Policy Evolution in Distributed Usage Control

A. Pretschner[1], F. Schütz[2], C. Schaefer[3], T. Walter[3]

[1] *Information Security, ETH Zurich, Switzerland*
[2] *RUAG Electronics, Berne, Switzerland*
[3] *DoCoMo Euro-Labs, Munich, Germany*

Abstract

Usage control is a generalization of access control that also addresses how data is handled after it is released. Controlling the future usage of data includes controlling the future distribution of data. The evolution of policies upon re-distribution must hence be defined. Intuitively, clients should only strengthen policies associated with a data item when they re-distribute it. We provide a role-based re-distribution model for usage control that encompasses strengthening both rights and duties. By introducing orderings for events and parameter values we show how both rights and duties can be strengthened with the traditional abstraction of trace inclusion.

1 Introduction

The subject of *usage control* [13,14] is the handling of data after it has been given away. Our focus is on usage control in a distributed setting, where processes act in the roles of data providers and data consumers. Data providers can give sensitive data to data consumers based on conditions both on the past and the future. The latter requirements come as *obligations* such as "don't re-distribute" and "delete after thirty days." Obviously, the roles of consumer and provider change dynamically: consumers become providers if they re-distribute previously received data. One fundamental usage control requirement then relates to restricting the re-distribution of data. Without special technology in place, in principle, a subject could distribute the data to itself, thus deleting any usage control requirements. This is obviously not in the interest of the data provider who issued the policy. These originating data providers likely want to specify how their data may and may not be used *after re-distribution*, and thus formulate distribution requirements. In other words, they are likely not only to specify a policy for the consumer, but also to specify policies for *all potential future* consumers of their data. One impractical solution is to specify one policy per potential consumer subject. A somewhat more practical solution consists of role-based policies—one policy per role rather than one policy per subject. However, this solution must not leave consumers total freedom in formulating their own policies when they become providers. Intuitively, given the policy from the originating data provider, they should only be allowed to restrict

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

usage rights and increase duties for the next consumer. If rights could be extended or duties decreased, then a consumer would be able to re-distribute data to himself with obligations that are weaker than those he received together with the respective data item.

In this paper, we tackle the problem of policy evolution in usage control when data is re-distributed. In what precise sense can policies be altered? We provide a conceptual framework for re-distribution in usage control. Roughly, the originating data provider states obligations for specific roles in a system, together with a default policy for all other roles. Whenever data consumers re-distribute data, they may only strengthen the policy that they received from the originating data provider, and they may only do this for precisely specified roles. We provide a formalization of the notion of strengthening—which, as trace inclusion or logical implication, is straightforward for access control but becomes more complicated when not only rights can be restricted but *duties may also be increased*. Our solution relies on specifying rights and duties as intervals on events and parameters. To the best of our knowledge, we are the first to investigate the duality of rights and duties in usage control when it comes to policy refinement, and are the first to provide a respective formalized framework.

2 Background

Our system model for usage control [8] is based on classes of parameterized events. The event classes include *usage* and *other*, with the latter including notification events, for instance. Parameters represent attributes. For example, a usage event must indicate on which data item it is performed, and a signaling event—one that is sent from the consumer to the outside—must name the recipient of the message. An *event* therefore consists of the event name and parameters, represented as a partial function ($\nrightarrow$) from names to values. We will describe event parameters as $(name, value)$ pairs. An example is the event $(play, \{(object, o)\})$, where *play* is the name of the event and the parameter *object* has the value o.

The definition of events in the Z language is shown below. *EventName*, *ParamName*, and *ParamValue* define basic types for event names, parameter names, and parameter values, respectively. In Z, such definitions are made by listing the types in square brackets. All such basic types are disjoint. EBNF-style definitions are also possible.

$$[EventName, ParamName, ParamValue]$$

$$EventClass ::= usage \mid other; \quad getclass : EventName \rightarrow EventClass$$

$$Params : ParamName \nrightarrow ParamValue; \quad Event == EventName \times Params$$

Events are ordered with respect to a refinement relation *refinesEv*. Event e_2 refines event e_1 iff e_2 has the same event name as e_1 and all parameters of e_1 have the same value in e_2. e_2 can also have additional parameters specified and hence the $\subseteq$ relation in the definition below. (In such *axiomatic definitions*, the defined mathematical object is named and typed above the line and its properties are given below the line.)

$$\_ \mathit{refinesEv} \_ : \mathit{Event} \leftrightarrow \mathit{Event}$$
$$\forall e_1, e_2 : \mathit{Event} \bullet e_2 \mathit{refinesEv}\ e_1 \Leftrightarrow e_1.1 = e_2.1 \land e_1.2 \subseteq e_2.2$$

The rationale is that when specifying usage control requirements, we do not want to specify all parameters. For instance, if the event $(\mathit{play}, \{(\mathit{object}, o)\})$ is prohibited, then the event $(\mathit{play}, \{(\mathit{object}, o), (\mathit{device}, d)\})$ should also be prohibited. $x.i$ identifies the i-th component of a tuple x. $\leftrightarrow$ introduces a binary relation. The event name *nil* of type EventName is reserved and denotes no event.

To define usage control requirements, we need a language for usage control. Its semantics is defined over traces: mappings from abstract points in time—represented by the natural numbers—to possibly empty sets of events. We cater for usage events that execute over a time interval, e.g., watching a movie, by distinguishing between starting and ongoing events. The data type *IndEvent* defines such indexed events. In Z, types can, among other things, be defined by enumeration or by Cartesian products ($\times$).

$$\mathit{Index} ::= \mathit{start} \mid \mathit{ongoing}; \quad \mathit{IndEvent} == \mathit{Event} \times \mathit{Index}$$

The *Obligation Specification Language* (OSL, formally captured by Φ^+) is a temporal logic similar to LTL, with additional operators for cardinality (that, in turn, can be expressed in LTL [9, §3.4.3]). OSL has been defined as part of earlier work [8]. We will later extend OSL with ordered events and intervals which are necessary to express distribution requirements. In Z, records can be defined by stating the name of a constructor and its arguments in angular brackets, and if there is more than one argument, this is expressed with the Cartesian product.

