



Proceedings
of the
First International Workshop
on the
Open Digital Rights Language (ODRL)
Vienna, Austria
April 22nd –23rd, 2004

Edited by:

Renato Iannella
Susanne Guth

Sponsored by:



Copyright remains with the authors. Request permission of authors to republish works that are included in these proceedings.

Proceedings of the First International ODRL Workshop
Edited by Renato Iannella and Susanne Guth
22nd – 23rd April, 2004

ISBN: 1-74064-500-6

Additional copies may be ordered from:

Renato Iannella
IPR Systems, Australia
Email: renato@iannella.it

Susanne Guth
Vienna University of Economics and Business Administration
Augasse 2-6
1090 Vienna
Austria

Email: susanne.guth@wu-wien.ac.at
Phone: +43 1 31336 4427
Fax: +43 1 31336 746

Preface

Welcome to the first Open Digital Rights Language (ODRL) International Workshop. ODRL is an XML-based rights expression language (REL). A rights expression language is a means of expressing usage and access rights of parties to assets. Rights expression languages provide a syntax and semantics that are sufficiently rich to formulate rights expressions for digital publications, audio and video files, images, games, software, and other digital or physical goods, including pricing models as well as terms and conditions, regardless of whether a monetary consideration is part of the transaction. Consequently, rights expression languages provide a metadata framework for the expression of rights.

The ODRL Initiative has gained international significance in the field of digital rights management (DRM) over the past years, culminating in ODRL being adopted as an international standard by the Open Mobile Alliance for supporting the process of mobile content distribution and management.

The objective of the ODRL International Workshop was to bring together the research and industry communities to share experiences and discuss the future developments of the ODRL language and to ensure its timeliness, usability, openness, and future success.

ODRL is seen as key infrastructure for the management and trading of content in the digital environment. ODRL enables to formulate machine readable, interoperable contracts between rights holders and content users and need to evolve as the community awareness increases and business models change. The role of the Workshop is to enable this process.

The workshop would not have been the success we trust it will be without the support of the two hosts, IPR Systems and The Vienna University of Economics and Business Administration, and our sponsors, LiveEvents Wireless and BluePrint Software. We also would like to thank the members of the Program Committee for their work refereeing the papers.

The Workshop website will include all the papers and presentation slides:

[<http://odrl.net/workshop2004/>](http://odrl.net/workshop2004/)

Renato Iannella and Susanne Guth
Workshop Chairs and Editors

Table of Contents

Workshop Organisation	v
Keynotes:	
DRM and the Web <i>Rigo Wenning, W3C, France</i>	3
The OPERA Project: Interoperability of Digital Rights Management (DRM) Technologies <i>Andreas Deppe, T-Systems, Germany</i>	17
OMA Secure Content Delivery for the Mobile World <i>Willms Buhse, OMA Download and DRM Group, CoreMedia, Germany</i>	35
Papers:	
A Pervasive Application Rights Management Architecture (PARMA) based on ODRL <i>Dominik Dahlem, Ivana Dusparic and Jim Dowling, Trinity College Dublin, Ireland</i>	45
Interoperability between ODRL and MPEG-21 REL <i>Josep Polo, Jose Prados and Jaime Delgado, Universitat Pompeu Fabra, Spain</i>	65
REAP: A System for Rights Management in Digital Libraries <i>Oyvind Vestavik, Norwegian University of Technology and Scienc, Norway</i>	79
A Proposal for the Evolution of the ODRL Information Model <i>Susanne Guth, Mark Strembeck, The Vienna University of Economics and BA, Austria</i>	87
Distributed Digital Rights Management: The EduSource Approach to DRM <i>Stephen Downes, Gilbert Babin, Luc Belliveau, Raphael Blanchard, Gerard Levy, Pierre Bernard, Gilbert Paquette, Sylvie Plourde, National Research Council Canada</i>	109
Towards Formal Semantics for ODRL <i>M. Holzer, S. Katzenbeisser, C. Schallhart, Technische Universität München, Germany</i>	137
Nonius: Implementing a DRM Extension to an XML Browser <i>Olli Pitkänen, Ville Saarinen, Jari Anttila, Petri Lauronen, Mikko Välimäki, Helsinki Institute for Information Technology, Finland</i>	151
Invited Talks:	
Flexible DRM: Real-life ODRL Implementations <i>Stephane van Hardeveld, VirtuosoMedia, The Netherlands</i>	165
An Interoperable and Flexible Infrastructure for DRM <i>Mariemma Yagüe, University of Malaga, Spain</i>	177

First International ODRL Workshop – Organisation

Workshop Chairs: *Renato Iannella*
IPR Systems, Australia
Susanne Guth
Vienna University of Economics and BA, Austria

Program Committee: *Grace Agnew Rutgers*,
The State University of New Jersey, USA
Gilbert Babin
National Research Council, Canada
Oliver Bremer
Nokia, Finland
Robin Cover
ISOGEN International LLC, USA
John Erickson
Hewlett-Packard Laboratories, USA
Eckhart Koeppen
Nokia, Finland
Jerome McDonough
New York University, USA
Murali Kaundinya
Sun Microsystems Inc, USA
Usva Kuusiholma
Avain Technologies, Finland
Gustaf Neumann
Vienna University of Economics and BA, Austria
Jon Mason
education.au limited, Australia
Craig Miller
ObjectLab, USA
Magda Mourad
IBM T.J. Watson Research Center, USA
Xavier Orri
Octalis S.A., Belgium
Steve Probets
Loughborough University, UK
Norma Richardson
Digital Concepts, USA
Peter Schirling
IBM, USA
Carlos Serrao
Adetti/ISCTE, Portugal
Mark Strembeck
Vienna University of Economics and BA, Austria

First ODRL International Workshop

Keynote Talks



DRM and the Web

**ODRL Workshop
Vienna, 22-23 April 2004**

*Rigo Wenning <rigo@w3.org>
W3C/ERCIM
Sophia Antipolis, France*

Plan

- Expectations
- Integration and Interoperability
- ID issues
- Enhancements and constraints
- DRM and Privacy
- DRM and Web Services
- DRM and P2P

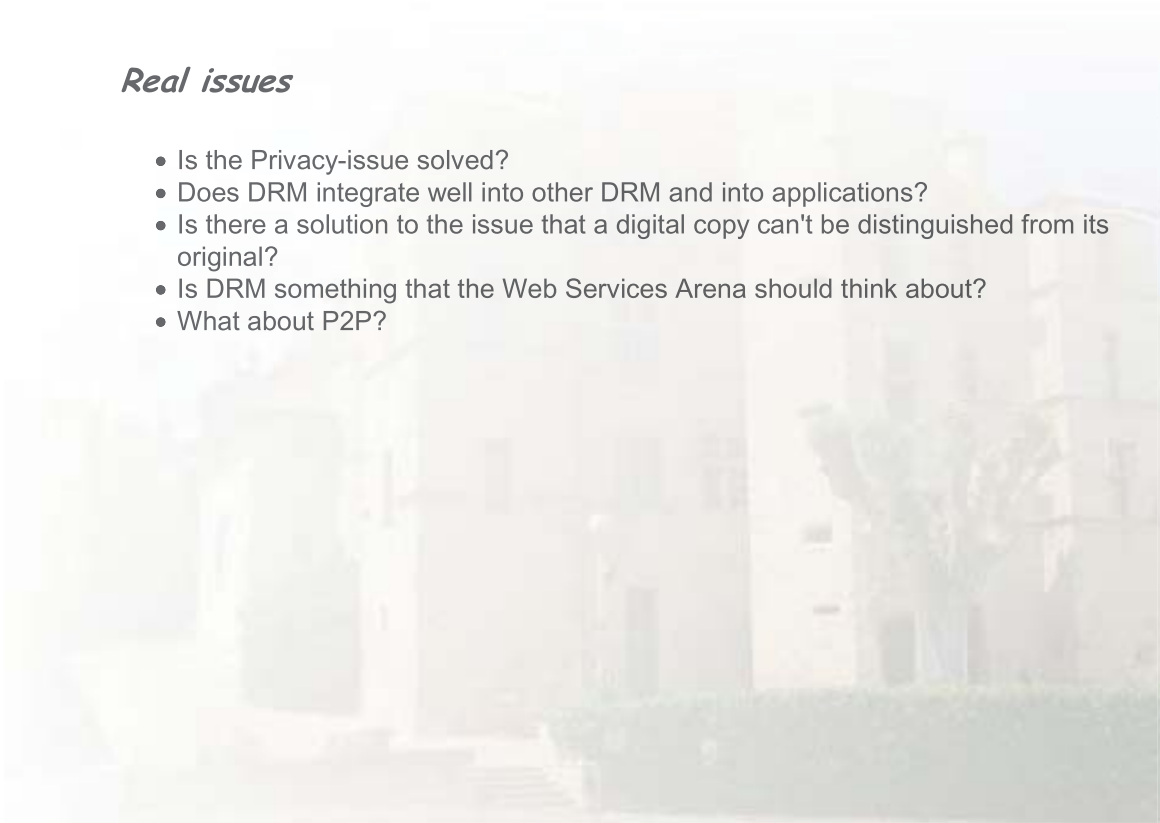


Expectations

- EC IST mentions lack of DRM as a roadblock?
- iTunes works!
- Piracy is (the only) reason for bad business in the music sector?
- Voices talk about high prices for bad food!

Real issues

- Is the Privacy-issue solved?
- Does DRM integrate well into other DRM and into applications?
- Is there a solution to the issue that a digital copy can't be distinguished from its original?
- Is DRM something that the Web Services Arena should think about?
- What about P2P?



The war is still going on

- Heavy lobbying to preserve a business model, where content is attached to tangible goods.
- Erosion of exceptions to copyright and Droit d'auteur
- P2P is getting more sophisticated and more of a closed group
- P2P starts using signature - mechanisms.

DRM and the war

- As long as the war is going on, DRM - Schemes have uncertainty about the semantics to use.
- In presence of unsecure semantics, application semantics are used.
- This will create a potential conflict with ever changing legal semantics.

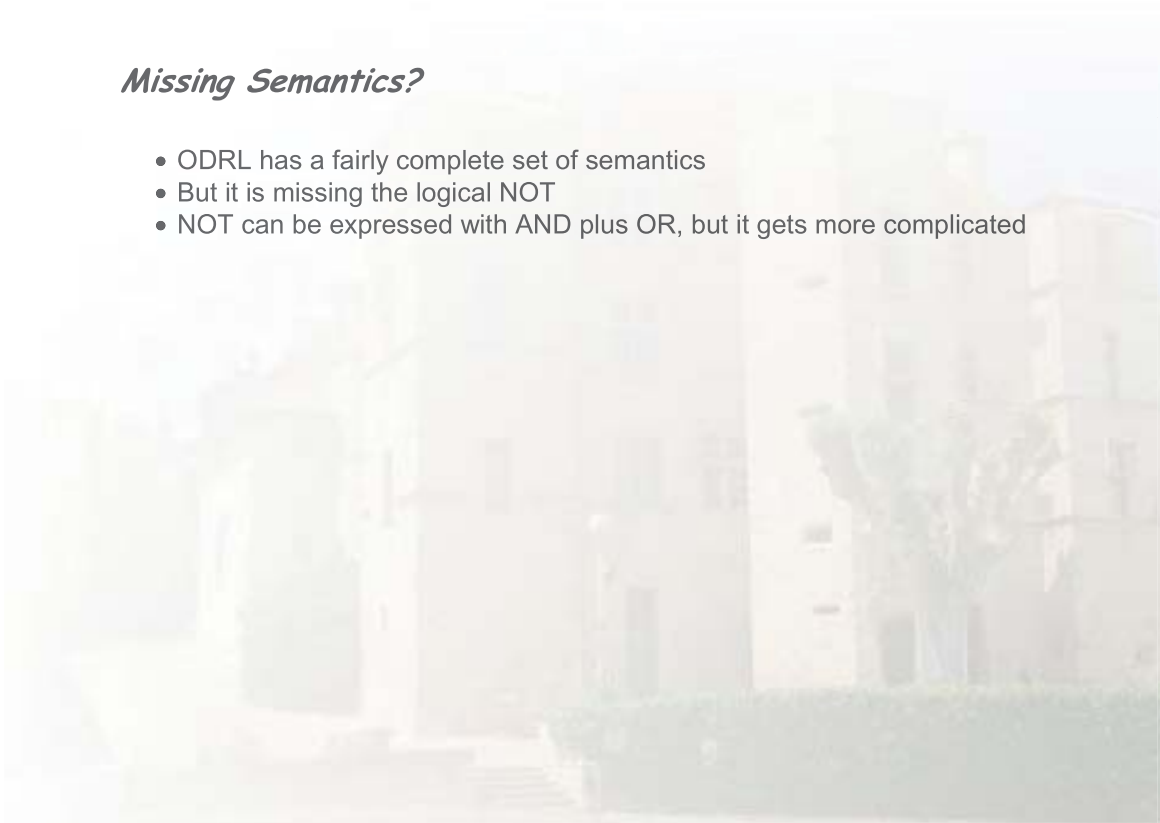


ODRL?

- ODRL is a framework to express semantics
- ODRL is not application semantics as it is a framework
- ODRL just perpetuates the semantic trouble
- ODRL will have to offer profiles to allow certainty

Missing Semantics?

- ODRL has a fairly complete set of semantics
- But it is missing the logical NOT
- NOT can be expressed with AND plus OR, but it gets more complicated



Example for NOT

Try to encode this simple sentence in ODRL:

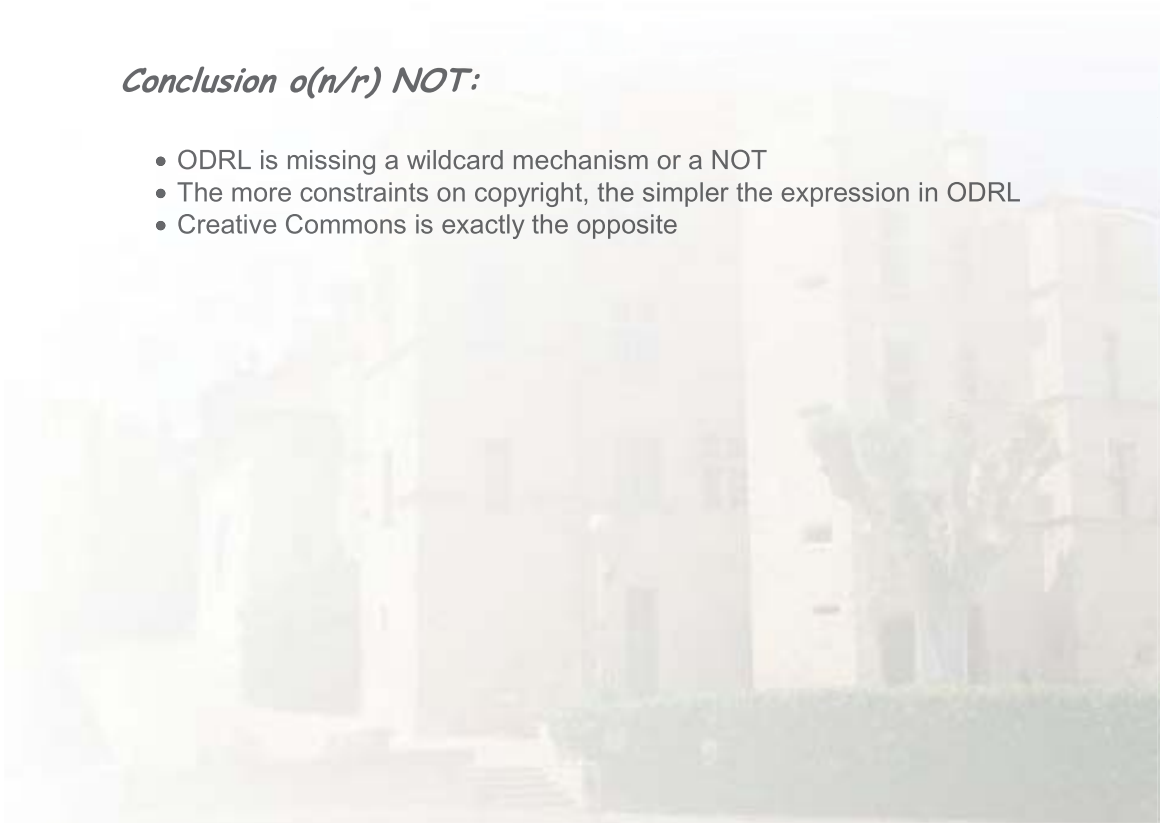
- You can do whatever you want with my work, but if you make money with it, give me 15%

ODRL says:

- A Permission that is not specified in any Rights Expressions is not granted.

Conclusion o(n/r) NOT:

- ODRL is missing a wildcard mechanism or a NOT
- The more constraints on copyright, the simpler the expression in ODRL
- Creative Commons is exactly the opposite



Interoperability and Integration

- Interoperability with other DRM languages
- Interoperability with the Web

Other Languages

Treat with both sides of the war:

- Interoperability with MPEG-21
- Interoperability with Creative Commons

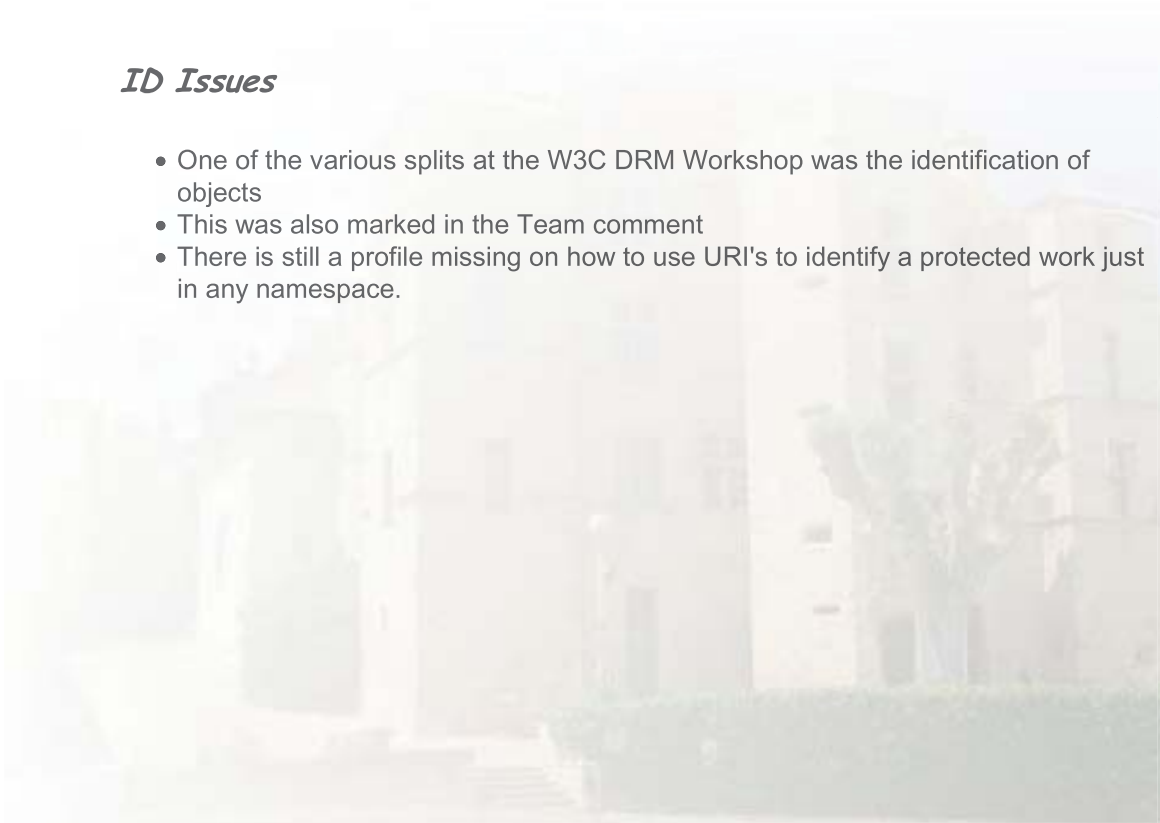
This has be explored but can not be considered done

Integration with the Web

- Difficult integration into HTML
- Would need some binding mechanisms like in P3P
- Bind DRM to HTTP (URI) or to HTML (link rel?)

ID Issues

- One of the various splits at the W3C DRM Workshop was the identification of objects
- This was also marked in the Team comment
- There is still a profile missing on how to use URI's to identify a protected work just in any namespace.



DRM and Privacy

- A big issue in the W3C DRM Workshop, especially for the enforcement engine
- Privacy is still a big issue
- Privacy might be misused as a term for general dissatisfaction

ODRL and Privacy

- Currently search for *Privacy* in ODRL gives *one* hit
- P3P 1.1 is developing a generic p3p attribute to bind a P3P Policy to a specific XML Element
- ODRL could use that solution to address privacy in a much better way.



DRM and Web Services

- DRM is just yet another set of metadata
- DRM should integrate into the Web Services Model
- SOAP and WSDL contain an advanced failure mechanism that could be used.

ODRL and Web Services

- ODRL is not defining a protocol but contains a binding
- The binding could be used to mix in to WSDL
- A Profile or Note would be nice



DRM and P2P

- Sounds like natural enemies, but it isn't
- DRM would allow to legitimize some of the P2P activities
- Psychology is key (see War above)
- Initiatives are missing to integrate effective DRM

ODRL and P2P

- Keep away from good/bad paradigms
- adjust your attacker model
- give people who want to do the right thing the opportunity to do the right thing, not just throw away the data



DRM and Exceptions

- Risk of losing our history
- Does DRM generate a right to hack for history?
- Library Privileges are not implemented yet
- Keep Society and the needs of a democratic society in mind while designing DRM.

ODRL and Exceptions

- This is still the weak point of ODRL and I still hope for a revision
- Missing wild cards and NOT-operator
- There is still research ahead
- The exceptions show a lot about the spirit of design



Merci bien

- I hope I gave you some points for discussion
- Presentation is available on the Web
- <http://www.w3.org/Talks/2004/04-odrl/>



OPERA

Eurescom Project 1207

Interoperability of
Digital Rights Management (DRM)
Technologies

Overview.

1. Introduction
2. System Overview
3. Architecture
4. Demonstrator
5. Conclusion

1. Introduction.

History, OPERA Mission, Why OPERA?
OPERA framework, Project facts, Participating partners.

History.

- Started with gathering requirements from the operators
 - Availability of content from content providers was the key bottleneck
- Gathered requirements from Hollywood studios
 - Identified the gaps in current DRM offerings

OPERA Mission.

The mission of the OPERA project was derived based on the requirements from the content value chain – content providers, operators/service providers and content users.

The objective of the project was the specification of an open DRM architecture that addresses the needs of:

- **Content Providers:** Addresses the needs of content providers to get a powerful and independent platform.
- **Operators:** Able to handle content of different media types, rights models and business models.
- **Users:** Addresses the needs of customers, to get an easy to handle, device and location independent service.

Mission Continued..

- Digital Rights Management (DRM) is the “central component” for delivery of digital content in eCommerce systems.
- Basic DRM requirements:
 - Reliable, simple, compositional and ubiquitous DRM infrastructure.
 - Available on every system, independent of platform, operating system and hardware.
- Current situation:
 - Several different, proprietary DRM systems.
 - Strongly altering market.
 - Different approaches to solve that problems.

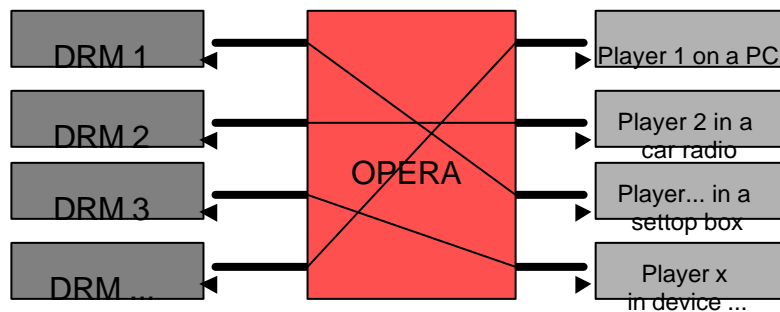
Why OPERA?

Limitations of current systems



Every DRM System works separately i.e. specialized in content types, players, end devices, system environments.

OPERA framework.



OPERA provides the framework to handle various DRM systems in a common environment.

Project facts.

Milestones:

- Architecture: May 03.
- Demonstrator: August 03.
- Project end: September 03.

Participating partners.

- DMDSecure.
- Exavio, Inc.
- Matáv Hungarian Telecommunications.
Company Limited.
- Sun Microsystems, Inc.
- T-Systems Nova GmbH, Berkom.

2. System Overview.

Concepts, Features, General Overview, 3rd Party Domain.

EURESCOM**11**

Concepts.

The main value of the OPERA framework is based on two conceptual approaches:

- The usage license is independent of the underlying DRM system.
- The usage license is bound to an user instead of (as is common) to a device.

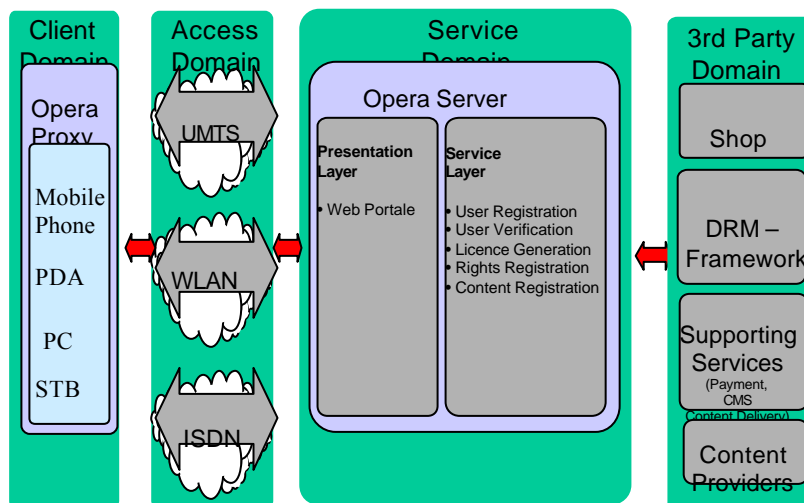
To achieve these functionalities Opera directly integrates major DRM systems and uses the already available DRM frameworks.

EURESCOM**12**

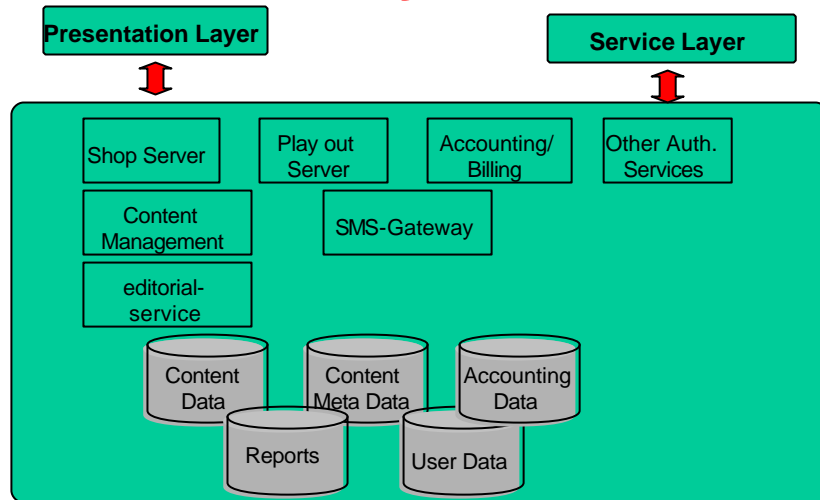
Features.

#	Feature	Description	Benefit
1	User bound licenses	Usage rules are independent of the end devices. Instead they are explicitly bound to a user.	Secure for content provider, ease of use for the user, "subscriber control" for operator.
2	Separation of rights from content	Rights management is completely separated from the DRM process and allows every content type that the underlying DRM system is able to support.	Content Providers and end users are able to use arbitrary DRM systems. The OPERA framework supports different DRM systems.
3	Support for multiple devices	Support for different end devices: (a) PC (b) Set-top Box (c) PDA.	Ease of use for the user, more revenue for operators and content providers.

General Overview.



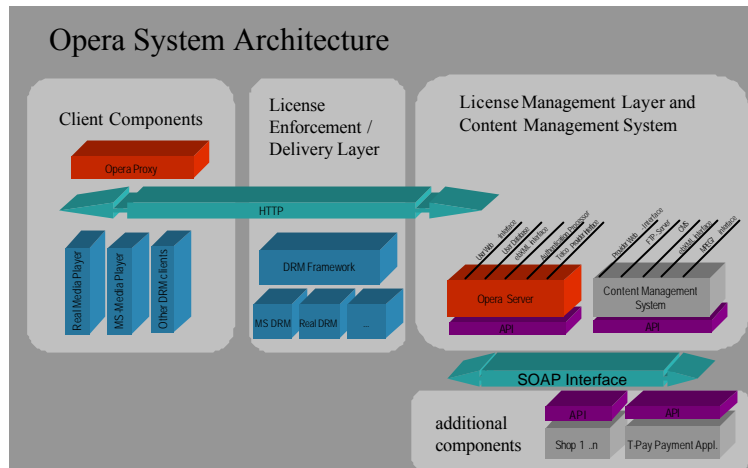
3rd Party Domain.



3. Architecture.

Component Overview, Client Components, Server Components, Additional Components.

Component Overview.



Client Components.

- Opera Proxy:
 - The Opera Proxy is responsible for connecting the user to the Opera Server and adds the users Authentication information to the incoming license request from the Player.
- DRM Player (or Viewer) application:
 - The media player or viewer allows users to consume the content they bought. This component is dependent on the particular used DRM system.

Server Components.

- The license management layer (Opera Server) is responsible for management of the rights the users have obtained. They are described through an Opera license and stored in a database on a per-user basis.
- The license delivery layer is responsible for delivering licenses of the underlying proprietary DRM-systems to the user's machines. Examples for such systems are DMDfusion and the DWS (Digital World Services) system.

Server Components.

- The enforcement layer is realized by one of the proprietary DRM-systems available on the market. Enforcement layer - enforces the compliance of usage rules (more exactly: those of the delivered license) by using mechanisms like encryption.
- Content management layer – This layer represents the content utility interface and a media asset management system. For the Opera project the focus is the additional possibility to specify usage rules for the Opera Server.

Additional Components.

- Shop Application: The shop application is used to buy content over a web based application. The Opera management layer uses the shop also to verify that the content was bought by the user who request the license.
- Payment Application: The payment application is often part of the shop application but may also be a separate application e.g. T-Pay or FirstGate.
- Content delivery system: – architecturally both download and streaming capability shall be supported. But for the demonstrator only download will be supported.

4. Demonstrator.

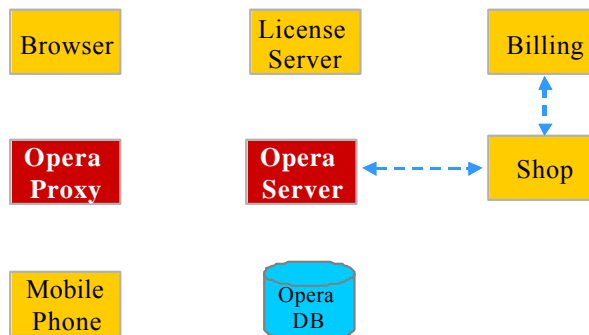
Components in demonstrator,
Work flow for Obtaining Licenses

Components.

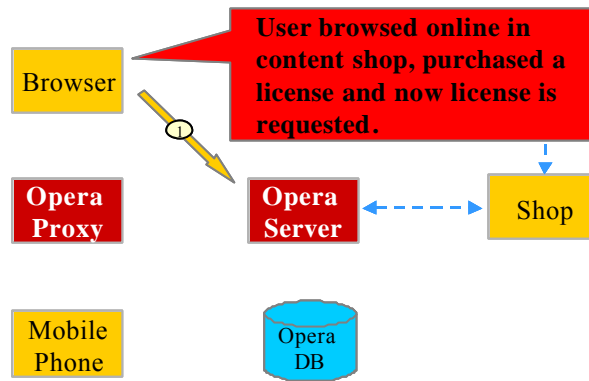
- License Delivery Layer (DRM Frameworks)
 - DMD Secure
- License Enforcement Layer (DRM Systems)
 - Digicont DRM from SDC AG.
 - Microsoft DRM
- Devices
 - PDA
 - Personal Computer
 - Set Top Box
 - (Mobile phone)

Workflow.

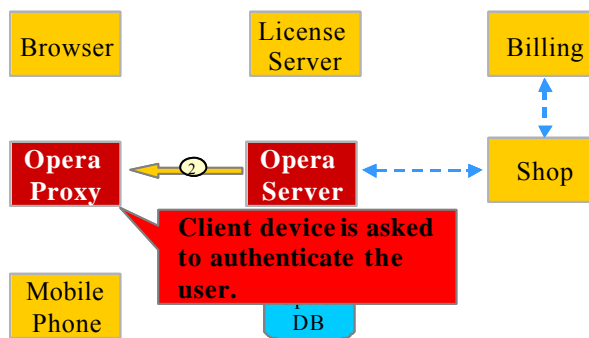
Components in Opera DRM solution



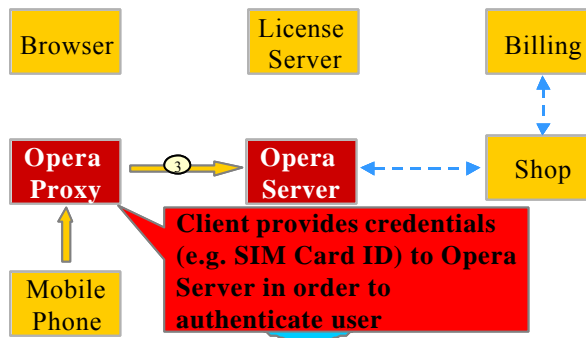
Workflow.



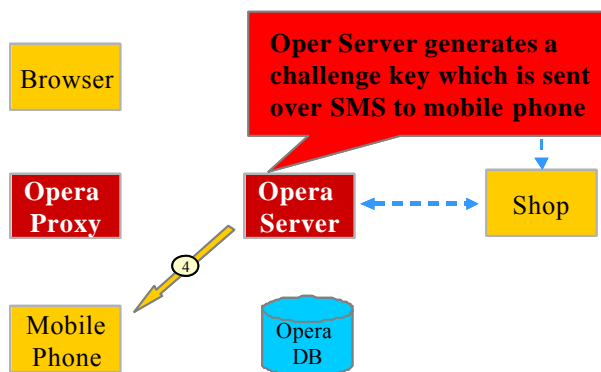
Workflow.



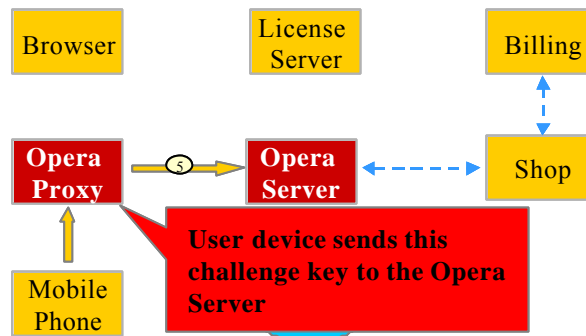
Workflow.



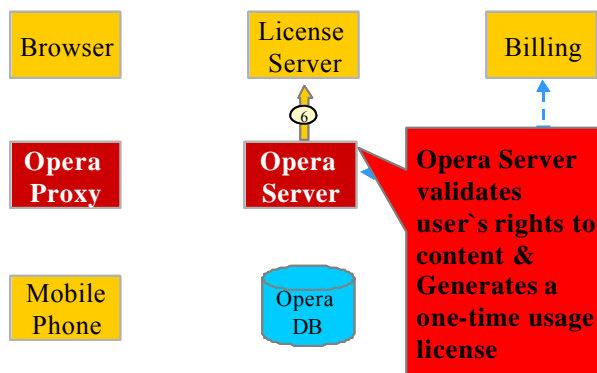
Workflow.



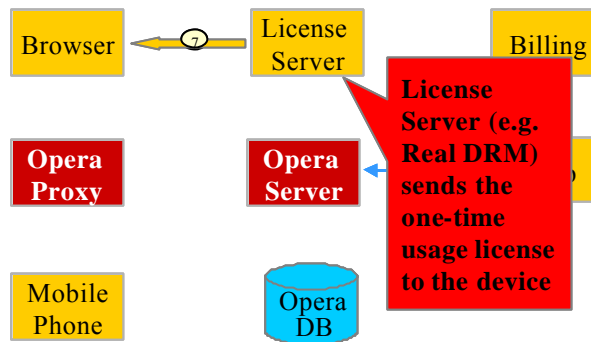
Workflow.



Workflow.



Workflow.



5. Conclusion.

- The Opera project has been successful in providing an innovative DRM architecture, and in implementing a demonstrator that satisfies all the requirements of content value-chain stakeholders.
- Tremendous interest from operators, service providers and content providers as well as technology providers in real deployment of Opera



OMA Secure Content Delivery for the Mobile World

ODRL Workshop, Vienna

Dr. Willms Buhse

Vice Chair, OMA Download and DRM group



Agenda

- OMA Digital Rights Management background
- OMA DRM v1 for light media
- New DRM v2 for premium content
- Benefits for content providers
- Benefits for end consumers
- Applying OMA DRM to protect content
- Summary
- Questions

Why OMA started working on DRM- 2001

- **A response to strong market demand**
 - **Content sales** to mobile devices becoming lucrative
 - Phones coming to market able to support light media
 - Content and service providers wanted to protect their investments
 - **Levels of protection** needed to commensurate with different content value
 - Need for a **mass market solution**:
 - Timely and inexpensive to deploy
 - For mass market mobile devices (not just high -end)
 - Did not require costly infrastructure to be rolled out



Copyright © 2004 Open Mobile Alliance Ltd. All Rights Reserved.

OMA DRM – Stakeholders

- **OMA DRM is developed by the entire mobile value chain for the mobile industry**
 - Content Providers
 - Information Technology Companies
 - Mobile Operators
 - Wireless Vendors
- **About 50 Companies participating in monthly meetings and weekly conference calls**
- **Consolidated from DRM standardization at 3GPP, WAPForum etc.**
- **Liasons created with industry organizations such as MPEG, RIAA, 3GPP, etc.**

Copyright © 2004 Open Mobile Alliance Ltd. All Rights Reserved.



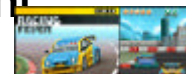
OMA DRM1.0 for Light Media Content

- **OMA DRM 1.0 was created to meet these market requirements**
 - Targeted protection for light media content
 - Three levels of functionality
 - **Forward Lock** prevents content from leaving device
 - **Combined Delivery** adds rights definition
 - **Separate Delivery** provides content encryption and supports legitimate viral distribution (SuperDistribution)
- **Specifications rapidly developed to reduce time-to-market**
 - Delivered specifications for implementation in Nov, 2002
 - OMA "Test Fest" in Seattle in Nov 2003 to test DRM v1 interoperability
 - Over 50 hand sets available in the market today
 - Several vendors have announced servers supporting OMA DRM 1.0
 - Operators are integrating OMA DRM into their infrastructure

Copyright © 2004 Open Mobile Alliance Ltd. All Rights Reserved.



OMA DRM 2.0 for Premium Content



- **DRM solution is evolving with the mobile industry**
 - High bandwidth cellular networks becoming widely available
 - Mobile devices with removable media and larger color screens support downloading and streaming rich media
 - Content and service providers eager to release rich audio/video content and applications
- **Greater security and trust management required to protect high value content**
 - Need to ensure target device can be trusted to keep content and trade secrets safe
 - Need greater security to prevent content from leaking out during distribution



Copyright © 2004 Open Mobile Alliance Ltd. All Rights Reserved.



OMA DRM 2.0 Benefits for Content Providers

- **Enhanced security**
 - Individually encrypted rights object using device's public key to provide cryptographic binding (to SIM/WIM)
 - Integrity protection for content and rights objects
- **Explicit trust mechanisms**
 - Mutual authentication between device and rights issuer
 - Device Revocation: Rights issuer can identify device revocation status
- **Secure multicast and unicast streaming**
 - Working with 3GPP and 3GPP2 on packetised file format for protected streaming and progressive download
- **Wide variety of business models**
 - Metered time and usage constraints
 - Subscription rights for content bundles
 - Gifting
- **Support for Peer-to-Peer and Messaging**
 - SuperDistribution: Viral marketing and reward mechanism

Copyright © 2004 Open Mobile Alliance Ltd. All Rights Reserved.



OMA DRM 2.0 Benefits for End Consumers

- **Advanced content management**
 - Storage and backup: move content and rights to remote or removable storage and later restore to device
 - Multiple devices: Move content and rights objects easily between several devices owned by a user (2nd phone)
- **Sharing between multiple user within domain**
 - Domain concept for sharing between devices in the same domain (e.g. family)
- **Unconnected devices**
 - Copy to SD card for a mobile music player
- **Complementary Preview**
 - Constraints for superdistributed content before purchase
- **Export to other copy protection schemes**
 - transfer music to DRM-enabled Settop box or computing device

Copyright © 2004 Open Mobile Alliance Ltd. All Rights Reserved.



