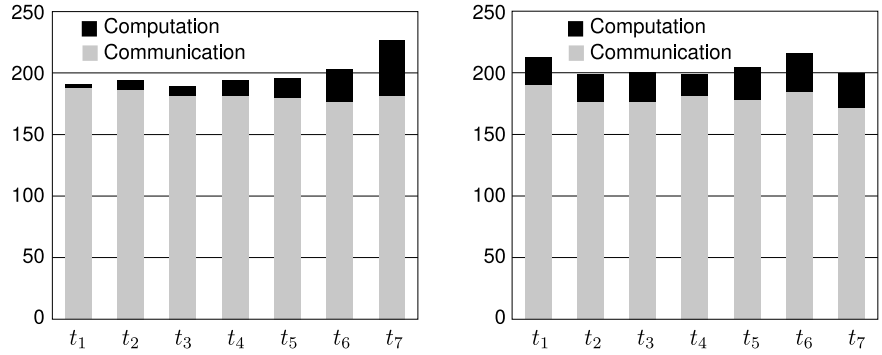


Figure 5: Average service call times in ms

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, UA_1 evolved to UA_2. Afterwards, we executed t_3 (Check Anonymization Requirements) as Fritz. This sequence of activities corresponds to a prefix of i_3. In the following, we report on the average performance of ten executions of workflow instances corresponding to i_3.

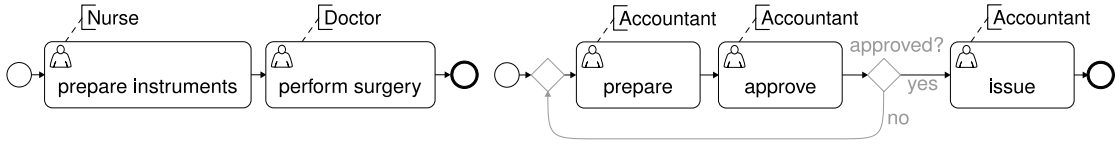
Compared to the runtime baseline, the runtime of our prototype system is increased by a refinement and a claim call for every task instance. We call the time it takes to call a web service and to retrieve its return values the *total runtime* of a web service. We decompose this runtime into two parts: 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 5 illustrates the averaged communication and computation times in milliseconds per task.

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 SoD-enforcement monitor on two different computers at the same geographical location, connected by a standard enterprise network with an average latency of 1ms. Both computers have off-the-shelf configurations.[1] The communication time averages between 150ms and 200ms per call.

The computation time for claim calls was always around 24ms. The computation time of refinement calls, however, increased with the number of executed task instances. As shown in Section 5.4, the operators in ϕ cause this time to increase exponentially.

Finally, we compare the total runtime of these additional calls to the time it takes to execute a task instance in a system without an SoD-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 runtime caused by the SoD-enforcement monitor calls is an order of magnitude smaller than the runtime baseline, which varied between 2 and 5 seconds.

[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.



(a) Medical workflow

(b) Payment workflow

Figure 6: Example workflows for evaluation

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 5.1.

6 Evaluation and Future Work

6.1 Limitations of an Automated Mapping

As elaborated above, 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 6 to evaluate our generalization of SoDA$^{\mathsf{S}}$ to SoDA$^{\mathsf{T}}$, our automated mapping to processes, their shortcomings, and explore directions for future work.

Consider the medical workflow shown in Figure 6a and assume that we want to enforce the SoD constraint $\mathsf{Nurse} \otimes \mathsf{Doctor}$, which does not specify whether the Doctor performs the surgery and the Nurse prepares the instruments or *vice versa*. We tackle this problem by incorporating an RBAC configuration into the SoD-secure process. For this medical workflow, the depicted permission-assignment relation rules out workflow instances where the Nurse performs the surgery and the Doctor prepares the instruments.

However, incorporating an RBAC configuration does not solve all refinement questions related to a mapping from terms to processes. In particular when duties are to be separated between tasks assigned to the same users, an RBAC configuration is of little help. Consider the payment workflow in Figure 6b, 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 SoD-secure process. A semi-automated approach where subterms are manually mapped to tasks would solve this problem. We have considered only fully automated mappings in this article and leave semi-automated solutions 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 6b, 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 SoD-enforcement processes that accept an arbitrary number of execution events. By SoDA's syntax $\mathfrak{S}$ however, the $^{+}$-operator only ranges over unit terms. [LW08] motivate this design decisions with the "psychological acceptability principle"

postulated by [SS75]. 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. Our recently introduced concept of *release* [BBK11b] supports a scoping of authorization constraints and is therefore a candidate solution for manually decomposing workflows and refining the subterm-task mapping.

6.2 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 $i \hat{} i'$ 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 $\checkmark$ is ϕ supposed to be satisfied. This is also reflected in Theorem 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. Wang and Li made some initial contributions in this direction in [WL07]. However, their solution is limited to a subset of SoDA terms and does not consider administrative activities that may change the underlying RBAC configuration during enforcement.

6.3 Design Decisions: Communication Versus Statefulness

An SoD-enforcement process $SOD_\phi(UA)$ is parametrized by the user-assignment relation UA that is modified when the process engages in administrative events. Our SoD-enforcement monitor, however, does not store all tuples of UA. 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 SoD-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.

Our SoD-enforcement monitor is stateful because the enforcement of SoD constraints ranges over multiple tasks and may depend on user-assignment relations. The service must therefore 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 SoD-enforcement monitor; the workflow engine and the user repository remain unchanged.

6.4 Abstractions

For simplicity, our SoD-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.

7 Related Work

The fundamental ideas pursued in this article are in line with the notion of *Model-driven Security (MDS)* [BDL06] that aims at synthesizing a system's implementation from the composition of abstract specifications of its business and security requirements. In particular, Basin, Doser, and Lodderstedt generate implementations of workflow systems that include access control requirements in [BDL03]. In contrast, we focus specifically on SoD constraints and our implementation is an integration of heterogeneous software components as opposed to an automatically generated, monolithic software application.

Our concept of an SoD-enforcement monitor can be seen as an instance of Schneider's *security automata* [Sch00] in that it is also composed with an insecure system and checks whether commands are authorized prior to their execution. To the best of our knowledge, Basin, Olderog, and Sevinç [BOS07] were the first to use CSP to formalize security automata. Like them, we encode what is widely known as *policy decision point (PDP)* as CSP process and the synchronous process composition constitutes the *policy enforcement point (PEP)*.

