LicenseScript: A Novel Digital Rights Language

Cheun Ngen Chong, Ricardo Corin, Sandro Etalle, Pieter Hartel, and Yee Wei Law

Universiteit Twente
The Netherlands
{chong,corin,etalle,pieter,ywlaw}@cs.utwente.nl

Abstract

We propose LicenseScript as a new multi-set rewriting/logic based language for expressing dynamic conditions of use of digital assets such as music, video or private data. LicenseScript differs from the other DRM languages in that it caters for the intentional but legal manipulation of data. We believe this feature is the answer to providing the flexibility needed to support emerging usage paradigms of digital data.

1 Introduction

Most information, such as books, music, video, personal data and sensor readings (we generalize this information as *data*), is intended for a specific use. This specific use should conform to particular terms and conditions, which are often governed by *licenses*. To describe a license, a specific language is needed. In fact, the last few years have witnessed a proliferation of *Digital Rights Languages* (DRL). These are usually based on XML, e.g. XrML [7] (`www.xrml.org`) and ODRL [8] (`www.odrl.net`).

It is now widely acknowledged that the above-mentioned XML-based DRLs have some important shortcomings: (1) the syntax is complicated and obscure when the conditions of use become complex, (2) these languages lack a formal semantics [5, 11]; the meaning of licenses relies heavily on the human interpretation, and (3) the language cannot express many useful copyright laws [10].

Gunter *et al* [5] overcome some of the drawbacks by introducing an abstract model and language with a corresponding formal semantics. Pucella and Weissman [11] follow up Gunter *et al*'s effort with more rigour. They reason about the licenses and the user's actions with respect to the licenses; this is done by means of a temporal *deontic* logic.

However, none of the DRL introduced so far are flexible enough to accommodate the sophisticated conditions of use that are needed for real use. Consider, for instance, the following two scenarios:

DJs Nowadays, anyone can compose, edit, and distribute music and videos. In fact, the success that DJs are having in the media world demonstrates that there is a clear need for a system that allows copyrighted music to be lawfully clipped, mixed, edited, and later be played in public and sold on the market.

Games Multiplayer network games are poised to become a multi million $ industry. For instance, the massively multiplayer online role-playing games (MMORPG) industry is booming in Korea: In 2002 the revenues created by MMORPG reached approximately USD 1.76 billions (Press Release dated 03 Feb 2003, by Global Information Inc. `www.gii.co.jp`). An interesting aspect of these games is that gamers may create and use *their own data* (e.g. characters, virtual belongings, etc.) within the game. Trading of virtual characters is already reality, leading to a situation in which characters belonging to different owners are integral part of a game, the rights of which belong to a third party. There is a strong need for a DRL that allows a gamer to create, share, edit and re-sell digital characters, and that takes care of the lawful integration of digital goods belonging to different owners.

The scenarios above require a licensing language that is capable of capturing the evolution of data and the corresponding licenses. Additionally, the licensing language should be able to capture the intention of copyright laws, such as: (1) Fair use (reproduction in copies for purposes of education, and critiques, etc.), (2) Exemption of Public Display (public display and performance of copyrighted content), (4) Ephemeral Recordings (for local transmissions, security or archival preservation), etc. These have become one of the main requirements of DRL yet the scope of the current DRLs is limited to rights expression [10].

Licenses should prevent unauthorised used, but at the same time should provide a flexible user-friendly tool for accessing content. For instance, a user who rightfully downloads a piece of music on her laptop may legitimately expect to be able to play it in the car as well, or to let her friends listen to it. Therefore, licenses should be bound to an authorized domain, rather than to a person or to a device. The concept of authorized domain was introduced by the DVB consortium (`www.dvb.org`), and is discussed in detail by Van den Heuvel *et al.* [13]. State-of-the-art languages do not address this issue.

In this paper, we propose LicenseScript, a language that is able to express conditions of use of dynamic and evolving data in authorized domains. LicenseScript is based on (1) multiset rewriting, which is able to capture the *dynamic* evolution of licenses, (2) logic programming, which captures the static terms and conditions on a licence, and (3) a judicious choice of the interfacing mechanism between the static and dynamic domains. LicenseScript makes it possible to express a multitude of sophisticated usage patterns precisely and clearly. The formal basis of LicenseScript (Multiset rewriting and logic programming) provides for a concise and explicit formal semantics.

The organization of the remainder of the paper is as follows: Section 2 explains the LicenseScript language, and the formal basis. Section 3 demonstrates some examples for the DJ system. Section 4 elaborates the related work of rights languages. Section 5 concludes the paper and discusses future work. In the Appendix, we compare our system to that of Pucella et al. [11], by showing how a central example of [11] can be rendered in the LicenseScript.

2 Preliminaries

As mentioned earlier, LicenseScript is based on multiset rewriting; licensed data is bound to the terms in the multiset. Furthermore, we also use logic programming for the specification of licenses; the reader is thus assumed to be familiar with the terminology and the basic results of the semantics of logic programs [1, 9]. In particular, we borrow the concept of *SLD-resolution*: we write $P \vdash_{SLD} Q$ when there is a successful SLD-derivation for goal (or *query*) Q in program P. This basically means that the execution of the query Q in the program P yields to *success*.

Also, since terms in the multiset may contain variables, we need to fix the notation: we use words that start with uppercase $(X, Y, ...)$ to denote variables, and lowercase $(music\_piece, video\_track, expires, ...)$ to denote constants. We also use the `typewriter` font to denote Prolog code.

