Decentralized Distributed Data Usage Control

Florian Kelbert and Alexander Pretschner

Technische Universität München, Germany
{kelbert,pretschn}@cs.tum.edu

Abstract. Data usage control provides mechanisms for data owners to remain in control over how their data is used after it has been shared. Many data usage policies can only be enforced on a global scale, as they refer to data usage events happening within multiple distributed systems: 'not more than three employees may ever read this document', or 'no copy of this document may be modified after it has been archived'. While such global policies can be enforced by a centralized enforcement infrastructure that observes all data usage events in all relevant systems, such a strategy involves heavy communication. We show how the overall coordination overhead can be reduced by deploying a decentralized enforcement infrastructure. Our contributions are: (i) a formal distributed data usage control system model; (ii) formal methods for identifying all systems relevant for evaluating a given policy; (iii) identification of situations in which no coordination between systems is necessary without compromising policy enforcement; (iv) proofs of correctness of (ii, iii).

1 Introduction

Consider a company's financial department in which the CFO and her employees collaborate via email. Business reports, contracts, and transactional information are exchanged via email and edited by multiple employees. Employees also use email to collaborate on documents that are *not* considered sensitive. However, due to the other documents' sensitivity and their decentral sharing, the company deploys usage control [1, 2] technologies on the employees' devices with the goal to enforce policies such as 'document D1 must not be edited' (**P1**), or 'there may be at most one ongoing edit process for document D2 at each point in time and no editing is allowed after the CFO archived the final version' (**P2**). What is usually meant by such policies is that not only one particular file pertaining to a document (e.g. D1) must be protected, but all copies and derivations of it: If the document is copied to another file, loaded into a Java application or sent via the network, all of these *representations* of D1 (file, java object, network packet) must be protected [3, 4]. We refer to policies P1 and P2 throughout this paper.

Once representations of documents D1 and D2 have been emailed to multiple employees and exist in different systems, each of those systems is in charge of enforcing the corresponding policies P1 and P2 [5]. Intuitively, policy P1 can be enforced locally by denying all edit requests for each local copy of D1. Policy P2, in contrast, refers to events happening within multiple systems and introduces dependencies between them. Thus, enforcement of policy P2 necessitates

In: Proc. 13th International Conference on Cryptology and Network Security (CANS), Springer LNCS 8813, pages 353-369, October 2014, Heraklion, Crete, Greece. The final publication is available at http://link.springer.com.

coordination between all systems potentially capable of editing and archiving representations of document D2.

Within one single system, usage control enforcement infrastructures are commonly implemented in correspondence with the XACML standard architecture [6]: System-layer specific policy enforcement points (PEPs, e.g. for MS Windows [7], OpenBSD/Linux [4], Mozilla Thunderbird [5]) intercept data usage events (e.g., *save*, *edit*, *send*, *archive*) and signal them to the local policy decision point (PDP) [3, 8, 9]. The PDP evaluates each event against the deployed data usage policies and signals its decision back to the corresponding PEP, which will then enforce it. For taking this decision, the PDP might need additional information about the system's state, such as subject and object attributes, or the data's current representations—also called the *data flow state*. Such information is collected by the policy information point (PIP), which is queried by the PDP.

As indicated earlier, policy P1 can be enforced locally: The system's PEPs signal all *edit* events to their local PDP, which in turn queries the local PIP to learn whether a particular *edit* event takes place on a representation of document D1. If so, the PDP's decision is to disallow the event.

Policy P2, however, can not be enforced by local PDPs/PIPs only: Assume three representations of document D2 on three different systems. Whenever an employee requests to edit a representation of D2, this *edit* event must only be allowed if no other employee is currently editing a representation of D2. Similarly, after the event *archive* has been performed by the CFO on a representation of D2, all future editing requests must be disallowed. Because the representations of D2 are decentrally shared and because *edit* and *archive* events can happen on any of those systems, purely local PDPs are generally unable to decide about this policy. Additional information is needed, e.g. 'how many employees are currently editing D2?', and 'has D2 been archived in the past?'.

Intuitively, policy P2, or, more generally, any policy referring to distributed data and data usage events, can be enforced by a centralized enforcement infrastructure, i.e. a single global PDP/PIP (Fig. 1). However, such a centralized infrastructure imposes the **problem** of heavy communication overhead, as all data usage events from all relevant systems must be signalled to the central PDP/PIP. Such an approach is particularly inappropriate if employees also work on unprotected documents. Moreover, if employees work while travelling, each event must be sent to the central PDP/PIP via a mobile internet connection. This is likely to make the work cumbersome due to large communication delays

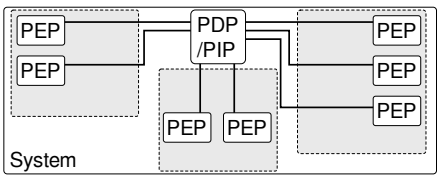

Fig. 1: Naive centralized enforcement.

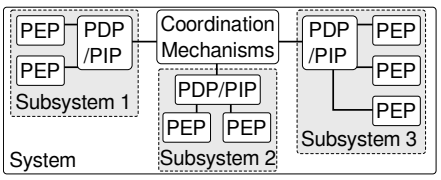

Fig. 2: Decentralized variant (§3-4).

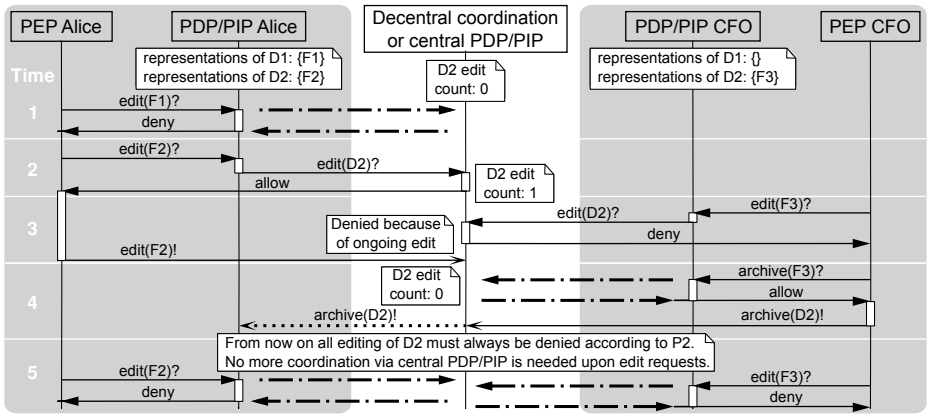

Fig. 3: Enforcement of policies P1 and P2 given the event trace of Table 1.

and the fact that PEPs usually block system execution upon each intercepted event until the PDP's decision is available [7, 10, 11].

Our **goal** is to improve on this situation by reducing the amount of communication needed whenever global policies of the kind of P2 ought to be enforced.

Our **solution** is an enforcement infrastructure that is inherently distributed. It deploys a local PDP/PIP on each site, e.g., a physical device or virtual machine (Fig. 2). PEPs signal events to these local PDPs/PIPs using fast inter process communication. While the local components can independently (i.e. without coordination with other PDPs/PIPs) decide P1, some coordination with other PDPs/PIPs is still needed for the enforcement of P2. Fig. 3 depicts the advantages of our proposed solution when enforcing P1 and P2, given the trace of events in Table 1; and assuming F1 to be a representation of document D1, and F2 and F3 representations of document D2. Dash-dotted arrows (·–·►) indicate expensive cross-system communication that is needed in a centralized enforcement infrastructure but not in our approach. Dotted arrows (····►) indicate communication that is introduced by our solution. Question marks (?) indicate decision requests from PEPs to PDPs if intended events are intercepted. Exclamation marks (!) indicate to a PDP that an event has actually happened.

Time	Alice	CFO
1	edit(F1)	
2	edit(F2)	
3		edit(F3)
4		archive(F3)
5	edit(F2)	edit(F3)

Table 1: Event trace.

This work aims at minimizing the communication overhead for such a decentralized enforcement infrastructure by providing the following **contributions**:

1. We provide a formal distributed data usage control system model (§3).
2. We provide formal methods to identify all systems potentially relevant for evaluating a given data usage policy at any point in time (§4.1).
3. We provide insights in which situations communication between PDPs and PIPs can be omitted (§4.2) without compromising policy enforcement.
4. We show the correctness of 2. and 3. in Appendices A and B.

2 A Formal Usage Control Model

We recap a formal model for specifying and enforcing usage control policies [3,4,12], where policies define constraints over system states and traces of events. Before defining the syntax and semantics of policies (§2.3), we describe its foundations, i.e. system events (§2.1) and system states (§2.2).