There are numerous models and frameworks to formalize and enforce SoD constraints [GGF98,SZ97]. Static SoD enforces SoD constraints at design time, *e.g.* by ensuring that no user is assigned to two conflicting tasks. In contrast, dynamic SoD is enforced at run time and is more flexible than static enforcement. Therefore, dynamic SoD is more interesting for real-world settings although it is in general more complex than static SoD. Our work is the first to model dynamic enforcement of SoD constraints with changing role assignments.

Most SoD mechanisms describe and enforce constraints between two or more explicit tasks and are therefore tightly coupled with a workflow definition [San88,BFA99,KS02]. In contrast, our approach allows a workflow-independent specification of SoD constraints and their enforcement on different workflows. This has the advantages presented in Section 1 but poses the challenges elaborated in Section 6.4.

In more detail, in his seminal paper [San88] Sandhu introduces *transaction control expressions* for specifying dynamic SoD constraints on data objects. Enforcement decisions are made at run-time, based on the history of executed tasks. 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. Constraints are less expressive than SoDA terms and they can only be defined in combination with a concrete workflow.

Bertino, Ferrari, and Atluri [BFA99] check the consistency of constraints defined over workflows in a logical framework, often referred to as *BFA model*. Their constraints are defined with respect to sequences of tasks that model workflows, applying (first-order) predicates to task occurrences. Schaad, Lotz, and Sohr extend SoD analysis to workflows with dynamic access rights in [SLS06]. They describe the workflow, the associated access control policy, and the delegation and revocation steps as transitions of a finite state automaton and apply model checking to verify the constraints expressed in linear temporal logic. However, neither of these papers provide a mapping to an enforcement mechanism.

Knorr and Stormer [KS02] map dynamic SoD constraints along with basic workflow models to Prolog clauses in order to compute all workflow instances that do not violate the specified SoD constraints. In Nash and Poland's *object-based separation of duties* [NP90], 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. This functionality can be modeled with our formalism if every data object is protected by an SoD-enforcement process.

There are many languages for modeling workflows. We used BPMN [BPMN2] for visualizing the drug dispensation workflow and BPEL [AAA+07], including its human task extension BPEL4People [AAD+07], to implement it in WPS. BPMN and BPEL are well-established whereas BPEL4People is only supported by a few workflow engines. The formalization of these modeling languages with a process calculus is commonplace, *e.g.* [WG08] map BPMN to CSP and [CCCV06] map BPEL to CCS.

Although not fully specified, the query language for people links in BPEL4People allows one to specify basic SoD constraints. By using SoDA, our architecture supports more expressive constraints. Paci, Bertino, and Crampton [PBC08] propose another access control extension for BPEL based on earlier work of Crampton [Cra05]. Authorizations, including SoD constraints, are enforced by a web service, which pools all information that is relevant for enforcement: the history of workflow instances, the RBAC configuration, and SoD constraints. The underlying workflow model, however, does not support loops, which is in conflict with the expressiveness of BPEL. Moreover, unlike our work, their constraint language requires a tight coupling between constraints and the workflow definition and does not support changing authorizations.

8 Conclusions

This work is motivated by internal threats, regulations, and best-practice frameworks that call for internal controls. Furthermore, it addresses the characteristics of today's dynamic, heterogeneous, component-based enterprise IT architectures that are typically integrated through SOAs. One of our central assumptions is that a workflow-independent, abstract policy language for SoD constraints facilitates a separation of concerns between business and security personnel. It thereby enables a higher degree of flexibility with respect to integration and policy enforcement in such dynamic, distributed environments.

Building on SoDA, we bridged the gap between an abstract specification of SoD constraints and their provisioning and enforcement in a service-oriented workflow environment. The key ideas were (1) to generalize SoDA's semantics to traces and to describe a mapping from terms to CSP processes that accept the respective traces, (2) building on our CSP formalization, to introduce the paradigm of SoD as a Service, which enables the dynamic integration and configuration of SoD enforcement in a SOA, and (3) to make our formalization and implementation account for changing authorizations. 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.

We analyzed the runtime complexity of our enforcement monitor and explained why its runtime performance is acceptable for the workflows used in practice. Furthermore, we experimentally validated our approach with a realistic case study. The performance measurements made in this study support the conclusions of our complexity analysis.

References

[SO02] SOX 2002. Sarbanes-Oxley Act of 2002. United States Government Printing Office.

[AAD+07] AGRAWAL, A., et al. 2007. WS-BPEL extension for people (BPEL4People), v1.0.

[AAA+07] ALVES, A., et al. 2007. Web services business process execution language (BPEL), v2.0. OASIS Standard.

[And05] ANDERSON, A. 2005. Hierarchical resource profile of XACML, v 2.0. OASIS Standard.

[AA09] APACHE, 2009. Apache Axis2, v1.5.1, `http://ws.apache.org/axis2`.

[BBK09] 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)*, Saint-Malo, France, M. BACKES AND P. NING, Eds. Springer-Verlag, Berlin, Germany, LNCS 5789, 250–267.

[BBK11a] 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 '07)*, Hong Kong, China. ACM Press, New York, NY, 423–429.

[BBK11b] BASIN, D., BURRI, S. J., AND KARJOTH, G. 2011b. Obstruction-free authorization enforcement: Aligning security with business objectives. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium (CSF '11)*, Cernay-la-Ville, France. IEEE Computer Society Press, Los Alamitos, CA, 99–113.

[BDL06] 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) 15,* 1, 39–91.

[BDL03] 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)*, Como, Italy. ACM Press, New York, NY, 100–109.

[BOS07] 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)*, Singapore. ACM Press, New York, NY, 70–81.

[BFA99] 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) 2,* 1, 65–104.

[CCCV06] CÁMARA, J., CANAL, C., CUBO, J., AND VALLECILLO, A. 2006. Formalizing WSBPEL business processes using process algebra. *Electronic Notes in Theoretical Computer Science (ENTCS) 154,* 1, 159–173.

[Cra05] 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)*, Stockholm, Sweden. ACM Press, New York, NY, 38–47.

[Eco01] THE ECONOMIST 2001. Enron, see you in court. The Economist, November 15, 2001.

[EY09] ERNEST & YOUNG 2009. European fraud survey 2009 – Is integrity a casualty of the downturn?. Technical Report, Ernest & Young.