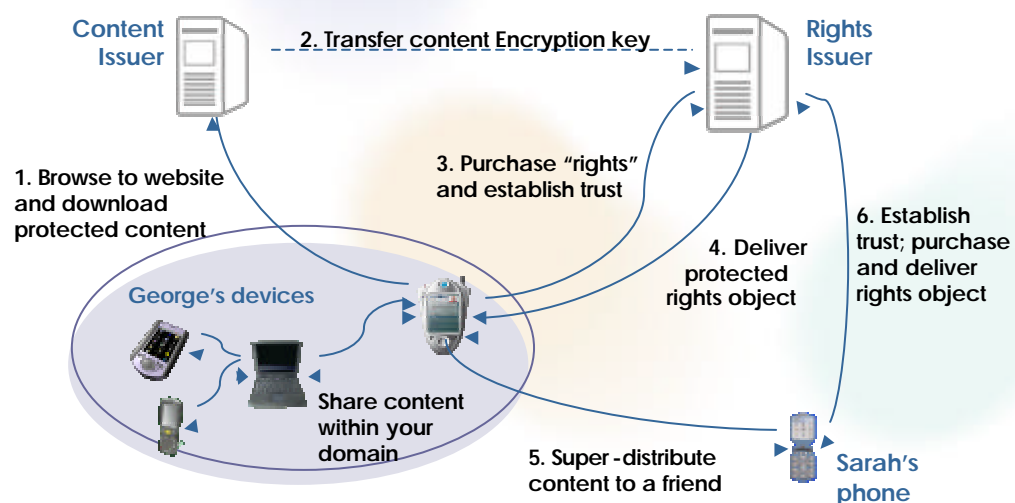
Applying DRM to Protect Content

- **Steps for distributing DRM protected content**
 - Encrypt the content and package in DRM Content Format
 - Assign permissions and constraints during consumer purchase transaction
 - Consumer receives content and “rights” and begins using the application
- **DRM enables viral distribution of content**
 - Consumers can send the protected content to all their friends that have OMA DRM enabled mobile devices
 - Their friends, in turn, purchase the permission to consume the content
 - Content and service providers make money through “word of mouth” advertising

Copyright © 2004 Open Mobile Alliance Ltd. All Rights Reserved.



Example of OMA DRM Deployment



Copyright © 2004 Open Mobile Alliance Ltd. All Rights Reserved.



Future Directions for OMA DRM

- **OMA DRM will continue to evolve**
 - New ideas and experiences will drive OMA DRM in new directions
- **OMA proactively seeking input from content industry and developer fora**
 - Ensures OMA DRM satisfies their business models
- **Please join the OMA DRM sub-working group and help define the future of DRM**
 - Browser and Content Group, Download+DRM
 - OMA-DOWNLOAD is the email reflector



Copyright © 2004 Open Mobile Alliance Ltd. All Rights Reserved.



Questions?

www.openmobilealliance.org



First ODRL International Workshop

Papers

A Pervasive Application Rights Management Architecture (PARMA) based on ODRL

Dominik Dahlem, Ivana Dusparic and Jim Dowling

Distributed Systems Group
Department of Computer Science
Trinity College Dublin

Dominik.Dahlem@cs.tcd.ie, Ivana.Dusparic@cs.tcd.ie, Jim.Dowling@cs.tcd.ie

Abstract. Software license management is currently expanding from its traditional desktop environment into the mobile application space, but software vendors are still applying old licensing models to a platform where application rights will be specified, managed and distributed in new and different ways. This paper presents an open-source pervasive application rights management architecture (PARMA) for fixed network and mobile applications that supports the specification of application rights in a rights expression language (REL) based on ODRL. Our rights specification model uses aspect-oriented programming to generate modularized rights enforcement behaviour, which reduces development time for rights models such as feature-based usage rights and nagware. PARMA manages vendor and customer application rights over multiple platforms using a web services architecture and a container model on the client-side. The container model also supports the integration of services such as payment and encourages the super distribution of the rights object with associated default (evaluation) rights.

Keywords: Application Rights Management, Pervasive Computing, ODRL, Aspect-Oriented Programming.

1 Introduction

Software licensing has been a relatively successful example of rights management based on enforcement. The movement of license management from its traditional desktop environment into the mobile application space has so far not resulted in the appearance of new licensing architectures that utilise mobile-device features such as unique device identification, super distribution of applications over multiple communication channels (e.g. Bluetooth and GPRS) and the separate delivery of application usage rights [01]. Such a licensing platform must also deal with mobile-specific problems such as disconnected operation and resource constrained environments. The DRM community has been more successful at identifying some of these features and problems than the software licensing community, but due to its focus more on rights management of content than applications existing DRM specifications and architectures have left many of the aforementioned issues unresolved. Additionally, there is a requirement for applications vendors who distribute both desktop and mobile applications for an integrated (pervasive) licensing or application rights management architecture that will manage their applications over many platforms.

One of the major limitations of existing licensing systems is how licensing behaviour (application usage rights) is specified and integrated into applications. Existing distributed software licensing platforms [02] and standards [03] support the specification of licensing behaviour in applications by providing language- and platform-specific bindings to their licensing architectures. These solutions are typically aimed at vertical markets, require programmer knowledge of proprietary application programming interfaces (API) and do not provide the integration and flexibility required for more ubiquitous software deployment. This all leads to significant problems in using existing licensing technologies in the mobile application domain.

This paper investigates how the use of a declarative programming language (or a rights expression language (REL)) for the specification of a user's application usage rights (or license terms) can greatly improve the application development process. We introduce a REL that extends ODRL [04] and supports the specification of fine-grained application rights, such as specifying access rights at the

object and method level in object-oriented applications. The REL is integrated into applications using aspect-oriented (AO) technology [05]. AO technology supports for the separation of rights management concerns from application concerns. Calls to a licensing API that are scattered [05] around application source code can be encapsulated into single, manageable aspects producing better modularization, and hence improved maintenance, of rights management for applications. The REL also supports the use of services external to application. For example, application usage rights can be associated with payment services allowing users with an evaluation copy of an application to acquire better usage rights for it.

The rights enforcement architecture provides a container model with plug-in services that mediates application interaction with the services. The container is designed for resource constrained devices and supports a base set of services, including application rights object management, security and the separate delivery of rights objects to applications. This paper also describe how the application rights management architecture handles the aforementioned pervasive computing and mobility problems. In particular, we introduce a novel audited pay-per-use model designed specifically for mobile phone application software.

Section 2 of this paper introduces and motivates the open-source PARMA model and Section 3 describes our rights expression language. Section 4 details the mapping of our REL to aspects and the weaving of aspects into applications. Section 5 describes the rights enforcement architecture and we conclude with the status of our work and future work.

2 Background and Motivation

An application rights management architecture for pervasive computing environments must encompass a broad range of platforms and stakeholders. Application vendors need to be able to specify and organize rights, application distributors need to be able to sell rights to consumers and client devices must be able to run the applications while enforcing the rights and allowing the modification of those rights. Application rights can be defined using traditional software licenses or with special purpose *rights expression languages* (REL) such as ODRL or XrML [06]. However, for fine-grained specification of application rights, i.e. at the application feature level, there is a requirement for rights to be associated with application functionality, such as at the method invocation level for objects. Traditional DRM is concerned with specifying whether or not a given user has a right to view the passive content on the given device at the given time. Application DRM differs because users can be allowed to execute one part of the application but not the other. For example, a user can create documents, but not save them with a demo version of the application. Or users are allowed to execute all parts of the application but billed differently for using different features, as opposed to being billed per application execution/viewing as is the case with passive content DRM. Therefore, DRM concerned with active content has to be more extensive than DRM over passive content.

Also, users will often receive evaluation versions of software or versions of software that degrade over time [07] that they can later upgrade to a full version. For this reason, it is necessary that application rights management architectures allow rights to be upgradeable and that support is provided for the integration of external services such as payment. In our opinion, an application rights specification language has to provide support for the association of changes in rights with use of external services, such as payment.

The PARMA model is being developed according to open-source principles and provides support for the specification of application rights and the mapping to a rights enforcement architecture that also allows the integration of external services using a container model. The PARMA architecture also addresses mobility-specific issues such as disconnected operation, resource constrained devices, super distribution of applications (e.g. over Bluetooth), and the separate and combined delivery of rights. A new audit-based rights model is specifically introduced for mobile devices with occasional connectivity, to account for possibility of loss of network coverage and for the costs of GPRS connection/traffic. Instead of checking application rights and initiating payment at the time of application execution, information about application usage is safely stored on the device. Once device is back on the network, it can transfer the data to the DRM server over GPRS, or via free short range protocols like Bluetooth or 802.11, and users can be billed for the application usage according to their log. A backend architecture is implemented as a web service [08] and allows vendors to manage application rights over a pervasive computing environment.

2.1 Motivation

Typical and popular applications for mobile devices differ from desktop applications due to issues such as user-interface disparities, ease of use and cost of network connectivity [09]. Retail applications for mobile devices, including games, typically have a shorter life-span than their desktop or console equivalents and customers appear less willing to make large up-front investments in expensive software licenses [10]. However, some vendors have had success using novel application licensing solutions such as FleetOnline [11], who distribute their base software for tracking the location of vehicle drivers in the U.K. free of charge and charge per location request or message using sms billing.

It is our belief that the current model of restricting the distribution of mobile application software is delaying the adoption of such software, as users appear unwilling to pay either the high costs of application downloading over GPRS networks or the full-cost license for the application software. For example, existing DRM architectures, such as Nokia's [12], encourage the use of forward lock to prevent the unauthorized further distribution of applications. However, the low cost of distribution of digital media and applications over Bluetooth contrasts sharply with the high cost of GPRS. Given a choice, users would prefer to acquire applications over a communication channel that is free instead of paying for distribution as well as the application or content.

In contrast to existing DRM systems, however, our model encourages the removal of restrictions on distributing application software. Application usage rights are determined during application usage by software. The architecture adapts licenses to the user's usage rights and integrates payment and other container, including a payment service. In particular for the retail and gaming software markets, this model will encourage users to distribute applications that come with default usage rights. More rights can be acquired after application installation by the user without the need to download a new copy of the software.

Application vendors have, in recent years, moved into the mobile application domain and are now releasing products for multiple platforms. With this comes a requirement to centrally manage licenses, payment and application rights. PARMA also provides a unified (or pervasive) application rights management architecture for both fixed network and mobile applications. The integration of application rights management architectures over multiple platforms and devices is achieved using a web services backend architecture. Web services are platform and language independent and enable the easy integration of the application rights management architecture with other backend systems, such as asset management and billing systems.

3 The PARMA Rights Expression Language

Many DRM systems use rights expression languages to define rights on the content usage. Rights expression languages are computer-readable, very often in XML format, and provide information on content, owners of the content, and rights of usage for content users. Traditionally DRM is concerned with copyrighted content (music, video, etc) but recently rights expression languages, such as Open Mobile Alliance's (OMA) REL [13], have been used to define rights on mobile software applications as well [12].

The PARMA architecture primarily manages application usage rights on mobile devices and is not concerned with usage rights on other types of mobile content (music, video, ringtones etc.). Existing DRM systems that support applications as a possible content type are limited and support only simplified models such as whether the user/device is permitted to run the application and whether the user/device is permitted to forward the application to another user/device [12]. PARMA's focus on application usage rights will provide more fine-grained specification of rights over how the application can be used, as well as integration with payment and other services.

PARMA supports an extensible set of rights models including traditional license for unlimited usage, named user license, time-limited, feature-based model, subscription-based, pay-per-use where payment can be in real-time or audited, node-locked, and concurrent usage rights models. We believe that feature-based and pay-per-use models will be most attractive to application vendors and customers alike. All of these rights models introduce requirements on a REL used to define application usage rights and permissions.

Open Mobile Alliance's (OMA) REL Version 1.0 [13] is designed specifically for mobile devices and its schema consists of a subset of ODRL elements extended with elements specific for mobile environments. The OMA REL is the basis for DRM systems by both Nokia and SonyEricsson.

PARMA REL, designed in part for mobile devices, reuses some of the permissions and constraints from the OMA REL (execute, datetime, count etc) but beside simple permissions and constraints PARMA REL requires information about parties involved in issuing rights, amount and means of payment for the application, as well as finer definition of constraints. OMA REL by itself proves too narrow for PARMA REL mobile software licensing requirements and needs to be extended.

ODRL [04], as a superset of OMA REL, introduces some new elements that define parties involved in defining the permissions on content and payment elements PARMA can reuse. ODRL elements, however, do not fully cover PARMA requirements and we required adaptations to ODRL to meet the PARMA REL requirements. ODRL can be adapted in two ways: either by modifying or by extending the ODRL schema. Modification of the schema would allow for finer and more strict schema level validation of elements but we decided to perform this validation separately and instead extend ODRL to stay compliant with other DRM architectures implementing it. PARMA REL, as an extension of ODRL, is backwards compatible with ODRL and therefore with OMA REL. In that way, PARMA REL is compatible with current DRM systems incorporated in mobile phones.

3.1 PARMA REL Schema

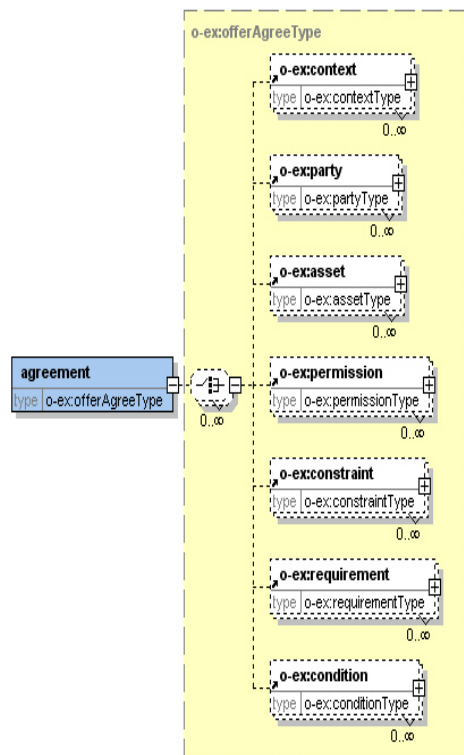


Figure 1 : ODRL Agreement Element

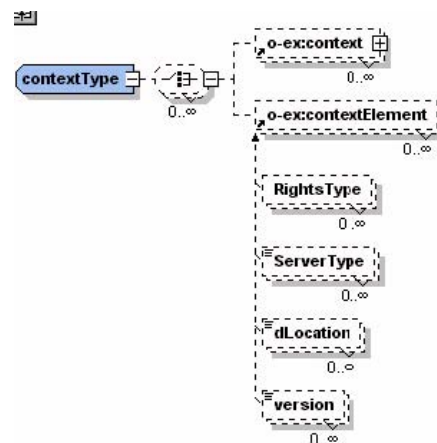


Figure 2 : PARMA Context Element (Extended ODRL)

The first element PARMA REL adds onto the ODRL schema is a rights type. In the PARMA schema the RightsType element is defined as a contextType element and it can contain the following values: NodeLocked, Concurrent, PayPerUse-Audit, PayPerUse-RT, Unlimited, TimeLimited, FeatureLimited, NamedUser, and Subscription. These are rights types currently supported by the PARMA architecture and every agreement issued by PARMA will fall in one of these categories. However, RightsType is open for extensions. Since a license model is directly related to offer and agreement types, the ideal case would be to make this element mandatory child element of context element that is child of agreement or offer type. However, ODRL uses the same definition of context element regardless of whether it is a child of agreement, offer, party, permission etc, therefore it cannot be defined as mandatory and this validation will have to be performed separately from the validation

against schema. A tool will be provided that allows application usage rights to be specified and allows validation of those rights against the schema and PARMA validation rules. After successfully validating the rights the tool will convert the rights to aspect code and weave it into the application.

```
<xsd:element name="RightsType" type="xsd:string" substitutionGroup="o-ex:contextElement"/ >
```

Rights objects in PARMA REL must be identified by their location therefore we need an element to store this information in the rights file. The ODRL definition of context elements includes the element `dLocation` that represents the digital location of the event/entity. PARMA REL uses this element to store a URI that refers to the rights object. In PARMA REL this element is mandatory and validation will again have to be performed against PARMA validation rules rather than against the schema. The context element also includes elements for the version name of the entity, used by PARMA REL to specify name and version of the application, although `dLocation` alone is often enough to uniquely identify the rights object.

Another group of elements introduced is the URI and type of the rights server.

```
<xsd:element name="ServerType" substitutionGroup="o-ex:contextElement">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="EDS"/>
      <xsd:enumeration value="VDS"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

The rights server is responsible for delivering application and usage rights, updates of usage rights, collecting audit data and billing for the application usage. Information on this server, following the ODRL structure, best fits within the party element because the server will represent a party that is either the owner or distributor of the application, or possibly an authorized enterprise that will distribute licenses to its employees. In the case that the party is the owner or distributor we could put this information in the `rightsholder` element, but in the situation where the party is an enterprise that would not be accurate. Therefore, PARMA REL extends the context element with the new element `ServerType`, and stores the information about the location of the server in the ODRL-defined element `dLocation`. Similar to the `RightsType` element, we would like to restrict the `ServerType` element to occur only within the context element that is a child of a party element, but again this validation is left to be done against PARMA validation rules before rights objects are converted to code. The `ServerType` element is mandatory (not by the schema but by PARMA rules validation) and can contain only two values- EDS, in the case of the server being Enterprise DRM Server, hosted by the company licensed to use the application, and VDS, in the case where the application is communicating with a Vendor DRM Server to obtain the rights. This information is important for configuration of the client side DRM engine that behaves differently depending on the type of server it is communicating with.

```
<o-ex:party>
  <o-ex:context>
    <o-dd:dLocation>URIofDRMServer</o-dd:dLocation>
    <parma:ServerType>EDS</parma:ServerType>
  </o-ex:context>
</o-ex:party>
```

Introducing these elements concludes the general part of all PARMA requirements – other elements PARMA included and/or distributed in already existing ODRL elements are directly related to rights information and usually specific to a license type declared in the new `RightsType` field.

3.2 PARMA Schema and Licensing Model Implementation

ODRL has defined an extensive list of permissions on content, however the only permission applicable to software usage is “execute”. All the constraints PARMA REL uses will be defined as constraints on the execute permissions.

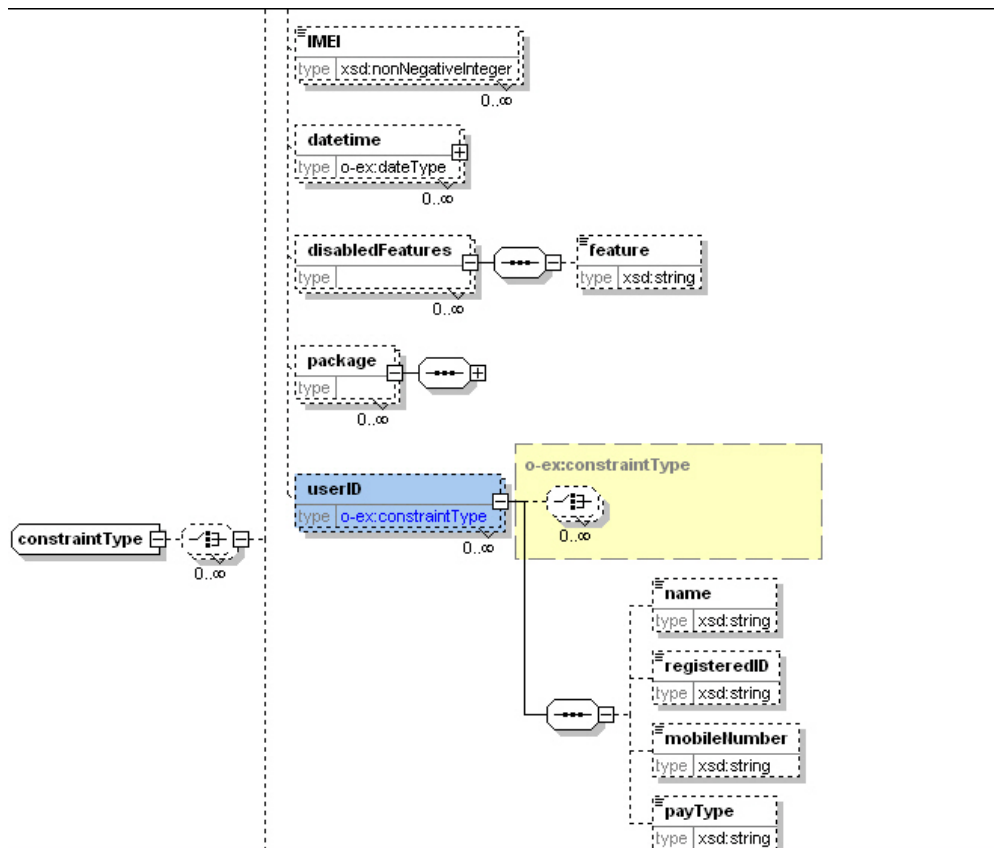


Figure 3 : PARMA Constraint Elements (Extended ODRL)

The only constraint that is mandatory (and its presence is ensured by validation of PARMA requirements) for all values of RightsType is the ODRL datetime constraint, that specifies start or end date, or both, for the period in which the user/device is permitted to execute the application. PARMA supports traditional usage rights models such as node-locked and named-user models by introducing restriction elements to uniquely identify the user. If a node is a user it is identified by the new restriction element IMEI number (International Mobile Equipment Identification number) and the element userID that optionally contains the user’s name or his mobile phone number.

- **Feature-Based Model**

Another DRM model PARMA REL is implementing is a feature-based model, where certain features of the software application will be enabled or disabled depending on the agreement. For example, for a game application, a multiplayer over Bluetooth option might be disabled for a demo version of the application. PARMA REL allows the specification of such features using a new extension to the permission element called “disabledFeatures”. It is a complex type that contains a sequence of zero or more “feature” elements that contain the name of the disabled feature.

```

<xsd:element name="disabledFeatures" substitutionGroup="o-ex:constraintElement">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="feature" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

However, PARMA also allows for much finer grained control over parts of the application. PARMA schema defines a “returnType packageName.className.methodName(argumentTypes)” structure as an extension of the requirement element. This structure contains the signature of the method within the application source code where the rights check should be performed. Permissions specify the signature of the method and the class/package it belongs to and the licensing code before calling this method

within the application checks the PARMA permissions to allow or deny the method call to be performed. In this way PARMA can, for example, enable game players with a demo license to play the game up to a 3rd level, but for higher levels prompt them to obtain an upgraded full license that supports all levels.

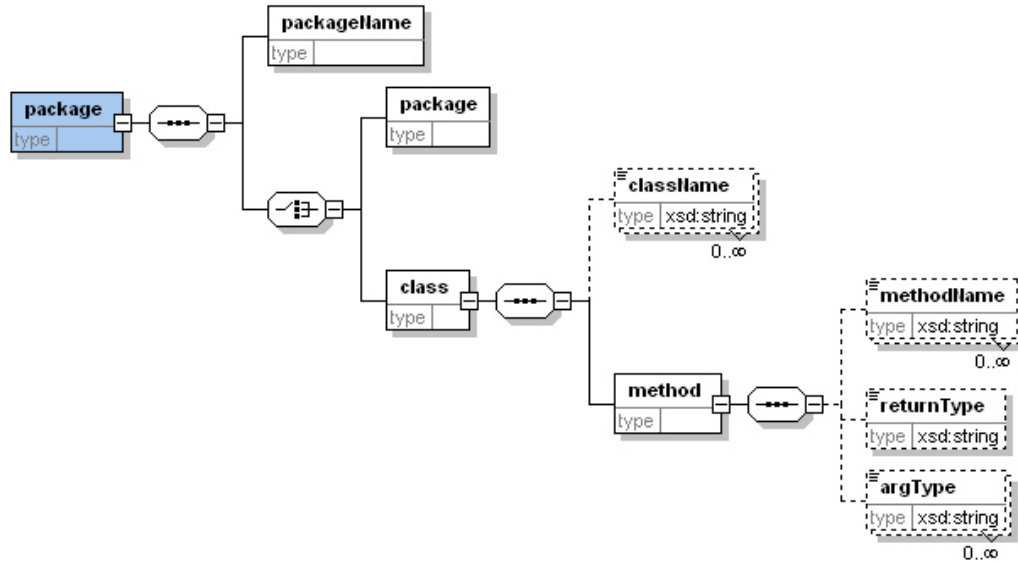


Figure 4 : PARMA Package Element

- Pay Per Use models (Audited and Real-Time)

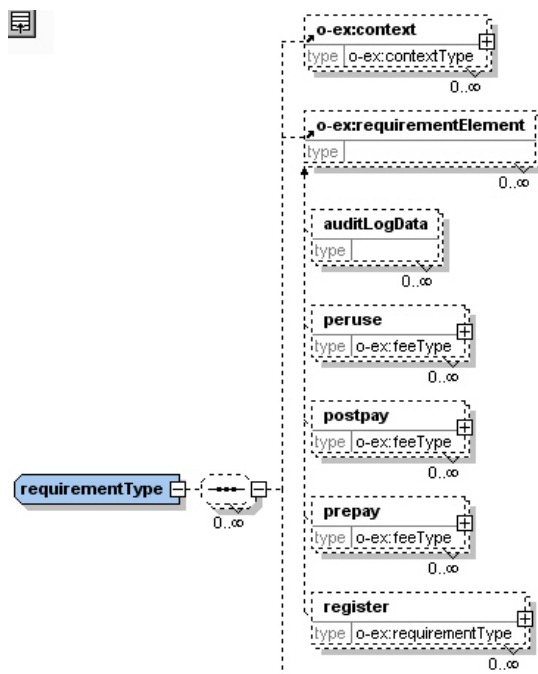


Figure 5 : PARMA Requirement Elements (Extended ODRL)

One of the features that distinguishes PARMA from other similar projects is the very flexible pay-per-use policy. It can accommodate real-time pay-per-use where users make payments immediately

before or after running the application for the certain amount of time or certain number of times. In the audited pay-per-use model information on the application usage is stored in a secure storage area of the mobile device and is occasionally transferred to the EDS and from there to VDS, or directly to VDS. Billing information is produced based on the usage data from this log. The PARMA schema defines the complex element auditLogData as an extension of requirementElement that has several boolean attributes that represent required parts of the log. All attributes are optional and their default values are true.

```
<xsd:element name="auditLogData" substitutionGroup="o-ex:requirementElement">
  <xsd:complexType>
    <xsd:attribute name="dateTime" type="xsd:boolean" use="optional" default="1"/>
    <xsd:attribute name="duration" type="xsd:boolean" use="optional" default="1"/>
    <xsd:attribute name="accumulated" type="xsd:boolean" use="optional" default="1"/>
  </xsd:complexType>
</xsd:element>
```

Other elements required for the implementation of PARMA payment policies are already defined in the ODRL schema. ODRL accounts for both prepaid and postpaid payment types and has corresponding elements to store this information. Another ODRL payment element PARMA is using is “peruse” where price per use (in PARMA’s case it is per execution) of the application is stored. Per use can be combined with both prepaid and postpaid payment types, and also both with real-time and audited approaches. Some payment types will require the user to be registered in advance with the rights server, so elements userID, register and dLocation of the registration server will be reused here as well.

```
<o-ex:requirement>
  <o-dd:peruse>
    <o-dd:payment>
      <o-dd:amount o-dd:currency="EUR">2.00</o-dd:amount>
    </o-dd:payment>
  </o-dd:peruse>
  <o-dd:prepay>
    <o-dd:payment>
      <o-dd:amount o-dd:currency="EUR">2.00</o-dd:amount>
    </o-dd:payment>
  </o-dd:prepay>
</o-ex:requirement>
<o-ex:requirement>
  <o-dd:register>
    <o-ex:context>
      <o-dd:dLocation>URlofApplication</o-dd:dLocation>
    </o-ex:context>
  </o-dd:register>
</o-ex:requirement>
```

4 Mapping REL into Application Rights Management Code

4.1 Introduction

In this section we explain the association of PARMA rights with the licensed software application (rights object). Rights have to be tightly-coupled with the application code in order to support fine-grained usage rights models. Before the particular method in the application is called, validation has to be performed that rights specifications allow for this call to be made. Generally, a rights object requires specific calls to a DRM system to be added to the code of the original application in order to perform validation, and these calls are usually scattered over several classes throughout the application. However, such modifications of the original source code to add rights enforcement are very inflexible and impose extra work on application developers.

As an example for the comparison we will describe rights enforcement using XSLM, a licensing standard proposed by the Open Group [03]. The first step of an XSLM-compliant application is to determine what level of application usage rights the server supports. Next, it has to establish the rights enforcing session and request the rights. After these calls are successfully completed, the user is allowed to run the application. During the execution, the application occasionally makes API calls to record application data, log the application usage, and confirm that the rights agreement is still in use. Before the application is closed, calls have to be made to release the rights in order to end the rights enforcing session. XSLM-compliant advanced DRM systems require rights objects to make at least eight API calls during the execution and in standard object-oriented applications. These calls cut across many different components of the application (on start up, during the execution and before application termination) and have to be added by the programmer with knowledge of licensing system APIs.

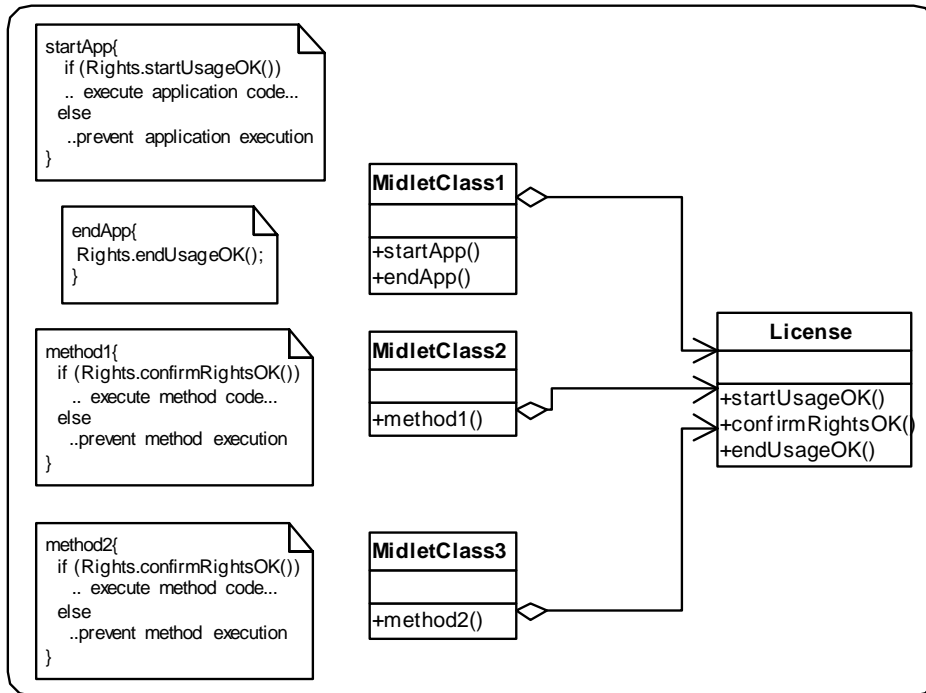


Figure 6 : Scattered Calls to Rights APIs

4.2 Generating code from Rights Files

PARMA uses aspect-oriented development techniques to add rights enforcing code to the application in a non-intrusive manner. Aspect code specifies join points in the original source code where rights enforcing APIs should be called. All the enforcement-related code is encapsulated in the aspect and it is woven into the original application at compile time using an Aspect Oriented Software Development (AOSD) tool, such as AspectJ [14] for Java applications. AOSD does not require any changes to the original source code and therefore it is very easy for developers to add rights enforcement to their application and maintain it in a single concern. In the example of an XSLM-compliant licensing system shown below, the application would still have to make at least eight calls to the DRM system. However, these calls do not cut across multiple, but rather are specified in a single aspect class and are injected into the application at compile time. Also, calls don't need to be added by the programmer with knowledge of APIs – a user-friendly tool is provided that allows the specification of rights without any programming knowledge. The tool generates the rights file in XML format, validates it against the PARMA schema, convert it to aspect-oriented code.

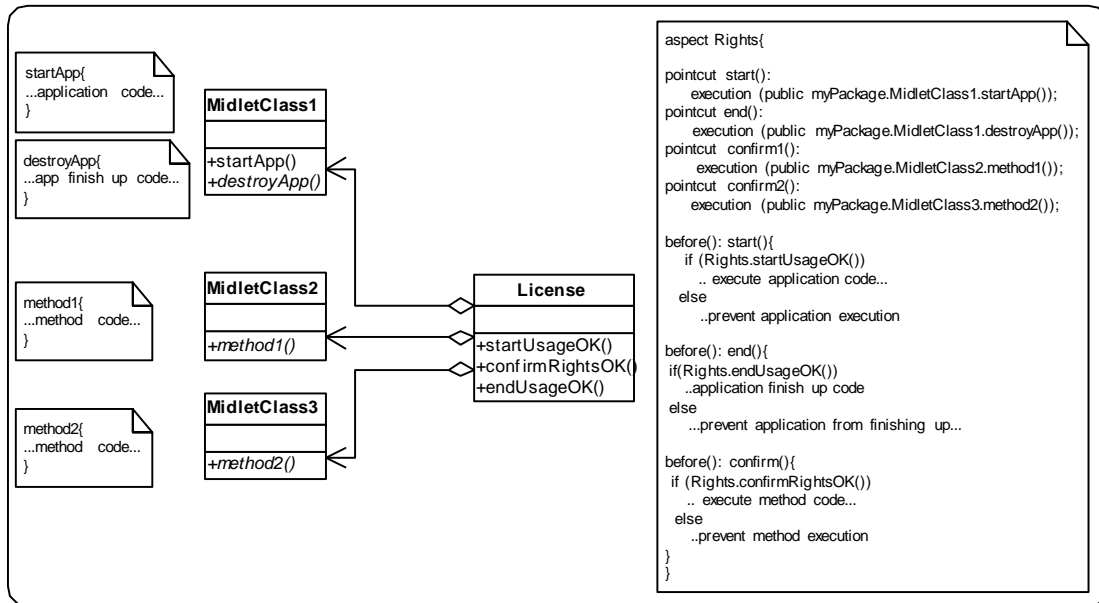


Figure 7 : PARMA AOP Approach

This approach introduces security risks associated with the rights enforcing code. A loosely-coupled approach of rights enforcement and the application decreases the effort of a potential hacker to rip off the licensing calls. We are investigating in techniques to render an application useless when it has been compromised with. PARMA also uses jar signing and verification techniques as well as code obfuscation to minimize the risk of a potential attack.

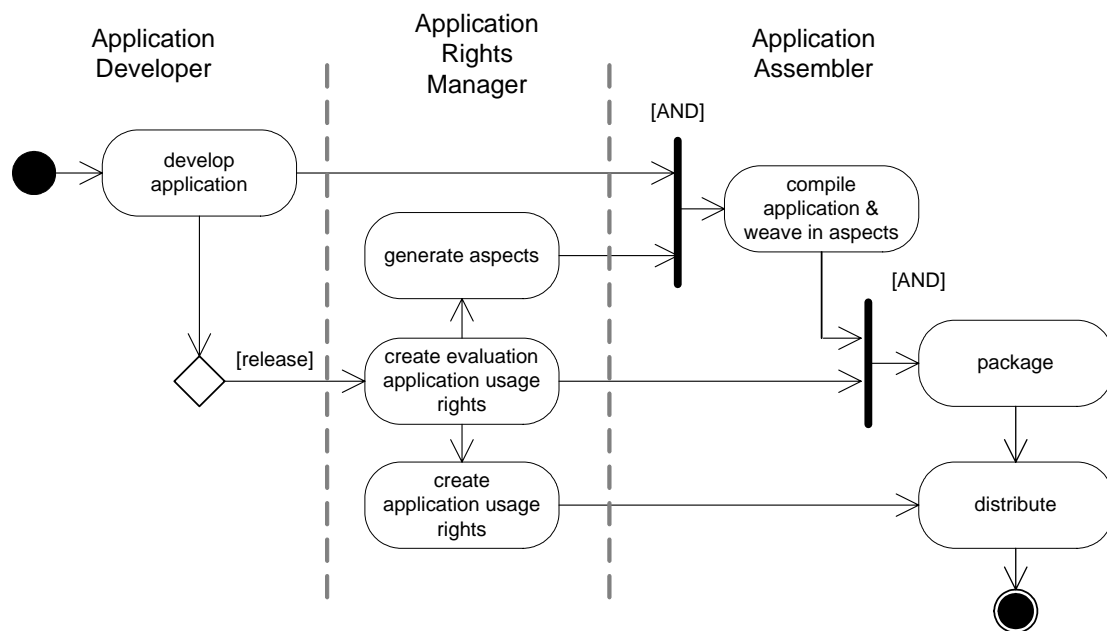


Figure 8 : PARMA Development Life-Cycle

The roles involved in the development life-cycle of an application and its associated usage rights are similar to those in the J2EE specification. On the one hand the developer is responsible for implementing the application without any rights management API calls. The application rights manager, on the other hand, is in charge of the declarative programming part. He maintains application usage rights and creates them for each application. Application rights can be categorized into two different rights models, the evaluation and any other 'higher' usage rights. The evaluation rights serve as a super set of the 'higher' usage rights and therefore are the basis of the aspect generation. It

presents all code-location where calls to the rights enforcement architecture should be inserted. All ‘higher’ application usage rights specialize rights based on the evaluation rights. Consequently, it is possible to distribute one rights object with a default evaluation rights agreement and upgrade at runtime of this application to another rights agreement without downloading a new version of this application with specific calls to the enforcement architecture. All ‘higher’ application usage rights are being made available on a rights locker server which can be co-located with the EDS or VDS.

The application assembler compiles the application with the generated aspects into a rights object, packages it with the evaluation usage rights agreement, and distributes it through a provisioning server.

4.3 Mapping of PARMA REL elements to aspect code

In traditional software licensing license validation is performed at different points in the application source code. Many of the supported rights models require the validation to be done only once, usually at the application start-up, while more fine-grained rights models specify validation to be performed in set time intervals, for real-time billing licenses, or at function or methods calls within the application. The latter type of model enables the use of feature-based rights and the release of upgradeable evaluation versions of software. Such two different licensing models define two different types of mapping from the PARMA rights file to join points in the aspect code.

For models that require the rights to be validated only at the beginning of the application, only one join point is specified in the aspect code, regardless of constraints in the rights definition. That point cut is a method `startApp()`, entry point to all MIDP applications. Rights validation is performed before this method is called and if validation fails application execution is not allowed to proceed.

```
pointcut checkLicense() :
    execution(public myPackage.MIDlet.startApp());

before() : checkLicense () {
    Context appContext = new Context();

    //make an API call to DRM rights enforcement engine
    boolean proceed = DRMEngine.validateLicense(appContext) ;
    if (proceed==true) {
        // do nothing, allow code in the startApp() to be executed and
        // start the application
    } else{
        // prevent the application from proceeding
    }
}
```

The aspect creates the application context object and stores all the available runtime information about the application in that object. This information includes join point information – the package, class, and method it belongs to, as well as types and values of the parameters. The aspect calls `DRMEngine.validateRights(appContext)` to validate the obtained runtime information against the rights. Based on the return value of this API call, the aspect either returns to the main application code and proceeds as normal, or exits the application with the error message informing users they are not authorized to execute the given application at the given time on the given device. Additionally, users can be provided with the option to purchase additional rights that will enable them to proceed with the execution, or instead of exiting the application, user can be allowed to run the demo version of the application.

The mapping of a PARMA rights definition to aspect code for feature-based rights model is more fine-grained and complex. The rights specifies the methods that require rights enforcement before their call. A sample right definition specifies that validation should be performed before the method `myMethod` that takes `int` as its only parameter, has `myReturnType` as a return type, and belongs to the class `myClassName` within `myPackage`. Pointcut can have values “before”, “after”, or “around” depending on whether rights check is needed before the execution, confirmation is required after the execution, or a replacement of a specified method is required.

```
<parma:pointcut>
  <parma:adviceType>before</parma:adviceType>
```

```

<parma:package>
  <parma:packageName>myPackage</parma:packageName>
  <parma:class>
    <parma:className>myClassName</parma:className>
    <parma:method>
      <parma:methodName>myMethod</parma:methodName>
      <parma:returnType>myReturnType</parma:returnType>
      <parma:argType>int</parma:argType>
    </parma:method>
  </parma:class>
</parma:package>
</parma:pointcut>

```

The REL-to-Aspects tool maps these XML elements into aspect joint point as follows.

```

pointcut validateLicense (int i) :
  execution(* myPackage.myClassName.myMethod(int))
  && args(i);

before (int i): validateLicense (i) {

  Context appContext = new Context();

  //make an API call to DRM enforcement engine
  boolean proceed = DRMEngine.validateLicense(appContext);
  if (proceed==true) {
    // do nothing, allow code in myPackage.myClassName.myMethod
    // (int) to be executed
  } else{
    // prevent the myPackage.myClassName.myMethod (int) from
    // proceeding
  }
}
}

```

As a result, the rights enforcement architecture can evaluate e.g. the level of a game the user is currently in and denies access to the next level according to the usage rights.

5 Rights Enforcement Architecture

In previous sections we introduced how we accomplish rights management with a well-established standard, ODRL, and our aspect-oriented approach to inject DRM enabling code into any JAVA implemented application. Our emphasis on this approach is its ease-of-use. In order to enable an application with DRM only an ODRL-specific rights file is needed.

The following sections deal with the design of our rights enforcement architecture, its components, and which aspects we consider to be important in a successful DRM architecture. We aim to implement a framework for resource constraint and occasional connected devices. PARMA targets the J2ME environment, especially the MIDP 2.0 profile. In contrast to desktop computers mobile devices are not permanently connected to the internet and hence the design of such a framework requires a flexible model to decouple any DRM infrastructure from network connections. Consequently, flexible rights models and DRM systems are necessary to encourage a scenario where a user is not able to connect to a network to comply or upgrade the DRM agreement. With ODRL and our extensions to it we defined the building block to express such a flexible agreement. This section gives an abstract view on the framework and how we partition the components being involved.

