Investigation of Third Party Rights Service and Shibboleth Modification to Introduce the Service

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Master of Science

in

Computer Science

by

Sanket Agrawal

DARTMOUTH COLLEGE

Hanover, New Hampshire

June 26, 2003

Examining Committee:

Sean Smith (chair)

Edward Feustel

Chris Hawblitzel

Carol Folt
Dean of Graduate Studies

Abstract

Shibboleth is an architecture to support inter-institutional sharing of electronic resources that are subject to access control. Codifying copyright in Shibboleth authorization policies is difficult because of the copyright exceptions which can be highly subjective. Third Party Rights Service is a high-level concept that has been suggested as a solution to approximate the exceptions of copyright law. In this thesis, I investigate the components of the Third Party Rights Service. I design and analyze a modified Shibboleth architecture based on these components. The resulting architecture allows for the phased addition of the resources to make use of the Third Party Rights Service, while keeping the existing resources in Shibboleth.

Acknowledgment

I would like to express my sincere gratitude and appreciation to many people who made this thesis possible. Special thanks are due to my adviser Prof. Sean Smith and Dr. John Erickson of HP Labs who guided me during my research work. I also thank Prof. Fillia Makedon for her help and guidance.

I would like to thank Prof. Ed Feustel and Prof. Chris Hawblitzel for agreeing to serve on the thesis committee. Their feedback and comments were very useful in the thesis writing. I am also grateful to Mellon Foundation, US DoJ, and NSF for funding the thesis research.

Finally, I feel especially indebted to Ms. Sally Kinlaw, Bev Parry and others who through their very warm and friendly hospitality made my stay at Dartmouth an experience to cherish.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The problem of codifying copyright law in current *Digital Rights Management* (DRM) systems is difficult because of the limitations of policy evaluation and enforcement in these systems [12], and the broad domain of copyright law. As I will discuss in this thesis, copyright law makes certain exceptions that are broad and subjective. As more and more efforts are devoted to bring these systems to the academic community, mapping copyright to the policies in these systems is important because of the research and education requirements for sharing of knowledge and information. For example, research activities greatly benefit from the free flow of ideas and information. To this end, the universities have depended on their libraries to provide and disseminate information to the scholars and public. Fair Use and other doctrines of copyright law are important in these efforts, and I explain these doctrines in Chapter 2.

Shibboleth is a framework for distributed authentication and authorization that has been designed for sharing of resources by the academic community. It can serve as a basis for a DRM framework. A Shibboleth-based DRM framework may have certain shortcomings that can make mapping copyright to that DRM framework difficult. I explain these shortcomings in Shibboleth in Chapter 2. *Third Party Rights Service* is a solution that has been proposed as a step towards correcting these shortcomings, and so, I will briefly examine the implementation issues for Third Party Rights Service in Chapter 2. I then examine these issues in more detail in Chapters 3, and 4. In Chapter 5, I modify the Shibboleth architecture to accommodate the Third Party Rights Service. I conclude the thesis with discussion on future work in Chapter 6.

Chapter 2

DRM and Fair Use

2.1 Introduction

Digital rights management (DRM) is a term that has been loosely applied to describe both copyright enforcement and copyright management. It includes a range of technologies that give parties varying degree of control over how the contents might be used, including by whom and under what conditions.

Copyright enforcement techniques commonly refer to the enforcement of copyright laws. Though DRM systems may prevent illegal copying and distribution of copyrighted works, they may also exceed the limits set by the copyright law, which the users may find undesirable. The DRM system might as well prevent the copying and distribution of public domain work as well as copyrighted work. Even though the copyright law confers on the copyright owners the right to control only public use and performance of the works, the DRM systems might be used to control the private use and performance of the work. They might also be used to compel the users to view the contents they will like to skip (such as FBI warning notices), thus exceeding the copyright's bounds [41].

DRM systems may also be used to manage the rights including *super-distribution* [33, 36, 38]. Super-distribution schemes model the rights for an object, along with attributes such as the rights that users are allowed to pass to others and further along, consideration, extents and type of users. While super-distribution

schemes allow for management of rights along the chain of content users, the license request and granting services allow for the negotiation and modification of content user's rights [26, 34]. But, DRM systems may manage the rights beyond the boundaries of copyright laws. For example, while copyright law allows for the sale of copyrighted work by users, DRM systems might disallow such selling of work.

