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 *MLFC*s (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.

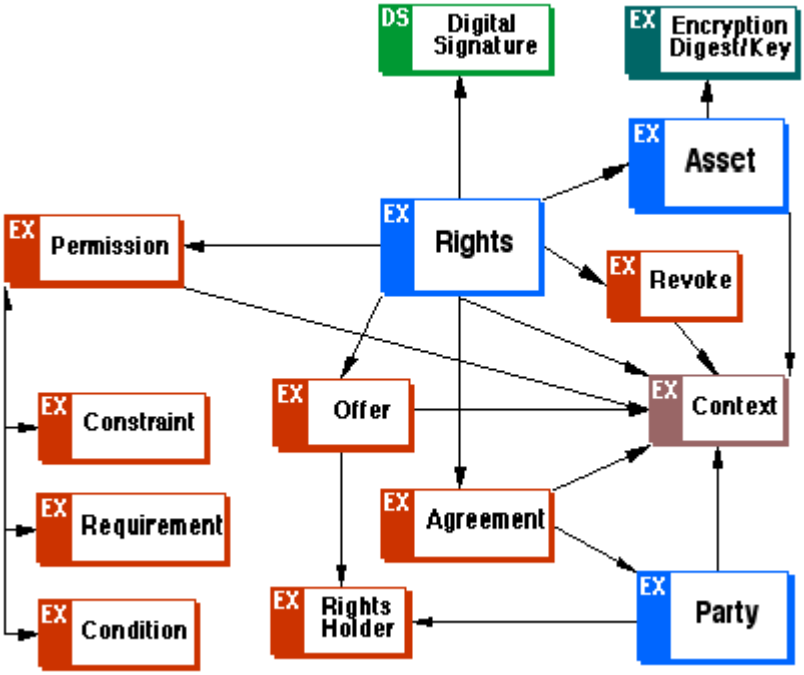

Figure 1. Concepts modeled by ODRL in a broad context [6]

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 OR B == NOT A AND 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. Kemppinen: 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.xmcl.org/>

[22] *X-Smiles homepage*, 2004, <http://www.x-smiles.org>

[23] *XrML standard homepage*, <http://www.xrml.org/>