Chapter 5

Design of a Rights Expression Exchange Framework

This thesis has the objective to develop methods and tools that support the exchange and processing of rights expressions (see Section 1.3). Rights expressions are valid instances of a rights expression language (see Section 3.1). Offers, contracts, and licenses formulated in a REL are specific rights expressions, i.e. instances with certain semantics in the contract life cycle (see Section 4.1). Whereas Chapter 4 has covered the contract (respectively rights expression) content, and its tailored composition that support later processing, this chapter focuses on the exchange practices of rights expressions. The first section introduces a general communication model and deploys its principles to the exchange of rights expressions (Section 5.1) which results in the rights expression communication model. A technical design for the implementation of such a rights expression communication model – the rights expression exchange framework – is presented in Section 5.2. Here, the framework’s functional and technical perspectives (see Section 2.2.2) are discussed in detail.

5.1 Exchanging Rights Expressions

In the following sections an outline is given of the exchange of rights expressions between sender and receiver. Therefore, Section 5.1.1 introduces
the general communication model, which is later adopted to the communication via rights expressions (see Section 5.1.2). Section 5.2.1 describes the component-based technical design of a rights expression exchange framework. Finally, in Section 5.2.2, a check list of technical requirements for such a rights expression exchange framework is provided.

5.1.1 The Communication Model

Basic components of communication can be found in the communication model [Sch71] of Schramm illustrated in Figure 5.1. According to Schramm, communication is the sharing of an orientation toward a set of informational signs ... where both source and destination must share a field of experience. Constituents of Schramm's communication model are the source, the encoder, the signal, the decoder, and the destination.

![Diagram of the communication model](image)

Figure 5.1: The communication model [Sch71]

- **Source.** According to Schramm's model, a message begins at a source. The source can be a person or computer system that aims at sending a message to a destination, where a person or a computer receives this message.

- **Encoder.** The communication encoder is responsible for taking the ideas of the source and putting them in code, expressing the source's information in the form of a message. Therefore, the encoder has to choose a coding language that is understandable for the destination.

- **Signal.** The signal is the encoded message, an instance of a language that source and destination are familiar with, which is sent through a communication channel towards the destination.

- **Decoder.** Like a source needs an encoder to translate the original message into a transportable message, the destination needs a decoder to retranslate. To decode the message the decoder needs to understand the language in which the signal has been coded. After decoding, the message is identical with the earlier-sent original.
• *Destination.* For a successful communication, there must be somebody at the other end of the channel. This person or computer system can be called the destination. After decoding, the destination is able to understand and process the message, and possibly send a reply on the message.

Schramm's communication model describes communication in general. In the next section, Schramm's model is deployed to the exchange of rights expressions between two computer systems.

### 5.1.2 The Rights Expression Communication Model

When exchanging rights expressions the communication between two cooperating computer systems corresponds to the basic communication model. In this section the communication model is adopted to the communication via rights expressions between two DRM system components, resulting in the *rights expression communication model.* In the following, the required steps are identified which facilitate the exchange of rights expressions between two computer systems. It is assumed that after exchanging the rights expressions, each system is capable of processing them internally, for example, in an access control service context, in which permissions are assigned to parties or in an accounting service context, relating parties to monetary duties.

The rights expression communication model has five stages: the rights expression source, the rights expression encoder, the message in form of a rights expression, the rights expression decoder, and the rights expression destination (see Figure 5.2). The communication takes place between two

![Diagram](image-url)

**Figure 5.2: The rights expression communication model**
DRM systems or DRM system components (see Section 2.3). A DRM component in this case is a DRM subsystem, such as the DRM platform or the DRM client (e.g. secure viewer).

- **The Rights Expression Source.** In the rights expression communication model a DRM system or a person is the origin of the the rights information to be delivered, i.e. the rights expression source.

- **The Rights Expression Encoder.** This stage provides three tasks: the message generation, the message wrapping, and the application of the respective protocol stack:
  
  - **The Generator.** The generator supports the coding of the rights expression source into a rights expression. The rights expression can either be generated by the DRM system or be provided by a human actor (e.g. the content provider). The rights expression is formulated in a rights expression language (see Chapter 3) that has to be well-known by the receiving DRM component in order to be understood.

  - **The Rights Expression Wrapper.** As rights expressions comprise sensitive information, they are usually not transmitted in plain text and/or without protection. The rights expression wrapper "wraps" the rights expression and thus provides confidentiality and integrity of the rights expression, as well as authentication and verification of the rights expression issuer or signer. Here, the term "wrapper" is not to be understood as the wrapper or object adapter, described in [GHVJ94].

  - **The Protocol Stack Application.** Referring to the TCP/IP protocol stack, the generator and wrapper are encodings on the application layer. To be sent over a network (e.g. the Internet), the protocols of lower layers, i.e. the transport layer (e.g. TCP [Pos81b] or UDP [Pos80]) and the internet layer (e.g. IP [Pos81a]) have to be applied to the rights expression. The protocol stack to be applied depends on the actual network and on the communication partner. Applying the protocol stack produces the rights expression message.