Thus, for the contents that are in digital formats, "digital rights management" refers to the efforts to control and manage the dissemination of the contents. In order to understand these efforts, it is very important to understand the framework underlying them. This framework derives from the copyright laws and the philosophical support for them. This philosophical support forms the basis for certain copyright law exceptions which are examined in Section 2.4.1.

Most of the DRM technologies control distribution of contents to enforce rights models [40, 32, 7, 10, 43], and they are variation of DRM reference architecture discussed in Section 2.2.

2.2 DRM Reference Architecture

The DRM reference architecture has been extensively discussed by Rosenblatt et. al. [38]. There are three major components of this reference architecture, as seen in Figure 2.1: the *content server*, the *license server*, and the *DRM client*. In this system, users are granted specific rights to the information. The DRM reference architecture process flow consists of the following, as discussed by Erickson [11, 12]:

1. The user receives the contents through some mechanism, such as download from a web server, or file transfer from a remote machine. The content is bundled together with content metadata (e.g., ISBN, price, file format, author) in an encrypted package.

2. The user requests use of the content, for example, by choosing a menu item in a rendering application. The request is passed to the *DRM controller*, which then determines through the policies bound to the content package that the requested use needs authorization from the license server.

3. Before making the rights request to the license server, the DRM controller might gather information, such as, the rights requested by the user, the identity information, such as user or device identification, and the information from the content package, such as ISBN metadata. It then sends the information

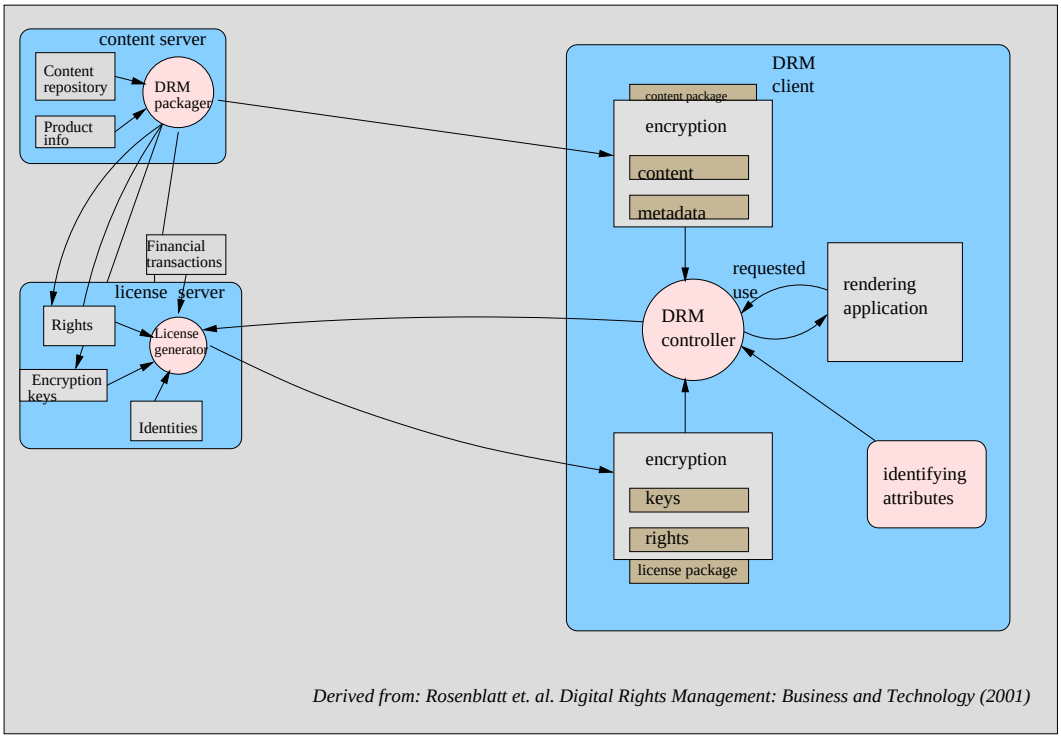


Figure 2.1: The DRM reference architecture

to the license server.

4. The license server authenticates the client's identity against its *identities* database, and looks up the content rights specifications using the content identifier. It processes the user's rights request to gather the rights information, and then, if the content specifications require, carry out a financial transaction. Finally, the license server creates a license using the rights information, identity information, and encryption keys, and securely packages it, possibly using encryption.

