RZ 3726 (# 99736) 01/12/2009
Computer Science 19 pages

Research Report

Dynamic Enforcement of Abstract Separation of Duty Constraints

D. Basin*, S.J. Burri‡ and G. Karjoth‡

*ETH Zurich
Department of Computer Science
basin@inf.ethz.ch

‡ IBM Research GmbH
Zurich Research Laboratory
{sbu, gka}@zurich.ibm.com

Dynamic Enforcement of Abstract Separation of Duty Constraints

David Basin[*]
[*] ETH Zurich
Department of Computer Science
basin@inf.ethz.ch

Samuel J. Burri[*†] and Günter Karjoth[†]
[†] IBM Research
Zurich Research Laboratory
{sbu, gka}@zurich.ibm.com

Abstract

Separation of Duties (SoD) is a well-established security principle that aims to prevent fraud and errors by distributing tasks and associated privileges for business processes among multiple users. Li and Wang recently 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 any specific workflow model. In this paper, we both generalize SoDA and map it to enforcement mechanisms. First, we increase SoDA's expressiveness by extending its semantics to multisets. This better suits policy enforcement over workflows, where users may execute multiple tasks. Second, we further generalize SoDA to allow for changing role assignments. This lifts the strong restriction that authorizations do not change during workflow execution. Finally, we map SoDA terms to CSP processes. Since CSP has an operational semantics, this mapping provides the critical link between abstract specifications of SoD requirements by SoDA terms and runtime-enforcement mechanisms.

1. Introduction

Most information security mechanisms protect resources from external threats. However, threats often reside within organizations [13] where authorized users may intentionally or accidentally misuse information systems. Examples are the scandals [7] that led to regulations such as the Sarbanes-Oxley Act [1]. These regulations require companies to document their processes, to identify conflicts of interests, to adopt countermeasures, and to audit and control those activities.

Separation of Duties (SoD) is a well-established extension to access control that aims to ensure the integrity of data, in particular the prevention of fraud and errors [4,5,16,27,28]. The main idea behind SoD is to split critical processes into multiple actions and to ensure that no single user can execute all of them. Therefore, at least two users must be involved in the process and fraud would require 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) of Li and Wang [19] constitutes an 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 P that requires a user other than `Bob` that acts in the role of a `Manager` and two additional users, each acting as an `Accountant`. Using SoDA, P can be modeled by the term

$$(\texttt{Manager} \sqcap \neg\{\texttt{Bob}\}) \otimes \texttt{Accountant} \otimes \texttt{Accountant}\,.$$

The above constraint specifies both the number and kinds of users who must take part in the workflow, independent of the details of the workflow itself. Separating concerns in this way allows business processes and security requirements to be developed independently. This is beneficial as business and security knowledge usually resides with different people.

Previous research on SoDA focused on requirements specification, complexity questions, and enforcement for a restricted class of terms. However, no general mapping existed from SoDA terms onto workflows or to dynamic enforcement mechanisms. This is the problem that we tackle in this paper.

We proceed by constructing formal models of workflows, access control enforcement, and separation of duty constraints, as well as their combination. We specify them using the process algebra Communicating Sequential Processes (CSP). First, we model a workflow as a CSP process W. Afterwards we define the process $RBAC$ that describes an enforcement mechanism for Role-Based Access Control (RBAC) [9]. Given a SoDA term ϕ, we then define a mapping from ϕ to the *SoD enforcement processes* SOD_ϕ that engages in all workflow executions corresponding to a satisfying assignment of ϕ. Finally, we combine W, $RBAC$, and SOD_ϕ, resulting in an *SoD secure (workflow) process*. CSP's operational semantics allows us to translate these processes, together or separately, to programs that serve as enforcement monitors for W, $RBAC$, and SOD_ϕ. The main focus of this paper, however, is on specifying these processes and their properties, rather than on their efficient implementation.

Contributions: We extend the original SoDA semantics [19] to multisets of users and interpret SoDA terms over workflow traces, allowing for changing role assignments (or, equivalently, sessions). The resulting semantics is well-suited for policy enforcement over workflows, where users may execute multiple tasks and authorizations may change during workflow execution. We further bridge the gap between the specification of high-level SoD constraints and their enforcement in a workflow environment by defining a mapping from SoDA terms to CSP processes. We provide a correctness proof for this mapping, establishing that every execution accepted by an SoD enforcement process complies with its corresponding SoD policy.

Organization: In Section 2, we provide background on CSP and multisets. In Section 3, we describe our modeling of workflows and access control enforcement. We define SoDA's syntax and describe our multiset semantics in Section 4. In Section 5, we present our trace semantics for SoDA and define a mapping from SoDA terms to CSP processes. Furthermore, we relate our two semantics to Li and Wang's. We discuss related work in Section 6 and conclude in Section 7. Proofs and a summary of Li and Wang's SoDA semantics follow in the Appendix.

2. Background

2.1. Communicating sequential processes

In this paper, we make use of Hoare's process algebra *Communicating Sequential Processes (CSP)* [14]. We provide here a brief introduction to CSP, highlighting the main concepts employed in our work. For a more detailed introduction to CSP, see [26].

CSP provides a language for describing concurrently executing processes that communicate with each other. CSP processes are built from several sets of symbols. Σ is the set of *events*, which processes use to communicate with their environment. Events can be structured using *channels*. Given a channel c and a set A, we can define c to be *of type* A. This means that for all $a \in A$, events of the form $c.a$ belong to Σ and represent the communication of a on the channel c. By $\{| c |\}$, we denote the set of all possible events involving channel c, i.e., $\{| c |\} := \{c.a \mid a \in A\}$. For a a tuple $(a_1, ..., a_n)$, where $n \geq 1$, we write $c.a_1. \ldots .a_n$.

A *process* describes a communication pattern. Processes are referred to by *process identifiers*. Let $\mathcal{I}$ be the set of all process identifiers. The set of processes $\mathcal{P}$ is then inductively defined by the following grammar:

$$\mathcal{P} ::= e \rightarrow \mathcal{P} \mid STOP \mid i \mid \mathcal{P} \Box \mathcal{P} \mid \mathcal{P} \underset{E}{\|} \mathcal{P} \, ,$$

where $i \in \mathcal{I}$, $e \in \Sigma$, and $E \subseteq \Sigma$.

For a process $P \in \mathcal{P}$ and an identifier $i \in \mathcal{I}$, $i = P$ denotes the *assignment* of P to i. Processes can be *parametrized*, for example $i(v) = P$ defines a process parametrized by variable v. This denotes a family of processes, one for each value of v.

Let $P, Q \in \mathcal{P}$ be two processes. The process $e \rightarrow P$ *engages* in the event e first and behaves like the process P afterward. When using channels, this notation can be extended. For $A' \subseteq A$, the expression $c?a : A' \rightarrow P$ represents a process that waits for an $a \in A'$ to be *received* on channel c of type A and afterwards behaves like P. Similarly, $c!a \rightarrow P$ represents a process that *sends* a on channel c and afterwards behaves like P. The process $STOP$ represents the terminated process, which cannot engage in any further events. For an assignment $i = P$, the process i behaves like P. $P \Box Q$ denotes a process that lets the environment choose whether it behaves like P or Q. The $\Box$ operator can be *replicated*. For a non-empty set of processes $S \subseteq \mathcal{P}$, for example, $\Box\, S$ lets the environment choose one of the processes in S. The process $P \underset{E}{\|} Q$ represents the parallel execution of the processes P and Q *synchronized* on the set of events $E \subseteq \Sigma$. That means, whenever one of the two processes engages in an event $e \in E$, the other process must engage in e as well. If one of them, say P, cannot (yet) engage in e, the other one, Q, waits until P is ready to engage in e.

A *trace* is a sequence of events. $\langle\rangle$ denotes the *empty* trace, $\langle e \rangle$ describes a *singleton* trace containing the event e, and $\langle e_1, ..., e_n \rangle$, for $n \geq 1$, denotes a trace containing the events e_1 to e_n. For two finite traces t and t', $t\hat{\ }t'$ denotes their *concatenation*. Moreover, for a set of events E, E^* denotes the set of all finite traces over E and E^+ denotes the set of all finite traces over E that contain at least one event.

The *denotational semantics* of CSP [26] describes a process as a set $\mathcal{T}(P) \subseteq \Sigma^*$ of finite traces. When

$t \in \mathcal{T}(P)$, we say that P *accepts* t; each such trace t describes a sequence of events that P can engage in with the environment. For example, $\mathcal{T}(STOP) := \{\langle\rangle\}$, $\mathcal{T}(e \to P) := \{\langle\rangle\} \cup \{\langle e\rangle\hat{}\,t \mid t \in \mathcal{T}(P)\}$, and $\mathcal{T}(P \,\Box\, Q) := \mathcal{T}(P) \cup \mathcal{T}(Q)$. We say that Q *refines* P, denoted $P \sqsubseteq_{\mathcal{T}} Q$, if and only if $\mathcal{T}(Q) \subseteq \mathcal{T}(P)$. *Failures-Divergence Refinement (FDR)* [11] is a model checker that decides whether one process refines another one.

2.2. Multisets

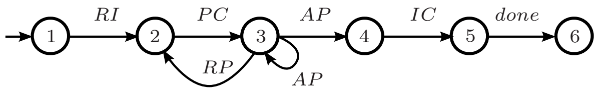
A *multiset*, or *bag*, is a collection of objects where repetition is allowed [34]. Formally, given a set A, a multiset $\mathbf{M}$ of A is a pair (A, f), where the function $f : A \to \mathbb{N}_0$ (where $\mathbb{N}_0$ is the set of natural numbers including zero) defines how often each element $a \in A$ occurs in $\mathbf{M}$. We write $\mathbf{M}(a)$ as shorthand for $f(a)$. We say that a is an *element* of $\mathbf{M}$, written $a \in \mathbf{M}$, if $\mathbf{M}(a) \geq 1$. We use standard set notation to define multisets, but allow duplicated elements, e.g., $\mathbf{M} := \{a_1, a_1\}$ is the multiset where $\mathbf{M}(a_1) = 2$ and for all other $a \in A$, $\mathbf{M}(a) = 0$. For a finite multiset $\mathbf{M}$, $|\mathbf{M}|$ denotes the *cardinality* of $\mathbf{M}$ and is defined as $\sum_{a \in A} \mathbf{M}(a)$. Given the multisets $\mathbf{M}$ and $\mathbf{N}$, their *intersection*, denoted $\mathbf{M} \cap \mathbf{N}$, is the multiset $\mathbf{O}$, where for all $a \in A$, $\mathbf{O}(a) := \min(\mathbf{M}(a), \mathbf{N}(a))$. Similarly, their *union*, denoted $\mathbf{M} \cup \mathbf{N}$, is the multiset $\mathbf{O}$, where for all $a \in A$, $\mathbf{O}(a) := \max(\mathbf{M}(a), \mathbf{N}(a))$ and their *sum*, denoted $\mathbf{M} \uplus \mathbf{N}$, is the multiset $\mathbf{O}$, where for all $a \in A$, $\mathbf{O}(a) := \mathbf{M}(a) + \mathbf{N}(a)$. The *empty multiset* $\emptyset$ of A is the multiset where $\emptyset(a) := 0$, for all $a \in A$. For a set A, the function multi returns the set of all multisets of A. Formally, $\mathsf{multi}(A) := \{(A, f) \mid f : A \to \mathbb{N}_0\}$. Note that $\mathsf{multi}(A)$ is infinite whenever $A \neq \emptyset$.

3. Secure workflow processes

3.1. Modeling workflows

We call a unit of work an *action*. The temporal ordering and logical dependencies of actions that together implement a business objective is called a *workflow*. There are various formalisms for modeling workflows. We use CSP. Other models and formalisms are mentioned in Section 6.

For the rest of this paper, let $\mathcal{U}$ be a set of *users* and $\mathcal{A}$ be a set of *actions*. We then model workflows as CSP processes with a channel $business$ of type $\mathcal{U} \times \mathcal{A}$. Let $\mathcal{E}_B := \{|business|\}$. We call an element of $\mathcal{E}_B$ a *business event*. For a user $u \in \mathcal{U}$ and an action $a \in \mathcal{A}$, the business event $business.u.a$ describes the execution of the action a by the user u.



$$
\begin{aligned}
RI &:= \{business.u.\texttt{receive invoice} \mid u \in \mathcal{U}\}\\
PC &:= \{business.u.\texttt{prepare check} \mid u \in \mathcal{U}\}\\
AP &:= \{business.u.\texttt{approve payment} \mid u \in \mathcal{U}\}\\
RP &:= \{business.u.\texttt{reject payment} \mid u \in \mathcal{U}\}\\
IC &:= \{business.u.\texttt{issue check} \mid u \in \mathcal{U}\}
\end{aligned}
$$

Figure 1. Payment workflow

We introduce the event $done$, which states that a workflow has finished.[1] We further define the auxiliary predicate done on traces, where for all $t \in \Sigma^*$, $\mathsf{done}(t)$ if and only if t contains exactly one event $done$ in the end. Formally, $\mathsf{done}(t) := \exists t' \in (\Sigma \setminus \{done\})^* \,.\, t = t'\hat{}\langle done\rangle$.

