Chapter 7

Case Study of the Rights Expression Exchange Framework

This chapter presents the application of the rights expression exchange framework that has been designed and implemented in this thesis (see Chapters 5 and Section 6). Normally, not all four components of the framework are located on the same machine. Usually, the rights expression generator and the wrapper are located on the sender's platform and the unwrapper and interpreter are running on the receiver's platform. However, due to the component-based approach of the rights expression exchange framework, there can be scenarios where only the interpreter is used and the other components are replaced by foreign components.

Generally, the framework is used for the exchange of rights expressions which state use or access rights of people to digital goods or services. The exchange of these rights expressions takes place between two or more interoperating (trusted) partners. There are various general scenarios in which rights expressions need to be exchanged, for example:

1. A consumer has concluded a contract with the marketer X of certain services (communication or multimedia). For example, the contract allows to use different service providers for sending SMS, faxes, emails, MMS, ring tones, logos, etc. via the Internet for a certain time, or to access online platforms to play videos, music files, games, etc. within
the next month. Each time the consumer desires to access a service s/he has to present the contract to the service provider. The service providers only accept contracts that have been concluded with certain marketers, e.g. marketer X. The rights expression exchange framework can provide all functionalities that are necessary for this scenario: formulating the contract and signing it is handled by the generator and the wrapper that are running on the marketer's system. Each service provider is runs the unwrapper and interpreter component to verify the marketer's signature on the contract in order to subsequently interpret the consumer's access rights. The access to the communication respectively multimedia services is then either granted or denied, accordingly to the rights expressions in the contract.

2. User A has certain access rights on platform A or system A (e.g. to the Intranet of a company). Platform A starts a cooperation with Platform B (e.g. a company merges with a second one). From this follows that all users of platform A shall receive the same rights on platform B. For all users electronic tickets (see Section 4.3.1) comprising their personal access rights are issued and transmitted to platform B. Platform B interprets the rights and implements them in their own access control mechanism. In this case, it might be necessary to use a expression generator which is different form that implemented in this thesis, e.g. one that generates rights expressions automatically. However, the wrapper, unwrapper, and interpreter from my implementation could be used for the remaining tasks of the rights expression exchange.

3. A person has a certain role in society or in a community, e.g. that of a student or a pensioner. An commonly accepted institution (e.g. a university or national office) certifies these roles to the respective persons in form of an identification (e.g. a student or pensioner id). Displaying this id entitles these people to receive certain access or usage rights, e.g. going to the library for free, or paying a reduced price for theater tickets. The role in the (electronic) id and possibly some constraints (e.g. expiration) are formulated and signed by the rights expression generator respectively the wrapper running at the certification institution. The providers that accept the ids need an unwrapper and interpreter to verify authenticity of the id and to extract the role from the id. Subsequently, each provider can grant different access or usage rights to the id holders. Note that the extrinsic format of such an id can be an x509 certificate and the rights expression can be stored in the extension field of the certificate.
The above examples display some basic shapes of rights expression exchange scenarios. The rights expression exchange framework supports the drawbacks of today's electronic commerce systems listed in Section 1.1, e.g. the standardised representation of rights expressions, more precisely of contracts, the usage of one contract at various platforms, the processing of rights expressions in various software services, the expression and enforcement of new and rich usage variants for electronic goods, etc. Rights expressions can be processed in various usage scenarios (see Section 4.4.2). As the particular focus of this work is on the usage scenario access control, Section 7.1 gives a short introduction to the processing of access requests by an access control mechanism. The section also addresses the handling of context constraints for usage rights (e.g. play video five times) that are frequently used in new pricing models for electronic goods. Section 7.2, describes the required rights expression exchange components and their interaction in the technical realisation of the usage scenario access control based on electronic tickets, i.e. the scenario 2 described above.

7.1 Access Control with Context Constraints

A typical example of a rights expression that occurs in DRM systems is "Person X may play the video Y five times on his/her own computer (IP address 137.224.208.84) until the end of this year." To enforce such a rights expression, the access control mechanism has to be able to handle context constraints (in the following called constraints). Constraints narrow access permissions, for example by time, location, individual, etc. (please confer also to Sections 4.5 and 3.3.1). In the example the permission play the video Y is constrained by location ("his/her own computer"), time ("until the end of this year"), and the number of accesses ("five times").