5. The license server sends the license back to the DRM controller that made the license request.

6. The DRM controller might authenticate the rendering application to make sure that it is authorized to render the contents. After the authentication is complete, and the license is received, it uses the license to open and send the contents to the rendering application for the particular requested use. The rendering application then renders the contents as requested.

In the reference model above, the DRM controller interacts with the content and license servers through content package and license respectively, along with rights request and other metadata information. As

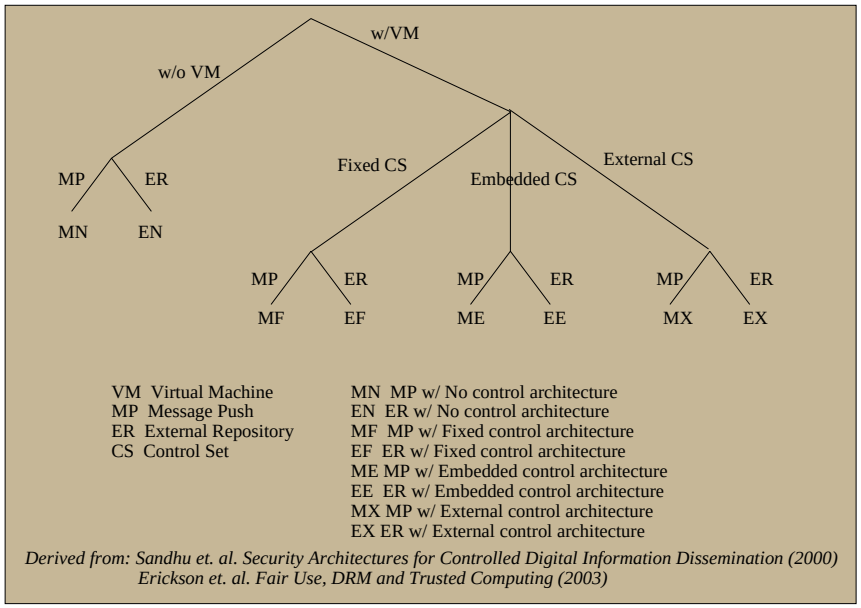


Figure 2.2: Classification of DRM architectures

pointed out by Erickson [11], this interaction is carried out using a *Rights Messaging Protocol (RMP)* , and the messages that make up the RMP are composed using the vocabulary of a *Rights Expression Language (REL)* [20, 23, 24]. Thus, any acceptable rights expression language must include the ability to express both the rights request and rights grant.

In the above reference architecture, usage of contents is controlled by the DRM controller. Encryption algorithms [27, 29] are typically used to protect the content and license packages. This is done to protect the integrity of the contents and to prevent the contents from being accessible in its native format at all time except when the DRM controller permits it. It is also the responsibility of the rendering application to allow the user to do only what the DRM controller permits.

2.3 Classification of DRM Architectures

Different variations of the above reference model exist, and they have been classified into different categories by Sandhu et. al. [35]. Figure 2.2 shows the taxonomy of DRM mechanisms. The categorization has been further discussed by Erickson [12] and is based on three factors :

- *Use of DRM client to control use of resources by users*: The DRM client (discussed in Section 2.2) is referred as VM (*virtual machine*) in Figure 2.2. The use of the resources is usually done through the rendering applications, such as Acrobat e-book, and the DRM client authorizes the use of resources by the rendering applications.

- *Enforcing Policies*: The VM implements the control specified by the policies, which are called *control sets* in Figure 2.2. The control set (CS) is used by the VM to control access and usage on resources, and is equivalent to "license" in Figure 2.1. They can be *Fixed*, *Embedded* or *External*. The fixed CS may be built-into the VM or their policy specification may be fixed. The embedded CS may be embedded or attached to the resource. In the DRM reference architecture, it is equivalent of having the license embedded in the content package. The external CS is external to the VM, and is the DRM reference architecture equivalent of external license package. Both fixed and embedded CS have limitations. In case of fixed CS, the originator can't change the policies once the VM is distributed. In case of embedded CS, policies can't be changed once the resource is deployed. The external CS is the most flexible of three, and allows for the policies to be managed since such management is external and separate from the VM and the deployed content.