[FSG+01] 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) 4,* 3, 224–274.

[GGF98] 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 19th IEEE Symposium on Security and Privacy (S&P '98)*. Oakland, CA, USA, May 1998. IEEE Computer Society Press, Los Alamitos, CA, 172–183.

[IAA] IBM Insurance Application Architecture (IAA). `www.ibm.com/software/sw-library/en_US/detail/N440171L95655L23.html`.

[TDS] IBM Tivoli Directory Server (TDS), v 6. `www.ibm.com/software/tivoli/products/directory-server`.

[WAS] IBM WebSphere Application Server (WAS), v6.1. `www.ibm.com/software/webservers/appserv/was`.

[WPS] IBM WebSphere Process Server (WPS), v 6.2. `www.ibm.com/software/integration/wps`.

[KS02] KNORR, K. AND STORMER, H. 2002. Modeling and analyzing separation of duties in workflow environments. *International Federation for Information Processing (IFIP) 65*, 199–212.

[LW08] LI, N. AND WANG, Q. 2008. Beyond separation of duty: An algebra for specifying high-level security policies. *Journal of the ACM 55,* 3.

[MPH+09] 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. Deliverable of FP7 EU Project MASTER.

[NP90] 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)*. Oakland, CA, USA, 1990. IEEE Computer Society Press, Los Alamitos, CA, 201–207.

[BPMN2] OMG 2011. Business process model and notation (BPMN), v2.0. Object Management Group (OMG) Standard.

[PBC08] PACI, F., BERTINO, E., AND CRAMPTON, J. 2008. An access-control framework for WS-BPEL. *International Journal of Web Services Research (JWSR). 5,* 3, 20–43.

[Ros97] ROSCOE, A. W. 1997. *The theory and practice of concurrency*. Prentice Hall, Upper Saddle River, NJ, USA.

[SS75] SALTZER, J. AND SCHROEDER, M. 1975. The protection of information in computer systems. *Proceeding of the IEEE 63,* 9, 1278–1308.

[San88] SANDHU, R. S. 1988. Transaction control expressions for separation of duties. In *Proceedings of the 4th IEEE Aerospace Computer Security Applications Conference*. IEEE Computer Society Press, Los Alamitos, CA, 282–286.

[SLS06] SCHAAD, A., LOTZ, V., AND SOHR, K. 2006. A model-checking approach to analysing organisational controls in a loan origination procbess. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT '06)*. ACM Press, New York, NY, 139–149.

[Sch00] SCHNEIDER, F. B. 2000. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC) 3,* 1, 30–50.

[SZ97] 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)*. IEEE Computer Society Press, Los Alamitos, CA, 183–194.

[Syr00] SYROPOULOS, A. 2000. Mathematics of multisets. In *Workshop on Multiset Processing (WMP '00)*. Curtea de Arges, Romania, C. S. CALUDE ET AL., Eds. Springer, Berlin, Germany, LNCS 2235, 347–358.

[TBB03] TURNER, M., BUDGEN, D., AND BRERETON, P. 2003. Turning software into a service. *Computer 36*, 38–44.

[WL07] WANG, Q. AND LI, N. 2007. Direct static enforcement of high-level security policies. In *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security (ASIACCS '07)*. ACM Press, New York, NY, 214–225.

[WG08] 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)*, Kitakyushu-City, Japan, S. LIU, T. MAIBAUM,AND K. ARAKI, Eds. Springer, Berlin, Germany, LNCS 5256, 355–374.

APPENDIX

A.1 Li and Wang's set semantics for SoDA

We summarize SoDA$^{\mathsf{S}}$, the semantics for SoDA terms that was originally introduced by Li and Wang in [LW08]. They define satisfaction with respect to a tuple (U, UR), where $U \subseteq \mathscr{U}$ and $UR \subseteq U \times \mathscr{R}$. In contrast, we defined multiset satisfaction SoDA$^{\mathsf{M}}$ simply with respect to the user-assignment relation UA.

Definition 13 *Let S be a non-empty set of users and $r \in \mathscr{R}$ a role. Furthermore, let U be a set of users and $UR \subseteq U \times \mathscr{R}$ a user-assignment relation. For two sets of users X and Y, and a term ϕ,* set satisfiability *is the smallest relation between two sets of users, user-assignment relations, and terms, written $X \models^{\mathsf{S}}_{(U,UR)} \phi$, closed under the following rules:*

(1) $\dfrac{}{\{u\} \models^{\mathsf{S}}_{(U,UR)} \mathsf{All}} \quad u \in U$

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

(3) $\dfrac{}{\{u\} \models^{\mathsf{S}}_{(U,UR)} S} \quad u \in (S \cap U)$

(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) $\dfrac{\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi,\ X \models^{\mathsf{S}}_{(U,UR)} \phi^{+}}{(\{u\} \cup X) \models^{\mathsf{S}}_{(U,UR)} \phi^{+}}$

(7) $\dfrac{X \models^{\mathsf{S}}_{(U,UR)} \phi}{X \models^{\mathsf{S}}_{(U,UR)} (\phi \sqcup \psi)}$

(8) $\dfrac{X \models^{\mathsf{S}}_{(U,UR)} \psi}{X \models^{\mathsf{S}}_{(U,UR)} (\phi \sqcup \psi)}$

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

(10) $\dfrac{X \models^{\mathsf{S}}_{(U,UR)} \phi,\ Y \models^{\mathsf{S}}_{(U,UR)} \psi}{(X \cup Y) \models^{\mathsf{S}}_{(U,UR)} (\phi \odot \psi)}$

(11) $\dfrac{X \models^{\mathsf{S}}_{(U,UR)} \phi,\ Y \models^{\mathsf{S}}_{(U,UR)} \psi}{(X \cup Y) \models^{\mathsf{S}}_{(U,UR)} (\phi \otimes \psi)} \quad (X \cap Y) = \varnothing\,.$

A.2 Proofs

In the following, we refer to rule i of Definition 6 as (MUi), to rule j of Definition 9 as (TRj), to rule k of Definition 10 as (MAk), and to rule l of Definition 13 as (SEl), respectively. 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.2.1 Proof of Lemma 1

We first establish two auxiliary propositions and then prove Lemma 1.