For a workflow w modeled by a process W, a trace $t \in \mathcal{T}(W)$ corresponds to a *workflow run* (or *workflow instance*) of w. A trace t represents a *finished* workflow run if $\mathsf{done}(t)$; otherwise t represents an *unfinished* workflow run. Note that given a trace t and a process W, it is straightforward to check, using CSP's operational semantics, whether $t \in \mathcal{T}(W)$. Moreover, we can also use FDR to decide whether the workflow run represented by t is a possible execution of the workflow modeled by W.

For a CSP process W that models a workflow, we require the set of traces $\mathcal{T}(W)$ to contain at least one trace that corresponds to a finished workflow run, i.e., $\exists\, t \in \mathcal{T}(W) \,.\, \mathsf{done}(t)$. This ensures that each workflow can be finished in at least one way.

We define two auxiliary functions that extract users from traces. First, the projection function $\mathsf{user} : \mathcal{E}_B \to \mathcal{U}$, given a business event $business.u.a$, returns u. Second, the function $\mathsf{users} : \Sigma^* \to \mathsf{multi}(\mathcal{U})$, given a trace t, returns the multiset of users that are contained in business events in t.

$$
\mathsf{users}(t) := \begin{cases}
\emptyset & \text{if } t = \langle\rangle,\\
\{\mathsf{user}(b)\} \uplus \mathsf{users}(t') & \text{for } t = \langle b\rangle\hat{}\,t' \\ & \text{and } b \in \mathcal{E}_B,\\
\mathsf{users}(t') & \text{for } t = \langle e\rangle\hat{}\,t' \\ & \text{and } e \notin \mathcal{E}_B.
\end{cases}
$$

To illustrate these notions, we introduce a running example of a payment process. A similar example is used in [28]. More complex workflows, modeling

1. We did not use CSP's special event ✓ and the process $SKIP$ because later we synchronize on $done$ with most, but not all, involved processes. By the semantics of CSP, all processes must synchronize on ✓.

real-world applications with SoD constraints, are given in [31,32].

Example 1 (Payment workflow) This workflow models the actions taken in a payment process where invoices are payed by check.

$$\begin{aligned}
W &= W_1\\
W_1 &= business?u:\mathcal{U}.\texttt{receive invoice} \rightarrow W_2\\
W_2 &= business?u:\mathcal{U}.\texttt{prepare check} \rightarrow W_3\\
W_3 &= (business?u:\mathcal{U}.\texttt{reject payment} \rightarrow W_2)\\
&\quad \Box\ (business?u:\mathcal{U}.\texttt{approve payment} \rightarrow W_3)\\
&\quad \Box\ (business?u:\mathcal{U}.\texttt{approve payment} \rightarrow W_4)\\
W_4 &= business?u:\mathcal{U}.\texttt{issue check} \rightarrow W_5\\
W_5 &= done \rightarrow STOP
\end{aligned}$$

In this workflow, first an invoice is received. After the payment check has been prepared, the payment is either approved or rejected. In the latter case, the payment must be prepared again. If the payment is finally approved, the check is issued and the workflow terminates, which is denoted by the event $done$.

Figure 1 illustrates W graphically as a labeled transition system, whose edges are annotated by sets of labels. The edge $s_1 \xrightarrow{\{l_1,\ldots,l_n\}} s'$ denotes the set of labeled transitions $s \xrightarrow{l_i} s'$, for $i \in \{1,\ldots,n\}$. For example, for every user $u \in \mathcal{U}$ there is a transition $business.u.\texttt{receive invoice}$ from state 1 to state 2.

3.2. Access control

We use *role-based access control (RBAC)* [8,9,29] to describe access control policies. We only make use of RBAC's core idea, which is the decomposition of the user-permission assignment relation into a user-role and a role-permission assignment relation. For the reminder of this paper, let $\mathcal{R}$ be a set of *roles*.

Definition 1 (RBAC configuration) *An* RBAC configuration *is a tuple* (UA, PA)*, where* $UA \subseteq \mathcal{U} \times \mathcal{R}$ *is the* user assignment relation *and* $PA \subseteq \mathcal{R} \times \mathcal{A}$ *is the* permission assignment relation.

We say that the user u *acts in role* r if $(u, r) \in UA$. Furthermore, the user u is *authorized to execute the action* a, if $\exists r \in \mathcal{R}\,.\ (u, r) \in UA \wedge (r, a) \in PA$.

In contrast to the RBAC standard of NIST [9], 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, where all assigned roles are always implicitly activated. Note that what we call actions are called *permissions* in [9].

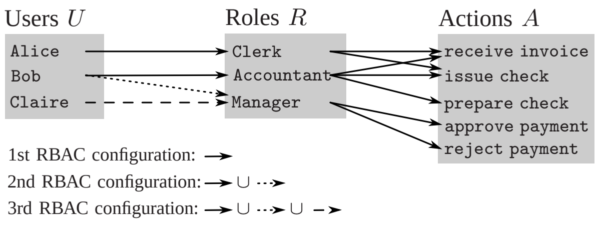


Figure 2. Example RBAC configuration

Administrative actions $\mathcal{A}_A \subseteq \mathcal{A}$ are the subset of actions that are used to modify RBAC configurations. For a user $u \in \mathcal{U}$, a role $r \in \mathcal{R}$, and a user assignment relation UA, the action $\texttt{addUA}.u.r$ adds the tuple (u, r) to UA and the action $\texttt{rmUA}.u.r$ removes (u, r) from UA. In this paper, we do not discuss administrative actions that change permission assignment relations. A formal description of the semantics of administrative actions is provided in Section 3.3.

3.3. The RBAC process

Administrative actions change RBAC configurations. We describe the configuration's evolution and the enforcement of the resulting access-control policy in terms of a CSP process that we call the *RBAC process*. As defined in Figure 3, the RBAC process is parametrized by a user assignment relation UA and a permission assignment relation PA, which together represent an RBAC configuration. Besides the channel $business$, introduced in Section 3.1, the RBAC process also has a channel called $admin$ of type $\mathcal{A}_A$. Let $\mathcal{E}_A := \{|\, admin \,|\}$. We call an element of $\mathcal{E}_A$ an *admin event*. The RBAC workflow engages in the following kinds of events:

1) $business.u.a$ for a user $u \in \mathcal{U}$ and an action $a \in \mathcal{A}$, if u is authorized to execute a under the current RBAC configuration.
2) $admin.\texttt{addUA}.u.r$ adds the tuple (u, r) to the current RBAC configuration, for $u \in \mathcal{U}$ and $r \in \mathcal{R}$.
3) $admin.\texttt{rmUA}.u.r$ removes the tuple (u, r) from the current RBAC configuration, for $u \in \mathcal{U}$ and $r \in \mathcal{R}$.

Note that the RBAC process does not terminate, i.e., it never behaves like $STOP$. This is consistent with our view of access-control monitors that outlive workflow execution.

Given a process W that models a workflow, we define the *secure (workflow) process* SW as the parallel execution of W and $RBAC$, synchronized on the

$$\begin{aligned} RBAC(UA,PA) \quad = \quad & \big(business?(u.a) : \{u.a \mid \exists r \in \mathcal{R} \,.\, (u,r) \in UA \land (r,a) \in PA\} \to RBAC(UA,PA) \,\big) \\ & \Box \;\; \big(admin.\texttt{addUA}?u : \mathcal{U}?r : \mathcal{R} \to RBAC(UA \cup \{(u,r)\}, PA) \\ & \Box \;\; \big(admin.\texttt{rmUA}?u : \mathcal{U}?r : \mathcal{R} \to RBAC(UA \setminus \{(u,r)\}, PA) \big) \end{aligned}$$

Figure 3. The RBAC process

business events $\mathcal{E}_B$. Like the RBAC process, a secure process is parametrized by an RBAC configuration.

$$SW(UA,PA) = W \underset{\mathcal{E}_B}{\|} RBAC(UA,PA) \ .$$

A secure process models a workflow that only executes actions authorized under the current RBAC configuration. By synchronizing only on business events, arbitrary admin events can be interleaved with business events and $done$ in any order. Thus, the RBAC configuration can change between the execution of workflow actions.

We have now introduced all the kinds of events that we need. Specifically, $\Sigma = \mathcal{E}_B \cup \mathcal{E}_A \cup \{done\}$.

Example 2 (Secure workflow process) We refine the workflow from Example 1 into a secure workflow process. Assume $\mathcal{U} := \{\texttt{Alice}, \texttt{Bob}, \texttt{Claire}\}$, $\mathcal{A} := \{$`receive invoice`, `issue check`, `prepare check`, `prepare check`, `approve payment`$\}$, and $\mathcal{R} := \{$`Accountant`, `Clerk`, `Manager`$\}$. Also, let the RBAC configuration (UA, PA) be initially given as depicted by the solid arrows in Figure 2. Consider the following trace, which corresponds to a finished workflow run.

$$\begin{aligned} t \quad := \quad & \langle business.\texttt{Alice.receive invoice}, \\ & business.\texttt{Bob.prepare check}, \\ & business.\texttt{Bob.approve payment}, \\ & business.\texttt{Alice.issue check},\ done\rangle \end{aligned}$$

This trace represents a workflow run of the payment workflow described in Example 1 and modeled by W. In contrast, $t \notin \mathcal{T}(SW(UA,PA))$ because no user is authorized to execute `approve payment`. This can be overcome by placing `Bob` in the `Manager` role with the admin event $admin.$`addUA.Bob.Manager` as follows.

$$\begin{aligned} t' \quad := \quad & \langle business.\texttt{Alice.receive invoice}, \\ & business.\texttt{Bob.prepare check}, \\ & admin.\texttt{addUA.Bob.Manager}, \\ & business.\texttt{Bob.approve payment}, \\ & business.\texttt{Alice.issue check},\ done\rangle \end{aligned}$$

The new admin event adds $(\texttt{Bob}, \texttt{Manager})$ to SW's RBAC configuration as indicated by the dotted arrow in Figure 2. Therefore, $t' \in \mathcal{T}(SW(UA,PA))$. However, it is risky to allow `Bob` to execute both the actions `prepare check` and `approve payment` as he could approve his own fraudulent payments. The next refinement of our running example solves this problem by enforcing an appropriate SoD constraint.

4. Abstract separation of duty constraints

4.1. Separation of duty algebra syntax

Our work builds on Li and Wang's *separation of duty algebra* [19], *SoDA*. We present below the syntax of SoDA terms.

Definition 2 (SoDA grammar $\mathfrak{G}$) *A* SoDA grammar $\mathfrak{G}$ *with respect to a* $\mathcal{U} := \{u_1, \ldots, u_n\}$ *and a* $\mathcal{R} := \{r_1, \ldots, r_m\}$ *is a quadruple*

$$\mathfrak{G} := (N, T, P, S)$$

where:

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

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

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

(1) $$\frac{}{\{u\} \vdash^{\mathcal{M}}_{UA} \mathsf{All}} \quad \exists r \in \mathcal{R} \,.\, (u,r) \in UA$$

(2) $$\frac{}{\{u\} \vdash^{\mathcal{M}}_{UA} r} \quad (u,r) \in UA$$

(3) $$\frac{}{\{u\} \vdash^{\mathcal{M}}_{UA} U} \quad u \in U \text{ and } \exists r \in \mathcal{R} \,.\, (u,r) \in UA$$

(4) $$\frac{\{u\} \nvdash^{\mathcal{M}}_{UA} \phi}{\{u\} \vdash^{\mathcal{M}}_{UA} \neg\phi}$$

(5) $$\frac{\{u\} \vdash^{\mathcal{M}}_{UA} \phi}{\{u\} \vdash^{\mathcal{M}}_{UA} \phi^{+}}$$

(6) $$\frac{\{u\} \vdash^{\mathcal{M}}_{UA} \phi,\ \mathbf{U} \vdash^{\mathcal{M}}_{UA} \phi^{+}}{(\{u\} \uplus \mathbf{U}) \vdash^{\mathcal{M}}_{UA} \phi^{+}}$$

(7) $$\frac{\mathbf{U} \vdash^{\mathcal{M}}_{UA} \phi}{\mathbf{U} \vdash^{\mathcal{M}}_{UA} (\phi \sqcup \psi)}$$

(8) $$\frac{\mathbf{U} \vdash^{\mathcal{M}}_{UA} \psi}{\mathbf{U} \vdash^{\mathcal{M}}_{UA} (\phi \sqcup \psi)}$$

(9) $$\frac{\mathbf{U} \vdash^{\mathcal{M}}_{UA} \phi,\ \mathbf{U} \vdash^{\mathcal{M}}_{UA} \psi}{\mathbf{U} \vdash^{\mathcal{M}}_{UA} (\phi \sqcap \psi)}$$

(10) $$\frac{\mathbf{U} \vdash^{\mathcal{M}}_{UA} \phi,\ \mathbf{V} \vdash^{\mathcal{M}}_{UA} \psi}{(\mathbf{U} \uplus \mathbf{V}) \vdash^{\mathcal{M}}_{UA} (\phi \odot \psi)}$$

(11) $$\frac{\mathbf{U} \vdash^{\mathcal{M}}_{UA} \phi,\ \mathbf{V} \vdash^{\mathcal{M}}_{UA} \psi}{(\mathbf{U} \uplus \mathbf{V}) \vdash^{\mathcal{M}}_{UA} (\phi \otimes \psi)} \quad (\mathbf{U} \cap \mathbf{V}) = \emptyset \,.$$

Figure 4. SoDA$^{\mathcal{M}}$: semantics for multisets of users

The terminal symbols $\otimes$, $\odot$, $\sqcup$, $\sqcap$, $^{+}$, and $\neg$ are called *operators*. The nonterminal symbol AT yields either a non-empty set of users, e.g. $\{\texttt{Alice}, \texttt{Bob}\}$, a single role, e.g. `Clerk`, or the keyword All. Note that without loss of generality, we omit the productions $CT ::= (T \sqcap CT)$ and $CT ::= (T \sqcup CT)$ in order to keep $\mathfrak{G}$ small. Li and Wang showed in [19] 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 that is constructible without them.

