Proceedings of the First International ODRL Workshop (Eds. Renato Iannella & Susanne Guth) 22–23 April, 2004, Vienna, Austria

Towards a Formal Semantics for ODRL
(Extended Abstract)

Markus Holzer, Stefan Katzenbeisser, Christian Schallhart

Institut für Informatik
Technische Universität München
Boltzmannstrasse 3
D–85748 Garching
{holzer,katzenbe,schallha}@in.tum.de

April 2, 2004

Abstract

We give a brief overview of a new way to model the semantics of ODRL permissions in a formal manner by using finite-automata like structures. The constructed automata capture the sequence of actions that a user is allowed to perform according to a specific permission. In contrast to previous approaches, our semantics is able to model sell and lend permissions.

1 Introduction

With the increasing availability and distribution of media in digital form, the protection of intellectual property faces new challenges. Popular file formats (like MPEG and MP3) facilitate the exchange of digital videos or sound clips, books are published electronically and films are distributed on DVDs. The possibility to easily and cheaply reproduce digital content without permission has raised the concern of the music, film and entertainment industries. Although classical analogue storage media (like VHS video or audio cassettes) can also be copied, the inevitable quality loss present in all analogue copies naturally limits the illegal distribution of copyrighted content. In the digital age, however, thousands of lossless copies can be produced easily and distributed over public networks.

In the past few years various techniques for preventing copying or restricting the access to copyrighted material (called *copy protection* mechanisms) were implemented. Examples of copy protection include encrypted digital TV broadcast (conditional access systems), access controls to copyrighted software through the use of license servers and technical copy protection mechanisms on the media (like the content management mechanism on DVDs). A copy protection scheme is a special flavor of a *digital rights management system*, which attempts to enable secure distribution of copyrighted content via open networks.

An integral part of a DRM system is a contract between the parties involved. This contract may (among other things) describe the permissions granted on an object distributed over a network, together with constraints and requirements to be met before the permission can be executed. For example, a typical constraint limits the number of times an object can be displayed or printed. A requirement may express that a certain fee has to be paid before executing the permission. Rights expression languages, such as ODRL [2] and XrML [5], allow to express such contracts in a formal manner. In this paper we deal with ODRL, but the construction is applicable to a large class of rights expression languages. Currently, the semantics of ODRL is described in the specification as English language text, whereas the syntax is expressed in XML. Unfortunately, the lack of a precisely defined formal semantics results in possible ambiguities. For example, ODRL allows a copyright holder to lend an object to another user; however, the specification does not precisely describe the the act of lending (and later returning) an object. Ambiguities can only be avoided if there exists a formal semantics for ODRL, specifying in an exact way the operations that are allowed by the contract.

This year, Pucella and Weissman [3, 4] presented a semantics for a fragment of ODRL (specifying agreements between two or more parties about one fixed object) based on many-sorted first order logic with equality. In this paper, we follow an alternative approach. More precisely, we give a semantics that models the actions that are allowed according to a contract; technically, this model is given in terms of automata. Each trace through the automaton describes a valid sequence of actions for one participant. Our model also enables to express **sell** and **lend** actions, where the object is transfered to different users. In Section 2 we introduce the fragment of ODRL that we are attempting to model in our semantics. Section 3 gives an informal overview of the model we propose for the semantics of ODRL. Finally, we present future research topics in Section 4.

2 A fragment of ODRL

ODRL is an extremely rich language that contains many different operations, requirements and constraints. For the sake of simplicity, we deal only with a fragment of ODRL in this paper. In fact, we are merely concerned with modeling offers that are not bound to a particular person. However, extensions of our model to the complete ODRL language (except descriptive elements like the ODRL context, which have no operational semantics) are possible.