2.1 System events and system runs

Events $\mathcal{E}$ are defined by a name (set *EName*) and a set of parameters, which are, in turn, defined by a name (set *PName*) and a value (set *PValue*): $\mathcal{E} \subseteq EName \times \mathbb{P}(PName \times PValue)$. For an event $e \in \mathcal{E}$, let $e.n$ denote the event's name and $e.p$ the set of its parameters. Furthermore, let $obj \in PName$ denote an event's primary object whose value can be accessed using notation $e.obj$.

Event refinement. When specifying policies, it is not useful to define all possible event parameters. Instead, one would like to specify only relevant parameters, quantifying over all unmentioned ones. In our example, it is irrelevant which particular user edits document D2, but not the fact *that* D2 is edited. Hence, $refines \subseteq \mathcal{E} \times \mathcal{E}$ defines a refinement relation on events: event e_1 refines event e_2 iff they have the same event name and the parameters of e_1 are a superset of the parameters of e_2: $\forall e_1, e_2 \in \mathcal{E} : e_1 \; refines \; e_2 \Leftrightarrow e_1.n = e_2.n \wedge e_1.p \supseteq e_2.p$.

System events $\mathcal{S}$, i.e. events intercepted by PEPs at runtime in a real system, are always maximally refined, i.e. all parameters are determined. Hence, $\mathcal{S} = \mathcal{E} \setminus \{e \in \mathcal{E} \mid \exists e' \in \mathcal{E} : \; e' \neq e \wedge e' \; refines \; e\}$.

System runs are modeled as traces, mapping each abstract moment in time to the set of system events happening at that time: $Trace : \mathbb{N} \to \mathbb{P}(\mathcal{S})$.

2.2 System states

Since the data to be protected may exist in multiple representations (e.g., document D1 might be represented as a file, a java object, or a network packet), a system's state is defined in terms of the distribution of data within that system [3, 4]. Hence, we also refer to it as the system's data flow state. We call the data's representations *containers* and $\mathcal{C}$ the global set of containers. The global set of *data* to be protected by usage control policies is denoted $\mathcal{D}$, and $\mathcal{C} \cap \mathcal{D} = \emptyset$.

As motivated earlier, data usage policies are specified in terms of data, implying that the imposed restrictions also apply to all copies and derivations. Thus, only elements $v \in \mathcal{D}$ are possible values for an event's *obj* parameter when specifying policies. In contrast, system events $e \in \mathcal{S}$ operate on containers, which is why elements $v \in \mathcal{C}$ are the only possible values for a system event's *obj* parameter. Taken together, elements $v \in \mathcal{C} \cup \mathcal{D}$ are possible values for an event's *obj* parameter, $(\mathcal{C} \cup \mathcal{D}) \subseteq PValue$. For the remainder of this paper we constrain the set of possible values for event parameter *obj* to $\mathcal{C} \cup \mathcal{D} \cup \{\epsilon\}$, reflecting the fact that an event operates on a container, a data, or neither of the two, respectively.

System states $\Sigma = \mathcal{C} \to \mathbb{P}(\mathcal{D})$ map containers to data potentially stored in them. In our example, $\sigma \in \Sigma$ records which files, emails, and editing processes are

representations of documents D1 and D2. Transition relation $\mathcal{R}$ describes how the execution of system events $\mathcal{S}$ changes the system's state: $\mathcal{R} \subseteq \Sigma \times \mathbb{P}(\mathcal{S}) \to \Sigma$.

Given a system trace $t \in \mathit{Trace}$ and a point in time $i \in \mathbb{N}$, the system's state is computed as $\sigma_t^i = \mathcal{R}(\sigma_t^{i-1}, t(i-1))$; $\sigma_t^0 = \emptyset$ represents the trace's initial state.

Instantiations of this generic data flow model, in particular semantics of $\mathcal{R}$, have been described for various system layers such as MS Windows [7], OpenBSD/Linux [4], X11 [13], Thunderbird [5], as well as distributed systems [10].

Event refinement in the presence of states. Extending the earlier event refinement, $\mathit{refines}_\Sigma \subseteq (\mathcal{S} \times \Sigma) \times \mathcal{E}$ describes the refinement between two events in the presence of a given system state. The reason is that policies (§2.3) are specified in terms of data ($\exists e_2 \in \mathcal{E}, d \in \mathcal{D} : e_2.\mathit{obj} = d$), while system events operate on containers ($\exists e_1 \in \mathcal{S}, c \in \mathcal{C} : e_1.\mathit{obj} = c$). Hence, we need to evaluate the system's current state $\sigma \in \Sigma$ in order to decide whether an event refines another. We say that (e_1, σ) refines e_2 iff $d \in \sigma(c)$ and if the parameters of e_1 are a superset of the parameters of e_2 when ignoring the *obj* parameter:

$$\forall e_1 \in \mathcal{S}, e_2 \in \mathcal{E}, \sigma \in \Sigma : (e_1, \sigma)\ \mathit{refines}_\Sigma\ e_2 \iff \exists c \in \mathcal{C}, d \in \mathcal{D} : e_1.n = e_2.n \\ \wedge\ e_1.\mathit{obj} = c \wedge e_2.\mathit{obj} = d \wedge d \in \sigma(c) \wedge e_1.p \backslash \{(\mathit{obj}, c)\} \supseteq e_2.p \backslash \{(\mathit{obj}, d)\}$$

For instance, consider a state $\sigma \in \Sigma$ in which file F1 is a representation of document D1. Then $((\mathit{edit}, \{(\mathit{obj}, \mathit{F1}), \ldots\}), \sigma)\ \mathit{refines}_\Sigma\ (\mathit{edit}, \{(\mathit{obj}, \mathit{D1})\})$.

2.3 Data usage policies

Building upon previous work [3, 14–16], we assume technical policies to be expressed as event-condition-action (ECA) rules: whenever a triggering **E**vent is detected and if it makes the **C**ondition *true*, then (additional) **A**ctions might be performed. Because the policies' conditions are formulated in terms of past temporal logics, this work focuses on the evaluation of such formulas. Based on the above foundations and [3], the syntax of ECA **C**onditions (Φ) is defined as:

$$\begin{aligned} \Psi &= \underline{\mathit{false}} \mid \mathcal{E} \\ \Phi^\Sigma &= \underline{\mathit{isNotIn}}(\mathcal{D}, \mathbb{P}(\mathcal{C})) \mid \underline{\mathit{isCombined}}(\mathcal{D}, \mathcal{D}, \mathbb{P}(\mathcal{C})) \mid \underline{\mathit{isMaxIn}}(\mathcal{D}, \mathbb{N}, \mathbb{P}(\mathcal{C})) \\ \Phi &= (\Phi) \mid \Psi \mid \Phi^\Sigma \mid \Phi\ \underline{\mathit{and}}\ \Phi \mid \underline{\mathit{not}}(\Phi) \mid \Phi\ \underline{\mathit{since}}\ \Phi \mid \Phi\ \underline{\mathit{before}}\ \mathbb{N} \mid \underline{\mathit{repmin}}(\mathbb{N}, \mathbb{N}, \mathcal{E}) \end{aligned}$$

Ψ is self-explanatory. Φ^Σdefines state-based operators for constraints on the system's data flow state: $\underline{\mathit{isNotIn}}(d, C)$ is true iff data d is not in any of the containers C; $\underline{\mathit{isCombined}}(d_1, d_2, C)$ is true iff there is at least one container in C that contains both data d_1 and d_2; $\underline{\mathit{isMaxIn}}(d, m, C)$ is true iff data d is contained in at most m containers in C. For Φ, the semantics of $\underline{\mathit{and}}$ and $\underline{\mathit{not}}$ are intuitive; $\alpha\ \underline{\mathit{since}}\ \beta$ is true iff β was true some time earlier and α was true ever since, or if α was always true; $\alpha\ \underline{\mathit{before}}\ j$ is true iff α was true exactly j timesteps ago; $\underline{\mathit{repmin}}(j, m, e)$ is true iff event e happened at least m times in the last j timesteps. Further shortcuts include those for $\underline{\mathit{true}}$ and $\underline{\mathit{or}}$; plus $\underline{\mathit{repmax}}(j, m, e) \equiv \underline{\mathit{not}}(\underline{\mathit{repmin}}(j, m+1, e))$; $\underline{\mathit{replim}}(j, m, n, e) \equiv \underline{\mathit{repmin}}(j, m, e)\ \underline{\mathit{and}}\ \underline{\mathit{repmax}}(j, n, e)$. The formal semantics of policies Φ are:

