A Policy Language for Distributed Usage Control

M. Hilty[1], A. Pretschner[1], D. Basin[1], C. Schaefer[2], and T. Walter[2]

[1] Information Security, ETH Zurich, Switzerland
{hiltym,pretscha,basin}@inf.ethz.ch
[2] DoCoMo Euro-Labs, Munich, Germany
{schaefer,walter}@docomolab-euro.com

Abstract. We present the Obligation Specification Language (OSL), a policy language for distributed usage control. OSL supports the formalization of a wide range of usage control requirements. We also present translations between OSL and two rights expression languages (RELs) from the DRM area. These translations make it possible to use DRM mechanisms to enforce OSL policies. Furthermore, the translations enhance the interoperability of DRM mechanisms and allow us to apply OSL-specific monitoring and analysis tools to the RELs.

1 Introduction

Many kinds of digitally stored and processed data should only be used in restricted ways. Personal data, for example, is collected during activities such as online shopping, using loyalty cards, interaction with public administrations, and using mobile phones. To protect the privacy of the data subjects, there exist laws and regulations governing the use of personal data. Private businesses also have a keen interest in protecting their trade secrets, which turns out to be difficult, for example, when different corporations collaborate in virtual enterprises. Similarly, the creators of music, video, or other artistic works want their intellectual property rights to be respected when others use their creations.

Usage control [23, 25] is an extension of access control that covers not only who may access which data, but also how the data may or may not be used afterwards. We study usage control in the context of distributed systems with different actors who take the roles of data providers (who distribute data) and data consumers (who request and receive data). When a data provider gives a data item to a data consumer, certain conditions apply. *Provisions* are those conditions that refer to the past and are concerned with whether the data item may be released in the first place. Other conditions govern the future usage of the data, so-called *obligations* [4]. Examples of obligations include "do not distribute document D to anyone outside of the organization," "play movie M at most 5 times," and "notify the author whenever document D is modified." In this paper, we focus exclusively on obligations because provisions have been thoroughly studied in the area of access control.

There are two strategies for enforcing obligations [15]. A *control mechanism* is a consumer-side component that makes sure that obligations cannot be violated. Existing control mechanisms have been developed in the DRM area. An *observation mechanism* consists of a consumer-side signaling mechanism and a provider-side monitor. When the monitor detects the violation of an obligation, it can trigger a compensating action such as a penalty [14].

Problem Statement We address three related problems. The first is the lack of a general-purpose policy language for usage control that provides adequate support for the common structures encountered in usage control requirements. While there exist specification languages both in the area of privacy (e.g., EPAL [2] and P3P [30]) and DRM (e.g., ODRL [28] and XrML [31]), these languages are special-purpose and only cover requirements that are encountered in their respective areas. The second is that policy languages usually lack a semantics that can be used for specifying or configuring enforcement mechanisms or for checking the adherence to policies. Conversely, for many mechanisms, it is not always clear what sort of policies they can enforce. The third problem is that the different specification languages and enforcement mechanisms in usage control (particularly DRM) are often not interoperable.

Contributions We present the Obligation Specification Language (OSL), a language for expressing requirements from many application areas of usage control. This includes constraints on the duration of a usage and the kinds of permission-like statements that are often used in digital rights management. We also define a formal semantics for OSL. Together with other results [14, 15]—which include a model of enforcement mechanisms, analysis techniques for reasoning about policies and mechanisms, and an approach to monitoring whether obligations expressed in OSL are adhered to—this language builds a framework that provides tools for specifying, reasoning about, and enforcing usage control requirements.

We also show how to define translations between OSL and a REL, which we have implemented for subsets of the two most widely used rights expression languages, namely XrML and ODRL. This yields a formal semantics for these RELs and has additional benefits. First, defining the translation from a REL to OSL makes it possible to use the analysis and monitoring techniques mentioned above for the REL. Second, RELs are often used to configure DRM mechanisms. By translating OSL into a REL, we can employ the mechanisms that use this REL to enforce OSL policies. We have implemented a proof of concept for Microsoft's RMS [20], which uses XrML. Third, once the translations between OSL and several RELs are defined, we can use OSL as an intermediate language to translate between the different RELs. This is a step towards increasing the interoperability of DRM mechanisms.

Structure We analyze usage control requirements in §2 and present the syntax and semantics of OSL in §3. In §4, we show how to translate between OSL and a REL. Related work is surveyed in §5 and in §6, we conclude with an outlook on future work. An extended version of this paper is available as a technical report [15].

2 Usage Control Requirements

We have performed a requirements study in usage control based on interviews with public administrations, data protection officers, health care providers, military organizations, and numerous commercial organizations [13]. We have also carried out a detailed study on usage control requirements in mobile communication [16]. We present the distilled results of our requirements analysis with respect to obligations. Afterwards, we discuss the two modalities that play a central role in describing usage control requirements.

2.1 Obligations

Obligational formulae are conditions on the usage of data, e.g., "delete document D within 30 days" or "do not give D to anybody else." Examples for data usage are the processing, rendering, execution, management, or distribution of such data. An obligational formula becomes an *obligation* once a data consumer has received the data and committed to the conditions. We refer to this process of data reception and commitment as the *activation* of an obligational formula. An obligation has thus the form "if (activation) then (obligational formula)." In OSL, we specify obligational formulae but refer to them as obligations for the sake of simplicity. Similarly, we also talk about the activation of an obligation.

Obligations can take two different forms. *Usage restrictions* prohibit certain usages under given circumstances, and *action requirements* express mandatory actions that must be executed either unconditionally (i.e., not in direct connection with a usage) or after a specified usage has been performed.

