

# The Problem with Rights Expression Languages\*

Pramod A. Jamkhedkar  
Department of Electrical &  
Computer Engineering  
University of New Mexico  
Albuquerque, NM 87131, USA  
pramod54@ece.unm.edu

Gregory L. Heileman  
Department of Electrical &  
Computer Engineering  
University of New Mexico  
Albuquerque, NM 87131, USA  
heileman@ece.unm.edu

Iván Martínez-Ortiz  
Centro de Estudios Superiores  
Felipe II  
C/Capitan, 39, 28300, Madrid,  
Spain  
imartinez@cesfelipesecondo.com

## ABSTRACT

In this paper we consider the functionality that a rights expression language (REL) should provide within a digital rights management (DRM) environment. We begin by noting the dearth of applications that make use of RELs, despite the fact that they have now been available since the late 1990's. We posit that one of the main impediments to the use of RELs is the complexity associated with understanding and using them. This results from the fact that the functionality needed to handle a wide variety of possible DRM scenarios is typically built into a REL, and it is often difficult to cleanly partition out only those pieces needed by a particular DRM application. Basing DRM system design on a layered architecture provides one way of achieving a partitioning and points to the need for a simple REL that is exclusively responsible for the expression of rights, while pushing much of the functionality found in current RELs into higher system layers. In order to demonstrate the usefulness of this approach, we provide an example implementation dealing with DRM-based negotiations.

## Categories and Subject Descriptors

C.2.6 [Computer-Communication Networks]: Internetworking—Standards; H.5.1 [Information Systems]: Interfaces and Presentation—Multimedia Information Systems; K.5.1 [Legal Aspects of Computing]: Hardware/Software Protection—Copyrights

## General Terms

Design, Legal Aspects, Security, Standardization

## Keywords

DRM, Rights Expression Language, Architecture

\*This research was partially supported by the National Science Foundation under a grant provided through the Future Internet Network Design (FIND) initiative.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DRM'06, October 30, 2006, Alexandria, Virginia, USA.  
Copyright 2006 ACM 1-59593-555-X/06/0010 ...\$5.00.

## 1. INTRODUCTION

Over the past several years, a number of rights expression languages (RELs) have been developed, with the eXtensible rights Markup Language (XrML) [19] and the Open Digital Rights Language (ODRL) [10] becoming the most popular. Recently, XrML was adopted as the standard REL for inclusion in the MPEG-21 standard [11, 12], and ODRL was accepted by the Open Mobile Alliance as the standard REL for mobile content [17]. Nevertheless, these RELs have not been extensively used in applications, despite the fact that many businesses suffer from problems that could be solved using appropriate digital rights management (DRM) technologies. For instance Apple and Microsoft have created their own lightweight DRM technologies, iTunes and Windows Media DRM10, respectively, that do not make use of any commercially available REL. At the very least, from an interoperability perspective, there would be some advantage to users if a simple standardized REL were used by both of these systems.

In this paper we make the case that both XrML and ODRL are much more than RELs, as they handle many additional aspects of rights management. For this reason, it may be more appropriate to refer to them as *DRM languages*. Furthermore, we contend that the comprehensive nature of these languages makes them difficult to analyze and hard to use in practice. What is needed are abstractions that logically partition the DRM functionalities provided by these languages into more manageable pieces. This would provide system developers with the ability to more easily incorporate DRM technologies into their designs. In addition, this would facilitate the growth of DRM markets as vendors can more easily participate and compete given a well-defined and standardized partitioning of DRM functionality.

There are numerous approaches that one could take in order to partition the functionality provided by a DRM system. The goal is to arrive at clearly defined and manageable levels of abstraction in the system architecture that better provide for reusability, portability, standardization and interoperability. One way of achieving this is to partition DRM functionality according to the well-known layered system design abstraction that has proved so useful in the development of other Internet-related protocol stacks [3, 6]. We have considered such a design, and have placed the REL at the center of the resulting DRM architecture [8, 13]. The scope of the REL we arrived at is significantly reduced from that of current commercial RELs. In a layered DRM architectural framework, the additional functionality is accounted for in higher system layers that may or may not be

needed in order to solve particular DRM problems.

This paper contains an investigation of the differences that result from basing DRM system design on two extremes, a framework with a simple REL at the core versus a complex REL that by itself is able to handle a wide range of DRM scenarios. This will enable us to uncover a number of problems associated with current RELs. The central problem we arrive at is this: current RELs are too complex, and lack a manageable standard partitioning of their functionality that would allow them to be more easily incorporated into DRM applications.

An overview of the remainder of this paper is as follows. In Section 2 we provide a brief overview of the development of RELs, describing how additional complexity continues to be added to these languages as perceived shortcomings are uncovered. Next, in Section 3 describe the current structure of popular RELs, and then we provide a brief overview of a layered architectural framework that describes how functionalities can be logically partitioned in a DRM system. We then show how ODRL and XrML language features reside across multiple layers in a reasonable DRM architecture. We believe it is necessary to partition these languages so that the elements associated with rights expression are separated out from those identified with rights negotiations, rights management and trust decisions, etc. The solution involves identifying a core which is exclusively concerned with expression of rights, as discussed in Section 4. In Section 5 we describe example implementations of a DRM scenario that allows one to more easily see the differences that result from using current RELs, versus a simplified REL obtained by refactoring according to the functionality provided in current RELs. Finally, in Section 6 we provide some recommendations and useful concluding remarks.