$$
\begin{aligned}
&\forall t \in \mathit{Trace}, i \in \mathbb{N}, \varphi \in \Phi \bullet (t,i) \models \varphi \iff (\varphi \neq \underline{\mathit{false}}) \wedge \\
&\quad \exists e \in \mathcal{E}, e' \in t(i) \bullet (\varphi = e \wedge (e', \sigma_t^i)\ \mathit{refines}_\Sigma\ e) \\
&\vee \exists d \in \mathcal{D}, C \subseteq \mathcal{C} \bullet (\varphi = \underline{\mathit{isNotIn}}(d, C) \wedge \forall c \in C \bullet d \notin \sigma_t^i(c)) \\
&\vee \exists d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C} \bullet (\varphi = \underline{\mathit{isCombined}}(d_1, d_2, C) \wedge \exists c \in C \bullet \{d_1, d_2\} \subseteq \sigma_t^i(c)) \\
&\vee \exists d \in \mathcal{D}, m \in \mathbb{N}, C \subseteq C \bullet (\varphi = \underline{\mathit{isMaxIn}}(d, m, C) \wedge |\{c \in C | d \in \sigma_t^i(c)\}| \leq m) \\
&\vee \exists \alpha, \beta \in \Phi \bullet ((\varphi = \underline{\mathit{not}}(\alpha) \wedge \neg((t,i) \models \alpha)) \\
&\qquad \vee (\varphi = \alpha\ \underline{\mathit{and}}\ \beta \wedge (t,i) \models \alpha \wedge (t,i) \models \beta) \\
&\qquad \vee (\varphi = \alpha\ \underline{\mathit{or}}\ \beta \wedge (t,i) \models \alpha \vee (t,i) \models \beta) \\
&\qquad \vee (\varphi = \alpha\ \underline{\mathit{since}}\ \beta \wedge \exists j \in [0,i] \bullet ((t,j) \models \beta \wedge \forall k \in (j,i] \bullet (t,k) \models \alpha) \\
&\qquad\quad \vee \forall k \in [0,i] \bullet (t,k) \models \alpha)) \\
&\vee \exists \alpha \in \Phi, j \in \mathbb{N} \bullet (\varphi = \alpha\ \underline{\mathit{before}}\ j \wedge (t, i-j) \models \alpha) \\
&\vee \exists j, m \in \mathbb{N}, e \in \mathcal{E} \bullet (\varphi = \underline{\mathit{repmin}}(j, m, e) \\
&\qquad \wedge m \leq \textstyle\sum_{k=0}^{j-1} |\{e' \in t(i-k) | (e', \sigma_t^{i-k})\ \mathit{refines}_\Sigma\ e\}|)
\end{aligned}
$$

Policy enforcement is usually performed as sketched in §1 [3, 8, 9, 11, 13, 17]. With $C_{EditProc}$ denoting the set of all processes with the capability to edit documents [4], one way of expressing our example policies as ECA rules is:

P1	**E**vent: $(edit, \{(obj, D1)\})$
	Condition: $\varphi = \underline{true}$
	Action: *inhibit*
P2	**E**vent: $(edit, \{(obj, D2)\})$
	Condition: $\varphi = \underline{not}(\underline{isMaxIn}(D2, 0, C_{EditProc})\ \underline{and}$ $\underline{not}((archive, \{(obj, D2), (user, CFO)\}))\ \underline{since}\ \underline{false})$
	Action: *inhibit*

3 A Distributed System Model

The model in §2 suggests a monolithic view on policy enforcement: at runtime there is one single global trace and system state at any point in time. Technically, one central PDP/PIP globally observes the entire system. As this is likely impractical in real-world distributed scenarios, we propose an extended model in which multiple PDPs and PIPs observe different parts of the global system.

3.1 Individual and Concurrently Executing Subsystems

Adapting to the terms used in §2, we refer to the distributed system as a whole as the *system*, which is, in turn, composed of a set of *subsystems*. In our terminology, each subsystem is a set of possibly distributed system layers whose PEPs share one single PDP. More technically, a subsystem may be an operating system instance, a physical or virtual machine, a set of applications, or a set of physical or virtual machines. A subsystem thus contains exactly one PDP/PIP and at least one PEP (Fig. 2). We assume each subsystem to be assigned a unique identifier $s \in \mathbb{N}$, which could map to a MAC address or UUID in practice.

For each subsystem $s \in \mathbb{N}$ we define $\mathcal{C}_s \subseteq \mathcal{C}$ as its unique set of containers, $\mathcal{S}_s \subseteq \mathcal{S}$ as its unique set of system events[1], $\Sigma_s : \mathcal{C}_s \to \mathbb{P}(\mathcal{D})$ as its set of states, and $\mathit{Trace}_s : \mathbb{N} \to \mathbb{P}(\mathcal{S}_s)$ as its set of all possible runs, $\mathit{Trace}_s \subseteq \mathit{Trace}$. Because we will 'overlay' traces of different subsystems shortly, we require each system event $e \in \mathcal{S}_s$ to carry parameter *sub* with value s:

$$\forall s, s' \in \mathbb{N}, e \in \mathcal{S}_s : (\mathit{sub}, s) \in e.p \wedge s \neq s' \implies \mathcal{S}_s \cap \mathcal{S}_{s'} = \emptyset \wedge \mathcal{C}_s \cap \mathcal{C}_{s'} = \emptyset.^2$$

Containers and system events for a set of subsystems $M \subseteq \mathbb{N}$ are defined by $\forall M \subseteq \mathbb{N} : \mathcal{S}_M = \bigcup_{s \in M} \mathcal{S}_s \ \wedge \ \mathcal{C}_M = \bigcup_{s \in M} \mathcal{C}_s$.

Concurrent system runs. In practice, subsystems run in parallel and produce independent system traces: Each subsystem's PDP observes a trace of system events, $t_s \in \mathit{Trace}_s$. Assuming sufficiently synchronized system clocks [18], it is the union of these local observations that one single global PDP would observe. When reasoning about this global behavior, the behavior of individual subsystems or of sets of subsystems, we will use notations t_s^τ and t_M^τ, in order to refer to the trace of a particular subsystem $s \in \mathbb{N}$ or set of subsystems $M \subseteq \mathbb{N}$ given a tuple τ of concurrently executing traces. The intuition is that t_M^τ overlays the concurrently executing traces of all subsystems $m \in M$ by unifying all events happening in all subsystems $m \in M$ at each point in time $i \in N$. Let $\prod$ denote the Cartesian product. Then $\tau \in \prod_{n \in \mathbb{N}} \mathit{Trace}_n$ is a tuple of traces of all subsystems, and the m-th element of τ, i.e. $\tau.m$, is a trace of subsystem $m \in \mathbb{N}$. The overlay of a set of traces of subsystems $M \subseteq \mathbb{N}$, t_M^τ, is $\forall \tau \in \prod_{n \in \mathbb{N}} \mathit{Trace}_n, i, s \in \mathbb{N}, M \subseteq \mathbb{N} : t_s^\tau = \tau.s \ \wedge \ t_M^\tau(i) = \bigcup_{m \in M} (\tau.m)(i)$.

In the following, we will mostly talk about sets of subsystems $M \subseteq \mathbb{N}$. The same considerations apply to the single subsystems $s \in \mathbb{N}$ by letting $M = \{s\}$.

3.2 Policy Projections

When considering a set of subsystems $M \subseteq \mathbb{N}$, it is generally not possible to conclusively evaluate a given policy $\varphi \in \Phi$, since evaluation of φ might depend on information unavailable within M. In our example (Fig. 3, Table 1), Alice's PDP cannot decide about event *edit(F2)* at time 2, since another employee might already be editing D2. However, Alice's PDP *can* evaluate a projection of formula φ of policy P2 by hiding parts that refer to other subsystems: Letting C_A denote all containers within Alice's subsystem, subformula $\underline{\mathit{isMaxIn}}(D2, 0, C_{\mathit{EditProc}} \cap C_A)$ can be evaluated by Alice's PDP. We will make use of these policy projections in §4.2 with the goal to omit unnecessary coordination between subsystems. In general, the projection $\varphi_M \in \Phi$ of $\varphi \in \Phi$ for subsystems M is defined as:

[1] Events belonging to multiple subsystems (such as *transfer(data,from,to)*) are attributed to the initiating one.