- **The Rights Expression Message.** The rights expression message comprises a wrapped REL instance that is sent to the receiving DRM component via a communication channel, i.e. the rights expression massage is a transport format for the encoded rights expression source.
The message can be transported via any kind of network, e.g. the Internet, a virtual private network (VPN), but also a smart card or floppy disc.

- **The Rights Expression Decoder.** This stage provides three tasks: the reverse application of the protocol stack, the message unwrapping, and the message interpretation:

  - *The Reverse Application of Protocol Stack.* This encoding task makes the wrapped rights expression available for the application layer, i.e. the unwrapper and interpreter.

  - *The Rights Expression Unwrapper.* The rights expression unwrapper is the complementary component to the rights expression wrapper. It unwraps the rights expression. This component also provides for the extrinsic checking of the digital contract, such as checking the contract integrity and the contract signature.

  - *The Interpreter.* The interpreter transforms the encoded rights expression, formulated in the rights expression language, back to the original rights information. This rights information can then be processed in the receiving DRM system component in various ways. The interpreter that has been implemented within the scope of this thesis stores the original information in a generic contract schema that is a means for flexible and open processing of rights information. To read more about the generic contract schema, please refer to Section 4.6.

- **The Rights Expression Destination.** The receiving DRM component is the destination of the rights expression, where it can be processed in various software services (see Section 4.4.2).

### 5.2 The Rights Expression Exchange Framework

This section provides a general technical design for a rights expression exchange framework and a check list for its implementation. The rights expression exchange framework has been derived from the rights expression communication model defined in Section 5.1.2. Such a framework facilitates the exchange of rights information between two dislocated computer
systems. The framework description is independent from a concrete technical implementation.

### 5.2.1 Technical Design

The rights expression exchange framework presented in this section is a framework that facilitates the exchange of rights information between two or more components, and is based on the rights expression communication model (see Section 5.1.2). The framework design deploys a component-based approach, i.e. a set of cooperating components that offer the necessary functions for rights expression exchange. Each of the components is autonomous and can be reused in other frameworks; new components can be easily added. A rights expression exchange framework is intended as additional module in a web server or application server and can be easily integrated into an existing software environment. In general, we propose a dynamically extensible component framework, as discussed in [GNZ00].

According to the rights expression communication model, the following components that are required for a rights expression exchange framework can be derived: the rights expression generator, the rights expression wrapper, the rights expression unwrapper, the rights expression interpreter, and the mediator. The generator and the wrapper are designed to encode rights expressions; the unwrapper and interpreter are designed to decode rights expressions (see Figure 5.3). Considering the four–layer protocol TCP/IP, the four components are resident on the application layer. It is assumed that the protocol stack below the application layer, such as TCP/IP [Pos81b, Pos81a] is available to the framework. The framework interacts with other software services, such as web servers or application servers, and reuses a database service for the (temporary) storage of rights expressions.

- **Rights Expression Generator.** The product of the rights expression generator is a schema and specification–conforming instance of a rights expression language. The rights expression generator has to know the syntax and the semantics of each REL it is supposed to create instances of. Coding is necessary for each new REL the generator adopts. Rights expressions can be created by machines or by human actors. Often rights expression, such as offers, licenses, contracts are created by human actors. Therefore, the rights expression generator component can be implemented as graphical user interface that guides the user through the generation process. If rights expressions and in particular electronic contracts undergo a tailored composition (see
Section 4.5), the number and type of application-specific objects and attributes are identified. A rights expression generator has to support the creation of such tailored contract templates. Completed rights expressions and rights expression templates can be (temporarily) stored in a database, the rights expression repository. The implementation and usage of a rights expression generator is described in Section 6.2 respectively in Section 7.2.2.

- **Rights Expression Wrapper.** The rights expression wrapper component is in charge of a further encoding of the rights expression that has been created by the rights expression generator. The wrapper applies technical means to the rights expression to provide various security services (such as listed in Section 5.1.2). The required security means to be applied (encrypting the rights expression or digitally signing the rights expression) depend on the software service that later processes the rights expression. Thus, the rights expression wrapper has to be customised and possibly extended for its actual application. The wrapper respectively all its methods that perform security applications can be controlled via its application programming interface. The wrapper receives and returns the rights expression from/to the mediator component (see below). The implementation and the usage
of a rights expression wrapper is described in Section 6.4 respectively in Section 7.2.3.

- **Rights Expression Unwrapper.** The rights expression unwrapper component is in charge of decoding all applied security means, and thus is the reverse functionality of the wrapper component. Here, the rights expression is decrypted and digital signatures are verified. Again, the security checks that have to be applied depend on the software service that processes the rights expression. Thus, also the rights expression unwrapper has to be customised and possibly extended for its actual application. The methods that the unwrapper provides can be called via its application programming interface. The usage of the unwrapper API is performed by the mediator. The implementation and the usage of a rights expression unwrapper is described in Section 6.4 respectively in Section 7.2.3.