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

4.2. SoDA semantics for multisets of users

Li and Wang define the satisfaction of SoDA terms for sets of users [19]. We refer to their semantics as SoDA$^{\mathcal{S}}$. We summarize their semantics and give examples of SoDA terms and their interpretation in the the appendix.

To motivate the need for an alternate semantics, consider the policy P that requires `Bob` to execute two actions, modeled by the SoDA term $\phi := \{\texttt{Bob}\} \odot \{\texttt{Bob}\}$. Under SoDA$^{\mathcal{S}}$, ϕ is satisfied by the set $\{\texttt{Bob}\}$. Our ultimate goal is to map ϕ to a CSP process that accepts all traces that correspond to satisfying assignments of ϕ. Mapping sets to traces leaves room for interpretation. If we define the correspondence between sets and traces in a way that $\{\texttt{Bob}\}$ maps to the set of traces containing *exactly one* business event executed by `Bob`, this would not satisfy P. On the other hand, if we map $\{\texttt{Bob}\}$ to the set of traces containing *arbitrarily many* business events executed by `Bob`, this set would also include traces that do not satisfy P, for example, the trace containing three business events executed by `Bob`.

To address the above problem, we introduce a new semantics, SoDA$^{\mathcal{M}}$, that defines term satisfaction based on multisets of users. This allows us to make finer distinctions concerning repetition (quantification requirements) than in SoDA$^{\mathcal{S}}$. As shown below, under SoDA$^{\mathcal{M}}$, ϕ is only satisfied by the multiset $\{\texttt{Bob}, \texttt{Bob}\}$. Mapping multisets to traces is straightforward and the corresponding traces include exactly two business events that are executed by `Bob`. In this respect, SoDA$^{\mathcal{M}}$ also allows a more precise mapping to traces than SoDA$^{\mathcal{S}}$.

Definition 3 (Multiset Satisfaction SoDA$^{\mathcal{M}}$) *Let ϕ be a term, $U \subseteq \mathcal{U}$ a non-empty set of users, and $r \in \mathcal{R}$ a role. A multiset of users* $\mathbf{U}$ satisfies ϕ with respect to a user assignment relation UA, *denoted by* $\mathbf{U} \vdash^{\mathcal{M}}_{UA} \phi$,

$$\dfrac{\dfrac{\dfrac{}{\{\texttt{Bob}\} \vdash^{\mathcal{M}}_{UA''} \texttt{Accountant}}\,(2),\quad \dfrac{\dfrac{}{\{\texttt{Claire}\} \vdash^{\mathcal{M}}_{UA''} \texttt{Manager}}\,(2)}{\{\texttt{Claire}\} \vdash^{\mathcal{M}}_{UA''} \phi'}\,(7)}{\{\texttt{Bob},\texttt{Claire}\} \vdash^{\mathcal{M}}_{UA''} (\texttt{Accountant} \otimes \phi')}\,(11),\quad \dfrac{\dfrac{}{\{\texttt{Alice}\} \vdash^{\mathcal{M}}_{UA''} \mathsf{All}}\,(1)\quad \dfrac{\dfrac{}{\{\texttt{Alice}\} \vdash^{\mathcal{M}}_{UA''} \mathsf{All}}\,(1)}{\{\texttt{Alice}\} \vdash^{\mathcal{M}}_{UA''} \mathsf{All}^{+}}\,(5)}{\{\texttt{Alice},\texttt{Alice}\} \vdash^{\mathcal{M}}_{UA''} \mathsf{All}^{+}}\,(6)}{\{\texttt{Alice},\texttt{Alice},\texttt{Bob},\texttt{Claire}\} \vdash^{\mathcal{M}}_{UA''} (\texttt{Accountant} \otimes (\texttt{Manager} \sqcup (\texttt{Accountant} \otimes \texttt{Accountant}))) \odot \mathsf{All}^{+}}\,(10)$$

for $\phi' := \texttt{Manager} \sqcup (\texttt{Accountant} \otimes \texttt{Accountant})$

Figure 5. Example derivation under $\text{SoDA}^{\mathcal{M}}$

if and only if $\mathbf{U}$ *can be derived from* ϕ *using the rules listed in Figure 4.*

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 either in $\mathbf{U}$ or $\mathbf{V}$, but not in both, the $\otimes$ operator separates duties between two multisets of users. In contrast, the $\odot$ operator allows overlapping duties. 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 $\mathbf{V}$.

Definition 4 (Term evaluation) *The* value *of a term* ϕ *with respect to a user assignment relation* UA*, denoted by* $S_{UA}(\phi)$*, is the set of all multisets of users that satisfy* ϕ *with respect to* UA*. Formally,* $S_{UA}(\phi) := \{\mathbf{U} \in \mathsf{multi}(\mathcal{U}) \mid \mathbf{U} \vdash^{\mathcal{M}}_{UA} \phi\}$.

We illustrate this using our running example.

Example 3 Suppose we have the term

$$\begin{aligned} \phi \quad := \quad & (\texttt{Accountant} \otimes (\texttt{Manager} \sqcup \\ & (\texttt{Accountant} \otimes \texttt{Accountant}))) \odot \mathsf{All}^{+} , \end{aligned}$$

and the user assignment relation shown in Figure 2, now including all arrows between users and groups,

$$\begin{aligned} UA'' \quad := \quad & \{(\texttt{Alice}, \texttt{Clerk}), (\texttt{Bob}, \texttt{Accountant}), \\ & (\texttt{Bob}, \texttt{Manager}), (\texttt{Claire}, \texttt{Manager})\}. \end{aligned}$$

Figure 5 shows a derivation $\{\texttt{Alice}, \texttt{Alice}, \texttt{Bob}, \texttt{Claire}\}$ from ϕ using the rules from Figure 4.

We conclude by relating $\text{SoDA}^{\mathcal{M}}$ and $\text{SoDA}^{\mathcal{S}}$. Under $\text{SoDA}^{\mathcal{S}}$, $X \vdash^{\mathcal{S}}_{(U,UR)} \phi$ denotes the satisfaction of a term ϕ by a set of users X with respect to a tuple (U, UR), where $U \subseteq \mathcal{U}$ and $UR \subseteq U \times \mathcal{R}$ (see appendix). Because actions can only be executed by users who have at least one role assignment, we simplify this tuple and extract the available users from UA, as one can see in Rule (3) of Figure 4. For a user assignment relation UA, the function $\mathsf{lwconf}(UA) := (\{u \in \mathcal{U} \mid \exists r \in \mathcal{R} \,.\, (u, r) \in UA\}, UA)$ maps UA to the corresponding tuple in $\text{SoDA}^{\mathcal{S}}$. Moreover, given a multiset of users $\mathbf{U}$, the function $\mathsf{userset}(\mathbf{U}) := \{u \mid u \in \mathbf{U}\}$ returns the the set of users contained in $\mathbf{U}$. We prove the following lemma in the appendix, showing that $\text{SoDA}^{\mathcal{M}}$ generalizes $\text{SoDA}^{\mathcal{S}}$ in the following sense.

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

5. Separation of duty enforcement

5.1. Approach and requirements

As shown above, SoDA specifies SoD constraints at a high level of abstraction. We now describe how, given a term ϕ, we can construct an enforcement monitor for ϕ. Our construction maps ϕ to a CSP 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. If $SOD_{\phi}(UA)$ does not engage in admin events, its definition is straightforward. In this case, $\mathcal{T}(SOD_{\phi}(UA))$ is equal to $\{t \in \mathcal{E}_B^* \mid \exists t' \in \mathcal{E}_B^* \,.\, \mathsf{users}(t \hat{\ } t') \in S_{UA}(\phi)\}$.

Disallowing (or ignoring) administrative events during workflow execution is too strong a restriction in practice. If `Bob` leaves his company, it should be possible to remove all his role assignments thereby preventing him from executing actions in currently executing workflow runs. Similarly, if `Alice` joins a company or changes positions, i.e. gets assigned new roles, she should also be able to execute actions in workflow runs that were started prior to the organizational change. The SoD enforcement process defined below accounts for such changes.

The function upd ("update") describes how a trace of admin events changes a user assignment relation.

Definition 5 (UA change) *Let $a \in \mathcal{E}_A^*$ be a trace of admin events and UA a user assignment relation. The function* upd *is defined as follows:*

$$\mathsf{upd}(UA, a) :=$$

$$\begin{cases} UA & \text{if } a = \langle\rangle, \\ \mathsf{upd}(UA \cup \{(u,r)\}, a') & \text{if } a = (admin.\mathtt{addUA}.u.r)\hat{}\ a', \\ \mathsf{upd}(UA \setminus \{(u,r)\}, a') & \text{if } a = (admin.\mathtt{rmUA}.u.r)\hat{}\ a', \end{cases}$$

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

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

(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) If $SOD_\phi(UA)$ accepts a trace t containing no admin events and reaches a final state, then $\mathsf{users}(t) \vdash_{UA}^{\mathcal{M}} \phi$.

(R3) $SOD_\phi(UA)$ must engage in a business event $business.u.a$, if $\{u\}$ satisfies at least one maximal unit term of ϕ with respect to UA and no restriction imposed by ϕ is violated.

(R4) The semantics of the operators $^+$, $\sqcup$, $\sqcap$, $\odot$, and $\otimes$ with respect to traces must agree with their definition in SoDA$^{\mathcal{M}}$.

(R1) says that administrative events are always possible and reflected in the user assignment relation. (R2) states that in the absence of admin events, $SOD_\phi(UA)$ agrees with the SoDA$^{\mathcal{M}}$ semantics. (R3) formulates agreement with SoDA$^{\mathcal{M}}$, where for a multiset of users $\mathbf{U}$, if $\mathbf{U} \vdash_{UA}^{\mathcal{M}} \phi$, 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 a business event if the corresponding user does not contribute to the satisfaction of ϕ. As for (R4), consider for example the terms $\phi \otimes \psi$ and $\phi \otimes \psi$. It must be possible to partition a trace satisfying $\phi \otimes \psi$ or $\phi \otimes \psi$ into two subtraces, one satisfying ϕ and the other one satisfying ψ. In the case of $\phi \otimes \psi$, the users who execute business events in one trace must be disjoint from the users executing business events in the other trace. In contrast, for $\phi \odot \psi$, the multisets of users need not be disjoint.

Figure 6 illustrates abstractly how an SoD enforcement process relates to the processes introduced so far. The X-axis represents time and the Y-axis lists a workflow process, the RBAC process, and an SoD enforcement process. We distinguish between two time phases. At *design time*, a business officer defines a workflow using a workflow language that can be modeled as a CSP process W, a security officer specifies the initial RBAC configuration c_1, and a compliance officer formulates SoD constraints as a term ϕ, which is mapped to the SoD enforcement process SOD_ϕ.

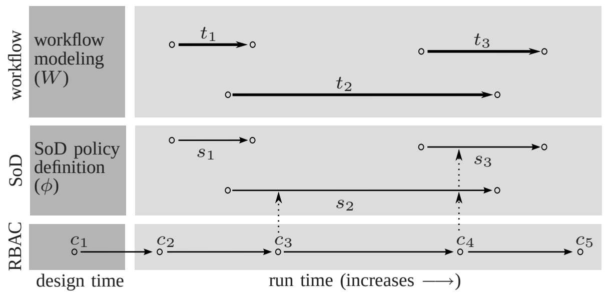


Figure 6. Relations between a workflow process, a SoD enforcement process, and the RBAC process

At *run time*, the workflow corresponding to W is executed an arbitrary number of times. Each workflow run corresponds to a trace of W, t_1, t_2, and t_3 in Figure 6. In parallel to each workflow run, an instance of SOD_ϕ is executed. For example, s_1 runs in parallel to t_1. An instance of SOD_ϕ keeps track of who has executed actions within the associated workflow run in the past and ensures that no SoD constraint is violated based on this history. The execution of the RBAC process is modeled as a single trace. Admin events change the configuration of the RBAC process. In Figure 6, the RBAC process evolves from c_1 to c_2, then to c_3 and so forth. Furthermore, RBAC configuration changes also affect the currently running instances of SOD_ϕ. For example, when the RBAC configuration of the process changes to c_4, this is reflected in s_2 and s_3 as indicated by the dotted arrows.

Without loss of generality, we look only at the execution of one instance of W, the RBAC process, and one instance of SOD_ϕ in the remainder of this paper. Furthermore, we describe the traces of W, $RBAC$, and SOD_ϕ as the single trace of the partially synchronized, parallel composition of W, $RBAC$, and SOD_ϕ. The formal definition follows.

5.2. SoDA semantics for traces

The following example shows that SoDA$^{\mathcal{M}}$ is not expressive enough to capture requirements (R1)–(R4).