2.1 Licenses

A license defines the terms and conditions of use for music, videos etc. Therefore, a license contains at least two relevant items of information: (i) a reference to the *data* that is being licensed, and (ii) the *conditions of use* on that data.

In our formalism, a license is represented by a term of the form $lic(content, \Delta, \mathcal{B})$ (see also Figure 1), where:

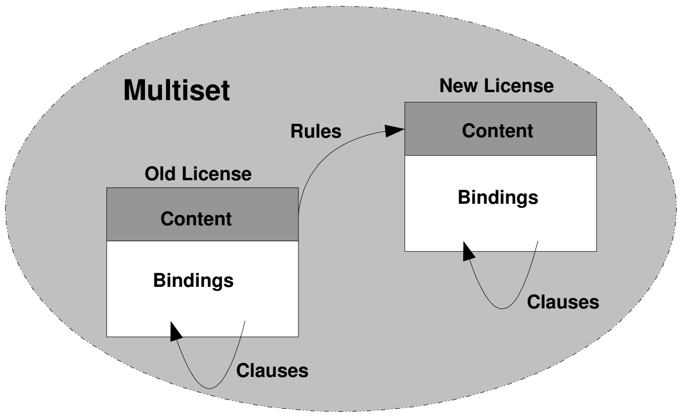

Figure 1: The transformation of licenses with *content* and *bindings* in a multiset caused by rules.

- *content* is a unique identifier representing the data the license refers to.
- Δ is a set of *clauses*, i.e., a Prolog program. This program defines when certain operations (like *play*) are allowed.
- $\mathcal{B}$ is a set of *bindings*, i.e., a set containing elements of the form *name* $\equiv$ *value*. For instance $\{\texttt{expires} \equiv 10/10/2003\}$ is a bindings set.

Bindings are needed to have a flexible way of storing counters and modifiable data. A license could be regarded as a database in which Δ is the intensional part, while $\mathcal{B}$ is the extensional one.

In order to interface licenses with the external world, we have to define an *interface*, i.e., a set of reserved calls that form the "API" of the license. The precise definition of this interface is beyond the scope of this paper. However, in the sequel, we use one particular call (from now reserved) called `canplay(·)`; this term is used to indicate when a license allows a given piece of music to be played: if the query $\texttt{canplay}(B, B')$ succeeds in the program Δ, this means that the license $lic(a, \Delta, B)$ allows the piece a to be played. Notice that we passed the set of bindings B as an argument to the query. As output we get $\mathcal{B}'$, the new set of bindings that hold *after* the operation has been carried out.

Example 1

1. *The license* $lic(mus, \{\texttt{canplay(X,X)} :- \texttt{true}.\}, \{\})$ *allows mus to be played.*
2. *The license* $lic(mus, \{\}, \{\})$ *does not allow any operation on* mus.
3. *The license*

$$lic(a, \Delta, \{\texttt{expires} \equiv 10/10/2003\}) \tag{1}$$

where Δ *is*

$$\{\texttt{canplay(B,B)} \quad :- \quad \texttt{today(D)},\ \texttt{get\_value(B,expires,Exdate)},\ \texttt{Exdate} > \texttt{D}.\} \tag{2}$$

allows to play a *until the given expiration day.*

`get_value(B,n,V)` is a primitive call that reports in `V` the value of the name `n` according to the set of bindings `B`. To further clarify how this last license actually works, we need to explain the role of the *domain*: notice that in the previous example we refer to the call `today(D)` that is not defined anywhere. `today(D)` should be also regarded as a primitive call, that binds the variable `D` to the current system date. In the remainder, we gather all such primitives in a special program that we call the *domain*, denoted $\mathcal{D}$. Notice that there can be many domains in which licenses reside, and probably a domain will have different meanings for the primitives than another domain.

Changing the Bindings There are situations in which the "execution" of a license should be followed by a modification on the set of bindings. Consider for instance a license that allows to play a piece of music only a given number of times: every time a *play* action is carried out, a counter should be increased. This is done by means of the primitive $\texttt{set\_value}(\mathit{Oldbindings}, \mathit{name}, \mathit{value}, \mathit{Newbindings})$. This primitive allows a name from a binding to be associated with a new value, which we use to support the evolution of licences. Consider, for instance, the following license:

$$lic(a, \Delta, \{\texttt{played\_times} \equiv 3\}) \tag{3}$$

where Δ consists of the following clause:

$$\texttt{canplay}(\texttt{B}, \texttt{B}') \;:-\; \texttt{get\_value}(\texttt{B}, \texttt{played\_times}, \texttt{R}),\; \texttt{R} < \texttt{10}, \texttt{set\_value}(\texttt{B}, \texttt{played\_times}, \texttt{R} + \texttt{1}, \texttt{B}').$$

Here, we first extract the value of variable `played_times` into local variable `R`. Then, if we have not exhausted the possible playing times allowed by the license (in this case, 10), we proceed to increase the value of `played_times` from bindings `B` to $\texttt{R} + \texttt{1}$, into new *output* bindings set $\texttt{B}'$.