5.1 Pervasive DRM Architecture

A flexible license model, such as an audit-based pay-per-use agreement, requires interaction with a DRM server to upload audit information and initiate payment. Additionally, the user should have the choice to migrate to a different rights model. As a result, interoperability with the content provider is crucial and ideally should be standardized. The PARMA architecture consists of Vendor DRM Server (VDS) hosted by the application vendor (or authorized distributor), Enterprise DRM Server (EDS) hosted by an enterprise that purchases sets of rights for its users, and mobile clients running the rights object and connecting to EDS or VDS. The EDS component can be left out in the case of retail rights distribution where clients would communicate directly to VDS to download applications and obtain usage rights.

On the server side, VDS exposes WSDL interfaces for enterprise customers (EDS) to register for the application usage, download the application, request/return usage agreements, select/change/cancel agreement options, send usage data for audit and make payments. The basis for a payment architecture incorporated in PARMA is Web Services Framework for Mobile Payment Services developed by David McKitterick [17]. EDS exposes similar WSDL interfaces to communicate with mobile clients, allow them to register for and download the given application, obtain permissions on its usage and send back auditing data. This architecture uses web services for communication between clients and EDS/VDS and is an upgrade of a licensing architecture developed at Trinity College Dublin by Niall Clarke where clients were communicating with servers using SOAP [18].

The development of the server side architecture is a great deal influenced by numerous U.S. patents on license management systems and in order to avoid infringing these patents several initial design decisions had to be altered. The VDS server will also implement JSR-124 J2EE Client Provisioning specification [30] to enable clients to detect suitable applications, download them together with usage rights and initiate payment.

Communication between clients and EDS somewhat differs from communication between EDS and VDS due to limited resources of mobile clients. The PARMA DRM file is created on EDS, aspect code is generated based on the rights specified in this file and packaged together with the original application. Both aspects and original PARMA DRM file are delivered to the client. The payment framework supports payment for applications and rights via SMS, mobile operators and credit card but is extensible to new payment methods. In the case of enterprise licensing, payment is not done between clients and EDS/VDS, but usage data is collected on EDS from all clients and then payment is settled between EDS and VDS.

For a successful DRM architecture security considerations should be taken into account at initial design decisions. However, cryptographic security measures are not the focal part of this paper. We assume the client participates in a trusted environment, her juridical privacy is respected, and communication channels are sufficiently secure. Further we assume that mobile code offers some degrees of security in terms of protecting the rights of the copyrights holder with watermarking, and provide some level of tamper-resistance with obfuscation techniques [04, 05]. We assume secure storage is achieved with encrypting usage data, keeping the DRM agreement and secret keys on a SIM card to prevent compromising with sensitive data. A detailed description of managing the persistent state of DRM systems is given in [19].

One of the goals of this project was to design generic components to be able to adapt new features easily. The following section on the container architecture sheds some light on the infrastructure of our framework. Then services are introduced and finally application provisioning, related and future work are discussed.

5.2 Container Architecture

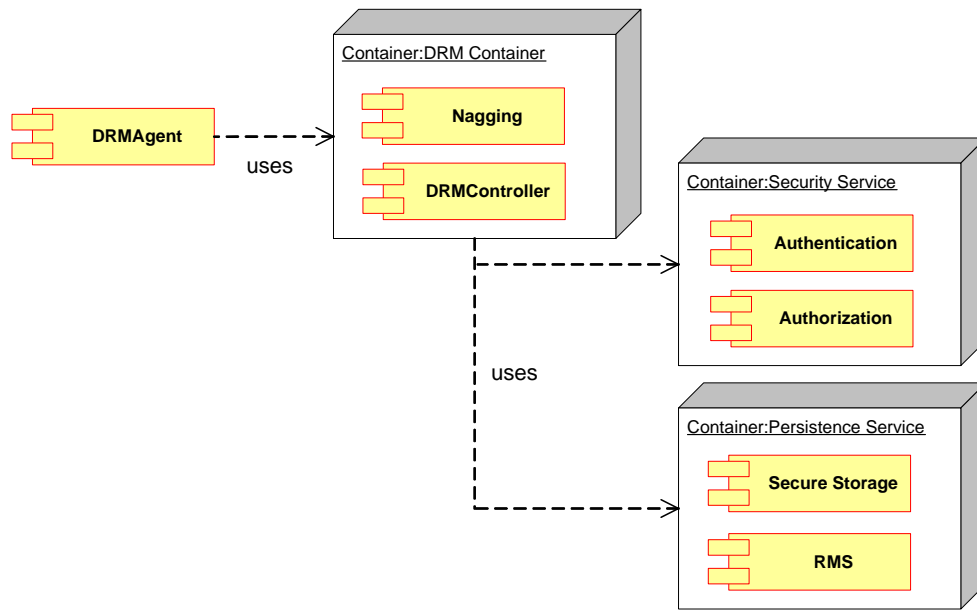


Figure 9 : DRM Container Architecture

The DRM framework provides a set of mechanisms whereby applications can be configured to support flexible rights management and billing options as well as a secure execution environment. The DRM framework is built around a light-weight interface-based Inversion of Control (type I) container architecture. The DRM support functionality is provided in isolated components (plug-ins). A component has to be registered with the container and then exposes its interface to other components which are interested in it. Components are not fixed to an implementation class, but rather to an interface. The advantage of this approach is the easy exchange of the implementation class without breaking dependent APIs. The fully qualified interface name of a component acts as a primary key. The container makes sure that only one class implementing this interface is instantiated. This can be seen as a pseudo singleton without implementing the singleton contract in a component explicitly. Consequently, components either need to be immutable or their methods synchronized. Instantiating a container is a two-phase process. First, all registered components are instantiated and in the second phase their dependencies are resolved. The implementation class can be either retrieved through a properties file which maps a plug-in to the implementation or it can be specified at registration time. All components implement a service provider interface (SPI) which declares callback methods for life-cycle, and dependency management.

The container is the central repository of instantiated components. Therefore memory management can be handled in one single place, rather than in multiple spaces. As well this approach offers a bit of security, because the container is the one and only instance which is responsible for creating components with a specialized factory class. This class and the repository can be exchanged, if special memory handling code is required. Common patterns in a memory constraint environment are object pooling and fixed allocation which both can be easily adapted in a new implementation [31].

The container can be arranged in a hierarchical structure whereby each child container is responsible for a specific task. In our architecture child-containers implement services which are only visible to its parent.

5.3 Services

The Services are controlled by a DRMController which is a component itself. The DRMController implements the delegation logic to each service and configures them when the container starts up. The following sequence diagram illustrates the interaction of the DRMController with the services provided by the child containers.

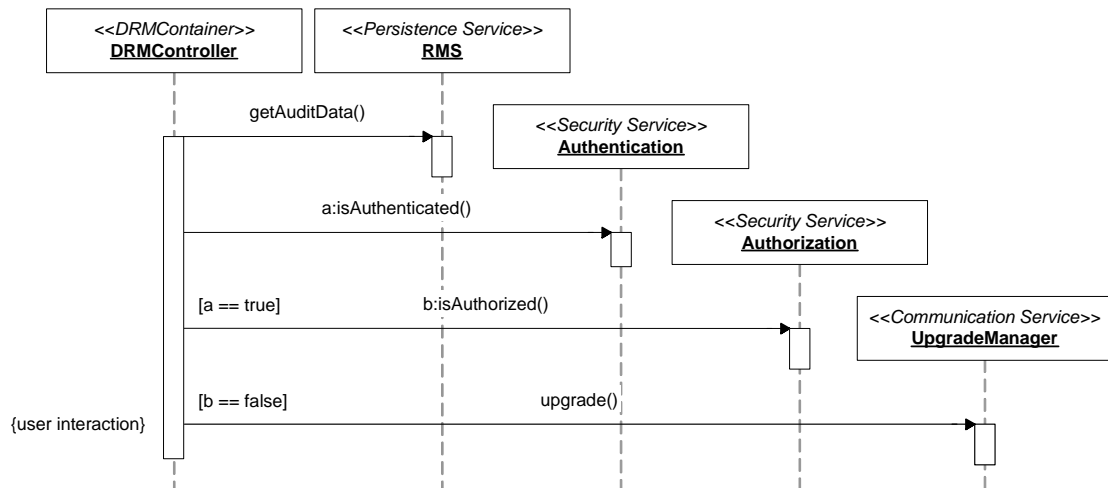


Figure 10 : DRMController Interaction

After the DRMContainer is initialized and a call to the DRMAgent is made, the controller first accesses the Persistence Service’s Record Management System (RMS) to retrieve the runtime audit data and updates them accordingly. These include usage data, subscription date, etc. This data is initially recorded to a persistent store when the application first starts up. Usage data is appended with a Message Authentication Code (MAC), encrypted and persisted into the RMS, whereas the usage rights according to the DRM agreement are kept on a secure tamper-resistant smart card. In addition the Persistence Service provides a one-way-counter for the usage statistics which is triggered at start up time of the application.

The user is then authenticated based on certificates delivered with the application. If authentication returns successfully, the authorization service is consulted to check against the DRM agreement. The authorization service implements a simplified role-based policy language. The roles participating in PARMA are shown in Figure 11:

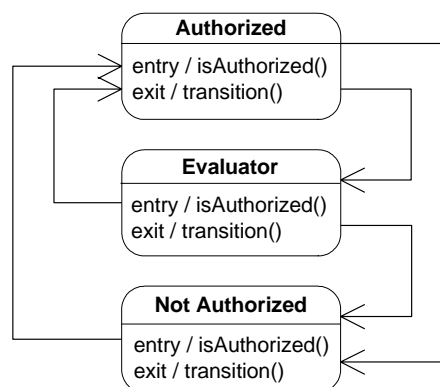


Figure 11 : DRM Roles

An authorized role represents a user with valid rights, whereas the evaluator represents a user who only is allowed to evaluate/preview an application. This approach helps in achieving a flexible rights model. Users should be able to overdraft rights agreements and comply at a later stage. The overdraft is expressed in ODRL. If the overdraft exceeds the tolerance level the user falls back into an evaluation mode. If the evaluation period is exceeded the user has to upgrade to a new rights agreement. This

process may involve payment of some sort. However, these roles are not fixed to our DRM system. If the ODRL agreement does not express such a flexible scenario, roles will be established according to the agreement.

If authorization fails the user has the option to upgrade the agreement via the communication service. The communication service abstracts the transport layer and applies security transparently, such as client side authentication, content encryption. On top of the transport and security layer are application-specific components, like the UpgradeManager.

5.4 Provisioning

OMA DRM 1.0 specified three methods for digital rights management: Forward-lock, separate and combined delivery of usage rights and the rights object.

A combined delivery of the rights object and the rights addresses the preview or evaluation scenario. We assume that evaluating applications based on restricted rights is a major aspect for customer and enterprise adoption of software. Separate delivery is a two stage process, where the rights object is delivered in an encrypted form to the client and the rights including the decryption key and maybe a certificate are delivered via WAP Push or alternatively via the Push registry which is defined in MIDP 2.0. However, the application management system (AMS) in Symbian OS does not yet support the installation of encrypted applications.

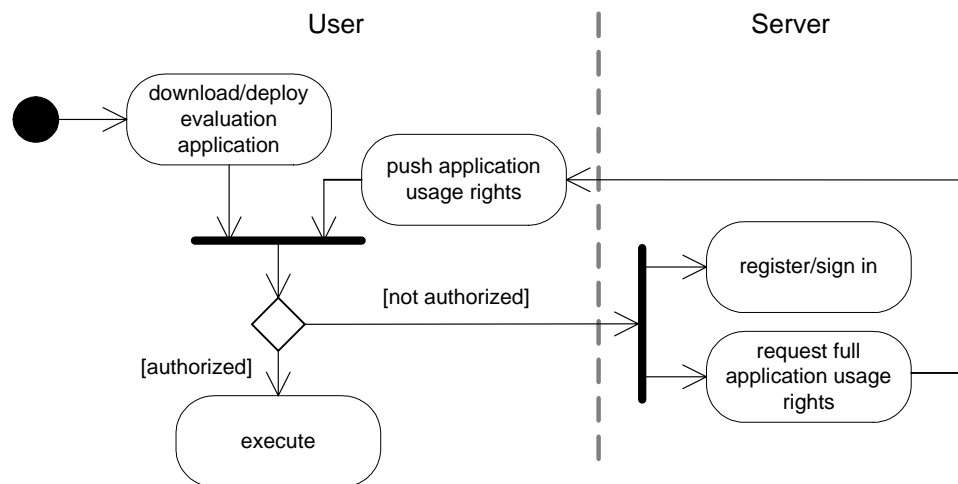


Figure 12 : Provisioning Cycle

PARMA combines both methods, separate and combined delivery. The user first downloads the application in combination with the evaluation usage rights from a content server, evaluates it and may upgrade later to a full version. Upgrading requires the negotiation of an agreement which is then delivered via the Push registry to the client. The MIDlet responsible for the Push request initializes the record store and stores the rights and the certificate on the smart card.

6 Related Work

In [22] a framework (FILIGRANE) is proposed which tackles relevant security issues in the mobile code commerce. The actors in a mobile code commerce scenario are identified which participate in the functional model of FILIGRANE. Although, this project does not address a DRM standard they implement practical security techniques to secure mobile code commerce including contract handling, provisioning, software protection, and usage control. A very interesting part of this paper is the analysis of the cost of piracy to break the protections and the measures they have taken to maximize the cost of piracy. Those measures involve encryption, obfuscation and watermarking of code and the use of smart cards to store sensitive data (e.g. the rules (rights) and the private key). However, encryption on a byte-code level is very easy to breach and so get access to the original byte-code. JAVA uses classloaders to

load classes and deliver them to the JVM via one well-defined final method `ClassLoader.defineClass(String, byte, int, int, ProtectionDomain)`. All classes must be delivered to the JVM via this aforementioned method. Although this method is final any class can be dumped in clear byte-code. To achieve this, the `ClassLoader` implementation has to be modified and repackaged (see `rt.jar` in the JAVA distribution) or specified in the `-Xbootclasspath` option [23].

This attack is more difficult to perform on J2ME devices. In contrast to J2SE classloading mechanisms can not be overridden or extended in application code. Further, it is more difficult to compromise with a J2ME installation on a mobile device, because the classloading capabilities are implemented in a native component, called Application Management System (AMS).

Nokia and SonyEricsson are in the process of implementing OMA REL based DRM systems on their mobile devices [1]. The version supported so far is OMA DRM 1.0 [13] that specifies three ways to protect copyrighted content – forward lock, combined delivery and separate delivery. Several Nokia phones so far support forward delivery while SonyEricsson's Z1010 is the first device that implements the full OMA DRM 1.0 specification supporting all three delivery methods. Both Nokia and SonyEricsson plan to support the full specification on all of their phones in forthcoming releases. Although efficient for content, this type of DRM system is not flexible enough for software applications as it does not support feature-based licensing. Also, the complete implementation of OMA DRM is done on the device, without DRM servers and back-end implementations that support client provisioning, payment and flexible license models.

ContentGuard[06] released their patented DRM system based on XrML eXtensible Rights Markup Language. XrML is not primarily aimed at wireless devices and currently does not have implementation on mobile phones.

7 Status of PARMA and Future Work

The development of our framework is in its early stages. Areas which require more attention are privacy, JavaCard access, a policy language in XML, and the interoperability with the server entities.

Privacy is a big issue in DRM systems and contributes significantly to the adoption of such a system, if done properly. The legal and technical privacy rights need to be explored in future versions of our framework. According to Brands' book [25] protecting one's privacy involves restricting the amount of data being submitted electronically to an absolute minimum because once submitted this information is not under control of the user anymore. It can be abused for e.g. marketing purposes. Additionally, different transactions of a user should not be linkable, unless the cost outweighs the benefits. In [24] security and privacy issues are assessed and applied to Digital Rights Management Systems. We are going to explore this area further and implement a workable solution with practical privacy measures.

The ability to store sensitive data in a tamper-resistant manner is crucial to every DRM system. In the very near future the Security and Trust Services API for J2ME (JSR-177) [32] will be available on J2ME mobile phones. This specification spans security services which rely on the interaction with a "Security Element" to provide secure storage, secure execution, and custom security features to allow e.g. payment. An implementation which facilitates the access to a smart card to store and retrieve sensitive data can be further abstracted with JSR-177. In combination with JSR-177, Security Assertion Markup Language (SAML) [26], XML digital signatures [27], XML encryption [28], Web Services secure XML protocol family (WS-Security [29]) we can provide a more generic and sophisticated communication layer to the content servers. An interesting task would be to define a Web Services-based Digital Rights Messaging protocol to exchange audit data and renew a DRM agreement.

Moreover, the generic design of our framework encourages the use of a dedicated XML-based rules description in order to configure the policy language used in our framework. The rules need to be transformed via XSL/T from the policies (rights) in the DRM agreement. It would then be possible to describe a DRM agreement in a different language, such as XrML without changing the implementation.

A big task is the interoperability with the content server. The entities involved in a complete DRM architecture need to be implemented either in a single server, which is not ideal, or as separate independent entities. Further, interoperability of the client with the back-end servers can be increased in a standard way with implementing the J2EE Client Provisioning specification [16].

8 Conclusions

We have introduced our extensions to the ODRL schema to represent a flexible software usage rights scenario. ODRL was extended to add more details about payment and user identification, to support constraints specific to mobile environment, and to differentiate between rights models for mobile software applications. Our experience with ODRL in general is that it is difficult to validate its schema because it is designed in an abstract way and allows many interpretations. Therefore, the semantics of application usage rights introduced in PARMA requires additional validation to make sure that all parties, the assets and rights for the assets conform to the PARMA model.

With our framework we accomplished a prototype for pervasive software licensing which enables and encourages the use of a flexible DRM agreement, such as feature-based policies. PARMA's aspect-oriented approach to usage rights code insertions also greatly reduces application developers' involvement in usage rights specifications for these flexible models and separates usage rights concerns from application and management concerns. Our framework is designed for extensibility and provides therefore a solid basis for our future work.

Bibliography

- [01] Johannes Rastas, "OMA DRM - Technical overview", Nokia Media & Music Digital Rights Management (DRM) Workshop, November 4th, 2003.
- [02] Macrovision, "Overcoming the Software Licensing Complexity Crisis", FlexNet White Paper, 17 Dec. 2003. <<http://www.macrovision.com/pdfs/PlatformWP.pdf>>.
- [03] XSLM Resource Center. Dec. 16 2003. <<http://www.xslm.org>>.
- [04] R. Iannella. Open Digital Rights Language (ODRL) Version 1.1, Aug. 8 2002, <<http://odrl.net>>.
- [05] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. "Aspect-oriented programming". In ECOOP'97, LNCS 1241, pages 220--242, 1997.
- [06] ContentGuard Inc. eXtensible rights Markup Language (XrML), Version 2.0. November 2001, <<http://www.xrml.org/>>.
- [07] Barry Fox, "Subversive code could kill off software piracy", New Scientist, 10 October 03, <<http://www.newscientist.com/news/news.jsp?id=ns99994248>>.
- [08] W3C (2003) Web Services Description Language (WSDL). Dec 17 2003. <http://www.w3.org/TR/wsdl>.
- [09] Bellamy, R., Brezin, J., Kellogg, W.A., Richards, J. and Swart, C. "Designing an E-Grocery Application for a Palm Computer: Usability and Interface Issues". IEEE Personal Communications, 8, 4, 60-64, August 2000.
- [10] Antony Adshead, "Cheaper field staff tracking services", Computer Weekly, May 13, 2003.
- [11] gamesindustry.biz, "Nokia N-Gage tops mobile game download chart", Dec 18 2003, <<http://www.theregister.co.uk/content/68/34241.html>>.
- [12] Matt Volpi, "Nokia DRM Tools", Nokia Media & Music Digital Rights Management (DRM) Workshop, November 4th, 2003.
- [13] Open Mobile Alliance, "OMA Digital Rights Management version 1.0", 19 Nov. 2002, <<http://www.openmobilealliance.org/tech/docs/index.htm#DRM>>.
- [14] IMEI codes, Dec. 17 2003, <<http://www.accesscomms.com.au/imei.htm>>
- [15] Gregor Kiczales et Al, "An Overview of AspectJ", In ECOOP, pages 327-- 353, 2001.
- [16] JCP, J2EE Client Provisioning Specification, JSR-124, <<http://www.jcp.org/aboutJava/communityprocess/first/jsr124/>>.
- [17] David McKitterick, A Web Services Framework for mobile payment services, M.Sc. thesis at the Department of Computer Science, Trinity College Dublin, 2003.
- [18] Niall Clarke, Distributed Software Licensing Framework based on SOAP, B.A. thesis at the Department of Computer Science, Trinity College Dublin, 2003.
- [19] William Shapiro and Radek Vringalek, How to manage persistent state in {DRM} systems, Digital Rights Management Workshop, 2001.
- [20] A. Monden and H. Iida and K. Matsumoto and Katsuro Inoue and Koji Torii, A practical method for watermarking JAVA programs, compsoc2000, 24th Computer Software and Applications Conference, 2000.
- [21] developers.sun.com, FAQ <<http://developers.sun.com/techtoc/mobility/midp/questions/obfuscate/>>.

- [22] G. Hachez, L. Den Hollander, M. Jalali, J.-J. Quisquater, and C. Vasserot, Towards a Practical Secure Framework for Mobile Code Commerce, In Pieprzyk et al. POS00, pages 164--178. 7, 2000.
- [23] Vladimir Roubtsov, Cracking JAVA byte-code encryption, <http://www.javaworld.com/javaqa/2003-05/01-qa-0509-jcrypt_p.html>, 2003.
- [24] Joan Feigenbaum, Michael Freedman, Tomas Sander, Adam Shostack, Digital Rights Management Workshop, p. 76-105, 2001.
- [25] Stefan A. Brands, Rethinking Public Key Infrastructures and Digital Certificates, MIT Press, Cambridge Massachusetts, 2000.
- [26] OASIS Security Services TC, SAML, 17 Dec. 2003
<<http://www.oasis-open.org/committees/security/>>
- [27] W3C, XML-Signature Syntax and Processing, 17 Dec. 2003
<<http://www.w3.org/TR/xmlsig-core/>>.
- [28] W3C, XML-Encryption Syntax and Processing, <<http://www.w3.org/TR/xmlenc-core/>>.
- [29] OASIS Security Services TC, WS-Security,
<http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss>.
- [30] JCP, J2EE Client Provisioning Specification, <<http://www.jcp.org/en/jsr/detail?id=124>>.
- [31] James Noble and Charles Weir, Small Memory Software – Patterns for systems with limited resources, Addison-Wesley, 2001.
- [32] JCP, Security and Trust Services API for J2ME, <<http://www.jcp.org/en/jsr/detail?id=177>>.

Interoperability between ODRL and MPEG-21 REL

Josep Polo, Jose Prados, Jaime Delgado
Universitat Pompeu Fabra (UPF), Departament de Tecnologia,
Pg. Circumval·lació 8, E-08003 Barcelona, Spain
{josep.polo, josep.prados, jaime.delgado}@upf.edu

Abstract

Two main Rights Expression Languages (RELs) exist to describe licenses governing the access to digital content: ODRL (Open Digital Rights Language) and MPEG-21 REL. Both RELs are powerful and complex enough. The use of different RELs could divide the network commerce in two separate factions. In this paper we propose a way for interoperability between them. They have many similarities that permit to translate expressions from one language into the other one. In the Distributed Multimedia Applications Group (DMAG) [12] we are developing utilities that permit to translate licenses between both RELs. Furthermore, the DMAG has developed a set of applications to generate and check licenses in both RELs.

This paper first describes the current situation of the RELs. Then the MPEG-21 REL and ODRL are introduced. Later, the interoperability between ODRL and MPEG-21 REL is exposed and the DMAG licenses generator and checker are described¹.

1 Rights Expression Languages

At present, network commerce of multimedia content is based on the trade of rights, but one of the limitations of Digital Rights Management (DRM) technology is due to the deficiency of means to express in an unambiguous, precise, machine-readable way the complex permissions on content. Furthermore it is not possible to create business relationships with distributors based on machine-readable licenses that can be automated to a significant degree. Consequently, today's business models have limited attraction to consumers and providers.

It is necessary to have a process by which the rights can be expressed in machine-readable licenses, guaranteed to be unambiguous and secure. A Rights Expression Language (REL) is the key to technical interoperability between proprietary DRM systems.

The basic component of a REL is the rights expression, which describes a permission granted to a user of protected content. Those rights expressions can be generated by any party authorised to grant permissions on content. In order for a rights expression language to be machine-readable, it must be based on a syntax that is recognised. Furthermore, there must be some measures built in, such as the ability to digitally sign rights expressions, so that their authenticity and tamper-resistance can be verified.

Several RELs have been proposed to describe licenses governing the terms and conditions of content access. In this field, ODRL [4, 5, 6] and MPEG-21 REL [16, 18] cover a prominent role. Both languages are powerful yet complex. This paper doesn't propose a way of analysing both languages to decide which of them is better, but it proposes to analyse the similarities and the interoperability of both languages.

¹ This work has been partly supported by the Spanish Ministry of Science and Technology (TIC2002-01336)

2 MPEG-21

At present, there is a standardisation committee, the Moving Picture Experts Group (MPEG), formally Working Group 11 of the ISO/IEC Joint Technical Committee, Sub-committee 29 [16], that covers most multimedia content subjects.

One of the standards produced by the MPEG is MPEG-21 [1]. Its aim is to offer interoperability in multimedia consumption and commerce. The standard is currently (after the March 2004 MPEG meeting) divided into 16 parts, most of them still under development. The number of parts may still increase.

Three of these parts are directly dealing with Digital Rights Management (DRM):

- **Part 4. Intellectual Property Management and Protection (IPMP):** provides the means to reliably manage and protect content across networks and devices.
- **Part 5. Rights Expression Language (REL):** specifies a machine-readable language that can declare rights and permissions using the terms as defined in the Rights Data Dictionary.
- **Part 6. Rights Data Dictionary (RDD):** specifies a dictionary of key terms required to describe users' rights.

This paper is focused in part 5: Rights Expression Language. This part explains the basic concepts of a machine interpretable language for expressing the rights and permissions of users that act on digital items, components, fragments, and containers.

2.1 MPEG-21 REL

The REL from MPEG-21 is based on the XrML proposal [10]. Using MPEG-21 REL it is possible to specify, for a digital resource (content, service, or software application), who is allowed to use that resource, the rights available to them and the terms, conditions or restrictions necessary to exercise those rights on the resource.

The core of MPEG-21 REL is the following four elements: principal, resource, right and condition (shown in Figure 1):

- **Principal:** identifies an entity such as the person, organisation, or device to whom rights are granted. Each principal identifies exactly one party. Typically, this information has an associated authentication mechanism by which the principal can prove its identity.
- **Right:** specifies the activity or action that a principal can be granted to exercise against some resource. Example rights include play, print, issue, obtain, etc.
- **Resource:** identifies an object which the principal can be granted a right. It can be a digital work, a service or a piece of information that can be owned by a principal. A Uniform Resource Identifier (URI) can be used to identify a resource. For example, a video file that a principal may play.
- **Condition:** specifies one or more conditions that must be met before the right can be exercised. For example, a principal may need to pay a fee to exercise a right, a limit to the number of times, a time interval within which a right can be exercised, etc.

The core data model is enhanced by a number of so-called “Extensions” which add both functionality and applicability.

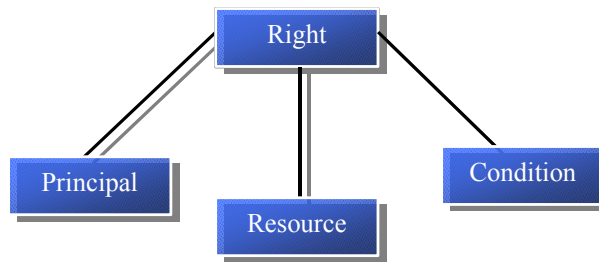


Figure 1. Core elements of MPEG-21 REL.

The MPEG-21 REL function is to express rights granted by some principals for specific resources and the conditions under which those rights apply. It does not provide any encryption functionality for content, though it does link to processes for ensuring the rights expressions themselves are tamper proof and capable of authentication.

The basic MPEG-21 REL element is the license. A license can contain one or more grants, the license issuer, that gives the grants that the license contains, and additional administrative information. Each grant must contain information to identify the four elements (principal, resource, right and condition) associated to it. Figure 2 shows a simple license structure.

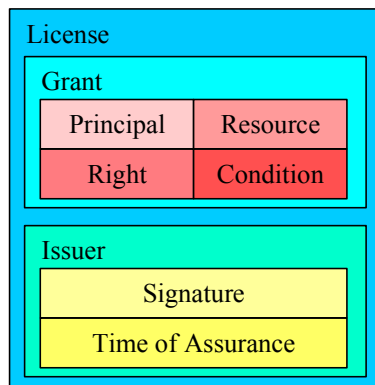


Figure 2. MPEG-21 REL license.

The license issuer who issues a license can digitally sign it, meaning that the issuer does really give the grants contained in it. Multiple issuers may sign a given license.

A grant is the part of an MPEG-21 REL license that conveys to an identified party the right to use a resource subject to certain conditions.

For example, consider an e-book named “Why Cats Sleep and We don’t” distributed to a consumer (Alice) that she can print 3 times. The MPEG-21 REL document has a sentence that says that Alice is granted with the right to print the book for 3 times. In this case, Alice is a *principal*, the book is a *resource*, print is a *right*, and “3 times” is a *condition*. In MPEG-21 REL the right-granting portion of this statement is called a *grant* and the entire statement is called a *license*. Figure 3 shows this example grant.

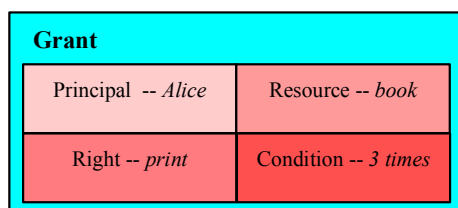


Figure 3. MPEG-21 REL grant example.

3 ODRL

The ODRL is a proposed language for the DRM community for the standardisation of expressing rights information over content. The ODRL is intended to provide flexible and interoperable mechanisms to support transparent and innovative use of digital resources in publishing, distributing and consuming of electronic publications, digital images, audio and movies, learning objects, computer software and other creations in digital form. This is an XML-based usage grammar.

ODRL is focused on the semantics of expressing rights languages and definitions of elements in the data dictionary. ODRL can be used within trusted or untrusted systems for both digital and physical assets (resources).

ODRL is based on an extensible model for rights expressions, which involves three core entities and their relationships. These are shown in Figure 4. They are:

- **Party** includes end users and Rights Holders. Parties can be humans, organisations, and defined roles. In the previous example, Alice is the party.
- **Right** includes permissions, which can then contain constraints, requirements, and conditions. Permissions are the actual usages or activities allowed over the assets (e.g. play, print, etc.) Constraints are limits to these permissions (e.g. print an e-book for a maximum of 3 times) Requirements are the obligations needed to exercise the permission. Conditions specify exceptions that, if they become true, expire the permissions and re-negotiation may be required. In the previous example, print is the right that includes the constrain of “3 times”.
- **Asset** includes any physical or digital content. They must be uniquely identified and may consist of many subparts and be in many different formats. Assets can also be non-tangible expressions of works and/or manifested in particular renditions. In the previous example, the book is the asset.

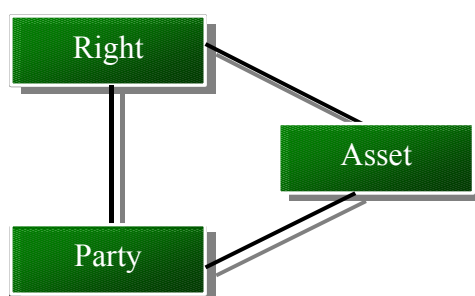


Figure 4. Core elements of ODRL.

4 Interoperability between ODRL and MPEG-21 REL

ODRL and MPEG-21 REL have many similarities: syntactically they are both based on XML, while structurally they both conform to the Stefik's axiomatic principles of rights modelling [2, 14].

A difference between ODRL and MPEG-21 REL is that ODRL seems more adapted to actual transactions in the commerce environment, whereas MPEG-21 REL has designs on broader cross-vertical applicability. ODRL's primitives map more directly onto the types of terms that are found in real-world commerce.

These RELs are widely used, so it is very important to permit interoperability between different systems that use these RELs. They have the same objective and they start from the same base.

They have different entities, but these try to represent the same information. After analysing both languages, we can conclude that there are four main entities in a license:

- **Subject:** actor who performs some operation. In ODRL, it is the party and in MPEG-21 REL it is the principal.
- **Right:** what a subject can do to an object. In ODRL it is the permission (right) and in MPEG-21 REL it is represented by the right.
- **Object:** content acted upon by a subject. In ODRL it is the asset and in MPEG-21 REL it is the resource.
- **Condition:** describes when a right can be performed. In ODRL it is the constraint and is included in the permission (right), and in MPEG-21 REL it is the condition.

As an illustration, we can consider the previous example: Alice has got a license to print an e-book 3 times.

Intuitively, the *subject* of this example is “Alice”, the *object* is “book”, the *right* is “print” and the *condition* is “3 times”.

Figure 5 shows the ODRL license for this example.

```
<?xml version="1.0" encoding="UTF-8" ?>
<o-ex:rights xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:o-dd="http://odrl.net/1.1/ODRL-DD"
  xsi:schemaLocation="http://odrl.net/1.1/ODRL-EX ../schemas/ODRL-EX-11.xsd
  http://odrl.net/1.1/ODRL-DD ../schemas/ODRL-DD-11.xsd">
  <o-ex:asset>
    <o-ex:context>
      <o-dd:uid>urn:ebook.world/999999/ebook/rossi-000001</o-dd:uid>
      <o-dd:name>Why Cats Sleep and We don't</o-dd:name>
    </o-ex:context>
  </o-ex:asset>
  <o-ex:permission>
    <o-dd:print>
      <o-ex:constraint>
        <o-dd:count>3</o-dd:count>
      </o-ex:constraint>
    </o-dd:print>
  </o-ex:permission>

  <o-ex:party>
    <o-ex:context>
      <o-dd:name>Alice</o-dd:name>
    </o-ex:context>
  </o-ex:party>
</o-ex:rights>
```

Figure 5. ODRL example license.

Figure 6 shows the equivalent MPEG-21 REL license.

```

<?xml version="1.0" encoding="UTF-8" ?>
<r:license xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
  xmlns:mx="urn:mpeg:mpeg21:2003:01-REL-MX-NS"
  xmlns:r="urn:mpeg:mpeg21:2003:01-REL-R-NS"
  xmlns:sx="urn:mpeg:mpeg21:2003:01-REL-SX-NS"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:mpeg:mpeg21:2003:01-REL-R-NS ../schemas/rel-r.xsd
    urn:mpeg:mpeg21:2003:01-REL-SX-NS ../schemas/rel-sx.xsd
    urn:mpeg:mpeg21:2003:01-REL-MX-NS ../schemas/rel-mx.xsd">
  <r:grant>
    <r:keyHolder>
      <r:info>
        <dsig:KeyName>Alice</dsig:KeyName>
      </r:info>
    </r:keyHolder>
    <mx:print />
    <r:digitalResource>
      <r:nonSecureIndirect URI="urn:ebook.world/999999/ebook/rossi-000001" />
    </r:digitalResource>
    <r:allConditions>
      <sx:exerciseLimit>
        <sx:count>3</sx:count>
      </sx:exerciseLimit>
    </r:allConditions>
  </r:grant>
</r:license>

```

Figure 6. MPEG-21 REL example license.

Table 1 shows the four main entities and their relationship with ODRL and MPEG-21 REL.

Table 1. The four main entities in the licenses.

ENTITY	ODRL	MPEG-21 REL
Subject	o-ex:party	r:keyHolder
Object	o-ex:asset	r:digitalResource
Right	o-ex:permission	r:grant
Condition	o-ex:constraint	r :allConditions

If we consider the similarities that can be seen in the previous example, as well as in the previous part of the paper, it can be concluded that the interoperability between both languages is possible. To transform an ODRL license into an MPEG-21 REL license, or vice versa, it is equivalent to transform a XML document to another XML document, where the information to represent is the same one, but with a different XML structure.

These are preliminary results. As it can be seen from the example, these licenses in ODRL and MPEG-21 REL are very simple. It is supposed that more complicated licensed will be more difficult to transform between ODRL and MPEG-21 REL, but it is also to be expected that the transformation will be done properly due to the languages similarity.

In order to obtain this transformation, XSL (Extensible Stylesheet Language) can be used. The XSL is one of the most important intricate specifications in the XML family. Using XSLT (XSL Transformation) [3] is not the only way to transform XML documents. A general purpose programming language like C, C++, or Java can also be used. XSLT has the advantage of being more lightweight than those languages and it is oriented to XML interaction. It is adequate for transformation and is well-equipped as a language to perform this main design goal. It allows to write programs that are much smaller than with a general purpose programming language. XSL can be broken in two parts: the said XSLT and XSL-FO (XSL Formatting Objects). XSLT applies transformation rules to the document source and, by changing the tree structure, produces a new document, such as another XML document. It can also amalgamate several documents into one, or even produce several documents starting from the same XML document.

In order to implement this interoperability, we have developed two utilities: one to transform an ODRL License into a MPEG-21 REL License and the other one to transform into the reverse direction. These utilities have been developed using XSL pages. The structures of both tools are very similar, but many differences exist at rules implementation level, given the syntactic and semantic differences of both languages. Figure 7 shows the XSLT document that allows to transform the previous ODRL license (Figure 5) to a MPEG-21 REL license (Figure 6). The other XSLT document (MPEG-21 REL to ODRL) is quite similar, so it is not shown.

```

<?xml version="1.0" encoding="UTF-8" ?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:o-
ex="http://odrl.net/1.1/ODRL-EX" xmlns:o-dd="http://odrl.net/1.1/ODRL-DD"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
  xmlns:mx="urn:mpeg:mpeg21:2003:01-REL-MX-NS"
  xmlns:r="urn:mpeg:mpeg21:2003:01-REL-R-NS"
  xmlns:sx="urn:mpeg:mpeg21:2003:01-REL-SX-NS"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:mpeg:mpeg21:2003:01-REL-R-NS ../schemas/rel-r.xsd urn:mpeg:mpeg21:2003:01-
REL-SX-NS ../schemas/rel-sx.xsd urn:mpeg:mpeg21:2003:01-REL-MX-NS ../schemas/rel-mx.xsd">

  <!-- Output file is a XML file -->
  <xsl:output method="xml"/>

  <!-- Substitute root node from ODRL file to this MPEG-21 REL template -->

  <xsl:template match="/">
    <r:license xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
      xmlns:mx="urn:mpeg:mpeg21:2003:01-REL-MX-NS"
      xmlns:r="urn:mpeg:mpeg21:2003:01-REL-R-NS"
      xmlns:sx="urn:mpeg:mpeg21:2003:01-REL-SX-NS"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:mpeg:mpeg21:2003:01-REL-R-NS ../schemas/rel-r.xsd
urn:mpeg:mpeg21:2003:01-REL-SX-NS ../schemas/rel-sx.xsd urn:mpeg:mpeg21:2003:01-REL-MX-NS
../schemas/rel-mx.xsd">
      <r:grant>
        <!-- Inside the grant, apply the rest of templates or replacements -->
        <xsl:apply-templates/>
      </r:grant>
    </r:license>
  </xsl:template>

  <!-- ODRL file "o-ex:party" node replacement for this MPEG-21 REL template -->
  <xsl:template match="o-ex:party">
    <r:keyHolder>
      <r:info>
        <dsig:KeyName><xsl:value-of select="o-ex:context/o-dd:name"/></dsig:KeyName>
      </r:info>
    </r:keyHolder>
  </xsl:template>

  <!-- ODRL file "o-ex:asset" node replacement for this MPEG-21 REL template -->
  <xsl:template match="o-ex:asset">
    <r:digitalResource>
      <xsl:element name="r:nonSecureIndirect">
        <xsl:attribute name="URI"><xsl:value-of select="o-ex:context/o-dd:uid"/></xsl:attribute>
      </xsl:element>
    </r:digitalResource>
  </xsl:template>

  <!-- ODRL file "o-ex:permission" node replacement for this MPEG-21 REL template -->
  <xsl:template match="o-ex:permission">
    <r:allConditions>
      <sx:exerciseLimit>
        <sx:count><xsl:value-of select="o-dd:print/o-ex:constraint/o-dd:count"/></sx:count>
      </sx:exerciseLimit>
    </r:allConditions>
  </xsl:template>

</xsl:stylesheet>

```

Figure 7. XSLT document that transforms the previous ODRL license in Figure 5, into the MPEG-21 REL license in Figure 6.

XSLT documents form a special class of XML documents. As all XML documents, it has one document element. This is the `xsl:stylesheet` element.

To specify the output method, the document has an `xsl:output` element. It is `xml` in this case.

The `xsl:template` element is the most important element in XSLT. It is the basis for matching patterns to perform transformation. The `match` attribute is used to match a pattern. The document has an `xsl:template match="/"` element. This element matches the root of a document. The transformation process always starts from an element that matches the root. This element allow us to substitute root node from the ODRL document to the MPEG-21 REL document.

The rest of nodes are inside the root node. This is the way to declare and call subroutines. Using an `xsl:apply-templates` element, we apply the rest of templates or replacements inside this root node to the other nodes.

The document has three more `xsl:template` elements: to transform `o-ex:party` to `r:keyHolder` (from ODRL to MPEG-21 REL), to transform `o-ex:asset` to `r:digitalResource` and to transform `o-ex:permission` to `r:allConditions`.

The `xsl:element` is used to construct an element in the result tree. The mandatory attribute is the `name` attribute, which specifies the name of the element to be generated.

The `xsl:attribute` element is used inside the template that match `o-ex:asset`, and it is used to add an attribute in the result tree.

The `xsl:value-of` element returns the string value of the expression given in the `select` attribute.