Example 4 Consider the policy P that requires one action to be executed by a user acting as `Manager`

$$
(1)\ \frac{\{\mathsf{user}(b)\} \vdash^{\mathcal{M}}_{UA} \phi_{ut}}{\langle b\rangle \vdash^{\mathcal{T}}_{UA} \phi_{ut}}
\qquad
(2)\ \frac{t \vdash^{\mathcal{T}}_{UA} \phi}{t \hat{}\langle a\rangle \vdash^{\mathcal{T}}_{UA} \phi}
\qquad
(3)\ \frac{t \vdash^{\mathcal{T}}_{UA\cup\{(u,r)\}} \phi}{\langle \mathtt{addUA}.u.r\rangle \hat{} t \vdash^{\mathcal{T}}_{UA} \phi}
$$

$$
(4)\ \frac{t \vdash^{\mathcal{T}}_{UA\setminus\{(u,r)\}} \phi}{\langle \mathtt{rmUA}.u.r\rangle \hat{} t \vdash^{\mathcal{T}}_{UA} \phi}
\qquad
(5)\ \frac{\langle b\rangle \vdash^{\mathcal{T}}_{UA} \phi_{ut}}{\langle b\rangle \vdash^{\mathcal{T}}_{UA} \phi^{+}_{ut}}
\qquad
(6)\ \frac{\langle b\rangle \vdash^{\mathcal{T}}_{UA} \phi_{ut}\ ,\ \ t \vdash^{\mathcal{T}}_{UA} \phi^{+}_{ut}}{\langle b\rangle \hat{} t \vdash^{\mathcal{T}}_{UA} \phi^{+}_{ut}}
$$

$$
(7)\ \frac{t \vdash^{\mathcal{T}}_{UA} \phi}{t \vdash^{\mathcal{T}}_{UA} \phi \sqcup \psi}
\qquad
(8)\ \frac{t \vdash^{\mathcal{T}}_{UA} \psi}{t \vdash^{\mathcal{T}}_{UA} \phi \sqcup \psi}
\qquad
(9)\ \frac{t \vdash^{\mathcal{T}}_{UA} \phi\ ,\ \ t \vdash^{\mathcal{T}}_{UA} \psi}{t \vdash^{\mathcal{T}}_{UA} \phi \sqcap \psi}
$$

$$
(10)\ \frac{t_1 \vdash^{\mathcal{T}}_{UA} \phi\ ,\ \ t_2 \vdash^{\mathcal{T}}_{UA} \psi}{t \vdash^{\mathcal{T}}_{UA} \phi \odot \psi}\ \ \mathsf{si}(t,t_1,t_2)
\qquad
(11)\ \frac{t_1 \vdash^{\mathcal{T}}_{UA} \phi\ ,\ \ t_2 \vdash^{\mathcal{T}}_{UA} \psi}{t \vdash^{\mathcal{T}}_{UA} \phi \otimes \psi}\ \ \mathsf{si}(t,t_1,t_2) \text{ and } \mathsf{users}(t_1) \cap \mathsf{users}(t_2) = \emptyset
$$

Figure 7. SoDA$^{\mathcal{T}}$: semantics for traces

and another action to be executed by a user who is not acting as `Manager`. We model P by the term $\phi := \mathtt{Manager} \odot \neg\mathtt{Manager}$. Under SoDA$^{\mathcal{M}}$, ϕ can only be satisfied by a multiset of users that contains two different users. Now, consider the trace

$$
\begin{aligned}
t \quad := \quad & \langle admin.\mathtt{addUA.Bob.Manager}, \\
& business.\mathtt{Bob}.a, \\
& admin.\mathtt{rmUA.Bob.Manager}, \\
& business.\mathtt{Bob}.a' \rangle\ ,
\end{aligned}
$$

for two arbitrary actions a and a'. From (R1)–(R4), it follows that $SOD_\phi(\emptyset)$ must accept t. By (R1), $SOD_\phi(\emptyset)$ engages in $admin.\mathtt{addUA.Bob.Manager}$ and behaves like $SOD_\phi(UA)$ afterwards, for $UA = \{(\mathtt{Bob}, \mathtt{Manager})\}$. Next, $SOD_\phi(UA)$ engages in $business.\mathtt{Bob}.a$ by (R3) and (R4) because `Bob` acts as `Manager`. Again by (R1), $SOD_\phi(UA)$ engages in $admin.\mathtt{rmUA.Bob.Manager}$ and behaves like $SOD_\phi(\emptyset)$ afterwards. Finally, by (R3) and (R4), $SOD_\phi(\emptyset)$ engages in $business.\mathtt{Bob}.a'$ because `Bob` does not act as `Manager`. In the end, SOD_ϕ engaged in a business event with a user that acted as `Manager` and in another one with a user not acting as `Manager`, satisfying the policy P. However, we have $\mathsf{users}(t) = \{\mathtt{Bob}, \mathtt{Bob}\}$, which contradicts the previous statement that ϕ is only satisfied by multisets containing two different users.

This contradiction motivates the introduction of a third semantics for SoDA terms, SoDA$^{\mathcal{T}}$, that accounts for administrative changes and defines satisfaction with respect to traces.

In SoDA$^{\mathcal{T}}$, subterms correspond to separate traces that may interleave with each other in any order. Admin events, though, must occur in all traces in the same order. We formalize this relation by the *synchronized interleaving* predicate si. For traces t, t_1, and t_2, $\mathsf{si}(t, t_1, t_2)$ holds if and only if t_1 and t_2 "partition" t such that each admin event in t is contained in t_1 and t_2, and each business event is either in t_1 or t_2. More formally:

Definition 6 (Synchronized interleaving) *Let $t, t_1, t_2 \in (\mathcal{E}_B \cup \mathcal{E}_A)^*$ be traces. The* synchronized interleaving *predicate $\mathsf{si}(t, t_1, t_2)$ is defined as follows:*

$$
\mathsf{si}(t,t_1,t_2) :=
$$
$$
\begin{cases}
true & \text{if } t = \langle\rangle, t_1 = \langle\rangle \text{ and } t_2 = \langle\rangle, \\
\mathsf{si}(t', t_1', t_2') & \text{if } t = \langle a\rangle \hat{} t', t_1 = \langle a\rangle \hat{} t_1', \\
 & \text{and } t_2 = \langle a\rangle \hat{} t_2', \\
\mathsf{si}(t', t_1', t_2) & \text{if } t = \langle b\rangle \hat{} t' \text{ and } t_1 = \langle b\rangle \hat{} t_1', \\
\mathsf{si}(t', t_1, t_2') & \text{if } t = \langle b\rangle \hat{} t' \text{ and } t_2 = \langle b\rangle \hat{} t_2', \\
false & \text{otherwise,}
\end{cases}
$$

where a ranges over $\mathcal{E}_A$, b over $\mathcal{E}_B$, and t', t_1', and t_2' over $(\mathcal{E}_B \cup \mathcal{E}_A)^*$.

We illustrate si with an example.

$$
\begin{aligned}
t &:= \langle\, b_1, b_2, b_3, a_1, b_4, b_5, a_2, b_6, a_3, b_7, a_4 \,\rangle \\
t_1 &:= \langle\, b_1, \quad b_3, a_1, b_4, \quad a_2, \quad a_3, b_7, a_4 \,\rangle \\
t_2 &:= \langle \quad b_2, \quad a_1, \quad b_5, a_2, b_6, a_3, \quad a_4 \,\rangle
\end{aligned}
$$

For these three traces, $\mathsf{si}(t, t_1, t_2)$ holds.

With si at hand, we define the satisfaction of SoDA terms by traces.

Definition 7 (Trace Satisfaction SoDA$^{\mathcal{T}}$) *Let ϕ be a term, ϕ_{ut} a unit term, $a \in \mathcal{E}_A$ an admin event, and $b \in \mathcal{E}_B$ a business event. A trace $t \in (\mathcal{E}_A \cup \mathcal{E}_B)^*$* satisfies *$\phi$ with respect to the user assignment relation UA, denoted by $t \vdash^{\mathcal{T}}_{UA} \phi$, if and only if t can be derived from ϕ using the rules listed in Figure 7.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\{\texttt{Bob}\} \vdash^{\mathcal{M}}_{\{(\texttt{Bob},\texttt{Manager})\}} \texttt{Manager}}{\langle \texttt{Bob}.a \rangle \vdash^{\mathcal{T}}_{\{(\texttt{Bob},\texttt{Manager})\}} \texttt{Manager}}\,(1)}{\langle \texttt{addUA.Bob.Manager}, \texttt{Bob}.a \rangle \vdash^{\mathcal{T}}_{\emptyset} \texttt{Manager}}\,(3)}{\langle \texttt{addUA.Bob.Manager}, \texttt{Bob}.a, \texttt{rmUA.Bob.Manager} \rangle \vdash^{\mathcal{T}}_{\emptyset} \texttt{Manager}}\,(2), \quad \dfrac{\dfrac{\dfrac{\{\texttt{Bob}\} \vdash^{\mathcal{M}}_{\emptyset} \neg\texttt{Manager}}{\langle \texttt{Bob}.a' \rangle \vdash^{\mathcal{T}}_{\emptyset} \neg\texttt{Manager}}\,(1)}{\langle \texttt{rmUA.Bob.Manager}, \texttt{Bob}.a' \rangle \vdash^{\mathcal{T}}_{\{(\texttt{Bob},\texttt{Manager})\}} \neg\texttt{Manager}}\,(4)}{\langle \texttt{addUA.Bob.Manager}, \texttt{rmUA.Bob.Manager}, \texttt{Bob}.a' \rangle \vdash^{\mathcal{T}}_{\emptyset} \neg\texttt{Manager}}\,(3)}{\langle \texttt{addUA.Bob.Manager}, \texttt{Bob}.a, \texttt{rmUA.Bob.Manager}, \texttt{Bob}.a' \rangle \vdash^{\mathcal{T}}_{\emptyset} (\texttt{Manager} \odot \neg\texttt{Manager})}\,(10)$$

Figure 8. Example derivation under SoDA$^{\mathcal{T}}$

Example 5 Consider again the term ϕ and the trace t given in Example 4. Figure 8 presents a derivation that t satisfies ϕ with respect to $UA = \emptyset$. For reasons of space, we omit the channel prefixes.

SoDA$^{\mathcal{T}}$ fulfills the requirements listed in Section 5.1: (R1) follows from rules (3) and (4) of Definition 7, (R3) from the rule (1), and (R4) from the rules corresponding to the respective operators. The satisfaction of (R2) is shown by the following lemma that relates SoDA$^{\mathcal{M}}$ and SoDA$^{\mathcal{T}}$, which we prove in the appendix.

Lemma 2 *For all terms ϕ, all user assignment relations UA, and all traces $t \in \mathcal{E}_B^*$, if $t \vdash^{\mathcal{T}}_{UA} \phi$, then* $\mathsf{users}(t) \vdash^{\mathcal{M}}_{UA} \phi$.

5.3. Mapping terms to processes

First, we introduce the auxiliary process FIN that engages in an arbitrary number of admin events before it engages in $done$, and finally behaves like $STOP$.

$$FIN = (done \rightarrow STOP) \,\square\, (admin.a : \mathcal{A}_A \rightarrow FIN)$$

Using FIN, we define the mapping $[\![.]\!]^U_{UA}$.

Definition 8 (Mapping $[\![.]\!]^U_{UA}$) *Given a set of users U, a user assignment relation UA, and a term ϕ, the mapping $[\![\phi]\!]^U_{UA}$ returns a CSP process parametrized by UA. For a unit term ϕ_{ut} and terms ϕ and ψ, the mapping $[\![.]\!]^U_{UA}$ is defined in Figure 9.*

Note that the equations (1) and (2) require determining whether $\{u'\} \vdash^{\mathcal{M}}_{UA} \phi_{ut}$. This problem is analogous to testing whether a propositional formula is satisfiable under a given assignment and is also decidable in polynomial time. In particular, we can apply bottom up the rules (1)–(4) and (7)–(9) from Definition 4. The only choice point arises as to whether to apply rule (7) or (8) when evaluating a term of the form $\phi \sqcup \psi$ and here one may simply try both possibilities.

Definition 9 (SoD enforcement process) *For a term ϕ and a user assignment relation UA, the* SoD enforcement process *is the CSP process*

$$SOD_\phi(UA) := [\![\phi]\!]^{\mathcal{U}}_{UA} \,.$$

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

Definition 10 (Correctness of $[\![.]\!]^U_{UA}$) *The mapping $[\![.]\!]^U_{UA}$ is* correct *if for all terms ϕ, all user assignment relations UA, and all traces $t \in \Sigma^*$, $t \in \mathcal{T}(SOD_\phi(UA))$ and* $\mathsf{done}(t)$ *if and only if $t' \vdash^{\mathcal{T}}_{UA} \phi$, for $t = t' \hat{}\langle done \rangle$, where t' ranges over $(\mathcal{E}_B \cup \mathcal{E}_A)^*$.*

Informally, the mapping $[\![.]\!]^U_{UA}$ is correct if for every term ϕ and every user assignment relation UA, all traces of the process $SOD_\phi(UA)$ that finish with $done$ also satisfy ϕ with respect to UA under SoDA$^{\mathcal{T}}$. Conversely, every trace that satisfies ϕ with respect to UA is also in $\mathcal{T}(SOD_\phi(UA))$. We prove the following theorem in the appendix.

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

The following corollary relates a subset of traces of SoD enforcement processes and their corresponding multisets of users under the multiset semantics. It follows directly from Theorem 1 and Lemma 2.

Corollary 1 *For all terms ϕ, all user assignment relations UA, and all traces $t \in \mathcal{E}_B^*$, if $t \hat{}\langle done \rangle \in \mathcal{T}(SOD_\phi(UA))$, then* $\mathsf{users}(t) \vdash^{\mathcal{M}}_{UA} \phi$.