[2] Parameter *sub* makes us redefine relations *refines* and $\mathit{refines}_\Sigma$, as this parameter must not influence event refinement: $\forall e_1, e_2 \in \mathcal{E}, s, s' \in \mathbb{N} : e_1 \ \mathit{refines} \ e_2 \Leftrightarrow e_1.n = e_2.n \wedge e_1.p \backslash \{(\mathit{sub}, s)\} \supseteq e_2.p \backslash \{(\mathit{sub}, s')\}$. We refrain from formally redefining $\mathit{refines}_\Sigma$.

$$
\begin{aligned}
&\forall \varphi \in \Phi, M \subseteq \mathbb{N}, \exists \varphi_M \in \Phi : (\varphi = \underline{false} \wedge \varphi_M = \underline{false}) \\
&\quad \vee (\exists\ \alpha, \beta \in \Phi\ \bullet (\varphi = \alpha\ \underline{and}\ \beta \wedge \varphi_M = \alpha_M\ \underline{and}\ \beta_M) \\
&\qquad \vee (\varphi = \alpha\ \underline{or}\ \beta \wedge \varphi_M = \alpha_M\ \underline{or}\ \beta_M) \\
&\qquad \vee (\varphi = \alpha\ \underline{since}\ \beta \wedge \varphi_M = \alpha_M\ \underline{since}\ \beta_M) \\
&\qquad \vee (\varphi = \underline{not}(\alpha) \wedge \varphi_M = \underline{not}(\alpha_M)) \\
&\quad \vee (\exists\ e \in \mathcal{E} \bullet (\varphi = e \wedge \varphi_M = e)) \\
&\quad \vee (\exists\ d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C} \bullet \varphi = \underline{isCombined}(d_1, d_2, C) \\
&\qquad \wedge \varphi_M = \underline{isCombined}(d_1, d_2, C \cap \mathcal{C}_M)) \\
&\quad \vee (\exists\ d \in \mathcal{D}, C \subseteq \mathcal{C} \bullet \varphi = \underline{isNotIn}(d, C) \wedge\ \varphi_M = \underline{isNotIn}(d, C \cap \mathcal{C}_M)) \\
&\quad \vee (\exists\ d \in \mathcal{D}, m \in \mathbb{N}, C \subseteq \mathcal{C} \bullet \varphi = \underline{isMaxIn}(d, m, C) \\
&\qquad \wedge\ \varphi_M = \underline{isMaxIn}(d, m, C \cap \mathcal{C}_M)) \\
&\quad \vee (\exists\ \alpha \in \Phi, j \in \mathbb{N} \bullet \varphi = \alpha\ \underline{before}\ j \wedge\ \varphi_M = \alpha_M\ \underline{before}\ j) \\
&\quad \vee (\exists\ j, m \in \mathbb{N}, e \in \mathcal{E} \bullet \varphi = \underline{repmin}(j, m, e) \wedge\ \varphi_M = \underline{repmin}(j, m, e))
\end{aligned}
$$

4 Coordinating Subsystems

Deploying one PDP/PIP per subsystem necessitates their coordination for the enforcement of certain policies: a PDP's decision might depend on past decisions and observations of other PDPs and PIPs, because policies might refer to events or system states of multiple subsystems. For enforcing policy P2, all subsystems (1) capable of editing or archiving documents *and* (2) having a representation of D2 stored locally must generally coordinate their decisions and data flow states if a representation of D2 is about to be edited. While naively each PDP/PIP could disclose all of its knowledge to all other PDPs/PIPs or to one central PDP/PIP, we aim at minimizing this coordination overhead. For this, we approximate the set of subsystems relevant for evaluating a formula $\varphi \in \Phi$ (§4.1), and analyze in which cases coordination between PDPs/PIPs can safely be omitted (§4.2).

4.1 Identifying Subsystems Relevant for Evaluating Formulas

Our goal is to approximate the subsystems relevant for evaluating $\varphi \in \Phi$ at time $i \in \mathbb{N}$, given the tuple of concurrently executing traces $\tau \in \prod_{n \in \mathbb{N}} \mathit{Trace}_n$, by function $\mathit{sys}(\varphi, i, \tau)$. In particular, if $|\mathit{sys}(\varphi, i, \tau)| \leq 1$, then no coordination is needed for evaluation of φ. We start by defining three auxiliary functions:

(1) $\mathit{awareC} : \mathbb{P}(\mathcal{C}) \to \mathbb{P}(\mathbb{N})$ returns for a given set of containers the set of subsystems that are aware of at least one of the given containers:

$$\forall C \subseteq \mathcal{C} : \mathit{awareC}(C) = \{s \in \mathbb{N} \mid \mathcal{C}_s \cap C \neq \emptyset\}$$

(2) $\mathit{awareD} : \mathbb{P}(\mathcal{D}) \times \mathbb{N} \times \prod \mathit{Trace} \to \mathbb{P}(\mathbb{N})$ returns for a given set of data items, a point in time, and a tuple of executing traces the set of subsystems in which there exists a container that contains at least one of those data items:

$$
\begin{aligned}
&\forall D \subseteq \mathcal{D}, i \in \mathbb{N}, \tau \in \textstyle\prod_{n \in \mathbb{N}} \mathit{Trace}_n : \\
&\mathit{awareD}(D, i, \tau) = \{s \in \mathbb{N} \mid \exists c \in \mathcal{C}_s : D \cap \sigma^i_{\tau.s}(c) \neq \emptyset\}
\end{aligned}
$$

(3) $mayHappen : \mathcal{E} \times \mathbb{N} \times \prod Trace \to \mathbb{P}(\mathbb{N})$ returns for an event $e \in \mathcal{E}$, a point in time, and a tuple of executing traces the set of subsystems in which an event refining e might happen. This set contains all subsystems that are able to perform events with name $e.n$ and that are 'aware of' the data addressed by e:

$$\forall e \in \mathcal{E}, i \in \mathbb{N}, \tau \in \textstyle\prod_{n\in\mathbb{N}} Trace_n : mayHappen(e,i,\tau) = \\ \{s \in \mathbb{N} \mid \exists e' \in \mathcal{S}_s : e.n = e'.n \wedge \exists d \in \mathcal{D} \bullet e.obj = d \wedge s \in awareD(\{d\},i,\tau)\}$$

$sys(\varphi,i,\tau)$ then returns all subsystems potentially relevant for evaluating φ:

$$\begin{aligned}
&\forall \varphi \in \Phi \cup \Phi^{\Sigma} \cup \Psi, i \in \mathbb{N}, \tau \in \textstyle\prod_{n\in\mathbb{N}} Trace_n : \\
&sys(\varphi,i,\tau) = \{s \in Y \mid (\varphi = \underline{false} \wedge Y = \{\}) \\
&\quad \vee (\exists\, \alpha,\beta \in \Phi \cup \Phi^{\Sigma} \cup \Psi \bullet ((\varphi = \alpha\ \underline{and}\ \beta \vee \varphi = \alpha\ \underline{or}\ \beta) \\
&\qquad \wedge Y = sys(\alpha,i,\tau) \cup sys(\beta,i,\tau)) \\
&\qquad \vee (\varphi = \alpha\ \underline{since}\ \beta \wedge Y = \textstyle\bigcup_{j=0}^{i}(sys(\alpha,j,\tau) \cup sys(\beta,j,\tau)))) \\
&\quad \vee (\exists\, \alpha \in \Phi \cup \Phi^{\Sigma} \cup \Psi \bullet \varphi = \underline{not}(\alpha) \wedge Y = sys(\alpha,i,\tau)) \\
&\quad \vee (\exists\, e \in \mathcal{E} \bullet \varphi = e \wedge Y = mayHappen(e,i,\tau)) \\
&\quad \vee (\exists\, d_1,d_2 \in \mathcal{D}, C \subseteq \mathcal{C} \bullet \varphi = \underline{isCombined}(d_1,d_2,C) \\
&\qquad \wedge Y = awareD(\{d_1\},i,\tau) \cap awareD(\{d_2\},i,\tau) \cap awareC(C)) \\
&\quad \vee (\exists\, d \in \mathcal{D}, m \in \mathbb{N}, C \subseteq \mathcal{C} \bullet (\varphi = \underline{isNotIn}(d,C) \vee \varphi = \underline{isMaxIn}(d,m,C)) \\
&\qquad \wedge Y = awareD(\{d\},i,\tau) \cap awareC(C)) \\
&\quad \vee (\exists\, j,m \in \mathbb{N}, e \in \mathcal{E} \bullet \varphi = \underline{repmin}(j,m,e) \wedge Y = \textstyle\bigcup_{k=0}^{j-1} mayHappen(e,i-k,\tau)) \\
&\quad \vee (\exists\, \alpha \in \Phi \cup \Phi^{\Sigma} \cup \Psi, j \in \mathbb{N} \bullet \varphi = \alpha\ \underline{before}\ j \wedge Y = sys(\alpha,i-j,\tau))\}
\end{aligned}$$

We claim that subsystems $sys(\varphi,i,\tau)$ are sufficient to evaluate φ at time i, given executing traces τ. Subsystems $\mathbb{N} \backslash sys(\varphi,i,\tau)$ do not influence evaluation of φ, and no coordination is needed if $|sys(\varphi,i,\tau)| \leq 1$. We provide proofs of correctness in Appendix A. We will refer to $t^{\tau}_{sys(\varphi,i,\tau)}$ as $t^{\tau}_{\mathbb{N}}$, indicating that the investigated trace is equivalent to what a single global PDP would have observed.