- *Distribution of control Information and resources*: *Message push* (MP) and *External repository* (ER) are two possible distribution styles for resources and control information. In the message push style, the resources and information are sent to the users. In the external repository style, the users obtain the resources and information from a dissemination server on the network.

MN and EN categories as defined in Figure 2.2 are the DRM architectures without VM, and don't impose control on the use of resources by the users. The existing web-servers serving unencrypted HTML web pages belong to EN category, and typical e-mail newsletters belong to MN category.

Of the remaining categories in Figure 2.2, MX and EX are most flexible because of external CS, which allows for the separation of policies and resources. The DRM systems may implement a combination of embedded and external control sets.

2.4 Copyright and DRM

While DRM systems are sometimes referred to as mechanisms for enforcing copyright, DRM systems can go far beyond copyright [41]. While DRM systems can certainly prevent illegal distribution and copying of copyrighted work, they can do more; they can as easily prevent the copying and distribution of public-domain work as copyrighted work. DRM systems can control the private display and performances of the copyrighted work, even though the copyright law confers on the copyright owners the right to control only the public display and performance.

Given that DRM systems allow copyright owners to exercise far more rights than the copyright law allows, they are not really digital rights management systems. They behave more like "permissions management" systems, where they may allow particular use of the contents given permissions for the use. If they were designed for digital rights management, then they would allow the users to express their rights under copyright law too. Certain exceptions in copyright law, and the current state-of-art in computer science make this difficult, as explained in the following section.

2.4.1 Copyright Exceptions

The relevant copyright exceptions have been discussed by Samuelson law clinic [25] in their submission to OASIS Rights Language TC , and are reproduced here.

The Copyright Act limits the rights of an author on his or her own work through some exceptions. Some of the exceptions that are relevant to the DRM system policies are given below.

> "**Fair Use (17 U.S.C. 107)** The fair use of a copyrighted work, including such use by reproduction in copies or phonorecords or by any other means specified by that section, for purposes such as criticism, comment, news reporting, teaching (including multiple copies for classroom use), scholarship, or research, is not an infringement of copyright."

Section 107 enumerates the following four non-exclusive fair use factors that must be analyzed to determine whether the particular use of a copyrighted work is a "fair use" of that work:

1. The purpose and character of the use, including whether such use is of a commercial nature or is for non-profit educational purposes
2. The nature of the copyrighted work
3. The amount and substantiality of the portion used in relation to the copyrighted work as a whole
4. The effect of the use upon the potential market for or value of the copyrighted work

> " Section 107 draws attention to certain kinds of uses "criticism, comment, news reporting, teaching scholarship, or research" that weigh in favor of a finding of fair use. Section 107 presents four broad factors rather than bright-line (rigid) rules. Fair use analysis therefore requires a fact-intensive, case-by-case approach. This inquiry is necessary to set the correct balance between the exclusivity of a Copyright and the public interest in being able to freely discuss others' works. These four factors are non-exclusive, leaving courts free to consider other factors in determining whether a use is fair."

Example: Bob is writing literature review for a non-profit magazine, and he wants to critique a literary work as part of the review. He quotes a small portion from the work that he is critiquing, to explain one of his points in the review.

Example: Bob owns a collection of movies on VHS tapes, and he wants to share one of the movie scenes with his friends. He wants to make a copy of that scene on a blank VHS tape, and share it privately with his friends.

> "**Reproductions by Libraries and Archives (17 U.S.C. 108)** Publicly-accessible libraries and archives are allowed to make one copy of a copyrighted work, as long as the reproduction is not for direct or indirect commercial advantage. Libraries may make three replacement copies of a damaged or deteriorating work when copies of that work are not available at a fair price or are available in an obsolete format."

Example: A public library has archived educational documentaries on the VHS tapes. The tapes are degrading with time, and the documentaries stored on them will be irretrievably lost unless they are transferred to a new

medium. No other known copy of these documentaries exist and so, the library must transfer them to an alternative medium to avoid losing them. The library decides to migrate the documentaries to the DVD format, and store them on the DVD disks for longer shelf life and for preservation of quality over time. These documentaries are available to the library patrons and general public for viewing at no cost in the library's in-house media center.

> "**First Sale (17 U.S.C. 109)** Once a person lawfully obtains a copy of a work, she "is entitled, without the authority of the Copyright owner, to sell or otherwise dispose of the possession of that copy" (emphasis added). This limitation on Copyright exclusivity applies to everyone who lawfully acquires a work, not just to libraries or non-profit organizations."