Given a process W that models a workflow 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.

$$\begin{aligned} &SSW_\phi(UA, PA) \\ &\qquad = (W \underset{\mathcal{E}_B}{\parallel} RBAC(UA, PA)) \underset{\Sigma}{\parallel} SOD_\phi(UA) \\ &\qquad = SW(UA, PA) \underset{\Sigma}{\parallel} SOD_\phi(UA) \,. \end{aligned}$$

Let $b := business.u.a$ be a business event. $SSW_\phi(UA, PA)$ engages in b if W, $RBAC(UA, PA)$, and $SOD_\phi(UA)$ each engage in b. In other words, b must be one of the next actions to be taken according to the workflow specification, the user u must be authorized to execute the action a according to the

$$
\begin{array}{llll}
(1) & [\![\phi_{ut}]\!]^{U}_{UA} & := & business?u : \{u' \in U \mid \{u'\} \vdash^{\mathcal{M}}_{UA} \phi_{ut}\}.a : \mathcal{A} \rightarrow FIN \\
& & & \Box\ admin.\mathtt{addUA}?u : \mathcal{U}?r : \mathcal{R} \rightarrow [\![\phi_{ut}]\!]^{U}_{UA \,\cup\, \{(u,r)\}} \\
& & & \Box\ admin.\mathtt{rmUA}?u : \mathcal{U}?r : \mathcal{R} \rightarrow [\![\phi_{ut}]\!]^{U}_{UA \,\setminus\, \{(u,r)\}} \\
(2) & [\![\phi^{+}_{ut}]\!]^{U}_{UA} & := & business?u : \{u' \in U \mid \{u'\} \vdash^{\mathcal{M}}_{UA} \phi_{ut}\}.a : \mathcal{A} \rightarrow (FIN \,\Box\, [\![\phi^{+}_{ut}]\!]^{U}_{UA}) \\
& & & \Box\ admin.\mathtt{addUA}?u : \mathcal{U}?r : \mathcal{R} \rightarrow [\![\phi^{+}_{ut}]\!]^{U}_{UA \,\cup\, \{(u,r)\}} \\
& & & \Box\ admin.\mathtt{rmUA}?u : \mathcal{U}?r : \mathcal{R} \rightarrow [\![\phi^{+}_{ut}]\!]^{U}_{UA \,\setminus\, \{(u,r)\}} \\
(3) & [\![\phi \sqcup \psi]\!]^{U}_{UA} & := & [\![\phi]\!]^{U}_{UA} \,\Box\, [\![\psi]\!]^{U}_{UA} \\
(4) & [\![\phi \sqcap \psi]\!]^{U}_{UA} & := & [\![\phi]\!]^{U}_{UA} \underset{\Sigma}{\parallel} [\![\psi]\!]^{U}_{UA} \\
(5) & [\![\phi \odot \psi]\!]^{U}_{UA} & := & [\![\phi]\!]^{U}_{UA} \underset{\{done\} \,\cup\, \mathcal{E}_A}{\parallel} [\![\psi]\!]^{U}_{UA} \\
(6) & [\![\phi \otimes \psi]\!]^{U}_{UA} & := & \underset{\{\,(U_\phi, U_\psi) \,\mid\, U_\phi \cup U_\psi = U \text{ and } U_\phi \cap U_\psi = \emptyset\}}{\Box} [\![\phi]\!]^{U_\phi}_{UA} \underset{\{done\} \,\cup\, \mathcal{E}_A}{\parallel} [\![\psi]\!]^{U_\psi}_{UA}
\end{array}
$$

Figure 9. Mapping SoDA terms to CSP processes

RBAC configuration (UA, PA), and u must not violate the SoD policy ϕ, given the history of previously executed business events and UA. Furthermore, $RBAC$ and SOD_ϕ can synchronously engage in an admin event at any time. Finally, $SSW_\phi(UA, PA)$ engages in $done$ if both W and $SOD_\phi(UA)$ synchronously engage in $done$.

Example 6 (SoD secure workflow process) Assume that the users who execute actions in our payment workflow must comply with the SoD policy described by the term ϕ of Example 3. Example 2 shows that $t' \in \mathcal{T}(SW(UA, PA))$. In contrast, $t' \notin \mathcal{T}(SSW_\phi(UA, PA))$ because `Bob` is not allowed to execute both actions `prepare check` and `approve payment`. Hence, the SoD secure workflow process reduces the risk of fraudulent payments pointed out in Example 2. We change t' to t'' by adding the admin event $admin.$`addUA.Claire.Manager` and let `Claire` execute the `approve payment` action.

$$
\begin{array}{lll}
t'' & := & \langle business.\mathtt{Alice.receive\ invoice}, \\
& & business.\mathtt{Bob.prepare\ check}, \\
& & admin.\mathtt{addUA.Bob.Manager}, \\
& & admin.\mathtt{addUA.Claire.Manager}, \\
& & business.\mathtt{Claire.approve\ payment}, \\
& & business.\mathtt{Alice.issue\ check},\ done\rangle
\end{array}
$$

The new admin event adds the role assignment $(\mathtt{Claire}, \mathtt{Manager})$ to SSW_ϕ's RBAC configuration as shown in Figure 2 by the dashed line. The trace t'' without $done$ satisfies ϕ with respect to UA under SoDA$^{\mathcal{T}}$. Furthermore, $t'' \in \mathcal{T}(SSW_\phi(UA, PA))$.

This completes our running example and illustrates how the three kinds of processes presented in this paper interact and how each of them enforces its corresponding policy: 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.

5.4. From processes to enforcement monitors

CSP's *operational semantics* interprets a process as a *labeled transition system (LTS)*. It is straightforward to translate an LTS into a program that, if synchronized with its environment, only allows the execution of actions as defined by the CSP process. The program thereby constitutes an enforcement monitor for the policy specified by the CSP process. The mapping $[\![\cdot]\!]^{U}_{UA}$ may yield a nondeterministic process. However, the corresponding LTS can either be determinized or the enforcement monitor can keep track of the set of reachable states after each transition, essentially performing a power-set construction, on-the-fly.

As shown in Section 5.3, an SoD secure workflow process is the parallel execution of three subprocesses, each responsible for a specific task. Due to the associativity of CSP's $||$-operator, these three processes can be grouped in any order. Furthermore, the set of events on which these processes synchronize defines the kinds of events each process engages in. Therefore, any subset of these three processes can be mapped to an enforcement monitor and the set of events synchronized with the remaining processes specifies the interface of the monitor. 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 admin events with the existing components.

6. Related work

There exist many formalisms for modeling workflows. Examples include the Workflow Reference Model [15], Activity Diagrams from the Unified Modeling Language (UML) [22], the Business Process Modeling Notation (BPMN) [23], and the Web Services Business Process Execution Language (WS-BPEL) [2]. Different formal semantics for these modeling languages have been given [6,10,20,24,25,38]. In particular, process algebras have been used to give a formal semantics to workflow languages, for example a semantics for BPMN is given using CSP [38].

There are also numerous models and frameworks to formalize and enforce separation of duty constraints. For a taxonomy of SoD constraints and SoD enforcement mechanisms, see for example [33] and [12]. Because static SoD enforcement mechanisms are considered not flexible enough for real-world settings, research has focused on dynamic enforcement. To the best of our knowledge, our work is the first to model dynamic enforcement of SoD constraints with changing role assignments.

In the seminal work of Sandhu [28], *Transaction Control Expressions* are introduced as a notation for defining dynamic SoD constraints on data objects; enforcement decisions are made at run-time, based on the history of executed actions. In this notation, a workflow, associated with a data object, is defined by a list of actions, each with one or more attached roles. A user is authorized to execute an action if she acts in one of the corresponding roles. By default, all actions must be executed by different users. Constraints are less expressive than SoDA terms and they can only be defined along with a concrete workflow.

Similar to [28], in Nash and Poland's *object-based separation of duties* [21], 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. The same functionality can be modeled with our formalism if every data object is protected by an SoD enforcement process.

In [3], Bertino, Ferrari, and Atluri check the consistency of constraints defined over workflows in a logical framework. Constraints are defined with respect to the sequence of individual workflow actions, applying (first-order) predicates to action occurrences. No mapping to an enforcement mechanism is given. Schaad, Lotz, and Sohr formalize SoD constraints in linear temporal logic (LTL) and check for safety violations using a model checker, taking delegation and revocation of access rights into account [30]. However, a declarative language for SoD constraints and its mapping to LTL is not provided. Wolter, Schaad, and Meinel extend BPMN to support specifying SoD constraints during workflow modeling [37]. Their constraints, which are less expressive than SoDA terms, are mapped to XACML policies enforced at run time. XACML policies, however, do not allow for changing user assignment relations.

In the framework of Knorr and Stormer [17], dynamic SoD constraints are graphically defined and mapped to Prolog clauses along with workflows that are modeled as Petri nets. The resulting Prolog program computes all workflow runs that do not violate the specified SoD constraints. Similar to [3,28], and in contrast to our results, SoD constraints can only be described with respect to a concrete workflow.

In [35], Wang and Li also presented an enforcement mechanism for SoDA terms. In contrast to our work, their approach is static and not applicable to all combinations of terms, roles, and permission assignment relations. In particular, the use of the $\neg$-operator can invalidate a large subset of assignment relations.

7. Conclusions

In this paper, we showed how to map SoDA terms onto workflows in a general way that also supports administrative actions. The key ideas were (1) to extend the semantics of SoDA to traces, handling both multiple actions by users and administrative actions, and (2) to map SoDA terms to processes, which interact with workflow and access control processes. Because all components are defined in CSP, we can directly employ CSP's operational semantics to map these processes to a workflow engine that performs the necessary security checks.

As future work, we will explore how to best implement our SoDA processes and integrate them with existing workflow engines. Efficiency is a central question in this regard. In our mapping to CSP, we focused on providing an abstract specification of a SoDA enforcement mechanism, rather than an efficient one. In particular, the rule (6) in Figure 9 yields a state space that is exponential in the number of system users. We will investigate translations with improved complexity and the use of data-structures for efficiently representing extended state-machines. We will also explore optimization techniques, e.g., pruning the state space to eliminate states of workflow runs from which no final state can be reached, no matter which changes are made to the RBAC configuration. The results of [3,18,36] should be helpful in this regard.

Acknowledgments

We thank Felix Klaedtke, Samuel Müller, and Christoph Sprenger for many inspiring discussions and their helpful reviews of this work.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement N° FP7-216917.

References

[1] "Sarbanes-Oxley Act of 2002," United States federal law, January 2002.

[2] *Web Services Business Process Execution Language (WS-BPEL)*, OASIS Standard, v. 2.0, 2007.

[3] E. Bertino, E. Ferrari, and V. Atluri, "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 2, no. 1, pp. 65–104, 1999.

[4] D. D. Clark and D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," in *Proc. IEEE Symposium on Security and Privacy (S&P'87)*, 1987, pp. 184–194.

[5] ——, "Evolution of a Model for Computer Integrity," in *Proc. National Computer Security Conference (NCSC'88)*, 1988, pp. 14–27.

[6] R. M. Dijkman, M. Dumas, and C. Ouyang, "Formal Semantics and Analysis of BPMN Process Models using Petri Nets," Queensland University of Technology, Tech. Rep., 2007.

[7] "Enron, see you in court," *The Economist*, November 15th, 2001.

[8] D. Ferraiolo and R. Kuhn, "Role-Based Access Control," in *Proc. National Computer Security Conference (NCSC'92)*, 1992, pp. 554–563.

[9] D. F. Ferraiolo, R. S. Sandhu, S. I. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.

[10] A. Ferrara, "Web Services: a Process Algebra Approach," in *Proc. International Conference on Service Oriented Computing (ICSOC'04)*, 2004, pp. 242–251.

[11] "Failures-Divergence Refinement, User Manual,", Formal Systems (Europe) Ltd., www.fsel.com, 2005.

[12] V. D. Gligor, S. I. Gavrila, and D. Ferraiolo, "On the Formal Definition of Separation-of-Duty Policies and their Composition," in *Proc. IEEE Symposium on Security and Privacy (S&P'98)*, 1998, pp. 172–183.

[13] R. J. Heffernan, M. D. Moberly, K. E. Peterson, and L. Runyon, "Trends in Proprietary Information Loss," ASIS Foundation, Survey Report, 2007.

[14] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice Hall International and Hoare, 2004.

[15] D. Hollingsworth, "The workflow reference model," Workflow Management Coalition Standard TC003v11, 1995.

[16] IT Governance Institute, *COBIT 4.0*, 2005.

[17] K. Knorr and H. Stormer, "Modeling and Analyzing Separation of Duties in Workflow Environments," in *Proc. International Conference on Information Security*, 2001, pp. 199–212.

[18] M. Kohler and A. Schaad, "Avoiding Policy-based Deadlocks in Business Processes," in *Proc. International Conference on Availability, Reliability and Security (ARES'08)*, 2008, pp. 709–716.

[19] N. Li and Q. Wang, "Beyond Separation of Duty: An Algebra for Specifying High-Level Security Policies," in *Proc. ACM Conference on Computer and Communications Security (CCS'06)*, 2006, pp. 356–369.

[20] R. Lucchi and M. Mazzara, "A $\pi-$Calculus Based Semantics for WS-BPEL," *Journal of Logic and Algebraic Programming*, vol. 70, no. 1, pp.96–118, 2007.