Conditions specify circumstances under which usage restrictions or action requirements apply. They are divided into time conditions, cardinality conditions, event-defined conditions, purpose conditions, and environment conditions. Usage restrictions are statements of a form equivalent to "if *condition* then not *usage*." Examples are "document D must not be printed after more than 20 days" and "movie M may only be played once." Action requirements are statements of a form equivalent to "if *condition* then *action*." Examples are "delete data D 30 days after reception" and "notify the data owner after each usage of data D." Note that action requirements and usage restrictions may look similar. For example, "notify the data owner before each usage of data D" is a usage restriction because it prohibits using D if the notification has not been not sent before. We briefly discuss each type of condition below. In the examples given, the respective conditions are typeset in *italics*.

Time Conditions include, for example, "file F must be deleted *within 7 days*", "F must *never* be distributed", or "movie stream M must *not* be viewed *for more than a total of 5 hours*." We have not encountered examples like "action A must *eventually* be executed" where no time limit is given for the execution of an action. Instead, one sets a time limit like "action A must be executed *within 2 years*." Cardinality Conditions refer to the number of occurrences of given events. Examples include "movie M may *only* be played *once*" or "view trailer D *at most twice* before the movie M is paid." Event-Defined Conditions

define situations in terms of the occurrence of events. An example of an event-defined condition is "*if the data provider revokes document D*, the document must not be used anymore," or "document D must not be further distributed *until the author officially releases D*." PURPOSE CONDITIONS refer to the purpose of use. For example, objects that are labelled "for *personal use only*" must not be used in a business context. ENVIRONMENT CONDITIONS relate to the internal and external environment of the data consumer. This includes the adherence to technical or organizational standards (e.g., Common Criteria or the Sarbanes-Oxley Act) as well as aspects of the physical environment such as the data consumer's geographical location. An example is "document D may only be opened *within Europe*."

Conditions can be combined to describe complex circumstances. For example, the obligation "movie M may be viewed at most 3 times and only within 30 days" combines a time condition and a cardinality condition. Moreover, usage restrictions and action requirements can be combined to form complex policy statements. For example, the following obligational formula combines an action requirement with a time condition and a usage restriction with a purpose condition: "document D needs to be deleted within 7 days and must not be shown in public."

2.2 Modalities

There are different ways of specifying policies. One approach, which is often employed in informal regulations (e.g., privacy regulations) and system specifications, is to explicitly define *requirements* on the system execution. This approach uses a "must" modality in the sense that every requirement must be satisfied. In contrast, a REL specifies exclusive *rights* to execute given actions under specific conditions. Such rights specify what may happen with data and therefore use a "may" modality. An exclusive right to perform a usage implies that all other usages are forbidden, but the prohibited usages are not specified explicitly.

Many usage control requirements are not equally easy to express in both modalities. In the DRM area, where typically only a few usages are allowed, rights are often easier to specify. This is particularly the case if the set of prohibited usages is large or even unbounded (e.g., if usages are parameterized). In the privacy area, where the laws and regulations often express explicit prohibitions, it is rather the other way around. Furthermore, the requirements document for ODRL version 2.0 [21] states that expressing prohibitions can also be desirable in the DRM area. As a consequence, we support both modalities in OSL. The "must" modality is inherited from temporal logic, and the "may" modality has been included via dedicated permission operators.

3 The Obligation Specification Language (OSL)

We introduce the syntax and semantics of OSL. We formalize both in Z, a formal language based on typed set theory and first-order logic with equality. We have

chosen Z because of its rich notation, which we explain as it is encountered. We have also given a more user-friendly syntax to OSL [15], which we do not present in this paper due to space restrictions. The current version of OSL supports all usage control requirements identified above, except environment conditions.

3.1 Events and Traces

The semantics of our language is defined over traces with discrete time steps. At each time step, a set of events can occur. An event corresponds to the execution of an action and we use these two terms interchangeably. We formalize different aspects of events and traces.

Event Classes and Parameters Each *Event* has a name and parameters, specifying additional details about the event. For example, a usage event can indicate on which data item it is performed or by which device. Parameters are represented using a partial function ($\pfun$) from parameter names to parameter values. We often describe parameters by their function graph. An example of an event in this syntax is $(play, \{(object, m)\})$, where *play* is the event name and the parameter with name *object* has value m (i.e., the object m is played).

Each event belongs to an event *class*. Possible event classes include *usage* and *other*, the latter standing for all non-usage events, e.g., payments or notifications. This distinction enables us to prohibit all usages on a data item while still allowing other events such as payments. The definition of events in Z is shown below. *EventName*, *ParamName*, and *ParamValue* define basic types for event names, parameter names, and parameter values, respectively. In Z, basic types are defined by listing their names in square brackets.

$$[EventName, ParamName, ParamValue] \qquad Params : ParamName \pfun ParamValue$$
$$EventClass == \{usage, other\} \qquad Event == EventName \times Params$$
$$getclass : EventName \to EventClass$$

Indexed Events An important usage control requirement is the restriction of the accumulated usage time. To cater for usages that last a specified time, we introduce *indexed events*. We assume that at each step of a trace, there is an indexed event for each usage that is currently executed. The start of the usage is represented by an indexed event with the index *start*, and all following indexed events have the index *ongoing*. For example, if the time step is 1 minute and a user plays a movie m for 3 minutes, the resulting indexed events occurring in the trace are $((play, \{(object, m)\}), start)$, $((play, \{(object, m)\}), ongoing)$, and $((play, \{(object, m)\}), ongoing)$. In OSL, we can explicitly refer to the start of an event or to all parts of it (cf. §3.2). *IndEvent* defines indexed events and *Trace* defines traces. The formalization of the above assumption is omitted due to space limitations but can be found in the technical report [15].