Example: Chris has a collection of books, including the "Harry Potter" series. The local children's library is soliciting donation of the children's books from the community. The library doesn't have the "Harry Potter" collection, and so, Chris decides to donate his collection of "Harry Potter" books to the library.

Example: Alice belongs to a book club where the members read the books, and share their reviews and comments on the books. One of the book club member has "The Art of The Fellowship of the Ring" book which Alice hasn't read and wants to read. Alice has "The No. 1 Ladies' Detective Agency" book which that member will like to read. So, they do a swap of these books, and lend it to each other for a few days.

> "**Exemption of Certain Performances and Displays (17 U.S.C. 110)** Under many circumstances, the public display, performance or transmission of a work does not constitute Copyright infringement. Teachers and students, religious organizations, persons performing before blind or otherwise disabled audiences, and many other non-profit groups may perform or display copyrighted works without infringement. Finally, even commercial users, such as restaurants and stores, may perform and display copyrighted works, within statutorily defined space and amplification requirements."
>
> "**Secondary Transmissions (17 U.S.C. 111)** Many of the performance and display exemptions in Section 110 also apply to secondary transmissions of copyrighted works. Section 111 allows music stores and video stores to perform works in their stores, and also grants hotels permission to relay broadcast television signals to guests' rooms. In addition, this Section provides statutory

licenses for certain kinds of secondary transmissions, such as cable television."

"**Ephemeral Recordings (17 U.S.C. 112)** Ephemeral recordings are permitted under some circumstances. Broadcasters, for example, may make one copy of a sound recording that is being broadcast, for local transmissions, security, or archival preservation. Archival copies may be preserved indefinitely. Non-profits and governmental bodies have additional rights of replication and distribution."

"**Computer Programs (17 U.S.C. 117)** It is not an infringement of Copyright for the owner of a copy of a program to make a permanent backup copy of the program. It is also not an infringement to make temporary RAM copies, or to make temporary copies for restoration during computer maintenance."

"**Reproduction for Blind or Other People with Disabilities (17 U.S.C. 121)** Authorized non-profit and governmental agencies are allowed to make copies of published works in specialized formats exclusively for use by blind or other persons with disabilities."

2.4.2 Coding Fair Use Exception in DRM

While the above exceptions need to be coded in a REL to enable expression of the rights permitted by copyright laws in DRM policies, the fair use exception presents biggest challenge of all, because of its subjective and vague nature. As Felten [15] point out, the legal definition of fair use is vague from the perspective of a computer scientist. No enumeration of fair use is provided and there is no precise algorithm for determining fair use as the law leaves the responsibility of making fair use decisions based on the four factors to the judges. The law does not specify the evaluation mechanism for evaluating these factors, or weighing them in determining whether a use is fair or not. The law makes the fair use a judgment call to allow the fair use doctrine to evolve with technological innovations and progress.

If the fair use test were to be coded into a DRM system, such a system would have to apply four-factor fair use test to the attempted use of a work. The test is hard due to two main factors discussed by Felten [15] :

- *Lack of contextual knowledge*: The test requires knowledge about the circumstances of use, but such knowledge is not available to the DRM system; for example, a certain use may be fair when done

for non-commercial educational purpose, but illegal when done for commercial purpose. The DRM system has to know about the circumstances outside the computer to decide whether the setting could be classified as teaching or as commerce.

- *Lack of adequate artificial intelligence*: Even if full contextual information was available, applying the four-factor fair use test would require highly sophisticated AI. Several of the factors involve "AI-hard" problems. For example, the fourth factor involves judging the effect of the use on the market for the work. It requires reasoning about the economics of the market, a task which is difficult even for the well-trained humans. No computer system may be able to approach the ability of humans to analyze the market in the near future.

A DRM system that gets all the fair use judgments correct would make prediction of the fair use judgments, predicting accurately how a real judge would make such judgments in the lawsuits. Currently, technology for such an implementation doesn't exist.