Handling constraints requires the implementation of constraints and their evaluation at the time of an access request. A constraint is a clause that contains one or more context conditions. A constraint is satisfied iff all its context conditions are met [Str03]. A context condition is a predicate (a boolean function) that consists of an operator and two operands, e.g. \( \langle \text{date, } <, \text{12/31/2004} \rangle \). Only if all conditions of a constraint hold, i.e. the overall evaluation of conditions returns "true", the respective permission is granted. Note that the first (or left) operand always represents a certain context attribute (i.e. a property of the environment, such as date), while the second operand may be either a context attribute (e.g. end-date) or a
constant value (e.g. 12/31/2004). To evaluate the expression in the previous paragraph the access control mechanism requires the current date, the local computer, and the number of times the video Y has already been played by person X. For this purpose, the access control mechanism uses of sensors that return current values for the context condition attributes. For the work at hand, two categories of attributes are be distinguished; environmental attributes and attributes administered by a DRM system database(also see to Section 4.7):

- **DRM System Database.** DRM systems observe all activities that are relevant for the overall rights management in the DRM system database, e.g. how many times a user has accessed a certain resource. Context conditions that define a certain number of usages are very common in DRM applications. To evaluate such conditions the sensors that retrieve the relevant information and DRM system databases have to be available.

- **Environmental Attributes.** Environmental attributes attributes whose values are not administered by the DRM system. These attributes exist independently of DRM systems, rights expressions or contracts. Examples are time, date, IP address, weather conditions, etc. For each required environmental attribute an adequate sensor has to be available to the access control mechanism.

Both categories may include static and dynamic attributes, for example IP address and date (environmental), respectively the user’s birth date and the number of access to a certain resource (DRM system knowledge). In section 5.2.1 we have learned that after decoding, the rights expression is available for processing. This section is focused on processing and enforcing the rights expressions in an access control service. Note that the following example is independent of any access control approach. Figure 7.1 depicts the relevant steps that are performed when processing rights expressions in the usage scenario access control:

1. **Implement Rights Expression.** In the first step, the rights expressions are retrieved from the rights repository and ”fed” to the access control mechanism, i.e. the parties, permissions, and constraints (e.g. person X; play video Y; five times, until the end of the year, on the computer with the IP address 137.224.208.84) are mapped to corresponding policies rules that can be enforced by the access control mechanism. Technically, the permission ”play video Y” and the subject X are created.
The permission is assigned to person X. The three constraints (date, IP address, maximum number of accesses) are created and assigned to the permission "play video Y" of person X. The resulting access control policy is depicted in Figure 7.2. This transformation is not performed automatically, but has to be supported by a mediator (see Section 6.5).

2. Access Request. Person X now aims at executing a certain permission (e.g. play video Y) and triggers the respective access request. The decision component has to evaluate the incoming access request and either grant or deny it. First, the access control mechanism queries the implemented policies for the permission play video Y of Person X. If person X does not hold this permission, the decision component denies the request. In the above example the permission play video Y is available, but with related conditions.
3. **Sensor Consultation.** To verify time, location, and count conditions, the decision component consults external sensors that are capable of delivering values for the boolean functions (conditions). To ascertain the current time, and the local IP address the decision component consults the environment sensors. For the number of times, that the video Y has already been accessed the DRM system database has to be queried.

4. **Provide State Information.** The respective sensors return the required context information, but do not make any decisions. Let us assume that the current date is 11/11/2004, the IP address of person X’s computer is 137.224.208.84, and the video has not been accessed before. *Context functions* [Str03] transform the received values into a readable format for the access control decision component.

5. **Return Evaluation Result.** The decision component receives the context information of the sensors and inserts them into the boolean functions. With the state information received above all boolean functions return true, and the access request can be granted. If one function returns "false", the access to the video Y is denied.