$$IndEvent == Event \times \{start, ongoing\} \qquad Trace : \mathbb{N} \to \mathbb{P}\, IndEvent$$

Event Declarations So far, we have not defined what events can occur in a concrete system. To this end, we introduce event declarations. An event

declaration contains the event name, the event class, and a partial function that defines the name and possible values of each parameter. Note that such an event declaration is purely syntactic and says nothing about the meaning of an event, i.e., which event in a real system it describes. The specification of dedicated ontologies is outside the scope of this paper.

$$EventDecl == EventName \times EventClass \times (ParamName \pfun \mathbb{P}\, ParamValue)$$

3.2 Syntax

An OSL policy consists of a set of event declarations and a set of obligational formulae. Each obligational formula consists of the data consumer's name and a logical expression. *SubID* is the set of possible names of data consumers.

$$OSLPolicy == \mathbb{P}\, EventDecl \times \mathbb{P}\, OblFormula$$
$$OblFormula == SubID \times \Phi$$

Φ defines the syntax of the logical expressions contained in obligational formulae. $E_{fst}(e)$ refers to the start of an event e and $E_{all}(e)$ to ongoing events as well (cf. §3.1). Z allows EBNF-style definitions as used below.

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

We define an additional restriction on the policy syntax (omitted here): we demand that all events that are mentioned in a policy are compliant with the event declaration, i.e., they may only contain parameters that are declared and corresponding values. Fewer parameters are allowed in a policy, because of the implicit universal quantification over unspecified parameters (cf. §3.4).

3.3 Informal Semantics

We first informally describe the semantics of OSL's operators. They are classified into propositional operators, temporal operators, cardinality operators, and permit operators. An example for a complete OSL policy is given in Section 4.2.

Propositional Operators The operators $\underline{not}$, $\underline{and}$, $\underline{or}$, and $\underline{implies}$ have the same semantics as their propositional counterparts $\neg$, $\wedge$, $\vee$, and $\Rightarrow$.

Temporal Operators The $\underline{until}$ operator corresponds to the *weak until* operator from LTL [24]. We use the weak version of the until operator because it is better suited for expressing usage control requirements (cf. §2.1). We generalize the *next* operator of LTL to $\underline{after}$, which takes a natural number n as input and refers to the time after n time steps. With $\underline{after}$, we can express concepts like $\underline{during}$ (something must hold constantly during a given time interval) and $\underline{within}$ (something must hold at least once during a given time interval).

Cardinality Operators Cardinality operators restrict the number of occurrences of a specific event or the accumulated duration of an event. The *repuntil* operator limits the maximum number of times an event may occur until another event occurs. For example,

$$\underline{repuntil}(3, E_{fst}(play, \{(object, m)\}),\\ E_{fst}((pay, \{(currency, USD), (amount, 10), (recipient, r)\})))$$

states that the movie m must not be played more than 3 times until a payment of \$10 is made to r. With *repuntil*, we can also define *repmax*, which is syntactic sugar for defining the maximum number of times an event may occur in the unlimited future. For example,

$$\underline{repmax}(5, E_{all}(play, \{(object, s)\}))$$

requires that the movie stream s must not be played for more than 5 time steps. This example shows that by using E_{all} instead of E_{fst}, the cardinality operators can be used to limit accumulated usage time. In the semantics definition below, we restrict the cardinality operators to arguments of these two forms. The reason for this restriction is that we allow multiple similar events to occur within one time step. For example, the movie m may be played on two devices simultaneously, which counts twice in the *repuntil* example above.

Permit Operators In OSL, 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: these operators allow one to specify that out of a given set of usages events, only selected usage events are allowed. The operator *permitonlyevname* defines the names of the usage events that are exclusively allowed with a set of given parameters. For example, the expression

$$\underline{permitonlyevname}(\{play, print\}, \{(object, oid)\})$$

states that the only usages permitted on the object *oid* are *play* and *print*. It does not say anything about non-usage events or events with different parameters (e.g., if a usage is applied to a different data object). Similarly, *permitonlyparam* only allows certain values for a given parameter of an event. It prohibits all other values for this parameter. For example, the expression

$$\underline{permitonlyparam}(\{s_1, s_2\}, recipient, send, \{(object, doc)\})$$

specifies that out of all *send* events with *doc* as the "object" parameter, only those where the "recipient" parameter has the value s_1 or s_2 are allowed. In other words, *doc* may only be sent to s_1 or s_2. The first argument of *permitonlyparam* is the set of allowed parameter values, the second argument is the name of the parameter whose values should be restricted, and the third and fourth argument define an underspecified event.

3.4 Formal Semantics

When specifying events in obligations, we implicitly quantify over unmentioned parameters. For example, if an obligation prohibits event $(play, \{(object, objB)\})$,

then the event $(play, \{(object, objB), (device, dev123)\})$ is prohibited as well. To specify this, we define the relation *refinesEv*, which checks whether one event e_2 refines another event e_1. This is the case iff both have the same event name and all parameters of e_1 have the same value in e_2. e_2 can also have additional parameters. With the help of *refinesEv*, we can also define the satisfaction relation for event expressions, $\models_e$. This defines whether an indexed event corresponds to an expression of the form $E_{fst}(e)$ or $E_{all}(e)$, where e is an event. The semantics of a logical expression $\varphi : \Phi$ is defined by the binary relation $\models_f$.