In this work, we concentrate on the following permissions:

- **play**, **print**, **display**, **execute**. If no constraints are specified, a **play**, **print**, or **display** permission enables a party to play, print or display an object an arbitrary number of times. Similarly, **execute** allows an executable file to be processed on a computer.
- **sell**, **give**. Here, one party is allowed to transfer all rights over an object to a different party (either with or without paying a fee).
- **lend**, **lease**. Here, one party transfers for a certain period of time all rights over an object to a different party (either with or without paying a fee). After the specified time period, the object is returned to the original person.

In addition, we allow the following constraints that restrict the rights listed above:

- **user**. A right is bound to a specific user.
- **device**. A right can only be executed on a specific output device.
- **bound**. A right can be executed a maximum number of times.
- **transfer**. The transfer constraints specifies the permissions that are associated to an object after it is transferred with the **sell**, **give**, **lend** or **lease** action. By default, no permission is granted on the transferred object unless it is stated in a **transfer** constraint.
- **temporal**. A right is constrained to a specific time period.

From the list of ODRL requirements, we only consider **payment**; a permission that has an associated **payment** requirement can only be executed if a certain fee is paid in advance. Again, our model can be extended to cover other requirements as specified in the ODRL language definition.

For more information on the syntax of ODRL, we refer to [2]. As XML documents are hard to parse for humans, we deviate from the official ODRL syntax and present all examples in this paper in a more "human-readable" format.

In the rest of this work, we consider the following offers:

Example 1 *A user is granted a print and display permission; no constraints are imposed on the number of times these actions may be performed.*

```
<offer>
   ...
   <permission>
      <display/>
      <print/>
   </permission>
</offer>
```

Example 2 *A user is allowed to display an object at most three times.*

```
<offer>
   ...
   <permission>
      <display>
        <constraint>
           <count>3</count>
        </constraint>
      </display>
   </permission>
</offer>
```

Example 3 *A user is allowed to display an object at most two times in the (fictive) time interval 2–5.*

```
<offer>
   ...
   <permission>
      <display>
        <constraint>
           <interval>2-5</interval>
           <constraint>
             <count>2</count>
           </constraint>
        </constraint>
      </display>
   </permission>
</offer>
```

Example 4 *A user is allowed to display and print the object; furthermore, he can sell it to a different user in such a way that the recipient is able to print and display the object.*

```
<offer>
   ...
   <permission>
     <display>
     <print>
     <sell>
       <constraint>
         <transferPerm>
           <display>
           <print>
         </transferPerm>
       </constraint>
     </sell>
   </permission>
</offer>
```

Example 5 *A user is allowed to display and print the object; furthermore, he can lend it to a different user in such a way that the recipient is able to print and display the object. The ODRL code is similar to the code above, except that* `<sell>` *is replaced by* `<lend>`*. In addition, a temporal constraint describes the time period for the lending process.*

Example 6 *A user is allowed to display and print the object; furthermore, he can sell it to a different user, who has the same permissions (i.e., display, print and sell).*

```
<offer>
   ...
   <permission id="perm">
     <display/>
     <print/>
     <sell>
       <constraint>
         <transferPerm
            downStream="equal"
            idref="perm">
       </constraint>
     </sell>
    </permission>
</offer>
```

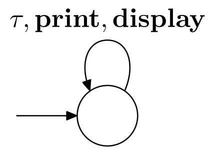
Example 7 *A user is allowed to display and print the object; furthermore, he can lend it to a different user, who has the same permissions (i.e., display, print and lend).*

3 A Semantics based on Automata

As said previously, we model the semantics of an ODRL contract in terms of an automaton. First we outline the construction for ODRL expressions that do not contain **sell** and **lend** permission; an extension that handles these permissions will be given in Sections 3.1 and 3.2.

The states of the automaton implicitly code the "state" of the license, i.e., which actions are allowed at which point in time, considering the actions that have occurred "in the past"; each edge of the automaton specifies an action that can be performed at a specific license state, according to a contract. In addition, there is a special null-action, called τ, which is applicable in (almost) each state. We make the simplifying assumption that each party can only perform one action at a time, where all actions of the party are atomic and take a fixed amount of time (a *tick*). By this convention, the τ action specifies a tick in which no action is performed by the user.