2.2 The Rules

Licenses typically reside inside a device. The communication between this device and the licenses is done by means of *rewrite rules*. Their syntax is that of *multiset rewriting* (we adopt Gamma notation [3, 2]), where rules are of the form:

$$\mathit{name(args)} \quad : \quad \mathit{lms} \rightarrow \mathit{rms} \;\Leftarrow\; \mathit{cond}$$

Here *name(args)* is a prolog atom representing the license name and arguments (we call this atom *rule label*); *lms* represents the original (left) multiset, which is to be replaced with (right) multiset *rms*; *cond* refers to the conditions of applying the rule. A condition is a list of *queries* to the clauses of a license, of the form $\Delta \vdash \phi$. We write $\Delta \vdash \phi$ whenever query ϕ succeeds in program Δ. (We consider the primitives of the mentioned *domain* $\mathcal{D}$ also available at run time.) An example for a rule is the following:

$$\begin{array}{rcl} play(X) & : & lic(X, \Delta, \mathcal{B}) \rightarrow lic(X, \Delta, \mathcal{B}') \\ & \Leftarrow & \Delta \vdash \texttt{canplay}(\mathcal{B}, \mathcal{B}') \end{array} \tag{4}$$

2.3 LicenseScript Execution Model

In the sequel we say that a term t *matches* with a term s if there exists a substitution σ, $\mathit{Dom}(\sigma) = \mathit{Var}(t)$ such that $t\sigma = s$. σ is called the *matching substitution*. It is clear that if a term matches with another one there exists a unique matching substitution.

As we already mentioned, licenses are represented by terms of the form $lic(\mathit{content}, \Delta, \mathcal{B})$. For the sake of exposure, we assume that all available licenses are stored in a multiset $\mathcal{MS}$. Intuitively, the whole process of execution of actions in LicenseScript, proceeds as follows:

1. the *environment* (e.g., the user) communicates to a device (e.g., a TV set) the desire of execution of a given command. This command is called a *request action*, and contains information about what action wants to be executed, and over what content the action should be applied to. Request actions are represented by Prolog atoms. For example, a possible action is $play(music\_piece)$.

2. The device that receives the action request, checks if there is a rule whose label matches with the requested action.

   For instance, action $play(a)$ matches with the head of the rule $play(X) : lic(X, \Delta, \mathcal{B}) \rightarrow lic(X, \Delta, \mathcal{B}') \Leftarrow \Delta \vdash \texttt{canplay}(\mathcal{B}, \mathcal{B}')$. The matching substitution in this case is $\sigma = \{X/a\}$.

   In principle, there could be more than one label matching the request action. However, it is very easy to avoid this situation, so we assume that at most one label will be able to match the request.

   If there are no labels matching the request, the request fails.

3. Suppose that the rule $rule(arg) : lms \rightarrow rms \Leftarrow cond$ matches with the request atom, with matching substitution σ_1.

   We have to check whether there exists one (or more) license(s) in $\mathcal{MS}$ that can be matched with the lhs of the rule.

   If there exists a sub-multiset *lics* of $\mathcal{MS}$ and a substitution σ_2 such that

   (a) $lms\ \sigma_1\sigma_2 = lics$, and

   (b) $cond\ \sigma_1\sigma_2$ succeeds with computed answer σ_3.

   Now, the requested action is authorized and can be carried out. Recall that σ_3 carries the new bindings.

   (Actually, there can be some nondeterminism here, since there could be different sub-multisets *lics* of $\mathcal{MS}$ satisfying the above conditions and even more than one σ_2. This corresponds to the possible situation in which the user possesses more than one license that allows him to effectuate the desired action. In this case, we can assume that the system asks the user which license should be used).

4. The final step involves updating the multiset: this is done by replacing *lics* in $\mathcal{MS}$ with $rms\sigma_1\sigma_2\sigma_3$.

Example 2 *Let $\mathcal{MS}$ be the multiset containing the following licenses: $\{lic(music, \Gamma, \mathcal{C}), lic(video, \Sigma, \mathcal{D})\}$, where $\mathcal{C} = \{\texttt{played\_times} \equiv 2\}$ and $\mathcal{D} = \{\texttt{played\_times} \equiv 10\}$, and*

$$\Gamma = \Sigma = \{\texttt{canplay(B,B')} : -\texttt{get(B,played\_times,N),N} < \texttt{10},\ \texttt{set(B,played\_times,N} + \texttt{1,B')}\}$$

Let R be the following singleton set of rewriting rules:

$$\{play(X) : lic(X, \Delta, \mathcal{B}) \rightarrow lic(X, \Delta, \mathcal{B}') \Leftarrow \Delta \vdash \texttt{canplay}(\mathcal{B}, \mathcal{B}')\}$$

Now, suppose the environment requests the action $play(music)$. This will match rule $play(X)$, giving matching $\sigma_1 = \{X/music\}$.

The next step involves for looking in $\mathcal{MS}$ for possible occurrences of $lic(music, \Delta, \mathcal{B})$. The only possible match is, of course, $lic(music, \Gamma, \mathcal{C})$. This gives us matching $\sigma_2 = \{\Delta/\Gamma, \mathcal{B}/\mathcal{C}\}$.

Now, condition $\Delta \vdash \texttt{canplay}(\mathcal{C}, \mathcal{B}')$ has to be evaluated. Since variable `played_times` *is less than* 10 *in $\mathcal{C}$, the $\texttt{canplay}(\mathcal{C}, \mathcal{B}')$ succeeds in the Prolog program Δ, hence the condition is satisfied. We get the computed answer substitution $\sigma_3 = \{\texttt{B}'/\{\texttt{played\_times} \equiv 3\}$.*

Finally, the update of $\mathcal{MS}$ is carried out. License $lic(music, \Gamma, \mathcal{C})$ is removed from $\mathcal{MS}$, and substituted with $lic(music, \Gamma, \mathcal{C}')$, where $\mathcal{C}' = \{\texttt{played\_times} \equiv 3\}$.