The relations *refinesEv*, $\models_e$, and $\models_f$ are defined in Figure 1. We specify them using an *axiomatic definition* in Z: the upper part of an axiomatic definition contains the signature and in the lower part, the properties of the functions and relations are specified. In Z, relations are declared with $\leftrightarrow$. We also use the following Z notation: $e_1.2$ refers to the second component of e_1, # denotes the size of a set, and dom refers to the domain of a function.

A policy is satisfied by a trace iff all obligations specified in the policy are satisfied by the trace. The definition of obligation satisfaction builds on the above semantics but requires a system model that includes activations of obligations (cf. §2.1). Such a system model is presented in [15].

4 Language Translations

In this section, we present translations between OSL and the most widely used rights expression languages, ODRL and XrML. We use the term *license* for policies expressed in a REL. We start by explaining our reasons for translating between OSL and rights expression languages, show how to define such translations, and highlight several key issues using the example of the translations between OSL and a subset of ODRL. We have published the formal specification of these translations in a technical report [15]. We have also implemented these translations in software, both for the ODRL subset mentioned above and for a comparable subset of XrML.

4.1 Purpose

There are several reasons for defining the translations between OSL and different RELs. The first reason is the need for enforcing OSL policies. Current enforcement mechanisms are almost exclusively from the DRM area and use licenses or rights objects written in a REL. Thus, the ability to translate OSL policies into rights objects makes it possible to re-use such mechanisms to enforce OSL policies. As OSL is not limited to DRM, this opens the door to automatically enforcing non-DRM policies (e.g., privacy policies) with DRM mechanisms. By defining translations from privacy policy languages into OSL (which is future work), we will be able to close the gap between the areas of privacy and DRM, which are seen as antipodes by many people.

The second reason for providing translation schemes is that the translation from a REL to OSL gives a formal semantic to the REL. While the formal

$$
\begin{array}{|l}
\_\ \mathit{refinesEv}\ \_ : \mathit{Event} \leftrightarrow \mathit{Event} \\
\hline
\forall\, e_1, e_2 : \mathit{Event} \bullet\ e_2\ \mathit{refinesEv}\ e_1 \Leftrightarrow e_1.1 = e_2.1 \land e_1.2 \subseteq e_2.2
\end{array}
$$

$$
\begin{array}{|l}
\_\ \models_e\ \_ : \mathit{IndEvent} \leftrightarrow \Phi \\
\hline
\forall\, ie : \mathit{IndEvent};\ \varphi : \Phi \bullet\ ie \models_e \varphi \Leftrightarrow \\
\quad \exists\, e : \mathit{Event} \bullet\ ie.1\ \mathit{refinesEv}\ e \\
\qquad \land\ ((\varphi = E_{fst}(e) \land ie.2 = \mathit{start}) \lor \varphi = E_{all}(e))
\end{array}
$$

$$
\begin{array}{|l}
\_\ \models_f\ \_ : (\mathit{Trace} \times \mathbb{N}) \leftrightarrow \Phi \\
\hline
\forall\, t : \mathit{Trace};\ n : \mathbb{N};\ \varphi : \Phi \bullet (t, n) \models_f \varphi \Leftrightarrow \\
\quad \varphi = \underline{\mathit{true}} \\
\quad \lor\ \exists\, e : \mathit{Event};\ ie : \mathit{IndEvent} \bullet (\varphi = E_{fst}(e) \lor \varphi = E_{all}(e)) \land ie \in t(n) \land ie \models_e \varphi \\
\quad \lor\ \exists\, \psi : \Phi \bullet \varphi = \underline{\mathit{not}}(\psi) \land \neg\ ((t, n) \models_f \psi) \\
\quad \lor\ \exists\, \psi, \chi : \Phi \bullet \varphi = \underline{\mathit{or}}(\psi, \chi) \land ((t, n) \models_f \psi \lor (t, n) \models_f \chi) \\
\quad \lor\ \exists\, \psi, \chi : \Phi \bullet \varphi = \underline{\mathit{until}}(\psi, \chi) \\
\qquad \land\ (\exists\, u : \mathbb{N} \mid n \le u \bullet \big((t, u) \models_f \chi \land (\forall\, v : \mathbb{N} \mid n \le v < u \bullet (t, v) \models_f \psi)\big) \\
\qquad\quad \lor\ (\forall\, v : \mathbb{N} \mid n \le v \bullet (t, v) \models_f \psi)) \\
\quad \lor\ \exists\, i : \mathbb{N};\ \psi : \Phi \bullet \varphi = \underline{\mathit{after}}(i, \psi) \land (t, n + i) \models_f \psi \\
\quad \lor\ \exists\, l : \mathbb{N};\ \psi, \chi : \Phi;\ e : \mathit{Event} \bullet \\
\qquad \varphi = \underline{\mathit{repuntil}}(l, \psi, \chi) \land (\psi = E_{fst}(e) \lor \psi = E_{all}(e)) \\
\qquad \land\ \big((\exists\, u : \mathbb{N} \mid n \le u \bullet (t, u) \models_f \chi \land (\forall\, v : \mathbb{N} \mid n \le v < u \bullet \neg((t, v) \models_f \chi)) \\
\qquad\quad \land\ (\sum_{j=0}^{u} \#\{ie : \mathit{IndEvent} \mid ie \in t(n + j) \land ie \models_e \psi\}) \le l) \\
\qquad \lor\ (\sum_{j=0}^{\infty} \#\{ie : \mathit{IndEvent} \mid ie \in t(n + j) \land ie \models_e \psi\}) \le l\big) \\
\quad \lor\ \exists\, ex : \mathbb{P}\, \mathit{EventName};\ ps : \mathit{Params} \bullet \varphi = \underline{\mathit{permitonlyevname}}(ex, ps) \\
\qquad \land\ \forall\, en : \mathit{EventName} \mid \mathit{getclass}(en) = \mathit{usage} \land en \notin ex \bullet \\
\qquad\quad (t, n) \models_f \underline{\mathit{always}}(\underline{\mathit{not}}(E_{all}((en, ps)))) \\
\quad \lor\ \exists\, ex : \mathbb{P}\, \mathit{ParamValue};\ pn : \mathit{ParamName};\ en : \mathit{EventName};\ ps : \mathit{Params} \bullet \\
\qquad \varphi = \underline{\mathit{permitonlyparam}}(ex, pn, en, ps) \land pn \notin \mathsf{dom}\, ps \\
\qquad \land\ \forall\, pv : \mathit{ParamValue} \mid pv \notin ex \bullet \\
\qquad\quad (t, n) \models_f \underline{\mathit{always}}(\underline{\mathit{not}}(E_{all}((en, ps \cup \{(pn, pv)\})))) \\
\quad \lor\ \exists\, \psi, \chi : \Phi \bullet \varphi = \underline{\mathit{and}}(\psi, \chi) \land (t, n) \models_f \underline{\mathit{not}}(\underline{\mathit{or}}(\underline{\mathit{not}}(\psi), \underline{\mathit{not}}(\chi))) \\
\quad \lor\ \exists\, \psi, \chi : \Phi \bullet \varphi = \underline{\mathit{implies}}(\psi, \chi) \land (t, n) \models_f \underline{\mathit{or}}(\underline{\mathit{not}}(\psi), \chi) \\
\quad \lor\ \exists\, \psi : \Phi \bullet \varphi = \underline{\mathit{always}}(\psi) \land (t, n) \models_f \underline{\mathit{until}}(\psi, \underline{\mathit{false}}) \\
\quad \lor\ \exists\, i : \mathbb{N};\ \psi : \Phi \bullet \varphi = \underline{\mathit{within}}(i, \psi) \land (t, n) \models_f \bigvee_{i=0}^{n-1} \underline{\mathit{after}}(i, \varphi) \\
\quad \lor\ \exists\, i : \mathbb{N};\ \psi : \Phi \bullet \varphi = \underline{\mathit{during}}(i, \psi) \land (t, n) \models_f \bigwedge_{i=0}^{n-1} \underline{\mathit{after}}(i, \varphi) \\
\quad \lor\ \exists\, l : \mathbb{N};\ \psi : \Phi \bullet \varphi = \underline{\mathit{repmax}}(l, \psi) \land (t, n) \models_f \underline{\mathit{repuntil}}(l, \psi, \underline{\mathit{false}})
\end{array}
$$