5 REL Tools

We have also developed a set of applications that permit to generate licenses in the REL languages: ODRL and MPEG-21 REL [7, 8, 9, 13]. The internal structure of the applications has similarities and differences between the two language versions, but from the point of view of the formal structure they are equivalents. For this reason, only the ODRL applications are described below. The MPEG-21 REL applications have the same formal description.

5.1 DMAG ODRL License Creator

The DMAG ODRL License Creator (DOLC) is a software implementation that creates ODRL Licenses. This software has been developed in Java. It can run on MS-Windows and Linux platforms. At present, the DOLC can only create basic licenses. The license types are offer and agreement. It is expected that, shortly, the software will be able to generate more complex licenses.

This software can create XML documents representing valid ODRL Licenses. Their implementation is based on the Document Object Model (DOM) API for HTML and XML documents. The DOM API provides a structural representation of the document, and it defines the way that a structure is to be accessed from a script. Essentially, it connects web pages or XML documents to programming languages.

From the point of view of a user, the DOLC has been implemented as a web application. It is composed by a web page containing an HTML form, a servlet for processing the information

introduced in the form and to generate the ODRL License, and finally, a web page containing the ODRL License created.

In order to syntactically validate the license against their schemas, another application has been developed, the DMAG Schema Checker (DSC), which is described in 5.2. So, the DSC software has been used to verify the validity of the ODRL License created by the DOLC.

The information needed to generate the license is introduced by means of the HTML form. This module allows the introduction of: asset, rights, requirements, constraints, parties, etc.

This information is sent to the servlet, which is installed in a servlet engine. Tomcat [11] is the servlet engine used. The License Creator servlet, extracts the information introduced in the web page form by the user that is creating the license, checking that all the needed fields have been filled. If all necessary information has been introduced, the XML file containing the license is created.

The license is conformant with its schemes using the DSC. Finally, the result is shown to the user, who can see the generated license.

Next steps to be taken are the addition of more complex permissions and/or constraints and the study of more complex licenses. The tools used to implement the DOLC application allow these additions in an easy way.

Figure 8 shows the structure of the DOLC developed.

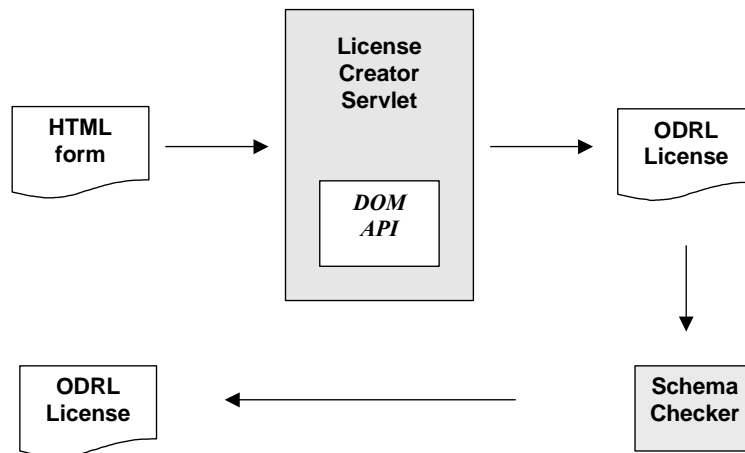


Figure 8. DMAG ODRL License Creator (DOCL) and its relation with the DMAG Schema Checker (DSC)

5.2 DMAG Schema Checker

As it was explained above, the DMAG Schema Checker (DSC) is an application that validates syntactically ODRL and other types of licenses such as MPEG-21 REL Licenses, against the XML schemas used by the licenses. This software has been developed in Java. It can run on MS-Windows and Linux platforms.

The parser used in the implementation is the Xerces parser [19]. The software validates syntactically a document with an ODRL license specified by the user.

The output of the DSC is a message reporting if the license is syntactically valid or not, according to the XML Schemas specified within the license. If the license is not valid, the DSC will inform about the reasons why.

Figure 9 shows the structure of the DSC developed.

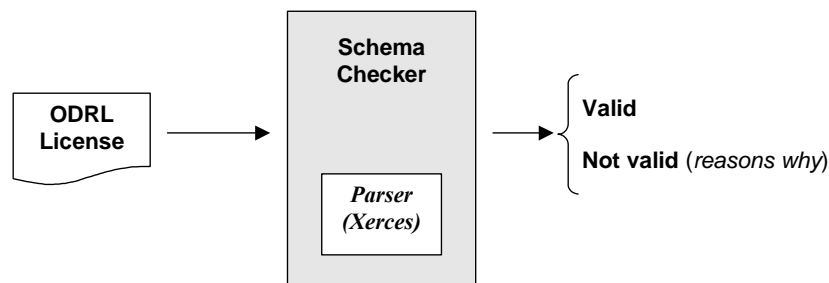


Figure 9. DMAG Schema Checker (DSC)

6 Conclusions and future work

A set of applications has been presented to permit interoperability between ODRL and MPEG-21 REL licenses. In the DMAG, we have developed utilities that permit to translate ODRL licenses into MPEG-21 REL licenses and in the other way around. The current licenses are simple, but we are working to expand the interoperability to more complex licenses.

Furthermore, a set of applications has been described to generate and check licenses in ODRL and MPEG-21 REL. At present, they can generate non-complex licenses, but we are also working to extend the capabilities of the applications.

The objective of our future work is to expand the scope of this set of applications (generators and converters) to permit that every system could work in ODRL or MPEG-21 REL without distinction, transparently to the user, generating the appropriate license, and, furthermore, when receiving a license, it would not matter the REL license format, because the system could work without distinction.

References

1. Burnett et al. *MPEG-21 Goals and Achievements*. IEEE Computer Society, October-November 2003
2. C.N. Chong et al. *Comparing Logic-based and XML-based Rights Expression Languages*. Confederated International Workshops: On The Move to Meaningful Internet Systems. Catania, Sicily, Italy. Published by Springer-Verlag. November, 2003.
3. K.Y. Fung. *XSLT Working with XML and HTML*. Addison-Wesley.
4. S. Guth et al. *Experiences with the Enforcement of Access Rights Extracted from ODRL-based Digital Contracts*. ACM Workshop on Digital Rights Management, Washington D.C./USA, October 2003.
5. S. Guth, E. Koeppen. *Electronic Rights Enforcement for Learning Media*. IEEE International Conference on Advanced Learning Technologies, Kazan/Russland, September, 2002.
6. R. Iannella. *Open Digital Rights Language (ODRL) Version 1.1*. <http://odrl.net>. August 2002.
7. S. Llorente, J. Delgado, E. Rodríguez. Enhanced versions of DMAG REL Reference Software. ISO/IEC JTC1/SC29/WG11 - MPEG2003/M10286. December 2003. <http://dmag.upf.edu/DMAGRELTtools/m10286.pdf>.
8. S. Llorente, J. Delgado, E. Rodríguez. *DMAG REL License Creator*. ISO/IEC JTC1/SC29/WG11 - MPEG2003/M10040. October 2003. <http://dmag.upf.edu/DMAGRELTtools/m10040.pdf>.
9. E. Rodríguez, J. Delgado, S. Llorente. *DMAG REL Schema Checker*. ISO/IEC JTC1/SC29/WG11 - MPEG2003/M10039. October 2003. <http://dmag.upf.edu/DMAGRELTtools/m10039.pdf>.
10. X. Wang et al. *XrML – eXtensible rights Markup Language*. ACM Workshop on XML Security, Nov. 22, 2002, Fairfax VA, USA.
11. Apache Tomcat. <http://jakarta.apache.org/tomcat/index.html>.
12. Distributed Multimedia Applications Group (DMAG), Technology Dept., Universitat Pompeu Fabra. <http://dmag.upf.edu>
13. DMAG MPEG REL reference software. <http://dmag.upf.edu/DMAGRELTtools/Index.htm>
14. eXtensible Rights Markup Language (XrML), Version 2.0. DRM watch. GiantSteps, Media Technology Strategies. <http://www.giantstepsmts.com/DRM%20Watch/xrml20.htm>
15. Information Technology - Multimedia Framework - Part 5: Rights Expression Language. FDIS 21000-5:2003(E). ISO/IEC JTC1/SC29/WG11/N5839.

16. Moving Picture Expert Group. ISO/IEC JTC/SC29 WG11.
<http://www.chiariglione.org/mpeg/index.htm>
17. Moving Picture Expert Group. ISO/IEC JTC/SC29 WG11. Programme of Work.
<http://www.itscj.ipsj.or.jp/sc29/29w42911.htm#MPEG-21>.
18. Rightscom Ltd. *The MPEG-21 Rights Expression Language. A white paper.*
http://www.rightscom.com/files/MPEG21_RELwhite_paper.pdf
19. Xerces Parser. <http://xml.apache.org/xerces-j/>

REAP

A System for Rights Management in Digital Libraries

Øyvind Vestavik
Norwegian University of Technology and Science
Oyvind.Vestavik@idi.ntnu.no

Abstract

This paper presents REAP, a system for rights management in digital libraries. REAP is aimed at demonstrating that intellectual property can be published in the Internet by digital libraries in accordance with copyright laws. The article proposes an architecture/paradigm for managing rights when publishing information through Digital Libraries involving digital rights management. Based on this architecture a working prototype called REAP has been implemented and is documented and discussed in this paper.

1. Introduction

The requirements for a Digital Rights Management system to be used in a digital library are a combination of the needs of the library's users and authors. Authors and rights holders want control over their work and possibly economic compensation for their effort. Information consumers want free access to as much information as possible without any administrative interaction like registering personal information like personal names, email address etc. The library tradition seems now to be challenged by the rise of the Internet as an information medium. Because of the ease of access to information in the Internet, users are beginning to vote out the traditional libraries when searching for information. For libraries to be able to continue their role as a society's source of quality information and a society's memory in a networked environment, libraries have to open up to the Internet and use it progressively to let users access the information and knowledge present in their collections and holdings. This poses radical challenges to the library tradition and the new actors in the library world. Digital libraries must, as far as possible, satisfy the needs of both its users and authors to survive. On the one hand, a too restricted access to information will not be satisfactory for information consumers as they will not be able to get the information they need. In these cases consumers will often turn to other sources of information. On the other hand, authors will not be willing to publish their information unless the digital library is able to handle the rights over the material in the digital library properly. It is therefore in the interest of a digital library to implement systems that preserves the needs of both groups. This means that the digital library must be able to trade with material or rights to material on behalf of its rights holders, providing its users with the material they need in a manner consistent with copyright and without violating the rights of users and authors. Well designed and adequate DRM Systems can hopefully balance the needs of the different patrons in a digital library. As a first attempt to create a DRM system for use in a digital library setting we have created a system called REAP, a Rights Enforcing Access Protocol. This system consists of a rights language to express rights over material in the digital library and a server side software that should be able to control that access to digital material published in the digital library is granted or denied based on rights description for the material expressed in the rights expression language. The rights language is based on ODRL [ODRL] and the software is inspired by a reference architecture for rights management systems proposed by [Rosenblatt, Trippe, Mooney 2002]

The rest of the paper is organized as follows: In part 2 the basis of REAP in terms of entities and concepts is discussed. Then, in part 3, a suggestion of how REAP fits into the larger setting of a digital library is given. Next, in part 4, the REAP software and what it does is presented before in part 5, the motivations and choices for a rights expression language for REAP is explained. Finally, in part 6, the properties of REAP is discussed.

2. The Basis of REAP

REAP [Vestavik2002] is an access protocol and is aimed at controlling users access to information resources on a server. It is not intended to support end-to-end chain services. As such, REAP can be seen as a part of the services a repository of a digital library offers its users. The architecture of REAP is based on a defined set of logical entities as shown in figure 1 below.

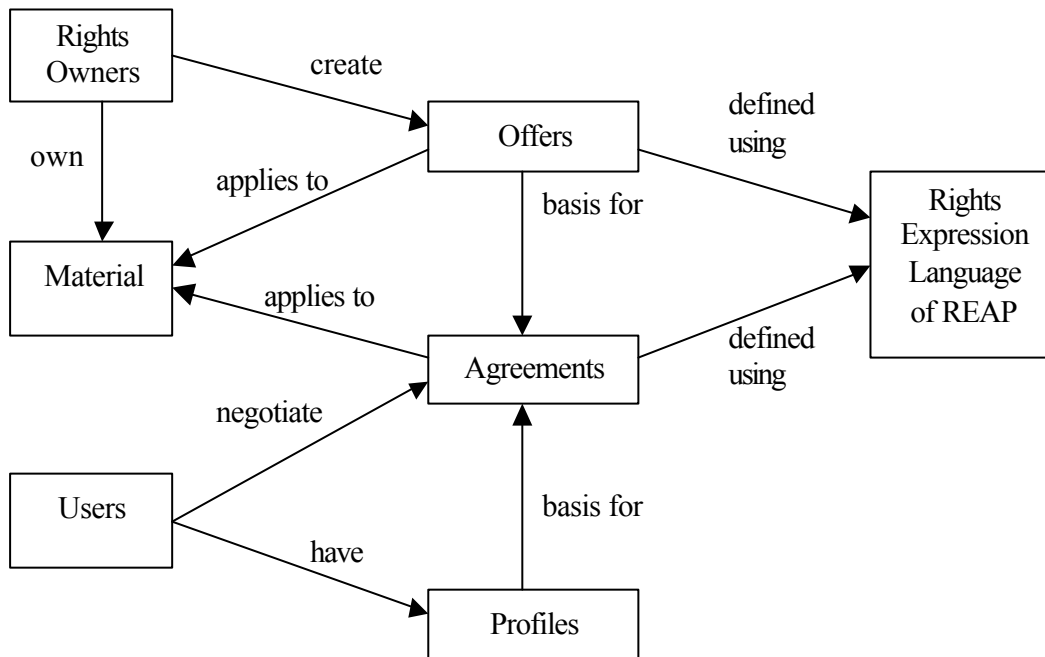


Figure 1: Logical Entities in REAP

Rights owners create rights offers for their material giving the rights that can be granted to users, the requirements to be fulfilled by users in order to be granted those rights and the constraint limiting the extent of the rights. *Note that conditions expiring rights is not supported by REAP.* Users have to register with the system to be able to create an agreement giving them access to rights over the material in the library and during registration a profile for the user is created. Both Offers and Agreements are expressed in an application specific rights language based on ODRL.

3. REAP and ADEPT

The proposed model/prototype is concerned with rights management in connection with retrieval of material. All other functions of a digital library is the responsibility of components and systems that are outside the scope of the prototype. For practical reasons all functionality normally associated with digital libraries is assumed to be carried out in Alexandria Digital Library (ADL) / Alexandria Digital Earth Prototype (ADEPT), a distributed digital library for georeferenced information [Alexandria].

Using an ADEPT client, users can search for information resources as shown in **figure 2**. The client sends a search/query to the middleware component of ADL/ADEPT which distributes the search to distributed collections located around the world. These collections have metadata collections containing metadata formatted for ADL, so-called ADEPT views or metadata for the ADL bucket framework on which the query is executed. The result of the query on the local metadata catalog is returned to the middleware which

returns it to the calling client at the user's location. Based on the responses from all collections that were queried an ADEPT access report presents the overall search results to the user.

The access report contains information about where to retrieve the information resource. This information is given as an URL. For REAP to be able to enforce rights policies in the retrieval process, the given URL must be formed as an http get request to the REAP software giving the resource to be retrieved as a parameter to the request. This means that the URL for the resource must be encoded in the metadata presented to the search engine of ADEPT. In REAP the identifier for a material is given by a combination of a collection id identifying the collection the resource is located in and an item id identifying the material within the given collection. In practice this information could have been given as a DOI, PURL or other persistent identifier as long as the resolved URL is of the form given below:

http://fenris.idi.ntnu.no:8080/REAP/get_Document?CollectionID=4?ItemId=2

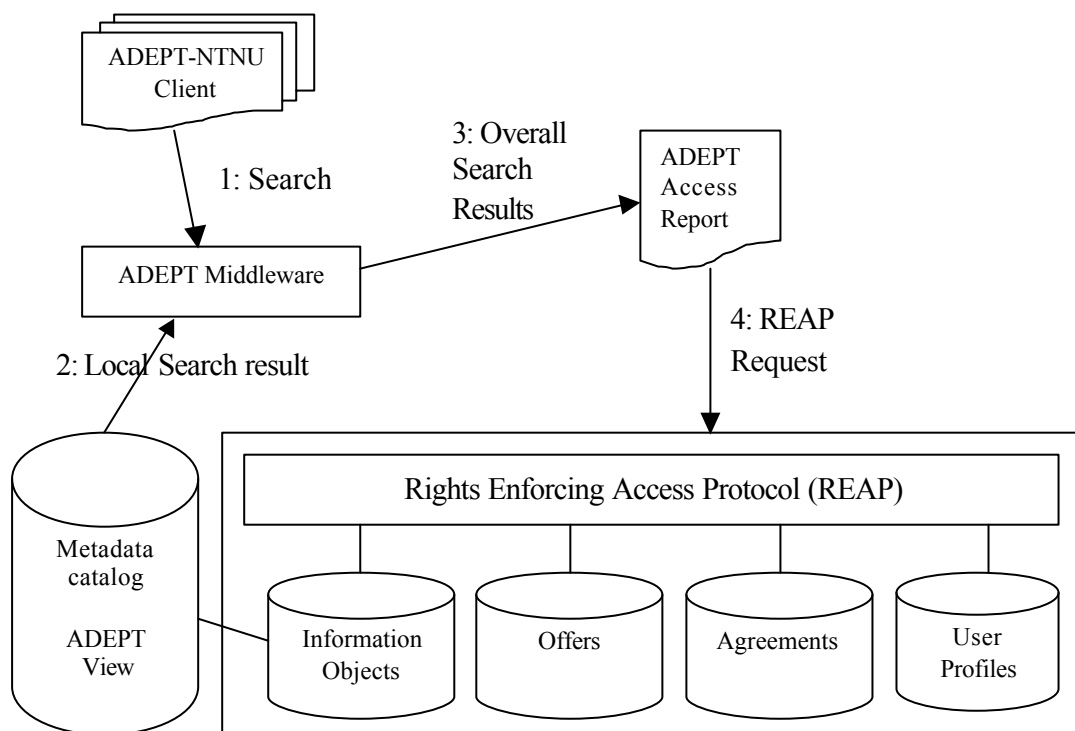


Figure 2: REAP in an ADEPT context

Although REAP is designed to be used with ADEPT, the intention is that the system is to be as autonomous as possible. This means that the system can be used with any system as long as the identifier for a material points to the REAP system giving the local identifier for the material as parameters to the http request. Also, REAP is not part of the ADEPT system itself, and collections under ADEPT do not have to implement or use REAP or any other rights enforcing software.

4. The REAP software

The REAP software is realized as a set of java servlets running on a Jakarta Tomcat Servlet Engine and uses an Apache Xindice native database to store rights description and other information.

When a user request access to information protected by REAP for the first time, an agreement between REAP and the user is set up regulating how the user can use the given resource. This agreement is based on the initial rights offer given for the material by the owner of the material, the information known about the user (recorded during registration), the requirements the user is willing to fulfill and the selection of rights from the initial offer that the user is interested in obtaining. Both offer and agreement is based on an application specific Rights Expressing Language described later.

First REAP checks if the given URL is well formed, that is if it contains a collection id and an item id. Then the system checks if the user is logged on. The system uses session variables, so being logged in means that there is a session registered on the server for the IP and process number of the browser/machine the user is using. If the user is not logged in the user is presented with a login page with a link to a registration page. Once the user is registered / logged on to the system checks whether the resource identified by the collection id / item id exists. If the user already has an agreement for accessing rights to the material the user can start executing rights over the material. If not, the system will interact with the user to set up such an agreement by letting the user select one or more rights from the initial rights offer for the material that he/she wants access to. Once the agreement is created, the user can start exercising the rights transferred in the agreement.

There is no rendering application created for REAP. The result of requesting access to execute rights on material as specified in the user's agreement for the material is a ticket granting or denying access to exercise the requested right. The system will keep track of the rights the user accesses and make sure he/she does not overstep the bounds of the agreement. The execution paths involved in getting access to material is given in **figure 3** below.

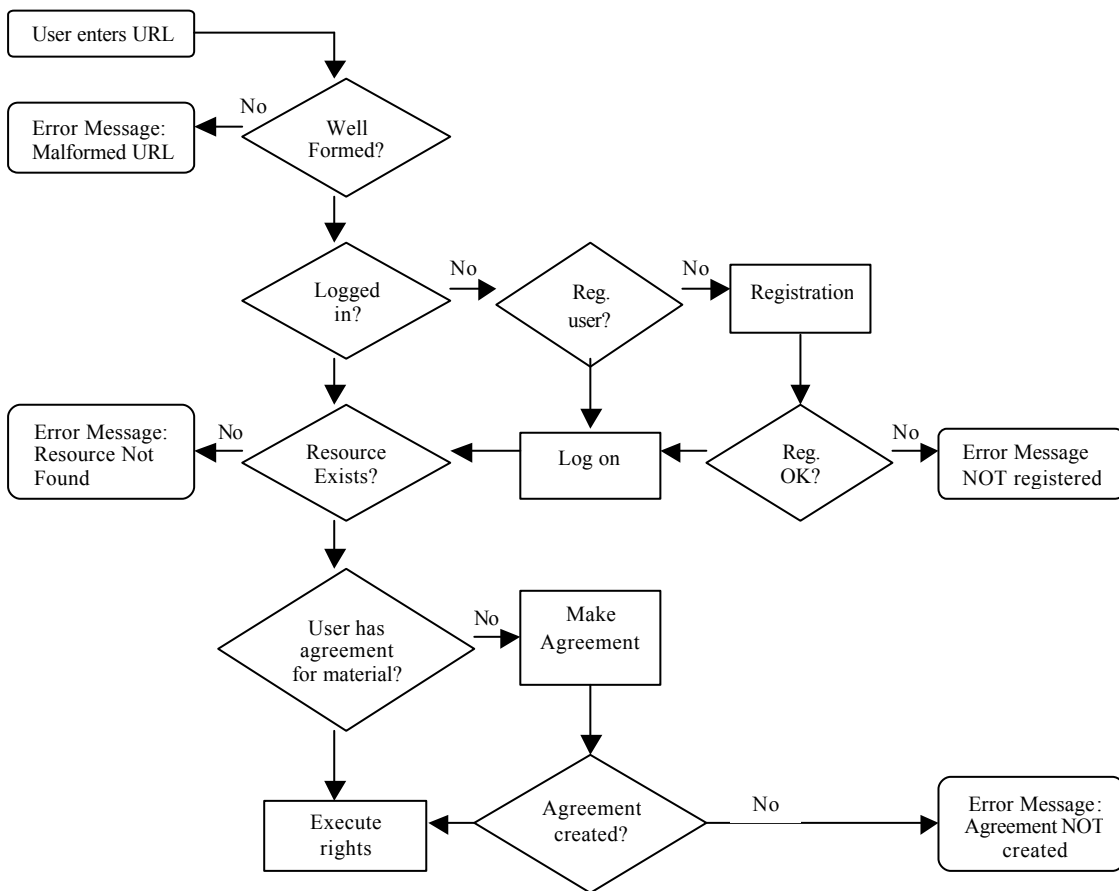


Figure 3: Execution Paths in REAP

5. The REAP Rights Expression Language

The rights expression language created for REAP is a subset of the ODRL rights expression language created during development of the prototype implementation with a few local adjustments and changes. Since the Rights Expression Language created for REAP is defined as a sublanguage to ODRL, the best way of describing it is to describe which parts of ODRL is not part of the language and which local additions have been made. This section presents a short introduction to the most important cut-aways and modifications made to ODRL to create the REAP rights language. Readers are assumed to be familiar with the ODRL rights language.

The advantage of using ODRL (or some other Rights Expression Language) without modifications would be that everyone familiar with the language (including scripts or programs translating from logical choices made by an author to a formal rights description) could make their own rights descriptions that REAP could understand and act upon. This would facilitate trading of rights over material over the entire life cycle of a material from creation to use and reuse, possibly including redistribution. However, the REAP software would then be required to take the entire ODRL language as input, leading to a situation where the software either had to be able to interpret all rights expressions in ODRL and act upon them, or to ignore parts of rights expressions it could not interpret correctly, denying users access to rights for which the software could not verify all requirements, constraints and conditions. Neither creating software that could understand all aspects of ODRL nor creating software that could detect expressions it could not verify seemed feasible given the limited time available for the development of REAP.

First of all, since the language is only intended to be used for describing which rights to be given to direct users of the library, the REAP rights language can only describe usage rights (display, print, execute and play). REAP is assumed to have been given the rights to the material needed for licensing these usage rights to the users of the library.

Next, the language does not have a security model. This means that offers and agreements can not be digitally signed and material and rights descriptions cannot be encrypted. Also, the language does not support the condition construct of the permission element that act like triggers revoking rights when certain conditions are no longer met. Neither are revoke constructs that, when entered into the system, revokes the rights previously offered and traded with. Utility constructs and functionality like Containers, Expression Linking and Inheritance are also not supported.

Assets are identified locally by a REAP identifier consisting of a collection id and an item id. This local identifier enables REAP to translate internally to a file/document to be retrieved without disclosing the location of the files to the user. (If the asset is not available in digital form, this kind of identifier is still used, although a resolution of the identifier will not result in a reference to a file. Assets can, as in ODRL, be further described by a context element. The context element contains an element called uid, which is a unique identifier for the material. It is important that this element does not point to an alternative download location for the material. The requirements model in the REAP rights expression language is highly simplified. The only requirement that can be defined is payment, which is limited to prepay, requiring the user to pay for access to rights before being given such rights.

As in ODRL, there is a Constraint model containing among other things a count element which is used to indicate how many times a given right can be executed over the material. Under the count element contains, in REAP, the elements max and executed. The executed element has been added to the Rights Language of REAP and is not part of the ODRL. It is initially set to zero and is incremented by the REAP software each time a right is executed. See **figure 4** below. Access to a right constrained by count is granted or denied as a result of comparing the value of max and executed. However, the information about how many times a user has executed a right over a material should have been recorded in the profile of the user, not in the agreement. Storing this information in the agreement requires the software to write to the agreement each time a user accesses the right in question and probably makes digital signing of agreements impossible.

.....
<permission>

```

    <print>
      <constraint>
        <count>
          <max>2</max>
          <executed>0</executed>
        </count>
      </constraint>
    </print>
  </permission>
  .....

```

Figure 4: *The executed element is used to indicate how many times a given permission has been executed.*

There are also other constructs from ODRL that are not part of the REAP rights expression language. Most of them have been excluded in order to make a language easy to understand by the REAP software. The drawback is that the REAP rights expression language is not as flexible as the ODRL language. The logic of the rights expression language of REAP is however mainly the same as that of ODRL.

6. Discussion

Being a first attempt to create an adequate DRM solution for digital libraries, there are some issues that have to be addressed in relation to REAP.

Works that are described by a rights language are often realized as a set of files. For instance, a work can often be a set of text documents, images, video and citations. HTTP transported documents resolves this by letting the rendering application (browser) issue subsequent request to the server, one for each part of the document. REAP is supporting one file per rights description as the internal resolution from a collection id and an item id results in a path to a single file. Issuing successive requests for delivery of material to REAP in its present form would require an agreement to be set up for each part of the information object/asset.

REAP does not support renegotiation of agreements. If there is a constraint on a right so that the particular right can only be exercised 4 times, there is no functionality to renegotiate the agreement, letting the user obtain extended rights or new rights to a material.

The terms offer and agreement inherited from ODRL can be misleading. One would think that making an agreement would involve some kind of negotiation or two-way dialog between parties where the parties makes an agreement based on some middle ground. However, since the user cannot influence the initial rights offers given by authors REAP is based more on a take it or leave it concept where the only rights a user can obtain is a full or partial subset of the rights from the initial rights offer. It is a

Authors are meant to be able to use the Rights Expression Language of REAP to express rights in order to protect the material from usage in violation of copyright. However, there is nothing stopping authors from making expressions in the language more targeted at protecting the business interests of the authors or publishers than protecting copyright over the material. Copyright legislation in most countries is balancing the needs and power of authors and users of information. However, since distribution of material over the Internet is in its nature transnational and copyright is defined nationally and varies, Rights Languages have to be flexible enough to express copyright independently of any nation's legislation. The actual expressions made in a language should be in according to a certain copyright legislation, but the expressional power of the language itself should not be limited by single nation's legislation.

REAP is a prototype of a DRM system. It is an attempt at creating a rights management system and can be seen as a first step to create more adequate solutions. The development and testing of REAP has provided valuable insight into the problems and opportunities inherent in DRM systems. The prototype is currently not under further development.

7. Acknowledgement

I wish to thank my supervisor Prof. Ingeborg Sølvsberg for invaluable help during the work with REAP and my master thesis in general and for valuable comments on this paper.

8. References

[Alexandria]

Alexandria Digital Library Project
<http://www.alexandria.ucsb.edu/>

[DRMWatch]

DRMWatch
GiantSteps Media Technology Strategies
<http://www.giantstepsmts.com/drmwatch.htm>

[Erickson 2001]

*Information objects and Rights Management
A Mediation-Based Approach to DRM Interoperability*
John S. Erickson, Hewlett-Packard Laboratories
<http://www.dlib.org/dlib/april01/erickson/04erickson.html>

[Iannela2001]

Digital Rights Management (DRM) architectures
Renato Iannela
Chief Scientist, IPR System
<http://www.dlib.org/dlib/june01/iannela/06iannela.html>

[ODRL]

Open Digital Rights Language Specification version 1.1
<http://www.odrl.net/1.1/ODRL-1.1.pdf>

[PayetteLagoze2000]

Policy-Carrying, Policy-Enforcing Digital Objects
Sandra Payette and Carl Lagoze
Research and advanced Technology for Digital Libraries
4th European Conference on ECDL
Lisbon, Portugal Sept 2000, Proceedings

[RosenBlatt, Trippe, Mooney 2002]

Digital Rights Management Business and Technology
Bill Rosenblatt, Bill Trippe, and Stephen Mooney
M & T Books 2002
ISBN 0-7645-4889-1

[Stefic1997]

Letting Loose the Light: Igniting Commerce in Electronic Publication
In: *Internet Dreams: Archetypes, Myths and Metaphors*
Authors Mark J Stefik, Vinton g.Cerf
ISBN: 0262692023 (may 9. 1997)

[Vestavik2002]

REAP Et system for rettighetsstyring i digitale bibliotek.
Øyvind Vestavik, 2002
Available in Norwegian from
<http://www.idi.ntnu.no/~oyvindve/MasterThesis.pdf>

A Proposal for the Evolution of the ODRL Information Model

Susanne Guth and Mark Strembeck
Department of Information Systems, New Media Lab
Vienna University of Economics and BA, Austria
{firstname.lastname}@wu-wien.ac.at

Abstract

In this paper, we discuss the information model of ODRL 1.1 with respect to the definition of rights and duties for contract parties. We identify a number of shortcomings, and propose an evolutionary advancement of the ODRL. In particular, we present a modified information model and corresponding XML schemas.

1 Introduction

A contract typically represents an agreement of two or more parties. The contract specifies rights and obligations of the involved stakeholders with respect to the subject matter of the respective contract. Digital contracts are most often defined via special purpose XML-based *rights expression languages* (REL), such as ODRL [13], XrML [6], or MPEG 21 REL [7].

On the one hand, a language for the definition of digital contracts should enable the automated processing of digital contracts via software programs. On the other hand, the resulting contracts should also be human-readable and also valid in law. In order to fulfill these requirements, languages for the definition of digital contracts must provide a straightforward grammar and a fixed (but extensible) and unambiguous vocabulary. Moreover, they should simultaneously be flexible enough to express a wide variety of different business cases.

In this paper, we discuss the information model of ODRL 1.1 with respect to the definition of rights and duties for different contract parties. We identify a number of shortcomings, and propose an evolutionary advancement of the ODRL. In particular, we present a modified information model and the corresponding XML schemas.

The remainder of this paper is structured as follows. In Section 2 we give an overview of ODRL 1.1 and identify a number of drawbacks in that specification. Section 3 then introduces a proposal for a future version of the ODRL information model. Subsequently, Section 4 briefly discusses our changes to the ODRL XML schemas. Moreover, we give an example to show how our approach helps to remove different disadvantages of ODRL 1.1. Section 5 provides an outline of (technical) issues which must be considered when mapping permissions and duties defined in digital contracts to enforceable policy rules in concrete software systems. Section 6 gives a brief overview of related work and Section 7 concludes the paper.

2 Definition of Digital Contracts with ODRL 1.1

In this section we introduce the language constructs offered by ODRL 1.1 to define the rights and duties of different contract parties. Section 2.1 presents the respective elements of the ODRL 1.1 information model.

Section 2.2 discusses drawbacks of the current ODRL information model with respect to the expressiveness, the understandability/comprehensibility for human users, and the automated processing of ODRL-based contracts via different software services, e.g. access control services (see also [11]).

2.1 Subset of the ODRL Information Model

Figure 1 shows a subset of the ODRL 1.1 information model [13]. *Party*, *Asset*, and *Permission* are the core elements of ODRL (these elements are subelements of the ODRL *Rights* element, the ODRL *Offer* element, or the ODRL *Agreement* element which are not shown in the figure). The three core elements allow for the definition of simple rights expressions, e.g. "Ms. Guth (party) has the right to *play* (permission) the movie *Lola Runs* (asset)". Note that in ODRL a permission is an operation, such as *play*, *print*, or *copy*, whereas in the area of access control a "permission" is an $\langle operation, object \rangle$ pair, such as $\langle play, movie \rangle$ (see also [8, 9, 5]). Sometimes permissions need to be constrained, e.g. to a specific time-interval or by the maximum number of uses (see e.g. [1, 14, 16]). ODRL offers three different language elements to define constraints:

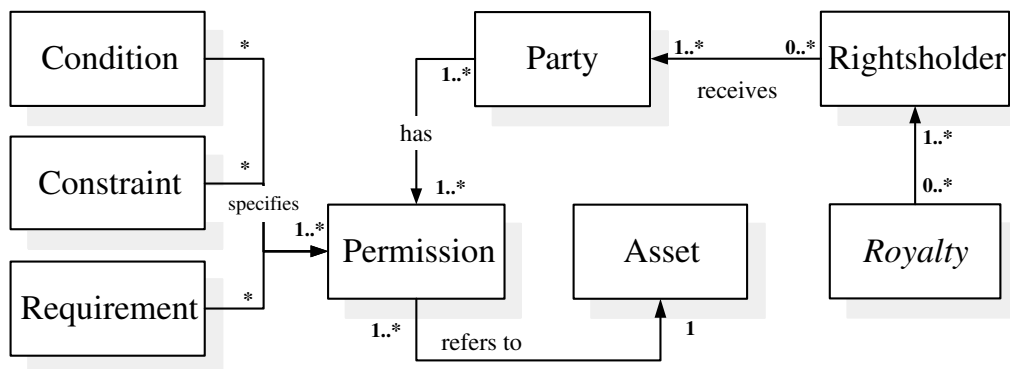


Figure 1: Excerpt of the ODRL 1.1 information model

- A *Requirement* element defines a specific type of precondition (for permission assignments). In particular, an ODRL requirement states that the permission it is related to may only be granted to the respective beneficiary if the corresponding requirement is fulfilled. In ODRL, monetary payments are the most common type of such “requirements”.
- The *Constraint* element of ODRL is intended to narrow ODRL permissions. For example, a “play” permission can be constrained to a maximum of five usages via a *count* constraint. ODRL provides a number of (predefined) constraints: user-, device- bound-, temporal-, aspect-, target-, and rights constraints (for details see [13]).
- An ODRL *Condition*, in essence, define constraints which restrict the validity of a permission. Once a condition is fulfilled, the condition renders the respective permission as no longer valid.

In contracts the party element usually occurs twice, once for the beneficiary (also: consumer or buyer) and once for the so called rightsholder (or seller). In ODRL a rightsholder is identified via a *Rightsholder* element nested in a party element. ODRL party elements that do not include a rightsholder element per definition

“automatically” reference a beneficiary. Additional information related to rightsholders can be specified via a *Royalty* element (see Figure 1). Concrete ODRL royalty elements are *Fixed Amount* and *Percentage* (for details see [13]). These constructs either define a fixed amount or a percentage of the revenues resulting from the corresponding business transaction, and can be assigned to rightsholder parties of a digital contract.

The elements described above are used to express rights and duties in ODRL-based digital contracts. The ODRL example below expresses that: The two parties “Ms. Guth” and “Mr. Strembeck” have reached an agreement on the purchase of the right “display” to an asset identified as “ODRL workshop proceedings”. Mr. Strembeck is the consumer and Ms. Guth is the rightsholder of the workshop proceedings. The “display” right costs €5.00 and is associated with a time constraint that expires on January 1st, 2011. 100 percent of the agreed upon royalties go to Ms. Guth.

```
<?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:agreement>
    <o-ex:party>
      <o-ex:context>
        <o-dd:uid>x500:c=AT;o=Registry;cn=sguth</o-dd:uid>
        <o-dd:name>Dr. Susanne Guth</o-dd:name>
      </o-ex:context>
      <o-ex:rightsholder>
        <o-dd:percentage>100</o-dd:percentage>
      </o-ex:rightsholder>
    </o-ex:party>
    <o-ex:party>
      <o-ex:context>
        <o-dd:uid>x500:c=AT;o=Registry;cn=mstrembe</o-dd:uid>
        <o-dd:name>Dr. Mark Strembeck</o-dd:name>
      </o-ex:context>
    </o-ex:party>
    <o-ex:asset>
      <o-ex:context>
        <o-dd:uid>urn:wu-wien.ac.at#proc01</o-dd:uid>
        <o-dd:name>ODRL Intl. Workshop '04 Proceedings</o-dd:name>
      </o-ex:context>
    </o-ex:asset>
    <o-ex:permission>
      <o-dd:display>
        <o-ex:constraint>
          <o-dd:datetime>
            <o-dd:end>2010-12-31</o-dd:end>
          </o-dd:datetime>
        </o-ex:constraint>
        <o-ex:requirement>
          <o-dd:peruse>
            <o-dd:payment>
              <o-dd:amount o-dd:currency="EUR">5.00</o-dd:amount>
            </o-dd:payment>
          </o-dd:peruse>
        </o-ex:requirement>
      </o-dd:display>
    </o-ex:permission>
  </o-ex:agreement>
</o-ex:rights>
```

2.2 Drawbacks of ODRL 1.1

We now discuss drawbacks that result from the ODRL 1.1 information model. The discussion focuses on the expressiveness, the understandability/comprehensibility for human users, and the automated processing of ODRL-based contracts via different software services.

- 1. Expression of rights and duties.** Generally (and non-technically) speaking, contracts specify the rights and duties of contract parties. In the example from Section 2.1, a customer receives the *right* to display some resource after fulfilling his *duty* of paying a certain amount of money. The rights and duties of the rightsholder, however, are less explicit. With respect to the same example, the rightsholder receives a certain amount of money in return for the usage right of her resource. Here, the respective royalty element (“percentage”) may also be interpreted as a right of the rightsholder (e.g. “the right to debit the consumers account”). But does the rightsholder have duties, too? A human reader of the example contract shown in Section 2.1 may use his knowledge on the nature of contracts in general to interpret the contract and to identify the duty of the rightsholder that, in return to receiving a certain amount of money, she has to make the asset (the digital good) available to the consumer. Nevertheless, such implicit information can, in the general case, not easily be derived via an automated processing of ODRL-based contracts. This results from the fact that ODRL does not provide language elements that can explicitly express (arbitrary) duties of contract parties (aside from monetary payments, as mentioned above).
- 2. Distinction between the rightsholder and consumer parties.** Let us assume that a certain agreement shall include duties of two contract parties, as it is common in barter for instance (a barter is an agreement on the exchange of one asset against another asset, in contrast to a monetary payment). For example, an Austrian university department offers its learning resources (presentations, papers, etc.) to a German university department. In return, the German department agrees to provide its learning resources to the Austrian department. In this example, two parties exchange usage rights for certain digital goods and no monetary payment is required of either department. Thus, both parties are rightsholder *and* beneficiary at the same time. If, however, both parties in an ODRL-based contract include the rightsholder element, the ODRL expression of granting mutual access rights gets ambiguous. Therefore, the simple example already indicates that ODRL 1.1 is not well-suited to model situations where different contract parties “act” in multiple contract roles. In particular, a rightsholder party cannot be treated as a consumer at the same time.
- 3. The expression of constraints on requirements.** A common type of expression used in contracts is the following: “. . . the payment has to be made within four weeks after receipt of the shipment.” With respect to ODRL 1.1 this expression is a requirement (the payment) that is narrowed by a constraint (within four weeks). However, ODRL 1.1 does not allow to constrain requirements.
- 4. The term “permission”.** In ODRL *permissions* refer to certain *operations*, for example “play” or “print”. However, in the area of system security and access control a permission refers to an $\langle operation, object \rangle$ pair as $\langle play, movie \rangle$, for example. Moreover, as in the area of access control, ODRL provides *constraints* that can be used to define specific “side-conditions” on *permissions*. Nevertheless, due to the above mentioned difference the approaches significantly differ, since in ODRL, constraints are associated with operations rather than $\langle operation, object \rangle$ pairs (see also Figure 2).
However, if we constrain operations rather than $\langle operation, object \rangle$ pairs, we need additional means to specify the asset(s) that a certain constraint applies to (at least if it should not apply to all possible assets

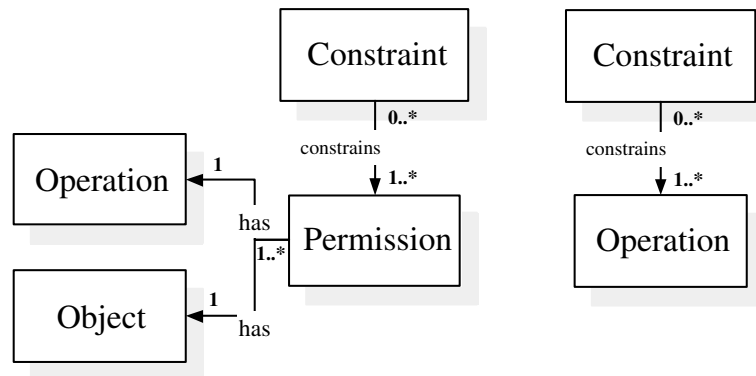


Figure 2: Constraints on Permissions vs. Constraints on Operations

the corresponding operation could be used on). In other words, if more than one asset (e.g. three PDF documents) is referenced in a contract, the operation (e.g. view or print) is constrained for each of these assets. This, however, causes obvious problems if a constraint should refer to one specific asset only (e.g. a specific PDF document, and thereby to a specific $\langle operation, object \rangle$ pair as $\langle print, lecturenotes \rangle$), while the same operation (e.g. print) should not be constrained for the other assets referenced in the contract. Therefore, in the general case, it is more flexible (and more expressive) to enable the definition of constraints on $\langle operation, object \rangle$ pairs than on operations only. In particular, this allows to unambiguously define constraints that clearly refer to a specific $\langle operation, object \rangle$ pair.