[21] M. J. Nash and K. R. Poland, "Some Conundrums Concerning Separation of Duty," in *Proc. IEEE Symposium on Security and Privacy (S&P'90)*, 1990, pp. 201–207.

[22] *Unified Modeling Language (UML)*, Object Management Group (OMG) Standard, v. 2.1.2, 2007.

[23] *Business Process Modeling Notation (BPMN)*, Object Management Group (OMG) Standard, v. 1.1, 2008.

[24] C. Ouyang, E. Verbeek, W. M. P. van der Aalst, S. Breutel, M. Dumas, and A. H. M. ter Hofstede, "Formal Semantics and Analysis of Control Flow in WS-BPEL," *Science of Computer Programming*, vol. 67, no. 2-3, pp. 162–198, 2007.

[25] F. Puhlmann and M. Weske, "Using the $\pi-$Calculus for Formalizing Workflow Patterns," in *Proc. International Conference on Business Process Management (BPM'05)* , 2005, pp. 153–168.

[26] A. W. Roscoe, *The Theory and Practice of Concurrency*. Pearson and Roscoe, 2005.

[27] J. Saltzer and M. Schroeder, "The Protection of Information in Computer Systems," *Proceeding of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[28] R. S. Sandhu, "Transaction Control Expressions for Separation of Duties," in *Proc. IEEE Aerospace Computer Security Applications Conference*, 1988, pp. 282–286.

[29] R. S. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[30] A. Schaad, V. Lotz, and K. Sohr, "A Model-checking Approach to Analysing Organisational Controls in a Loan Origination Process," in *Proc. ACM Symposium on Access Control Models and Technologies (SACMAT'06)*, 2006, pp. 139–149.

[31] A. Schaad and J. Moffett, "Separation, Review and Supervision Controls in the Context of a Credit Application Process – A Case Study of Organisational Control Principles," in *Proc. ACM Symposium on Applied Computing (SAC'04)*, 2004, pp. 1380–1384.

[32] A. Schaad, P. Spadone, and H. Weichsel, "A Case Study of Separation of Duty Properties in the Context of the Austrian 'eLaw' Process," in *Proc. ACM Symposium on Applied Computing (SAC'05)*, 2005, pp. 1328–1332.

[33] R. Simon and M. E. Zurko, "Separation of Duty in Role-based Environments," in *Proc. IEEE Workshop on Computer Security Foundations (CSFW'97)*, 1997, pp. 183–194.

[34] A. Syropoulos, "Mathematics of Multisets," in *Workshop on Multiset Processing*, 2000, pp. 347–358.

[35] Q. Wang and N. Li, "Direct Static Enforcement of High-Level Security Policies," in *Proc. ACM Symposium on Information, Computer and Communications Security (ASIACCS'07)*, 2007, pp. 214–225.

[36] ——, "Satisfiability and Resiliency in Workflow Systems," in *Proc. European Symposium On Research In Computer Security (ESORICS'07)*, 2007, pp. 90–105.

[37] C. Wolter, A. Schaad, and C. Meinel, "Deriving XACML Policies from Business Process Models," in *Proc. International Workshop on Web Information Systems Engineering (WISE'07)*, 2007, pp. 142–153.

[38] P. Y. H. Wong and J. Gibbons, "A Process-Algebraic Approach to Workflow Specification and Refinement," in *Proc. International Symposium on Software Composition (SC'07)*, 2007, pp. 51–65.

Appendix

1. Li and Wang's semantics for SoDA

We summarize $\text{SoDA}^{\mathcal{S}}$, the semantics for SoDA terms that was originally introduced by Li and Wang in [19]. They define satisfaction with respect to a tuple (U, UR), where $U \subseteq \mathcal{U}$ and $UR \subseteq U \times \mathcal{R}$. This is in contrast to our $\text{SoDA}^{\mathcal{M}}$ semantics, which is defined with respect to just the user assignment relation UA.

Definition 11 (Set Satisfaction $\text{SoDA}^{\mathcal{S}}$) *Let ϕ be a term, $S \subseteq \mathcal{U}$ a non-empty set of users, $U \subseteq \mathcal{U}$ a set of users, $r \in \mathcal{R}$ a role, and $UR \subseteq U \times \mathcal{R}$. A set of users X* satisfies ϕ with respect to the tuple (U, UR), *denoted by $\mathbf{U} \vdash^{\mathcal{S}}_{(U,UR)} \phi$, if and only if X can be derived from ϕ using the rules listed in Figure 10.*

The following examples illustrate the expressivity of SoDA terms.

- The term $(\mathsf{All} \otimes \mathsf{All}) \otimes \mathsf{All}$ is satisfied by three different users.
- The term $\texttt{Manager} \sqcup (\texttt{Clerk} \otimes \texttt{Clerk})$ is satisfied by either a user acting as `Manager` or two different users, both acting as `Clerk`.
- The term $(\texttt{Manager} \otimes \texttt{Accountant}) \sqcap (\texttt{Clerk} \sqcap \neg\{\texttt{Alice}, \texttt{Bob}\})^{+}$ requires a user acting as `Manager` and another user acting as `Accountant`. In addition, both users must act as `Clerk` and must not be `Alice` or `Bob`.
- The term $(\texttt{Manager} \sqcap \neg\texttt{Accountant}) \sqcup (\neg\texttt{Manager} \sqcap \texttt{Accountant}))^{+}$ is satisfied by a non-empty set of users, each either acting as `Manager` or as `Accountant` but not both.

2. Proofs

In the following, we refer to rule i of Definition 3 as (MUi), to rule j of Definition 7 as (TRj), to rule k of Definition 8 as (MAk), and to rule l of Definition 11 as (SEl), respectively.

$$(1)\quad \frac{}{\{u\} \vdash^{\mathcal{S}}_{(U,UR)} \mathsf{All}}\quad u \in U \qquad\qquad (2)\quad \frac{}{\{u\} \vdash^{\mathcal{S}}_{(U,UR)} r}\quad (u,r) \in UR$$

$$(3)\quad \frac{}{\{u\} \vdash^{\mathcal{S}}_{(U,UR)} S}\quad u \in (S \cap U) \qquad\qquad (4)\quad \frac{\{u\} \nvdash^{\mathcal{S}}_{(U,UR)} \phi}{\{u\} \vdash^{\mathcal{S}}_{(U,UR)} \neg\phi}$$

$$(5)\quad \frac{\{u\} \vdash^{\mathcal{S}}_{(U,UR)} \phi}{\{u\} \vdash^{\mathcal{M}}_{UA} \phi^+} \qquad\qquad (6)\quad \frac{\{u\} \vdash^{\mathcal{S}}_{(U,UR)} \phi,\ X \vdash^{\mathcal{M}}_{UA} \phi^+}{(\{u\} \cup X) \vdash^{\mathcal{M}}_{UA} \phi^+}$$

$$(7)\quad \frac{X \vdash^{\mathcal{S}}_{(U,UR)} \phi}{X \vdash^{\mathcal{S}}_{(U,UR)} (\phi \sqcup \psi)} \qquad\qquad (8)\quad \frac{X \vdash^{\mathcal{S}}_{(U,UR)} \psi}{X \vdash^{\mathcal{S}}_{(U,UR)} (\phi \sqcup \psi)}$$

$$(9)\quad \frac{X \vdash^{\mathcal{S}}_{(U,UR)} \phi,\ X \vdash^{\mathcal{S}}_{(U,UR)} \psi}{X \vdash^{\mathcal{S}}_{(U,UR)} (\phi \sqcap \psi)} \qquad\qquad (10)\quad \frac{X \vdash^{\mathcal{S}}_{(U,UR)} \phi,\ Y \vdash^{\mathcal{S}}_{(U,UR)} \psi}{(X \cup Y) \vdash^{\mathcal{S}}_{(U,UR)} (\phi \odot \psi)}$$

$$(11)\quad \frac{X \vdash^{\mathcal{S}}_{(U,UR)} \phi,\ Y \vdash^{\mathcal{S}}_{(U,UR)} \psi}{(X \cup Y) \vdash^{\mathcal{S}}_{(U,UR)} (\phi \otimes \psi)}\quad (X \cap Y) = \emptyset\ .$$

Figure 10. SoDA$^{\mathcal{S}}$: semantics for sets of users

2.1. Proof of Lemma 1.

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

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

Proof: The only rules of Definition 3 that allow for the derivation of a unit term are $(MU1)$–$(MU4)$ and $(MU7)$–$(MU9)$. By their respective definitions, the rules $(MU1)$–$(MU4)$ have a multiset containing exactly one user as their conclusion. 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 $u \in \mathcal{U}$, $\{u\} \vdash^{\mathcal{M}}_{UA} \phi_{ut}$ if and only if $\{u\} \vdash^{\mathcal{S}}_{\mathsf{lwconf}(UA)} \phi_{ut}$.*

Proof: We proceed in two steps. We prove for all unit terms ϕ_{ut}, all user assignment relations UA, and all users $u \in \mathcal{U}$, (1) $\{u\} \vdash^{\mathcal{M}}_{UA} \phi_{ut} \Rightarrow \{u\} \vdash^{\mathcal{S}}_{\mathsf{lwconf}(UA)} \phi_{ut}$ and (2) $\{u\} \vdash^{\mathcal{M}}_{UA} \phi_{ut} \Leftarrow \{u\} \vdash^{\mathcal{S}}_{\mathsf{lwconf}(UA)} \phi_{ut}$. In both cases we reason by induction on the structure of ϕ_{ut}. Let UA and u be given and $(U, UR) := \mathsf{lwconf}(UA)$.

(1) $\Rightarrow$-direction:

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

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

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

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 $\{u\} \vdash^{\mathcal{M}}_{UA} \neg\phi_{ut}$. By $(MU4)$, $\{u\} \nvdash^{\mathcal{M}}_{UA} \phi_{ut}$. From the induction hypothesis, it follows that $\{u\} \nvdash^{\mathcal{S}}_{(U,UR)} \phi_{ut}$. Therefore, $\{u\} \vdash^{\mathcal{S}}_{(U,UR)} \neg\phi_{ut}$ by $(SE4)$.

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

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

(2) $\Leftarrow$-direction:

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

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

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

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 $\{u\} \vdash^{\mathcal{S}}_{(U,UR)} \neg\phi_{ut}$. By $(SE4)$, $\{u\} \nvdash^{\mathcal{S}}_{(U,UR)} \phi_{ut}$. From the induction hypothesis, it follows that $\{u\} \nvdash^{\mathcal{M}}_{UA} \phi_{ut}$. Therefore, $\{u\} \vdash^{\mathcal{M}}_{UA} \neg\phi_{ut}$ by $(MU4)$.

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

Consider the term $\phi_{ut} \sqcap \psi_{ut}$ and let $\{u\} \vdash^{\mathcal{S}}_{(U,UR)} \phi_{ut} \sqcap \psi_{ut}$. By $(SE9)$, $\{u\} \vdash^{\mathcal{S}}_{(U,UR)} \phi_{ut}$ and $\{u\} \vdash^{\mathcal{S}}_{(U,UR)} \psi_{ut}$. By the induction hypothesis, $\{u\} \vdash^{\mathcal{M}}_{UA} \phi_{ut}$ and $\{u\} \vdash^{\mathcal{M}}_{UA} \psi_{ut}$. Therefore, $\{u\} \vdash^{\mathcal{M}}_{UA} \phi_{ut} \sqcap \psi_{ut}$ 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 $\mathbf{U} \vdash^{\mathcal{M}}_{UA} \phi_{ut}$. By Proposition 1, $\mathbf{U} = \{u\}$, for a user $u \in \mathcal{U}$. From Proposition 2 it follows that $\{u\} \vdash^{\mathcal{S}}_{(U,UR)} \phi_{ut}$. By the definition of $\mathsf{userset}$, $\{u\} = \mathsf{userset}(\mathbf{U})$ and therefore $\mathsf{userset}(\mathbf{U}) \vdash^{\mathcal{S}}_{(U,UR)} \phi_{ut}$.

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

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

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

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

Consider the term $\phi \otimes \psi$ and let $\mathbf{U} \vdash^{\mathcal{M}}_{UA} \phi \otimes \psi$. By $(MU11)$, there are two multisets of users $\mathbf{V}$ and $\mathbf{W}$ such that $\mathbf{V} \vdash^{\mathcal{M}}_{UA} \phi$, $\mathbf{W} \vdash^{\mathcal{M}}_{UA} \psi$, and $\mathbf{V} \cap \mathbf{W} = \emptyset$. By the induction hypothesis, $\mathsf{userset}(\mathbf{V}) \vdash^{\mathcal{S}}_{(U,UR)} \phi$ and $\mathsf{userset}(\mathbf{W}) \vdash^{\mathcal{S}}_{(U,UR)} \psi$. 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 $(SE11)$ $\mathsf{userset}(\mathbf{U}) \vdash^{\mathcal{S}}_{(U,UR)} \psi \otimes \psi$. □

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

Proposition 3 *For* $t, t_1, t_2 \in \Sigma^*$, *if* $\mathsf{si}(t, t_1, t_2)$ *then* $\mathsf{users}(t) = \mathsf{users}(t_1) \uplus \mathsf{users}(t_2)$.