To illustrate this concept, consider Example 1; based on the above mentioned model, we can model its semantics as:

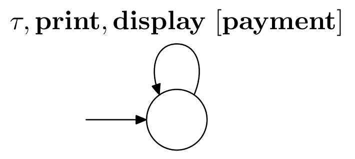


Here, the automaton has only one state and a self-loop. Printing and displaying the document once does not affect the state of the license; therefore, we only need one automaton state to code the permission. It is thus allowed to print and display an object an arbitrary number of times (by our convention, we also allow the null operation τ).

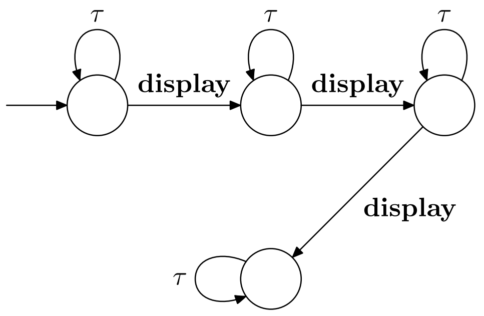
Formally, the actions that are allowed by an automaton (i.e., its "computation") is defined in terms of a labeling function f that assigns each state $s \in S$ of the automaton a finite set of integers, $f : S \mapsto 2^{\mathbb{N}}$. For the moment, assume that each state of the automaton is labeled either with the set $\emptyset$ or the set $\{1\}$; only the modeling of **lend** permissions will require more complex labels. The state that is labeled with a nonempty set is called the active state. Intuitively, the numbers in the labels represent a person. At a specific point in time, a person can only perform the actions that are represented as transitions from an active state that contains its identity (i.e., number) in the label. The construction of the automaton will assure that all labels are mutually disjoint sets.

The labeling function changes with each tick. Initially, only the initial state of the automaton is labeled with $\{1\}$, each other state with $\emptyset$. Whenever the user (encoded with the symbol "1") performs an action, the labeling function changes. In particular, the label $\{1\}$ is erased from the currently active state (it is replaced by $\emptyset$) and applied to the state that can be reached by the edge representing the action. This way we get, for each possible sequence of transitions, an infinite sequence of labeling functions $f_1, f_2, \ldots$ representing the "status" of the license at each point in time.

Requirements and some of the constraints are modeled as labels that are associated to edges; such an edge can only be taken if the annotated constraint or requirement is fulfilled. For example, if we augment Example 1 with a **payment** requirement, we get the following automaton:

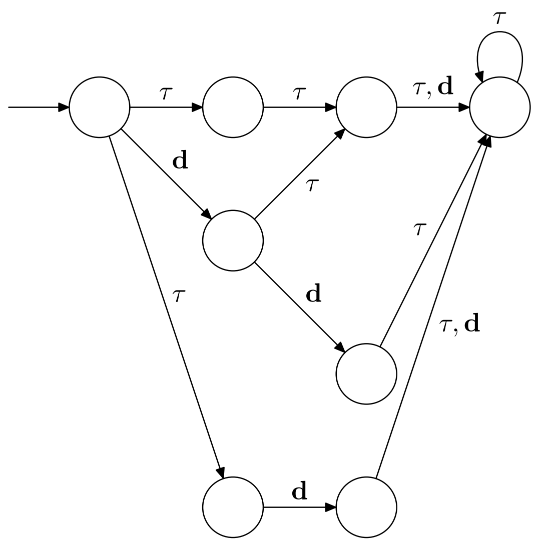


Let us turn to example 2; here, the automaton is more complex, as it requires to count the number of times an object was displayed "in the past." This can be done by using four different states:



Here, the second state can only be reached from the initial state by a **display** action. It is easy to see that from this state each (infinite) sequence of actions that is allowed by the automaton contains at most two other **display** actions, which shows that each path through the automaton contains at most three **display** actions. Note that the license also allows users not to make use of their display rights at all; this is modeled by the self loop in the initial state.