5. **Different types of “conditions” in ODRL.** ODRL version 1.1. distinguishes between constraints, requirements and conditions (see Section 2.1) which can be assigned to ODRL permissions (i.e. operations). As an example, let us consider an ODRL payment requirement, an ODRL time condition, and an ODRL count constraint (see also [13]). A permission that has these elements assigned to it, may then be granted *iff* the payment was settled, *iff* the time restriction is not met (i.e. the right is not yet expired), and *iff* the count limit is not exceeded. Each of these clauses defines a certain “side-condition” for a respective permission. With respect to our discussion points 1 and 3 mentioned above, we believe that it would be more reasonable to distinguish between ODRL conditions/constraints on the one hand and ODRL requirements (which in essence define duties) on the other. ODRL conditions and ODRL constraints directly relate to a permission (e.g. do not play *after 12/31/05*; or play at most *five times*) whereas requirements are more of a precondition that has to be fulfilled before a right may actually be assigned to the corresponding beneficiary. Therefore, we argue that ODRL requirements are classical duties, whereas ODRL conditions and ODRL constraints should be seen as constraints on $\langle operation, object \rangle$ pairs.

3 Proposal for a Future ODRL Information Model

In this section, we propose an information model that can be considered in a future version of ODRL. In particular, we introduce *duty* as a new element that can be assigned to contract parties and we use *constraint* as the sole element to define “side-conditions” on duties or permissions (which are defined as $\langle operation, object \rangle$ pairs rather than operations in ODRL 1.1, see below). The rightsholder element and the related royalties have been removed. Figure 3 shows the a proposal for an information model which could serve as an replacement

for the model depicted in Figure 1. The elements of the proposed information model and their relations are described in this section. In subsequent paragraphs, contracts that are compliant to the proposed ODRL information model are referred to as *future* (ODRL) contracts.

Each future ODRL contract contains two or more contract parties. A *contract party* may be either a physical person (an individual), a legal person (an organization), or an abstract type of party (a role). Typical types of parties in a contract are buyer, seller, rightsholder, or beneficiary. A *permission* essentially consists of an $\langle operation, object \rangle$ pair and grants the right to perform the corresponding operation (e.g. play) on the respective object (e.g. a particular MPEG video file). Here, the object represents a reference to a certain asset, and an *asset* is defined as a good or service, be it digital or physical. Each permission in an future ODRL contract is assigned to at least one contract party, and each contract party may possess a number of permissions.

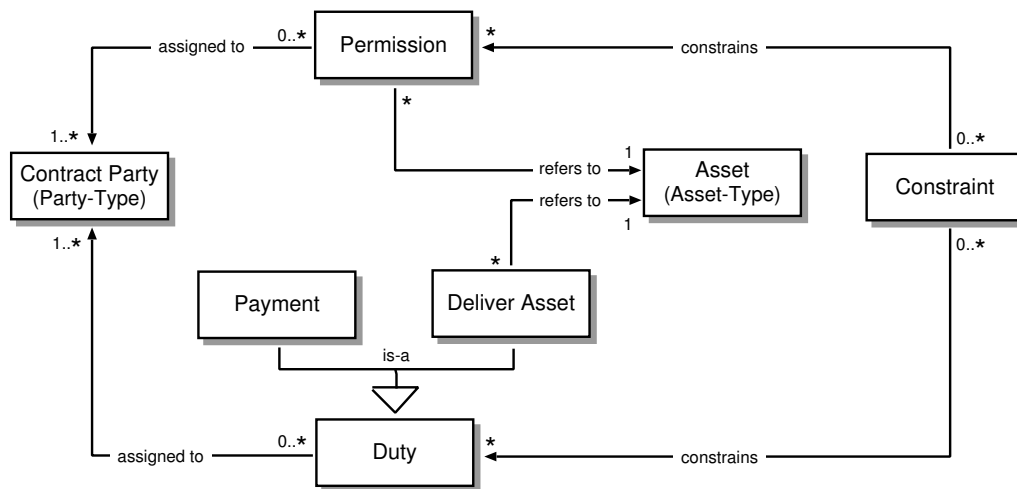


Figure 3: Proposal of a Future ODRL Information Model

A *duty* defines a reward for a certain permission. This reward may be, for example, a (monetary) *payment* or the duty to *deliver an asset*. Duties are optional elements in an electronic contract. Each duty is assigned to at least one contract party which is thereby (i.e. via the contract) obliged to fulfill this duty. On an abstract (business process) level, a duty can be seen as the “counterpart” of a permission, which means that a party receives one or more permissions in exchange for the fulfillment of one or more corresponding duties.

Permissions and duties may be associated with constraints. Here, a *constraint* is a predicate and defines a (side-)condition on the respective permission or duty. A typical example of a constraint may be a time constraint which checks if the current date (today's date) is previous to “December 31st, 2005”. Such a constraint could be associated with:

- a permission to define that this particular permission is valid while the corresponding constraint is fulfilled (i.e. while the respective predicate returns true), or
- a duty to define that the corresponding duty must be fulfilled before the respective constraint "expires" (i.e. before the respective predicate returns false). For example, such a constraint can indicate, that a certain duty, e.g. a payment, has to be made before "December 31st, 2005".

The information model proposed in this section thereby eliminates the drawbacks described in Section 2.2.

4 Proposed Changes for the ODRL Expression Language XML Schema

We now give an overview of proposed modifications of the ODRL XML schemas. These modifications should allow for the definition of ODRL-based digital contracts which adhere to the information model suggested in Section 3. Corresponding XML schemas can be found in Appendix A and B of this paper. These schemas include the following modifications:

- A new `duty` element. This element has the type `dutyType` and includes `dutyElements`. Concrete `dutyElements`, such as `prePayment`, `perusePayment`, or `deliverAsset` are defined in the corresponding ODRL data dictionary schema. The data dictionary currently does not include other duties that are available as requirements in ODRL 1.1, such as `tracked`, `attribution`, `accept`, and `register` (see also [13]). Those element have to be newly defined accordingly.
- The `permissionType` element has been revised and now includes the mandatory elements `operation` and `object` as well as the element `beneficiary` and the attribute `grantor`. The `hasConstraint` sub-element is used to a link to constraint elements that are associated with the corresponding `permission` element.
- Conditions and requirements, as defined in ODRL 1.1, have been removed and are not part of our proposed ODRL schemas. Requirements are now represented as duties. All conditions that narrow permissions are modeled as constraints.
- The `constraintType` has been heavily modified and now includes `operand` and `operator` elements. Moreover, a corresponding `operatorType` and an `operandType` were added to the expression language. The type of an operator is determined via a corresponding `type` attribute that defines if an operator is a prefix, a postfix or an infix operator. Furthermore, the `operatorType` includes a `valence` attribute, defining if the operator is an unary, binary, ternary or n-ary operator. The `operandType` includes the `position` attribute which indicates whether the the operand is located e.g. to the left or to the right of a binary infix operator. An example of a constraint using the binary infix operator `<=` could be `date <= 2005-01-01`.
- The `rightsholder` element of ODRL 1.1 has been removed. Rights and duties of a rightsholder can now be represented by the `permission` and `duty` elements.

The listing below depicts an XML instance of the proposed future ODRL XML schemas and shows how the identified shortcomings of ODRL 1.1 can be removed. With respect to its content, the example resembles the contract described in Section 2.1. To demonstrate the increased expressiveness the contract includes additional statements:

- The listing includes rights (or permissions) and duties. In particular, Mark Strembeck receives the permissions `display` and `print`, and in return, he accepts the duty of a prepayment over the amount of €5.00. For each permission a `grantor` and a `beneficiary` can be specified. In the same way a `bearer` can be specified for each duty. Therefore, permissions and duties can be assigned to each contract party, be it a seller or a buyer. This issue addresses drawback 1 discussed in Section 2.2.

- The example no longer distinguishes between a rightsholder party or a consumer party. This permits that each contract party may receive permissions, grant permissions, and/or be associated with duties. This issue addresses drawback 2 discussed in Section 2.2
- The new *duty* element helps eliminating drawback 3 explained in Section 2.2. In particular, the listing shows that constraints can be related to duties. In the example, a time constraint is related to the `prePayment-Duty`, expressing that the payment has to be settled until April 4th, 2004.
- The example also shows the new shape of the `pppermission` element that now includes operations and objects. Constraints are now related to the $\langle operation - object \rangle$ pair and no longer to the operation only. This method permits a clear translation from ODRL expressions to access control information, and thus addresses drawback 4, as discussed in Section 2.2. Additionally, the listing shows that each constraint (e.g. `constraint01`) can be assigned to various permissions.
- The rights expression in the listing below defines various constraints. ODRL 1.1 condition elements now are also expressed as constraints. This means that former elements *ODRL Constraint* and *ODRL Condition* are now expressed with one newly shaped element called `constraint`. For example, a condition in ODRL 1.1 states: "If the country where a usage right should be claimed is Australia, then the permission must not be granted." `Constraint03` in the listing shows how this former *ODRL Condition* can be formulated as constraint. This change in the ODRL language addresses drawback 5. Additionally, the constraint element now includes the elements `operator`, `operand`, and `constraint type`. Thus, an arbitrary number of constraints can be formulated without changing the data dictionary. Thereby we are also able to check the fulfillment of duties via constraints (see `constraint02`).

```
<?xml version="1.0" encoding="UTF-8"?>
<o-ex20:rights xmlns:o-ex20="http://odrl.net/2.0/ODRL-EX"
  xmlns:o-dd20="http://odrl.net/2.0/ODRL-DD">
  <o-ex20:agreement>

    <o-ex20:party partyID="party01">
      <o-ex20:context>
        <o-dd20:uid>x500:c=AT;o=Registry;cn=sguth</o-dd20:uid>
        <o-dd20:name>Dr. Susanne Guth</o-dd20:name>
      </o-ex20:context>
    </o-ex20:party>
    <o-ex20:party partyID="party02">
      <o-ex20:context>
        <o-dd20:uid>x500:c=AT;o=Registry;cn=mstrembe</o-dd20:uid>
        <o-dd20:name>Dr. Mark Strembeck</o-dd20:name>
      </o-ex20:context>
    </o-ex20:party>

    <o-ex20:asset assetID="asset01">
      <o-ex20:context>
        <o-dd20:uid>urn:wu-wien.ac.at#proc01</o-dd20:uid>
        <o-dd20:name>ODRL Intl. Workshop '04 Proceedings</o-dd20:name>
      </o-ex20:context>
    </o-ex20:asset>

    <o-ex20:permission grantor="party01">
      <o-ex20:operation>display</o-ex20:operation>
      <o-ex20:object>asset01</o-ex20:object>
      <o-ex20:beneficiary>party02</o-ex20:beneficiary>
```

```

        <o-ex20:hasConstraint id="constraint01"/>
        <o-ex20:hasConstraint id="constraint02"/>
        <o-ex20:hasConstraint id="constraint03"/>
    </o-ex20:permission>
    <o-ex20:permission grantor="party01">
        <o-ex20:operation>print</o-ex20:operation>
        <o-ex20:object>asset01</o-ex20:object>
        <o-ex20:beneficiary id="party02"/>
        <o-ex20:hasConstraint id="constraint01"/>
    </o-ex20:permission>

    <o-ex20:constraint constraintID="constraint01">
        <o-ex20:type>datetime</o-ex20:type>
        <o-ex20:operator type="infix" valency="Binary"> LessThan </o-ex20:operator>
        <o-ex20:operand position="left">today</o-ex20:operand>
        <o-ex20:operand position="right">2011-01-01</o-ex20:operand>
    </o-ex20:constraint>
    <o-ex20:constraint constraintID="constraint02">
        <o-ex20:type>dutyfulfilled</o-ex20:type>
        <o-ex20:operator type="prefix" valency="Unary"> Fulfilled </o-ex20:operator>
        <o-ex20:operand>duty01</o-ex20:operand>
    </o-ex20:constraint>
    <o-ex20:constraint constraintID="constraint03">
        <o-ex20:type>spatial</o-ex20:type>
        <o-ex20:operator type="infix" valency="Binary"> NotEqual </o-ex20:operator>
        <o-ex20:operand position="left">Country</o-ex20:operand>
        <o-ex20:operand position="right">Australia</o-ex20:operand>
    </o-ex20:constraint>
    <o-ex20:constraint constraintID="constraint04">
        <o-ex20:type>datetime</o-ex20:type>
        <o-ex20:operator type="infix" valency="Binary"> LessThan </o-ex20:operator>
        <o-ex20:operand position="left">today</o-ex20:operand>
        <o-ex20:operand position="right">2004-04-23</o-ex20:operand>
    </o-ex20:constraint>

    <o-ex20:duty dutyID="duty01" bearer="party02">
        <o-dd20:prePayment>
            <o-dd20:amount currency="EUR"> 5.00 </o-dd20:amount>
        </o-dd20:prePayment>
        <o-ex20:hasConstraint>constraint04</o-ex20:hasConstraint>
    </o-ex20:duty>
</o-ex20:agreement>
</o-ex20:rights>

```

The next listing shows an example of a barter contract. Here, one permission (modify, ODRL_1.1) is simply exchanged against a second permission (print, ODRL_Workshop_proceedings) between Renato Iannella and Susanne Guth. No monetary payment has to be made by either party; both permissions expire with the end of year 2010.

```

<?xml version="1.0" encoding="UTF-8"?>
<o-ex20:rights
  xmlns:o-ex20="http://odrl.net/2.0/ODRL-EX"
  xmlns:o-dd20="http://odrl.net/2.0/ODRL-DD">
  <o-ex20:agreement>
    <o-ex20:party o-ex20:partyID="party01">
      <o-ex20:context>
        <o-dd20:uid>x500:c=AT;o=Registry;cn=sguth</o-dd20:uid>
        <o-dd20:name>Susanne Guth</o-dd20:name>
      </o-ex20:context>
    </o-ex20:party>
    <o-ex20:party o-ex20:partyID="party02">

```

```

    <o-ex20:context>
      <o-dd20:uid>x500:c=AU;o=Registry;cn=riannel</o-dd20:uid>
      <o-dd20:name>Renato Iannella</o-dd20:name>
    </o-ex20:context>
  </o-ex20:party>

  <o-ex20:asset o-ex20:assetID="asset01">
    <o-ex20:context>
      <o-dd20:uid>urn:wu-wien.ac.at#proc01</o-dd20:uid>
      <o-dd20:name>ODRL Intl. Workshop '04 Proceedings</o-dd20:name>
    </o-ex20:context>
  </o-ex20:asset>
  <o-ex20:asset o-ex20:assetID="asset02">
    <o-ex20:context>
      <o-dd20:uid>urn:odrl.net#ODRLspec1.1</o-dd20:uid>
      <o-dd20:name>ODRL 1.1</o-dd20:name>
    </o-ex20:context>
  </o-ex20:asset>

  <o-ex20:permission o-ex20:grantor="party01">
    <o-ex20:operation>print</o-ex20:operation>
    <o-ex20:object>asset01</o-ex20:object>
    <o-ex20:beneficiary>party02</o-ex20:beneficiary>
    <o-ex20:hasConstraint>constraint01</o-ex20:hasConstraint>
  </o-ex20:permission>
  <o-ex20:permission o-ex20:grantor="party02">
    <o-ex20:operation>modify</o-ex20:operation>
    <o-ex20:object>asset02</o-ex20:object>
    <o-ex20:beneficiary>party01</o-ex20:beneficiary>
    <o-ex20:hasConstraint>constraint01</o-ex20:hasConstraint>
  </o-ex20:permission>

  <o-ex20:constraint o-ex20:constraintID="constraint01">
    <o-ex20:type>datetime</o-ex20:type>
    <o-ex20:operator o-ex20:type="infix" o-ex20:valency="Binary">LessThan</o-ex20:operator>
    <o-ex20:operand o-ex20:position="left">today</o-ex20:operand>
    <o-ex20:operand o-ex20:position="right">2011-01-01</o-ex20:operand>
  </o-ex20:constraint>
</o-ex20:agreement>
</o-ex20:rights>

```

Note that the XML schemas proposed in this paper (listed in the Appendix) are an approach to a more flexible and expressive future version of ODRL and should be further discussed by the ODRL initiative.

5 Mapping to Enforceable Policies

Another essential aspect that was not yet discussed is the mapping of permissions and duties that are defined on the level of digital contracts to policy rules that can be enforced in an actual software system. Though this topic is by far too complex to be discussed in this paper, we would like to mention the corresponding problem domain and outline some issues arising in this context. Figure 4 shows a simplified information model for permissions and duties as they can be defined on a technical level, i.e. on the level of actual software systems that need to enforce the corresponding policy rules.

A particular difference between the information model for digital contracts introduced in Section 3 and the model shown in Figure 4 is the direct relation between duties and permissions. In a software system, a subject (a party) that must fulfill a certain duty inevitably needs a corresponding permission. Therefore, each duty must

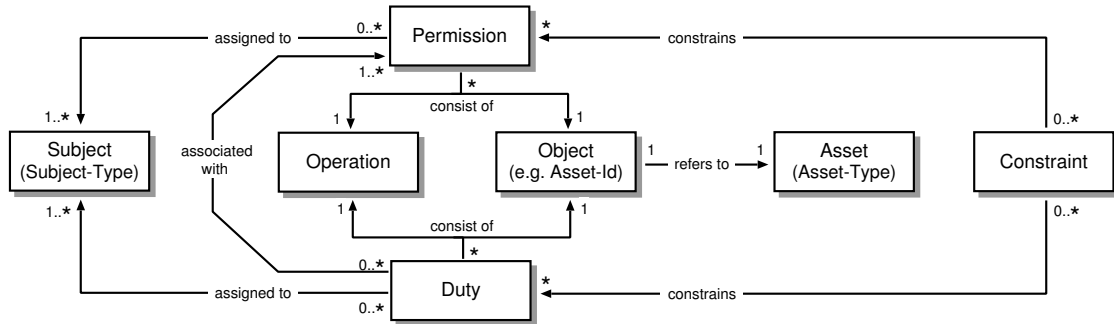


Figure 4: Simplified Information Model for System-level Permissions and Duties

be associated with (at least) one corresponding permission (note that while Figure 4 indicates that permissions and duties consist of $\langle operation, object \rangle$ pairs respectively, two related permission and duty objects do *not* necessarily refer to the same $\langle operation, object \rangle$ pair!). For example, a subject may only fulfill the duty “transfer money” iff the subject simultaneously possesses a corresponding permission which grants access to a specific banking account. Such information, however, is not (and typically should not be) modeled on the level of digital (business) contracts.

Nevertheless, in order to automatically process digital contracts and to enforce permissions and duties that are defined via ODRL contracts, one needs to specify a mapping of contract level elements to corresponding elements on the level of corresponding permissions and duties in concrete software systems. In the general case, each duty (or permission) defined on the contract level maps to one or more permission *and* duty objects on the “technical level” (as indicated by Figure 5). Moreover, constraints defined via a digital contract may (of course) only be enforced on a technical level if a respective (software) service exists which supports the corresponding type of constraints (see e.g. [16]). Another important issue that needs to be addressed is the definition and verification of “trust chains”, for example to examine if a certain grantor (a party granting/assigning a permission to another party) actually is in possession of the respective control right, i.e. if the grantor was allowed/legitimated to pass the corresponding permission to another party (see e.g. [2, 4, 17, 18]).

6 Related Work

In the work of Keller et al. [15] a management architecture for specifying, deploying, monitoring, and enforcing service contracts is proposed to provide a basis for service level agreements. Contracts contain agreements about quality of service (QoS) attributes, they are concluded between service providers and a service integrator. This contract model is tailored to the needs of service level agreements, and thus contains different contract objects than the model discussed in this paper. However, their model also contains basic contract objects, such as provider, customer, and service, as well as objects that represent the guaranteed service parameters (rights). Keller et al., however, do not envision the exchange of contract information between the involved components in a standardized format, such as a rights expression language.

In [3], Beugnard et al. introduce a general model of software contracts that aims at increasing trust and reliability between software components. To conclude contracts between components, every component publishes a feature set to describe its services in a common language (e.g. CORBA IDL). Contracts are established between a client and server component in a negotiation phase where the contract parties agree on certain services.

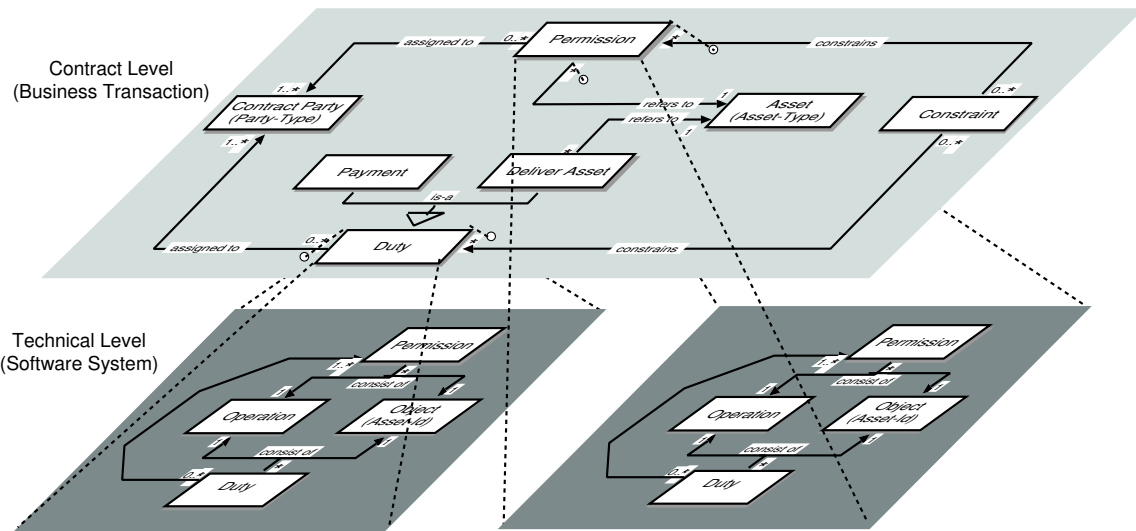


Figure 5: Mapping of Contract Level Elements to Enforceable Policies on the Software System Level

The work provides a basic interface description for the negotiation phase. Beugnard et al. suggest an “XML-formatted description of the contracts” that is applied for negotiation purposes. This is somewhat similar to the approach presented in this paper.

The eXtensible Access Control Markup Language (XACML) [10] is a standard adopted by the Organization for the Advancement of Structured Information Standards (OASIS). XACML provides an XML-based language for the definition of access control policies. The language syntax is formalized via an XML schema. Beside standard access control policies including subjects, operations, and objects, it also allows for the definition of obligations and conditions. Conditions are boolean functions over attributes associated with a subject, an operation, an object, or the system environment. XACML environment attributes are attributes which are relevant to an authorization decision but are independent of a particular subject, operation, or object (see [10]).

The Security Assertion Markup Language (SAML) [12] is an other standard adopted by OASIS. SAML defines an XML-based framework for exchanging security information via computer networks. It is based on the SAML protocol which consists of XML-based request and response messages. By this protocol, clients can request assertions from so-called “SAML authorities” (trusted servers). SAML authorities can make three different kinds of assertion statements: authentications, authorization decisions, and attributes. An authentication assertion confirms that a specific subject has been authenticated by a particular means at a particular time. An authorization decision assertion states that a particular access request consisting of a $\langle \text{subject}, \text{operation}, \text{object} \rangle$ triple has been granted by the corresponding SAML authority. Finally, an attribute assertion confirms that a specific subject is associated with a certain set of attributes.

7 Conclusion and Future Work

In this paper we discussed the information model of ODRL version 1.1. In particular, we focussed on the ODRL *Rightsholder* element, the ODRL *Permission* element, as well as the elements that further describe ODRL Permissions, such as ODRL *Condition*, ODRL *Constraint*, and ODRL *Requirement*. When investigating

the current ODRL information model we found that it has several drawbacks in terms of expressiveness and mapping the ODRL rights information to common access control information models. For example, with the current ODRL information model, rights and duties can only be expressed for the purchasing contract party but not for the seller. Each identified drawback is explained in detail. Consequently, we worked out an improved information model for ODRL that is eliminating the identified drawbacks. The improved information model is translated to new XML schemas as a proposal for future ODRL specifications. The resulting XML schemas can be found in the appendix and may serve as basis for a future version of ODRL. The future work in this field is clearly to embed the improvements into an official, future ODRL version with regard to ensure the timeliness and usability of ODRL.

A Proposed XML Schema for the ODRL Expression Language

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://odrl.net/2.0/ODRL-EX"
  xmlns:o-ex20="http://odrl.net/2.0/ODRL-EX"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
  elementFormDefault="qualified" attributeFormDefault="qualified" version="2.0">
<xsd:import
  namespace="http://www.w3.org/2000/09/xmldsig#"
  schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd"/>
<!-- NOTE: The W3C Encryption Namespace URI will be updated as the specification is advanced -->
<xsd:import
  namespace="http://www.w3.org/2001/04/xmlenc#"
  schemaLocation="http://www.w3.org/Encryption/2001/Drafts/xmlenc-core/xenc-schema.xsd"/>
<xsd:element name="rights" type="o-ex20:rightsType"/>
<xsd:element name="offer" type="o-ex20:offerAgreeType"/>
<xsd:element name="agreement" type="o-ex20:offerAgreeType"/>
<xsd:complexType name="offerAgreeType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element ref="o-ex20:context" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:party" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:asset" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:permission" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:duty" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:constraint" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>
</xsd:complexType>
<xsd:complexType name="rightsType">
  <xsd:complexContent>
    <xsd:extension base="o-ex20:offerAgreeType">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="o-ex20:revoke" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="o-ex20:offer" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="o-ex20:agreement" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="ds:Signature" minOccurs="0"/>
      </xsd:choice>
      <xsd:attributeGroup ref="o-ex20:IDGroup"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="context" type="o-ex20:contextType"/>
<xsd:element name="contextElement" abstract="true"/>
<xsd:complexType name="contextType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element ref="o-ex20:context" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>
</xsd:complexType>
```

```

        <xsd:element ref="o-ex20:contextElement" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:choice>
    <xsd:attributeGroup ref="o-ex20:IDGroup"/>
</xsd:complexType>
<xsd:element name="duty" type="o-ex20:dutyType"/>
<xsd:element name="dutyElement" abstract="true"/>
<xsd:complexType name="dutyType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="o-ex20:dutyElement" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="o-ex20:context" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="o-ex20:hasConstraint" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:choice>
    <xsd:attribute name="dutyID" type="xsd:string" use="required"/>
    <xsd:attribute name="bearer" type="xsd:string" use="required"/>
    <xsd:attributeGroup ref="o-ex20:IDGroup"/>
</xsd:complexType>
<xsd:complexType name="partyType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="o-ex20:context" minOccurs="0"/>
        <xsd:element ref="o-ex20:party" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="o-ex20:container" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="o-ex20:asset" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:choice>
    <xsd:attributeGroup ref="o-ex20:IDGroup"/>
    <xsd:attribute name="partyID" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:element name="party" type="o-ex20:partyType"/>
<xsd:complexType name="assetType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="o-ex20:context"/>
        <xsd:element ref="o-ex20:inherit"/>
        <xsd:element name="digest">
            <xsd:complexType>
                <xsd:choice minOccurs="0" maxOccurs="unbounded">
                    <xsd:element ref="ds:DigestMethod"/>
                    <xsd:element ref="ds:DigestValue"/>
                </xsd:choice>
            </xsd:complexType>
        </xsd:element>
        <xsd:element ref="ds:KeyInfo"/>
    </xsd:choice>
    <xsd:attributeGroup ref="o-ex20:IDGroup"/>
    <xsd:attribute name="assetID" type="xsd:string" use="required"/>
    <xsd:attribute name="type">
        <xsd:simpleType>
            <xsd:restriction base="xsd:NMTOKEN">
                <xsd:enumeration value="work"/>
                <xsd:enumeration value="expression"/>
                <xsd:enumeration value="manifestation"/>
                <xsd:enumeration value="item"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
</xsd:complexType>
<xsd:element name="asset" type="o-ex20:assetType"/>
<xsd:complexType name="inheritType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="o-ex20:context" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:choice>
    <xsd:attribute name="override" type="xsd:boolean" default="false"/>
    <xsd:attribute name="default" type="xsd:boolean" default="false"/>
</xsd:complexType>

```

```

<xsd:element name="inherit" type="o-ex20:inheritType"/>
<xsd:element name="permission" type="o-ex20:permissionType"/>
<xsd:element name="operation" type="operationType"/>
<xsd:element name="beneficiary" type="o-ex20:linkType"/>
<xsd:element name="object" type="o-ex20:assetType"/>
<xsd:complexType name="permissionType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element ref="o-ex20:context" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:operation" minOccurs="1" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:object" minOccurs="1" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:hasConstraint" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:beneficiary" minOccurs="1" maxOccurs="unbounded"/>
  </xsd:choice>
  <xsd:attribute name="exclusive" type="xsd:boolean" use="optional"/>
  <xsd:attribute name="grantor" type="xsd:string" use="required"/>
  <xsd:attributeGroup ref="o-ex20:IDGroup"/>
</xsd:complexType>
<xsd:complexType name="operandType">
  <xsd:attribute name="position" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="type" type="o-ex20:cType"/>
<xsd:element name="operand" type="o-ex20:operandType"/>
<xsd:complexType name="operatorType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="type" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="prefix"/>
            <xsd:enumeration value="infix"/>
            <xsd:enumeration value="post fix"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="valence" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="unary"/>
            <xsd:enumeration value="binary"/>
            <xsd:enumeration value="ternary"/>
            <xsd:enumeration value="n-ary"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="operator" type="o-ex20:operatorType"/>
<xsd:complexType name="constraintType">
  <xsd:choice>
    <xsd:element ref="o-ex20:container" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:type" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="o-ex20:operand" minOccurs="1" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:operator" minOccurs="1" maxOccurs="1"/>
  </xsd:choice>
  <xsd:attribute name="constraintID" type="xsd:string" use="required"/>
  <xsd:attribute name="type" type="xsd:NMTOKEN" use="required"/>
</xsd:complexType>
<xsd:element name="hasConstraint" type="linkType"/>
<xsd:complexType name="linkType">
  <xsd:attribute name="id" type="xsd:string" use="required"/>
</xsd:complexType>

```

```

<xsd:element name="constraint" type="o-ex20:constraintType"/>
<xsd:complexType name="revokeType">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element ref="o-ex20:context" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="o-ex20:IDGroup"/>
</xsd:complexType>
<xsd:element name="revoke" type="o-ex20:revokeType"/>
<xsd:complexType name="sequenceType">
  <xsd:sequence>
    <xsd:element ref="o-ex20:seq-item" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="order" default="total">
    <xsd:simpleType>
      <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="total"/>
        <xsd:enumeration value="partial"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
<xsd:element name="sequence" type="o-ex20:sequenceType"/>
<xsd:complexType name="containerType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element ref="o-ex20:container" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:permission" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:constraint" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:sequence" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:party" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>
  <xsd:attribute name="type" default="and">
    <xsd:simpleType>
      <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="and"/>
        <xsd:enumeration value="in-or"/>
        <xsd:enumeration value="ex-or"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attributeGroup ref="o-ex20:IDGroup"/>
</xsd:complexType>
<xsd:element name="container" type="o-ex20:containerType"/>
<xsd:complexType name="seqItemType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element ref="o-ex20:container" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:permission" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:constraint" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="o-ex20:sequence" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>
  <xsd:attribute name="number" type="xsd:integer" use="required"/>
</xsd:complexType>
<xsd:element name="seq-item" type="o-ex20:seqItemType"/>
<xsd:attributeGroup name="IDGroup">
  <xsd:attribute name="id" type="xsd:ID"/>
  <xsd:attribute name="idref" type="xsd:IDREF"/>
</xsd:attributeGroup>
</xsd:schema>

```

B Proposed XML Schema for the ODRL Data Dictionary

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://odrl.net/2.0/ODRL-DD"
  xmlns:o-ex20="http://odrl.net/2.0/ODRL-EX"
  xmlns:o-dd20="http://odrl.net/2.0/ODRL-DD"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="qualified" version="2.0">
<xsd:import namespace="http://odrl.net/2.0/ODRL-EX" schemaLocation="http://odrl.net/2.0/ODRL-EX-20.xsd"/>
  <!-- Declare the operation Vocabulary -->
  <xsd:simpleType name="operationType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="display"/>
      <xsd:enumeration value="print"/>
      <xsd:enumeration value="play"/>
      <xsd:enumeration value="execute"/>
      <xsd:enumeration value="sell"/>
      <xsd:enumeration value="lend"/>
      <xsd:enumeration value="give"/>
      <xsd:enumeration value="lease"/>
      <xsd:enumeration value="modify"/>
      <xsd:enumeration value="excerpt"/>
      <xsd:enumeration value="aggregate"/>
      <xsd:enumeration value="annotate"/>
      <xsd:enumeration value="move"/>
      <xsd:enumeration value="duplicate"/>
      <xsd:enumeration value="delete"/>
      <xsd:enumeration value="verify"/>
      <xsd:enumeration value="backup"/>
      <xsd:enumeration value="restore"/>
      <xsd:enumeration value="install"/>
      <xsd:enumeration value="uninstall"/>
      <xsd:enumeration value="save"/>
    </xsd:restriction>
  </xsd:simpleType>
  <!-- Declare the Payment Elements -->
  <xsd:element name="payment">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="amount">
          <xsd:complexType>
            <xsd:simpleContent>
              <xsd:extension base="xsd:decimal">
                <xsd:attribute name="currency" type="xsd:NMTOKEN" use="required"/>
              </xsd:extension>
            </xsd:simpleContent>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="taxpercent" minOccurs="0">
          <xsd:complexType>
            <xsd:simpleContent>
              <xsd:extension base="xsd:decimal">
                <xsd:attribute name="code" type="xsd:NMTOKEN" use="required"/>
              </xsd:extension>
            </xsd:simpleContent>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!-- Declare all the Duty Elements -->
  <xsd:element name="perusePayment" substitutionGroup="o-ex20:dutyElement">

```

```

        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="o-ex20:dutyType">
                    <xsd:sequence>
                        <xsd:element ref="o-dd20:payment"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="postPayment" substitutionGroup="o-ex20:dutyElement">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="o-ex20:dutyType">
                    <xsd:sequence>
                        <xsd:element ref="o-dd20:payment"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="prePayment" substitutionGroup="o-ex20:dutyElement">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="o-ex20:dutyType">
                    <xsd:sequence>
                        <xsd:element ref="o-dd20:payment"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="deliverAsset" substitutionGroup="o-ex20:dutyElement">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="o-ex20:dutyType">
                    <xsd:sequence>
                        <xsd:element ref="o-ex20:asset"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <!--Duty elements for accept, register, attribution, tracked have to be formulated accordingly
    Declare all the Context Elements -->
    <xsd:simpleType name="uriAndOrString">
        <xsd:union memberTypes="xsd:anyURI xsd:string"/>
    </xsd:simpleType>
    <xsd:element name="uid" type="o-dd20:uriAndOrString" substitutionGroup="o-ex20:contextElement"/>
    <xsd:element name="role" type="xsd:anyURI" substitutionGroup="o-ex20:contextElement"/>
    <xsd:element name="name" type="xsd:string" substitutionGroup="o-ex20:contextElement"/>
    <xsd:element name="remark" type="xsd:string" substitutionGroup="o-ex20:contextElement"/>
    <xsd:element name="event" type="xsd:string" substitutionGroup="o-ex20:contextElement"/>
    <xsd:element name="pLocation" type="xsd:string" substitutionGroup="o-ex20:contextElement"/>
    <xsd:element name="dLocation" type="xsd:anyURI" substitutionGroup="o-ex20:contextElement"/>
    <xsd:element name="reference" type="xsd:anyURI" substitutionGroup="o-ex20:contextElement"/>
    <xsd:element name="version" type="xsd:string" substitutionGroup="o-ex20:contextElement"/>
    <xsd:element name="transaction" type="xsd:string" substitutionGroup="o-ex20:contextElement"/>
    <xsd:element name="service" type="xsd:anyURI" substitutionGroup="o-ex20:contextElement"/>
    <xsd:element name="date" type="o-dd20:dateType" substitutionGroup="o-ex20:contextElement"/>
    <!-- Declare all the Constraint Elements -->
    <xsd:simpleType name="cType">

```



```

<xsd:restriction base="xsd:string">
  <xsd:enumeration value="dutyfulfilled"/>
  <xsd:enumeration value="individual"/>
  <xsd:enumeration value="group"/>
  <xsd:enumeration value="cpu"/>
  <xsd:enumeration value="network"/>
  <xsd:enumeration value="screen"/>
  <xsd:enumeration value="storage"/>
  <xsd:enumeration value="memory"/>
  <xsd:enumeration value="printer"/>
  <xsd:enumeration value="software"/>
  <xsd:enumeration value="hardware"/>
  <xsd:enumeration value="spatial"/>
  <xsd:enumeration value="quality"/>
  <xsd:enumeration value="format"/>
  <xsd:enumeration value="unit"/>
  <xsd:enumeration value="watermark"/>
  <xsd:enumeration value="purpose"/>
  <xsd:enumeration value="industry"/>
  <xsd:enumeration value="count"/>
  <xsd:enumeration value="minimum"/>
  <xsd:enumeration value="maximum"/>
  <xsd:enumeration value="datetime"/>
  <xsd:enumeration value="accumulated"/>
  <xsd:enumeration value="interval"/>
  <xsd:enumeration value="recontext"/>
</xsd:restriction>
</xsd:simpleType>
<!-- Transfer Permission is defined as a ContainerType to enable complete expression of
rights in the Constraint -->
<xsd:element name="transferPerm" substitutionGroup="o-ex20:container">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="o-ex20:containerType">
        <xsd:attribute name="downstream" default="equal">
          <xsd:simpleType>
            <xsd:restriction base="xsd:NMTOKEN">
              <xsd:enumeration value="equal"/>
              <xsd:enumeration value="less"/>
              <xsd:enumeration value="notgreater"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
</xsd:schema>

```

References

- [1] G.J. Ahn and R. Sandhu. Role-based Authorization Constraints Specification. *ACM Transactions on Information and System Security (TISSEC)*, 3(4), November 2000.
- [2] J. Bacon and K. Moody. Toward Open, Secure, Widely Distributed Services. *Communications of the ACM*, 45(6), June 2002.

- [3] A. Beugnard, J.-M. Jezequel, N. Plouzeau, and D. Watkins. Making Components Contract Aware. *IEEE Computer Magazine*, 32(7), July 1999.
- [4] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proc. of the IEEE Conference on Security and Privacy*, May 1996.
- [5] National Computer Security Center. A Guide to Understanding Discretionary Access Control in Trusted Systems, September 1987. NCSC-TG-003-87.
- [6] ContentGuard Inc. eXtensible rights Markup Language (XrML), Version 2.0. <http://www.xrml.org/>, November 2001.
- [7] T. DeMartini, X. Wang, and B. Wragg. MPEG-21 Working Documents - Part 5 & Part 6, MPEG-21 Rights Expression Language. http://www.chiariglione.org/mpeg/working_documents.htm, March 2003.
- [8] D.E. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*, 19(5), May 1976.
- [9] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security*, 4(3), August 2001.
- [10] S. Godik and T. Moses (eds.). eXtensible Access Control Markup Language (XACML) Version 1.0. <http://www.oasis-open.org>, February 2003. OASIS Standard.
- [11] S. Guth, G. Neumann, and M. Strembeck. Experiences with the Enforcement of Access Rights Extracted from ODRL-based Digital Contracts. In *Proc. of the 3rd ACM Workshop on Digital Rights Management (DRM)*, October 2003.
- [12] P. Hallam-Baker and E. Maler (eds.). Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML). <http://www.oasis-open.org>, November 2002. OASIS Standard.
- [13] R. Iannella. Open Digital Rights Language (ODRL), Version 1.1. <http://odrl.net>, August 2002.
- [14] T. Jaeger. On the Increasing Importance of Constraints. In *Proc. of the ACM Workshop on Role-Based Access Control*, 1999.
- [15] A. Keller, G. Kar, H. Ludwig, A. Dan, and J.-L. Hellerstein. Managing Dynamic Services: A Contract Based Approach to a Conceptual Architecture. In *Proc. of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, April 2002.
- [16] G. Neumann and M. Strembeck. An Approach to Engineer and Enforce Context Constraints in an RBAC Environment. In *Proc. of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT)*, June 2003.
- [17] K.E. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills, and L. Yu. Requirements for Policy Languages for Trust Negotiation. In *Proc. of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY)*, June 2002.
- [18] W.H. Winsborough and N. Li. Towards Practical Automated Trust Negotiation. In *Proc. of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY)*, June 2002.