Proof: By Definition 6, each business event in t is either in t_1 or t_2, but not in both. Therefore, $\mathsf{users}(t) = \mathsf{users}(t_1) \uplus \mathsf{users}(t_2)$ since the function users returns the multiset of users that are contained in the business events of its argument. □

Proposition 4 *For* $t \in \mathcal{E}_B^*$, $t_1, t_2 \in \Sigma^*$, *if* $\mathsf{si}(t, t_1, t_2)$ *then* $t_1 \in \mathcal{E}_B^*$ *and* $t_2 \in \mathcal{E}_B^*$.

Proof: By Definition 6, each event that is in t_1 or t_2 is also in t. Since $t \in \mathcal{E}_B^*$, we therefore have that t_1 and t_2 contain only business events. □

Proof of Lemma 2: 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 $t \in \mathcal{E}_B^*$. Let $t \vdash^{\mathcal{T}}_{UA} \phi_{ut}$. The only rules of Definition 7 that have a unit term as their conclusion are $(TR1)$–$(TR4)$. Because t contains no admin events, only $(TR1)$ is applicable. Therefore, t must be of the form $\langle b \rangle$, for a business event $b \in \mathcal{E}_B$. By $(TR1)$, $\{\mathsf{user}(b)\} \vdash^{\mathcal{M}}_{UA} \phi_{ut}$, which is equivalent to $\mathsf{users}(t) \vdash^{\mathcal{M}}_{UA} \phi_{ut}$ by the definition of users.

Consider a term of the form ϕ_{ut}^+ and a trace $t \in \mathcal{E}_B^*$. Let $t \vdash^{\mathcal{T}}_{UA} \phi_{ut}^+$. The only rules of Definition 7 that have a term of the form ϕ_{ut}^+ as the conclusion are $(TR2)$–$(TR6)$. Because t contains no admin events, only $(TR5)$ and $(TR6)$ are applicable. For both rules, the trace t must contain at least one business event b such that $\langle b\rangle \vdash^{\mathcal{T}}_{UA} \phi_{ut}$. By the same argument as in the unit term case, it follows that $\mathsf{users}(\langle b\rangle) \vdash^{\mathcal{M}}_{UA} \phi_{ut}$ and therefore, by $(MU5)$, $\mathsf{users}(\langle b\rangle) \vdash^{\mathcal{M}}_{UA} \phi^{+}_{ut}$. By induction over the length of t, with $\langle b\rangle$ as the induction basis, it follows that $\mathsf{users}(t) \vdash^{\mathcal{M}}_{UA} \phi^{+}_{ut}$ 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 $t \in \mathcal{E}^*_B$. Let $t \vdash^{\mathcal{T}}_{UA} \phi \sqcup \psi$. By $(TR7)$ and $(TR8)$, either $t \vdash^{\mathcal{T}}_{UA} \phi$ or $t \vdash^{\mathcal{T}}_{UA} \psi$. In the first case, by the induction hypothesis, $\mathsf{users}(t) \vdash^{\mathcal{M}}_{UA} \phi$ and therefore $\mathsf{users}(t) \vdash^{\mathcal{M}}_{UA} \phi \sqcup \psi$ by $(MU7)$. The second case is analogous. Hence, $\mathsf{users}(t) \vdash^{\mathcal{M}}_{UA} \phi \sqcup \psi$.

Consider the term $\phi \sqcap \psi$ and a trace $t \in \mathcal{E}^*_B$. Let $t \vdash^{\mathcal{T}}_{UA} \phi \sqcap \psi$. By $(TR9)$, $t \vdash^{\mathcal{T}}_{UA} \phi$ and $t \vdash^{\mathcal{T}}_{UA} \psi$. From the induction hypothesis, $\mathsf{users}(t) \vdash^{\mathcal{M}}_{UA} \phi$ and $\mathsf{users}(t) \vdash^{\mathcal{M}}_{UA} \psi$. Therefore, $\mathsf{users}(t) \vdash^{\mathcal{M}}_{UA} \phi \sqcap \psi$ by $(MU9)$.

Consider the term $\phi \odot \psi$ and a trace $t \in \mathcal{E}^*_B$. Let $t \vdash^{\mathcal{T}}_{UA} \phi \odot \psi$. By $(TR10)$, there exist two traces t_1 and t_2 such that $\mathsf{si}(t, t_1, t_2)$, $t_1 \vdash^{\mathcal{T}}_{UA} \phi$, and $t_2 \vdash^{\mathcal{T}}_{UA} \psi$. By Proposition 4, t_1 and t_2 consist only of admin events because $t \in \mathcal{E}^*_B$. Therefore, from the induction hypothesis, $\mathsf{users}(t_1) \vdash^{\mathcal{M}}_{UA} \phi$ and $\mathsf{users}(t_2) \vdash^{\mathcal{M}}_{UA} \psi$. Moreover, by Proposition 3, $\mathsf{users}(t) = \mathsf{users}(t_1) \uplus \mathsf{users}(t_2)$. Hence, $\mathsf{users}(t) \vdash^{\mathcal{M}}_{UA} \phi \odot \psi$ by $(MU10)$.

Finally, consider the term $\phi \otimes \psi$ and a trace $t \in \mathcal{E}^*_B$. Let $t \vdash^{\mathcal{T}}_{UA} \phi \otimes \psi$. By $(TR11)$, there exist two traces t_1 and t_2 such that $\mathsf{si}(t, t_1, t_2)$, $\mathsf{users}(t_1) \cap \mathsf{users}(t_2) = \emptyset$, $t_1 \vdash^{\mathcal{T}}_{UA} \phi$, and $t_2 \vdash^{\mathcal{T}}_{UA} \psi$. By Proposition 4, t_1 and t_2 consist only of admin events because $t \in \mathcal{E}^*_B$. Therefore, from the induction hypothesis, $\mathsf{users}(t_1) \vdash^{\mathcal{M}}_{UA} \phi$ and $\mathsf{users}(t_2) \vdash^{\mathcal{M}}_{UA} \psi$. Furthermore, by Proposition 3, $\mathsf{users}(t) = \mathsf{users}(t_1) \uplus \mathsf{users}(t_2)$. Therefore, since $\mathsf{users}(t_1) \cap \mathsf{users}(t_2) = \emptyset$, $\mathsf{users}(t) \vdash^{\mathcal{M}}_{UA} \phi \otimes \psi$ by $(MU11)$. □

2.3. Proof of Theorem 1. We establish four auxiliary propositions and prove Theorem 1 afterwards. For a trace t ending with *done*, we use the convention that $\tilde{t}$ denotes the trace t without the last event *done*, i.e., if $\mathsf{done}(t)$, then $t = \tilde{t}\,\hat{}\,\langle done\rangle$. Recall Definition 10. We prove that for all terms ϕ, all user assignments UA, and all traces $t \in \Sigma^*$, $t \in \mathcal{T}(SOD_\phi(UA))$ and $\mathsf{done}(t)$ if and only if $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi$. We refer to the left-hand side as LHS, and to the right-hand side as RHS.

Proposition 5 *For a term ϕ, a trace $t \in \Sigma^*$, a trace of admin events $a \in \mathcal{E}^*_A$, a user assignment relations UA, and $UA' := \mathsf{upd}(UA, a)$, $t \vdash^{\mathcal{T}}_{UA'} \phi$ if and only if $a\hat{}t \vdash^{\mathcal{T}}_{UA} \phi$.*

Proof sketch: The proof is by induction on a using $(TR3)$, $(TR4)$, and Definition 5.

Proposition 6 *For a user assignment relations UA, a term ϕ, a trace $t \in \Sigma^*$, and a trace of admin events $a \in \mathcal{E}^*_A$, $t \vdash^{\mathcal{T}}_{UA} \phi$ if and only if $t\hat{}a \vdash^{\mathcal{T}}_{UA} \phi$.*

Proof: This follows directly by applying $(TR2)$ for each admin event in a to $t \vdash^{\mathcal{T}}_{UA} \phi$. □

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 a business event b, if $\mathsf{user}(b) \in U$.*

Proof: Let b be a business event. 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 b, if $\mathsf{user}(b) \in U$. 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 b if either $[\![\phi]\!]^U_{UA}$ or $[\![\psi]\!]^U_{UA}$ engage in b. By $(MA6)$, the process $[\![\phi \otimes \psi]\!]^U_{UA}$ only engages in b if either $[\![\phi]\!]^{U'}_{UA}$ or $[\![\psi]\!]^{U'}_{UA}$ engage in b, for $U' \subseteq U$. From the induction hypothesis, it follows that all processes only engage in b if $\mathsf{user}(b) \in U$. □

Proposition 8 *For three traces $t, t_1, t_2 \in (\mathcal{E}_B \cup \mathcal{E}_A)^*$ and two processes $P, Q \in \mathcal{P}$, if $t_1 \in \mathcal{T}(P)$, $t_2 \in \mathcal{T}(Q)$ and $\mathsf{si}(t, t_1, t_2)$, then $t \in \mathcal{T}(P \underset{\mathcal{E}_A \cup \{done\}}{\|} Q)$.*

Proof sketch: The proof is by induction over t. Proposition 8 follows by the definition of the $||$-operator under the denotational semantics of CSP and by Definition 6. The synchronization on *done* can be ignored because t, t_1, and t_2 do not contain *done*.

Proof of Theorem 1: We proceed in two steps. We prove that for all terms, all user assignment relations UA, and all traces $t \in \Sigma^*$, (1) $LHS \Rightarrow RHS$ and (2) $LHS \Leftarrow RHS$. Let UA be given. In both cases we reason by induction on the structure of ϕ.

(1) $LHS \Rightarrow RHS$:

Base cases: Consider a unit term ϕ_{ut} and let $t \in \mathcal{T}(SOD_{\phi_{ut}}(UA))$ and $\mathsf{done}(t)$. By $(MA1)$ and the denotational semantics of CSP, t is of the form $a_1\hat{}\,\langle b\rangle\hat{}\,a_2\hat{}\,\langle done\rangle$, for $a_1, a_2 \in \mathcal{E}^*_A$ and $b \in \mathcal{E}_B$. Let $UA' := \mathsf{upd}(UA, a_1)$. Because $[\![\phi_{ut}]\!]^{\mathcal{U}}_{UA'}$ engages in b, $\{\mathsf{user}(b)\} \vdash^{\mathcal{M}}_{UA'} \phi_{ut}$ by $(MA1)$. From $(TR1)$ follows that $\langle b\rangle \vdash^{\mathcal{T}}_{UA'} \phi_{ut}$. Therefore, by Proposition 5, $a_1\hat{}\langle b\rangle \vdash^{\mathcal{T}}_{UA} \phi_{ut}$ and by Proposition 6 $a_1\hat{}\langle b\rangle\hat{}a_2 \vdash^{\mathcal{T}}_{UA} \phi_{ut}$. Hence, $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi_{ut}$.

Consider a term of the form ϕ^{+}_{ut} and let $t \in \mathcal{T}(SOD_{\phi^{+}_{ut}}(UA))$ and $\mathsf{done}(t)$. By $(MA2)$ and the denotational semantics of CSP, t is of the form $a_1\hat{}\,\langle b_1\rangle\hat{}\,\ldots\hat{}\,a_n\hat{}\,\langle b_n\rangle\hat{}\,a_{n+1}\hat{}\,\langle done\rangle$, for $a_i \in \mathcal{E}_A^*$, $a_{n+1} \in \mathcal{E}_A^*$, and $b_i \in \mathcal{E}_B$, 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 $a_1\hat{}\,\langle b_1\rangle\hat{}\,a_2 \vdash^{\mathcal{T}}_{UA} \phi_{ut}$. By $(TR5)$, $a_1\hat{}\,\langle b_1\rangle\hat{}\,a_2 \vdash^{\mathcal{T}}_{UA} \phi^+_{ut}$. We now assume $n > 1$ and $a_2\hat{}\,\langle b_2\rangle\hat{}\,a_3\hat{}\,\ldots\hat{}\,a_n\hat{}\,\langle b_n\rangle\hat{}\,a_{n+1} \vdash^{\mathcal{T}}_{UA'} \phi^+_{ut}$, for $UA' := \mathsf{upd}(UA, a_1)$. Because $[\![\phi_{ut}]\!]^{\mathcal{U}}_{UA'}$ engages in b_1, $\{\mathsf{user}(b_1)\} \vdash^{\mathcal{M}}_{UA'} \phi_{ut}$ by $(MA2)$. From $(TR1)$ it follows that $\langle b_1\rangle \vdash^{\mathcal{T}}_{UA'} \phi_{ut}$ and from $(TR6)$ that $\langle b_1\rangle\hat{}a_2\hat{}\langle b_2\rangle\hat{}a_3\hat{}\ldots\hat{}a_n\hat{}\langle b_n\rangle\hat{}a_{n+1} \vdash^{\mathcal{T}}_{UA'} \phi^+_{ut}$. By Proposition 5, $a_1\hat{}\langle b_1\rangle\hat{}a_2\hat{}\langle b_2\rangle\hat{}a_3\hat{}\ldots\hat{}a_n\hat{}\langle b_n\rangle\hat{}a_{n+1} \vdash^{\mathcal{T}}_{UA'} \phi^+_{ut}$ and hence $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi^+_{ut}$.

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

Consider the term $\phi \sqcap \psi$, let $t \in \mathcal{T}(SOD_{\phi\sqcap\psi}(UA))$, and $\mathsf{done}(t)$. By $(MA4)$, $SOD_{\phi\sqcap\psi}(UA) = SOD_\phi(UA) \underset{\Sigma}{\parallel} SOD_\psi(UA)$. From the denotational semantics of CSP follows that $t \in \mathcal{T}(SOD_\phi(UA))$ and $t \in \mathcal{T}(SOD_\psi(UA))$. By the induction hypothesis, $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi$ and $\tilde{t} \vdash^{\mathcal{T}}_{UA} \psi$. By $(TR9)$, $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi \sqcap \psi$.