Fig. 1. Semantics of OSL

semantics for some parts of ODRL and XrML have been defined in earlier work, the translations that we present immediately provide a formal semantics to parts of ODRL, including parts for which this has not yet been done (cf. §5). It is important to note that the formal semantics of OSL makes it possible to perform logical analysis on policies [15] and to check for adherence to obligations at runtime [14]. So we gain both (1) the possibility to use control mechanisms from the DRM area to enforce OSL policies as described above and (2) the possibility to analyze DRM licenses and to use our observation mechanisms to enforce them.

Finally, there is a need for more interoperability among DRM technologies in order to increase user acceptance. One problem in this area is that different mechanisms use different languages for describing their licenses [18, 10]. There are two possible solutions to this problem. One solution is to standardize a language for describing licenses and use it for all developed mechanisms. However, such a solution is currently not on the horizon. The other solution is to translate between the different RELs. Some attempts have been made to directly translate between RELs (e.g., [8]). In contrast, by defining translations from OSL to the different RELs and vice versa, we enable the use of OSL as an intermediate language for translating between different RELs. This approach scales up well and is not only limited to XrML and ODRL, but also other RELs like PDRL [1] or Octopus [19]. However, we have not yet defined the translations for more languages; this remains as future work.

4.2 Specification and Implementation

We have implemented the translations between OSL and a subset of ODRL as well as between OSL and a subset of XrML. The XrML subset used is comparable to the ODRL subset, which is described below. As the translations for ODRL and XrML are similar, we focus here on ODRL and use this example to point out the strengths and weaknesses of our approach. More details about translating ODRL into OSL are published in a technical report [15].

A Brief Introduction to ODRL ODRL [28] is an XML-based language for describing the terms and conditions of using intellectual property in digital form. We give a short, incomplete overview of ODRL. ODRL subjects are intellectual property *rights holders* and *end users*. Data objects are called *assets*. ODRL expresses *offers*, which are proposals from rights holders for specific rights on their assets, and *agreements*, which result when two parties commit to a set of rights on an asset. Agreements in ODRL can be compared to obligations, while offers are outside of the scope of this paper.

A *permission* is the right to perform certain *activities* with an asset and can be accompanied by *constraints* and *requirements*. Examples of activities are *play*, *print*, *display*, and *execute*. In our model, these activities correspond to usages. Constraints express conditions that the end user must satisfy to be allowed to perform the corresponding activity, and requirements specify additional actions that the end user must execute, such as payments. An example for an ODRL license is provided at the end of this section.

The ODRLc Subset of ODRL We have defined the translations for a subset of ODRL to keep the definition of the translation reasonably sized and because ODRL contains concepts that are not within the scope of OSL. The subset we consider, ODRLc ("ODRL compact"), is very close to the REL used by the Open Mobile Alliance (OMA) [22] and therefore of practical relevance. We have introduced a few structural simplifications with regard to the OMA REL that do not limit the expressiveness of the language. Also, we have not included those elements of the OMA REL that are not part of ODRL. However, we also support a few ODRL concepts that are not included in the OMA REL. For example, we have included a reduced set of payment requirements in ODRLc to illustrate that such requirements can easily be expressed in OSL. We also support device constraints, which restrict the set of devices that are allowed to perform a usage. A more detailed description of ODRLc can be found in [15].