Distributed Digital Rights Management: The EduSource Approach to DRM

April 15, 2004

Stephen Downes, Gilbert Babin, Luc Belliveau, Raphael Blanchard, Gerard Levy, Pierre Bernard, Gilbert Paquette, Sylvie Plourde

Abstract

This paper describes the design and implementation of the distributed digital rights management (DDRM) system as undertaken by the eduSource project, a national network of learning object repositories built by a consortium of universities and other agencies in Canada. DDRM is foremost a system of rights expression, with transactions managed by a set of brokers acting on behalf of purchasers and providers. Rights are described using ODRL, contained in files managed by the provider broker, and accessed by means of pointers in the learning object metadata exchanged within the eduSource network.

1. eduSource

The eduSource project is a network of Canadian learning object repositories providing access to all Canadian educational institutions to a broad array of educational resources. Funded by the contributions of project partners (more than 30 universities, agencies, and businesses across Canada) and by CANARIE, Canada's Advanced Internet Development Organization, the intent of eduSource is to "create a testbed of linked and interoperable learning object repositories." (McGreal, et.al., 2003) The development of eduSource involves not only the design of a suit of software applications, referred in project documentation as the Repository in a Box (RiB), it is also intended support the ongoing development of standards based tools, systems, practices and protocols necessary for a national learning infrastructure.

eduSource is based in part on three prior CANARIE funded initiatives: Explor@, "a software environment for the delivery of courses or distance learning events" (Technologies Cogigraph 2003), POOL (Portal for Online Objects in learning), a peer to peer learning object distribution network (eduSplash, 2003), and CanLOM, a learning object metadata repository. (CanLOM, 2003) Added to these were CAREO (Campus Alberta Repository of Educational Objects), a learning object metadata repository (CAREO, 2003), institutional services provided by Athabasca University, and a variety of smaller initiatives.

The eduSource project team identified four major goals:

1. To promote and refine a repository metadata framework through the ongoing development of the CanCore protocol;
2. To support experimental research in key areas such as pedagogy, accessibility, protocols, network engineering, hardware integration, quality of service, security, rights management, content development and software applications;
3. To implement a national testbed to investigate processes such as peer review, content repurposing, user support, professional development and content transactions; and
4. To communicate and disseminate its findings through cooperation and partnership with other federal and provincial agencies, institutions and the private sector. (McGreal, et.al., 2003)

Work on the eduSource project began in the summer of 2002. As of this writing, eduSource is projected for launch at the end of March, 2004.

2. eduSource Vision

The eduSource Digital Rights Management initiative has its origins in the eduSource vision. Making eduSource unique was not only its distributed nature, it being an attempt to link a geographically dispersed set of online resources and services, but also the diverse and sometimes conflicting points of view characterizing member initiatives at the outset. For example, while POOL is fundamentally a peer to peer system, similar in many ways to products such as Napster, Explor@ was a relatively traditional learning management system and CAREO a centralized metadata repository.

Moreover, as additional projects came into the fold, a wider array of points of view was added. With the addition of business partners came a desire to see implemented a digital rights management solution, and as a consequence this was incorporated into the original approach. Through the duration of the project, the emergence of such projects as the Open Archives Initiative (OAI) and Rich Site Summary (RSS) added yet another content distribution model for members to consider.

At the crux of many of these different visions lay digital rights management, and accordingly, Canada's National Research Council e-Learning group, as the DRM package manager for eduSource, initiated a 'Vision Committee' to draft broad parameters for the eduSource project. After wide consultation, the Vision Committee produced the following statement as part of its overall document: (Downes, et.al., 2002)

eduSource is to be designed not as a single software application, but rather, as a set of related components, each of which fulfills a specific function in the network as a whole. This enables users of eduSource to employ only those tools or services that suit their need, without requiring that they invest in the entire system. It also allows for distributed functionality; an eduSource user may rely on a third party to provide services to users. The purpose of this principle is to allow for

specialization. Additionally, it allows eduSource users to exercise choice in any of a variety of models and configurations.

Any given software tool provided by eduSource may be replicated and offered as an independent service. Thus, it is anticipated that there will be multiple instances of each type of repository in the network. The purpose of this principle is to provide robustness. Additionally, it is to ensure that no single service provider or software developer may exercise control over the network by creating a bottleneck through which all activities must pass.

In order to realize this objective, the vision committee also endorsed the principle of open standards and open source. Accordingly, they wrote: (Downes, et.al., 2002)

EduSource repositories will use Open Rights Management standards and protocols. The purpose of this is to ensure that there is no a priori overhead cost incurred by agencies wishing to offer services compatible with eduSource. Imposing an a priori cost immediately poses a barrier to small and medium sized enterprises that may wish to participate and it biases the network toward the provision of commercial content only.

This vision was endorsed by the eduSource Steering Committee, which in turn resolved to license all software under the Lesser GNU Public License (LGPL, 1999) and to endorse the use of Open Digital Rights Language (Ianella, 2002) to express digital rights in the eduSource project.

3. DRM Vision

In addition to statements about the design of eduSource as a whole, the Vision Committee defined specific parameters for the development of eduSource Digital Rights Management: (Downes, et.al., 2002)

Any provider of learning materials may prepare and distribute learning materials through the eduSource repository network. eduSource will support the registration and indexing of various providers, this registration will be free and optional. The purpose of this principle is to ensure that providers are not faced with a priori 'membership fees' or similar tariffs in order to gain access to potential purchasers. This does not preclude restrictions, tariffs or controls on specific instances of an eduSource-compliant repository. However, in any case where a restricted component, such as a for-profit metadata repository, exists, an equivalent unrestricted component, such as a public metadata repository, will also exist.

There will be no prior restraint imposed on the distribution model selected by participants in eduSource. Specifically, eduSource will accommodate free content distribution, co-op or shared content distribution, and

commercial fee-based content distribution. The purpose of this principle is to ensure fair and open competition between different types of business models, to ensure that users are not 'locked in' to the offerings provided by certain vendors, to provide the widest possible range of content options, and to ensure that prices charged for learning content most accurately reflect the true market value of that content.

Multiple parties may provide metadata describing a given learning resource. There is no prior restraint exercised by providers of learning materials on evaluations, appraisals, comments and other descriptions of their learning material. The purpose of third party metadata may be to provide alternative classification schemes, to indicate certification compliance, or to provide independent assessments and evaluations of learning resources. The purpose of this principle is to ensure that potential users of learning resources can obtain and input multiple descriptions of that material. It is also to create an environment for the creation of optional but value-added third party services for which fees or other costs may be charged.

eduSource should be considered as an implementation of and an extension of the semantic web. This means that metadata and services provided by eduSource repositories should be available to the semantic web as a whole. It also means that eduSource repositories and tools can and should incorporate elements of the semantic web, such as sector-specific ontologies, into its own design. The purpose of this principle is to ensure that eduSource is capable of the widest reach possible. It is also to reduce the duplication of effort between developers working in specific domains and educators working in the same domain.

The principle behind fee-based and subscription-based transactions is that it should be easier to buy material than to steal it. Thus where possible, the acquisition of rights and the exchange of funds will be automated. The purpose of this principle is to reduce transaction and clearance costs for purchasers of learning materials.

In addition, the structure of DRM within the network should be such as to allow for multiple digital rights models. For example, it should be possible for a government or agency to distribute free materials, for a college association to establish a cooperative system for sharing, and for a commercial provider to sell content on a per-view or subscription based model. Individual learners should have the option to access and, if necessary, purchase materials directly, or they should be able to obtain access to materials through their school board, provincial learning ministry, or employer.

Thus there is no single rights agency governing all transactions. A given provider of learning materials will work with one of many brokers who sell to multiple purchasers, and a given user may one of many agents who conduct transactions with multiple vendors. Vendors and users may select from any number of brokering services, so that no single transaction agent controls the network. Vendors and purchasers may act as their own brokers. A vendor or purchaser may elect to employ multiple brokers. Brokers acting on behalf of, say, a provincial department of education, may represent a given populations, such as the students of that province. The purpose of this provision is to eliminate the need for the creation of multiple accounts, to allow users to user resources from multiple vendors, and to provide a choice of brokers, and therefore a greater likelihood of trust.

In addition to describing digital rights on behalf of content providers, the network should assert individual rights and preferences on behalf of users. Users of the system own their own personal data. Brokers within the network may operate on behalf of the user, and releases information or money only with the agent's explicit consent. The purpose of this principle is to engender trust in the system and to ensure privacy when dealing with multiple agencies.

4. DRM in eduSource Use Cases

The eduSource architecture development process employed a standard methodology, preceding from the vision document, through a set of use cases, and the creation of a UML diagram describing the overall system. The Digital Rights Management package participated in this part of the development.

Use cases provided by the DRM package described typical procedures whereby a person (using IMS DRI terminology, an 'infoseeker') would search for, select, and ultimately purchase an online learning resource through eduSource.

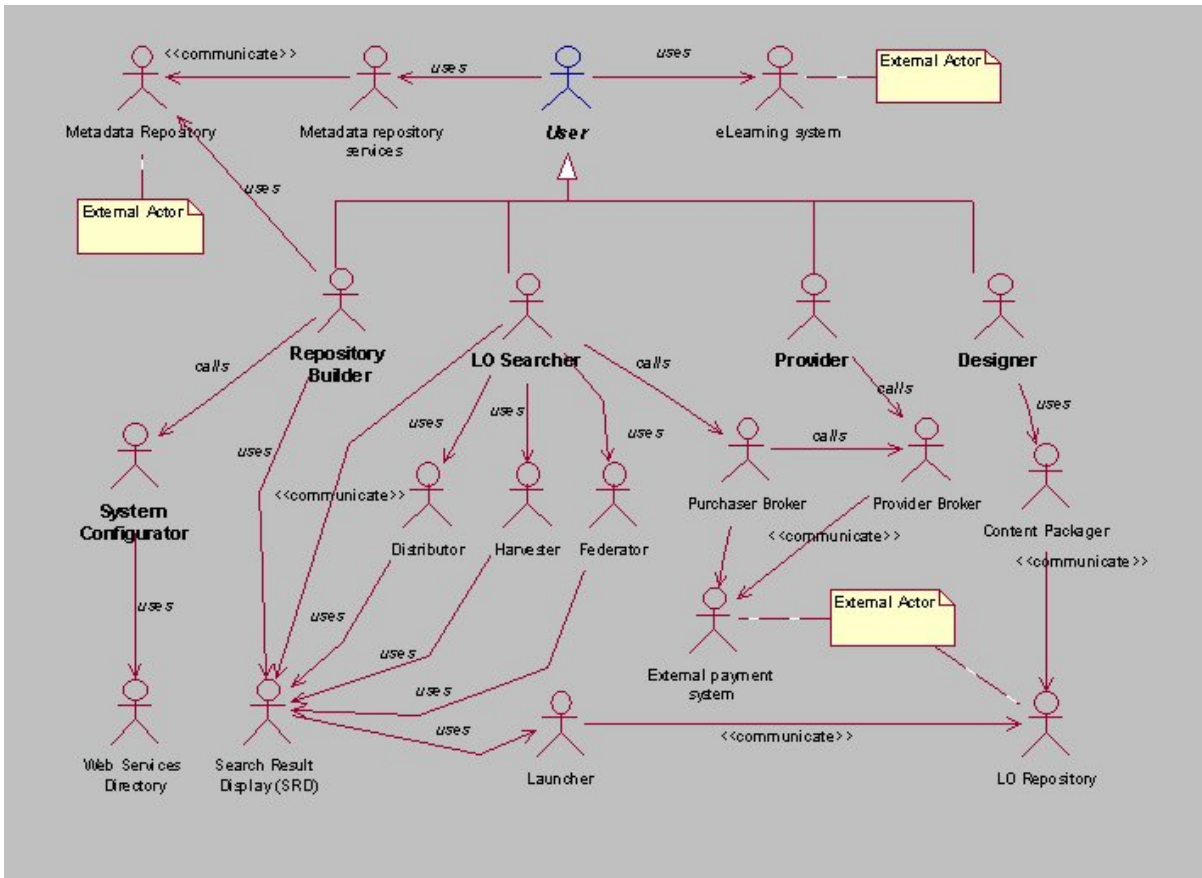


Figure 1. eduSource Use Case Diagram - Overview. Paquet, et.al., 2003

In figure 1, several features of the DRM system to be described below are evident. Digital rights functionalities are provided by two major actors within the eduSource system, the 'purchaser broker' and the 'provider broker' (sometimes documented as the 'vendor broker'). Financial transaction between the two brokers are managed by an external payment agency, such as PayPal or a credit card transactions company. The purchaser broker, in turn, interacts with the LO Searcher (infoseeker), while the provider broker interacts with the provider. This process is displayed in more detail in figure 2.

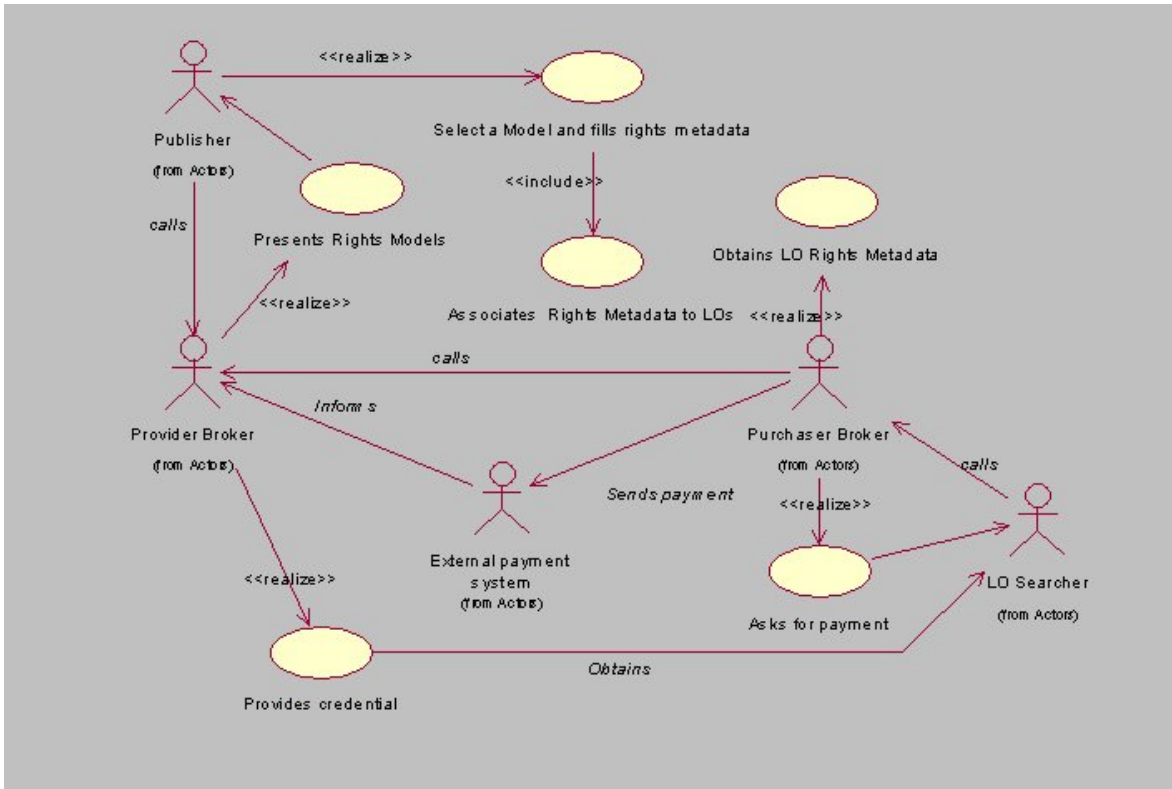


Figure 2. eduSource Use Case Diagram - Digital Rights Management. Paquette, et.al., 2003

As this expanded diagram shows, the provider (or 'publisher') works with the provider broker to create or select a 'rights model'. Information about this rights model is then embedded in learning object metadata. When a searcher retrieves the learning object metadata, he or she may then locate the rights model, which is provided on request by the provider broker. If specified by the rights model, a payment is made for use of the object, via the purchaser broker, and access to the learning object is granted, which is then returned to the infoseeker.

The process of assigning and employing rights is included in the overall eduSource process diagram:

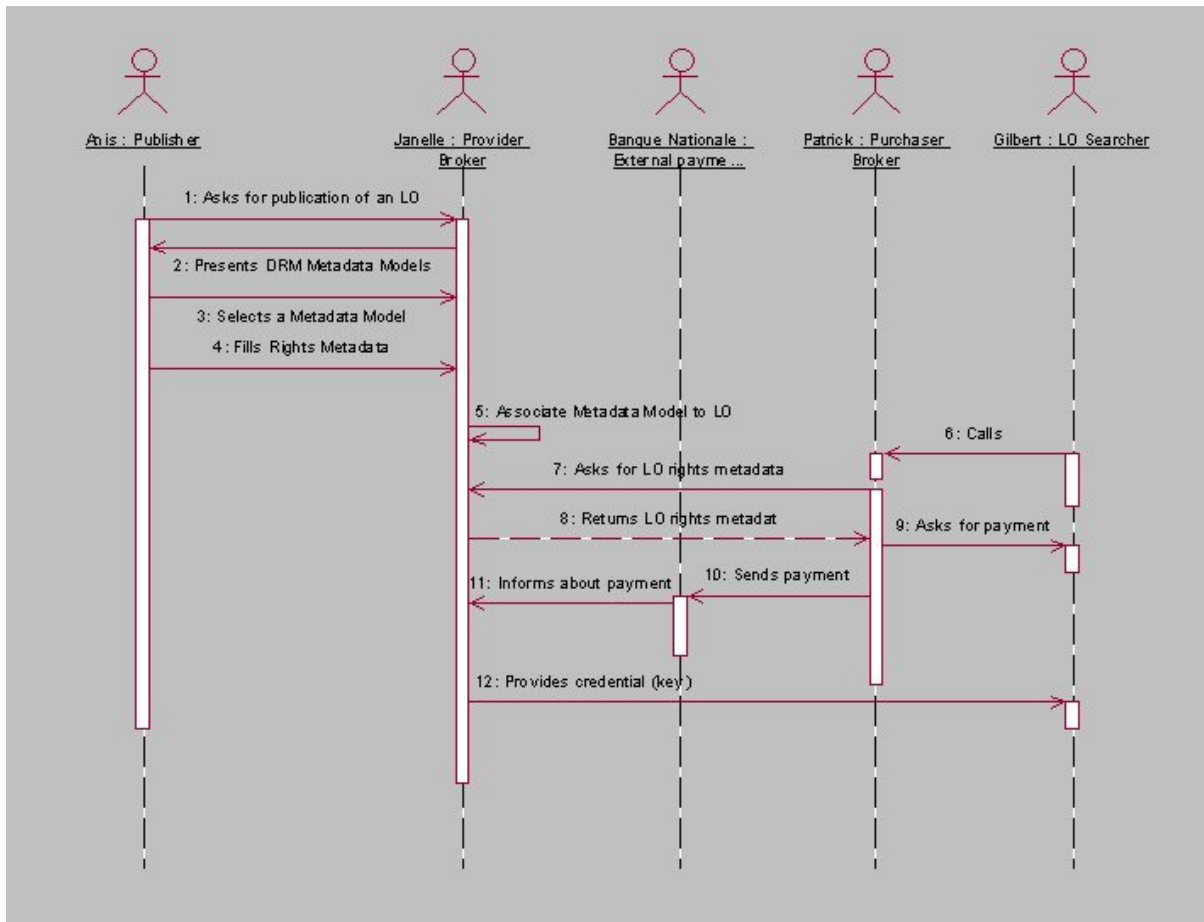


Figure 3. eduSource Process Model - Digital Rights Management. Paquet, et al., 2003

The process diagram in figure 3 is more explicit about the payment and delivery mechanism. What should be noted is that the payment is made, not directly to the provider or even to the provider broker, but rather, to the purchaser broker. The purchaser broker then notifies the vendor broker of the payment, which in turn returns a key that provides access to the learning object.

5. Explanation of the Use Cases

The digital rights model proposed in the use cases introduces some major new features to online digital rights management. First, it introduces the idea of the purchaser broker, in addition to a vendor broker (which, under various names, may be found in other systems, such as the Microsoft Rights Management Server). Second, instead of embedding digital rights metadata in an object or object metadata, it stores only a pointer to that metadata. These are depicted in figure 4.

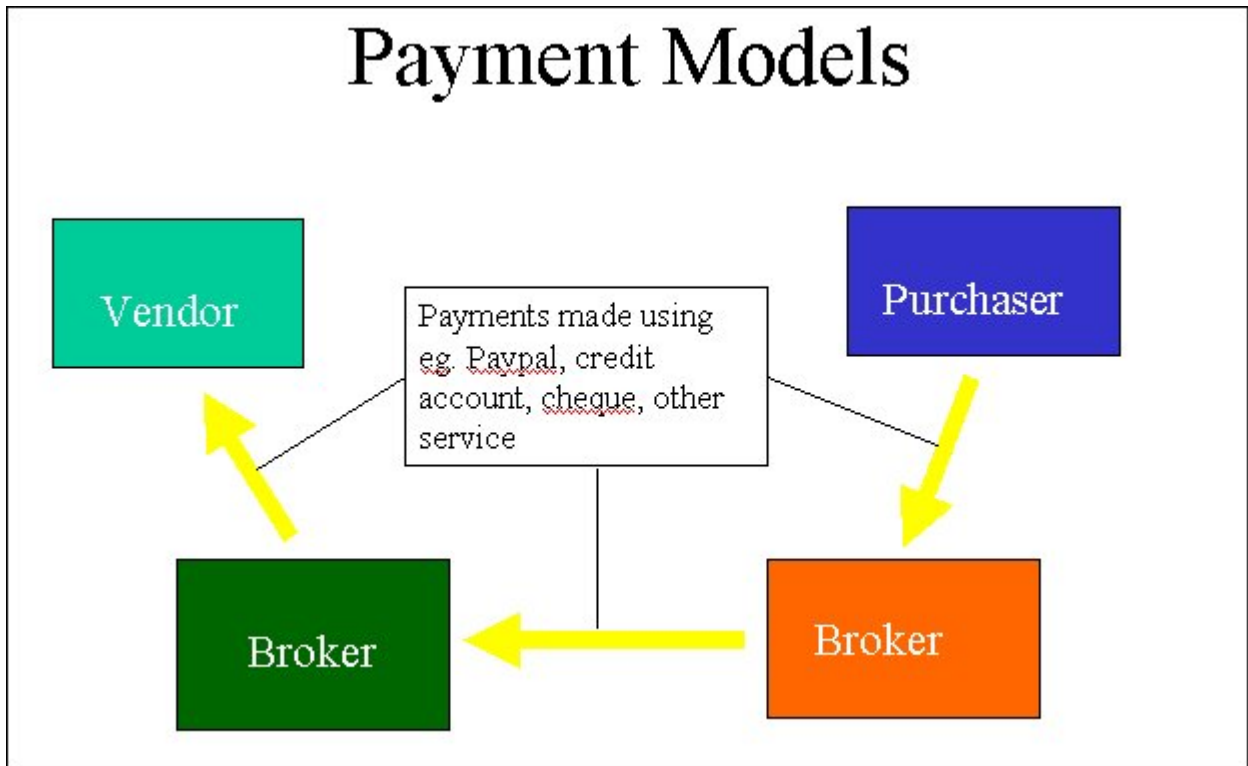


Figure 4. Distributed Digital Rights Management Model. Downes, 2004

The purpose of these features is to instantiate some of the requirements set out in the vision statement. In the vision statement, for example, it was proposed that eduSource be designed not as a single software application, but rather, as a set of related components. Thus, the various functions required of a digital rights system are displayed in the use cases as separate actors. This proposal was adopted by the eduSource Vision Committee in order to ensure that there are no sole-source components of the system. Any given function performed by eduSource may be provided by any number of providers.

In practical terms, what this means is that, in eduSource, there is not only one vendor broker; there can be many, each vendor broker representing one or more vendors. In a similar manner, there is not only one purchaser broker, there may be many. This allows both vendors and purchasers to choose the entity that will provide digital rights management services. No vendor can lock in a purchaser to a given rights management service, and no purchaser can require than a vendor employ a given rights management service.

The division of the eduSource model into separate actors also allows some actors to be bypassed if they are not needed. This performs a critical function for eduSource: it allows for the distribution of both free and commercial content in the same system. Because the digital rights management component, and in particular, the lock and key mechanism, is not an essential part of any eduSource service, it can be bypassed if not needed. Thus, even though free content is distributed through the same network as commercial content, it is not encumbered by the needs of a locking and payment system.

The addition of a purchaser broker into the system, in addition to protecting a purchaser's privacy and security, allows eduSource to "make it easier to buy than to steal." The price of learning resources, and particularly small items such as images, may be very low. Transactions involving such small amounts of money are called 'micropayments'. A major objection to micropayments is that the effort required to make a payment is greater than the payment is worth. There are financial transaction costs, and also what Szabi (1996) calls "mental transaction costs," the hesitation a user experiences when deciding to pay a minute amount for a learning resources.

Typically, the purchase of an inexpensive item occurs as a part of a purchase of a larger item. This practice, known as 'bundling', typifies most online content sales. Corbis, for example, sells not a single image but access to an image library. Elsevier sells access not to a single journal article but to a journal library. However, this approach creates barriers for both content providers and content consumers. Content providers must assemble and market bundles of content, usually through a publisher, before they can enter the marketplace. Moreover, free content is not bundled (since there is no need) and may be excluded from the set of available content. And users, when accessing content through a bundle, are able to search and use only resources provided by the vendor of a particular bundle.

The eduSource digital rights management system addresses these problems by bundling, not content, but financial transactions. Through the use of vendor and purchaser brokers, many small transactions from different vendors may be lumped into a single payment. A purchaser, therefore, may opt to use only one purchaser broker, making a single monthly payment, and even pre-authorize transactions under a certain amount. A vendor, in turn, may work with a single vendor broker, receiving (and managing) only a single monthly payment, no matter how many purchasers are supplied.



Figure 5. *Distributed Digital Rights Management Analogy*. Downes, 2004

Though new to online management of digital rights, the use of vendor and purchaser brokers is widespread in other commercial communities, as the analogy shown in figure 5 suggests. Vendors typically employ wholesalers to distribute their products. And purchasers typically access goods from a wide variety of marketers through their local store. It is rare, indeed, that a purchaser pays a provider directly for a good or service.

Finally, the use of a rights model, rather than an embedded description of rights, was necessitated by the commitment to a distributed system. Once metadata is released by content providers, it is beyond their reach. Thus, once an offer is made, through the provision of rights metadata, it cannot be rescinded or amended. This makes it difficult for vendors to adjust the prices of their products to react to changing consumer demand, timeliness of the information, or changing economic needs. By maintaining the rights metadata in a separate environment, one that is within the vendors control, the terms for the use of an object may be changed at any time up to the point of purchase. Additionally, the use of rights models allows one model to be applied to many objects, greatly simplifying the creation and maintenance of rights metadata for the vendor.

6. The eduSource Architecture

In order to enable a distributed network of object repositories involving many different search and distribution models, the eduSource architecture was designed around a set of commonly available web services. At the heart of these services is the Edusource Communications Layer (ECL). Instances of particular eduSource components, such as a

repository or search service, are expressed to the network as a whole through eduSource registries. Common tasks, such as providing object identifiers, are provided by the same means.

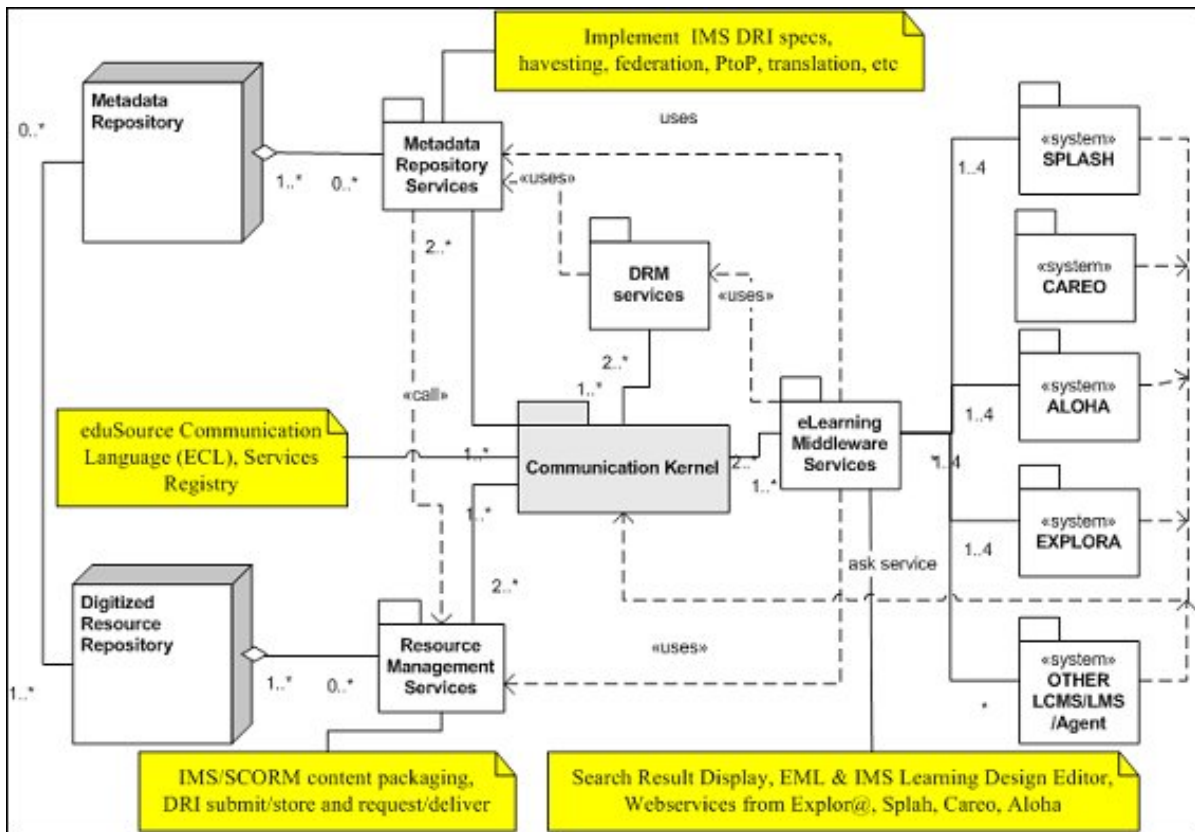


Figure 6. eduSource General Functional Diagram. Cogigraph Technologies, 2002

In the eduSource architecture, the digital rights management system is one of five major software packages; the others are the ECL communications kernel, e-learning middleware, metadata repository services, and resource management services. Since any function from any service must be available to all eduSource instances, communication and data transfer is handled through the use of web services. Thus the eduSource architecture committed the digital rights management system to providing a certain set of web services.

The eduSource architecture defines a 'broker' as "A software agent representing a person that wants to publish a new Learning Object to a metadata repository or to modify rights metadata of an existing LO. The Provider Broker presents a set of rights metadata models to the Provider. Each model includes secondary metadata that specify conditions, for example a certain form of payment that must be fulfilled in order to gain access to the object. The Provider selects a model and fill out specific conditions that are associated by the Provide Broker to the LO to be integrated by the Repository Builder."

In particular, a 'provider broker' is "A software agent representing a person that wants to publish a new Learning Object to a metadata repository or to modify rights metadata of an existing LO. The Provider Broker presents a set of rights metadata models to the Provider. Each model includes secondary metadata that specify conditions, for example a certain form of payment that must be fulfilled in order to gain access to the object. The Provider selects a model and fill out specific conditions that are associated by the Provide Broker to the LO to be integrated by the Repository Builder." And a 'purchaser broker' is "A software agent acting on behalf of a person that want to buy access to an object, obtains rights metadata and asks the purchaser (a utilizer) for payments prescribed in the rights metadata. It sends any required payment to an External Payment System," where an 'external payment system' is a "A computerized system that receives payment from a infoseeker or its Purchaser Broker in a DRM system. It informs the Provider of the learning object so that it can send a credential (a key) to the infoseeker." (Paquette, et.al., 2003a)

7. eduSource DRM Architecture

Based on the eduSource architecture and use cases, eduSource DRM functionality was expressed in greater detail through a set of DRM use cases. For example, figure 7 describes the requests that an LCMS must be able to make of the eduSource DRM system.

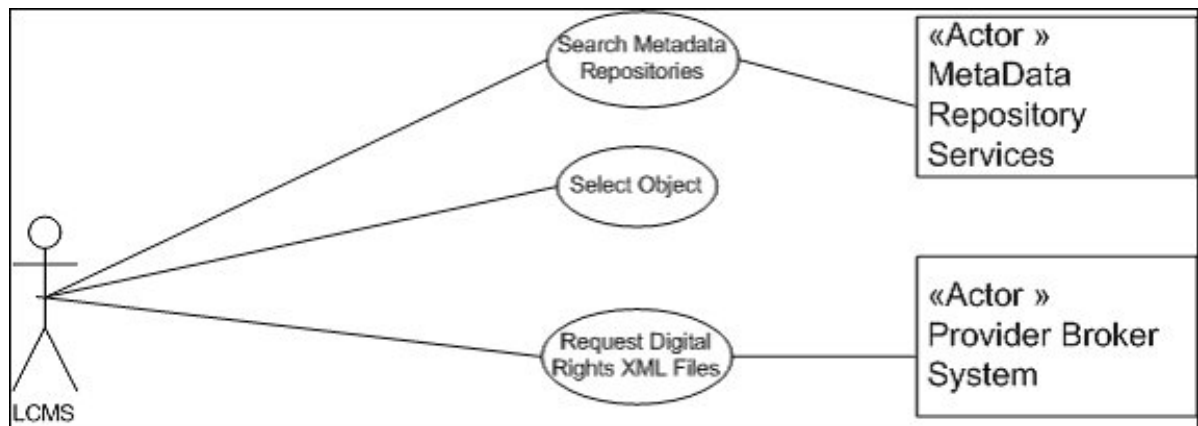
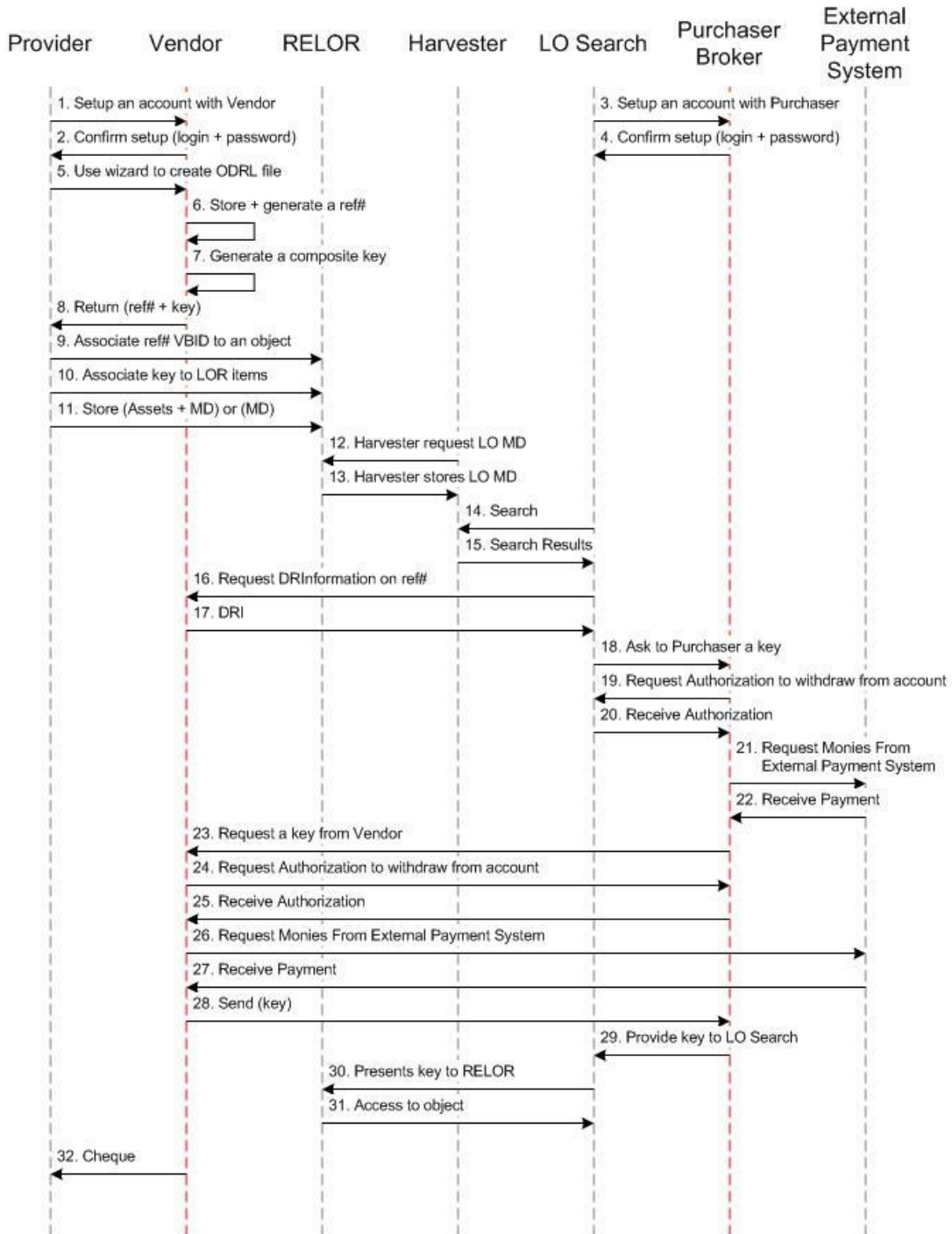


Figure 7. Request Digital Rights Information. Babin, et.al., 2003

Use cases were incorporated into the overall architecture of the DRM system. This architecture first captured in a sequence diagram to describe the steps of the DRM process (figure 8). An system data flow architecture was employed to specify more precisely the communications requirements between the various systems (figure 9). It describes the two brokers along with the end user, learning object repository and external broker system.

Digital Rights Management Sequence Diagram



Copyright NRC 2003

Figure 8. Digital Rights Management Sequence Diagram. Babin and Downes, 2003

The Sequence Diagram provides a preliminary understanding of how the digital rights will flow within a network.

1. A Learning Resource Provider sets up an account with a Vendor System.
2. The Vendor System generates a username and a password for each Learning Resource Provider.
3. After logging into the Vendor System, the Learning Resource Provider is presented with an ODRL Wizard which can be used to express rights in a XML format.
4. The generated ODRL File is stored on the Vendor Broker System and a REF# to the record is generated.
5. Based on the requirements expressed in the ODRL file a Composite Key is generated.
6. The REF# and in some cases a KEY are returned to the Learning Resource Provider.
7. The Learning Resource Provider then associates the REF# to an object in a Learning Object Repository.
8. The Learning Resource Provider also associates a KEY to an object in a Learning Object Repository.
9. The REF# is added to the LOM Metadata and the LOM is stored in the Learning Object Repository.
10. The metadata containing the REF# is harvested by some harvesters.
11. The harvester stores the metadata containing a pointer to the ODRL file in its database.
12. An object consumer searches for Learning Objects via a search agent.
13. The search results contain MetaData, of which one element is a pointer to the ODRL file
14. In some cases the Learning Object Consumer or his client software requests the Digital Rights information .
15. The Vendor Broker System sends out the ODRL file for reading.
16. If a key is required (that is, there is some cost or condition of access expressed in the ODRL) the Learning Object Consumer asks the Purchaser broker for a key.
17. The Purchaser Broker asks the vendor broker for the purchasing information and compares this to the consumer's profile file.
18. The Purchaser Broker receives the information and acts accordingly, either acting automatically or requesting confirmation from the consumer.
19. If necessary money will be requested from an external payment system.
20. The Purchaser Broker receives the money. Because of the small amounts the Purchaser Broker may deduct from an account rather than conducting a transaction with an external system every time a learning object is requested.
21. The Purchaser Broker then contacts the Vendor Broker System to buy the key.
22. The Vendor Broker request permission to withdraw from the purchaser broker's account.
23. The Vendor Broker receives OK to withdraw
24. The Vendor Broker requests money from external system. Again this may be done on an account basis if transaction costs exceed object costs.