Consider the term $\phi \odot \psi$, let $t \in \mathcal{T}(SOD_{\phi\odot\psi}(UA))$, and $\mathsf{done}(t)$. By $(MA5)$, $SOD_{\phi\odot\psi}(UA) = SOD_\phi(UA) \underset{\{done\}\,\cup\,\mathcal{E}_A}{\parallel} SOD_\psi(UA)$. From the denotational semantics of CSP follows that there are two traces $t_\phi, t_\psi \in \Sigma^*$ such that $t_\phi \in \mathcal{T}(SOD_\phi(UA))$ and $t_\psi \in \mathcal{T}(SOD_\psi(UA))$. Furthermore, because $\mathsf{done}(t)$ and because $SOD_\phi(UA)$ and $SOD_\psi(UA)$ synchronize on $done$, $\mathsf{done}(t_\phi)$ and $\mathsf{done}(t_\psi)$. From the induction hypothesis, it follows that $\widetilde{t_\phi} \vdash^{\mathcal{T}}_{UA} \phi$ and $\widetilde{t_\psi} \vdash^{\mathcal{T}}_{UA} \psi$. Moreover, because $SOD_\phi(UA)$ and $SOD_\psi(UA)$ synchronize on $\mathcal{E}_A$ but not on $\mathcal{E}_B$, $\mathsf{si}(\tilde{t}, \widetilde{t_\phi}, \widetilde{t_\psi})$. By $(TR10)$, $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi \odot \psi$.

Finally, consider the term $\phi \otimes \psi$, let $t \in \mathcal{T}(SOD_{\phi\otimes\psi}(UA))$, and $\mathsf{done}(t)$. By $(MA6)$, $SOD_{\phi\otimes\psi}(UA) = ([\![\phi]\!]^{U_\phi}_{UA} \underset{\{done\}\,\cup\,\mathcal{E}_A}{\parallel} [\![\psi]\!]^{U_\psi}_{UA}) \,\Box\, \ldots$. From the denotational semantics of CSP follows that there are two disjoint sets of users U_ϕ and U_ψ such that $t \in \mathcal{T}([\![\phi]\!]^{U_\phi}_{UA} \underset{\{done\}\,\cup\,\mathcal{E}_A}{\parallel} [\![\psi]\!]^{U_\psi}_{UA})$. Analogous to the previous case, there are two traces $\widetilde{t_\phi}$ and $\widetilde{t_\psi}$ such that $\widetilde{t_\phi} \vdash^{\mathcal{T}}_{UA} \phi$, $\widetilde{t_\psi} \vdash^{\mathcal{T}}_{UA} \psi$, and $\mathsf{si}(\tilde{t}, \widetilde{t_\phi}, \widetilde{t_\psi})$. By Proposition 7, users in $\mathsf{users}(\widetilde{t_\phi})$ are in U_ϕ and users in $\mathsf{users}(\widetilde{t_\psi})$ are in U_ψ. Because $U_\phi \cap U_\psi = \emptyset$, and, moreover, $\mathsf{users}(\widetilde{t_\phi}) \cap \mathsf{users}(\widetilde{t_\psi}) = \emptyset$. Therefore, by $(TR11)$, $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi \otimes \psi$.

(2) $LHS \Leftarrow RHS$:

Base cases: Consider a unit term ϕ_{ut} and $\tilde{t}$ be a trace such that $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi_{ut}$. The only rules of Definition 7 that allow for the derivation of ϕ_{ut} are $(TR1)$–$(TR4)$. Of these, only $(TR1)$ does not have a trace in its premises. Therefore, $(TR1)$ is at the leaves of every derivation of $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi_{ut}$ and, thus, $\tilde{t}$ contains a business event b. By iteratively applying the rules $(TR2)$–$(TR4)$, one can add admin events before and after b but no additional business event (otherwise ϕ_{ut} would not be a unit term). It follows that $\tilde{t}$ is of the form $a_1\hat{}\,\langle b\rangle\hat{}\,a_2$, for $a_1, a_2 \in \mathcal{E}_A^*$ and $b \in \mathcal{E}_B$. Let $UA' := \mathsf{upd}(UA, a_1)$. From Proposition 5 it follows that $\langle b\rangle\hat{}\,a_2 \vdash^{\mathcal{T}}_{UA'} \phi_{ut}$ and therefore, by $(TR1)$, $\{\mathsf{user}(b)\} \vdash^{\mathcal{M}}_{UA'} \phi_{ut}$. By $(MA1)$, $SOD_{\phi_{ut}}(UA)$ accepts a_1 and behaves like $SOD_{\phi_{ut}}(UA')$ afterwards. Because $\{\mathsf{user}(b)\} \vdash^{\mathcal{M}}_{UA'} \phi_{ut}$, $SOD_{\phi_{ut}}(UA')$ engages in b and behaves like FIN afterwards. From FIN's definition, it follows that FIN accepts $a_2\hat{}\,\langle done\rangle$ and finally behaves like $STOP$. Hence, $t \in \mathcal{T}(SOD_{\phi_{ut}}(UA))$.

Consider a term of the form ϕ^+_{ut} and let $\tilde{t}$ be a trace such that $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi^+_{ut}$. The only rules of Definition 7 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 $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi^+_{ut}$ contains one application of $(TR5)$ and, thus, $\tilde{t}$ contains at least one business event b. By the rules $(TR2)$–$(TR4)$ and $(TR6)$, it follows that $\tilde{t}$ is of the form $a_1\hat{}\langle b_1\rangle\hat{}\ldots\hat{}a_n\hat{}\langle b_n\rangle\hat{}a_{n+1}$ for $a_i \in \mathcal{E}_A^*$, $a_{n+1} \in \mathcal{E}_A^*$, and $b_i \in \mathcal{E}_B$, for $i \in \{1, \ldots, n\}$. Because there is at least one b in $\tilde{t}$, $n \geq 1$. We reason inductively over n. For $n = 1$, it follows analogous to the unit term case, by $(MA2)$, that $a_1\hat{}\,\langle b_1\rangle\hat{}\,a_2\hat{}\,\langle done\rangle \in \mathcal{T}(SOD_{\phi^+_{ut}}(UA))$. For $n > 1$, assume $a_2\hat{}\langle b_2\rangle\hat{}\ldots\hat{}a_n\hat{}\langle b_n\rangle\hat{}a_{n+1}\hat{}\langle done\rangle \in \mathcal{T}(SOD_{\phi^+_{ut}}(UA'))$, for $UA' := \mathsf{upd}(UA, a)$. Because $a_1\hat{}\,\langle b_1\rangle\hat{}\,\ldots\hat{}\,a_n\hat{}\,\langle b_n\rangle\hat{}\,a_{n+1} \vdash^{\mathcal{T}}_{UA} \phi^+_{ut}$, $\{\mathsf{user}(b_1)\} \vdash^{\mathcal{M}}_{UA'} \phi_{ut}$ by $(TR1)$, $(TR6)$, and Proposition 5. By $(MA2)$, $SOD_{\phi^+_{ut}}(UA)$ accepts a_1 and behaves like $SOD_{\phi^+_{ut}}(UA')$ afterwards. Because $\{\mathsf{user}(b_1)\} \vdash^{\mathcal{M}}_{UA'} \phi_{ut}$, $SOD_{\phi^+_{ut}}(UA')$ engages in b_1 and behaves like $SOD_{\phi^+_{ut}}(UA')$ and FIN afterwards. Therefore, by the induction hypothesis, $a_1\hat{}\,\langle b_1\rangle\hat{}\,a_2\hat{}\,\langle b_2\rangle\hat{}\,\ldots\hat{}\,a_n\hat{}\,\langle b_n\rangle\hat{}\,a_{n+1}\hat{}\,\langle done\rangle \in \mathcal{T}(SOD_{\phi^+_{ut}}(UA))$. Hence, $t \in \mathcal{T}(SOD_{\phi^+_{ut}}(UA))$.

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 $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi \sqcup \psi$. By $(TR7)$ and $(TR8)$, either $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi$ or $\tilde{t} \vdash^{\mathcal{T}}_{UA} \psi$. Consider the first case. By $(MA3)$ and the denotational semantics of CSP, $\mathcal{T}(SOD_{\phi\sqcup\psi}(UA)) = \mathcal{T}(SOD_\phi(UA)) \cup \mathcal{T}(SOD_\psi(UA))$. From the induction hypothesis it follows that $t \in \mathcal{T}(SOD_\phi(UA))$ and therefore, $t \in \mathcal{T}(SOD_{\phi\sqcup\psi}(UA))$. The second case is analogous. Hence, $t \in \mathcal{T}(SOD_{\phi\sqcup\psi}(UA))$.

Consider the term $\phi \sqcap \psi$ and let $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi \sqcap \psi$. By $(TR9)$, $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi$ and $\tilde{t} \vdash^{\mathcal{T}}_{UA} \psi$. By $(MA4)$ and the denotational semantics of CSP, $\mathcal{T}(SOD_{\phi\sqcap\psi}(UA)) = \mathcal{T}(SOD_\phi(UA)) \cap \mathcal{T}(SOD_\psi(UA))$. From the induction hypothesis it follows that $t \in \mathcal{T}(SOD_\phi(UA))$ and $t \in \mathcal{T}(SOD_\psi(UA))$ and therefore, $t \in \mathcal{T}(SOD_{\phi\sqcap\psi}(UA))$.

Consider the term $\phi \odot \psi$ and let $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi \odot \psi$. By $(TR10)$, there exist two traces $\tilde{t_\phi}$ and $\tilde{t_\psi}$ such that $\tilde{t_\phi} \vdash^{\mathcal{T}}_{UA} \phi$, $\tilde{t_\psi} \vdash^{\mathcal{T}}_{UA} \psi$, and $\mathsf{si}(\tilde{t}, \tilde{t_\phi}, \tilde{t_\psi})$. By the induction hypothesis, $t_\phi \in \mathcal{T}(SOD_\phi(UA))$ and $t_\psi \in \mathcal{T}(SOD_\psi(UA))$, and therefore also $\tilde{t_\phi} \in \mathcal{T}(SOD_\phi(UA))$ and $\tilde{t_\psi} \in \mathcal{T}(SOD_\psi(UA))$. From $(MA5)$ and Proposition 8 it follows that $\tilde{t} \in \mathcal{T}(SOD_{\phi\odot\psi}(UA))$. Moreover, by $(MA5)$, because $SOD_\phi(UA)$ and $SOD_\psi(UA)$ both engage in $done$ after having accepted $\tilde{t_\phi}$ and $\tilde{t_\psi}$ respectively, $SOD_{\phi\odot\psi}(UA)$ engages in $done$ too, after having accepted $\tilde{t}$. Hence, $t \in \mathcal{T}(SOD_{\phi\odot\psi}(UA))$.

Finally, consider a term of the form $\phi \otimes \psi$ and let $\tilde{t} \vdash^{\mathcal{T}}_{UA} \phi \otimes \psi$. By $(TR11)$, there exist two traces $\tilde{t_\phi}$ and $\tilde{t_\psi}$ such that $\tilde{t_\phi} \vdash^{\mathcal{T}}_{UA} \phi$, $\tilde{t_\psi} \vdash^{\mathcal{T}}_{UA} \psi$, $\mathsf{si}(\tilde{t}, \tilde{t_\phi}, \tilde{t_\psi})$, and $\mathsf{users}(\tilde{t_\phi}) \cap \mathsf{users}(\tilde{t_\psi}) = \emptyset$. Because $\mathsf{users}(\tilde{t_\phi}) \cap \mathsf{users}(\tilde{t_\psi}) = \emptyset$, there exist two sets of users $U_\phi \subseteq \mathcal{U}$ and $U_\psi \subseteq \mathcal{U}$ such that $U_\phi \cup U_\psi = \mathcal{U}$, $U_\phi \cap U_\psi = \emptyset$, $\mathsf{users}(\tilde{t_\phi}) \subseteq U_\phi$, and $\mathsf{users}(\tilde{t_\psi}) \subseteq U_\psi$. By the induction hypothesis, $t_\phi \in \mathcal{T}(SOD_\phi(UA))$ and $t_\psi \in \mathcal{T}(SOD_\psi(UA))$, and therefore also $\tilde{t_\phi} \in \mathcal{T}(SOD_\phi(UA))$ and $\tilde{t_\psi} \in \mathcal{T}(SOD_\psi(UA))$. From Proposition 7, $[\![\phi]\!]^{U_\phi}_{UA}$ therefore accepts $\tilde{t_\phi}$ and $[\![\psi]\!]^{U_\psi}_{UA}$ accepts $\tilde{t_\psi}$. By $(MA6)$ and the denotational semantics of CSP, $SOD_{\phi\otimes\psi}(UA)$ also behaves like $[\![\phi]\!]^{U_\phi}_{UA} \underset{\{done\} \cup \mathcal{E}_A}{\parallel} [\![\psi]\!]^{U_\psi}_{UA}$. Analogous to the previous case, it follows that $\tilde{t} \in \mathcal{T}(SOD_{\phi\otimes\psi}(UA))$ and $t \in \mathcal{T}(SOD_{\phi\otimes\psi}(UA))$. $\square$