Translating ODRLc into OSL The translation from ODRLc to OSL is defined for all ODRLc licenses because OSL is strictly more expressive than ODRLc. Since both OSL policies and ODRL licenses are tree structured, we define the translation top-down on the ODRLc tree. Because an ODRL license specifies rights, we use a $\underline{permitonlyevname}$ expression to prohibit all usages not explicitly permitted in the license. In ODRL, all specified constraints and requirements must be simultaneously satisfied and therefore form a logical conjunction. In OSL, we create a separate obligation for each of them. Since all obligations inside an OSL policy are implicitly conjoined, the conjunction of the constraints and requirements naturally follows.

Translating OSL into ODRLc Because OSL is strictly more expressive than ODRLc as mentioned above, only a subset of OSL can be translated to ODRLc. Identifying this subset is the difficult part of defining the translation. We take a pragmatic approach to this by employing pattern matching over syntax. For example, all formulae of the form $\big(sid, \underline{repmax}(n, E_{fst}(ue))\big)$, where sid is a subject ID, $n \in \mathbb{N}$, and ue is a usage event, are translated into a <count> constraint in ODRL, which expresses a cardinality condition. The problem is that $\big(subjA, \underline{and}(\underline{repmax}(n, E_{fst}((backup, \{(object, mov)\}))), \underline{true})\big)$ is semantically a cardinality condition as well, but not a syntactic instance of the above pattern. Because syntactic pattern matching requires obligations to be in an implicitly defined canonical form, the translation for this obligation is therefore undefined.

This limitation on the translation could be lifted by extending it to semantically equivalent representations. This would, however, involve computationally expensive deductive reasoning. In particular, since LTL can be completely embedded into OSL and checking the semantic equivalence of two LTL formulae is PSPACE-complete [29], checking the semantic equivalence of two OSL formulae is PSPACE-hard.

Example We now show an example of a translation from ODRLc to OSL. The corresponding ODRLc license is shown below. This license states that *Alice* may play the movie *mov* for at most 5 hours, and only on player *pl*. Furthermore, *Alice* may create at most one backup of the movie. No usage other than *play* and *backup* is allowed.

```
<o-ex:rights>
  <o-ex:agreement>
    <o-ex:asset>
      <o-ex:context><o-dd:uid>mov</o-dd:uid></o-ex:context>
    </o-ex:asset>
    <o-ex:permission>
      <o-dd:play>
        <o-ex:constraint><o-dd:accumulated>P5h</o-dd:accumulated></o-ex:constraint>
        <o-ex:constraint><o-dd:hardware>
          <o-ex:context><o-dd:uid>pl</o-dd:uid></o-ex:context>
        </o-dd:hardware><o-ex:constraint>
        <o-ex:constraint><o-dd:individual>Alice</o-dd:individual></o-ex:constraint>
      </o-dd:play>
      <o-dd:backup>
        <o-ex:constraint><o-dd:count>1</o-dd:count></o-ex:constraint>
        <o-ex:constraint><o-dd:individual>Alice</o-dd:individual></o-ex:constraint>
      </o-dd:backup>
    </o-ex:permission>
  </o-ex:agreement>
</o-ex:rights>
```

This ODRL license is translated into the OSL policy shown below. The first obligation prohibits all usages except *play* and *backup*. The second one corresponds to the <`accumulated`> constraint on the *play* action, the third one to the <`hardware`> constraint on the *play* action, and the fourth obligation to the <`count`> constraint on the *backup* action. In the OSL policy shown below, we assume that the time step in a trace is set to 1 hour.

$$
\begin{array}{l}
(\{ \\
\quad (play, usage, \{(object, ObjID), (device, DevID)\}), \\
\quad (backup, usage, \{(object, ObjID), (device, DevID)\}), \\
\}, \{ \\
\quad (Alice, \underline{permitonlyevname}(\{play, backup\}, \{(object, mov)\})), \\
\quad (Alice, \underline{repmax}(5, E_{all}((play, \{(object, mov)\})))), \\
\quad (Alice, \underline{permitonlyparam}(\{pl\}, device, play, \{(object, mov)\})), \\
\quad (Alice, \underline{repmax}(1, E_{fst}((backup, \{(object, mov)\})))) \\
\})
\end{array}
$$

Summary of the Results While the translation from ODRLc to OSL is total, the translation in the other direction is only partial. This is partly because OSL is more expressive than ODRLc and partly because the translation is defined by syntactic pattern matching. Using deductive reasoning to compute semantic equivalence classes would allow us to extend the translation, but this is computationally expensive. The same issues also apply to the translations that we have implemented for XrML.

The translation from ODRLc to OSL yields a formal semantics for a significant subset of ODRL. Within the limitations mentioned above, the translations from OSL to ODRL and XrML enable us to issue licenses for existing DRM mechanisms based on OSL policies. We have implemented both translations in Java. On the basis of the OSL-XrML translator, we have implemented a proof of concept that automatically creates licenses for Microsoft's RMS [20] from OSL policies.

The translations between OSL and ODRL/XrML can be composed to translate between ODRL and XrML, using OSL as an intermediate language. We have done this using the above mentioned implementations. Of course, this only works for requirements that can be expressed in both RELs. For the subsets we have defined for ODRL and XrML, this is not a problem because they are similar in their expressivity. But generally, this issue must be taken into account.