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

Proof of Proposition 1. The only rules of Definition 6 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 2 *For all unit terms ϕ_{ut}, all user-assignment relations UA, and all users $u \in \mathscr{U}$, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \phi_{ut}$ if and only if $\{u\} \models^{\mathsf{S}}_{\mathsf{lwconf}(UA)} \phi_{ut}$.*

Proof of Proposition 2. We reason by induction on the structure of ϕ_{ut}. The only rules of Definition 6 that allow for the derivation of a unit term are $(MU1)$–$(MU4)$ and $(MU7)$–$(MU9)$. All other rules can be safely ignored. Let UA and u be given and let $(U, UR) = \mathsf{lwconf}(UA)$.

$LHS \Rightarrow RHS$:

Base cases: Consider the term All and let $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \mathsf{All}$. By $(MU1)$, there exists an $r \in \mathscr{R}$ such that $(u,r) \in UA$. 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 $r \in \mathscr{R}$, and let $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} r$. From $(MU2)$ it follows that $(u,r) \in UA$. By the definition of lwconf, $(u,r) \in UR$ and therefore, $\{u\} \models^{\mathsf{S}}_{(U,UR)} r$ by $(SE2)$.

Consider a term of the form S, for $S \subseteq \mathscr{U}$, and let $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} S$. By $(MU3)$, $u \in S$ and there exists an $r \in \mathscr{R}$ such that $(u,r) \in UA$. 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 2 holds for two unit terms ϕ_{ut} and ψ_{ut}. Consider now the term $\neg\phi_{ut}$ and let $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \neg\phi_{ut}$. By $(MU4)$, $\{u\} \not\models^{\mathsf{M}}_{UA} \phi_{ut}$. 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 $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \phi_{ut} \sqcup \psi_{ut}$. By $(MU7)$ and $(MU8)$, either $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \phi_{ut}$ or $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \psi_{ut}$. 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 $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \phi_{ut} \sqcap \psi_{ut}$. By $(MU9)$, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \phi_{ut}$ and $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \psi_{ut}$. 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 $r \in \mathscr{R}$ such that $(u,r) \in UA$, by the definition of lwconf. From $(MU1)$ it follows that $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \mathsf{All}$.

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

Consider a term of the form S, for $S \subseteq \mathscr{U}$, 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 $r \in \mathscr{R}$ such that $(u,r) \in UA$. By $(MU3)$, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} S$.

Step cases: Assume that Proposition 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 $\{u\} \not\models^{\mathsf{M}}_{UA} \phi_{ut}$. Therefore, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \neg\phi_{ut}$ 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, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \phi_{ut}$ and therefore $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \phi_{ut} \sqcup \psi_{ut}$ by $(MU7)$. The second case is analogous. Hence, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \phi_{ut} \sqcup \psi_{ut}$.

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, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \phi_{ut}$ and $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \psi_{ut}$. Therefore, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \phi_{ut} \sqcap \psi_{ut}$ by $(MU9)$. ■

Proof of Lemma 1. Assume an arbitrary user-assignment relation UA and a multiset of users $\mathbf{U}$. Let $(U,UR) = \mathsf{lwconf}(UA)$. We reason inductively over the structure of SoDA terms.

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

Step cases: Assume that Lemma 1 holds for two terms ϕ and ψ. Consider now the term ϕ^+ and let $\mathbf{U} \models^{\mathsf{M}}_{UA} \phi^+$. Let $X = \mathsf{userset}(\mathbf{U})$. From $(MU5)$ and $(MU6)$ follows that for every user $u \in \mathbf{U}$, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \phi$. By the induction hypothesis and the definition of $\mathsf{userset}$ follows that for every user $u \in X$, $\{u\} \models^{\mathsf{S}}_{(U,UR)} \phi$. From $(SE5)$ and $(SE6)$ it follows that $X \models^{\mathsf{S}}_{(U,UR)} \phi^+$.

Consider the term $\phi \sqcup \psi$ and let $\mathbf{U} \models^{\mathsf{M}}_{UA} \phi \sqcup \psi$. By $(MU7)$ and $(MU8)$, either $\mathbf{U} \models^{\mathsf{M}}_{UA} \phi$ or $\mathbf{U} \models^{\mathsf{M}}_{UA} \psi$. 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 $\mathbf{U} \models^{\mathsf{M}}_{UA} \phi \sqcap \psi$. By $(MU9)$, $\mathbf{U} \models^{\mathsf{M}}_{UA} \phi$ and $\mathbf{U} \models^{\mathsf{M}}_{UA} \psi$. 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 $\mathbf{U} \models^{\mathsf{M}}_{UA} \phi \odot \psi$. By $(MU10)$, there are two multisets of users $\mathbf{V}$ and $\mathbf{W}$ such that $\mathbf{V} \models^{\mathsf{M}}_{UA} \phi$ and $\mathbf{W} \models^{\mathsf{M}}_{UA} \psi$. 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 $\mathsf{userset}$, $\mathsf{userset}(\mathbf{U}) = \mathsf{userset}(\mathbf{V}) \cup \mathsf{userset}(\mathbf{W})$. 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 $\mathbf{U} \models^{\mathsf{M}}_{UA} \phi \otimes \psi$. By $(MU11)$, there are two multisets of users $\mathbf{V}$ and $\mathbf{W}$ such that $\mathbf{V} \models^{\mathsf{M}}_{UA} \phi$, $\mathbf{W} \models^{\mathsf{M}}_{UA} \psi$, 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 $\mathsf{userset}$, $\mathsf{userset}(\mathbf{U}) =$

$\mathsf{userset}(\mathbf{V}) \cup \mathsf{userset}(\mathbf{W})$. 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.2.2 Proof of Lemma 2

We first establish two auxiliary propositions and then prove Lemma 2.

Proposition 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 of Proposition 3. By Definition 8, 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 4 *For $i \in \mathscr{X}^*$, $i_1, i_2 \in \Sigma^*$, if $\mathsf{si}(i, i_1, i_2)$ then $i_1 \in \mathscr{X}^*$ and $i_2 \in \mathscr{X}^*$.*

Proof of Proposition 4. By Definition 8, each event that is in i_1 or i_2 is also in i. Since $i \in \mathscr{X}^*$, we therefore have that i_1 and i_2 contain only business events. ■

Proof of Lemma 2. We assume an arbitrary user-assignment relation UA and reason inductively over the structure of SoDA terms.