Example 3 *Consider the same multiset and rules of the previous example. Suppose now request action* $play(video)$ *is issued. This action, even though has a matching rule and a matching license in the multiset, cannot be carried out completely. This is so since, in the unique matched license (that is,* $lic(video, \Sigma, \mathcal{D})$*) condition* $\Delta \vdash \texttt{canplay}(\mathcal{B}, \mathcal{B}')$ *does not hold.*

3 DJ Examples

In this section we provide some additional examples, showing the flexibility of LicenseScript.

3.1 Authorized Domain

We start by showing a simple way how *authorized domains* could be implemented in LicenseScript. To this end, we use a unique identification to represent an authorized domain. This unique identification could be the certificate of a system. Furthermore, we introduce two license bindings, `in_domain` and `to_domain` in the license: `in_domain` is used to represent the domain where the license is currently valid, `to_domain` represents the domain to which the license is allowed to move.

For example, to state that the *edit* operation is *only* valid in the domain `cert`, we can use the following license

$$\begin{array}{ll} & lic(mus, \Delta, \{\texttt{in\_domain} \equiv \texttt{cert}\}) \\ \text{where} & \Delta = \texttt{canedit}(\mathcal{B}, \mathcal{B}') :- \texttt{identify}(\texttt{Id}_1),\ \texttt{get\_value}(\mathcal{B}, \texttt{in\_domain}, \texttt{Id}_2), \texttt{Id}_1 = \texttt{Id}_2. \end{array} \tag{5}$$

The rule for the *edit* operation is then:

$$\begin{array}{rcl} edit(mus) & : & lic(mus, \Delta, \mathcal{B}) \rightarrow lic(mus, \Delta, \mathcal{B}') \\ & \Leftarrow & \Delta \vdash \texttt{canedit}(\mathcal{B}, \mathcal{B}') \end{array} \tag{6}$$

The subquery $\texttt{identify}(\texttt{Id}_1)$ is a primitive in $\mathcal{D}$, which is used to model the identification of the current domain (i.e. to retrieve the identity of the current domain). This license checks that the identification of the current authorized domain must be equal to the identification of the authorized domain stated in the license.

Changing the Domain The *edit* rule does not change the authorized domain. To illustrate the influence of changing the authorized domain, we use another operation in the DJ system, *move* operation. The license to move a music track from one domain to another domain may look like this:

$$lic(mus, \Delta, \{\texttt{in\_domain} \equiv \texttt{cert}_1, \texttt{to\_domain} \equiv \texttt{cert}_2\}) \tag{7}$$

This license signifies that the license is allowed to move from authorized domain with identification value of $\texttt{cert}_1$ to authorized domain with $\texttt{cert}_2$. The clauses Δ of the license can be written as follows:

$$\begin{array}{rcl} \texttt{canmove}(\mathcal{B}, \mathcal{B}') & :- & \texttt{identify}(\texttt{Id}_1),\ \texttt{get\_value}(\mathcal{B}, \texttt{to\_domain}, \texttt{Id}_2),\ \texttt{Id}_1 = \texttt{Id}_2, \\ & & \texttt{set\_value}(\mathcal{B}, \texttt{in\_domain}, \texttt{Id}_2, \mathcal{B}'), \texttt{set\_value}(\mathcal{B}, \texttt{to\_domain}, \texttt{Id}_1, \mathcal{B}'). \end{array} \tag{8}$$

When the license is moved to authorized domain $\texttt{cert}_2$, the primitive atom of the authorized domain $\mathcal{D}$, $\texttt{identify}(\texttt{Id}_1)$ retrieves the identification value of the current authorized domain (where the license is moved to). A check is made to see whether the license is allowed to move to this authorized domain, akin to Clause 5. If the check succeeds, the values of the bindings `in_domain` and `to_domain` are exchanged. Thereby indicating the license is allowed to move back to the original authorized domain.

Therefore, the rule for the *move* operation can be built as follows:

$$\begin{array}{rcl} move(mus) & : & lic(mus, \Delta, \mathcal{B}) \rightarrow lic(mus, \Delta, \mathcal{B}') \\ & \Leftarrow & \Delta \vdash \texttt{canmove}(\mathcal{B}, \mathcal{B}') \end{array} \tag{9}$$

3.2 Payment

We now show how various forms of payment can be modelled in LicenseScript.

We assume that a license can carries balance represented by the binding `wallet` $=$ `x`. We postulate the existence of a new primitive, *add_to_balance* to increase the balance. Before attempting to perform any other operations on the music track, the Wallet must be loaded. The relevant clause of the license for this purpose would be:

$$\begin{aligned}\texttt{canload}(\mathcal{B},\mathcal{B}') \quad :- \quad & \texttt{add\_to\_balance(M)},\ \texttt{get\_value}(\mathcal{B},\texttt{wallet},\texttt{W}),\ \texttt{W}'\ \texttt{is}\ \texttt{W}+\texttt{M},\\ & \texttt{set\_value}(\mathcal{B},\texttt{wallet},\texttt{W}',\mathcal{B}').\end{aligned}$$

The rule for *load* operation may be written as follows:

$$\begin{aligned}load(mus) \quad : \quad & lic(mus,\Delta,\mathcal{B}) \rightarrow lic(mus,\Delta,\mathcal{B}')\\ \Leftarrow \quad & \Delta \vdash \texttt{canload}(\mathcal{B},\mathcal{B}')\end{aligned} \tag{10}$$

There are at least three common alternatives of payment: *before*-use, *after*-use and *per*-use [5]. In the rest of this section take the *play* operation as our illustration of the aforementioned payment methods.