25. The Vendor Broker receives money from the external system.
26. The Vendor Broker sends the key to the Purchaser Broker.
27. The key is forwarded to the object consumer (the one who did the search)
28. The object consumer presents a request for an object + the proper key to the LOR
29. The LOR validates the key and returns the object

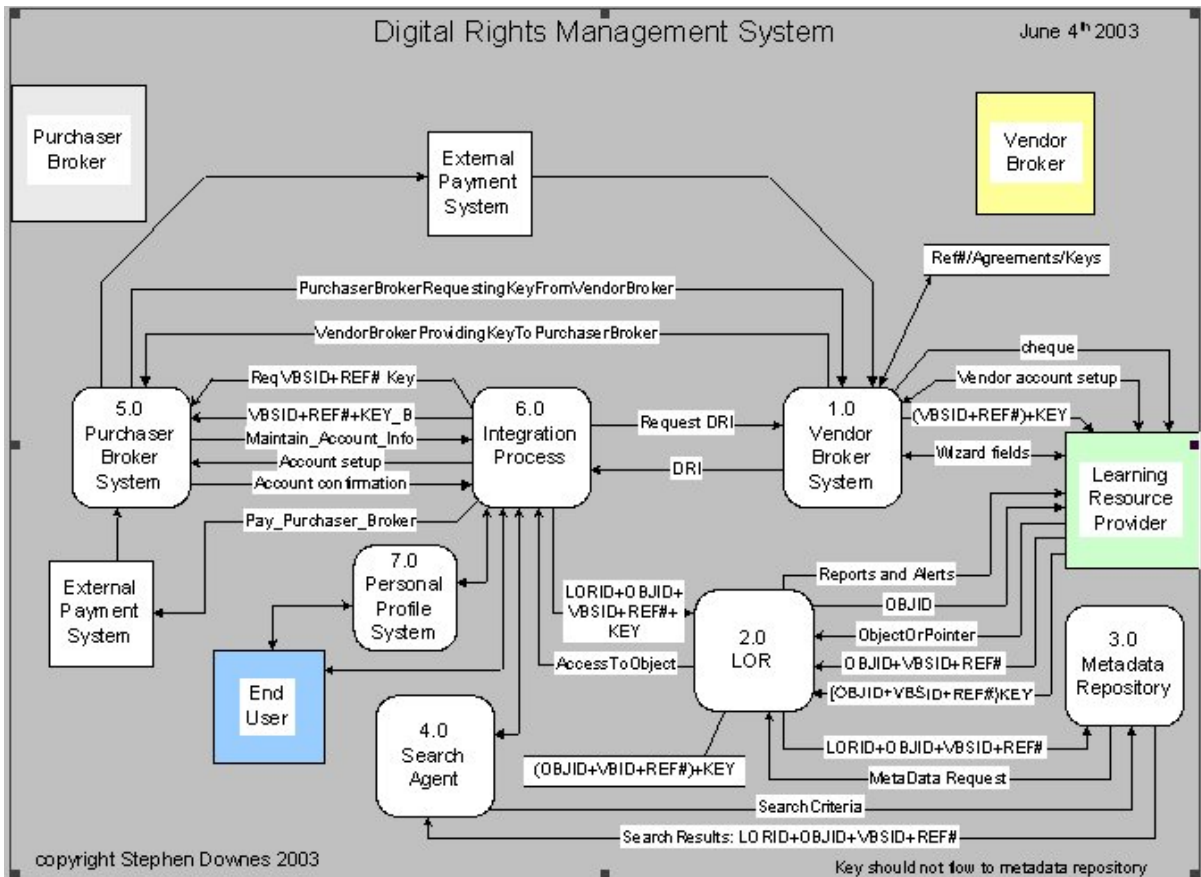


Figure 9. System Data Flow Architecture Diagram. Babin and Downes, 2003

The DRM System works in conjunction with other Systems such Search Agents, LORs, Harvesters, External Payment Systems. Keep in mind that this system is designed for a near future where there will be many low cost or free objects available with most transactions happening transparently at a machine level. A learning resource provider (LRP) wishing to sell objects (LO) sets up an account on the Vendor System. The Learning Resource Provider uses a wizard to create an ODRL XML rights description file which is going to be stored on the Vendor System. The ODRL file is parsed and a key token is created for every ODRL item requiring a key. The key tokens are aggregated into a composite key. The ODRL file and key are stored the Vendor System Database generating a REF# id. The Learning Resource Provider receives the REF# and the Composite Key.

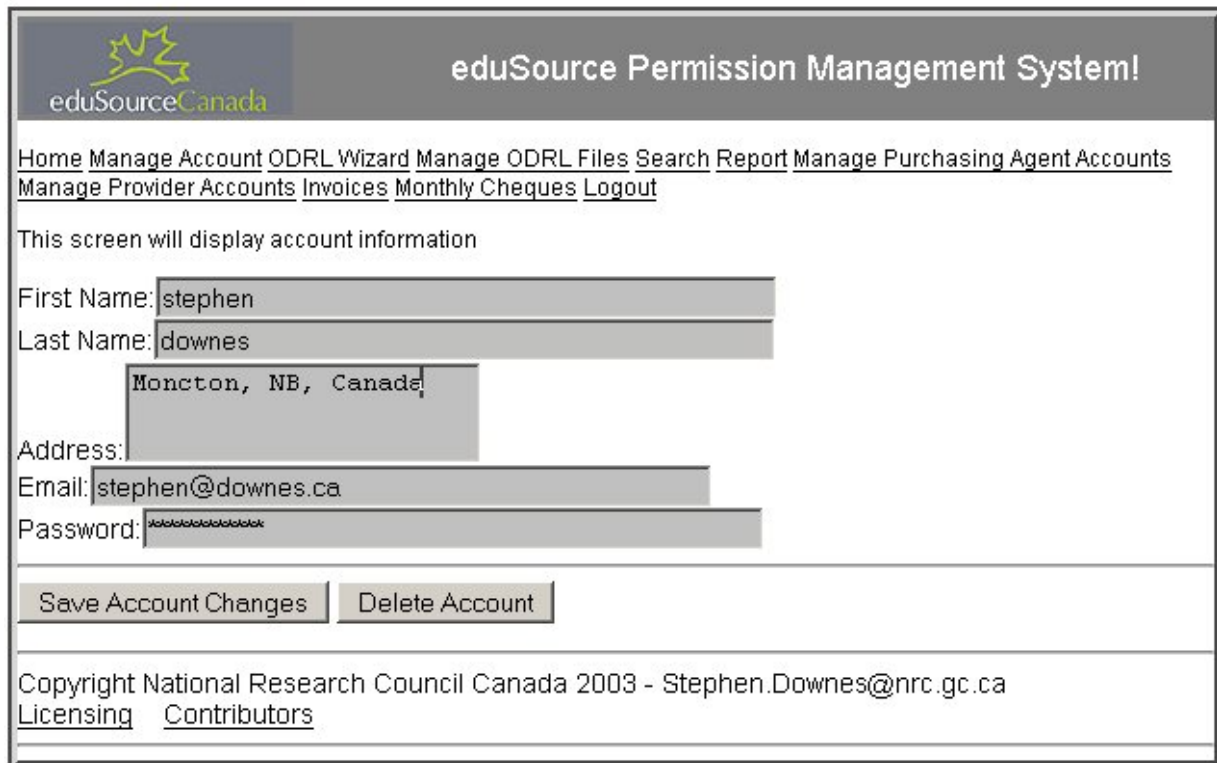
To transform a LO into a Rights Enabled Learning Object (RELO) the Learning Resource Provider will associate the REF#/Composite Key to one or more of the

Learning Object Repository (LOR) Learning Objects. The Harvester plays an important role in this model. As LOR get harvested the REF# become visible to end users thereby making it possible to access the ODRL files from the Vendor Broker System. A Rights Enabled Learning Object Repository (RELO) will be able to process keys and release info upon presentation of a proper key. The REF# will be exposed to harvesters but the composite key will not.

8. Provider Broker Web Interface

The Provider Broker provides a web interface where resource vendors may manage their account. The Provider Broker demonstration is available at <http://drm.elg.ca/ProviderBrokerSystem/ProviderBrokerSystemLRP>

The following figures demonstrate Provider Broker functionality. Figure 10 displays the account management screen and the list of functions available in the eduSource Provider Broker: Manage Account, ODRL Wizard, Manage ODRL Files, Search Report, Manage Purchasing Agent Accounts, Manage Provider Accounts, Invoices, and Monthly Cheques.

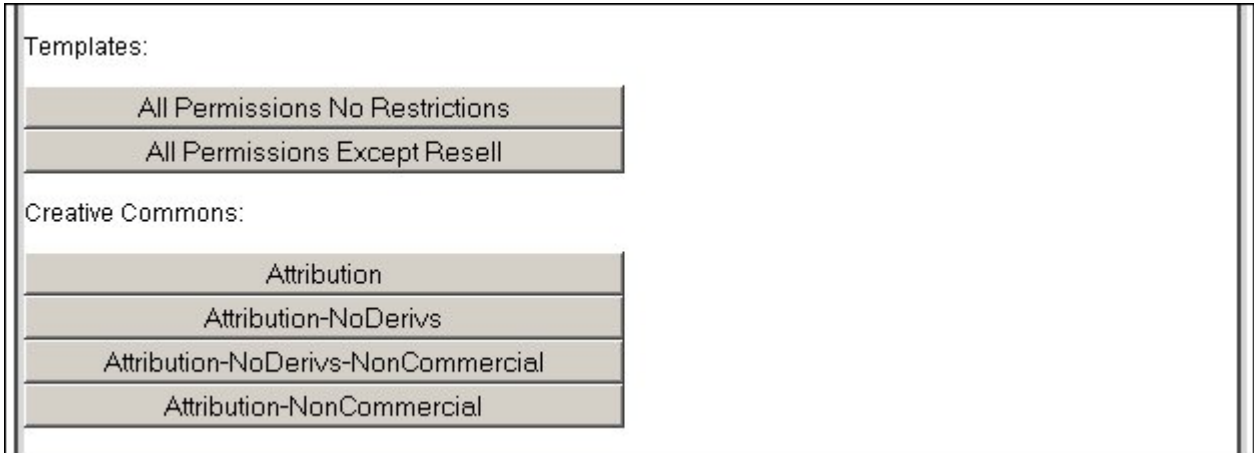


The screenshot shows the 'eduSource Permission Management System!' interface. At the top left is the 'eduSourceCanada' logo. A navigation menu includes: Home, Manage Account, ODRL Wizard, Manage ODRL Files, Search Report, Manage Purchasing Agent Accounts, Manage Provider Accounts, Invoices, Monthly Cheques, and Logout. The main content area displays account information for 'stephen downes' from 'Moncton, NB, Canada'. Fields for Address, Email (stephen@downes.ca), and Password are also visible. At the bottom are 'Save Account Changes' and 'Delete Account' buttons. The footer contains copyright information for the National Research Council Canada 2003 and links for Licensing and Contributors.

Figure 10. eduSource Provider Broker Manage Accounts.

In order to create an ODRL model, a provider accesses the ODRL wizard. This program is available at <http://drm.elg.ca/english/ODRLGenerator>

The ODRL wizard allows the user to create a rights model automatically, by selecting one of several preset options, as demonstrated in Figure 11.



Templates:

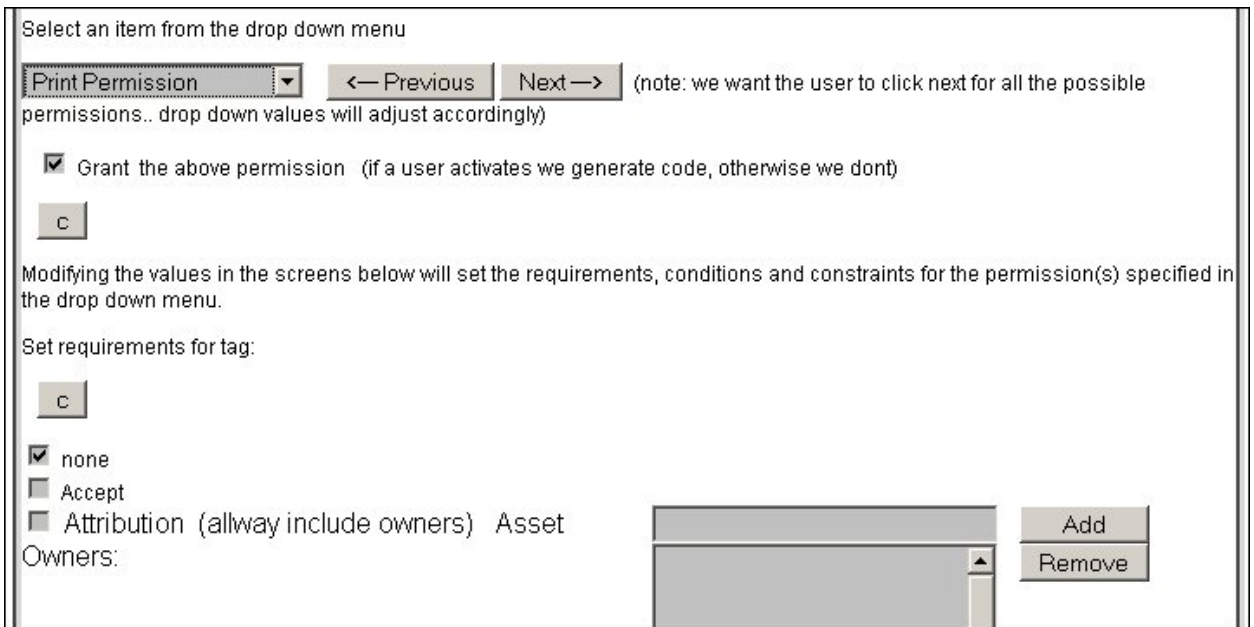
- All Permissions No Restrictions
- All Permissions Except Resell

Creative Commons:

- Attribution
- Attribution-NoDerivs
- Attribution-NoDerivs-NonCommercial
- Attribution-NonCommercial

Figure 11. eduSource Provider Broker ODRL Wizard Preset Options.

Additionally, the wizard allows a vendor to customize the present options or generate a new offer or agreement from scratch by selecting from the web based form. A partial screen shot is shown in figure 12.



Select an item from the drop down menu

Print Permission (note: we want the user to click next for all the possible permissions.. drop down values will adjust accordingly)

Grant the above permission (if a user activates we generate code, otherwise we dont)

Modifying the values in the screens below will set the requirements, conditions and constraints for the permission(s) specified in the drop down menu.

Set requirements for tag:

none
 Accept
 Attribution (allway include owners) Asset

Owners:

Figure 12. eduSource Provider Broker ODRL Wizard Modify Options.

When the user desires, the Wizard generates ODRL for the current set of options. This XML listing may then be edited at the source level, if desired (note: this is generally not recommended). The output display is demonstrated in figure 13.

```

Total XML for all pages
[Generate ODRL for all pages]

<?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:offer>
    <o-ex:permission>
      <o-ex:requirement>
        <o-dd:attribution/>
      </o-ex:requirement>
    </o-ex:permission>
  </o-ex:offer>
</o-ex:rights>

```

Figure 13. eduSource Provider Broker ODRL Wizard Generated ODRL Markup.

When the ODRL XML listing has been generated, the vendor may now save the file as an ODRL model by giving the file a name and selecting key options. The model may also be displayed in human readable form (English, French and Spanish output is currently supported). Figure 14 demonstrates the creation of an ODRL model. Subsequent to creation, a vendor may edit an ODRL file by selecting the rights model from a list provided on the 'Manage ODRL Files' screen. A sample ODRL file generated by the system may be found at <http://drm.elg.ca/GetODRL?id=38>

After Generating the ODRL for all pages, enter a filename for later reference.

What type of key is required:

Filename:

Validate against ODRL xsd documents:

Human-Readable ODRL language:

stephen:6

Copyright National Research Council Canada 2003 - Stephen.Downes@nrc.gc.ca [Licensing](#) [Contributors](#)

Figure 14. eduSource Provider Broker ODRL Wizard Create ODRL Model.

9. Provider Broker - Tagger Interaction

A 'tagger' is a tool used by learning object authors to create metadata for a learning object. It may be a stand-alone tool, or it may be incorporated as a part of a more comprehensive authoring tool. The eduSource Repository in a Box includes a tagger as part of the software set available to users.

Once a vendor has created an account with a Provider Broker, the functionality of the Provider Broker may be accessed from within the tagging tool through the use of web services. Figure 15 is a mock-up of what such an interface would look like. The list of ODRL Models available is shown in the drop-down. The tagger obtains this list from the Provider Broker using a web service.



Figure 15. eduSource Provider Broker Tagger Interface.


When the rights model is selected, a second web service is called by the tagger, and the Provider Broker returns the address of the ODRL model. The tagger then inserts this address into the rights description field of the Learning Object Metadata (or the appropriate field if a different XML format is being used). The resulting rights XML is as follows:



Figure 16. ODRL Model Reference in Learning Object Metadata.

10. Purchaser Broker

As described above, a purchaser broker acts as an agent for a content purchaser (or 'infoseeker'). To use the Purchaser Broker, the infoseeker creates a purchaser broker account. Included as part of the account, as depicted in Figure 17, are conditions for pre-authorized purchases. As noted, the creation of pre-authorized purchases eliminates the mental transaction costs associated with micropayments.

 **eduSource Digital Rights Purchaser Broker System**

[Home](#) [Manage Account](#) [Credit Card Info](#) [Preferred Payment Method](#) [View History](#) [Logout](#)

The following information will be kept confidential.

First Name:

Last Name:

Address:

Email:

Password:

Conditions

Pre-authorize individuals purchases up to:

Amount purchased to date: null

Maximum for all purchases:

Figure 17. eduSource Purchaser Broker Manage Account.

Though services offered from one Purchaser Broker to the next may vary, the idea is that a purchaser may employ any number of payment methods, including monthly invoice, addition to an Internet Service Provider billing, credit card payment, or online payment service such as PayPal. Figure 18 displays this selection in the Purchaser Broker Account Manager.

eduSourceCanada eduSource Digital Rights Purchaser Broker System

[Home](#) [Manage Account](#) [Credit Card Info](#) [Preferred Payment Method](#) [View History](#) [Logout](#)

stephen:5

Please select a Payment Option and fill in appropriate information.

Record just created. If you do not see Payment Options, Click on Preferred Payment Method again. Will fix bug later.

Payment Option 0: Not ready to select a payment mode
 Payment Option 1: Send a Monthly Invoice
 Please provide a Running PO number if required:
 Billing Address:

Payment Option 2: Bill my Credit Card on a monthly basis.
 Please ensure that you have properly setup your credit card information.

Payment Option 3: Bill my Credit Card on a continual basis.
 Please ensure that you have properly setup your credit card information.

Payment Option 4: I wish to pay via a generated PayPal PayNow page.

Figure 18. eduSource Purchaser Broker Manage Account.

11. Learning Object Browser

To demonstrate the functionality of the eduSource DRM system, a learning object browser (LOB) has been created. The LOB conducts a search across the eduSource network and displays the search results. <http://drm.elg.ca/ObjectBrowser/ObjectBrowser>

The LOB provides a user with access to Purchaser Broker functions. Figure 19 displays the LOB search form, with a choice of Purchaser Brokers displayed (recall that a user may opt to use one or more purchaser brokers). By selecting a purchaser broker, the user determines which of these services will conduct transactions on his or her behalf.



Figure 19. Learning Object Browser Purchaser Broker Select.

12. LOB - Vendor Broker Interaction

When search results are returned and displayed to an infoseeker, client software should also retrieve rights information automatically (using the URL located in of the metadata to retrieve the ODRL rights file from the rights broker). Displays may vary, of course, but an infoseeker would not typically click 'rights' -- they would select an action (view, print, etc). *If* rights clearance is required in order to perform the action, then the rights subroutines take effect; if no rights clearance is required, then the action simply happens.

For example, a person types in a search request that is sent to eduSource: 'Roman History' eduSource returns a set of LOM records. In each LOM record is a reference to an ODRL file. The person's client requests each ODRL file. This information is now displayed together.

For example, imagine the possible set of search results:

History of the Roman Empire
 Fred's Roman History \$
 New Edited Roman History
 All the Romans in the World \$\$

No Romans No More \$
Rome Are Us

We can see from this display that some resources have a cost, and others are free. You do not need to click on anything to see this.

If a person clicks on the title of a free resource, it simply displays (that is, a request is sent directly to the provider's learning object repository, the resource is returned, and then displayed to the viewer).

If a person clicks on the title of a pay resource (indicated with \$), then the request is sent instead to the purchaser broker. The purchaser broker retrieves the ODRL file from the vendor. In some cases, the payment is preapproved, so it simply conducts the transaction and sends a key to the users client, which then presents the key along with the request to the provider's learning object repository. In other cases, it must ask the user to select from a set of one or more options (offers) to approve (or reject) payment. If payment is rejected, the transaction terminates. If payment is made, then the transaction is conducted, a key obtained, and sent to the client program, which makes the request from the learning object repository.

We do not display ODRL information (except in rare cases). We use ODRL information to make decisions.

13. DRM Security Model

Digital rights management has three major aspects:

- Expression — the description of the resource, ownership of the resource, and the terms and conditions of use
- Authentication — verification that the person using the resource has the right to use the resource, and
- Protection — means, such as encryption, to ensure only authorized users have access

In addition, DRM may be applied in any of three domains:

- Resource — a particular document or digital resource — for example, a document may be locked or encrypted
- Access Point — a content server, such as a website — for example, a website may require a login
- Network — the connections between servers — for example, ATM network

This creates a DRM design decision metric, as displayed in figure 20.

	Resource	Access	Network
Expression	Copyright notice	Terms of use notice	Rights expression language
Authentication	Password to open document	Password to access website	PIN to use ATM system
Protection	Encrypted document	Secure sockets layer	Virtual private network (VPN)

Figure 20. DRM Design Decision Metric.

In the decision metric, it is possible to identify increasing degrees of security, as demonstrated in Figure 21.

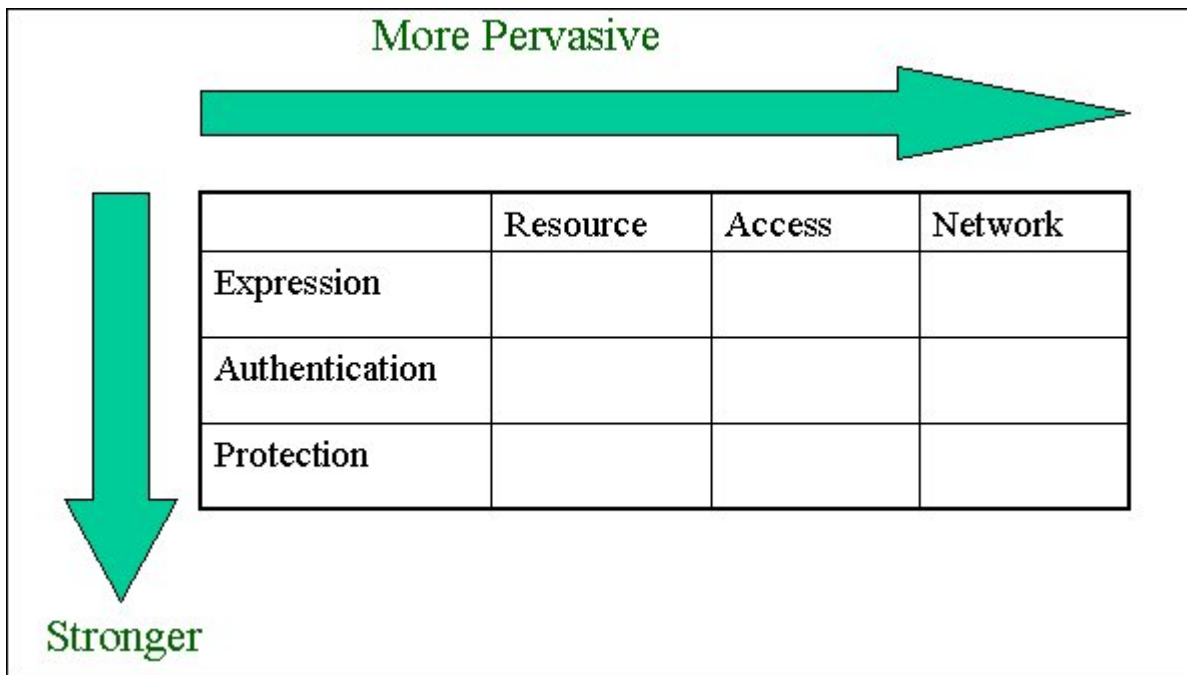


Figure 21. DRM Degrees of Security.

We may therefore distinguish between: *weak DRM*, where expression is in the resource only, there is no authentication and no protection, as in a web page with a copyright notice, book with a copyright page, property with a "keep out" sign; and *strong DRM*,

where expression is in the resource, access point, or network, authentication is in the network using a single login, and protection is network wide, as in the ATM Bank Machine system requires that you provide credentials to use the system, and encrypts all data and communication.

In the debates regarding DRM, two major positions have evolved, corresponding to these degrees of security:

1. *DRM is too weak*: in networks like the web and Napster, expression alone is insufficient to ensure that rights are respected
2. *DRM is too strong*: proposed DRM systems require a unique userid (eg., MS Passport) and fully secured network (eg., Rights management server, “trusted” applications), violate privacy, fair use

The DRM mechanism proposed by eduSource DRM is a 'middle way' between these two extremes. Expression is supported at the network level through the use of a rights expression language (and specifically, ODRL). Authentication is supported at the access level through the use of keys. And protection is supported at the document level with locks or encryption.

Criticism regarding the proposed system has, not surprisingly, originated from both extremes. From one point of view, the eduSource DRM system is too strong. Advocates of open content, for example, fear any DRM system will prevent people from freely sharing content. However, it is arguable weak enough. In order to use free resources, rights must be declared, and any further level of authentication and protection is at the discretion of the resource owner. On the other hand, others have argued that the eduSource DRM system is too weak. Commercial providers, for example, want stronger protection, such as authentication at the network level, to prevent file sharing. But in response, it may be argued that it's strong enough. A key system makes it difficult to obtain unauthorized access to content, but leaves it easier to buy content than to steal it.

Critics of eduSource DRM must ask themselves, "What causes file sharing?" There are two answers. When DRM is too weak, there is no incentive to go through the extra work and cost to pay for content; commercial content is not viable. But when DRM is too strong, free content is not viable, and the transaction cost is too high, so it is easier to look elsewhere for the same content.

References

Babin, et.al., 2003. EduSource DRM Statement of Requirement. Version 2.0 Final. Gilbert Babin, Stephen Downes and Luc Belliveau. May 25, 2003. eduSource, <http://www.edusource.ca>

Cogigraph Technologies, 2002. Software Development and Integration Work Package Plan. Version 0.3. eduSource. <http://www.edusource.ca>

Downes, et.al., 2002. EduSource Vision. Version 3. Stephen Downes, Toni Roberts, Rory McGreal, Norm Friesen, John King, Terry Anderson, Michael Magee, Mike Mattson, Gilbert Paquette, Griff Richards. eduSource. <http://www.downes.ca/files/vision3.doc>

Downes, 2004. Distributed Digital Rights Management. Presentation to TeleEducation Online Forum, January 27, 2004. Power Point Slides. <http://www.downes.ca/files/ddrm.ppt>

Ianella, 2002. Open Digital Rights Language (ODRL) Version 1.1 W3C Note 19 September 2002. Renato Ianella. <http://www.w3.org/TR/odrl/>

LGPL, 1999. GNU Lesser General Public License. Version 2.1, February 1999. Free Software Foundation, Inc. <http://www.gnu.org/copyleft/lesser.html>

McGreal, et.al., 2003. eduSource: Creating learning object repositories in Canada. Rory McGreal, Griff Richards, Norm Friesen, Gilbert Paquette and Stephen Downes. Learning Technology, Volume 5, Issue 1. IEEE Computer Society Learning Technology Task Force. http://lutf.ieee.org/learn_tech/issues/january2003/#1

Paquette, et.al., 2003. eduSource Suite of Tools Use Cases Specifications Version 0.6. Gilbert Paquette, Anis Masmoudi, Gérard Lévy, Terry Anderson, Chris Hubrick, Stephen Downes, Gilbert Babin, Mike Matson, Marek Hatala and Griff Richards. June 7, 2003. eduSource. <http://www.edusource.ca>

Paquette, et.al., 2003. eduSource Suite of Tools Glossary of Terms Version 1.0. Gilbert Paquette, Karin Lundgren-Cayrol, Gérard Levy and Stephen Downes. eduSource. September 29, 2003. <http://www.edusource.ca>

Szabo, 1996. The Mental Accounting Barrier to Micropayments. Unpublished. <http://szabo.best.vwh.net/micropayments.html>

Towards a Formal Semantics for ODRL

(Extended Abstract)

Markus Holzer, Stefan Katzenbeisser, Christian Schallhart

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

April 17, 2004

Abstract

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

1 Introduction

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

In the past few years various techniques for preventing copying or restricting the access to copyrighted material (called *copy protection* mechanisms) were implemented. Examples of copy protection include encrypted

digital TV broadcast (conditional access systems), access controls to copyrighted software through the use of license servers and technical copy protection mechanisms on the media (like the content management mechanism on DVDs). A copy protection scheme is a special flavor of a *digital rights management system*, which attempts to enable secure distribution of copyrighted content via open networks.

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

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

2 A fragment of ODRL

ODRL is an extremely rich language that contains many different operations, requirements and constraints. For the sake of simplicity, we deal only with a fragment of ODRL in this paper. In fact, we are merely concerned with modeling offers that are not bound to a particular person. However,

extensions of our model to the complete ODRL language (except descriptive elements like the ODRL context, which have no operational semantics) are possible.

In this work, we concentrate on the following permissions:

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

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

```

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

```

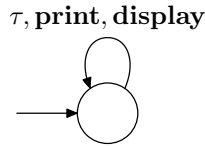
Example 7 *A user is allowed to display and print the object; furthermore, he can lend it to a different user, who has the same permissions (i.e., display, print and lend).*

3 A Semantics based on Automata

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

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

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



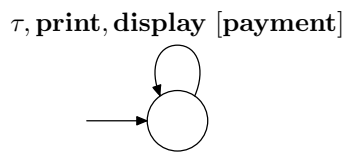
Here, the automaton has only one state and a self-loop. Printing and displaying the document once does not affect the state of the license; therefore, we only need one automaton state to code the permission. It is thus allowed to print and display an object an arbitrary number of times (by our convention,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

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

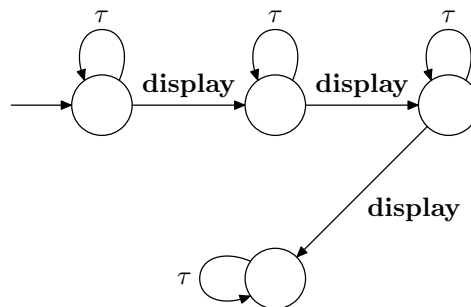
The labeling function changes with each tick. Initially, only the initial state of the automaton is labeled with $\{1\}$, each other state with \emptyset . Whenever the user (encoded with the symbol “1”) performs an action, the

labeling function changes. In particular, the label $\{1\}$ is erased from the currently active state (it is replaced by \emptyset) and applied to the state that can be reached by the edge representing the action. This way we get, for each possible sequence of transitions, an infinite sequence of labeling functions f_1, f_2, \dots representing the “status” of the license at each point in time.

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

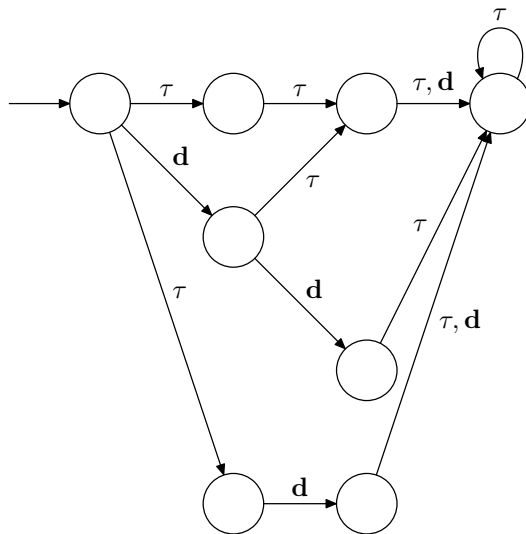


Let us turn to example 2; here, the automaton is more complex, as it requires to count the number of times an object was displayed “in the past.” This can be done by using four different states:



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

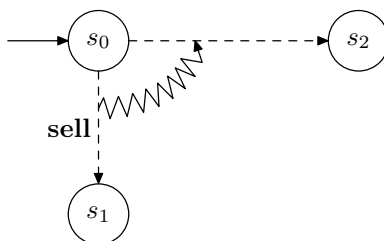
Modeling of time constraints may require to introduce a large number of different states; for example, the following automaton represents Example 3; for space reasons we abbreviate **display** by **d**:



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

3.1 Extending the model to support SELL and GIVE

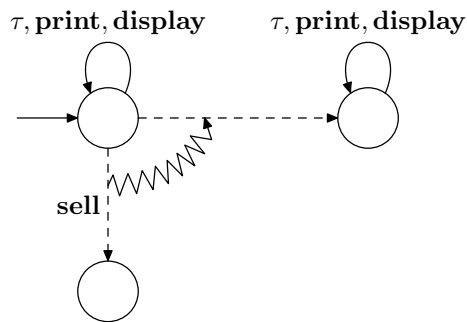
So far, we only had to consider one person that is active in the automaton. This changes if we model the **sell** and **give** permissions. Once a person (say, 1) performs a **sell** action, this person loses all rights on the object; the rights are transferred to a *new* person (say, 2). Once person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



Consider the last figure. The trigger and the dashed transitions have the following intuitive meaning: Suppose that state s_0 is labeled with $\{1\}$, i.e., person 1 is active. If person 1 performs a **sell** action, the label $\{1\}$ disappears

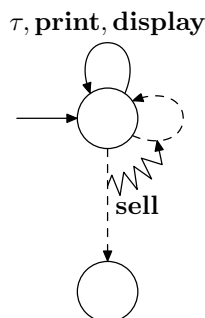
in the next tick and is replaced with \emptyset (you can think of 1 moving into the dead-end state s_1 , where he is removed, yielding to label \emptyset for s_1). In the same tick, a different person (say, 2) appears in state s_2 , yielding to the label $\{2\}$ for s_2 . That is, dashed transitions triggered by other transitions introduce a new person, whereas dashed transitions that are not triggered remove a person.

Using this convention, we can model Example 4 as:



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

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

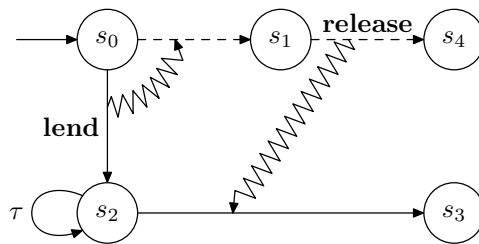


Note that, up to now, the labels of each state contain *at most* the identity of one person.

3.2 Extending the model to support LEND and LEASE

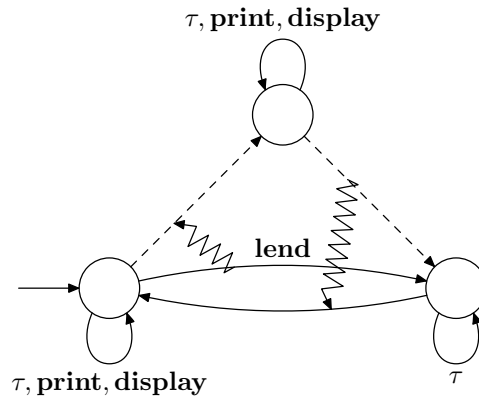
It turns out that the **lend** and **lease** permissions are more complex to handle semantically; in our semantics, both **lend** and **lease** are modeled in the same manner. In contrast to **sell**, where the identity of a person is removed from the labels after a **sell** action has occurred, this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

We model this situation with two triggers:



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

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



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

4 Conclusion

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

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

References

- [1] E. Clarke, O. Grumberg, D. Peled, *Model Checking*, MIT Press, 1999.
- [2] R. Ianello, *Open Digital Rights Language (ODRL)*, Specification, Version 1.1, available at www.odrl.net, 2002.
- [3] R. Pucella and V. Weissman, "A Logic For Reasoning about Digital Rights", Proceedings of the Computer Security Foundations Workshop, 2002.

- [4] R. Pucella, V. Weissman, “A Formal Foundation for ODRL”, in Workshop on Issues in the Theory of Security (WITS), 2004.
- [5] *eXtensible rights Markup Language (XrML) 2.0*, Specification, available at www.xrml.org, 2001.

Nonius: Implementing a DRM Extension to an XML Browser

Olli Pitkänen*, Ville Saarinen, Jari Anttila, Petri Lauronen, Mikko Välimäki

Helsinki Institute for Information Technology HIIT
P.O. Box 9800, 02015 HUT, Finland
<http://www.hiit.fi/>

Abstract

The paper describes experiences, ideas, and problems that were discovered while developing a digital rights management (DRM) extension to an XML browser. The supported rights description language is ODRL. The most significant implemented features are restrictions related to an individual, time, and usage-counts. On the other hand, some interesting features were intentionally left out. They include for example, aspect and target constraints as well as many security features. The most difficult tasks in implementing a DRM system are related to security and parsers. A secure DRM system requires the support of hardware devices. A DRM document parser depends profoundly on a flexible software architecture. Merging certificates and implementing an interface for creating certificates are also demanding. The three most challenging features in ODRL specification are logical operators, documents' internal links to its elements, and the requirement that child elements may depend on ancestor elements' children. General technical problems that are discussed in the article, but largely left unanswered, include how we can make secure software in open source model, on which layer DRM should be supported, and how secure the system should be.

Keywords

Digital rights management (DRM), open source implementation, XML browser, rights description languages, Open Digital Rights Language (ODRL)

1 Introduction

1.1 Background

The concept of digital rights management (DRM) is very timely, but also ambiguous. It is a kind of buzz word and often includes a lot of hype. It can refer to rather broad set of actions, procedures, policies, product properties, and tools that an entity uses to manage its rights in digital information. Actually, the term "digital rights management" is somewhat misleading. Rights are not digital. In general, they do not have much to do with digits, but they are rather analog. The word "digital" refers supposedly to the subject matter, to information in digital form, not to rights in that information. It is also possible to think that the word "digital" refers to the fact that digital information technology is often used to manage the rights. Yet, DRM does not refer to computer-aided rights management in general. [10], [16]

In this article, DRM is used in a quite narrow sense. It refers to technologies that control copyright in information products in a digital environment. It usually requires that the content is encrypted and can be safely distributed. Decrypting and using the product, then again, is allowed only if the user has a suitable certificate for the product. The certificate gives the right to use the content. It can include various restrictions. DRM enables new ways to publish information. For example, a user can buy instances of use or a limited usage-time for a product. In general, such restrictions are not feasible in the material world.

* e-mail: olli.pitkanen@hiit.fi

Helsinki Institute for Information Technology (HIIT) had MobileIPR research project that studies problems related to the rights management of information products on the mobile Internet from legal, technical and economic perspectives. The project started in 2000 and ended in the beginning of 2004. [16]

In connection with MobileIPR, a student group called *Nonius* made a DRM extension to X-Smiles XML-browser. This article contains some ideas and describes experiences and difficult areas that appeared in the Nonius-project. Of the article's authors, Olli Pitkänen is the project manager of MobileIPR and he supervised Nonius team with help of his colleague Mikko Välimäki. Ville Saarinen, Jari Anttila, and Petri Lauronen were members of Nonius team – the other members were Jani Poikela, Kalle Anttila, Arto Jalkanen, and Jarno Salimäki. The students accomplished the project as a part of T-76.115 *Software Project* class at Helsinki University of Technology. [14]

1.2 Digital Rights Management Systems

The intention of a DRM system, as defined above, is to protect digital content so that it cannot be used without a valid certificate. It separates information from the right to access it. The certificate can be sent to the users either in connection with the content or separately.

This kind of a DRM system requires strong encryption. The certificate includes a secret key needed to decrypt the content data. The encryption algorithm must be strong enough so that it is not broken easily. The data encryption, however, is not the weakest link of a certificate-based DRM system. The problem is that in order to be viewed, the encrypted data has to be decrypted by the legitimate customer. In an open environment, like a standard personal computer, users are able to do anything they wish with the decrypted data, including copying and further distributing it. This problem must be dealt with by applying regulations in either software or hardware. [18]

The easiest way to control the copying of decrypted data is to make a special browser or player for the content, so that it will decrypt and play the data on an analog device like a monitor or a loudspeaker system, but it will not output the decrypted data in a digital form. Analog outputs are not that big a problem, since analog copy is never the same quality as the original, unlike the perfect digital copies. The special software browser is currently the most commonly used DRM implementation. [4], [13]

Software regulations are usually easy to circumvent. By special debugger and disassembler software, crackers can monitor the execution of a program and get their hands on the decrypted content. A hardware implementation of the copy prevention mechanism is a lot harder to break. A special hardware device that decrypts the data and gives it out only through an analog output is very hard to break since the decrypted digital content exists only inside the device.

A hardware copy prevention system does not necessary have to be a separate player device. Some computer industry companies, like Intel and IBM, have made a proposal to include special digital rights management schemes in computer hard drives and operating systems that would make it possible to mark a file as copyrighted, and all copy operations for the file would then be denied by the operating system. This proposal is known as Content Protection for Recordable Media (CPRM). This kind of arrangement is problematic since all hard drive manufacturers and all operating system manufacturers would have to co-operate in order to effectively force the restrictions. Making all major manufacturers work together is a task close to impossible, especially since it would require a lot of new technology to be developed and the hardware manufacturers do not really have much to gain in the decreased piracy problem. Open-source operating systems are also a problem. If the users have the source code, they can modify it and bypass the restrictions. [5]

There is also a proposal for applying the digital rights management scheme closer to the user, in the output devices which make the actual conversion of digital data to some analog signal like sound or picture (monitors, loudspeakers, and other similar devices). Since the encrypted content data would be decrypted for example inside a monitor, there is no way a user could get his hands on the decrypted data with software. Only minimum co-operation would be required from the operating system, but there are still some disadvantages. If the content data is in encrypted form all the way to the output device, only an output device that supports the technology can show it. This would require all end-users to update their hardware, which is not an easy process. The new hardware will be a noteworthy cost to customers, and a lot of opposition would be encountered. Even at best, it will take years before a significant number of potential buyers have the hardware needed to use the protected content.

The digital rights management for output devices can also be done in a more backwards-compatible way. The data itself does not have to be encrypted, the content producer only needs to mark it copyrighted through a watermarking scheme like the SDMI (Secure Digital Music Initiative) watermarking. All compliant output devices would then detect the watermark and apply whatever restrictions the content producer wants for the use of the material. Using this kind of approach, people would still be able to view the content without authorization using old output devices, but since any digital device gets old pretty quick the older devices would be pretty much history in five or ten years after all new devices started to include the required restrictions. The problem is that all new devices would have to include the digital rights management features in order to make them actually work, and it is not easy to include a restriction in all output devices by different manufacturers. There is certainly consumer demand for a player without the restrictions. So unless it is illegal, some manufacturer will surely create a player without the limits and make good profits.