## 2. BACKGROUND

The precursors to current RELs were developed in the early 1990's. In 1994, Kahn proposed an electronic rights management system that included automated copyright registration and recordation, along with a transactional framework for on-line clearance of rights [15]. This system was mainly concerned with copyright management issues in a digital library setting, and proposed the use of electronic mail for communicating copyright-related data within the computer network environment.

Also in 1994, Stefik filed a patent for DRM technology he developed at Xerox PARC. This included the description of a "usage rights grammar" that was subsequently implemented in LISP and called the Digital Rights Property Language (DRPL). Through an evolutionary process DRPL became XrML. Specifically, in 1998 an XML implementation of DRPL (version 2.0) was released by Xerox PARC. Then, in 2000, ContentGuard, a Xerox/Microsoft joint venture, released XrML version 1.0, an evolution of DRPL 2.0. In 2002 ContentGuard released XrML version 2.0, a radical departure from all of the preceding versions. XrML 2.0 included an abstract rights language with very few core elements. The rights elements from previous versions were carried forward in this new version via an extension called the Content Extension. Finally, In 2003, the Motion Picture Experts Group (MPEG) released MPEG-21 Part 5, Rights Expression Language (ISO/IEC 21000-5), a substantially modified version of XrML 2.0 in which the Content Extension is removed, and a MultiMedia Extension appears in its place.

A similar evolution took place with respect to ODRL. Iannella introduced ODRL in 2000 out of concern for the closed approaches that were being used for DRM [9]. XML-based ODRL version 0.5 was released at that time as a work in progress, with the goals of providing clear DRM principles focused on interoperability across multiple sectors and support for fair-use doctrines. In 2001, ODRL Version 1.0 was submitted to ISO/IEC MPEG in response to a call for a rights data dictionary (RDD)-REL. By that time, Nokia's Mobile Rights Voucher and Real Networks' Extensible Media Commerce Language had been merged into the language. Then, in 2002, an Open Mobile Alliance (OMA) REL, based on ODRL 1.1, was proposed and called the OMA DRM 1.0 Enabler Release. ODRL 1.1 was also submitted to the W3C regarding possible chartering of a DRM/Rights Language activity within the W3C. By 2004, OMA released the OMA DRM 2.0 Enabler Release working drafts that addressed expanded device capabilities, improved support for audio/video rendering, streaming content, and access to protected content using multiple devices.

What we have witnessed in the evolution of these RELs is an increase in their complexity over time. This was driven by the desire for these languages to have the capabilities needed to handle a large range of possible DRM scenarios. The goal was to make them highly flexible so that they could support a wide variety of business models. Thus, over time, new rights scenarios were described, and if a particular REL could not implement the scenario, then new tags or entire extensions were added to the language. Indeed, recently we have seen a number of papers where a particular DRM-related problem is described, the difficulty of solving the problem using current RELs is noted, and a solution is proposed that invariably increases the complexity of the resulting REL. For example, it was pointed out that XrML is unable to directly handle the following DRM use-case scenario [4]:

Licenses from various vendors are combined in order to create rights for access to a particular piece of content.

Due to this problem, the authors proposed a new Prolog-based REL that is able to directly solve this problem, and also use the particular capabilities of this programming language to "reason" about rights.

As another example, the importance of conducting negotiations in DRM environments was recently described, along with the inability of XrML and ODRL to function in this setting [1]. The authors then describe two solutions, one that involves adding these capabilities outside of the REL, and another that incorporates the ability to conduct negotiations within ODRL through the addition of new tags. It should be noted that the prior problem involving license aggregation could also be solved in these two ways. In fact, for any DRM problem that cannot be handled directly by a REL, we have the choice of adding the necessary capabilities to the REL, or providing them through mechanisms external to the REL. The former will increase the complexity of the REL itself, while the latter does not.

The major focus of this paper is the notion that rights management mechanisms should be kept out of RELs, which therefore should heavily bias DRM architectures towards the latter approach described above. That is, RELs should focus on rights expression, allowing a simple REL core to be defined. Such a core for rights expression is similar in spirit

to the simple tuple calculus for query specification at the core of database management systems. This simplicity has a number of benefits, including greatly facilitating the ability to conduct mathematical analyses. It is important to recognize that the REL core we refer to is far different from the core defined in the XrML and ODRL specifications. Once such a simple core is in place and well defined with respect to its rights expression capabilities, protocols can be developed around it to handle different scenarios. Furthermore, this simplification will require manufacturers of rendering devices to support only the core of the language, allowing heavy-duty rights management functionalities to be implemented on the server side, where they belong.

Reconsideration of what capabilities a REL should provide is necessary in the wake of recent developments. Specifically, the current pattern of development in RELs reflects a phenomenon where one language, such as XrML, is faulted because it cannot implement some particular DRM scenario, and a new language or extension is proposed that does have the desired capabilities. The result is “language bloat”, and a situation where RELs by themselves can implement ever more complicated DRM scenarios. Although this trend may have its benefits in allowing heavyweight DRM solutions to be constructed, the flip side is that it makes simple lightweight DRM applications incorporate more features than may be necessary for their purposes. We propose that these additional management capabilities found in RELs such as XrML and ODRL can and should in fact be handled at higher levels in a layered DRM architectural framework. Before discussing this issue in more detail, it is first necessary to provide a brief overview of DRM architectures.