$$\begin{aligned}
&\Phi^+ ::= \underline{\mathit{true}} \mid \underline{\mathit{false}} \mid E_{\mathit{fst}}\langle\langle \mathit{Event}\rangle\rangle \mid E_{\mathit{all}}\langle\langle \mathit{Event}\rangle\rangle \mid \underline{\mathit{not}}\langle\langle \Phi^+\rangle\rangle \mid \underline{\mathit{and}}\langle\langle \Phi^+ \times \Phi^+\rangle\rangle \mid \underline{\mathit{or}}\langle\langle \Phi^+ \times \Phi^+\rangle\rangle \mid \\
&\underline{\mathit{implies}}\langle\langle \Phi^+ \times \Phi^+\rangle\rangle \mid \underline{\mathit{until}}\langle\langle \Phi^+ \times \Phi^+\rangle\rangle \mid \underline{\mathit{always}}\langle\langle \Phi^+\rangle\rangle \mid \underline{\mathit{after}}\langle\langle \mathbb{N} \times \Phi^+\rangle\rangle \mid \underline{\mathit{within}}\langle\langle \mathbb{N} \times \Phi^+\rangle\rangle \mid \\
&\underline{\mathit{during}}\langle\langle \mathbb{N} \times \Phi^+\rangle\rangle \mid \underline{\mathit{repmax}}\langle\langle \mathbb{N} \times \Phi^+\rangle\rangle \mid \underline{\mathit{replim}}\langle\langle \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \Phi^+\rangle\rangle \mid \underline{\mathit{repuntil}}\langle\langle \mathbb{N} \times \Phi^+ \times \Phi^+\rangle\rangle \mid \\
&\underline{\mathit{permitonlyevname}}\langle\langle \mathbb{P}\, \mathit{EventName} \times \mathit{Params}\rangle\rangle \mid \\
&\underline{\mathit{permitonlyparam}}\langle\langle \mathbb{P}\, \mathit{ParamValue} \times \mathit{ParamName} \times \mathit{EventName} \times \mathit{Params}\rangle\rangle
\end{aligned}$$

Fig. 1. Syntax of OSL [8]

The syntax of OSL is given in Fig. 1. We distinguish between the start of an action (syntactically: E_{fst}; semantically: an indexed event with index *start*) and any lasting action (syntactically: E_{all}; semantically: indexed events with any index). When specifying events in obligations, by virtue of the refinement relation, there is an implicit universal quantification over unmentioned parameters. *not*, *and*, *or*, *implies* have the usual semantics, and we will use the infix operators $\neg, \land, \lor, \Rightarrow$ as shorthand. *until*, *after*, *during*, and *within* are temporal operators with an intuitive meaning that is formalized in Fig. 2.

Cardinality operators restrict the number of occurrences or the duration of an action. The *replim* operator specifies lower and upper bounds of time steps within a fixed time interval in which a given formula holds. The *repuntil* operator does the same, but independent of any time interval. Instead, it limits the maximal number of times a formula holds until another formula holds (e.g., the occurrence of some event). With the help of *repuntil*, we can also define *repmax*, which defines the maximal number of times a formula may hold in the indefinite future. These cardinality operators are also used to express limits on the accumulated usage time.

We support both the "must" and the "may" modalities. The former is given by OSL's LTL-like semantics, and the latter is supported by two designated operators. The operator *permitonlyevname* defines the names of the usage events that are exclusively allowed with a set of given parameters. Similarly, *permitonlyparam* only allows specific values for a given parameter of an event. It prohibits all other values for this parameter. The semantics of Φ^+ is formally defined by the binary relation $\models_f$ in Fig. 2. The definition makes use of a shorthand, $\models_e$, to relate single indexed events (rather than traces) to formulae of the form $E_{fst}(\cdot)$ or $E_{all}(\cdot)$. As we will see, to express the refinement of policies, $\models_e$ will be almost all we have to adjust.

3 Re-Distribution

To illustrate the problems related to policy distribution, we consider the example scenario of a movie provider. *FairMovies* produces all sorts of movies and distributes them through licensed *dealers*. These dealers sell the movies to *customers*. FairMovies requires all dealers to provide movies to *film students* and *reporters* at no cost. The distribution scheme is depicted in Fig. 3.

FairMovies wants to *receive money* for each copy of a movie that is transferred to a private customer. However, FairMovies *limits the price range* for movies because a bad price strategy could damage its reputation. FairMovies does not want movies to be sold before they are shown in theaters. Authorities may request FairMovies to take measures that ensure that movies are only sold to customers at a proper age. Thus, FairMovies wants to *control distribution* based on environmental and subject attributes (see below). *Dealers* want to *receive payments* from their customers for each copy sold and they want to be able to *adjust prices* according to their selling strategy. *Film Academia* wants its students to learn about camera techniques, special effects, cutting and the like. To this end, academics must be able to *play and edit* the full quality content. *Reporters* must be able to *preview the full movie* to write movie reviews. *Customers* want to *play the movie in full length and quality.* They want to *share* movies with others. And they want to *lend* movies to friends and *sell* old movies that they don't watch anymore. Customers want to *recover fast* from data loss. Therefore they demand backup capabilities.

The customer requirement to share or trade movies is in conflict with FairMovies requirement to receive money. As a solution FairMovies allows customers to share movies for free under the condition that the new copy's video quality is reduced by 50% upon each re-distribution. Giving away free copies to academia and media does not conflict with FairMovies requirements, since this improves publicity. Unfortunately, pirated copies often originate from one of these parties. For this reason, FairMovies wants to apply conditions that restrict reporters and academics. Reporters are allowed to watch a movie at most three times. Academics are allowed to re-distribute (modified) movies to fellow academics only. — Studying the example scenario, we identify four requirements a system must support.