Creating a completely unbreakable copy prevention system is impossible. The user, or at least some component in the user's hardware, must have a proper key to decrypt the digital content in order to actually view it. And if digital data can be read and viewed, it can also be copied. If the security measures are done in software, it is easy for an experienced hacker to break the software and get the decrypted content data. And even if the security measures are done by hardware, some hacker or a professional pirate will have enough resources to analyze how the hardware works and modify it to bypass the security measures. But the professional pirates are not such a big problem as long as the average user pays for the product.

Besides making copying harder, a good DRM system has also other functions that create possibilities for completely new business models. For example, a DRM certificate may limit the user's right to view the document so that he can only view it one or two times. This is very practical for giving out free samples of the product – user can see what the content is like but if he wants to continue using it he has to buy a new certificate. Other limitations that are good for free samples are limited period of time (for example the product can only be accessed during May 2004) or limited cumulative usage time (the user can view the content for one hour). The DRM system may also limit the usage by user, hardware, physical location or many other rules. [4], [13]

1.3 Rights Description Languages

The technical implementation of a DRM system requires a language to describe the rights granted to users. There are many competing rights description language standards. We selected three of them for a closer look: XMCL (eXtensible Media Commerce Language) [21], XrML (eXtensible rights Markup Language) [23], and ODRL (Open Digital Rights Language) [9].

All the candidates had enough expressive power for our needs and they all have numerous supporters in the technology and content producing industry. However, the industry support was not essential for us since we were not developing a commercial product, but had a more academic perspective to the problem. We decided to use *ODRL*, mainly because it is an open standard and therefore more flexible than the other candidates. The current version of ODRL when we made our choice was 1.0 thus the implementation is based on that version. Since we had made our choice, both ODRL and XrML have achieved considerable victories over the others. For example XrML won the competition for rights expression language for MPEG-21 media distribution standard and ODRL has been the choice of Open eBook Forum. The overall winner of the rights description language competition is yet to remain unsettled. [3], [7]

2 Nonius implementation

2.1 The design of the program

The intention of *Nonius* implementation project was to enrich our rather theoretical *MobileIPR* research project with more practical experiences on digital rights management. We had already studied legal, technical, and economic issues related to rights management on the Mobile Internet, [8] but we were willing to find out what kind of challenges come up when putting rights management into operation. Luckily, we had a “sister-project”, *XML Devices* that was creating a Java based XML browser, *X-Smiles*. The browser was intended for both desktop use and embedded network devices and to support multimedia services. It did not have any support for DRM, but it formed an excellent platform to develop a DRM extension. [20], [22] Another HIIT's project, *STAMI*, gave us background support in security technologies, which were the project's speciality. [17] The

Nonius project started in September 2001 and was finished in April 2002. It took a total of 1218 working hours including the overhead required by the software project class. [15]

X-Smiles is an open source XML browser implemented in *Java*. It supports several XML technologies such as *SMIL*, *XSL*, and *XForms* [20], [22]. *X-Smiles* is continually developed further. *X-Smiles* version 0.5 was used in Nonius. The majority of the extension's functionality was implemented as separate modules. Only a couple of changes to the GUI code of *X-Smiles* itself were made. The development environment was running JDK 1.3.1 on Microsoft Windows 2000 and Windows XP operating systems.

X-Smiles offers an easy-to-use interface for connecting tailored modules for handling different types of XML documents. These modules are called *MLFCs* (Markup Language Functional Components) in the *X-Smiles* terminology. Basically, to connect an *MLFC* to *X-Smiles*, one first writes the implementing class and support classes for the needed functionality. After this, a mapping between the wanted XML document type (*namespace*) and the name of the implementing *MLFC* class is added in the *X-Smiles* configuration file. After this, the browser core automatically calls the module when that type of XML document is requested.

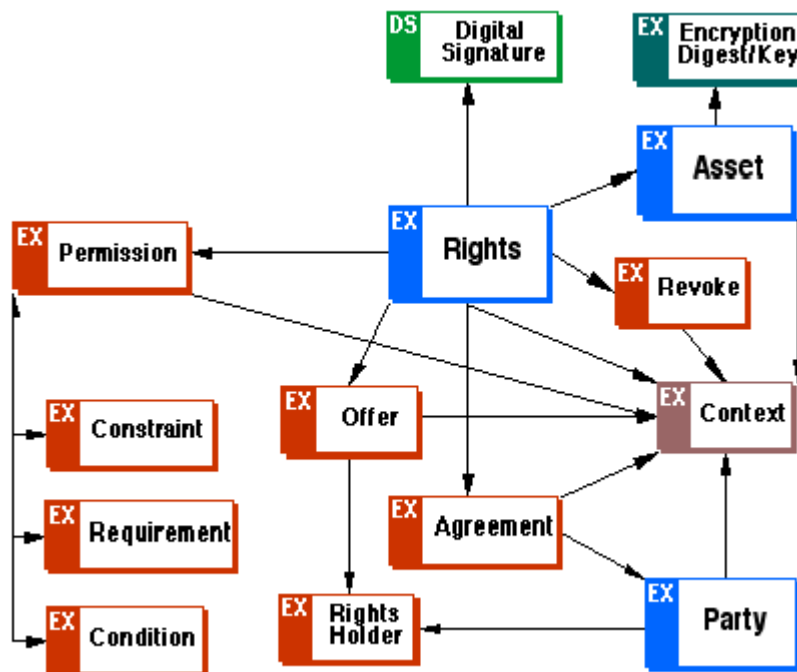


Figure 1. Concepts modeled by ODRL in a broad context [6]

In our case, an *MLFC* for handling ODRL documents was written. The architecture of Nonius extension consists of the following components:

- *DRMMLFC*, the interface to *X-Smiles*;
- *DRMStore*, a certificate database;
- *HandlerDRMOffer* for displaying the asset;
- *HandlerDRMAgreement* for displaying and saving a certificate into the database;
- *AssetDecode* for enforcing the correct handling of rights and possibly decryption;
- *SecurityModule* for security functions.

What comes to implementation of ODRL, the most interesting module is *AssetDecode*, which essentially consists of the parser of ODRL documents. Our aim was that the parser should be highly modular. The architecture should allow a change of the underlying DRM language by implementing a common set of

interfaces. The interfaces were designed using ODRL as a basis: the conceptual model for the architecture follows very closely what is presented in the ODRL 1.0 specification [6].

Figure 1 presents the most important concepts of ODRL and is explained in more detail in the ODRL 1.0 specification. Similar concepts are common in all DRM scenarios. An *Asset* is the product or work that DRM aims to protect. An *Agreement* is a contract between some parties that defines the terms under which a party may use the Asset. An Agreement is essentially a *certificate* in many contexts. An *Offer* is like an Agreement, but it does not specify any user. An Offer may be freely distributed for viewing what kind of usage terms are related to an Asset.

In the implementation, each conceptual class was modeled as a Java interface with accessor methods for reading business logic related data and some standardized methods for constructing objects from general XML elements. After we had designed the interfaces, we wrote implementing classes with ODRL specific parser functionality.

The concepts of the DRM language are presented in practice inside XML documents as XML elements. The basic idea in the parser architecture was that each class knows what kind of term or restriction it represents in the DRM scenario. Each class represents one XML element in the DRM language, and knows what kind of information it should extract from the XML element when constructed. Further, each class needs to know what elements might be related to it and knows to pass the request for verifying a term to the relating child elements if it does not know how to verify it directly itself. An additional requirement was that the certificates should be viewed in a more humanly understandable way than XML: for this purpose, each class was assigned the responsibility to be able to represent the information it contains in a human language. These common features were implemented in one common interface, which was extended by all the other interfaces. The responsibility to create objects was given to separate factory classes when this was feasible, thus enabling easy development and rollover of new functions.

2.2 Assumptions and alternative solutions

At first, we planned that the opening of any Asset in the browser would trigger the validation of DRM rules that were related to it. This immediately proved impossible, since X-Smiles only allows us to map the execution of our module with certain XML documents, not all. Of course, this obstacle would have been possible to circumvent by changing X-Smiles' code so that it would always call our module first. One of the goals of the project was to be able to implement DRM functionality with as few changes in the X-Smiles itself as possible, so this solution was rejected.

Another problem is that each Asset needs to be uniquely identified, and a plain URL is not enough for this purpose. As a solution, we thought of defining and implementing an own XML based language. The new language would have contained metadata and execution instructions for opening ODRL documents. After we examined ODRL closer, we found out that an Offer document contains all the information necessary to implement the same functionality. Only four assumptions needed to be made, and they were quite reasonable, so this is the solution we chose.

The following assumptions were made:

1. Each Asset is always accessible via a URL (which may be local as long as X-Smiles supports it) and points to an XML document that X-Smiles can display.
2. Displaying of an Asset is initiated by opening an Offer document
3. Offers need to contain a URL pointing to the Asset
4. Offers need to contain a unique identifier for the Asset

Once an ODRL Offer document is opened (by assumption 2, each protected asset is opened this way), our module is loaded. Then it may identify the Asset that is being opened (by assumption 3). After this, it can search for the corresponding Agreement in the database. If an Agreement is found, the DRM terms are validated and, if everything is acceptable, the module can ask X-Smiles to open the Asset (by assumption 1) pointed to by the URL in the Offer (by assumption 3).

2.3 DRM features implemented in Nonius

ODRL defines lots of DRM capabilities. The purpose of this project was not to implement them all. In the requirements capture phase, the most important capabilities and functionalities were chosen and prioritized. We also chose to implement some general functionality not directly related to ODRL, such as user data management features or saving DRM documents in a user's system. The implemented ODRL features are:

- viewing the content of Offers and Agreements in a human understandable way,
- restricted time of validity of certificates,
- restricted individual (only a specific user may use an Asset),
- restricted software (only the specific instance of X-Smiles browser may use an Asset),
- restricted instances of use,
- restricted accumulated time of use,
- restricted geographical area, and
- any combination of these (AND –operator).

In order to implement these, a lot of the basic ODRL concepts – such as Permission, Constraint, Context and Party – also needed to be implemented.

In contrast, some of the most interesting ODRL features that were left out include:

- different usage types (such as view, print, modify, sell): we implemented only display, since this is currently the only act that X-Smiles supports;
- super distribution (X-Smiles does not support any such methods),
- device constraints other than software (printer, CPU, screen etc.),
- aspect constraints such as quality or watermark,
- target constraints such as the purpose of use,
- combination of constraints using OR and NOT operators, and
- secure encryption methods, such as public key cryptography (security was not the focus area of this project).

3 Experiences

This chapter discusses the problems we faced and experiences we gathered during the implementation project. Some of the problems are specific to the technical environment, some relate to the architecture we chose, and some derive from the ODRL specification. We are also discussing about more general issues related to DRM. However, although we are aware of numerous non-technical challenges related to DRM, like the lack of common standards, patent problems, consumer protection, privacy, user rights, and so on, we have excluded them from this article. (For more information on the non-technical issues, see e.g. [10], [16].)

3.1 Experiences from the programming process

One goal of the project was to design a scalable and modular architecture for the parser, one that might even allow a change of DRM language if necessary. Scalability was important for the process point of view: functionality was added in iterations and we had planned only two or three weeks for the implementation per iteration.

The team found that the ODRL specification was well written and relatively easy to understand. From it, business logic aspects of the architecture were quite easy to design and it did not require many modifications once designed.

On the other hand, other parts of the architecture such as the parser framework and utility classes were quite dynamic and hard to freeze. We did not have any prior experience in implementing DRM systems. The inexperience of the team in implementing parsers that create object structure from XML documents caused many changes during the project. Requirements for new XML parsing utility functions were discovered one by one. Luckily, these features had been implemented in a separate class that was easy to improve when new requirements arose.

We did not test the modularity of the architecture by changing the DRM language. Basically, it should be a straightforward task as long as the new language is XML based and its concepts can be unambiguously mapped to ODRL's concepts. Interpreting the interfaces correctly might require some knowledge of ODRL, though.

3.2 Shortcomings in the implementation

The implementation in itself is far from perfect: certificates are easily accessible in a human readable and modifiable format. All the user related data is also easy to change. This could be done in a more secure way by sealing the certificates. For example, hash values could be used for verifying integrity.

Another problem comes from the implementation of the software constraint. For this purpose, the program chooses a random device ID when it is first needed. This, along with other user data, is stored in a user's hard disk. If X-Smiles needs to be reinstalled, the device ID vanishes and certificates containing software constraint will not work, since the ID has changed. This could be fixed by storing the ID, for instance, in a remote server, where it could be retrieved during reinstallation.

The only "encryption" method supported is gzip. It was good enough solution for our demonstration as its only purpose was to make the documents unreadable for humans. Public key cryptography could be used, but to be effective in practice, it should be supported by the hardware.

These shortcomings are decisions that were deliberately made during the project implementation. For the most part, these do not lessen the product's usability for demonstrating DRM capabilities.

Two X-Smiles specific problems are related to the implementation of accumulated time constraint. Firstly, there is no way of stopping the displaying of a document once the viewing has started. This means that if the certificate allows a certain period of accumulated time, users may still view the document for as long as they like, if they never shut the browser down or load a different document. A workaround could be implemented, for example, with a specific timer thread that tells the X-Smiles to load a blank page after the certain period of time, but we did not consider this as good design and a lasting solution. Therefore it was not implemented.

The program starts the timer that counts accumulated time already before it tries to load the asset, although it can take a long time to load a large document over a slow connection. Again, X-Smiles does not have an "afterLoad" event that could trigger the accumulated timer for this purpose.

An X-Smiles bug prevented the program to display other than SMIL documents. This, however, does not affect the usability of the program for demonstration purposes. SMIL documents are good enough with music, image, video and sound effect capabilities.

3.3 Difficult areas

ODRL specification states that if the software comes across some DRM restriction that it does not know how to validate, it should not allow the action. With our architecture, this is a challenging area. It has been implemented where possible. Easy places to add this kind of functionality are the factory classes. If a factory class gets an XML element that it has not got a mapping for, a `DRMParseException` with an explanatory message is thrown. When such an exception is thrown, the user will not be able to perform the action and an error message is presented. `DRMParseExceptions` are thrown also if there is something wrong with the data parsed (wrong type, missing information etc.). But, since the program uses the XML DOM approach for parsing, when an object is being constructed from an XML element, it asks the DOM object for the information it needs. If there is an extra element in a place that some factory does not handle, it goes unnoticed. This violates the ODRL specification. A simple solution would be to write a custom XML schema file for the implemented subset of ODRL, and use that to verify documents. This was not implemented in the project.

The program currently supports only one certificate per asset. No certificate merging was implemented. This means that if a user gets another certificate for an asset, the latter automatically overwrites the previous one. The merging of certificates is not a trivial task especially if the certificates are complicated. The biggest problem is that when merging two DRM documents, one would need to consider the semantics of the XML elements. The merging problem has been studied in software configuration management, especially in relation to version control. There exists dozens of algorithms for textual based merge (line per line comparison), and some for syntactic or semantic merge for a specific programming language [2], but studying and implementing one of these for ODRL would be a project in itself.

One challenging feature was adding support for incremental requirements. These are requirements that are not necessarily known when a user wants to use an asset but may come up later disallowing the use of the asset. We designed the framework for adding such requirements and implemented one: pay per view. As the result of the query to open a document passes on through the object structure, each DRM component may add its own additional constraint to the result. The additional constraints are implemented as a class that knows how to merge itself with another object of the same class. These objects gather together like streams into a river, and in the end, if there are additional constraints, the user will be asked to fulfill the requirements before the document is used. One problem here is the order of additional constraints. For example, if there are two pay-per-view constraints in the document and the user agrees to pay when the first constraint is displayed, but when the second one pops up, the user decides to back off. Now, the first payment has already been made and cannot be returned without a comprehensive rollback mechanism.

3.4 Challenges in the ODRL specification

The ODRL specification 1.0 defines a huge amount of functionalities and places a lot of requirements for an implementation. We found that three requirements were more challenging than others. These should be carefully studied when starting to implement an ODRL based DRM software product. We implemented none of these features. All of them would require some changes in the software architecture, and combined they would require a serious refactoring effort to keep the design complexity from rising exponentially. Also, implementing these would easily triple or even square the amount of work needed to run thorough tests. These requirements basically make it easier to write certificates, but they also make the implementation of a software tool that parses DRM documents quite a bit more complex.

The first requirement that complicates implementation is *logical operators*. AND is the easiest operator to implement with the architecture that we used: every object validates the action and if there is a conflict, an exception is thrown. AND is the only operator we implemented. NOT operator would also be easy to implement. Even though we did not implement it as such, it can be implemented by the one who writes the certificates by inverting the rules. An OR is difficult. Even though in theory it can be implemented by the certificate writer by the rule $A \text{ OR } B == \text{NOT } A \text{ AND } \text{NOT } B$, this makes the job of writing the certificates very difficult. Our architecture does not bend easily to support OR, it would require some kind of transactional support: if one rule fails (throws an exception), another may still be valid and “roll back” the exception.

Another challenging requirement in the ODRL specification is *document's internal references* to its elements. The specification states that any element may be linked from any other place of a document. This is fine as long as the semantics of the element does not depend on its context. The third requirement presented in the next paragraph destroys this assumption, thus making this requirement very hard to implement. A reasonable way to

implement this would be to run the document through a preprocessor before handing it over to the parser. The preprocessor would expand internal links to full XML elements and the parser would not need to know anything about the links.

The third challenge is that some deeper level element may depend on its ancestor elements' data. For example, "if a Requirement appears at the same level as a number of Permissions, then the Requirement applies once to all of the Permissions" [6]. From the parser point of view, this is one special case more: when parsing a Permission, the parser should also check the parent element if it has a requirement. If it has, then append it as it had been a child of the Permission. Alternatively, a preprocessor could be the solution to this one, too: move all Requirements that are not under any Permission, under all the Permissions that they are on same level with.

3.5 Technical problems with DRM

Major issues in implementing the DRM extension were security and open source. These issues are also close to user rights and user needs. From the security perspective it should be stressed that rights description languages hardly touch security issues. Secure packaging of the content and secure transfer is an independent issue of the content usage rights and rules. It is anyhow central because users must trust on both the availability and usability of the information they receive for consumption.

Security is always a problematic issue. The copyrighted material should be encrypted so that it cannot be illegally used. The main problem is that information should not be decrypted by the application itself, because then the decrypted information would reside in digital format in the memory of the operating environment. For example, if digital music is decrypted by the player application, another application can read the decrypted digital music data and create a perfect copy. If decryption is done by hardware, in this situation by sound adapter or a loudspeaker, it makes unauthorized copying significantly more difficult.

Surely, unauthorized copying, usage, and distribution can be forbidden by laws and agreements, but those hardly affect evil users. As long as the source code is open, it has to be assumed that anyone may modify it to capture decrypted content. Even if the source code is not publicly distributed, it is possible to decompile and debug the executable code, insert hooks and affect what the program does. Therefore these problems do not touch open source only, but because open source code is so easy to modify, the challenges in connection with it are most serious.

X-Smiles browser and our DRM implementation are open source software. Also, ODRL language is documented and developed in open source fashion. This means that any user can download the source code of our DRM system and ODRL specifications of rights certificates we use. From user perspective open source is as easy (or difficult) to market as for example Linux operating system. For content owners, it is a further question if one can ever trust on a DRM system from which any would-be hacker can download the source code. However, some studies imply that from security perspective it is not relevant if the source code of a DRM system is open or not. Therefore, we would be eager to suggest that open source has only positive impacts from both user and copyright holder perspectives.[1]

DRM should be supported by the operating system and hardware. That would be a much better approach than application layer. If the implementation is made on the application layer, it is too easy to circumvent. If an operating system supported DRM, it would, for example, guarantee that no other application can read or change saved certificates information. However before DRM will be supported by operating systems and hardware, there must be a widely supported standard. Somebody must also pay for these features, so the market must be ready and interested about DRM.

4 Conclusions

In the digital world, immaterial products are cheap to distribute through fast, global data networks. This introduces a lot of business potential, but also raises a big piracy problem. A digital rights management system separates information from the right to use it. With a DRM system, a content publisher can limit the usage of an information product so that it can be used only in accordance with defined rules.

So far there is no widely used standard for rights description languages. We studied three competing standard proposals and selected ODRL to be used in our project. We succeeded in implementing an XML browser

extension that demonstrates DRM features. The most important implemented features enable usage rules based on identity, time of validity, and counts of use. Some of the interesting features that were left out are aspect and target constraints.

Based on our experiences, merging certificates and developing an interface for editing certificates are among the most challenging tasks in implementing a DRM system. The ODRL specification includes also challenges. In our opinion the three most challenging features in the specification are logical operators, documents' internal links, and the requirement that in some cases child elements may depend on some ancestor elements' children.

All in all, Nonius project succeeded in its main goal of producing a piece of software for demonstrating DRM functions although DRM in itself is a very complex and largely controversial subject including many non-technical problems not discussed in this article.

5 Acknowledgements

MobileIPR project is funded by *Tekes* – the National Technology Agency of Finland, Elisa Communications, Nokia, Sonera, and Finnish Broadcasting Company (YLE). In addition to the authors, especially Mr. Ville Oksanen participated in defining the project objectives and gave useful comments on this article. Discussions with *XML Devices* project team that has created X-Smiles browser and *STAMI* project team focusing on security issues were most important. Nonius team was also guided by the teachers of Helsinki University of Technology's T-76.115 Software Project class, especially by Mr. Risto Sarvas and Mr. Jari Vanhanen.

Bibliography

- [1] R. Anderson: Security in Open versus Closed Systems -- The Dance of Boltzmann, Coase and Moore, *Conference on the Economics, Law and Policy of Open Source Software*, Toulouse, France, 2002, <http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/toulouse.pdf>
- [2] J. Buffenbarger: *Syntactic Software Merging*, Seattle, June 1995, pp153-172.
- [3] R. Cover: *The XML Cover Pages: XML and Digital Rights Management (DRM)*, 13.5.2002, <http://xml.coverpages.org/drm.html>
- [4] *Features of DRM*, Microsoft Corporation. <http://www.microsoft.com/windows/windowsmedia/WM7/DRM/features.asp>
- [5] J. Gilmore: *What's wrong with copy protection*, 2001, <http://www.linux.it/GNU/articoli/whatswrong.shtml>
- [6] R. Iannella: *ODRL Specification*, 2002, <http://www.odrl.net/1.0/ODRL-10.pdf>
- [7] N. McAllister Freedom of Expression: Emerging standards in rights management, March 2002, <http://www.newarchitectmag.com/documents/s=2453/new1011651985727/index.html>
- [8] MobileIPR homepage, 2003, <http://www.hiit.fi/de/mobileipr/>
- [9] *ODRL standard homepage*, <http://www.odrl.net/>
- [10] O. Pitkänen: *Managing Rights in Information Products on the Mobile Internet*, HIIT Publications 2002-4, Helsinki Institute for Information Technology HIIT, 2002.
- [11] O. Pitkänen, M. Välimäki: Towards A Digital Rights Management Framework. *IeC2000 Proceedings*, UMIST, Manchester, UK, 2000.
- [12] O. Pitkänen, M. Mäntylä, M. Välimäki, J. Kempainen: Assessing Legal Challenges on the Mobile Internet, *International Journal of Electronic Commerce*, Fall 2003, Vol. 8, No. 1, pp. 101-120, 2003.

- [13] A. Pruneda: *Windows Media Technologies: Using Windows Media Rights Manager to Protect and Distribute Digital Media*, Microsoft Corporation (white paper)
<<http://msdn.microsoft.com/msdnmag/issues/01/12/DRM/DRM.asp>>
- [14] V. Saarinen, J. Anttila, P. Lauronen, O. Pitkänen: *Implementing a DRM System*, HIIT Technical Reports 2002-3, Helsinki Institute for Information Technology HIIT, 2002
- [15] V. Saarinen, J. Poikela: *Nonius Final Report*, (in Finnish) 2002, <<http://jt7-332.tky.hut.fi/nonius/lu/loppuraportti/loppuraportti.doc>>
- [16] A. Soininen (ed.), O. Pitkänen, M. Välimäki, V. Oksanen, T. Reti: *MobileIPR Final Report*, HIIT Publications 2003-3, Helsinki Institute for Information Technology HIIT, 2003.
- [17] *STAMI project homepage*, 2003, <<http://www.tml.hut.fi/Research/STAMI/>>
- [18] *Understanding DRM Systems*, Intertrust Corporation (white paper).
<<http://www.intertrust.com/main/research/index.html>>
- [19] H. Varian, C. Shapiro: *Information Rules*, 1999, Harvard Business School Press
- [20] P. Vuorimaa, T. Ropponen, N. von Knorring, M. Honkala: *A Java based XML Browser for Consumer Devices*, *Proceedings of SAC*, Madrid, Spain, 2002.
- [21] *XMCL standard homepage*, <<http://www.xml.org/>>
- [22] *X-Smiles homepage*, 2004,
<<http://www.x-smiles.org>>
- [23] *XrML standard homepage*, <<http://www.xrml.org/>>

First ODRL International Workshop

Invited Talks



VirtuosoMedia
secure digital rights management

Flexible DRM

Real-life ODRL Implementations
By
VirtuosoMedia

© VirtuosoMedia



VirtuosoMedia
secure digital rights management

Content

- Introduction VirtuosoMedia
- General Broker Idea
- What is leading
- Why ODRL
- Implementation
- Remarks

© VirtuosoMedia

VirtuosoMedia

- We protect rights-owners' business
- Focus on multimedia (audio, video)
- Open standards
- Multiple platforms
- Our own, small-footprint, players
- Complete end-to-end protection!

© VirtuosoMedia

Our Solutions

- Offline professional multimedia protection for “Bouwbox Digitaal”
- Online music solution with “VirTunes”
 - We keep track of the tracks
- Set-top box solution with “Virtiq Technology”
 - Key to paid content



© VirtuosoMedia



VirtuosoMedia
secure digital rights management

“Bouwbox Digitaal”

- Subscription based
- Offline content (e.g. CD-ROM or DVD)
- Online retrieval of permissions
- Offline playback
- Retrieve statistical data
- Add new or improved content
- Rights-Management system ready for online (if bandwidth available)



© VirtuosoMedia



VirtuosoMedia
secure digital rights management

VirTunes Music Downloads

“We keep track of the tracks”

- Contract with Buma/Stemra society
- No subscription for consumers
- Online content
- Online retrieval of permissions
- Offline (download) playback
- Statistics via server
- Future: download to other devices



© VirtuosoMedia

- Combined DRM & secure payments
- Subscription and “Impulse”
- Online retrieval of permissions
- Online (streaming) or offline (recorded) playback
- Statistics via server and via STB
- VirtuosoMedia DRM integrated
- Banking approved payment integrated (EMV2)

© VirtuosoMedia

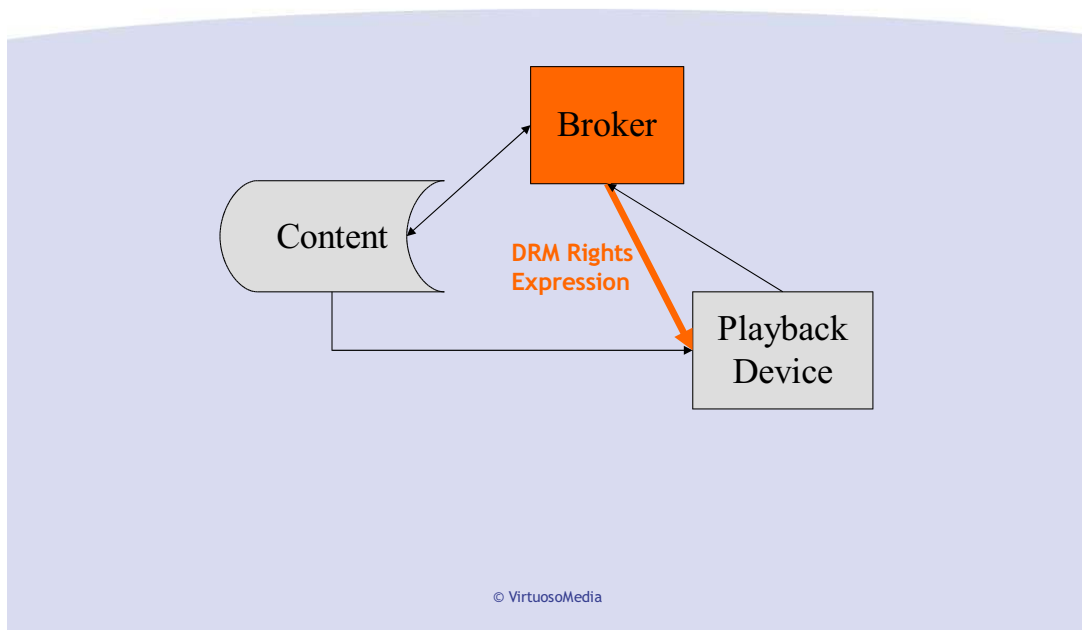
General Broker Idea I

Requirements:

- Flexible
- Connectible
- Extendable
- Multiple application areas

© VirtuosoMedia

General Broker Idea II



General Broker Idea III

DRM Rights Expression

- Permission to playback content
- Where and when
- (Personalised) decryption info
- Conditions (payment, registration)
- Statistics

What is leading I

Content

- + Defined
- + Traceable
- Lot of redundant information
- Not adequate as Normalized key

Consumer

- Integrity problem
- Redundant information
- + Traceable

© VirtuosoMedia

What is leading II

New entity: Contract

- Permission
- Where and When
- Conditions
- Identification

© VirtuosoMedia



VirtuosoMedia
secure digital rights management

What is leading III

Contract can be combined with Content
Consumer agrees to a Contract and
obtains a Personalised
Agreement

Datamodel is:

- More natural
- Easy to maintain
- Less redundant information

© VirtuosoMedia



VirtuosoMedia
secure digital rights management

Why ODRL

- Basic idea is the same
- Open, and thus more applications (hopefully)
- Fits with our idea of rights holders as separate entities (e.g. music societies)
- Extendable
- Faster 'time to market'

© VirtuosoMedia

Implementation I

Define Building blocks, to create:

- Assets
- Contracts or Offers
- Agreements
- Parties

© VirtuosoMedia

Implementation II

‘Bouwbox Digitaal’ Contract

- Subscription based (Time restrictions, renewal)
- Titles
- Other Restrictions
 - Maximum number of playback
 - Maximum number of titles to play
 - Etc.

© VirtuosoMedia



VirtuosoMedia
secure digital rights management

Implementation II

VirTunes Music Download

- Define some Contracts
- Couple requested track on the fly to contract
- If all requirements are met (registration, payment etc.) create personalised Agreement and sent this to Consumer

© VirtuosoMedia



VirtuosoMedia
secure digital rights management

Implementation III

Virtiq Technology is a combination of previous solutions, plus:

- hardware ID for authentication
- integrated banking system for payment

© VirtuosoMedia

Conclusion

VirtuosoMedia
plays it safe
with
ODRL!



The Open Digital Rights Language Initiative. International Workshop.

Vienna, Austria, 22-23 April 2004

An Interoperable and Flexible Infrastructure for DRM

Invited Talk

Dr. Mariemma I. Yagüe
Computer Science Department
University of Málaga

e-mail: yague@lcc.uma.es



ODRL International Workshop. Vienna, Austria, 2004 2

Motivation

- **Rights management applies to a wide variety of systems and objects.**
 - Almost impossible to consider all **potential application** scenarios.
 - The approach must be developed in an **open** an **extensible** way.
- **Most DRM systems are very complex.**
 - the underlying framework tries to capture all possible details and features of the targeted scenarios





Motivation

Flexibility and extensibility


A **general framework** capable of supporting **very heterogeneous** DRM applications and scenarios



Motivation

- **Rights enforcement involving an access decision about a resource subject to intellectual property rights** in highly dynamic, open and heterogeneous scenarios, with very large numbers of users, resources and stakeholders.
- **Current access control models are not appropriate** for the DRM and other open, heterogeneous and dynamic scenarios.
 - Access Control erroneously considered to apply to “**locations**” instead of “**resources**”.







ODRL International Workshop, Vienna, Austria, 2004 5

Motivation


- **Need to manage access control parameters, in a distributed, dynamic and transparent way.**
 - Automatic establishment of access conditions based on semantics
- **A DRM system must also deal with digital content protection.**
 - production of self-protected software objects that convey contents (software or data) and access control, rights and obligations enforcement mechanisms.



ODRL International Workshop, Vienna, Austria, 2004 6

Objectives

- **Extensibility.**
- **Interoperability.**
- **Scalability.**
- **Interoperable Access Control Scheme.**
- **Copyright agreements, payment, and other operations bound to access to the contents.**
- **Persistent Protection**





Fundamentals

Independence of the certification of attributes function

Interoperability

it allows attributes to be safely communicated avoiding the necessity of being locally emitted by the system administrator




Fundamentals

The security requirements of all processes related to the secure transmission and commerce of digital contents can be fulfilled if we guarantee that the software running at the other side of the communication line is

protected

it is neither possible to discover nor to alter the function that the software performs and it is also impossible to impersonate the software







ODRL International Workshop. Vienna, Austria, 2004 9

Contribution

- Development of an infrastructure (**XML-based Secure Content Distribution, XSCD**) providing distributed access control, fulfillment of obligations (payment, etc) and persistent protection on the basis of:
 - A **new access control model** based on **semantics (Semantic Access Control, SAC)**, implemented with a **policy language (SPL)** and a set of **smart tools** for the easy specification and automatic validation of access control criteria.
 - A mechanism for the **semantic integration of a PMI (Source of Authorization Description meta-model)** with the access control.
 - A **software protection mechanism (SmartProt)** [Maña, 2001] for mobile environments which enables to implement active containers for the contents to distribute.




ODRL International Workshop. Vienna, Austria, 2004 10

Main Advantages of SmartProt

Robustness against different attacks

- bypassing the check,
- code substitution and
- attacks to the license management protocols.

- **Confidence** for the user,
 - efficient use of the computational resources of the smart cards,
 - free distribution and copy of the software,
 - selective license transfer,
 - control of the expiration of the licenses and
 - applicable in distributed computing.





Main Advantages of SAC

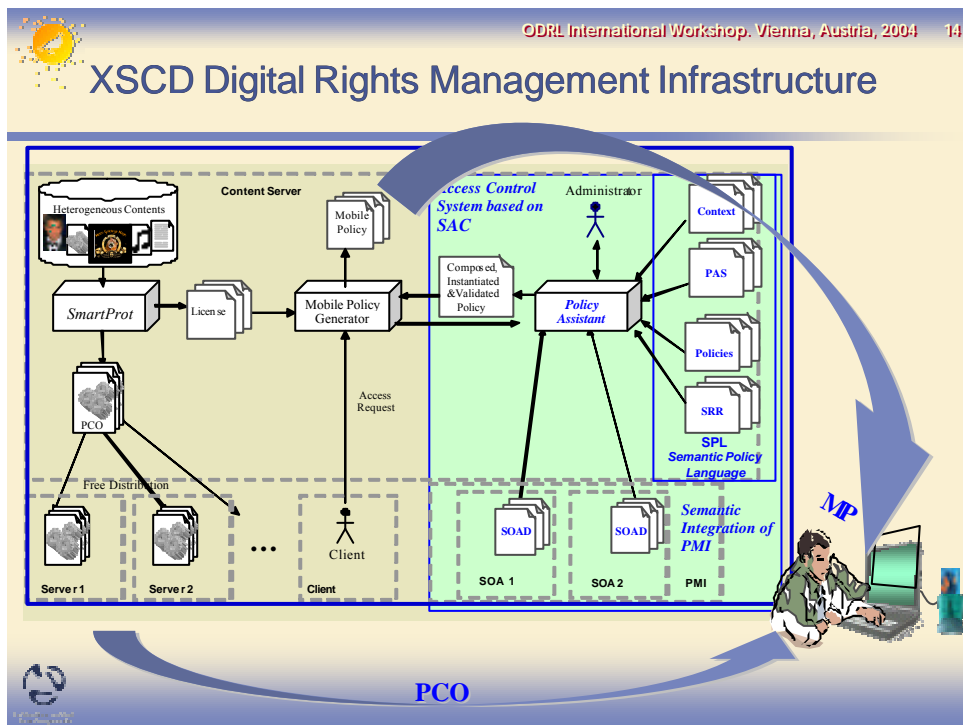
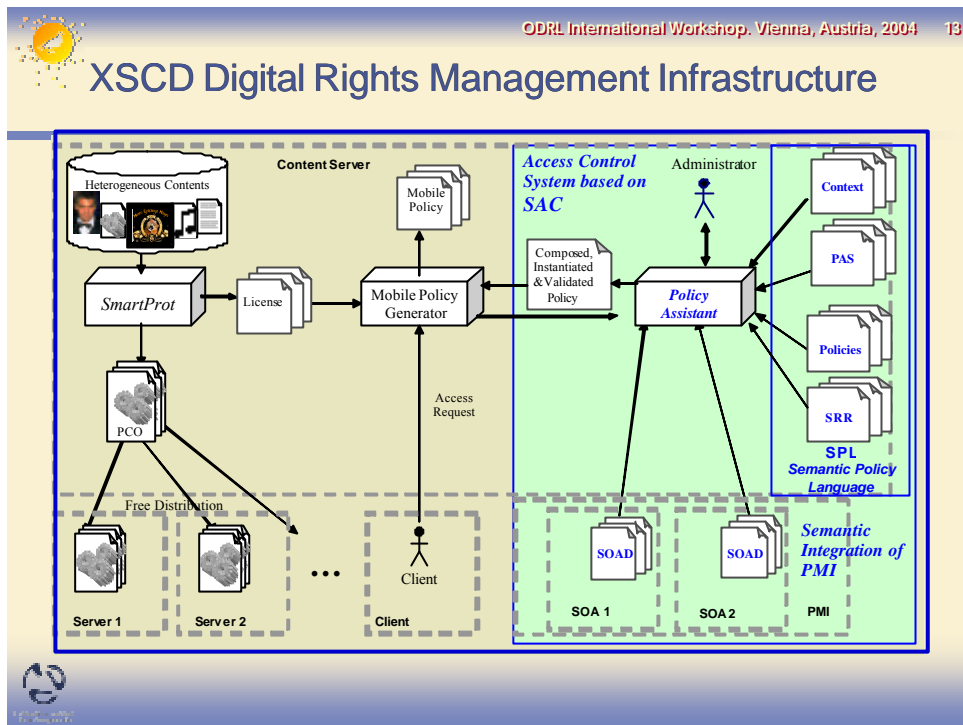
- **Simple Specifications.**
- **High expressiveness.**
- **Unambiguous specifications.**
- **Modular policies, with dynamically instantiated parameters.**
- **Semantic validation of access control criteria.**
- **Content-based access.**
- **High Scalability.**



Main Advantages of SAC

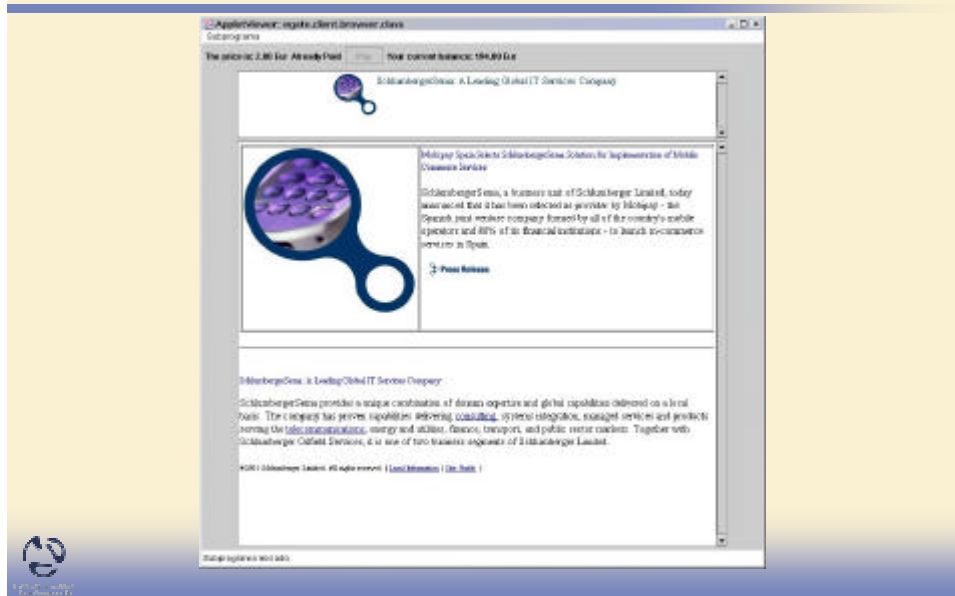
- **Fully distributed** scheme based on attribute certificates.
- **No subscription** required.
- High level of **interoperability**.
- **Transparent** and **dynamic** modification of policies in **distributed** env.
- General **applicability**: software and data objects.
- Integration with **external authorization** entities.
- Support of **payment** and other **actions/obligations** bound to access.







XSCD Applied to E-news



Conclusions

- **Flexible and Interoperable DRM Solution** that enables content owners to enforce access control policies, copyright agreements, payments and other obligations, to digital objects in a distributed environment.
 - **Is not dependant of Autentication Infrastructures**
 - **Semantic integration of PMI**
 - **Access control based on semantics of the resources and the context**
 - **We can define access conditions and obligations for each piece of information**
 - Non subscription, identification needed.
 - Applied to **different scenarios** such as
 - E-commerce, Digital libraries, Web Services, Grid computing,...
 - Supports **pay per use** and **pay per adquisition** business models.





An Interoperable and Flexible Infrastructure for DRM

Dr. Mariemma I. Yagüe
Computer Science Department
University of Málaga
e-mail: yague@lcc.uma.es