Considering example policy P1 ($\varphi = \underline{true}$), $sys(\varphi,i,\tau)$ returns the empty set, matching the intuition that P1 can always be evaluated locally. Considering policy P2, $sys(\varphi,i,\tau)$ returns both Alice's and the CFO's subsystem, since representations of document D2 exist in both subsystems and both subsystems exhibit editing capabilities. Hence, both subsystems might influence policy evaluation.

4.2 Omitting Unnecessary Coordination

While in general coordination between subsystems is needed if an ECA mechanism's triggering event is observed and $|sys(\varphi,i,\tau)| > 1$, there are situations in which no coordination is required. We have seen that $sys(\varphi,i,\tau)$ returns both Alice's and the CFO's subsystems for policy P2. However, at timestep 5 no coordination takes place (cf. Fig. 3). This is because the CFO archived a representation of D2, in which case all further edit requests must be denied. Once Alice's PDP learns that this archiving event has happened, all further editing request can immediately be denied by Alice's PDP without any further coordination.

Given $\tau \in \prod_{n\in\mathbb{N}} Trace_n$ and a policy $\varphi \in \Phi$, there are special situations in which we can deduce a formula $\varphi' \in \Phi$ such that (i) trace t^{τ}_{M} satisfies φ' at

time $i \in N$, and (ii) this local satisfaction of φ' implies global satisfaction of φ, formally: $(t^\tau_M, i) \models \varphi' \implies (t^\tau_{\mathbb{N}}, i) \models \varphi$. For example, a part of the condition of policy P2 is $\underline{not}(\underline{isMaxIn}(D2, 0, C_{EditProc}))$ when converting P2's condition into disjunctive normal form (DNF). If Alice is already editing a representation of D2, any further concurrent edit requests can be denied without coordination.

We formalize this intuition by predicate $S \subseteq \prod \mathit{Trace}_n \times \mathbb{P}(\mathbb{N}) \times \mathbb{N} \times \Phi$ that holds *true* iff for the tuple of executing traces $\tau \in \prod_{n\in\mathbb{N}} \mathit{Trace}$ and a set of subsystems $M \subseteq \mathbb{N}$, trace t^τ_M satisfies $\varphi_M \in \Phi$ at time $i \in \mathbb{N}$ $((t^\tau_M, i) \models \varphi_M)$ and if this implies global satisfaction of formula $\varphi \in \Phi$ at the same point in time $((t^\tau_{\mathbb{N}}, i) \models \varphi)$. Similarly, the same argument holds for the violation of formula φ, which can intuitively be expressed by negating formula φ:

$$\begin{aligned}
&\forall \tau \in \textstyle\prod_{n\in\mathbb{N}} \mathit{Trace}_n, M \subseteq \mathbb{N}, i \in \mathbb{N}, \varphi \in \Phi : \\
&\quad (t^\tau_M, i) \models \varphi_M \wedge S(\tau, M, i, \varphi) \implies (t^\tau_{\mathbb{N}}, i) \models \varphi \\
&\wedge\ (t^\tau_M, i) \not\models \varphi_M \wedge S(\tau, M, i, \neg\varphi) \implies (t^\tau_{\mathbb{N}}, i) \not\models \varphi.
\end{aligned}$$

Demanding $\varphi \in \Phi$ to be given in DNF, we define $S \subseteq \prod \mathit{Trace} \times \mathbb{P}(\mathbb{N}) \times \mathbb{N} \times \Phi$ as follows. Proofs of correctness are provided in Appendix B.

$$\begin{aligned}
&\forall \tau \in \textstyle\prod_{n\in\mathbb{N}} \mathit{Trace}_n, M \subseteq \mathbb{N}, i \in \mathbb{N}, \varphi \in \Phi \cup \Phi^\Sigma \cup \Psi : S(\tau, M, i, \varphi) \\
&\iff \varphi = \underline{true} \vee \mathit{sys}(\varphi, i, \tau) \subseteq M \\
&\quad \vee(\exists\ e \in \mathcal{E}, j, m \in \mathbb{N} \bullet (\varphi = e \vee \varphi = \underline{repmin}(j, m, e))) \\
&\quad \vee(\exists\ d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C} \bullet \varphi = \underline{isCombined}(d_1, d_2, C)) \\
&\quad \vee(\exists\ d \in \mathcal{D}, m \in \mathbb{N}, C \subseteq \mathcal{C} \bullet (\varphi = \neg\underline{isNotIn}(d, C) \vee \varphi = \neg\underline{isMaxIn}(d, m, C))) \\
&\quad \vee(\exists\ \alpha \in \Phi \cup \Phi^\Sigma \cup \Psi, j \in \mathbb{N} \bullet (\varphi = \alpha\ \underline{before}\ j \wedge S(\tau, M, i - j, \alpha)) \\
&\qquad \vee(\varphi = \neg(\alpha\ \underline{before}\ j) \wedge S(\tau, M, i - j, \neg\alpha))) \\
&\quad \vee(\exists\ \alpha, \beta \in \Phi \cup \Phi^\Sigma \cup \Psi \bullet (\varphi = \alpha\ \underline{since}\ \beta \\
&\qquad\quad \wedge(\exists j \in [0, i] : ((t^\tau_M, j) \models \beta_M \wedge S(\tau, M, j, \beta) \\
&\qquad\qquad \wedge\forall k \in (j, i] : (t^\tau_M, k) \models \alpha_M \wedge S(\tau, M, k, \alpha))) \\
&\qquad \vee(\forall k \in [0, i] : (t^\tau_M, k) \models \alpha_M \wedge S(\tau, M, k, \alpha))) \\
&\quad \vee(\varphi = \neg(\alpha\ \underline{since}\ \beta) \\
&\qquad\quad \wedge(\forall j \in [0, i] : ((t^\tau_M, j) \not\models \beta_M \wedge S(\tau, M, j, \neg\beta) \\
&\qquad\qquad \vee\exists k \in (j, i] : (t^\tau_M, k) \not\models \alpha_M \wedge S(\tau, M, k, \neg\alpha))) \\
&\qquad \wedge(\exists k \in [0, i] : (t^\tau_M, k) \not\models \alpha_M \wedge S(\tau, M, k, \neg\alpha))) \\
&\quad \vee(\varphi = \alpha\ \underline{and}\ \beta \wedge S(\tau, M, i, \alpha) \wedge S(\tau, M, i, \beta)) \\
&\quad \vee(\varphi = \alpha\ \underline{or}\ \beta \wedge ((t^\tau_M, i) \models \alpha_M \wedge S(\tau, M, i, \alpha) \\
&\qquad \vee(t^\tau_M, i) \models \beta_M \wedge S(\tau, M, i, \beta))))
\end{aligned}$$

This formalism allows us to identify situations such as in timesteps 4 and 5 of our example: After the CFO's PDP has observed event *archive(F3)* at time 4, subformula $\underline{not}((\mathit{archive}, \{(\mathit{obj}, D2), (\mathit{user}, CFO)\}))\ \underline{since}\ \underline{false}$ will always evaluate to *false*, implying that policy P2's overall condition φ will always evaluate to *true*. Consequently, all further *edit* requests can safely be disallowed by the CFO's PDP despite the fact that $|\mathit{sys}(\varphi, i, \tau)| > 1$. Once Alice's PDP gets informed that *archive(D2)* happened (Fig. 3, time 4), it is capable of disallowing any further *edit* requests (time 5) without coordination. In sum, all further coordination for enforcing policy P2 can be omitted. Because of space limitations, we do not detail this additional information exchange between PDPs here.

5 Related Work

Chadwick et al. [9] investigate the coordination of distributed stateless PDPs in the access control context. To synchronize resource access across time and space, PDPs synchronize via central 'coordination objects' holding the coordination attributes. Our work is different in that the distributed components are stateful: even policies of a global scale might be evaluated locally (§4.2). Further, our focus is on usage control rather than on access control and our contributions might be implemented in a purely decentralized fashion. Also, our approach enforces policies on all copies of the protected data rather than on only one instance.

Service Automata [11] realize distributed decisions by delegation: If a local monitor's ('service automaton' in [11]) knowledge is insufficient for taking a decision, the decision process is delegated to another local monitor. However, this delegatee is fixed for any pair of conflicting events, thus effectively being a central enforcement point for all corresponding policies. Our approach, in contrast, can be implemented in a pure decentral fashion. As [11] exclusively discusses 'critical events', it remains unclear to which extent Service Automata are able to enforce policies on all copies and derivations of data across systems.

Basin et al. [19] monitor compliance with data usage policies in distributed systems in a *detective* manner: Locally collected logs are merged and a-posteriori evaluated against data usage policies. While [19] also considers propagation of data through the system, our solution targets preventive enforcement.