Usage Conditions: The originating data provider wants to specify conditions for individual subjects and/or subject groups. E.g., only customers are allowed to make backups. Thus, restrictions can be seen as individual usage policies for subjects or groups of subjects.

$$\_ \models_e \_ : IndEvent \leftrightarrow \Phi^+$$

$$\forall\, ie : IndEvent;\ \varphi : \Phi^+ \bullet ie \models_e \varphi \Leftrightarrow \exists\, e : Event \bullet$$
$$ie.1\ refinesEv\ e \wedge ((\varphi = E_{fst}(e) \wedge ie.2 = start) \vee \varphi = E_{all}(e))$$

$$\_ \models_f \_ : (Trace \times \mathbb{N}) \leftrightarrow \Phi^+$$

$$\begin{aligned}
&\forall\, s : Trace;\ t : \mathbb{N};\ \varphi : \Phi^+ \bullet (s,t) \models_f \varphi \Leftrightarrow \\
&\quad \varphi = \underline{true} \vee \varphi = E_{fst}((nil, \varnothing)) \vee \varphi = E_{all}((nil, \varnothing)) \\
&\quad \vee \exists\, e : Event \setminus \{(nil, \varnothing)\};\ ie : IndEvent \bullet (\varphi = E_{fst}(e) \vee \varphi = E_{all}(e)) \wedge ie \in s(t) \wedge ie \models_e \varphi \\
&\quad \vee \exists\, \psi : \Phi^+ \bullet \varphi = \underline{not}(\psi) \wedge \neg\, ((s,t) \models_f \psi) \\
&\quad \vee \exists\, \psi, \chi : \Phi^+ \bullet \varphi = \underline{or}(\psi, \chi) \wedge ((s,t) \models_f \psi \vee (s,t) \models_f \chi) \\
&\quad \vee \exists\, \psi, \chi : \Phi^+ \bullet \varphi = \underline{until}(\psi, \chi) \\
&\qquad \wedge (\exists\, u : \mathbb{N} \mid t < u \bullet \Big((s,u) \models_f \chi \wedge (\forall\, v : \mathbb{N} \mid t < v < u \bullet (s,v) \models_f \psi)\Big) \vee (\forall\, v : \mathbb{N} \mid t < v \bullet \\
&\qquad\quad (s,v) \models_f \psi)) \\
&\quad \vee \exists\, i : \mathbb{N};\ \psi : \Phi^+ \bullet \varphi = \underline{after}(i, \psi) \wedge (s, t+i) \models_f \psi \\
&\quad \vee \exists\, i : \mathbb{N}_1;\ m, n : \mathbb{N};\ \psi : \Phi^+;\ e : Event \bullet \\
&\qquad \varphi = \underline{replim}(i, m, n, \psi) \wedge (\psi = E_{fst}(e) \vee \psi = E_{all}(e)) \wedge \\
&\qquad\quad m \leq (\sum_{j=1}^{i} \#\{ie : IndEvent \mid ie \in s(t+j) \wedge ie \models_e \psi\} \leq n \\
&\quad \vee \exists\, n : \mathbb{N};\ \psi, \chi : \Phi^+;\ e : Event \bullet \varphi = \underline{repuntil}(n, \psi, \chi) \wedge (\psi = E_{fst}(e) \vee \psi = E_{all}(e)) \\
&\qquad \wedge \Big((\exists\, u : \mathbb{N}_1 \bullet (s, t+u) \models_f \chi \wedge (\forall\, v : \mathbb{N}_1 \mid v < u \bullet \neg((s, t+v) \models_f \chi)) \\
&\qquad\quad \wedge (\sum_{j=1}^{u} \#\{ie : IndEvent \mid ie \in s(t+j) \wedge ie \models_e \psi\}) \leq n) \\
&\qquad \vee (\sum_{j=1}^{\infty} \#\{ie : IndEvent \mid ie \in s(t+j) \wedge ie \models_e \psi\}) \leq n\Big) \\
&\quad \vee \exists\, ex : \mathbb{P}\, EventName;\ ps : Params \bullet \varphi = \underline{permitonlyevname}(ex, ps) \\
&\qquad \wedge \forall\, en : EventName \mid getclass(en) = usage \wedge en \notin ex \bullet \\
&\qquad\quad (t,n) \models_f \underline{always}(\underline{not}(E_{all}((en, ps)))) \\
&\quad \vee \exists\, ex : \mathbb{P}\, ParamValue;\ pn : ParamName;\ en : EventName;\ ps : Params \bullet \\
&\qquad \varphi = \underline{permitonlyparam}(ex, pn, en, ps) \wedge pn \notin \mathsf{dom}\ ps \\
&\qquad \wedge \forall\, pv : ParamValue \mid pv \notin ex \bullet \\
&\qquad\quad (t,n) \models_f \underline{always}(\underline{not}(E_{all}((en, ps \cup \{(pn, pv)\})))) \\
&\quad \vee \varphi = \underline{false} \wedge (s,t) \models \underline{not}(\underline{true}) \\
&\quad \vee \exists\, \psi, \chi : \Phi^+ \bullet \varphi = \underline{and}(\psi, \chi) \wedge (s,t) \models_f \underline{not}(\underline{or}(\underline{not}(\psi), \underline{not}(\chi))) \\
&\quad \vee \exists\, \psi, \chi : \Phi^+ \bullet \varphi = \underline{implies}(\psi, \chi) \wedge (s,t) \models_f \underline{or}(\underline{not}(\psi), \chi) \\
&\quad \vee \exists\, \psi : \Phi^+ \bullet \varphi = \underline{always}(\psi) \wedge (s,t) \models_f \underline{until}(\psi, \underline{false}) \\
&\quad \vee \exists\, i : \mathbb{N};\ \psi : \Phi^+ \bullet \varphi = \underline{within}(i, \psi) \wedge (s,t) \models_f \underline{replim}(i, 1, i, \psi) \\
&\quad \vee \exists\, i : \mathbb{N};\ \psi : \Phi^+ \bullet \varphi = \underline{during}(i, \psi) \wedge (s,t) \models_f \underline{replim}(i, i, i, \psi) \\
&\quad \vee \exists\, n : \mathbb{N};\ \psi : \Phi^+ \bullet \varphi = \underline{repmax}(n, \psi) \wedge (s,t) \models_f \underline{repuntil}(n, \psi, \underline{false})
\end{aligned}$$

Fig. 2. Semantics of OSL [8]

Distribution Conditions: Data providers want to specify pre- and postconditions for distribution. An example postcondition is the requirement that dealers must pay royalties to FairMovies after selling a movie. An exemplary precondition is that FairMovies does not allow distribution before a movie has been shown in cinema.

Distribution Path: An originating data provider wants to control which subjects are allowed to get the data. Moreover, he might want to control who can distribute to whom, thus effectively describing "paths" between subjects along which data and policies can be sent.