Even if the current technologies can't make accurate fair use judgment in every case, perhaps they could approximate the law, and get the judgment right most of the time. But, in DRM systems, usually automated REL-based policy evaluations are used for decisions on the authorization requests by the users for access to resources. So, deciding whether a requested use by the user is fair use or not requires automated policy evaluation of such authorization requests. As discussed by Erickson [12]:

> "Only those policies that can be reliably reduced to yes/no decisions can be automated successfully. Access control policies that fit within narrow application domains (such as the handling of confidential documents within corporations) are well suited to automated policy enforcement; policies that are subject to many exemptions or based on conditions that may be indeterminate or external are difficult or impossible to automate with DRM."

Accommodating the fair use concept in automated policy evaluation is difficult because of the broad nature of fair use factors. The factors such as market for the work, and nature and context of use are very subjective and not easily measured. It is difficult to codify these factors, and in practice the evaluation algorithm will have to either ignore these factors or approximate them.

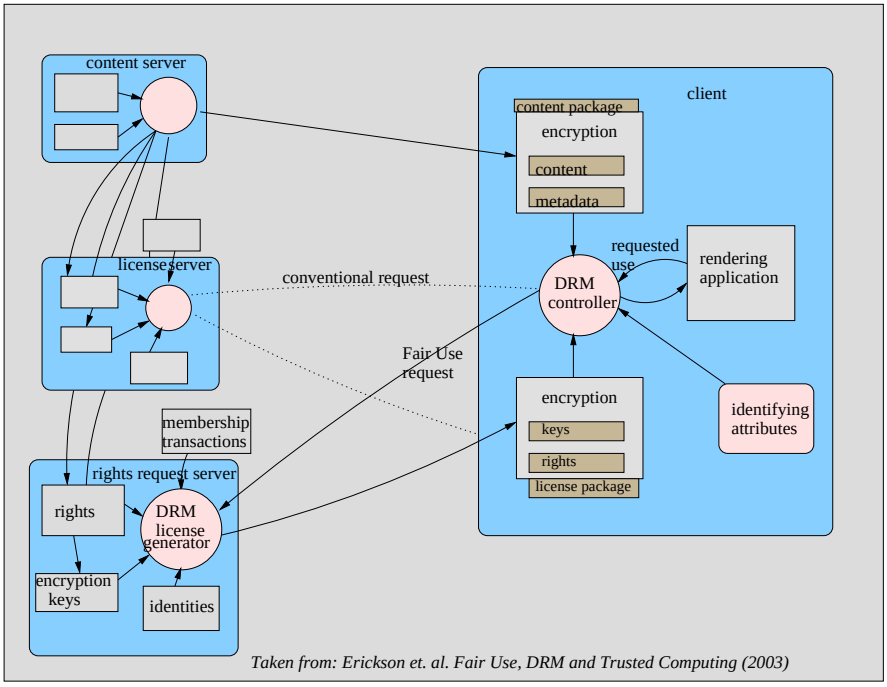


Figure 2.3: The DRM architecture with Third Party Rights Service

2.5 Accommodating Fair Use Concept in DRM

2.5.1 Third Party Rights Service

As we discussed before, the issue of coding fair use exception in DRM systems is a hard and open problem. The hardness of this problem has been acknowledged by Burk et. al. who proposed key escrow [2] as a step towards solving the problem. In their scheme, users might apply for keys for access to encrypted work for fair use purposes, and the human decision makers at the key escrow service decide on their requests. Erickson [12] have expanded on it to introduce a Third Party Rights Service in the DRM reference architecture as shown in Figure 2.3.

In that model, the Third Party Rights Service offer an impartial authorization authority while taking advantage of the current generation of DRM system architectures. The users can request new licenses to include other uses that are not part of the original licenses, and the Third Party Rights Service serves as an arbitrator in deciding whether such uses are fair or not, before granting the license requests from the users. It also shows the potential of the connection between the DRM client and the licensing server for rich dialogue beyond

today's simple permission requests.

Spontaneity and anonymity are two main issues involved in fair use requests [2] and have to be considered in the lower-level design process of the above model. A number of fair uses are spontaneous, such as using a portion of a published paper in course homework, and considerable social benefit accrue from this sort of unplanned and spontaneous use. Anonymity is the current default of fair use access in physical domain - a copyright holder does not need to know who has made use of the work, or at what time, or for what purpose. More generally, there exist a wide range of situations, for example in the case of a parody or a negative critique, in which the user may prefer to remain anonymous. Requiring parodists or other fair users to apply for access to any third party with identifying information may chill such uses, and is also a good argument in favor of impartiality of third party.