Base cases: Consider a unit term ϕ_{ut} and a trace $i \in \mathscr{X}^*$. Let $i \models^{\mathsf{T}}_{UA} \phi_{ut}$. The only rules of Definition 9 that may have a unit term in their conclusion are $(TR1)$–$(TR4)$ and $(TR7)$–$(TR9)$. Because i contains no admin events, only $(TR1)$ and $(TR7)$–$(TR9)$ are applicable. In the case of $(TR7)$–$(TR9)$ i is already contained in at least one premise. In a derivation tree for $i \models^{\mathsf{T}}_{UA} \phi_{ut}$, i must therefore be in the conclusion of rule $(TR1)$ by the structure of terms, *i.e.* Definition 5. Therefore, i must be of the form $\langle t.u \rangle$, for an execution event $t.u$. By $(TR1)$, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA} \phi_{ut}$, which is equivalent to $\mathsf{users}(i) \models^{\mathsf{M}}_{UA} \phi_{ut}$ by the definition of users.

Consider a term of the form ϕ^+_{ut} and a trace $i \in \mathscr{X}^*$. Let $i \models^{\mathsf{T}}_{UA} \phi^+_{ut}$. The only rules of Definition 9 that have a term of the form ϕ^+_{ut} in the conclusion are $(TR5)$ and $(TR6)$. For both rules, the trace i must contain at least one execution event x such that $\langle x \rangle \models^{\mathsf{T}}_{UA} \phi_{ut}$. As derived before, $\mathsf{users}(\langle x \rangle) \models^{\mathsf{M}}_{UA} \phi_{ut}$ and therefore, by $(MU5)$, $\mathsf{users}(\langle x \rangle) \models^{\mathsf{M}}_{UA} \phi^+_{ut}$. By induction over the length of i, with $\langle x \rangle$ as the induction basis, it follows that $\mathsf{users}(i) \models^{\mathsf{M}}_{UA} \phi^+_{ut}$ from $(TR6)$ and $(MU6)$.

Step cases: Assume that Lemma 2 holds for two terms ϕ and ψ. Consider now the term $\phi \sqcup \psi$ and a trace $i \in \mathscr{X}^*$. Let $i \models^{\mathsf{T}}_{UA} \phi \sqcup \psi$. By $(TR7)$ and $(TR8)$, either $i \models^{\mathsf{T}}_{UA} \phi$ or $i \models^{\mathsf{T}}_{UA} \psi$. In the first case, by the induction hypothesis, $\mathsf{users}(i) \models^{\mathsf{M}}_{UA} \phi$ and therefore $\mathsf{users}(i) \models^{\mathsf{M}}_{UA} \phi \sqcup \psi$ by $(MU7)$. The second case is analogous. Hence, $\mathsf{users}(i) \models^{\mathsf{M}}_{UA} \phi \sqcup \psi$.

Consider the term $\phi \sqcap \psi$ and a trace $i \in \mathscr{X}^*$. Let $i \models^{\mathsf{T}}_{UA} \phi \sqcap \psi$. By $(TR9)$, $i \models^{\mathsf{T}}_{UA} \phi$ and $i \models^{\mathsf{T}}_{UA} \psi$. From the induction hypothesis, $\mathsf{users}(i) \models^{\mathsf{M}}_{UA} \phi$ and $\mathsf{users}(i) \models^{\mathsf{M}}_{UA} \psi$. Therefore, $\mathsf{users}(i) \models^{\mathsf{M}}_{UA} \phi \sqcap \psi$ by $(MU9)$.

Consider the term $\phi \odot \psi$ and a trace $i \in \mathscr{X}^*$. Let $i \models^{\mathsf{T}}_{UA} \phi \odot \psi$. By $(TR10)$, there exist two traces i_1 and i_2 such that $\mathsf{si}(i, i_1, i_2)$, $i_1 \models^{\mathsf{T}}_{UA} \phi$, and $i_2 \models^{\mathsf{T}}_{UA} \psi$. By Proposition 4, i_1 and i_2 consist only of admin events because $i \in \mathscr{X}^*$. Therefore, from the induction hypothesis, $\mathsf{users}(i_1) \models^{\mathsf{M}}_{UA} \phi$ and $\mathsf{users}(i_2) \models^{\mathsf{M}}_{UA} \psi$. Moreover, by Proposition 3, $\mathsf{users}(i) = \mathsf{users}(i_1) \uplus \mathsf{users}(i_2)$. Hence, $\mathsf{users}(i) \models^{\mathsf{M}}_{UA} \phi \odot \psi$ by $(MU10)$.

Finally, consider the term $\phi \otimes \psi$ and a trace $i \in \mathscr{X}^*$. Let $i \models^{\mathsf{T}}_{UA} \phi \otimes \psi$. By $(TR11)$, there exist two traces i_1 and i_2 such that $\mathsf{si}(i, i_1, i_2)$, $\mathsf{users}(i_1) \cap \mathsf{users}(i_2) = \varnothing$, $i_1 \models^{\mathsf{T}}_{UA} \phi$, and $i_2 \models^{\mathsf{T}}_{UA} \psi$.

By Proposition 4, i_1 and i_2 consist only of admin events because $i \in \mathscr{X}^*$. Therefore, from the induction hypothesis, $\mathsf{users}(i_1) \models^{\mathsf{M}}_{UA} \phi$ and $\mathsf{users}(i_2) \models^{\mathsf{M}}_{UA} \psi$. Furthermore, by Proposition 3, $\mathsf{users}(i) = \mathsf{users}(i_1) \uplus \mathsf{users}(i_2)$. Hence, $\mathsf{users}(i) \models^{\mathsf{M}}_{UA} \phi \otimes \psi$ by $(MU11)$. ■

A.2.3 Proof of Theorem 1

We establish four auxiliary propositions and prove Theorem 1 afterwards. Recall Definition 12. We prove that for all terms ϕ, all user assignments UA, and all traces $i \in \Sigma^*$, $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_\phi(UA))$ if and only if $i \models^{\mathsf{T}}_{UA} \phi$.