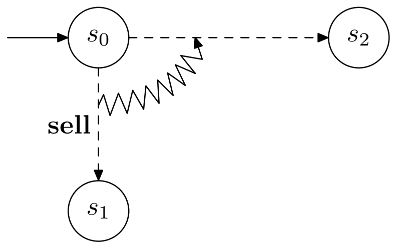
Modeling of time constraints may require to introduce a large number of different states; for example, the following automaton represents Example 3; for space reasons we abbreviate **display** by **d**:



Here, the number of states increases dramatically, as it is necessary to count both the elapsed time as well as the number of times the object was displayed. Again, it is easy to see that the number of display actions on each path throughout the automaton is at most two, as required by the license.

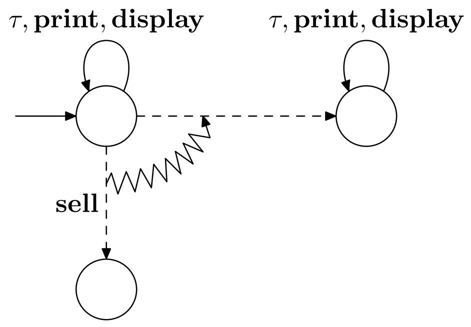
3.1 Extending the model to support SELL and GIVE

So far, we only had to consider one person that is active in the automaton. This changes if we model the **sell** and **give** permissions. Once a person (say, 1) performs a **sell** action, this person loses all rights on the object; the rights are transferred to a *new* person (say, 2). Once person 1 performed the **sell** action, he cannot perform any more actions and is removed from further considerations. In the automaton, both the **sell** and **give** actions are modeled by two dashed transitions, connected with a trigger:



Consider the last figure. The trigger and the dashed transitions have the following intuitive meaning: Suppose that state s_0 is labeled with $\{1\}$, i.e., person 1 is active. If person 1 performs a **sell** action, the label $\{1\}$ disappears in the next tick and is replaced with $\emptyset$ (you can think of 1 moving into the dead-end state s_1, where he is removed, yielding to label $\emptyset$ for s_1). In the same tick, a different person (say, 2) appears in state s_2, yielding to the label $\{2\}$ for s_2. That is, dashed transitions triggered by other transitions introduce a new person, whereas dashed transitions that are not triggered remove a person.

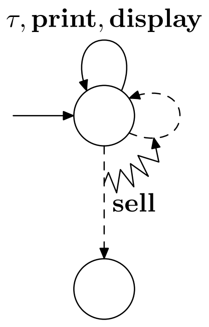
Using this convention, we can model Example 4 as:



τ, **print**, **display**
τ, **print**, **display**
sell

Here, we have a self-loop in the initial state, as the user 1 is allowed to print and display the object an arbitrary number of times. Once he chooses the **sell** action, the user 1 disappears and the triggered transition introduces a new person (2) in the state on the right. This new user can print and display the object, but is not allowed to sell it a second time.

Example 6 differs from Example 4 in the way that the **sell** action may be performed an arbitrary number of times. That is, person 1 can sell the object to person 2, who in turn can sell it to 3, etc. This situation can be modeled by a second self-loop in the initial state that is triggered by the **sell** transition:

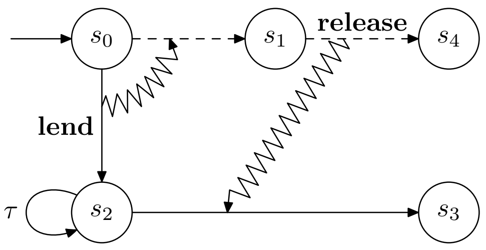


Note that, up to now, the labels of each state contain *at most* the identity of one person.