**The Access Control Mechanism xoRBAC**

xoRBAC [NS01, NS03a] is an access control mechanism that, among other things, supports the handling of context constraints. It is used as access control component in the subsequent framework application example. xoRBAC provides a role-based access control (RBAC) service that can be used on Unix and Windows systems in applications providing C or Tcl linkage. xoRBAC is well-suited to be used within a component framework. While originally developed as an RBAC service, xoRBAC was extended to provide a multi-policy access control system which can enforce RBAC-, as well as DAC- (discretionary access control) or MAC- (mandatory access control) based policies including *conditional permissions*. With respect to this thesis, the dynamic constraint management subsystem is of central significance. It comprises the *environment mapping*, which captures context information via sensors, and the *constraint evaluation*, which checks if the collected values match the context constraints associated with a certain conditional permission. Thus, it allows for the definition and enforcement of context constraints.
The subsequent paragraphs, describe features of xoRBAC that are necessary to enforce the access control policies with context constraints as described above. A **context constraint** is defined through the terms context attribute, context function, and context condition:

![Diagram](image)

**Figure 7.3: xoRBAC access control decisions with context constraints**

- A **context attribute** represents a certain property of the environment whose actual value might change dynamically (like time, date, or session-date), or for different instances of the same abstract entity (e.g. location, ownership, birthday, or nationality). Thus, context attributes are a means to make (exogenous) context information explicit.
On the programming level each context attribute $CA$ represents a variable that is associated with a domain $domain_{CA}$ which determines the type and range of values this attribute may take (e.g. date, real, integer, string).

- A **context function** is a mechanism to obtain the current value of a specific context attribute (i.e. to explicitly capture context information). For example, a function $date()$ could be defined to return the current date. Of course a context function can also receive one or more input parameters. For example, a function $age(subject)$ may take the subject name out of the (subject, operation, object) triple to find out the age of the subject which initiated the current access request, e.g. the age can be acquired from some database.

- A **context condition** is a predicate (a Boolean function) that compares the current value of a context attribute either with a predefined constant or another context attribute of the same domain. The corresponding comparison operator must be an operator that is defined for the respective domain. All variables must be ground before evaluation. Therefore, each context attribute is replaced with a constant value by using the according context function prior to the evaluation of the respective condition. Examples for context conditions are $cond_1: date() \leq "2003/01/01"$, $cond_2: date() == birthday(subject)$, or $cond_3: age(subject) > 21$.

- A **context constraint** is a clause containing one or more context conditions. It is satisfied iff all its context conditions are met. Otherwise it returns false.

With respect to the terms defined above, a **conditional permission** is a permission that is associated with one or more context constraints and grants access only if each corresponding context constraints evaluates to "true". Figure 7.3 shows a message sequence chart for access control decisions in xORBAC including conditional permissions. For a detailed description of xORBAC see [NS01, NS03a].

### 7.2 Access Control Decision Based on Electronic Tickets

The rights expression exchange framework usage exemplified in this section handles the following scenario: Platform A generates and digitally signs a
license that grants certain access rights to user M. Strembeck on a various platforms within a network, e.g. platform B. Platform A sends out the valid license to M. Strembeck. If Mr Strembeck desires access rights on Platform B he chooses the rights license and presents it to platform B to receive access rights. Platform B unwraps and interprets the license and grants or denies the user's access request accordingly to the permissions in the license." This technical use case support a number of higher level applications, e.g. the example, described as item 1 at the beginning of this chapter, but also the following classical DRM application: A DRM platform issues a license that grants access rights to a resource with certain constraints. The platform delivers the license (with or without the resource) in a secure container (see Section 2.3) to the consumer. The secure viewer receives the license and subsequently handles the access request to the resource (that is either in the container or locally stored) and, according to the rights in the license, either renders the resource or not.

The subsequent sections apply the concepts and implemented components of this doctoral thesis to the DRM scenario described above. Section 7.2.1 develops the application-specific CoSa that comprises all necessary contract objects for the scenario. In Section 7.2.2 the adequate licens is created with the rights expression generator. Subsequently, the license is wrapped (see Section 7.2.3). The last section, Section 7.2.4 describes the unwrapping, interpretation and processing of the license. Here, the unwrapper and interpreter that have been developed in this thesis are deployed as well as the access control mechanism (xORBAC).