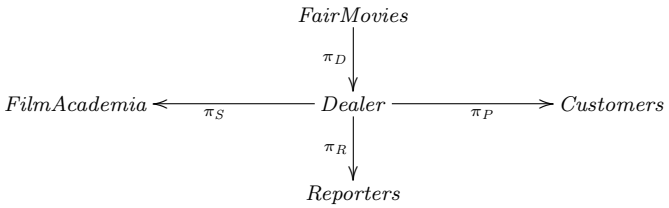

Fig. 3. Fair Movies distribution scheme

Policy Modifications: There are two forms of policy modifications. The explicit form is what a data provider is allowed to negotiate. The originating data provider may allow any subsequent data provider to negotiate within certain boundaries. For example, FairMovies allows dealers to negotiate the price with the customer, but does not allow the movie to be sold too cheap or too expensive. The second form of policy modifications is automatic policy evolution: the originating data provider specifies functions that must be applied to the policy before it is propagated, thus evolving the policy. An example for this is the request for quality reduction when distributing movies to others.

3.1 Strengthening Policies

Our notion of strengthening policies assumes without loss of generality the perspective of the data provider. A policy is strengthened by reducing the consumer's rights (benefits) or by increasing the consumer's duties.

Rights. We start by assuming that a subject can only strengthen policies upon re-distribution of a data item. For now, we will only consider rights, as done in access control. Duties are considered below. Intuitively, one would argue that upon re-distribution, rights can only be decreased. In formal terms, the set of traces that satisfy the original policy must become smaller, or, logically speaking, the formula that corresponds to the new policy must imply the old policy.

Consider an event name *play* with one parameter for video quality. Video quality, *vq*, ranges from 0 to 100. One example event is $(play, \{(obj, movA), (vq, 75)\})$, indicating that *movA* is played with 75% video quality (throughout the paper, we assume that the maximum value is 100). A policy specifying that *movA* may only be played with this restricted quality is given by $\underline{permitonlyparam}(\{75\}, vq, play, \{(obj, movA)\})$. Now, intuitively, the data provider is unlikely to object to any consumer playing the movie at a quality *below* 75%. In other words, the intended policy is probably $\underline{permitonlyparam}(\{0, ..., 75\}, vq, play, \{(obj, movA)\})$. Note that the second policy can intuitively be refined to a policy which allows less quality, while the first policy cannot. Similarly, let us assume a policy $\underline{permitonlyevname}(\{edit\}, \{(obj, movA)\})$, specifying that *movA* may be edited both for sound and video content. It is likely that the data provider would not object to the movie also being played, or edited for video content only, or edited for sound only. The intended policy is hence likely to be $\underline{permitonlyevname}(\{edit, sndedit, vdoedit, play\}, \{(obj, movA)\})$. One possible restriction of this policy would then allow to edit the movie's sound track only (and, at the same time, also allow to play the movie).

Ordered Events and Parameters. In sum, policies tend to formulate permissions not for single events or single parameters only. Instead, they assume an

implicit ordering of events. In order to simplify policies, and in order to provide a formal framework for strengthening policies, we make these orderings explicit. Events and parameters will be partially ordered, that is, ordered w.r.t. a relation that is reflexive, antisymmetric, and transitive. For brevity's sake, we omit the straightforward formalization of these concepts. In the following, all orders $\leq$, possibly indexed, are assumed to be partial orders.

As a first step, we define a partial order for event names, which is, in fact, a bounded lattice. In addition to the order provided by the system designer, we assume the existence of two special event names $\top_e$ and $\bot_e$ that are larger, respectively smaller than any other event name. We can then define lattices on event names, $\leq_e$, and event parameters, $\leq_p$; the straightforward formalization is again omitted for brevity's sake. Fig. 4 shows an example for a partial order of events (left) and one particular set of parameter values, e.g., video quality.

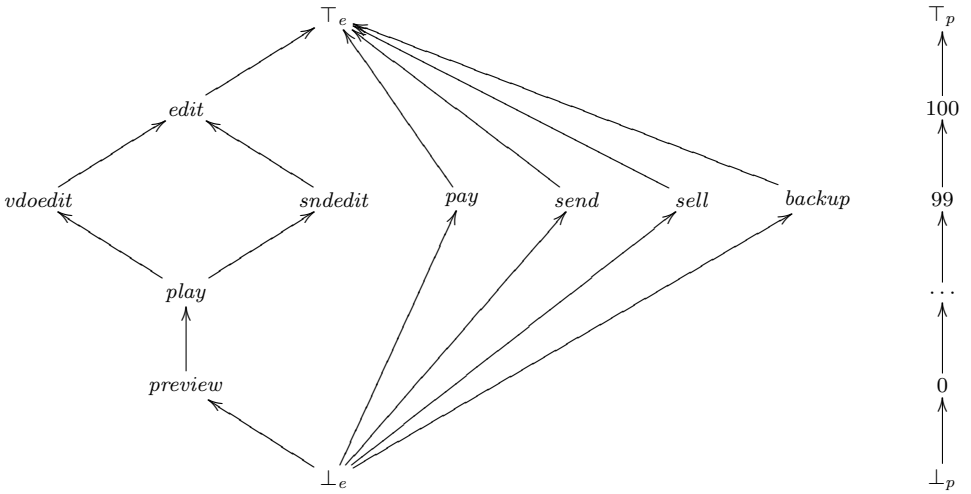

Fig. 4. Partially ordered event names (left) and parameter values (right)

In order to use ordered events and ordered parameter values in OSL policies, we first have to extend the syntax. We introduce *ordered events* that consist of a pair of event names and a pair of parameter values, captured by the following definitions.

$$OrdParams == ParamName \nrightarrow (ParamValue \times ParamValue)$$
$$OrdEventName == EventName \times EventName; \quad OrdEvent == OrdEventName \times OrdParams$$

Intuitively, if an ordered event is allowed to happen, then all events in between the two event names with all parameter values in between the two boundary parameter values are also allowed to happen. Postponing the motivation for the moment, we stipulate that for each ordered event,

- one of the following is true: (1) the left boundary of the event name is $\bot_e$; or (2) the right boundary of the event name is $\top_e$; or (3) both boundaries are identical;
- and that in addition one of the following also holds for each ordered parameter value: (1) the left boundary is $\bot_p$; or (2) the right boundary is $\top_p$; or (3) the left and right boundaries are identical.