The rule for the *pay* operation can be written as follows:

$$\begin{aligned}pay(mus) \quad : \quad & lic(mus,\Delta,\mathcal{B}) \rightarrow lic(mus,\Delta,\mathcal{B}')\\ \Leftarrow \quad & \Delta \vdash \texttt{dopay}(\mathcal{B},\mathcal{B}')\end{aligned}$$

The license bindings should include

$$\{\texttt{wallet} \equiv \texttt{m}, \texttt{owner} \equiv \texttt{cert}_1, \texttt{rate} \equiv \texttt{x}, \texttt{provider} \equiv \texttt{cert}_2, \texttt{paid} \equiv \texttt{false}, \texttt{used} \equiv \texttt{false}\} \tag{11}$$

Here `rate` designates the rate of payment, while `provider` represents the content provider to whom the money must be transferred; `paid` is a boolean indicating whether the payment has already taken place; similarly, `used` is used to indicate if the license has been used. The clause for pay-before-use is:

$$\begin{aligned}\texttt{dopay}(\mathcal{B},\mathcal{B}') \quad :- \quad & \texttt{get\_value}(\mathcal{B},\texttt{used},\texttt{Used}),\ \texttt{Used}=\texttt{false},\ \texttt{get\_value}(\mathcal{B},\texttt{paid},\texttt{Paid}),\\ & \texttt{Paid}=\texttt{false},\ \texttt{get\_value}(\mathcal{B},\texttt{wallet},\texttt{Balance}),\ \texttt{get\_value}(\mathcal{B},\texttt{rate},\texttt{Rate}),\\ & \texttt{get\_value}(\mathcal{B},\texttt{provider},\texttt{Pro}),\ \texttt{Balance} \geq \texttt{Rate},\ \texttt{N}\ \texttt{is}\ \texttt{Balance}-\texttt{Rate},\\ & \texttt{transfers}(\texttt{Pro},\texttt{Rate}),\ \texttt{set\_value}(\mathcal{B},\ \texttt{wallet},\texttt{N},\mathcal{B}'),\\ & \texttt{set\_value}(\mathcal{B},\texttt{paid},\texttt{true},\mathcal{B}').\end{aligned} \tag{12}$$

Here the value of `rate` is deducted from the balance; the binding of `paid` is set to *true* to indicate the payment has been made; the primitive `transfers(Pro, Rate)` models the transfer of money to the `provider`.

The license clause for the *play* operation can be constructed as follows:

$$\begin{aligned}\texttt{canplay}(\mathcal{B},\mathcal{B}') \quad :- \quad & \texttt{get\_value}(\mathcal{B},\texttt{paid},\texttt{Paid}),\ \texttt{Paid}=\texttt{true},\ \texttt{get\_value}(\mathcal{B},\texttt{used},\texttt{Used}),\\ & \texttt{Used}=\texttt{false},\ \texttt{set\_value}(\mathcal{B},\texttt{used},\texttt{true},\mathcal{B}').\end{aligned} \tag{13}$$

We may introduce two types of pay-after-use: (1) bulk payment, and (2) payment for the used period. The bulk payment license contains the similar license bindings of pay-before-use license, as shown in bindings 11.

The clause of the license for the bulk payment can be written as follows:

$$\begin{aligned}\texttt{dopay}(\mathcal{B},\mathcal{B}') \quad :- \quad & \texttt{get\_value}(\mathcal{B},\texttt{paid},\texttt{Paid}),\ \texttt{Paid}=\texttt{false},\ \texttt{get\_value}(\mathcal{B},\texttt{used},\texttt{Used}),\\ & \texttt{Used}=\texttt{true},\ \texttt{get\_value}(\mathcal{B},\texttt{wallet},\texttt{Balance}),\ \texttt{get\_value}(\mathcal{B},\texttt{rate},\texttt{Rate}),\\ & \texttt{get\_value}(\mathcal{B},\texttt{provider},\texttt{Pro}),\ \texttt{Balance}\geq\texttt{Rate},\ \texttt{N is Balance}-\texttt{Rate},\\ & \texttt{transfers}(\texttt{Pro},\texttt{Rate}),\ \texttt{set\_value}(\mathcal{B},\texttt{wallet},\texttt{N},\mathcal{B}'),\\ & \texttt{set\_value}(\mathcal{B},\texttt{paid},\texttt{true},\mathcal{B}'). \end{aligned} \tag{14}$$

where the binding `used` is checked if the license is used.

Notice the difference between the bulk payment after use and pay before use is that the binding `used` must be `true` for bulk payment after use, otherwise for pay before use.

For the second pay-after-use method (just pay for the period of use), the license bindings include:

$$\{\texttt{wallet}\equiv\texttt{m},\texttt{owner}\equiv\texttt{cert}_1,\texttt{rate}\equiv\texttt{x},\texttt{provider}\equiv\texttt{cert}_2,\texttt{paid}\equiv\texttt{false},\texttt{used}\equiv\texttt{0}\} \tag{15}$$

The novelty here is that `used` is now a positive integer that records the length of time the license has been used. The clauses for the corresponding license is:

$$\begin{aligned}\texttt{dopay}(\mathcal{B},\mathcal{B}') \quad :- \quad & \texttt{get\_value}(\mathcal{B},\texttt{paid},\texttt{Paid}),\ \texttt{Paid}=\texttt{false},\ \texttt{get\_value}(\mathcal{B},\texttt{used},\texttt{Used}),\\ & \texttt{Used}>\texttt{0},\ \texttt{get\_value}(\mathcal{B},\texttt{wallet},\texttt{Balance}),\ \texttt{get\_value}(\mathcal{B},\texttt{rate},\texttt{Rate}),\\ & \texttt{get\_value}(\mathcal{B},\texttt{provider},\texttt{Pro}),\ \texttt{N is Used}*\texttt{Rate},\\ & \texttt{Balance}\geq\texttt{Rate},\ \texttt{M is Balance}-\texttt{N},\ \texttt{transfers}(\texttt{Pro},\texttt{N}),\\ & \texttt{set\_value}(\mathcal{B},\texttt{wallet},\texttt{M},\mathcal{B}'),\ \texttt{set\_value}(\mathcal{B},\texttt{paid},\texttt{true},\mathcal{B}'). \end{aligned} \tag{16}$$