Proposition 5 *For a term ϕ, a trace $i \in \Sigma^*$, a trace of admin events $a \in \mathscr{A}^*$, a user-assignment relations UA, and $UA' = \mathsf{upd}(UA, a)$, $i \models^{\mathsf{T}}_{UA'} \phi$ if and only if $a\hat{}i \models^{\mathsf{T}}_{UA} \phi$.*

Proof Sketch of Proposition 5. Proposition 5 follows by induction on a directly from $(TR3)$, $(TR4)$, and Definition 7. □

Proposition 6 *For a user-assignment relations UA, a term ϕ, a trace $i \in \Sigma^*$, and a trace of admin events $a \in \mathscr{A}^*$, $i \models^{\mathsf{T}}_{UA} \phi$ if and only if $i\hat{}a \models^{\mathsf{T}}_{UA} \phi$.*

Proof Sketch of Proposition 6. Proposition 6 follows directly by applying $(TR2)$ for each admin event in a to $t \models^{\mathsf{T}}_{UA} \phi$. □

Proposition 7 *For a user-assignment relations UA, a term ϕ, and a set of users U, the process $[\![\phi]\!]^U_{UA}$ engages only in an execution event $t.u$, for a task t and a user u, if $u \in U$.*

Proof of Proposition 7. Assume a user-assignment relations UA, a term ϕ, and a set of users U. Let $t.u$ be 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 $(MA1)$ and $(MA2)$, $[\![\phi]\!]^U_{UA}$ and $[\![\phi^+]\!]^U_{UA}$ only engage in $t.u$, if $u \in U$. For two terms ϕ and ψ, assume that Proposition 7 holds. By $(MA3)$, $(MA4)$, and $(MA5)$, the processes $[\![\phi \sqcup \psi]\!]^U_{UA}$, $[\![\phi \sqcap \psi]\!]^U_{UA}$, and $[\![\phi \odot \psi]\!]^U_{UA}$ only engage in $t.u$ if either $[\![\phi]\!]^U_{UA}$ or $[\![\psi]\!]^U_{UA}$ engage in $t.u$. By $(MA6)$, the process $[\![\phi \otimes \psi]\!]^U_{UA}$ only engages in $t.u$ if either $[\![\phi]\!]^{U'}_{UA}$ or $[\![\psi]\!]^{U''}_{UA}$ 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 8 *For the traces $i, i_1, i_2 \in \Sigma^*$ and the processes $P, Q \in \mathscr{P}$, if $i_1 \in \mathsf{T}(P)$, $i_2 \in \mathsf{T}(Q)$ and $\mathsf{si}(i, i_1, i_2)$, then $i \in \mathsf{T}(P \underset{\mathscr{A}}{\parallel} Q)$.*

Proof Sketch of Proposition 8. The proof is by induction over i. Proposition 8 follows by the definition of the $||$-operator under the denotational semantics of CSP and by Definition 8. The implicit synchronization on $\checkmark$ can be ignored because i, i_1, and i_2 do not contain $\checkmark$. □

Proof of Theorem 1. We prove that for all terms ϕ, all user-assignment relations UA, 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 UA be given.

$LHS \Rightarrow RHS$:

Base cases: Consider a unit term ϕ_{ut} and let $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_{\phi_{ut}}(UA))$. By $(MA1)$ and the denotational semantics of CSP, i is of the form $a_1\hat{}\langle t.u\rangle\hat{}a_2$, for $a_1, a_2 \in \mathscr{A}^*$, a task t, and a user u. Let $UA' = \mathsf{upd}(UA, a_1)$. Because $[\![\phi_{ut}]\!]^{\mathscr{U}}_{UA'}$ engages in $t.u$, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA'} \phi_{ut}$ by $(MA1)$. From $(TR1)$

it follows that $\langle t.u\rangle \models^{\mathsf{T}}_{UA'} \phi_{ut}$. Therefore, by Proposition 5, $a_1\hat{}\langle t.u\rangle \models^{\mathsf{T}}_{UA} \phi_{ut}$ and by Proposition 6 $a_1\hat{}\langle t.u\rangle\hat{}a_2 \models^{\mathsf{T}}_{UA} \phi_{ut}$. Hence, $i \models^{\mathsf{T}}_{UA} \phi_{ut}$.

Consider a term of the form ϕ^+_{ut} and let $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_{\phi^+_{ut}}(UA))$. 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 $a_i \in \mathscr{A}^*$, $a_{n+1} \in \mathscr{A}^*$, $t_i \in \mathscr{T}$, and $u_i \in \mathscr{U}$, for $i \in \{1\ldots n\}$ and $n \geq 1$. We reason inductively over n. Assume $n = 1$ and let $UA' = \mathsf{upd}(UA, a_1)$. Analogous to the previous case, it follows that $a_1\hat{}\langle t_1.u_1\rangle\hat{}a_2 \models^{\mathsf{T}}_{UA} \phi_{ut}$. By ($TR5$), $a_1\hat{}\langle t_1.u_1\rangle\hat{}a_2 \models^{\mathsf{T}}_{UA} \phi^+_{ut}$. We now assume $n > 1$ and $a_2\hat{}\langle t_2.u_2\rangle\hat{}a_3\hat{}\ldots\hat{}a_n\hat{}\langle t_n.u_n\rangle\hat{}a_{n+1} \models^{\mathsf{T}}_{UA'} \phi^+_{ut}$, for $UA' = \mathsf{upd}(UA, a_1)$. Because $[\![\phi_{ut}]\!]^{\mathscr{U}}_{UA'}$ engages in $t_1.u_1$, $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA'} \phi_{ut}$ by ($MA2$). From ($TR1$) it follows that $\langle t_1.u_1\rangle \models^{\mathsf{T}}_{UA'} \phi_{ut}$ and from ($TR6$) that $\langle t_1.u_1\rangle\hat{}a_2\hat{}\langle t_2.u_2\rangle\hat{}a_3\hat{}\ldots\hat{}a_n\hat{}\langle t_n.u_n\rangle\hat{}a_{n+1} \models^{\mathsf{T}}_{UA'} \phi^+_{ut}$. By Proposition 5, $a_1\hat{}\langle t_1.u_1\rangle\hat{}a_2\hat{}\langle t_2.u_2\rangle\hat{}a_3\hat{}\ldots\hat{}\ a_n\hat{}\langle t_n.u_n\rangle\hat{}a_{n+1} \models^{\mathsf{T}}_{UA} \phi^+_{ut}$ and hence $i \models^{\mathsf{T}}_{UA} \phi^+_{ut}$.