Without a respective formalization, we furthermore require the following "sanity" conditions to hold. Firstly, any ordered event must have valid lower and upper bounds for the event parameters. This means that the bounds must consist of

$$\Phi_o^+ ::= \underline{true} \mid \underline{false} \mid E_{fst}\langle\langle OrdEvent\rangle\rangle \mid E_{all}\langle\langle OrdEvent\rangle\rangle \mid \underline{not}\ \langle\langle \Phi_o^+\rangle\rangle \mid \underline{and}\ \langle\langle \Phi_o^+ \times \Phi_o^+\rangle\rangle \mid \underline{or}\ \langle\langle \Phi_o^+ \times \Phi_o^+\rangle\rangle \mid$$
$$\underline{implies}\ \langle\langle \Phi_o^+ \times \Phi_o^+\rangle\rangle \mid \underline{until}\ \langle\langle \Phi_o^+ \times \Phi_o^+\rangle\rangle \mid \underline{always}\ \langle\langle \Phi_o^+\rangle\rangle \mid \underline{after}\ \langle\langle \mathbb{N} \times \Phi_o^+\rangle\rangle \mid \underline{within}\ \langle\langle \mathbb{N} \times \Phi_o^+\rangle\rangle \mid$$
$$\underline{during}\ \langle\langle \mathbb{N} \times \Phi_o^+\rangle\rangle \mid \underline{repmax}\ \langle\langle \mathbb{N} \times \Phi_o^+\rangle\rangle \mid \underline{replim}\ \langle\langle \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \Phi_o^+\rangle\rangle \mid \underline{repuntil}\ \langle\langle \mathbb{N} \times \Phi_o^+ \times \Phi_o^+\rangle\rangle \mid$$
$$\underline{permitonlyevname}\ \langle\langle \mathbb{P}\ OrdEventName \times OrdParams\rangle\rangle \mid$$
$$\underline{permitonlyparam}\ \langle\langle \mathbb{P}(ParamValue \times ParamValue) \times ParamName \times OrdEventName \times OrdParams\rangle\rangle$$

$$\_ \models_e^o \_ : IndEvent \leftrightarrow \Phi_o^+$$
$$\forall\, ie : IndEvent;\ \varphi : \Phi_o^+ \bullet ie \models_e^o \varphi \Leftrightarrow \exists\, en, en_l, en_u : EventName;\ ps : Params;\ ops : OrdParams \bullet$$
$$ie = (en, ps) \wedge ((\varphi = E_{fst}(((en_l, en_u), ops)) \wedge ie.2 = start) \vee \varphi = E_{all}(((en_l, en_u), ops)))$$
$$\wedge\ en_l \leq_e en \wedge en \leq_e en_u \wedge \forall\, pn : ParamName;\ pv, pv_l, pv_u : ParamValue \mid (pn, pv) \in ps \bullet$$
$$(pn, (pv_l, pv_u)) \in ops \Rightarrow pv_l \leq_p pv \wedge pv \leq_p pv_u$$

$$\_ \models_f^o \_ : (Trace \times \mathbb{N}) \leftrightarrow \Phi_o^+$$
$$\forall\, s : Trace;\ t : \mathbb{N};\ \varphi : \Phi_o^+ \bullet (s, t) \models_f^o \varphi \Leftrightarrow$$
$$\varphi = \underline{true} \vee \varphi = E_{fst}((nil, \varnothing)) \vee \varphi = E_{all}((nil, \varnothing))$$
$$\vee\, \exists\, e : Event \setminus \{(nil, \varnothing)\};\ ie : IndEvent \bullet (\varphi = E_{fst}(e) \vee \varphi = E_{all}(e)) \wedge ie \in s(t) \wedge ie \models_e^o \varphi$$
$$\ldots$$
$$\vee\, \exists\, ex : \mathbb{P}\ OrdEventName;\ ps : OrdParams \bullet$$
$$\varphi = \underline{permitonlyevname}\ (ex, ps) \wedge$$
$$\forall\, en, en_u, en_l : EventName;\ pn : ParamName;\ pv_l, pv_u : ParamValue \mid getclass(en) = usage$$
$$\wedge\ (en_l, en_u) \in ex \wedge (en_u <_e en \vee en <_e en_l \bullet (s, t) \models_f^o \underline{always}\ (\underline{not}\ (E_{all}((en, ps))))$$
$$\vee\, \exists\, ex : \mathbb{P}(ParamValue \times ParamValue);\ oen : OrdEventName;\ ps : OrdParams;\ pn : ParamName \bullet$$
$$\varphi = \underline{permitonlyparam}\ (ex, pn, oen, ps) \wedge pn \notin \mathsf{dom}\ ps \wedge$$
$$\forall\, pv, pv_l, pv_u : ParamValue \mid (pv_l, pv_u) \in ex \wedge (pv_u <_p pv \vee pv <_p pv_l) \bullet$$
$$(s, t) \models_f^o \underline{always}\ (\underline{not}\ (E_{all}((oen, ps \cup \{(pn, (pv, pv))\}))))$$
$$\ldots$$

Fig. 5. Syntax (Φ_o^+) and semantics ($\models_f^o$) of OSL with ordered events and parameters

values that are defined for the respective parameters. Secondly, every parameter name given in an ordered event must be declared for at least one event from the interval given by the ordered event name. Finally, the values used for the lower and upper bounds of parameter values must exist for this event parameter. We omit a concrete formalization of these constraints here. Instead, we adjust the definition of Φ^+ to cater for ordered events and ordered parameter values as defined by Φ_o^+ in Fig. 5. Semantically, we only have to adjust $\models_e$ to check if an indexed event lies in the specified interval, and the permission operators. Since $\models_e$ is the anchor of every definition in $\models_f$, with most operators it suffices to replace events by ordered events. Essentially, an event satisfies a formula with an ordered event as subformula if the event name and all parameters are within the given bounds. In addition to $\models_e$, only *permitonlyevname* and *permitonlyparam* must be redefined in such a way that not only single events or parameters can be whitelisted, but also whole intervals. For *permitonlyevname* this is done by checking for each event whether it falls in one of the permitted intervals. If it does not, then this single event is forbidden. *permitonlyparam* is defined analogously. The formalization is provided in Fig. 5.