Lazouski et al. [8] allow for continuous usage control enforcement of data whose copies are distributed. Among policies, also PDP/PIP allocation policies are embedded into the protected data, and they are used by PEPs to locate the PDPs/PIPs responsible for taking decisions. Different to our approach, the responsible PDP is fixed throughout the data's lifetime and for all its copies

Kelbert et al. [10] enable tracking of usage controlled data across systems, as well as enforcement of *local* usage control policies. While distributed PDPs and PIPs exchange information upon cross-system data flows, policies that are of a global scale can not be enforced due to missing coordination between PDPs.

Complementary to our work, Janicke et al. [20] perform *static* analysis of usage control policies with the goal to identify (in)dependencies between PDPs ('Controllers' in [20]). Their analysis results reveal which concurrent decision processes do (not) require synchronization via a central PIP.

Bauer et al. [21] monitor LTL formulas in distributed systems. By leveraging formula rewriting techniques and exchanging rewritten formulas, local monitors can detect satisfaction or violation. Instead of rewriting formulas, our approach exchanges additional information between local monitors. Further, we leverage peculiarities of data usage control policies to minimize communication overhead.

6 Conclusion, Discussion, and Future Work

We have shown how to reduce overall communication overhead when enforcing global data usage control policies such as "only one employee may be editing

document D1 at each point in time". While a naive centralized enforcement infrastructure would impose heavy communication overhead, we provide a distributed data usage control model that supports decentral monitoring of multiple concurrently running systems. Once copies of the protected data, as well as their corresponding usage policies, have been distributed, enforcement of policies that refer to data and data usage events within multiple systems, necessitates the coordination of the decentrally deployed enforcement mechanisms (i.e. PDPs and PIPs). While naively each PDP/PIP could disclose all of its knowledge to all other PDPs/PIPs, our contributions aim at reducing this communication overhead. Hence, we provide formal methods to approximate all systems potentially relevant for evaluating a given policy at each point in time. Subsequently, we can limit coordination to this set of identified systems for enforcement of the given policy. Moreover, we provide insights in which situations coordination between distributed PDPs/PIPs can safely be omitted although the policy to be enforced is of a global scale. Further, we show the correctness of our formal approaches.

We occasionally omitted details for simplicity's sake. The literature [3,10,12, 22,23] discusses more complex data flow states, a slightly more expressive policy language, and the differentiation between intended and actual system events: While intended events can be intercepted, and consequently denied, before their execution, actual events can only be observed thereafter. As our intention was to prevent policy violations, we implicitly assumed events to be intended rather than actual. However, the considerations in §3 and §4 apply to actual events as well. We also tacitly assumed policies to be shipped along with the protected data in case of cross-system data flows. Corresponding mechanisms have been described in the literature [5,8,10,24].

While we have exemplified our general contributions along a running example, the performance of our approach depends on the event traces being observed (predicate S). While in our example no more coordination is needed starting from timestep 5, other formulas might necessitate coordination between subsystems at each point in time. Because of this and because there are usually several ways to technically implement high-level usage policies, we see our contributions as a basis for future work that investigates how policies ought to be specified or transformed to allow for their most efficient enforcement. Along the same lines, our contributions can serve as a basis for building efficient usage control enforcement infrastructures: Given a set of concrete uses cases, i.e. event traces and policies, our contributions can help to answer questions such as where to place PDPs/PIPs in order to minimize communication and performance overhead.

We have not investigated whether the described coordination mechanisms should be implemented in a centralized or decentralized fashion. Since both is possible, we plan to implement both approaches and to compare them for several use cases. Depending on the use case, we expect diverse evaluation results, thus providing further insights into how an efficient enforcement infrastructure can be built. While we have an intuitive understanding which information must be exchanged between PDPs/PIPs (e.g. parts of the data flow state or events happening), the planned implementation will shed further light on this question.

References

1. J. Park and R. Sandhu. The UCON_{ABC} Usage Control Model. *ACM Transactions on Information and System Security*, 7(1):128–174, February 2004.
2. A. Pretschner, M. Hilty, and D. Basin. Distributed Usage Control. *Communications of the ACM*, 49(9):39–44, September 2006.
3. A. Pretschner, E. Lovat, and M. Büchler. Representation-Independent Data Usage Control. In *Data Privacy Management*, LNCS 7122, pages 122–140. Springer, 2012.
4. M. Harvan and A. Pretschner. State-Based Usage Control Enforcement with Data Flow Tracking using System Call Interposition. In *3rd International Conference on Network and System Security*, pages 373–380, 2009.
5. M. Lörscher. Data Usage Control for the Thunderbird Mail Client. Master's thesis, University of Kaiserslautern, Germany, 2012.
6. T. Moses (ed.). eXtensible Access Control Markup Language (XACML) Version 2.0. *OASIS Standard*, pages 1–141, 2005.
7. T. Wüchner and A. Pretschner. Data Loss Prevention Based on Data-Driven Usage Control. In *IEEE 23rd Intl. Symp. Software Reliability Eng.*, pages 151–160, 2012.
8. A. Lazouski, G. Mancini, F. Martinelli, and P. Mori. Architecture, Workflows, and Prototype for Stateful Data Usage Control in Cloud. In *IEEE Security and Privacy Workshops*, 2014.
9. D. Chadwick, L. Su, O. Otenko, and R. Laborde. Coordination between Distributed PDPs. In *7th IEEE Intl. Works. on Policies for Distr. Systems and Networks*, 2006.
10. F. Kelbert and A. Pretschner. Data Usage Control Enforcement in Distributed Systems. In *Proc. 3rd ACM Conference on Data and Application Security and Privacy*, pages 71–82, 2013.
11. R. Gay, H. Mantel, and B. Sprick. Service Automata. In *Formal Aspects of Security and Trust*, LNCS 7140, pages 148–163. Springer, 2012.
12. M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. A Policy Language for Distributed Usage Control. In *Computer Security – ESORICS 2007*, LNCS 4734, pages 531–546. Springer, 2007.
13. A. Pretschner, M. Büchler, M. Harvan, C. Schaefer, and T. Walter. Usage Control Enforcement with Data Flow Tracking for X11. In *Proc. 5th International Workshop on Security and Trust Management*, pages 124–137, 2009.
14. A. Pretschner, M. Hilty, D. Basin, C. Schaefer, and T. Walter. Mechanisms for Usage Control. In *Proc. 2008 ACM Symposium on Information, Computer and Communications Security*, pages 240–244, 2008.
15. P. Kumari and A. Pretschner. Deriving Implementation-level Policies for Usage Control Enforcement. In *Proc. 2nd ACM Conference on Data and Application Security and Privacy*, pages 83–94, 2012.
16. P. Kumari and A. Pretschner. Model-Based Usage Control Policy Derivation. In *Eng. Secure Software and Systems*, LNCS 7781, pages 58–74. Springer, 2013.
17. A. Fromm, F. Kelbert, and A. Pretschner. Data Protection in a Cloud-Enabled Smart Grid. In *Smart Grid Security*, LNCS 7823, pages 96–107. Springer, 2013.
18. C. Kloukinas, G. Spanoudakis, and K. Mahbub. Estimating Event Lifetimes for Distributed Runtime Verification. In *Proc. 20th Intl. Conf. on Software Eng.*, 2008.
19. D. Basin, M. Harvan, F. Klaedtke, and E. Zalinescu. Monitoring Data Usage in Distributed Systems. *IEEE Trans. on Software Eng.*, 39(10):1403–1426, 2013.
20. H. Janicke, A. Cau, F. Siewe, and H. Zedan. Concurrent Enforcement of Usage Control Policies. In *IEEE Workshop on Policies for Distributed Systems and Networks*, pages 111–118, 2008.

21. A. Bauer and Y. Falcone. Decentralised LTL Monitoring. In *FM 2012: Formal Methods*, LNCS 7436, pages 85–100. Springer, 2012.
22. E. Lovat, J. Oudinet, and A. Pretschner. On Quantitative Dynamic Data Flow Tracking. In *Proc. 4th ACM Conference on Data and Application Security and Privacy*, pages 211–222, 2014.
23. E. Lovat and F. Kelbert. Structure Matters – A new Approach for Data Flow Tracking. In *IEEE Security and Privacy Workshops*, May 2014.
24. F. Kelbert and A. Pretschner. Towards a Policy Enforcement Infrastructure for Distributed Usage Control. In *Proc. 17th ACM Symposium on Access Control Models and Technologies*, pages 119–122, 2012.