7.2.1 Application-Specific CoSa

For the scenario mentioned above the required objects in the license are Party, Resource, Permission and Constraint. It is assumed that the DRM system is designed to operate with discretionary access control, as licenses are rather issued to individuals than to roles. Therefore, the element Role is not required in this DRM application, nor is the element Duty. Therefore, for this DRM application the implemented CoSa is satisfactory, yet only the subset of those shown in Figure 7.4 is used. The Party objects basically require a unique id; additional attributes are optional. Similarly, the Resource objects require a value in the attribute "Identifier". The Permission objects must comprise the operation (e.g. display) and the object (resource) the operation refers to. Each permission can be constrained by one ore more conditions. Constraint objects comprise the instance variables name, operator, value, attribute, and type (see also Section 6.3.3).
Please note that the rights expression exchange framework is not responsible for the attribute values, i.e. the data in the license. It is assumed that the numbering systems for user ids and resource ids are aligned between platform A and B and that stated access rights, as well as their constraints, are either globally unique or well-understood by platform B. The possible access rights are dependent on the capabilities of the rendering software (e.g. the secure viewer). If the rendering software is able to process the rights ”display”, ”preview”, and ”copy”, then a license comprising the operation ”give” will cause an error during rendering. The same is also true for constraints.

7.2.2 Generating DRM-Specific Licenses

According to the developed CoSa, tailored license templates should be provided to the person who fills the license with data (see Section 4.5 and 4.5.3). With the rights expression generator implemented in this thesis, the respective 'empty' templates could be created by the system administrator and stored in the license repository. Users of the generator would restore such licenses and fill them with ids, permissions, and constraints. Figure 7.5 shows, among other rights expressions, a predefined license template for the operation display (license no. 2). Clicking on the modify link would restore the license template and allow value entries by the current user. Storing it again allows to store the copy under a different name (license no. 1).
Figure 7.5: Provide license templates with the generator