As an example, we can specify that a movie *movA* may be previewed or played with a quality of at most 75%: $\underline{always}(\underline{not}(E_{all}(((\bot_e, play), \{(obj, (movA, movA)), (vq, (76, \top_p))\}))))$. Note that we have to use a singleton interval for the *obj* parameter—of course, the implementation

of our language expands a single parameter to the singleton interval. Among other things, however, the above allows *movA* to be edited by anyone, and the intended policy likely was rather $\underline{permitonlyevname}(\{(\bot_e, play)\}, \{(obj, (movA, movA))\}) \wedge \underline{permitonlyparam}(\{(\bot_p, 75)\}, vq, (\bot_e, play), \{(obj, (movA, movA))\})$, which specifies that the exclusively allowed operations on *movA* are those in between $\bot_e$ and *play* with, if applicable, values in between $\bot_p$ and 75 for parameter name *vq*. A possible restriction of this policy would be to restrict the set of events that can be executed on object *movA*, or to reduce the interval for video quality.

Logical Characterization. Intuitively, a formula φ' is stronger than a formula φ if every trace satisfying φ' does not give more rights and imposes at least as many duties as every trace that satisfies φ (note that we have not discussed the issue of duties yet, but, as we will see, this will be easy to incorporate). In other words, if every trace that satisfies φ' also satisfies φ, φ' is as strong or stronger than φ. Semantically, our notion of stronger policies then naturally boils down to trace inclusion. Having introduced the semantics for formulas with ordered events, we lift $\leq_e$ and $\leq_p$, the partial orders for events and parameter values, to OSL formulas by the following definition.

$$\frac{\_ \leq_f \_ : \Phi_o^+ \leftrightarrow \Phi_o^+}{\forall \varphi, \varphi' : \Phi_o^+ \bullet \varphi' \leq_f \varphi \Leftrightarrow \quad \forall s : Trace;\ t : \mathbb{N} \bullet (s,t) \models_f^o \underline{implies}(\varphi', \varphi)}$$

For instance, the intuition that $E_{all}(((play, play), \{(obj, (movA, movA)), (vq, (\bot_p, 50))\}))$ is stronger than $E_{all}(((play, play), \{(obj, (movA, movA), (vq, (\bot_p, 100))\})))$ is met by the formal definition.

Duties. We have seen that usage control policies do not only specify permissions but also duties. Strengthening policies relates to both aspects. We have seen that one possibility to strengthen policies is the reduction of rights. Another possibility is to impose further duties on the consumer. As we will see, this can be coped with in exactly the same way as the restriction of rights.

Consider a policy that specifies that \$1 has to be paid whenever movie *movA* is played, formally captured by $\underline{always}(E_{fst}((play, play), \{(obj, (movA, movA))\}) \Rightarrow E_{fst}((pay, pay), \{(amt, (1, 1))\}))$. Upon re-distribution of the movie, the new provider may decide that the new consumer should pay \$2 rather than \$1. If the initial policy was $\underline{always}(E_{fst}((play, play), \{(obj, (movA, movA))\}) \Rightarrow E_{fst}((pay, pay), \{(amt, (1, \top_p))\}))$, this could, again simply by restricting intervals, be strengthened into $\underline{always}(E_{fst}((play, play), \{(obj, (movA, movA))\}) \Rightarrow E_{fst}((pay, pay), \{(amt, (2, \top_p))\}))$. This is not possible with the original policy that specified precisely \$1 to be paid.

Interval Boundaries. One way to strengthen policies is hence to shorten intervals. The reason for requiring either boundaries to be identical or left or right boundaries to be bottom or top elements of the respective lattices is the following. If a provider required payments to be in between \$1 and \$500, then one possibility to strengthen the policy would be to require a payment in between \$1 and \$250. While this clearly reduces the consumer's *number of possibilities* to fulfill his duties, it is not an intuitive strengthening of the duties. Increasing the lower bound, however, intuitively strengthens the duty. Conversely, it does not seem reasonable for a provider to grant the right of watching a movie at a quality that is lower than 75%

but above 25%.

As it turns out, rights usually have $\bot_e$ or $\bot_p$ as left boundary, and duties have $\top_e$ or $\top_p$ as right boundary. In other words, requiring lower bounds to be $\bot_e, \bot_p$ or upper bounds to be $\top_e, \top_p$ amounts to specifying *maximum rights* and *mininum duties*. Assuming that there are no composite events that combine both rights and duties, any event can hence be classified as right or duty by simply regarding the interval boundaries. This obviously also gives rise to a slightly different syntax: upon policy specification, we could classify events into rights and duties and omit either left or right boundaries for both names and parameter values. These missing boundaries could then automatically be filled in.

One might argue that it is not always clear if an event denotes a permission or a duty. For instance, a *play* event may relate to both playing a movie (right) and playing a commercial (duty). However, it seems that events can be assigned a clear status of being either a right (*playMovie*) or a duty (*playCommercial*). Furthermore, one might be concerned that negating a duty leads to a right and that negating a right leads to a duty. This can indeed be the case but need not necessarily: the negation of "pay $5 or more" would be "do not pay $5 or more", and this is not a right in the intuitive sense ("you may pay less than $4"). Either way, not labeling events as rights or duties but rather classifying them implicitly on the grounds of interval boundaries avoids any confusion. For this reason, we stick to explicit boundaries in this paper.

Note that there is a noteworthy asymmetry. Restricting rights can, in principle, also mean to increase the lower bound. The interpretation is then that *consumers are restricted in their freedom to choose*, even though in the example of increasing the minimum quality, this may appear odd. In other words, one can argue that reducing rights can happen at both sides of the spectrum. On the other hands, this does not seem too reasonable for duties, as explained above. In fact, we decided to fix lower boundaries for rights to be $\bot_p$ or $\bot_e$ only on the grounds of symmetry considerations.

Finally, note that our notion of strengthening policies defined by logical implication "automatically" caters for strengthening along the temporal dimension: having to do something within 5 time steps can be strengthened to doing it within 3 steps.

3.2 Variables

Our policy language for distribution also includes a few variables, the most important ones being %SENDER and %RECEIVER. At runtime, these are bound to the provider and the consumer of the last distribution step. With these variables it is, for instance, possible to state that money must be paid to the last provider but not necessarily the originating data provider. Similarly, variables can relate to the current state of a system—which assumes that this local or global state can be accessed. A possible extension is the introduction of variables that allows to express requirements such as "quality to be halved upon every distribution step", which we illustrate below.