- **Rights Expression Interpreter.** This component is in charge of decoding the rights expressions to the original rights information. The rights expression interpreter has to know the syntax and the semantics of each REL it is supposed to interpret instances of. Coding is necessary for each new REL the interpreter adopts. After the interpretation the rights information should be ready for processing in software services. Therefore, the interpreter maps the rights expression to an application-specific contract schema (see Section 4.6.1). The application-specific CoSa covers all attributes that are required by later software services. The application-specific CoSa is queried by the mediator via the generic CoSa API (see Section 4.6.3).

- **Mediator.** The mediator plays a central role in the rights expression exchange framework: it coordinates or *glues* the functions of the remaining four components in the framework. The mediator interacts with the framework components and other scenario-specific software services, such as access control mechanism or web server. With the knowledge of the APIs of the remaining four framework components, the mediator is able to coordinate them. By coordinating various components via their API, the mediator controls the workflow between the framework and the software services. The mediator code can be implemented as separate software class/program, or integrated in existing software classes/programs. The mediator code is adapted to the needs of a specific application. Very little of the mediator code can be reused in other applications, therefore in most cases each new
application that uses the framework components requires a new mediator implementation.

5.2.2 Implementation Check List

The following technical requirements are necessary for the implementation of a rights expression exchange framework:

1. Rights Expression Language. For the exchange of rights information a respective language is required. There are a number of rights expression languages, which are usually freely available (see Section 3.4). The cooperating DRM systems should agree on a rights expression language(s) and on REL application policies.

2. Implementation of a Generator. Each DRM component that aims at encoding rights information requires a rights expression generator. If DRM components run on different operating systems or platforms, a generator has to be developed for every operating system, respectively platform. The generator has to support a suitable interface. Offers or contracts that are created by content providers usually require an interface for the manual input of rights, license or contract information. However, there might be applications that require an API for the formulation of rights expressions. Furthermore, a generator might require a database connection for the temporary storage of licenses, because sometimes offers need to be restored for modification or further processing by a different component, e.g. the mediator.

3. Implementation of a Parser. A parser is required for all DRM components that receive rights expression messages. A parser usually provides validating the rights expressions in terms of syntactic correctness. A parser has to be available for the different operating systems respectively platforms of the DRM components. For rights expression languages that are XML-based, a large number of XML parser implementations are freely available on the Internet.

4. Implementation of an Interpreter. After a rights expression has been parsed, an interpreter maps the rights expression to semantics. The semantics are derived from REL in which the rights expression is coded. Additionally, the interpreter provides the rights information in a processable format. Again, an interpreter has to be available for the different operating systems respectively platforms of the DRM components. In this thesis a rights expression language interpreter for
ODRL (see Section 3.4.1) has been developed that is freely available in the spirit of open source.

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 is available.

5. **Secure Transportation Channel.** The exchange of a rights expression requires a secure transportation channel between two appointed DRM components. The channel has to assure that the rights expression is not deleted or modified during transportation. With the adequate security provided, the Internet or private networks are potential transportation channels.

6. **Concept for Runtime Presentation.** After the message has been interpreted the rights information is processed in software services. A general runtime representation of the rights information has the advantage that it is suitable to serve various software services, such as access control, accounting, and CRM services. The information in such a general representation is usually accessed via a predefined application programming interface. A generic schema for the representation of rights information as objects has been introduced in Section 4.6.

7. **Temporary Storage.** Rights expressions, in particular licenses and contracts, undergo several phases in their life cycle (see Section 4.1). Very often, such right expressions need to be temporarily stored. Depending on the current phase, different types of storage have to be provided. For example, prior to contract conclusion it is reasonable to store a contract in software objects, an XML database, or a relational database rather than as an XML document in the respective rights expression language, because it might be easier to restore the contract information for modifications in the negotiation phase.

8. **Framework Integration.** Most systems have not been designed from the beginning to handle rights expressions. A rights expression exchange framework often has to be integrated belatedly into application servers. To enable an easy integration, the framework respectively all its components should be coded in a suitable programming language and provide a well-defined interface.
9. *Mediator Implementation.* The mediator does not provide a general functionality, like the contract wrapper, unwrapper or the interpreter. It combines given components for a specific usage scenario. Therefore, a new mediator has to be implemented for each usage scenario. There are mediator tasks that need to be performed in every usage scenario, such as contract unwrapping and interpretation which follow a fixed pattern. In contrast to these, scenario-specific mediator tasks are differing in each application. A scenario-specific mediator task, for example, implements all rights expressions from a given contract into an access control mechanism. Another example of a scenario-specific mediator task is gathering the purchased products from all concluded contracts for the purpose of customer relationship management.

10. *Security Means.* To establish a reliable rights expression exchange framework, a large number of security means have to be applied. Attacks on rights expressions and their reliable processing can have various facets. Attacks that are concentrated on the rights expression message should be fended by the contract wrapper, respectively unwrapper. However, this provides neither yet tamper resistance of the framework components nor prevention from internal attacks (e.g., personnel). Certainly, building real-world tamper resistant systems is a complex task especially in open, distributed environments (see also [LTM⁺00, AK96])