Step cases: For two terms ϕ and ψ, assume that $LHS \subseteq RHS$ holds. Consider now the term $\phi \sqcup \psi$ and let $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_{\phi\sqcup\psi}(UA))$. By ($MA3$), $SOD_{\phi\sqcup\psi}(UA) = SOD_\phi(UA) \,\square\, SOD_\psi(UA)$. From the denotational semantics of CSP it follows that either $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_\phi(UA))$ or $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_\psi(UA))$. Consider the first case. From the induction hypothesis $i \models^{\mathsf{T}}_{UA} \phi$. By ($TR7$), $i \models^{\mathsf{T}}_{UA} \phi \sqcup \psi$. The second case follows analogously by ($TR8$). Hence, $i \models^{\mathsf{T}}_{UA} \phi \sqcup \psi$.

Consider the term $\phi \sqcap \psi$ and let $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_{\phi\sqcap\psi}(UA))$. By ($MA4$), $SOD_{\phi\sqcap\psi}(UA) = SOD_\phi(UA) \parallel SOD_\psi(UA)$. From the denotational semantics of CSP it follows that $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_\phi(UA))$ and $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_\psi(UA))$. By the induction hypothesis, $i \models^{\mathsf{T}}_{UA} \phi$ and $i \models^{\mathsf{T}}_{UA} \psi$. By ($TR9$), $i \models^{\mathsf{T}}_{UA} \phi \sqcap \psi$.

Consider the term $\phi \odot \psi$ and let $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_{\phi\odot\psi}(UA))$. By ($MA5$), $SOD_{\phi\odot\psi}(UA) = SOD_\phi(UA) \underset{\mathscr{A}}{\parallel} SOD_\psi(UA)$. From the denotational semantics of CSP it follows that there are two traces $i_\phi, i_\psi \in \Sigma^*$ such that $i_\phi\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_\phi(UA))$ and $i_\psi\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_\psi(UA))$. From the induction hypothesis, it follows that $i_\phi \models^{\mathsf{T}}_{UA} \phi$ and $i_\psi \models^{\mathsf{T}}_{UA} \psi$. Moreover, because $SOD_\phi(UA)$ and $SOD_\psi(UA)$ synchronize on $\mathscr{A}$ but not on $\mathscr{X}$, $\mathsf{si}(i, i_\phi, i_\psi)$. By ($TR10$), $i \models^{\mathsf{T}}_{UA} \phi \odot \psi$.

Finally, consider the term $\phi \otimes \psi$ and let $i\hat{}\langle\checkmark\rangle \in \mathsf{T}(SOD_{\phi\otimes\psi}(UA))$. By ($MA6$), $SOD_{\phi\otimes\psi}(UA) = ([\![\phi]\!]^{U_\phi}_{UA} \underset{\mathscr{A}}{\parallel} [\![\psi]\!]^{U_\psi}_{UA}) \,\square\, \ldots$. From ($MA6$) and the denotational semantics of CSP it follows that there are two disjoint sets of users U_ϕ and U_ψ such that $i\hat{}\langle\checkmark\rangle \in \mathsf{T}([\![\phi]\!]^{U_\phi}_{UA} \underset{\mathscr{A}}{\parallel} [\![\psi]\!]^{U_\psi}_{UA})$. Analogous to the previous case, there are two traces $i_\phi, i_\psi \in \Sigma^*$ such that $i_\phi \models^{\mathsf{T}}_{UA} \phi$, $i_\psi \models^{\mathsf{T}}_{UA} \psi$, and $\mathsf{si}(i, i_\phi, i_\psi)$. By Proposition 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 ($TR11$), $i \models^{\mathsf{T}}_{UA} \phi \otimes \psi$.

$LHS \Leftarrow RHS$:

Base cases: Consider a unit term ϕ_{ut} and i be a trace such that $i \models^{\mathsf{T}}_{UA} \phi_{ut}$. The only rules of Definition 9 that allow for the derivation of ϕ_{ut} are ($TR1$)–($TR4$) and ($TR7$)–($TR9$). We can safely ingnore ($TR7$)–($TR9$) 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 every premise. Out of ($TR1$)–($TR4$), only ($TR1$) does not have a trace in its premises. Therefore, ($TR1$) is at the leaves of every derivation of $i \models^{\mathsf{T}}_{UA} \phi_{ut}$ and, thus, i contains an execution event $t.u$ for a task t and a user u. By iteratively applying the rules ($TR2$)–($TR4$), 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 $a_1, a_2 \in \mathscr{A}^*$. Let $UA' = \mathsf{upd}(UA, a_1)$. From Proposition 5 it follows that $\langle t.u\rangle\hat{}a_2 \models^{\mathsf{T}}_{UA'} \phi_{ut}$ and therefore, by ($TR1$), $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA'} \phi_{ut}$. By ($MA1$), $SOD_{\phi_{ut}}(UA)$ accepts a_1 and behaves like $SOD_{\phi_{ut}}(UA')$ afterwards. Because $\{\!\{u\}\!\} \models^{\mathsf{M}}_{UA'} \phi_{ut}$, $SOD_{\phi_{ut}}(UA')$ engages in $t.u$

and behaves like *END* afterwards. From *END*'s definition, it follows that *END* accepts $a_2 \hat{} \langle \checkmark \rangle$. Hence, $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\phi_{ut}}(UA))$.