### 3. DRM SYSTEM ARCHITECTURES

A DRM system can be highly complex, and for this reason it is useful to consider abstract DRM architectures that can assist in understanding and analyzing existing DRM systems, as well as aid in the design of new DRM systems with specific desirable properties. The effort taken to develop abstract architectural models for modern communication systems has greatly facilitated the development of the Internet. The same can be said for simulation environments, parallel computing systems, etc. Thus, the effort to create an abstract DRM architecture is worthwhile, and will hopefully aid in the development of DRM systems by providing useful abstractions for thinking about DRM systems, along with a common vocabulary for talking about DRM systems.

In Section 3.1 we describe the structure of current RELs and make some conjectures as to why they developed the way they did. Then in Section 3.2 we briefly review a useful abstract DRM architecture that allows us to partition the functionality contained in DRM systems. This review will provide us with the vocabulary necessary to discuss how current RELs fit into this framework, allow us to identify what layers different parts of these languages reside in, and observe the extent of the functionality provided by these RELs.

#### 3.1 Existing Architectures

Generally, design documents associated with RELs state that the main purpose of a REL is to express rights in an unambiguous and machine-readable manner. However, supplementary to this basic goal, a number of additional requirements for RELs are often specified. For example, the

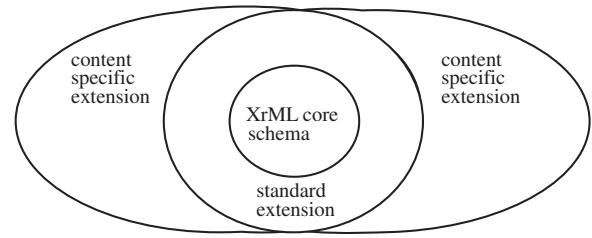


Figure 1: Relationship between XrML schemas.

requirements for the MPEG-21 RDD and REL include the ability to support various business models, the authentication and verification of rights expressions, extensibility, configuration permissions that allow content to be added or removed from repositories, just to name a few. Thus, the range of functionality is extensive. In order to more fully understand this scope, let us consider specific REL implementations.

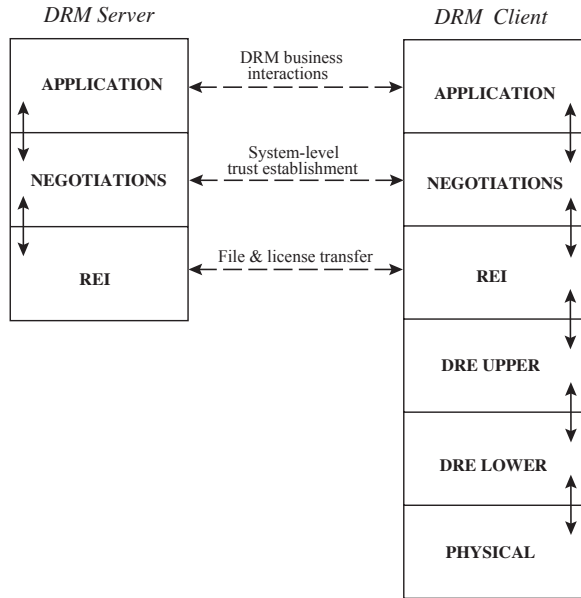
The XrML and ODRL RELs are based on the XML markup language. Markup languages provide a set of tags for identifying the components of a document that enables each component to be appropriately formatted, displayed, or used. In XML, it is easy to create new tags with new semantics according to the requirements of some environment. Thus, XrML and ODRL both provide a core language that can be easily supplemented with XML extensions as required by a particular application area. Figure 1 uses a Venn diagram to show the relationships between the various parts of XrML. The language consists of a core schema containing the most basic semantics, a standard extension containing broadly applicable DRM concepts, and content specific extensions related to particular content types or businesses. A similar arrangement occurs in ODRL.

The core of XrML is built around the following basic use-case scenario:

A license is issued by an issuer which consists of a grant of a particular right to a principal over some resource under some conditions.

Any DRM use-case having this basic use-case as its root can be effectively expressed in the language by using extensions if necessary. Users can define their own extensions using XML schema and namespace mechanisms. In addition to the expression of rights, XrML incorporates other features such as specifying the location of content, supplementary information regarding principals, the algorithm used for encrypting resources, the key required to decrypt the resource, etc. The inclusion of these features extends the scope of XrML far beyond rights expression and into the realm of rights management.

The features we have just mentioned, along with many others contained in XrML, appear to exist in order to satisfy certain DRM usage scenarios and protocols. Indeed, the incorporation of the term “license” at the heart of the XrML language suggests DRM concepts well beyond the expression of rights. Thus, the principles that have guided design decisions for these languages seem to be primarily focused on the end-to-end functionality that should be provided, with less consideration given to the logical design of the overall system architecture. Therefore, although XrML and ODRL do a good job of satisfying end-to-end functional DRM re-



**Figure 2: A layered DRM architectural framework.**

quirements, they do this in such a way that they cross many of the boundaries that should exist between DRM system components. That is, no clear boundaries are provided between important DRM functionalities such as rights expression, rights interpretation, negotiations, etc. Furthermore, in order to support new DRM use-cases related to particular content types, the language must be continually expanded through extensions. However, as described in Section 4, the extensions are generally not well-suited for reuse.

In software engineering the term *refactoring* is used to describe the process of changing the structure and/or design of existing code in order to improve its understandability, internal consistency, performance, etc. We believe that a refactoring of RELs may prove useful, particularly if it involves a partitioning of the language into more manageable pieces. One means of achieving this partitioning is described next.