To show how the *play* operation acquires pay after use (for certain used period), we build the license clause as follows:

$$\begin{aligned}\texttt{canplay}(\mathcal{B},\mathcal{B}') \quad :- \quad & \texttt{get\_value}(\mathcal{B},\texttt{paid},\texttt{Paid}),\ \texttt{Paid}=\texttt{false},\ \texttt{get\_value}(\mathcal{B},\texttt{used},\texttt{Used}),\\ & \texttt{Used}=\texttt{0},\ \texttt{logs}(\texttt{U}),\ \texttt{set\_value}(\mathcal{B},\texttt{used},\texttt{U},\mathcal{B}'). \end{aligned} \tag{17}$$

where the primitive `logs(U)` models the system logging the length of the time the DJ has played (say, by streaming) the music track.

The license bindings for pay-per-use may be listed as follows:

$$\{\texttt{wallet}\equiv\texttt{m},\texttt{owner}\equiv\texttt{cert}_1,\texttt{rate}\equiv\texttt{x},\texttt{provider}\equiv\texttt{cert}_2\} \tag{18}$$

where the binding $\texttt{rate}\equiv\texttt{x}$ designates the rate of the payment, while $\texttt{provider}\equiv\texttt{cert}_2$ represents the content provider who provides the music track, and to whom the money transfers.

The clause for pay-per-use license is similar to Clause 12. The difference between pay-before-use and pay-per-use is that there is no payment indicator (the binding `paid`) in the pay-per-use license:

$$\begin{aligned}\texttt{canplay}(\mathcal{B},\mathcal{B}') \quad :- \quad & \texttt{get\_value}(\mathcal{B},\texttt{wallet},\texttt{Balance}),\ \texttt{get\_value}(\mathcal{B},\texttt{rate},\texttt{Rate}),\\ & \texttt{get\_value}(\mathcal{B},\texttt{provider},\texttt{Pro}),\ \texttt{Balance}\geq\texttt{Rate},\ \texttt{N is Balance}-\texttt{Rate},\\ & \texttt{transfers}(\texttt{Pro},\texttt{Rate}),\ \texttt{set\_value}(\mathcal{B},\ \texttt{wallet},\texttt{N},\mathcal{B}'). \end{aligned} \tag{19}$$

3.3 Clipping

In our system, a DJ who has purchased a music track from a content provider requires some comments from other DJs. She can *clip* the license and the content, and then she may send the clipped results to other DJs to listen to.

The license looks like this:

$$lic(mus, \Delta, \{\texttt{length} \equiv \texttt{240}\}) \tag{20}$$

Here the binding of `length` represents the length of the music track, which in this case 240 seconds. The license clause for *clip* operation may be written as follows:

$$\begin{aligned} \texttt{canclip}(\mathcal{B}, \mathcal{B}') \quad :- \quad & \texttt{get\_value}(\mathcal{B}, \texttt{length}, \texttt{L}),\ \texttt{L} > \texttt{0},\ \texttt{copy\_bindings}(\mathcal{B}, \mathcal{B}'), \\ & \texttt{N is L} * 0.5,\ \texttt{set\_value}(\mathcal{B}, \texttt{length}, \texttt{N}, \mathcal{B}'). \end{aligned} \tag{21}$$

Here the primitive $\texttt{copy\_bindings}(\mathcal{B}, \mathcal{B}')$ copies the bindings of one license to another, in this case from $\mathcal{B}$ to $\mathcal{B}'$. The new license generated, with the new set of bindings $\mathcal{B}'$ represents 120 seconds.

The corresponding rule for *clip* operation may be written as follows:

$$\begin{aligned} clip(mus) \quad : \quad & lic(mus, \Delta, \mathcal{B}) \to lic(mus, \Delta, \mathcal{B}), lic(mus, \Delta, \mathcal{B}') \\ \Leftarrow \quad & \Delta \vdash \texttt{canclip}(\mathcal{B}, \mathcal{B}') \end{aligned} \tag{22}$$

The clipped license can be sent to other DJs to achieve the purpose of fair use.

4 Related Work

In this section, we briefly discuss the related work. We elaborate the digital rights language proposed by Gunter *et al.* and Pucella and Weissman.

Gunter et al. [5] from InterTrust Technologies Corporation and Pucella and Weissman [11] from Cornell University have presented two logics for licenses. Gunter et al. by borrowing techniques from programming semantics [6], have developed a model and a language for describing licenses. Their logic consists of a domain of sequences of events called *realities*. In their logic, an event $e \in Event$ is modelled as a pair of time $t \in Time$ and action $a \in Action$:

$e ::= t : a$

Two kinds of action have been envisaged:

$a ::= render[w, d] \mid pay[x]$

Here $w \in Work$ denotes the copyrighted work (content); $d \in Device$ represents a device; and x is a decimal number, representing the amount of payment. $render[w, d]$ denotes the action of rendering the work w on device d. $pay[x]$, as the name implies, symbolizes the action of paying an amount of x for using the work. Therefore, the event $t : render[w, d]$ implies that the work w is rendered on device d at time t. Only one event is allowed at a time. A finite set of events is embodied in a reality, $r \in Reality$. A license, $l \in License$, is a set of realities. Most licenses consist of infinitely many realities in order to allow the use of a work at one or more of infinitely many times during some period.