2.5.2 Importance of Fair Use in Academic Settings

While commercial DRM systems [26, 7, 13, 43, 36] can enable the copyright owners to enforce the rules beyond those allowed by copyright laws, in academic settings, balancing the interests of copyright users and owners can be an important issue. While the Third Party Rights Service has been proposed as a solution, there is a lack of working implementation of such a service. So far, the Third Party Rights Service has remained a high-level concept, with little open research on the components that make up this service, and of the issues involved in the design of such a service.

The importance of accommodating fair use in DRM systems within academic settings can not be overemphasized. Fair Use is critical to research and education, one of the core activities in academic community [5], including Dartmouth College. Fair Use is an important tool that promotes public access to ideas, information and work of authorship in a number of useful ways. The most common form of fair use is the ability of an author to quote from previous work (thereby copying a small part of it) in order to comment on it, or to promote further knowledge based on it. Such uses provide for a foundation on which new artistic and scholarly works are developed. The non-commercial aspects of fair use have also enabled wide access to copyrighted material in classroom settings, including privileges to display or perform such materials in the course of face-to-face teachings in the classroom.

The above forms of fair use (and other copyright exceptions) derive from the constitutional provision for

intellectual property protection with the pragmatic goal of promoting the public interest in access to knowledge and innovation [5]. The Constitution grants the Congress the power "to promote the progress of science and useful arts, by securing for limited times to authors and inventors the exclusive right to their respective writings and discoveries". The Supreme Court considered the concern with access to information in *Sony Corp. v. Universal City Studios, Inc.*[1]:

> "*As the text of the Constitution makes plain, it is Congress that has been assigned the task of defining the scope of the limited monopoly that should be granted to authors or to inventors in order to give the public appropriate access to their work product. Because this task involves a difficult balance between the interests of authors and inventors in the control and exploitation in their writings and discoveries on the one hand, and society's competing interest in the free flow of ideas, information and commerce on the other hand, our patent and Copyright statutes have been amended repeatedly.*"

Thus, maintaining a balance between the interests of copyright owners and copyright users is very important, especially in academic environments. DRM systems should try to strike this balance when being used in academic settings. If they were to provide the copyright owners greater control over the use of their works than they are entitled under copyright law, then the balance may shift more in favor of copyright owners. On the other hand, if they were to provide them with less control than entitled under copyright law, the balance may shift other way.

2.5.3 Academic DRM: Shibboleth Project

Shibboleth is an Internet2/MACE project [8] for inter-institutional sharing of resources that are subject to access controls. The involvement of universities in the project has made Shibboleth a potential framework for controlling cross-organizational access to resources among universities and other resource providers.

Shibboleth belongs to the EN category of DRM taxonomy discussed before, as it doesn't have a DRM client. In Shibboleth, once the user has access to resources, he/she can make use of them as they see fit. This design is not by accident, and comes partly from the recognition that users should be allowed to make use of

[1] 464 U.S. 417 (1984)

the contents as they see fit, in the interest of knowledge and innovation. So, while Shibboleth has a strong leaning towards the research and education interests of academic users, this excludes the use of resources that copyright owners won't allow to be *shibbolized* for fear of copyright infringement. In order to shibbolize the resources where copyright owners have wide commercial interests, the interests of copyright owners have to be balanced against the copyright users for these resources in Shibboleth. The Third Party Rights Service provides us with a mechanism to do this, after design changes are made in Shibboleth to accommodate the DRM client.

The evolving design of Shibboleth also makes it very attractive for modification to introduce the Third Party Rights Service. Also, as I will discuss later in Chapter 5, the Handle Service and Attribute Authority components in Shibboleth make it very suitable for a Third Party Rights Service. I discuss the architecture of Shibboleth and the modifications to introduce a Third Party Rights Service in Chapter 5. I briefly discuss some implementation issues for the Rights Service in Section 2.5.4.

2.5.4 Implementation Issues for Third Party Rights Service

Third Party Rights Service (referred to as TPRS) has so far remained a high-level concept. No known research is available on the architecture of TPRS, and so, I investigate the architecture of TPRS in this section. As seen in Figure 2.3, two of the main components of Third Party Rights Service are: the client and the server. On the client side, the user makes the fair use requests through the DRM client to the license server which serves as the server side of the TPRS. As I discussed before, the human mediators process these requests on the server side, and they need information about the four factors of fair use for processing them.