### 3.2 Layered Architectures

The partitioning approach described in this section is based on what has proved successful in the design of many computer/communication systems, namely, the notion of partitioning the system components according to a layered architectural framework [3, 6]. This manner of decomposition provides a very effective way of managing the complexity of large complex system designs. The central idea involves guiding decisions during the design process by defining the hierarchy of management responsibilities that should exist in the architecture, and then arranging these responsibilities in layers that involve high-level operations, in upper system layers, that rely upon low-level operations, in lower system layers.

A suggested layered DRM architectural framework is provided in Figure 2 [13]. This figure contains two protocol stacks, one for the client and one for the server in a DRM transaction. The arrows shown within and between the protocol stacks indicate where interfaces need to exist in order

for the layers to interact (see [8] for more details). Obviously, these interfaces are important in standardization efforts.

The Rights Expression and Interpretation (REI) Layer in Figure 2 is an important layer in that it links higher level DRM functionality related to applications with lower level DRM functionality related to tracking and enforcement. The goal of this layer is to provide the minimal set of DRM services necessary to facilitate this connection. Thus, the services provided by the REI Layer are the minimal ones required by any DRM application. More specifically, at a minimum we must have rights in a DRM application. For this reason, the rights expression functionality should be as simple as possible in order to support desired properties such as portability, interoperability and interchangeability. The necessity of right interpretation comes into play when different DRM systems must interact, or when what is required by an application or decided through negotiation must be reconciled with what rights enforcement can actually be achieved in the lower layers.

Considering what we have just discussed, it is clear that the most important technology in the REI Layer is the REL, which provides for the formal specification of rights. Moreover, as we move away from the REI Layer (either up or down) the ability to formally specify rights seems to become less important. For instance, the Negotiations Layer is responsible for such things as system level trust establishment, as well as other types of negotiations between the client and server. Although rights may be used during these negotiations, the ability to understand their meaning is not required, as each negotiating party can refer to its REI Layer when this functionality is needed. Similarly, the DRE Upper Layer is concerned with the technologies used in rights enforcement, such as encryption algorithms, authorization algorithms, watermarking, etc. Once again, it is not necessary for services operating at this layer to understand the formal meaning of rights in order to provide their functionality.

In order to set the stage for studying some of the problems associated with current RELs, let us start by overlaying the functionalities they provide onto the abstract DRM architecture we have just described. The core schema of XrML consists of the definitions that are at the heart of the XrML semantics. Thus, let us examine this schema in order to determine the layers in which it resides. The central element in the core schema is a license that plays the role of a container for grants. A grant conveys to certain principles certain rights to certain resources under certain conditions. Thus, since grants specify information about principles, rights, resources, and conditions, they in fact play the role of “rights expression” in the REI Layer of Figure 2.

An XrML license also contains information such as the date of issue of the license, the digital signature of the license issuer(s), the place of issue, etc. This information is generally used as part of some protocol or set of protocols related to a DRM transaction. For example, the license is a data element that is often determined via a two-party negotiation. Similarly, authentication is a service that may be used during or after these negotiations, and in itself may also involve a protocol. Thus, just by introducing the concept of a license, we see that the core schema extends to at least the Negotiations and DRE Upper Layers shown in Figure 2.

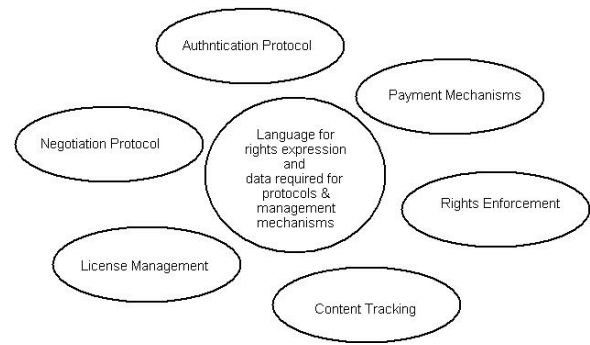
## 4. PROBLEMS WITH CURRENT RELS

Once again, we believe the central problem that exists with respect to the use of RELs concerns their complexity, and the fact that manageable abstractions have not been created that enable developers to deal with this complexity. The partitioning of functionality along the lines of what has been described in Section 3.2 is one way of packaging this complexity into more manageable pieces. In any event, it is useful to refer to the layered architecture in Section 3.2 when discussing the current problems associated with RELs.

The first problem we note with current RELs is the difficulty of reuse. In particular, because the extensions in current RELs are content-based, rather than functionality-based, reuse is difficult. This is because in these languages DRM functions such as trust management, authentication, encryption, or negotiation services are not completely separable from the language, as we believe they should be. Thus the “units” of reuse provided by XrML, i.e., the core schema, the standard extension, and content specific extensions, are too coarse and cut across too many areas of functionality. Reuse becomes more natural if it is based on smaller units of DRM functionality such as authentication or encryption, rather than the end-to-end functionality described in a large use case. For instance, not all e-book companies may require the same functionality in order to conduct their businesses. This makes it difficult to design a single content-based extension that can satisfy the needs of all businesses in this industry. On the other hand, even companies having vastly different businesses may find it useful to use a specific DRM functionality, e.g., authentication, negotiation services, etc., particularly if it can be easily integrated into their existing infrastructures.

The problem we have just discussed may stem from the fact that the architecture of current RELs appear to have been derived from use-case scenarios. Typically, use-cases are used in the design of software-intensive applications for the purpose of eliciting requirements. Hence, use-cases tend to capture end-to-end functional requirements in a DRM environment; however, they are unable to express system-theoretic properties such as reusability, portability and interchangeability. An important consideration in the creation of current RELs appears to have been the ability to handle the largest possible range of DRM scenarios. Furthermore, the extensible nature of these languages enables them to incorporate ever more complex DRM scenarios as they are derived.