Using the proposed model, Gunter et al. have formularized several standard license types, which they call *simple licenses*. The simple licenses are "Up Front" (pay before use), "Flat Rate"(pay after use) and "Per Use"(pay per use). Simple licenses can be used as the building blocks of more complex licenses.

Pucella and Weissman follow up Gunter et al.'s effort with more rigour [11]. The following summarizes the core concepts:

- There are 4 standard domains: (1) $\mathcal{N}$ for license names (every license is assigned a name), (2) $\mathcal{W}$ for works (copyrighted works), (3) $\mathcal{D}$ for devices (for rendering works), (4) $\mathcal{A}$ for atomic actions (which is a union of the *render* and the *pay* action).

- There are 3 syntactic categories: (1) action expression, (2) license, (3) formula.
- An action expression α is composed of action-name pairs (i.e. pairs of (a, n) where $a \in \mathcal{A}$ and $n \in \mathcal{N}$). Action expressions are either *permitted* ($P\alpha$) or *obligatory* ($O\alpha$). (This distinction is what makes their logic more accessible and complete than Gunter et al.'s.)
- A license l is an action sequence (not to be confused with an action expression).
- A formula is made up of $n : l$ terms and α terms. $n : l$ means the action sequence l is valid for the license labelled n.
- A run r associates a time t with the licenses issued at that time and the actions performed by the client at that time. At most, one action per time per license can occur.
- An interpretation π is a tuple (P, O) where P is a permission assignment and O is an obligation assignment. Simply speaking, if $(a, n) \in P(t)$, then action a is permitted by license n at time t. Similar intuition applies to $O(t)$.
- The *consistency* notion says that if an interpretation π enforces all the permissions and obligations required by the licenses issued by a run r, then π is consistent with r. In other words, checking for license violation in a run boils down to checking whether the prevailing interpretation is consistent with the run.

LicenseScript uses multiset rewriting which is more expressive than the denotational semantics of Gunter et al. LicenseScript is also readily subject to logical parallelism. Pucella et al.'s logic is only a starting point, with the assumption of one client and one provider and therefore definitely does not cater for concurrency, like LicenseScript does. To state the obvious, Pucella et al. also have not yet taken into account the malleability of licenses and contents (e.g. as a result of "clipping" and "mixing"), and the concepts of authorized domains.

5 Conclusions and Future Work

We propose a novel rights language based on multiset rewriting and logic programming: LicenseScript. We present the design of the language using a scenario that represents an elaborate pattern of use of content. In this scenario, a DJ edits, clips and mixes music such that the terms and conditions on the music used *and* produced by the DJ are satisfied.

LicenseScript differs from other DRLs in that it has an explicit static and dynamic part. The terms and conditions on content form the static part. These terms and conditions usually derive from legal, regulatory and business rules, and are therefore appropriately expressed using Prolog clauses [12]. A license is used in a changing context and must therefore have the ability to evolve. The dynamics are represented by interpreting a license as an element of a multi-set to which multi-set rewrite rules are applied. These rules represent the way in which the context (devices and systems) act upon licences. The dual nature of a license (static vs dynamic) is thus represented by a two-tier structure of LicenseScript. The two levels are linked by a set of bindings that represents the current state of the evolution.

Future work is threefold. Firstly, we will endow the language with a trace-based semantics, which allows us to reason about the evolvement of licences. This will make it possible to design licences in such a way that no unexpected patterns of use will emerge (safety) and secondly that desirable patterns of use can emerge (liveness). Secondly we will implement the language, using an existing DRM platform [4]. Thirdly, we plan to study in detail relevant legal, regulatory and business cases to ensure that the language is convenient to use.

Acknowledgement

We like to thank Prof. Wim Jonker (Philips Research) and Ernst-Jan Goedvolk (Telematica Institute) for their valuable help on this paper. This work was partially supported by the Telematica Institute.

Appendix

In this section, we compare our system to that of Pucella et al. [11], by showing how a central example of [11] can be rendered in LicenseScript.

Pucella *et al.*[11] consider the scenario of an owner of an online journal requiring a fee to be paid before each access. This scenario is similar to the pay-per-use operation that we have shown in section 3.2, with a slight dissimilarity: the users must *pay* before even activates the application to read the digital journal (our payment is activated when the user activates the application).

Pucella *et al.*'s license is written as follows:

$$l \quad = \quad ((pay[fee](\bot)^{*}render[journal, d]) \cup \bot)^{*} \tag{23}$$

where d is the device that the user uses to access the journal; $\bot$ represents the null or "do nothing" action; $pay[fee]$ is the action of paying amount fee; $render[journal, d]$ is the action of accessing the journal using the device d (Refer to Reference [11] for more details on their logic).