There is one additional point worth noting: XrML contains so-called *stateful conditions* that use an external piece of data to store the effect of previous usages on the license. For example, cardinality conditions may be specified with the help of an external counter that has to be checked each time a usage is attempted. This approach mixes the specification of a requirement (how many times a usage may be performed in total) with the implementation of its enforcement (where the counter is located and when it must be decremented). Because OSL is a pure specification language, we have excluded state references from the XrML subset we use in the translations and only specify the initial values instead.

5 Related Work

Some specification languages for usage control requirements have been developed in the area of privacy protection. P3P [30] is a language for stating the privacy practices of websites. It is tailored to this domain and not extensible. EPAL [2] is a more flexible language for privacy policies that adds the purpose of use to the access decision. It also allows for obligations, but does not treat them in detail. We have already mentioned XrML [31] and ODRL [28] as the most prominent policy languages in DRM.

There have been previous attempts to give a formal semantics to ODRL and XrML. Pucella and Weissman [27] give a formal semantics to a subset of ODRL by a translation into many-sorted first-order logic. They treat temporal aspects rather rudimentarily and the duration of events is not considered. Furthermore, their semantics is not suited for monitoring the adherence to policies at runtime. Holzer et al. [17] present a semantics based on automata. They present automata for different conditions but do not define how they compose for multiple rights and conditions. For example, the automaton presented on the lower half of page 7 cannot cope with events other than *display*. García et al. [9] formalize ODRL policies using the OWL-based framework IPROnto [7], which is an ontology for DRM. Like other approaches, this formalization cannot, in its current state, be used for checking the adherence to complex policies at runtime and does not consider the duration of usage. We are only aware of one formal semantics that has been defined for XrML. Halpern and Weissman [12] have chosen an approach similar to the one for ODRL mentioned above [27], with similar strengths and weaknesses.

Gunter et al. [11] present a semantics for DRM licenses that is based on sequences of events, but they do not apply this semantics to any existing REL. Chong et al. [5] present LicenseScript, a language for expressing DRM licenses. The main difference between LicenseScript and OSL is that OSL is a language

for specifying policies (i.e., we describe which executions are allowed), whereas LicenseScript can express complicated licenses (e.g., involving cardinalities) only by including instructions that concern the enforcement of the licenses (i.e., decreasing counters). Therefore, licenses expressed in LicenseScript do not explicitly tell the data consumer what is allowed and what is not allowed. Pucella and Weissman [26] present a logic for reasoning about digital rights based on temporal logic. The main focus of their work is to define whether a set of policies permits or obligates certain actions. Only one action is permitted per time step and neither temporal conditions nor cardinality conditions can be expressed conveniently. Barth et al. [3] have developed a privacy policy language that is based on LTL and has data subjects as a central concept. Their language is only concerned with the distribution of data.

UCON [23, 32] extends access control with the concepts of decision continuity and attribute mutability. In UCON, an access can last for some duration with multiple related and subsequent actions, for example, performing calculations on data on a server. Access decisions can be made before or during the access (decision continuity) and subject or object attributes can change during an access (attribute mutability). While in OSL, we specify what the data consumer is allowed to do (for example, playing a movie for at most 20 minutes), UCON can be used to specify how a mechanism counts the elapsed time and compares it to the maximal allowed value. In this regard, UCON is complementary to our approach because it can be used for implementing mechanisms on different devices. What UCON cannot cover, however, are action requirements. For example, UCON-based mechanisms cannot enforce that a piece of data is deleted after 30 days, independently of whether it is used during that time. Also, we can use OSL to specify that the above movie may be played for maximal 20 minutes even if different players are involved, which cannot be expressed in UCON.

Cooper and Montague [6] discuss differences between ODRL and XrML and suggest that the usage of profiles that reduce a language to the part that can be translated into the other language is a good way to proceed. Other interoperability-related work is surveyed in §4.1.

6 Conclusions

We have presented OSL, a rich language that can specify policies from many different application areas. We have determined the usage control requirements that OSL supports in dedicated requirements studies. The semantics of OSL is based on temporal logic, which enables the use of different analysis methods and runtime monitoring techniques, as has been shown in related work [14, 15]. The translations that we have presented allow us to enforce, in part, policies specified in OSL using existing enforcement mechanisms from the DRM area. Our goal is to be able to flexibly employ different mechanisms for enforcing OSL policies, depending on which mechanism is applicable to a given requirement. The proof of concept that we have implemented for RMS is just a first step in this direction. The formal semantics we get for the RELs goes beyond what has

been previously defined for XrML and ODRL and we have also provided a step towards more interoperability in DRM.

We conclude by discussing current limitations of OSL and suggest directions for future work. From the types of conditions we have identified in our requirements study, OSL cannot fully cover environment conditions. Some environment conditions can be expressed with the help of event parameters but others would require the introduction of subject attributes into our model, which is future work. The fact that OSL uses abstract events instead of concrete system events comes at a cost: the need to define the semantics of these events. One problem here is that the mapping from abstract events like "play" or "display" to concrete events of a system is usually done by the mechanism vendors and is not transparent. This complicates the selection of suitable mechanisms for a given requirement. The definition of dedicated ontologies for events is an area for future work. It is not entirely clear how to assign a semantics to events and, while it seems desirable to have device-independent policies, the device-specific semantics cannot be ignored. Additional areas for future work are addressing rights propagation, defining a translation based on semantic equivalence classes of OSL policies, defining translation schemes for additional RELs, and the implementation of these translations. Last but not least, the ideas presented in this paper should be evaluated in case studies.