An analogy using a system more familiar to us may help us to understand the difficulties this leads to. Specifically, consider the design of a database management system architecture from the perspective of only the functionality it will provide. The core of any database management system is a database along with a set of associated queries. Typically, database management systems include a query language over which different database management protocols are implemented. If we think of database management systems along use-case lines, their use involves the process by which a user queries the system, and the result of the query is displayed back to the user, or changes are made to the database accordingly. In addition, the complete process of using a database management system involves not only the query to the database, but other functions such as identifying and authenticating users, security, desired display properties of query results, and provisions for operating on these



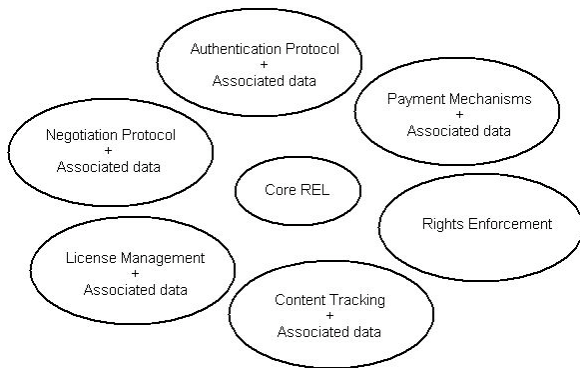
**Figure 3: DRM Services implemented using existing RELs.**

results. A markup language along the lines of XrML can be developed for database management systems which captures all of the important scenarios and protocols involved in database management systems. Such a language will include tags for user authentication, tags for information about desired display properties, information about the encryption method used in the case of secure systems, and also provisions for including decryption keys. Such a language must then be continually expanded in order to include different database management scenarios that may arise.

Database management systems, however, have not been designed in the manner we have just described. Rather, the functionality we have described for them is typically organized according to a well-known layered design pattern for database managements systems that consists of a Presentation Layer, an Application Logic Layer, a Domain Layer, and finally the Database Layer. In the case of relational databases, at the core of the system is a simple tuple calculus whose capabilities have been rigorously analyzed. Because of the simplicity of this tuple calculus, researchers have been able to build a user-friendly structured query language (SQL) on top of it. With this language at the core, protocols in database management systems are easily implemented around it. Thus the protocols can be expanded according to the requirement of the application, whereas the core query language remains more or less the same. A similar capability should be developed for DRM. Specifically, a REL corresponding to a simple logic should be developed, and on top of this foundation, more sophisticated DRM capabilities should be built.

This analogy suggests a refactoring of DRM systems is needed using some architectural framework, such as the one described in Section 3.2, as a guide. In the previous section we noted that the language components of ODRL and XrML reside across three different layers in this layered framework. Thus, the language used for the expression of rights, as well as the much of the data required for protocols and management mechanisms are combined. This situation is depicted in Figure 3. In this figure, we show a number of protocols that make use of language components within the REL in order to complete their tasks. That is, the semantics of the protocols are merged into the REL, and it therefore becomes extremely difficult to change these protocols as it requires changing the REL itself.

Let us contrast this with a different architectural approach. The layered framework suggests a partitioning in which the



**Figure 4: DRM services making use of a simplified core REL.**

semantics of the protocols are completely separated from the REL, as shown in Figure 4. With this implementation, it is easy to see that rights-based protocols may be changed and new ones introduced without affecting the core REL. Indeed, with this separation, it would be possible for a new rights-based business to redefine the core REL in order to use a new rights model. The key point is that with the architecture shown in Figure 4, existing DRM services can be packaged so that they can immediately use the new rights model.

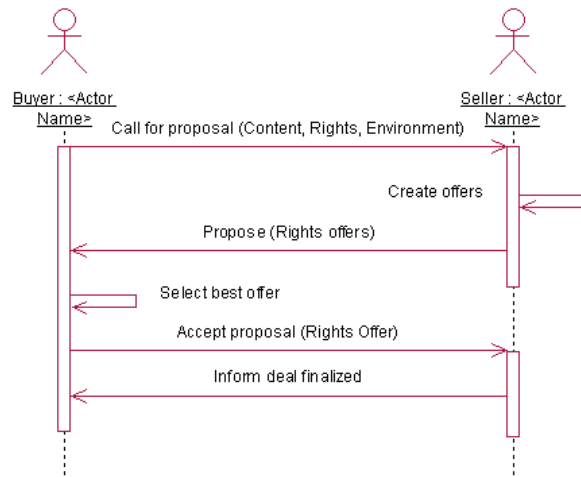
In fact, based on this discussion, the arrangement shown in Figure 4 supports the development of middleware services for DRM, a situation that we have begun to investigate [14]. The use of middleware services has greatly facilitated the development of many Internet-based businesses, and they have the potential to do the same for the DRM industry.

The implications of the problems we have discussed in this section may become clearer after considering the implementations of an example DRM scenario described in the next section. This example contrasts an implementation conducted according to what is shown in Figure 3, with one that is organized according to what is shown in Figure 4.

## 5. AN EXAMPLE IMPLEMENTATION

Let us consider a rights negotiations scenario in which a buyer negotiates with a seller the rights associated with some content purchase. Figure 5 provides a high-level description of the rights negotiations that will take place between these two parties. First, the buyer requests a content object and the rights that should be associated with it. In addition, the buyer supplies the environment within which the content will be used and the rights exercised (e.g., a rendering device id and its security level). This is shown as a call for proposal request in Figure 5. The seller then responds by proposing a set of offers that are consistent with the policy statement of the content provider. Next, the buyer chooses a particular offer through an accept proposal request. Finally the seller informs the buyer that the request has been granted. This is actually a simplification of the Foundation for Intelligent Physical Agents (FIPA) Contract Net Interaction Protocol [5].