3.2 Extending the model to support LEND and LEASE

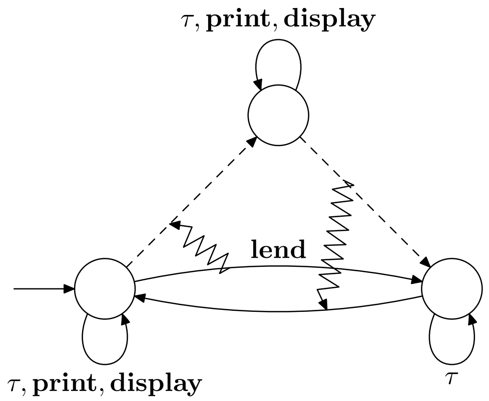
It turns out that the **lend** and **lease** permissions are more complex to handle semantically; in our semantics, both **lend** and **lease** are modeled in the same manner. In contrast to **sell**, where the identity of a person is removed from the labels after a **sell** action has occurred, this cannot be done in case of **lend** or **sell**, as the person can perform other actions after the object is returned.

We model this situation with two triggers:



Suppose person 1 is in state s_0 (i.e., s_0 is labeled with $\{1\}$) and performs a **lend** action; now, 1 moves into state s_2, where it loops without action (note that there is only the null action that can be performed by 1) until the object is returned. The **lend** action triggers a dashed transition, indicating that a new person (say, 2) is created, whose identity is placed in state s_1 (now, state s_1 is labeled with $\{2\}$). This new person can use the object until it gives it back to the original user; in the automaton, we model this action by the operation **release**. Once person 2 returns the object, its identity is destroyed, as indicated by the dashed transition between states s_1 and s_4. The **release** action triggers a transition that "frees" person 1 from state s_1 (the identity of 1 moves to a different state, in this case s_3).

Using this convention, we can model Example 5 in the following manner:



τ, **print**, **display**
lend
τ, **print**, **display**
τ

In fact, this is only half of the truth, since arbitrarily long chains of **lend** actions cannot be modeled this way. Consider the following example: person 1 lends an object to person 2, who in turn lends it to person 3, etc. Now, as the object must be returned to the person that initiated the **lend** operation, 3 must return it to 2 who can eventually give the object back to 1. If there is no upper bound on the lend operations, it is not possible to model this behavior by finite automata (intuitively, we had to construct an automaton that accepts the word ww^R, where w^R denotes the mirror image of w, which is impossible). In order to overcome this situation, we propose to introduce a pushdown (stack) that controls the update of the labeling function when the object is returned. Returning to the above example, when person 1 lends the object to 2, the number 1 is stored (with some appropriate additional information) on the pushdown; the same is done if 2 lends to 3. When 3 gives the object back, the top element of the pushdown holds the necessary information for the user 2 to be activated. This topic is subject of current research.

4 Conclusion

We have shown in this paper that finite-automaton like structures are a promising tool to formally define the semantics of rights expression languages. We have seen that most ODRL expressions can intuitively be modeled as automata. However, the **lend** permission turns out to be the most complex operation in ODRL; to model infinite chains of model operations, one has to go beyond finite automata.

We believe that the automata resulting out of our construction can be used to intuitively visualize the meaning of an ODRL expression. In addition, once automata can be constructed automatically from ODRL expressions, it becomes possible to verify properties of the ODRL contract by logics like CTL or LTL that are commonly used in current verification software [1].

References

[1] E. Clarke, O. Grumberg, D. Peled, *Model Checking*, MIT Press, 1999.

[2] R. Ianello, *Open Digital Rights Language (ODRL)*, Specification, Version 1.1, available at `www.odrl.net`, 2002.

[3] R. Pucella and V. Weissman, “A Logic For Reasoning about Digital Rights”, Proceedings of the Computer Security Foundations Workshop, 2002.

[4] R. Pucella, V. Weissman, “A Formal Foundation for ODRL”, in Workshop on Issues in the Theory of Security (WITS), 2004.

[5] *eXtensible rights Markup Language (XrML) 2.0*, Specification, available at `www.xrml.org`, 2001.