References

1. Adobe. Portable Document Rights Language (PDRL) Specification, 2005. `www.adobe.com/devnet/livecycle/policyserver/articles/pdrl.pdf`.
2. M. Backes, B. Pfitzmann, and M. Schunter. A toolkit for managing enterprise privacy policies. In *8th European Symposium on Research in Computer Security*, LNCS 2808, pages 162–180. 2003.
3. A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *Proc. of the 2006 IEEE Symposium on Security and Privacy*, pages 184–198, 2006.
4. C. Bettini, S. Jajodia, X. S. Wang, and D. Wijesekera. Provisions and obligations in policy rule management. *Journal of Network and System Management*, 11(3):351–372, 2003.
5. C. N. Chong, R. J. Corin, J. M. Doumen, S. Etalle, P. H. Hartel, Y. W. Law, and A. Tokmakoff. Licensescript: A logical language for digital rights management. *Annals of telecommunications special issue on Network and Information systems security*, 61(3-4):284–331, March 2006.
6. B. Cooper and P. Montague. Translation of rights expressions. In *Proc. the 4th Australasian Information Security Workshop*, pages 137–144, 2005.
7. J. Delgado, I. Gallego, S. Llorente, and R. Garcá. IPROnto: An Ontology for Digital Rights Management. In *Proc. Jurix 2003: The Sixteenth Annual Conference on Legal Knowledge and Information Systems*, pages 111–120, 2003.
8. J. Delgado, J. Prados, and E. Rodriguez. A new Approach for Interoperability between ODRL and MPEG-21 REL. In *Proc. 2nd Intl. ODRL Workshop*, 2005.
9. R. García, R. Gil, I. Gallego, and J. Delgado. Formalising ODRL Semantics using Web Ontologies. In *Proc. 2nd Intl. ODRL Workshop*, pages 1–10, 2005.

10. D. Geer. Digital Rights Technology Sparks Interoperability Concerns. *IEEE Computer*, 37:20–22, December 2004.
11. C. A. Gunter, S. T. Weeks, and A. K. Wright. Models and languages for digital rights. In *Proc. 34th Annual Hawaii Intl. Conference on System Sciences*, 2001.
12. J. Halpern and V. Weissman. A Formal Foundation for XrML. In *Proc. 17th IEEE Computer Security Foundations Workshop*, pages 251–265, 2004.
13. M. Hilty, A. Pretschner, and F. Akeret. Anforderungen für verteilte Nutzungskontrolle. Technical report, Siemens Schweiz AG, November 2005.
14. M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. Monitors for usage control. In *Proc. Joint iTrust and PST Conferences on Privacy, Trust Management and Security*, 2007.
15. M. Hilty, A. Pretschner, T. Walter, and C. Schaefer. A system model and an obligation lanugage for distributed usage control. Technical Report I-ST-20, DoCoMo Euro-Labs, 2006.
16. M. Hilty, A. Pretschner, T. Walter, and C. Schaefer. Usage control requirements in mobile and ubiquitous computing applications. In *Proc. International Conference on Systems and Networks Communication*, page 27, 2006.
17. M. Holzer, S. Katzenbeisser, and C. Schallhart. Towards a Formal Semantics for ODRL. In *Proc. 1st International workshop on ODRL*, pages 137–148, 2004.
18. R. H. Koenen, J. Lacy, M. MacKay, and S. Mitchell. The long march to interoperable digital rights management. *Proceedings of the IEEE*, 92(6):883–897, 2004.
19. Marlin Developer Community. The Role of Octopus in Marlin, 2006. `www.marlin-community.com/images/wp/RoleofOctopusinMarlin.pdf`.
20. Microsoft Corporation. Technical overview of windows rights management services for windows server 2003, April 2005. Available at `http://www.microsoft.com/windowsserver2003/techinfo/overview/rmenterprisewp.mspx`.
21. Open Mobile Alliance. DRM Architecture, March 2006. Available at `www.openmobilealliance.org/release_program/drm_v2_0.html`.
22. Open Mobile Alliance. DRM Rights Expression Language, March 2006. Available at `www.openmobilealliance.org/release_program/drm_v2_0.html`.
23. J. Park and R. Sandhu. The UCON ABC Usage Control Model. *ACM Transactions on Information and Systems Security*, 7:128–174, 2004.
24. A. Pnueli. The temporal semantics of concurrent programs. In *Proc. International Sympoisum on Semantics of Concurrent Computation*, pages 1–20, 1979.
25. A. Pretschner, M. Hilty, and D. Basin. Distributed Usage Control. *CACM*, September 2006.
26. R. Pucella and V. Weissman. A logic for reasoning about digital rights. In *Proc. 15th IEEE Computer Security Foundations Workshop*, page 282, 2002.
27. R. Pucella and V. Weissman. A Formal Foundation for ODRL. In *Proc. Workshop on Issues in the Theory of Security*, 2004.
28. R. Iannella (Ed.). Open Digital Rights Language - Version 1.1, August 2002. `odrl.net/1.1/ODRL-11.pdf`.
29. A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.
30. W3C. The Platform for Privacy Preferences 1.1 (P3P1.1) Specification, 2005.
31. X. Wang, G. Lao, T. DeMartini, H. Reddy, M. Nguyen, and E. Valenzuela. XrML – eXtensible rights Markup Language. In *ACM workshop on XML security*, pages 71–79, 2002.
32. X. Zhang, J. Park, F. Parisi-Presicce, and R. Sandhu. A logical specification for usage control. In *Proc. 9th ACM symposium on access control models and technologies*, pages 1–10, 2004.