The DRM scenario we have just described cannot be entirely implemented in either ODRL or XrML. Thus, there are two approaches we can take. The first is to include the semantics of the protocol and the supporting data repre-



**Figure 5: A high level view of a negotiations protocol.**

sentation within the REL, and then develop a lightweight protocol that understands the extended REL and can process the necessary information. In the second approach, the semantics of the protocol and the supporting data are kept separate from the REL, which allows the negotiation protocol to be developed independent of the REL. Let us consider each of these implementation choices in turn.

**Protocol Integrated REL Approach.** In this approach, the semantics of the protocol are integrated into the REL. Hence, this approach requires that the protocol processing logic understand not only the semantics of protocol but also the semantics of the language, since the two are integrated. In other words, the protocol processing logic is tied to the REL. The pseudocode for the implementing the FIPA Contract Net Interaction Protocol is described next.

First a call-for-proposal request from the buyer to the seller is issued. Here, the buyer is requesting a set of rights over some content, along with the environment under which the rights will be exercised:

```
<call-for-proposal>
  <environment> ..... </environment>
  <rights>
    <party> ..... </party>
    <content> ..... </content>
    <permission>.....</permission>
  </rights>
</call-for-proposal>
```

The seller now analyzes this call for proposal and creates a set of offers consistent with its policies:

```
<propose>
  <offer1>
    <rights>.....</rights>
  </offer1>
  <offer2>
    <rights>.....</rights>
  </offer2>
</propose>
```

Finally, the buyer selects a particular offer. This involves issuing an accept-proposal request informing the seller which of the offers will be purchased:

```
<accept-proposal>
  <offer>
    <rights>.....</rights>
  </offer>
</accept-proposal>
```

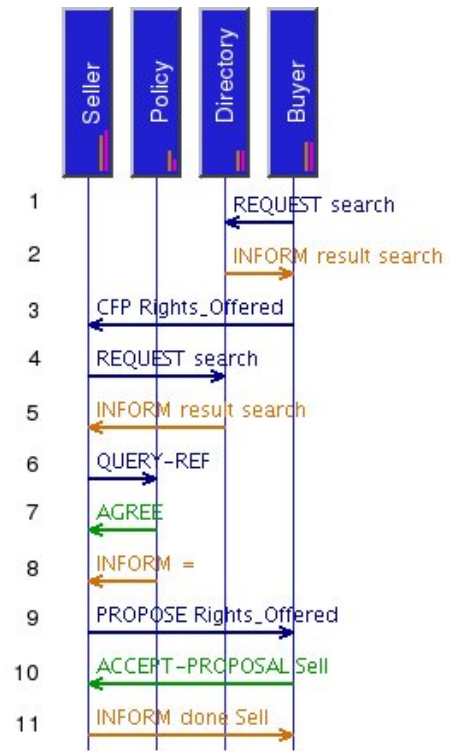
The performatives used in this implementation, namely `call-for-proposal`, `propose` and `accept-proposal`, which represent the semantics of the message type, form just a subset of those used in the Contract Net Interaction protocol. Hence it is also necessary to define tags for rest of the performatives used in the protocol. Also, the Contract Net Interaction protocol is just one of the various protocols specified by FIPA. There are many other protocols useful in DRM scenarios such as Request Interaction Protocol and Query Interaction Protocol. Since, these protocols share the performatives amongst themselves, it is also necessary to create separate tags to specify the type of the protocol to ensure correct implementation.

**Agent-based Modular Approach.** In a modular approach, the protocol semantics are separated from the REL, which is used purely for rights expression. We used an agent-based approach for carrying out rights negotiations between the buyer and seller. The buyer specifies its agent by providing a set of goals in terms of rights and desired content. It is the responsibility of the buyer agent to obtain offers from the seller agent. Similarly, the seller agent is required to process requests from different buyer agents and propose a set of offers that are consistent with the policy statement of its content provider. In this implementation, the rights are merely used as a payload, and the buyer and seller do not need to understand how they are expressed. We will see that only the policy agent in this simulation needs to understand the semantics of the rights. The DRM scenario was implemented using a multi-agent framework called the Java Agent DEvelopment (JADE) Framework ver. 3.4 [2].

The agents in an agent-based system are independent but they need to interact in order to achieve any functionality. In order for this to happen, the agents need to share some common domain knowledge. An ontology is an explicit specification of the concepts within a domain. An ontology therefore provides a common understanding of the structure of information among agents. Thus, before the initiation of the protocol, the agents in this implementation must agree on the ontology that will be used for the interactions. This makes it possible to change the agent interface, which represents the protocol, without requiring changes to be made to the ontology, and vice versa. In this implementation of the DRM scenario of Figure 5, the agents negotiate about rights. Thus, an ontology was defined that specified a rights model using the Protégé ontology editor [16]. This tool also created corresponding Java classes representing the rights model that were then used within JADE.

Three different agents were created for this scenario:

- **Buyer.** The user provides this agent with a set of goals in terms of rights and content. The Buyer agent then sends call-for-proposals to different seller agents. After receiving different proposals, the buyer agent selects the best proposal and sends an accept-proposal to the corresponding Seller agent.