3.3 Role-Based Distribution

We are now ready to define a role-based distribution schema. We assume that the initial policy is defined by the originating data provider. The originating data provider defines one policy per role and possibly per subject. We will assume that a subject's policy has priority over a role's policy. A subject can be in multiple roles, and in case of conflict, we assume some further conflict resolution mechanism to exist. In case a data item is sent to a subject for which neither a dedicated (subject) policy nor a policy for the subject's role is defined, a default rule applies. This default rule must be provided by the original data provider.

Whenever data items are re-distributed, the provider can at most strengthen the (originating policy provider's) policy. Any distribution component must hence be able to decide if $p_1 \leq_f p_2$ holds for policies p_1 and p_2.

However, it may be that the originating data provider does not want future providers to be able to strengthen the policies of *all* roles. To this end, we introduce a partial order on roles, $\leq_r$. A potential data provider in role r_1 may only strengthen the originating data provider's policies for all those roles r where $r \leq_r r_1$ (that is, a "larger" role may only change the policies of "lower" roles). Note that this is a *distribution* hierarchy only; we *do not* require that if (sub-)policies p_1 and p_2 are associated with roles r_1 and r_2 with $r_1 \leq_r r_2$, then also $p_1 \leq_f p_2$.

3.4 Example: FairMovies

We now present the formal policy for our running example. For brevity's sake, we will only discuss the most interesting requirements.

Usage Conditions: First, FairMovies must come up with usage conditions for each party. A dealer can sell a movie to customers or send it for free to reporters and academics (these potential receivers are specified below). Customers can play a movie, back it up for fast recovery, and re-distribute it under specific conditions. Reporters may play a movie up to three times. Academics can play and edit a movie. Furthermore they can send it to other academics.

$\{(dealer, \underline{permitonlyevname}(\{(\perp_e, sell), (\perp_e, send)\}, (obj, (movA, movA))\})),$
$(customer, \underline{permitonlyevname}(\{(\perp_e, play), (\perp_e, backup), (\perp_e, send)\}, \{(obj, (movA, movA))\})),$
$(reporter, \underline{and}(\underline{permitonlyevname}(\{(\perp_e, play)\}, \{(obj, (movA, movA))\}),$
$\underline{repmax}(3, E_{fst}((play, play), \{(obj, (movA, movA))\}))))),$
$(academia, \underline{permitonlyevname}(\{(\perp_e, edit), (\perp_e, send)\}, \{(obj, (movA, movA))\})),$
$(default, \underline{permitonlyevname}(\{(\perp_e, preview)\}, \{(obj, (movA, movA))\}))\}.$

Distribution Path: FairMovies must control distribution paths. In that way, dealers are allowed to distribute to customers, reporters and film students. Academics may only distribute to other academics. Customers may distribute to their peers. Reporters are not allowed to distribute at all. In the following, we assume that the *send* event has a parameter *recv* that designates the receiver's role.

$\{(dealer, \underline{permitonlyparam}(\{(academia, academia), (reporter, reporter), (customer, customer)\},$
$recv, (send, send), \{(obj, (movA, movA))\})),$
$(academia, \underline{permitonlyparam}(\{(academia, academia)\}, recv, (send, send), \{(obj, (movA, movA))\})),$
$(customer, \underline{permitonlyparam}(\{(customer, customer)\}, recv, (send, send), \{(obj, (movA, movA))\}))\}.$

Distribution conditions: Dealers are allowed to sell copies to customers at an

increased price. Whenever doing so, the dealer must pay royalties to FairMovies.
$\{(dealer, \underline{always}(E_{fst}((sell, sell), \{(obj, (movA, movA))\}) \Rightarrow E_{fst}((pay, pay), \{(amt, (10, \top_p)),$
$(rcv, (FairMovies, FairMovies))\}))))\}$.

Policy Modification: Customers may distribute to others, but then the quality of the content must automatically be reduced by 50%. Therefore we introduce a local variable %QUALITY as proposed in §3.2. The resulting policy is then $\{(default, \underline{permitonlyparam}(\{(\bot_p, \%QUALITY/2), vq, (play, play), \{(obj, (movA, movA))\}))))\}$, where we omit a formal definition of the syntax for halving the quality.

Strengthening: We have seen several examples for strengthening by decreasing intervals. Assume now that a film student gets a movie from a dealer. The applicable subpolicy is that for Academia that results from combining the above aspects.
$\underline{and}(\underline{permitonlyevname}(\{(\bot_e, edit), (send, send)\}, \{(obj, (movA, movA))\}),$
$\underline{permitonlyparam}(\{(academia, academia)\}, recv, (send, send), \{(obj, (movA, movA))\}))$.

The student edits the movie and introduces some special effects. He then sends the modified content to his supervisor. However, he does not want his supervisor to alter anything and also wants him to not send it to anyone else. To prevent his supervisor from showing it to too many people, he also limits the number of times the movie can be played to once. Therefore he strengthens the Academic's sub-policy as follows (which he can do because he has the same role as his supervisor), and leaves the other policies as they are: $\underline{and}(\underline{permitonlyevname}(\{(\bot_e, play)\}, \{(obj, (movA, movA))\}), \underline{repmax}(1, E_{fst}((play, play), \{(obj, (movA,$
$movA))\})))$. It is easy to see that this new policy is stronger in the above sense, since every event that satisfies the new policy also satisfies the original one.

3.5 Checking Policy Entailment

Policy entailment can be checked automatically with a model checker. To this end, policies are simply translated into a dialect of LTL. This is, however, not the subject of this paper [16].

4 Related Work

Usage control has been discussed by several authors [13,3,2,7,14]. The issue of redistribution, the subject of this paper, is not discussed in these approaches, in some cases because of the centralized perspective on usage control. Related to our work are approaches to policy and rights delegation or propagation for access control. These approaches enable a user to delegate certain access rights to other users (see, among others, [6] for references). In discretionary access control, for instance, a subject may override the system's policy under specific circumstances (e.g., [4]). Our work differs in that we also take into account duties, and that we impose precise constraints on the discretionary modification of policies (strengthening).