The resulting ODRL rights expression contains a license, sometimes also called digital ticket (see Section 4.3.1). The license that has been just formulated with the rights expression generator has the following ODRL version 1.1 conform XML code:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<o-ex:rights xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
xmlns:o-dd="http://odrl.net/1.1/ODRL-DD">
  <o-ex:asset>
    <o-ex:context>
      <o-dd:uid>s Guth-9999</o-dd:uid>
      <o-dd:name>Ebook on the framework design</o-dd:name>
    </o-ex:context>
  </o-ex:asset>
  <o-ex:party>
    <o-ex:context>
      <o-dd:uid>m strem</o-dd:uid>
      <o-dd:name>M. Strembeck</o-dd:name>
    </o-ex:context>
  </o-ex:party>
  <o-ex:permission>
    <o-dd:display>
      <o-ex:constraint>
        <o-dd:datetime>
          <o-dd:end>2004-12-31T00:00:00</o-dd:end>
        </o-dd:datetime>
      </o-ex:constraint>
    </o-ex:display>
  </o-ex:permission>
</o-ex:rights>
```

ODRL is defined in two linked XML schemata, the ODRL grammar (prefix: o-ex), and the ODRL data dictionary (prefix: o-dd)
listing grants the rights \textit{display} to the asset \texttt{aguth-9999} with the title \textit{Ebook on the framework design} to the party \textit{M. Strembeck} with the (locally) unique id \texttt{matrem}. The operation \textit{display} is narrowed by the following constraint: \textit{display} may only be executed if the current date one before 12/31/2004.

\section*{7.2.3 Wrapping DRM Licenses}

Figure 7.5 shows two licenses in the current repository. License no. 1 is a ready-to-use license (see XML serialisation above) that can now be wrapped and transmitted to DRM platform B. In the very right section two actions can be performed with the licenses: \textit{Modify} and \textit{Sign}. The sign action is relevant for the license wrapping. The wrapping functionality is performed by the rights expression exchange framework package \texttt{reWrapper} described in Section 6.4 and is here provided to the user. To integrate the wrapper functionality into the rights expression generator, the \texttt{reWrapper} package has been included into the generator package. The mediator code in this case is written in the extra method \texttt{wrap(args)} of the \texttt{0DRLHttpd:Wrk} that uses an instance of the class \texttt{Wrapper}. The method restores the XML code of the respective license from the repository, signs it with the private key of platform A, packs the license into an archive, and (currently) stores it in a folder, which the creator of the license has access to. Please note that the method \texttt{wrap(\ldots)} serves as a Facade \cite{GHVJ94} for the rights expression wrapper, which subsequently calls all wrapping functions that are adequate for this application. The program in Figure 7.6 below shows extracts from the mediator code within the generator package. The license creator on Platform A sends the wrapped license to M. Strembeck via a secure connection, i.e. HTTP over secure socket layer (SSL) \cite{FKK96}.

\section*{7.2.4 Unwrapping, Interpreting and Processing DRM Licenses}

Let us assume that some day M. Strembeck desires to access the resource "Ebook on the framework design" that is stored on platform B. Therefore, he authenticates himself to platform B with user id "mstrem" and password via \textit{HTTP Basic Authentication} \cite{FGM+99, FHBH+99}. Then he uses an HTML form to specify the resource he desires to access (e.g. \textit{display}) and to upload his license. Again, the the HTTP connection is secured by SSL. Now, the license has to be processed and the access request has to be evaluated, i.e. either granted or denied. According to the rights expression communication model developed in Section 5.1.2, the processing of the license proceeds with
Figure 7.6: Mediator code combining generator and wrapper functionality

unwrapping and interpreting the license. All involved activities from the rights expression exchange framework and environmental components are sketched by the activity diagram in Figure 7.7. The different actors are:

- The **beneficiary**, (or a corresponding client program) who requests a service respectively the access to digital goods and presents a digital license.

- In this case study the **secure viewer** is the **mediator** (see Section 6.5) that coordinates and controls the beneficiary interaction, the rights expression unwrapper and interpreter, as well as the access control service.

- The rights expression **unwrapper** (see Section 6.4) that performs encoding and validity checks on the license.

- The rights expression **interpreter** (see Section 6.3) that parses and interprets the license, and builds a runtime model of the application-specific CoSa.

- The **access control service** which decides if the beneficiary may perform the requested operations according to the permissions granted through the presented license.
Figure 7.7: Sequence diagram with basic activities of the secure viewer

The contract processing procedure is triggered by the access request of the beneficiary via the HTML form. An access request expresses the demand to perform a specific operation/action on a particular object/resource. In this case study approach, each access request consists of a method call with four parameters (subject, operation, object, contract-location) where the subject is mstrem, the operation is display, the object (requested resource) is sguth-9999 and the contract-location states concrete location of the license that has been uploaded by the beneficiary. The license is fetched by the contract-id via the lookup procedure of the secure viewer. After the contract has been loaded the unwrapper component performs the following activities:

- **Unpack License**: unpacks the license from the archive.
- **Check Signature**: verifies the digital signatures of the corresponding license issuer, here platform A, to ensure its integrity and authenticity. The public key of platform A and the information of the applied hash algorithm is required for the signature check. In this case study the public key of platform A is the default key for signature verifications, and the algorithm SHA1 [EJ01] is used as default hash algorithm by the wrapper and unwrapper component.

Please note that the unwrapper serves as a Facade [GHVJ94] which subsequently calls all unwrapping functions that are adequate for this application (here: unpacking and signature verification). After the license has successfully passed all checks, it is considered to be *valid* (see Section 4.8) and the wrapper returns the license in plain text. The license is now forwarded to the rights expression interpreter. If the license fails one of the unwrapper checks, the license is invalid and the access requests is automatically denied.