**Figure 6: Activity diagram showing the messages exchanged among different agents**

- **Seller.** This agent processes call-for-proposal messages and queries the Policy agent to generate different offers according to the requests. After getting the set of offers, the Seller agent sends a proposal to the Buyer agent consisting of the set of offers. Finally, the Seller agent receives the accept-proposal messages from the Buyer agent, and informs the Buyer agent that the deal is finalized.
- **Policy.** The content provider supplies this agent with a set of policies. After receiving a query from the Seller agent, the Policy agent responds to the query with a set of offers consistent with the policies.

The Buyer agent and the Seller agent carry out the negotiations using the Foundation for Intelligent Physical Agents (FIPA) specified Contract Net Interaction Protocol. The Seller agent queries the Policy agent for offers using the FIPA Query Interaction Protocol. In addition to these agents, there is a directory service that is provided automatically by the JADE application. Basically, this is where all agents in the application register so that they can be known to the other agents in the application. Thus, this Directory agent is where different Seller agents register in order to offer their services. The Buyer agent queries this Directory agent in order to learn about different Seller agents. Similarly, Policy agents offer their services to Seller agents via the Directory agent.

Figure 6, shows the messages exchanged among the specified agents. Messages 1 and 2 represent the query from the Buyer agent to the Directory service for the list of available seller agents. After getting the Seller agent, the Buyer

agent sends a call-for-proposal message to the Seller agent requesting the rights for the desired content in message 3. Messages 4 and 5, represent the query message by the Seller agent to inquire about the policy agents. In messages 6, 7 and 8 the Seller agent obtains the various rights offers that match the request by the Buyer agent and are consistent with the content providers policies. In message 9, the Seller agent proposes the offers to the Buyer agent, which accepts the proposal in message 10. Finally, in message 11, the Seller agent informs the Buyer agent that the deal is finalized.

In summary, it can be seen that in the case of the first implementation, integration of the protocol semantics within the REL imposes severe restrictions on the development of DRM systems. Any change in the existing protocol, or development of a new protocol, necessitates that the changes be reflected in the language meant for rights expression. Also such an approach forces the system to be developed in a monolithic manner, since the protocol is tied to the REL. In the second approach, the protocol is separate from the REL. This separation, makes it possible to use the rights just as a payload to carry our negotiations. As can be seen from the implementation, only the Policy agent needed to understand the semantics of the rights since it makes a policy decision. The Seller agent only understands the semantics of the protocol. Thus, with this implementation, it is possible to use a totally different rights model to carry out negotiations, and this requires changes only in the Policy agent, while the rest of the protocol remains the unchanged. Similarly, the current rights model can be used with a different negotiations protocol without requiring any changes to the rights representation. Thus, with this approach, where the rights model and the negotiations protocols are completely separate, each can be independently developed and modified, yet continue to work together. This makes it easy to package the negotiations services, and many other DRM services for that matter, as middleware. Thus, it is the clean separation of rights from services that is the key to creating interoperable DRM systems.

## 6. RECOMMENDATIONS AND CONCLUSIONS

Based on the discussions in the previous sections detailing the problems with current RELs, along with what has been learned from the example we have just presented, it makes sense to recommend the following set of design principles related to RELs and DRM services. Adherence to these principles would lead to DRM systems that are easier to design and use, and will facilitate the creation of more complicated DRM systems through the composition of middleware services from disparate vendors.

1. **The core REL should only contain the rights model.**
2. **The core REL should be stateless.**
3. **The core REL should be language neutral.**
4. **REL primitives, and DRM services in general, should refer generically to the services they use.**
5. **A DRM service should only know what it absolutely needs to know in order to complete its task.**

### 6. A DRM service should encapsulate the operations it performs, along with the data used by these operations.

The first three recommendations deal with the simplicity of the core REL. The rights model referred to in the first recommendation consists of the *rights* (i.e., activities), the *constraints* within which the right can be exercised, and the *environment* over which the rights are given and the constraints operate. Thus, the core REL should not specify how rights should be created, authenticated, communicated, audited, enforced, etc. This leads directly to the second recommendation regarding the statelessness of RELs. This recommendation follows from the fact that pure rights expression does not require the ability to understand or recognize the state of a content object, device or principle. Once information about the state of these entities is required or used, we have moved beyond rights expression into the realm of rights management. In this case, the functionality should be handled at layers other than the REI Layer in the DRM protocol stack. Next, the third recommendation, which may appear to be contradictory, suggests that the core REL should be thought of as something that exists independent of a particular implementation language. Because rights are much less transient than technology, their expression should not be tied to any language. Ideally, the core REL is specified entirely in terms of a mathematical system of logic, and can be translated into particular programming languages as needed. We have seen attempts to work backwards in order to discover the logic structure of ODRL and XrML [7, 18]. This reveals a highly complex structure that is extremely difficult to understand let alone reason about. It makes more sense to start from a simple logical framework, and work forwards from it in order to discover what can and cannot be achieved within the framework. Once again, this is similar to what has been done with relational databases.

The next recommendation deals with DRM services, and the fact that we should avoid entangling them within other services or the REL. In particular, the fourth recommendation deals with the fact that RELs and DRM services should not refer to particular implementations of the services they need to use. As an example, an authentication service should not “hardcode” a particular encryption service, but should instead parameterize this choice. Similarly, a particular negotiations protocol should not appear “hard-coded” into a REL or DRM service. We should also guard against the problem of pieces of a service slipping into the REL because they seem important. For instance, MD5 message digests and RSA key pairs should have no place in the REL, and indeed should only appear as part of some particular DRM service, not as standalone entities in a DRM system. This problem, however, is already covered by the first recommendation. Following these recommendation leads to the creation of services that can be easily changed or updated without invalidating the things that use them. This enhances reusability and changeability, and increases the possibilities for interoperability.