The audit logic of Cederquist et al. [6] includes a notion of refinement of administrative policies that is also based on logical implication. It does not take into account intervals though, which means that it is difficult to express that "pay $6" is stronger than "pay $5." Furthermore, this framework does not include obligations, even though the authors state that they would be straightforward to implement.

SecPAL [1] is a powerful formal authorization language that includes constructs

for stating facts about delegation. Usage control is not considered, however. SecPAL policies are static, that is, there is no way to modify (strengthen) policies at runtime, and as a consequence, there is no support for expressing that policies may be strengthened. Given the formal semantics, however, it seems possible to include this into SecPAL.

Park et al. [12] present policies that require recipients to gain an originator's approval for the distribution of data. The distribution process itself only concerns access control requirements. This means that the policies only specify whether the objects can be accessed by others. Park et al. present several approaches how these policies can be integrated in different usage control settings.

Other work on delegation policies and models [15,2] was published. This is similar to the work discussed above. Several authors [5,11,10] discuss the problem of delegation in distributed system. However, they sketch problems rather than solutions. If solutions are discussed, these are very application specific.

5 Conclusions

In this paper, we have introduced a framework for re-distribution of data objects in distributed usage control. Essentially, given a data object that is to be distributed, the originating data provider specifies a policy for each role in a system. Upon re-distribution, these sub-policies can only be strengthened by roles that have the right to do so; this is regulated by a partial ordering of roles. These roles reflect a re-distribution hierarchy and not necessarily a hierarchy of rights and duties.

We have introduced intervals for events and parameters. For rights, the left boundary usually is the minimum value ($\bot$); for duties, the right boundary usually is the maximum value ($\top$). This means that we specify maximum rights and minimum duties; upon re-distribution, rights can be decreased and duties can be increased. Formally, we have shown that this notion of strengthening policies amounts to logical implication. For rights, such a formal notion is intuitive; for duties, it is less so, and the main contribution of this paper is a common formal framework that encompasses both rights and duties in a uniform manner.

Whether or not sub-policies are indeed strengthened must be checked upon re-distribution. Our approach allows for syntactic checking in many cases (add conjuncts, move interval boundaries in the appropriate direction). In other cases, this can be performed automatically, for instance on the grounds of model checkers, which has been done but is not described in this paper [16]. This requires of course the existence of a respective infrastructure: any distribution request must be intercepted at the potential provider's device and routed through a policy checking mechanism. This device must also take care of policy transformation of the kind of halving the quality (§3.4).

Policy transformations w.r.t. the current consumer's state deserve more attention. For instance, if an originating data provider wants the policy to be distributed only three times and the first consumer has distributed it already once, then the next consumer should be allowed to only distribute it once more. This also points at another problem: does "distribute no more than n times" mean that the overall number of copies in the system should be n or that every consumer can distribute

it n times? Similarly, if a resource usage is allowed for two hours, and the first consumer has already spent one hour using it, then every subsequent policy may have to restrict the usage to only *one* more hour. Essentially, we may need to distinguish between this kind of "global" policies and the "local" policies we used in this paper.

We have deliberately not discussed the consumer-side enforcement of policies. We are, however, of course aware that this is a crucial ingredient of any implementation of distributed usage control concepts. Moreover, the management of policies obviously is a particularly challenging task in distributed settings.

We have assumed that objects do not change when distributed. Consequently, future work is bound to the problem of rights delegation in those cases where a data item is modified (e.g., a color image is transformed into black and white), or where different objects are merged (e.g., by aggregating data in the form of statistics). Furthermore, the decision that the lattice of roles is a re-distribution rather than a rights hierarchy deserves some further considerations. In this vein, we will also have to find out if the default rule should be required to be stronger than all other sub-policies.

References

[1] M. Becker, C. Fournet, and A. Gordon. Design and semantics of a decentralized authorization language. pages 3–15, 2007.

[2] E. Bertino, C. Bettini, and P. Samarati. A temporal authorization model. In *Proc. CCS*, pages 126–135, 1994.

[3] C. Bettini, S. Jajodia, X. S. Wang, and D. Wijesekera. Provisions and obligations in policy rule management. *J. Network and System Mgmt.*, 11(3):351–372, 2003.

[4] M. Bishop. *Introduction to Computer Security.* Addison Wesley, 2005.

[5] O. Canovas and A. F. Gomez. Delegation in distributed systems: Challenges and open issues. In *Proc. 14th Intl. Workshop on Database and Expert Systems Applications*, page 499, 2003.

[6] J. Cederquist, R. Corin, M. Dekker, S. Etalle, J. den Hartog, and G. Lenzini. Audit-based compliance control. *Int. J. Inf. Secur.*, 6:133–151, 2007.

[7] M. Hilty, D. Basin, and A. Pretschner. On obligations. In *Proc. ESORICS*, Springer LNCS 3679, pages 98–117, 2005.

[8] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. A Policy Language for Distributed Usage Control. In *Proc. ESORICS*, pages 531–546, 2007.

[9] M. Hilty, A. Pretschner, C. Schaefer, and T. Walter. Enforcement for Usage Control: A System Model and a Policy Language for Distributed Usage Control. Technical Report I-ST-20, DoCoMo EuroLabs, December 2006.

[10] L. Kagal, T. Finin, and A. Joshi. Trust-based security in pervasive computing environments. *IEEE Computer*, 34(12):154–157, 2001.

[11] L. Kagal, T. Finin, and A. Joshi. A policy language for a pervasive computing environment. In *Proc. POLICY*, pages 63–74, 2003.

[12] J. Park and R. Sandhu. Originator control in usage control. In *POLICY '02: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, page 60, Washington, DC, USA, 2002. IEEE Computer Society.

[13] J. Park and R. Sandhu. The UCON ABC Usage Control Model. *ACM Transactions on Information and Systems Security*, 7:128–174, 2004.

[14] A. Pretschner, M. Hilty, and D. Basin. Distributed Usage Control. *CACM*, 49(9):39–44, September 2006.

[15] C. Ruan, V. Varadharajan, and Y. Zhang. Logic-based reasoning on delegatable authorizations. In *Proc. 13th Intl. Symp. on Foundations of Intelligent Systems*, pages 185–193, 2002.

[16] J. Rüesch. Model Checking Usage Control Policies, 2008. Master's Thesis, Department of Computer Science, ETH Zürich.