Notice that the notations we apply in the following examples refer to section 3.2. We can express the similar license by using the LicenseScript language:

$$lic(journal, \Delta, \{\texttt{wallet} \equiv \texttt{m}, \texttt{paid} \equiv \texttt{false}, \texttt{rate} \equiv \texttt{n}, \texttt{provider} \equiv \texttt{cert}\}) \tag{24}$$

To express the *pay* and *render* operation, we build the license clauses, Δ as follows:

$$\begin{aligned}
\texttt{dopay}(\mathcal{B}, \mathcal{B}') \quad :- \quad & \texttt{get\_value}(\mathcal{B}, \texttt{paid}, \texttt{Paid}),\ \texttt{get\_value}(\mathcal{B}, \texttt{wallet}, \texttt{Balance}),\\
& \texttt{get\_value}(\mathcal{B}, \texttt{rate}, \texttt{Rate}),\ \texttt{Paid} = \texttt{false},\ \texttt{Balance} \geq \texttt{Rate},\\
& \texttt{get\_value}(\mathcal{B}, \texttt{provider}, \texttt{Provider}),\ \texttt{transfers}(\texttt{Provider}, \texttt{Rate}),\\
& \texttt{N is Balance} - \texttt{Rate},\ \texttt{set\_value}(\mathcal{B}, \texttt{wallet}, \texttt{N}, \mathcal{B}'),\\
& \texttt{set\_value}(\mathcal{B}, \texttt{paid}, \texttt{true}, \mathcal{B}').\\[1em]
\texttt{canrender}(\mathcal{B}, \mathcal{B}') \quad :- \quad & \texttt{get\_value}(\mathcal{B}, \texttt{paid}, \texttt{Paid}),\ \texttt{Paid} = \texttt{true},\\
& \texttt{set\_value}(\mathcal{B}, \texttt{paid}, \texttt{false}, \mathcal{B}').
\end{aligned} \tag{25}$$

Beware that after the *render* operation (shown in Clause 25), the binding `Paid` is reset to `false`. Thereby, the user needs to make the payment *again* if she likes to access the journal again.

Finally, our rules for both of the operations above can be built as follows:

$$\begin{aligned}
pay(journal) \quad : \quad & lic(journal, \Delta, \mathcal{B}) \rightarrow lic(journal, \Delta, \mathcal{B}')\\
\Leftarrow \quad & \Delta[\mathcal{B}] \cup \mathcal{D} \vdash \texttt{dopay}(\mathcal{B}, \mathcal{B}')
\end{aligned} \tag{26}$$

$$\begin{aligned}
render(journal) \quad : \quad & lic(journal, \Delta, \mathcal{B} \rightarrow lic(journal, \Delta, \mathcal{B}')\\
\Leftarrow \quad & \Delta[\mathcal{B}] \cup \mathcal{D} \vdash \texttt{canrender}(\mathcal{B}, \mathcal{B}')
\end{aligned} \tag{27}$$

References

[1] K. R. Apt. *From Logic Programming to Prolog.* International Series in Computer Science. Prentice Hall, 1997.

[2] J-P. Banâtre, P. Fradet, and D. L. Métayer. Gamma and the chemical reaction model: Fifteen years after. In C. Calude, G. Paun, G. Rozenberg, and A. Salomaa, editors, *Workshop on Multiset Processing (WMP)*, volume 2235 of *Lecture Notes in Computer Science*, pages 17–44. Springer-Verlag, Berlin, August 2001.

[3] M. R. V. Chaudron and E. D. de Jong. Towards a compositional method for coordinating gamma programs. In *Coordination Languages and Models, First International Conference (COORDINATION '96)*, pages 107–123. Lecture Notes in Computer Science 1061, Springer-Verlag, April 1996.

[4] C. N. Chong, R. van Buuren, P. H. Hartel, and G. Kleinhuis. Security attributes based digital rights management. In F. Boavida, E. Monteiro, and J. Orvalho, editors, *Joint Int. Workshop on Interactive Distributed Multimedia Systems / Protocols for Multimedia Systems (IDMS/PROMS)*, volume LNCS 2515, pages 339–352, Coimbra, Portugal, Nov 2002. Springer-Verlag, Berlin.

[5] C. Gunter, S. Weeks, and A. Wright. Models and languages for digital rights. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, pages 4034–4038, Maui, Hawaii, United States, January 2001. IEEE Computer Society Press.

[6] C. A. Gunter. *Semantics of Programming Languages: Structures and Techniques.* MIT Press, 1992. ISBN: 0262071436.

[7] H. Guo. Digital rights management (DRM) using XrML. In *T-110.501 Seminar on Network Security 2001*, page Poster paper 4, 2001. http://www.tml.hut.fi/Studies/T-110.501/2001/papers/.

[8] R. Iannella. Open digital rights management. In *World Wide Web Consortium (W3C) DRM Workshop*, page Position paper 23, January 2001. http://www.w3.org/2000/12/drm-ws/pp/.

[9] J. W. Lloyd. *Foundations of Logic Programming.* Symbolic Computation – Artificial Intelligence. Springer-Verlag, Berlin, 1987. Second edition.

[10] D. K. Mulligan, A. Burstein, and J. Erikson. Supporting limits on copyright exclusivity in a rights expression language standard. Comments and requirements, Samuelson Law, Technology & Public Policy Clinic and Clinic and the Electronic Privacy Information Center, Boalt Hall, School of Law, Berkeley CA 94720-7200, USA, August 2002.

[11] R. Pucella and V. Weissman. A logic for reasoning about digital rights. In *IEEE Proceedings of the Computer Security Foundations Workshop*, pages 282–294, Cape Breton, Nova Scotia, Canada, June 2002. IEEE Computer Society Press.

[12] M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory. The british nationality act as a logic program. *Communications ACM*, 29(5):370–386, May 1986.

[13] S.A.F.A. van den Heuvel, W. Jonker, F.L.A.J. Kamperman, and P.J. Lenoir. Secure content management in authorised domains. In *The World's Electronic Media Event IBC 2002, Sept. 13-17, Amsterdam RAI, The Netherlands*, September 2002. To appear.