The last two recommendations are closely related to the idea of how information is arranged in object-oriented systems. The encapsulation of operations and data, along with well-defined interfaces is a correct architectural approach for the development of modular, distributed and interoperable DRM systems. Specifically, the fifth recommendation asserts that a DRM service implementation should know only

what it absolutely needs to know in order to complete its task. For example, high-level license management mechanisms that deal with license acquisition, authentication and encryption need not understand the semantics of rights. This requirement necessitates a clear separation of the semantics of different DRM services, which leads to the last requirement. Specifically, a particular DRM service defines and understands the semantics of only the data required for its implementation. Such a service then provides an interface to other services that wish to use it. For example, an encryption service understands the semantics of the protocols and keys, however, it does not need to understand the meaning of the data that is being encrypted and therefore simply treats the data as a payload. In the end, this allows complex compositions of various DRM services to be more easily created. The identification of a simple core REL is the first step in this direction, as the separation of the semantics associated with different DRM services is impossible given the scope of current RELs.

In this paper we have pointed out how current RELs incorporate for more functionality than pure rights expression. Specifically, in order to adapt to different DRM scenarios, these languages incorporate rights management mechanisms beyond rights expression which leads to “language bloat”. We have demonstrated that by taking a system-theoretic approach, it is possible to identify different DRM functionalities within these languages. Thus, it is possible to partition these functionalities among the layers of a standard DRM layered architecture. This allows a much simpler core REL to be identified that deals exclusively with expression of rights. This also allows the properties of the resulting language to be more easily analyzed in terms of its capabilities and limitations. Finally, a core REL provides the much needed separation of rights management mechanisms from rights expression. This separation is essential for the focussed development of the DRM services that will be necessary in order to achieve system-level goals that include reusability, portability, interchangeability, and interoperability.

## 7. REFERENCES

- [1] Alapan Arnab and Andrew Hutchison. Fairer usage contracts for DRM. In *Proceedings of the Fifth ACM Workshop on Digital Rights Management*, pages 1–7, Alexandria, VA, Nov. 2005.
- [2] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa. JADE programmer’s guide. Technical report, Telecom Italia, Nov. 2005.
- [3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern Oriented Software Architecture—A System of Patterns*. John Wiley & Sons, 1996.
- [4] C. N. Chong, R. Corin, S. Etalle, P. H. Hartel, W. Jonker, and Y. W. Law. LicenseScript: A novel digital rights language and its semantics. In *Third International Conference on the Web Delivery of Music*, pages 122–129, Los Alamitos, CA, 2003.
- [5] FIPA Contract Net Interaction Protocol Specification. Technical report, Foundation for Intelligent Physical Agents, Dec. 2002. [www.fipa.org/specs/fipa00029/](http://www.fipa.org/specs/fipa00029/).
- [6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns—Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.
- [7] Joseph Y. Halpern and Vicky Weissman. A formal foundation for XrML licenses. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, pages 251–265, Asilomar, CA, June 2004.
- [8] G. L. Heileman and P. A. Jamkhedkar. DRM interoperability analysis from the perspective of a layered framework. In *Proceedings of the Fifth ACM Workshop on Digital Rights Management*, pages 17–26, Alexandria, VA, Nov. 2005.
- [9] Renato Iannella. Open digital rights language (ODRL), Version 0.5. <http://odrl.net/ODRL-05.pdf>, Aug. 2000.
- [10] Renato Iannella. Open digital rights language (ODRL), Version 1.1. <http://odrl.net/1.1/ODRL-11.pdf>, Aug. 2002.
- [11] International Standards Organization. Information Technology – Multimedia Framework (MPEG-21) – Part 5: Rights Expression Language. ISO/IEC 21000-5:2004.
- [12] International Standards Organization. Information Technology – Multimedia Framework (MPEG-21) – Part 6: Rights Data Dictionary. ISO/IEC 21000-6:2004.
- [13] P. A. Jamkhedkar and G. L. Heileman. DRM as a layered system. In *Proceedings of the Fourth ACM Workshop on Digital Rights Management*, pages 11–21, Washington, DC, Oct. 2004.
- [14] P. A. Jamkhedkar, G. L. Heileman, and I. Martínez-Ortiz. Middleware services for drm. Technical report, University of New Mexico, ECE Dept. Technical Report, Aug. 2006.
- [15] Robert E. Kahn. Deposit, registration and recordation in an electronic copyright management system. In *Proceedings: Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment*, Annapolis, MD, 1994. [www.cni.org/docs/ima.ip-workshop/www/Kahn.html](http://www.cni.org/docs/ima.ip-workshop/www/Kahn.html).
- [16] H. Knublauch. An AI tool for the real world: Knowledge modeling with Protégé. *Javaworld*, June 2003.
- [17] Open Mobile Alliance. Rights expression language version 1.0, June 2002. [www.openmobilealliance.org/release\\_program/drm\\_v1\\_0.html](http://www.openmobilealliance.org/release_program/drm_v1_0.html).
- [18] R. Pucella and V. Weissman. A formal foundation for ODRL, 2006. [www.citebase.org/abstract?id=oai:arXiv.org:cs/0601085](http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0601085).
- [19] XrML 2.0 technical overview, version 1.0. [www.xrml.org/reference/XrMLTechnicalOverviewV1.pdf](http://www.xrml.org/reference/XrMLTechnicalOverviewV1.pdf), March 2002.