A Proofs: Correctness of function *sys*

Assuming the formulas to be given in disjunctive normal form (DNF), we show that function *sys* as defined in §4.1 is correct in the following sense: For any tuple of concurrently executing traces $\tau \in \prod_{n\in\mathbb{N}} \mathit{Trace}_n$, point in time $i \in \mathbb{N}$, formula $\varphi \in \Phi$, set of subsystems $M = \mathit{sys}(\varphi, i, \tau)$ and $N \subseteq \mathbb{N}\backslash M$ it holds that

$$(t^\tau_M, i) \models \varphi \iff (t^\tau_{M\cup N}, i) \models \varphi.$$

In other words, the set of subsystems $\mathit{sys}(\varphi, i, \tau)$ is sufficient to evaluate φ at time i given τ. Adding any other set of subsystems to the evaluation process does not change the evaluation's result. For each of the following proofs,
part a) shows $(t^\tau_M, i) \models \varphi \implies (t^\tau_{M\cup N}, i) \models \varphi$, while
part b) shows $(t^\tau_M, i) \models \varphi \impliedby (t^\tau_{M\cup N}, i) \models \varphi$.

Because subsystems' states do not overlap ($\Sigma_s : \mathcal{C}_s \to \mathbb{P}(\mathcal{D})$ and $\mathcal{C}_s \cap \mathcal{C}_{s'} = \emptyset$ for $s \neq s'$), for any tuple of concurrently executing traces $\tau \in \prod_{n\in\mathbb{N}} \mathit{Trace}_n$, any point in time $i \in \mathbb{N}$, and any set of subsystems $M \subseteq \mathbb{N}$ we can define their common state as $\sigma^i_{t^\tau_M} = \{x \in (\mathcal{C} \to \mathbb{P}(\mathcal{D})) \mid \exists m \in M : x \in \sigma^i_{t^\tau_m}\}$. It follows that $\forall M, N \subseteq \mathbb{N}, M \subseteq N : \sigma^i_{t^\tau_M} \subseteq \sigma^i_{t^\tau_N}$. We will make use of this relation between states of sets of subsystems throughout the following proofs.

Proof. For $\varphi = e$.

a) $\forall \tau \in \prod_{n\in\mathbb{N}} \mathit{Trace}_n, i \in \mathbb{N}, e \in \mathcal{E}, \varphi = e, M = \mathit{sys}(\varphi, i, \tau), N \subseteq \mathbb{N}\backslash M :$
$(t^\tau_M, i) \models \varphi$
$\iff \exists e' \in t^\tau_M(i) : (e', \sigma^i_{t_M})\mathit{refines}_\Sigma\ e$
$\implies \exists e' \in t^\tau_{M\cup N}(i) : (e', \sigma^i_{t_{M\cup N}})\mathit{refines}_\Sigma\ e$
$\iff (t^\tau_{M\cup N}, i) \models \varphi$ □

b) Assume: $\exists \tau \in \prod_{n\in\mathbb{N}} \mathit{Trace}_n, i \in \mathbb{N}, e \in \mathcal{E}, \varphi = e, M = \mathit{sys}(\varphi, i, \tau)$
$N \subseteq \mathbb{N}\backslash M : (t^\tau_{M\cup N}, i) \models \varphi \wedge (t^\tau_M, i) \not\models \varphi$
$\iff \exists e' \in t^\tau_{M\cup N}(i) : (e', \sigma^i_{t_{M\cup N}})\ \mathit{refines}_\Sigma\ e$
$\wedge \nexists e'' \in t^\tau_M(i) : (e'', \sigma^i_{t_M})\ \mathit{refines}_\Sigma\ e$
$\implies \exists e' \in t^\tau_N(i) : (e', \sigma^i_{t_N})\ \mathit{refines}_\Sigma\ e$
$\implies N \cap \mathit{mayHappen}(e, i, \tau) \neq \emptyset$
Since $M = \mathit{sys}(e, i, \tau) = \mathit{mayHappen}(e, i, \tau)$ and $N \subseteq \mathbb{N}\backslash M$
$\implies N \cap M \neq \emptyset \wedge N \cap M = \emptyset$. **Contradiction.** □

Proof. For $\varphi = \underline{isCombined}(d_1, d_2, C)$.

$$
\begin{aligned}
&\text{a) } \forall \tau \in \textstyle\prod_{n\in\mathbb{N}} \mathit{Trace}_n, i \in \mathbb{N}, d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C}, \varphi = \underline{isCombined}(d_1, d_2, C),\\
&\quad M = \mathit{sys}(\varphi, i, \tau), N \subseteq \mathbb{N}\backslash M :\\
&\quad (t^\tau_M, i) \models \varphi \iff \exists c \in C : \{d_1, d_2\} \subseteq \sigma^i_{t^\tau_M}(c)\\
&\implies \exists c \in C : \{d_1, d_2\} \subseteq \sigma^i_{t^\tau_{M\cup N}}(c) \iff (t^\tau_{M\cup N}, i) \models \varphi \ \square\\
&\text{b) Assume } \exists \tau \in \textstyle\prod_{n\in\mathbb{N}} \mathit{Trace}_n, i \in \mathbb{N}, d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C},\\
&\quad \varphi = \underline{isCombined}(d_1, d_2, C), M = \mathit{sys}(\varphi, i, \tau), N \subseteq \mathbb{N}\backslash M :\\
&\quad (t^\tau_{M\cup N}, i) \models \varphi \wedge (t^\tau_M, i) \not\models \varphi\\
&\iff \exists c \in C : \{d_1, d_2\} \subseteq \sigma^i_{t^\tau_{M\cup N}}(c) \wedge \nexists c' \in C : \{d_1, d_2\} \subseteq \sigma^i_{t^\tau_M}(c')\\
&\implies \exists c \in C : \{d_1, d_2\} \subseteq \sigma^i_{t^\tau_N}(c) \implies \exists c \in \mathcal{C}_N : \{d_1, d_2\} \subseteq \sigma^i_{t^\tau_N}(c)\\
&\text{Since } N \subseteq \mathbb{N}\backslash M \text{ and } M = \mathit{sys}(\underline{isCombined}(d_1, d_2, C), i, \tau)\\
&\qquad = \mathit{awareD}(\{d_1\}, i, \tau) \cap \mathit{awareD}(\{d_2\}, i, \tau) \cap \mathit{awareC}(C)\\
&\implies M \cap N = \emptyset \wedge M \cap N \neq \emptyset. \textbf{ Contradiction. } \square
\end{aligned}
$$

We omit proofs for further operators due to space limitations.

B Proofs: Correctness of predicate *S*

We show that predicate S as defined in §4.2 is correct in the following sense: For any tuple of concurrently executing traces $\tau \in \prod_{n\in\mathbb{N}} \mathit{Trace}_n$, set of subsystems $M \subseteq \mathbb{N}$, point in time $i \in \mathbb{N}$, formula $\varphi \in \Phi$, it holds that

$$(t^\tau_M, i) \models \varphi_M \wedge S(\tau, M, i, \varphi) \implies (t^\tau_{\mathbb{N}}, i) \models \varphi.$$

Proof. For $\mathit{sys}(\varphi, i, \tau) \subseteq M$.

Follows immediately with the claims and proofs presented in §4.1 and §A.

Proof. For $\varphi = e$

$$
\begin{aligned}
&\forall \tau \in \textstyle\prod_{n\in\mathbb{N}} \mathit{Trace}_n, e \in \mathcal{E}, M \subseteq \mathbb{N}, i \in \mathbb{N}, \varphi = e :\\
&(t^\tau_M, i) \models \varphi_M \iff \exists e' \in t^\tau_M(i) : (e', \sigma^i_{t^\tau_M}) \ \mathit{refines}_\Sigma \ e\\
&\qquad \implies \exists e' \in t^\tau_{\mathbb{N}}(i) : (e', \sigma^i_{t^\tau_{\mathbb{N}}}) \ \mathit{refines}_\Sigma \ e \iff (t^\tau_{\mathbb{N}}, i) \models \varphi \ \square
\end{aligned}
$$

Proof. For $\varphi = \underline{isCombined}(d_1, d_2, C)$

$$
\begin{aligned}
&\forall \tau \in \textstyle\prod_{n\in\mathbb{N}} \mathit{Trace}_n, d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C}, M \subseteq \mathbb{N}, i \in \mathbb{N}, \varphi = \underline{isCombined}(d_1, d_2, C) :\\
&(t^\tau_M, i) \models \varphi_M \iff (t^\tau_M, i) \models \underline{isCombined}(d_1, d_2, C \cap \mathcal{C}_M)\\
&\iff \exists c \in C \cap \mathcal{C}_M : \{d_1, d_2\} \subseteq \sigma^i_{t^\tau_M}(c) \implies \exists c \in C : \{d_1, d_2\} \subseteq \sigma^i_{t_{\mathbb{N}}}(c)\\
&\iff (t^\tau_{\mathbb{N}}, i) \models \underline{isCombined}(d_1, d_2, C) \iff (t^\tau_{\mathbb{N}}, i) \models \varphi \ \square
\end{aligned}
$$