The interpreter parses the license, extracts all relevant information, and builds a runtime model of the application-specific CoSa. The runtime model is shown in Figure 7.8. It consists of several objects, that are all aggregated by the instance co01 which is of the *Contract*. The reminders of contract objects are py01 (type *Party*), re01 (type *Resource*), p01 (type *Permission*), and c01 (type *Permission*). The relations among the different runtime objects are expressed via the intrinsic attribute *relations* (see Section 4.6). For example, the party py01 is related to the objects contract co01 and permission p01 via the roles *agg_parent* and *has_perm*. Whereas the permission p01 is related to the objects co01, py01, re01, and c01 via the roles *agg_parent*, *granted_to*, *refers_to*, respectively *has_constraint*. For a complete list of possible roles, please refer to Section 6.3.2.

The runtime model may then be queried for the contract objects and their respective attributes, e.g. the unique ID of the beneficiary, his/her roles, the resources, the granted permissions to these resources, the constraints that apply to the granted permissions, etc. In the next step, the secure viewer component extracts the access control-relevant contract information from the runtime model of the contract to initialise the corresponding access control service. Finally, the secure viewer calls the method `checkAccessRequest(...)` to handle the access request of the beneficiary, i.e. it evaluates whether the requested access rights (display the resource

---

2This allows to delete all contract objects if the contract itself is erased
sguth-9999) are granted in the ODRL license. The implementation, uses the xoRBAC access control service, which is described in Section 7.1. The access control mechanism grants the access requests, and therefore the secure viewer display the respective Ebook to M. Strembeck. The XOTcl code that implements the unwrapping, interpreting, and processing of the ODRL license is listed below.

#!/usr/local/bin/xotclsh

package require xoRBAC 0.6.1
package require rex::reInterpreter 0.1
package require rex::relContract 0.1
package require rex::reUnwrapper 0.1

Figure 7.8: Runtime model of the DRM CoSa objects
Class SecureViewer

SecureViewer instproc requestAccess {subject operation object contract-location} {

    Unwrapper cc ${contract-location}
    set contract [cc unpack]
    set valid [cc verifySignature]

    if {($valid == "true")} {
        ODRLContract op $contract
        RightsManager rm
        foreach c $contracts {
            set assets [op getAssets $c]
            set parties [op getRelatedObjects $c agg_child CoSaParty]
            set consumers [op selectObjects $parties ROLE "consumer"]

            foreach asset $assets {
                set assetId [op getAttributeValue $asset uid]

                foreach con $consumers {
                    set conID [op getAttributeValue $con uid]
                    rm createSubject $conID
                    set rights [op getRelatedObjects $con rPerms]

                    foreach r $rights {
                        set right [op getAttributeValue $r name]
                        rm createPermission "$right $assetId"
                        rm subjectPermAssign "$conID" "$right $assetId"
                        set constraints [op getRelatedObjects $r rConstr]

                        foreach constr $constraints {
                            set cname [op getAttributeValue $constr name]
                            set cvalue [op getAttributeValue $constr value]
                            set cop [op getAttributeValue $constr operator]
                            if { $cname == "datetime"} {
                                # this secure viewer can only handle time constraints

                                rm createCondition $cname
                                rm setConditionLeftOperand $cname "LocalhostSensor"
                                "LocalhostSensor" "%Y-%m-%dT%H:%M:%S"
                                rm setConditionOperator $cname $cop
                                rm setConditionRightOperandAsConstant $cname $cvalue
                                rm buildConditionScript $cname

                                rm createContextConstraint "$cname" "$cvalue"
                                rm addConditionToContextConstraint "$cname" "$cvalue"
                                rm addContextConstraintToPerm "$cname" "$cvalue" "$right $assetId"
                            } else {

                                # continue processing...
                            }
                        }
                    }
                }
            }
        }
    }
}
Evaluation of the Case Study

The rights expression interpreter is coded in approximately 3000 lines of XOTcl code, whereas the unwrapper component was implemented with approximately 500 lines of XOTcl code. Consequently, the be above 70 lines represent the functionality coded into the 3500 lines of the unwrapper and interpreter package (not including the xoRBAC code). The secure viewer above was run with the following performance:

<table>
<thead>
<tr>
<th>party</th>
<th>user</th>
<th>system</th>
<th>real</th>
</tr>
</thead>
<tbody>
<tr>
<td>elapsed time</td>
<td>0m0.310s</td>
<td>0m0.020s</td>
<td>0m0.359s</td>
</tr>
</tbody>
</table>

Here, the user time is the time the secure viewer is running, the system time is the time spend in system calls and real time is the total time the secure viewer has been running.

It is planned to make the source code of all components of the rights expression exchange framework freely available at the XOTcl web site\(^3\) as well as on the web site of the ODRL initiative\(^4\). In terms of scalability, the framework can be extended to support a different application-specific CoSa and/or other metadata standards for resources and parties (e.g. LOM instead of Dublin Core) at low expense. The support of another rights expression language such as XrML, in return, would be more costly. In order to support the processing of more context constraints the mediator and/or the sensors of the access control service have to be extended.

---

\(^3\)See: http://www.xotcl.org/

\(^4\)See: http://odrl.net/
As mentioned earlier, it was a challenge to read the detailed semantics of the rights expression language ODRL from the written specification. Renato Iannella, the founder of the ODRL initiative, was very supportive in this matter. The development of rights expression language interpreters is at the very beginning. Apart from the work at hand, no design or comprehensive implementation of a rights expression language interpreter or rights expression generator is available. The most challenging issues of the generator implementation were: to design a comfortable, intuitive user interface, and to design a sensible policy for restricting the nestings of ODRL expressions.