Consider a term of the form ϕ_{ut}^+ and let i be a trace such that $i \models_{UA}^{\mathsf{T}} \phi_{ut}^+$. The only rules of Definition 9 that allow for the derivation of ϕ_{ut}^+ are $(TR2)$–$(TR6)$. Out of these, only $(TR5)$ does not have a trace that satisfies ϕ_{ut}^+ in its premises. Therefore, every derivation of $i \models_{UA}^{\mathsf{T}} \phi_{ut}^+$ contains one application of $(TR5)$ and, thus, i contains at least one execution event $t.u$, for a task t and a user u. By the rules $(TR2)$–$(TR4)$ and $(TR6)$, 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 $a_i \in \mathscr{A}^*$, $a_{n+1} \in \mathscr{A}^*$, $t_i \in \mathscr{T}$, and $u_i \in \mathscr{U}$, for $i \in \{1, \ldots, n\}$. 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, by $(MA2)$, that $a_1 \hat{} \langle t_1.u_1 \rangle \hat{} a_2 \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\phi_{ut}^+}(UA))$. For $n > 1$, assume $a_2 \hat{} \langle t_2.u_2 \rangle \hat{} \ldots \hat{} a_n \hat{} \langle t_n.u_n \rangle \hat{} a_{n+1} \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\phi_{ut}^+}(UA'))$, for $UA' = \mathsf{upd}(UA, a_1)$. Because $a_1 \hat{} \langle t_1.u_1 \rangle \hat{} \ldots \hat{} a_n \hat{} \langle t_n.u_n \rangle \hat{} a_{n+1} \models_{UA}^{\mathsf{T}} \phi_{ut}^+$, $\{\!\{u\}\!\} \models_{UA'}^{\mathsf{M}} \phi_{ut}$ by $(TR1)$, $(TR6)$, and Proposition 5. By $(MA2)$, $SOD_{\phi_{ut}^+}(UA)$ accepts a_1 and behaves like $SOD_{\phi_{ut}^+}(UA')$ afterwards. Because $\{\!\{u\}\!\} \models_{UA'}^{\mathsf{M}} \phi_{ut}$, $SOD_{\phi_{ut}^+}(UA')$ engages in $t_1.u_1$ and behaves like the external choice between $SOD_{\phi_{ut}^+}(UA')$ and *END* afterwards. We can therefore decide that the process behaves like $SOD_{\phi_{ut}^+}(UA')$. Therefore, by the induction hypothesis, $a_1 \hat{} \langle t_1.u_1 \rangle \hat{} a_2 \hat{} \langle t_2.u_2 \rangle \hat{} \ldots \hat{} a_n \hat{} \langle t_n.u_n \rangle \hat{} a_{n+1} \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\phi_{ut}^+}(UA))$. Hence, $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\phi_{ut}^+}(UA))$.

Step cases: For two terms ϕ and ψ, assume that $LHS \supseteq RHS$ holds. Consider now a term of the form $\phi \sqcup \psi$ and let $i \models_{UA}^{\mathsf{T}} \phi \sqcup \psi$. By $(TR7)$ and $(TR8)$, either $i \models_{UA}^{\mathsf{T}} \phi$ or $i \models_{UA}^{\mathsf{T}} \psi$. Consider the first case. By $(MA3)$ and the denotational semantics of CSP, $\mathsf{T}(SOD_{\phi \sqcup \psi}(UA)) = \mathsf{T}(SOD_{\phi}(UA)) \cup \mathsf{T}(SOD_{\psi}(UA))$. From the induction hypothesis it follows that $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\phi}(UA))$ and therefore, $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\phi \sqcup \psi}(UA))$. The second case is analogous. Hence, $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\phi \sqcup \psi}(UA))$.

Consider the term $\phi \sqcap \psi$ and let $i \models_{UA}^{\mathsf{T}} \phi \sqcap \psi$. By $(TR9)$, $i \models_{UA}^{\mathsf{T}} \phi$ and $i \models_{UA}^{\mathsf{T}} \psi$. By $(MA4)$ and the denotational semantics of CSP, $\mathsf{T}(SOD_{\phi \sqcap \psi}(UA)) = \mathsf{T}(SOD_{\phi}(UA)) \cap \mathsf{T}(SOD_{\psi}(UA))$. From the induction hypothesis it follows that $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\phi}(UA))$ and $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\psi}(UA))$ and therefore, $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\phi \sqcap \psi}(UA))$.

Consider the term $\phi \odot \psi$ and let $i \models_{UA}^{\mathsf{T}} \phi \odot \psi$. By $(TR10)$, there exist two traces $i_\phi, i_\psi \in \Sigma^*$ such that $i_\phi \models_{UA}^{\mathsf{T}} \phi$, $i_\psi \models_{UA}^{\mathsf{T}} \psi$, and $\mathsf{si}(i, i_\phi, i_\psi)$. By the induction hypothesis, $i_\phi \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\phi}(UA))$ and $i_\psi \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\psi}(UA))$, and therefore also $i_\phi \in \mathsf{T}(SOD_{\phi}(UA))$ and $i_\psi \in \mathsf{T}(SOD_{\psi}(UA))$. From $(MA5)$ and Proposition 8 it follows that $i \in \mathsf{T}(SOD_{\phi \odot \psi}(UA))$. Moreover, by $(MA5)$, because $SOD_{\phi}(UA)$ and $SOD_{\psi}(UA)$ both engage in $\checkmark$ after having accepted i_ϕ and i_ψ respectively, $SOD_{\phi \odot \psi}(UA)$ engages in $\checkmark$ too, after having accepted i. Hence, $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\phi \odot \psi}(UA))$.

Finally, consider a term of the form $\phi \otimes \psi$ and let $i \models_{UA}^{\mathsf{T}} \phi \otimes \psi$. By $(TR11)$, there exist two traces i_ϕ and i_ψ such that $i_\phi \models_{UA}^{\mathsf{T}} \phi$, $i_\psi \models_{UA}^{\mathsf{T}} \psi$, $\mathsf{si}(i, i_\phi, i_\psi)$, 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 $U_\phi, U_\psi \subseteq \mathscr{U}$ such that $U_\phi \cup U_\psi = \mathscr{U}$, $U_\phi \cap U_\psi = \varnothing$, $\mathsf{userset}(\mathsf{users}(i_\phi)) \subseteq U_\phi$, and $\mathsf{userset}(\mathsf{users}(i_\psi)) \subseteq U_\psi$. By the induction hypothesis, $i_\phi \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\phi}(UA))$ and $i_\psi \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\psi}(UA))$, and therefore also $i_\phi \in \mathsf{T}(SOD_{\phi}(UA))$ and $i_\psi \in \mathsf{T}(SOD_{\psi}(UA))$. From Proposition 7, $[\![\phi]\!]_{UA}^{U_\phi}$ therefore accepts i_ϕ and $[\![\psi]\!]_{UA}^{U_\psi}$ accepts i_ψ. By $(MA6)$ and the denotational semantics of CSP, $SOD_{\phi \otimes \psi}(UA)$ also behaves like $[\![\phi]\!]_{UA}^{U_\phi} \underset{\mathscr{A}}{\parallel} [\![\psi]\!]_{UA}^{U_\psi}$. Analogous to the previous case, it follows that $i \in \mathsf{T}(SOD_{\phi \otimes \psi}(UA))$ and $i \hat{} \langle \checkmark \rangle \in \mathsf{T}(SOD_{\phi \otimes \psi}(UA))$. ■