Proof. For $\varphi = \neg\underline{isNotIn}(d, C)$

$$
\begin{aligned}
&\forall \tau \in \textstyle\prod_{n\in\mathbb{N}} \mathit{Trace}_n, d \in \mathcal{D}, C \subseteq \mathcal{C}, M \subseteq \mathbb{N}, i \in \mathbb{N}, \varphi = \neg\underline{isNotIn}(d, C) :\\
&(t^\tau_M, i) \models \varphi_M \iff (t^\tau_M, i) \models \neg\underline{isNotIn}(d, C \cap \mathcal{C}_M)\\
&\iff \neg(\forall c \in C \cap \mathcal{C}_M : d \notin \sigma^i_{t^\tau_M}(c)) \iff \exists c \in C \cap \mathcal{C}_M : d \in \sigma^i_{t^\tau_M}(c)\\
&\implies \exists c \in C : d \in \sigma^i_{t^\tau_{\mathbb{N}}}(c) \iff \neg(\forall c \in C : d \notin \sigma^i_{t^\tau_{\mathbb{N}}}(c))\\
&\iff (t^\tau_{\mathbb{N}}, i) \models \neg\underline{isNotIn}(d, C) \iff (t^\tau_{\mathbb{N}}, i) \models \varphi \ \square
\end{aligned}
$$

Proof. For $\varphi = \neg \underline{isMaxIn}(d, m, C)$

$$\begin{aligned}
&\forall \tau \in \textstyle\prod_{n\in\mathbb{N}} Trace_n, d \in \mathcal{D}, m \in \mathbb{N}, C \subseteq \mathcal{C}, M \subseteq \mathbb{N}, i \in \mathbb{N}, \varphi = \neg \underline{isMaxIn}(d, m, C) : \\
&(t^\tau_M, i) \models \varphi_M \iff (t^\tau_M, i) \models \neg \underline{isMaxIn}(d, m, C \cap \mathcal{C}_M) \\
&\iff |\{c \in C \cap \mathcal{C}_M \mid d \in \sigma^i_{t^\tau_M}(c)\}| > m \implies |\{c \in C \mid d \in \sigma^i_{t_\mathbb{N}}(c)\}| > m \\
&\iff (t^\tau_\mathbb{N}, i) \models \neg \underline{isMaxIn}(d, m, C) \iff (t^\tau_\mathbb{N}, i) \models \varphi \ \square
\end{aligned}$$

Proof. For $\varphi = \alpha \ \underline{before} \ j$

$$\begin{aligned}
&\forall \tau \in \textstyle\prod_{n\in\mathbb{N}} Trace_n, \alpha \in \Phi, j \in \mathbb{N}, M \subseteq \mathbb{N}, i \in \mathbb{N}, \varphi = \alpha \ \underline{before} \ j : \\
&(t^\tau_M, i) \models \varphi_M \wedge S(\tau, M, i-j, \alpha) \iff (t^\tau_M, i) \models \alpha_M \ \underline{before} \ j \wedge S(\tau, M, i-j, \alpha) \\
&\iff (t^\tau_M, i-j) \models \alpha_M \wedge S(\tau, M, i-j, \alpha) \implies (t^\tau_\mathbb{N}, i-j) \models \alpha \\
&\iff (t^\tau_\mathbb{N}, i) \models \alpha \ \underline{before} \ j \iff (t^\tau_\mathbb{N}, i) \models \varphi \ \square
\end{aligned}$$

Proof. For $\varphi = \alpha \ \underline{since} \ \beta$

$$\begin{aligned}
&\forall \tau \in \textstyle\prod_{n\in\mathbb{N}} Trace_n, \alpha, \beta \in \Phi, M \subseteq \mathbb{N}, i \in \mathbb{N}, \varphi = \alpha \ \underline{since} \ \beta : \\
&\quad (t^\tau_M, i) \models \varphi_M \\
&\qquad \wedge (\exists j \in [0, i] : ((t^\tau_M, j) \models \beta_M \wedge S(\tau, M, j, \beta) \\
&\qquad\quad \wedge \forall k \in (j, i] : (t^\tau_M, k) \models \alpha_M \wedge S(\tau, M, k, \alpha)) \\
&\qquad \vee \forall k \in [0, i] : (t^\tau_M, k) \models \alpha_M \wedge S(\tau, M, k, \alpha)) \\
&\iff (\exists j \in [0, i] : ((t^\tau_M, j) \models \beta_M \wedge \forall k \in (j, i] : (t^\tau_M, k) \models \alpha_M \\
&\qquad \vee \forall k \in [0, i] : (t^\tau_M, k) \models \alpha_M)) \\
&\qquad \wedge (\exists j \in [0, i] : ((t^\tau_M, j) \models \beta_M \wedge S(\tau, M, j, \beta) \\
&\qquad\quad \wedge \forall k \in (j, i] : (t^\tau_M, k) \models \alpha_M \wedge S(\tau, M, k, \alpha)) \\
&\qquad \vee \forall k \in [0, i] : (t^\tau_M, k) \models \alpha_M \wedge S(\tau, M, k, \alpha)) \\
&\iff \exists j \in [0, i] : ((t^\tau_M, j) \models \beta_M \wedge S(\tau, M, j, \beta) \\
&\qquad \wedge \forall k \in (j, i] : (t^\tau_M, k) \models \alpha_M \wedge S(\tau, M, k, \alpha)) \\
&\qquad \vee \forall k \in [0, i] : (t^\tau_M, k) \models \alpha_M \wedge S(\tau, M, k, \alpha) \\
&\implies \exists j \in [0, i] : ((t_\mathbb{N}, j) \models \beta \wedge \forall k \in (j, i] : (t_\mathbb{N}, k) \models \alpha) \\
&\qquad \vee \forall k \in [0, i] : (t_\mathbb{N}, k) \models \alpha \\
&\iff (t_\mathbb{N}, i) \models \alpha \ \underline{since} \ \beta \iff (t_\mathbb{N}, i) \models \varphi \ \square
\end{aligned}$$

Proof. For $\varphi = \alpha \ \underline{or} \ \beta$

$$\begin{aligned}
&\forall \tau \in \textstyle\prod_{n\in\mathbb{N}} Trace_n, \alpha, \beta \in \Phi, M \subseteq \mathbb{N}, i \in \mathbb{N}, \varphi = \alpha \ \underline{or} \ \beta : \\
&(t^\tau_M, i) \models \varphi_M \wedge ((t^\tau_M, i) \models \alpha_M \wedge S(\tau, M, i, \alpha) \vee (t^\tau_M, i) \models \beta_M \wedge S(\tau, M, i, \beta)) \\
&\iff ((t^\tau_M, i) \models \alpha_M \vee (t^\tau_M, i) \models \beta_M) \\
&\quad \wedge ((t^\tau_M, i) \models \alpha_M \wedge S(\tau, M, i, \alpha) \vee (t^\tau_M, i) \models \beta_M \wedge S(\tau, M, i, \beta)) \\
&\iff (t^\tau_M, i) \models \alpha_M \wedge S(\tau, M, i, \alpha) \vee (t^\tau_M, i) \models \beta_M \wedge S(\tau, M, i, \beta) \\
&\implies (t^\tau_\mathbb{N}, i) \models \alpha \vee (t^\tau_\mathbb{N}, i) \models \beta \iff (t^\tau_\mathbb{N}, i) \models \alpha \ \underline{or} \ \beta \iff (t^\tau_\mathbb{N}, i) \models \varphi \ \square
\end{aligned}$$

Proof. For $\varphi = \underline{repmin}(j, m, e)$

$$\begin{aligned}
&\forall \tau \in \textstyle\prod_{n\in\mathbb{N}} Trace_n, j, m \in \mathbb{N}, e \in \mathcal{E}, M \subseteq \mathbb{N}, i \in \mathbb{N}, \varphi = \underline{repmin}(j, m, e) : \\
&(t^\tau_M, i) \models \varphi_M \iff (t^\tau_M, i) \models \underline{repmin}(j, m, e) \\
&\iff m \leq \textstyle\sum_{k=0}^{j-1} |\{e' \in t^\tau_M(i-k) \mid (e', \sigma^{i-k}_{t^\tau_M}) \ refines_\Sigma \ e\}| \\
&\implies m \leq \textstyle\sum_{k=0}^{j-1} |\{e' \in t^\tau_\mathbb{N}(i-k) \mid (e', \sigma^{i-k}_{t_\mathbb{N}}) \ refines_\Sigma \ e\}| \\
&\iff (t^\tau_\mathbb{N}, i) \models \underline{repmin}(j, m, e) \iff (t^\tau_\mathbb{N}, i) \models \varphi \ \square
\end{aligned}$$

Again, we omit proofs for further operators due to space limitations.