I look at an example transaction to explain the issues involved in the design of the TPRS. Suppose Bob has a license for a journal that allows him to view or print the contents. He attempts to copy some excerpts from the journal so that he can use them for his work. The DRM client detects that the license policies don't allow this use, and so, it intervenes, suggesting that Bob contact the TPRS from the list of the TPRS that is enumerated in the license. Bob believes that his use is fair use, and so, he applies for the zero-cost license from the TPRS (from the list) using fair use contention. Bob should supply the TPRS with his request and the information supporting his fair use contention in a language that they can understand. The TPRS should process the request while trying to meet the spontaneity and anonymity requirements of fair use. After it

processes the request, the resulting zero-cost license has to be sent to Bob in case the decision is positive (in case of negative decision, the decision is communicated to Bob). Further, the license has to be in a form that Bob's DRM client can understand and enforce. So, the TPRS has to find out if Bob's DRM client has the required capabilities and tailor the license accordingly. So, if the DRM client doesn't have the capability to do selective copying of excerpts as requested by Bob, but does have the capability to do selective copying of pages, TPRS might issue a license to copy a single page or few pages corresponding to the excerpts. The license issued by the TPRS is a technical license [12] that contains the authorization policies needed by the DRM client to permit the requested use.

From the above example, three main components of fair use request process follow:

Component 1. *Composing the fair use request*: The user should supply the information for four factors of fair use which the TPRS will need for its decision, in a form that the TPRS can understand. Also, the resulting license will be used by the DRM client to provide the granted rights to the users, and as I have discussed before in Section 2.2, the license will be composed using a REL. The REL provides the mechanism for expressing rights information and usage control policies in the license. So, the rights request step in this component and the license grant step in Component 3 should be meshed in a way that translates the user's request into a suitable REL-based license format.

Component 2. *Sending the request and receiving the response*: A mechanism is required to discover the TPRS, and after the TPRS is discovered, the request should be sent to the TPRS over a transport layer, such as SSL [27]. The response to the request may be sent to the user through the transport layer to their contact address, which should be communicated to the TPRS at the time of the request.

Component 3. *Processing the request and sending the response*: After the TPRS receives the request, it may assign it to human mediators for processing, while assigning priority to the spontaneity and anonymity requirements of fair use during processing. Before a license is granted, it may need to know the capability of the DRM client to handle the policies specified in the license. A mechanism should be provided for communicating these capabilities to the TPRS, so that a suitable license could be granted for the user's DRM client. The result of the processing, either a license or a refusal, is sent to the user.

I focus on devising a mechanism to compose and send the requests to the TPRS, and to smoothly mesh the rights request and license granting steps for translating the user's request into a suitable license format for

the user's DRM client if the license is granted. To meet this objective, I design a *Fair Use Request Interface* to provide the copyright users with a tool for composing fair use requests, and sending it to TPRS. I also devise a load-balancing technique to assign the requests to human mediators while taking into account the spontaneity requirement for request processing. I discuss the Fair Use Request Interface in Chapter 3. In Chapter 4, I discuss the load-balancing technique.

Chapter 3

Fair Use Request Interface

3.1 Introduction

In this chapter, I discuss the Fair Use Request Interface (referred as FURI from now on). The Fair Use Request Interface is a user-side component for making the fair use requests to the Third Party Rights Service.

The FURI is an interface that provides the rights users with a mechanism to compose, express and send fair use requests to the TPRS in a format it can understand. The primary design goal of the interface is to provide a mechanism to the users to express the rights requests under fair use exception, and to provide sufficient details to the TPRS to help them in making decisions on these requests. The four broad and subjective factors for fair use decisions [38] are explained in more detail below:

1. The purpose and character of the use
    - Is the use for non-profit, educational or personal purposes?
    - Is the use for criticism, commentary, news reporting, parody and other interpretive use based upon the copyrighted work?
    - Is the use for commercial purposes?
2. The nature of the copyrighted work

- Is the work factual? Is it published?
- Is the work a mixture of fact and imaginative work? Is it purely imaginative? Is it unpublished?

3. The amount and substantiality of the portion used in relation to the copyrighted work as a whole
   - Is the taken portion a small portion of the whole work?
   - Is the portion a greater amount of the whole?
   - Could the taken portion be considered the 'essence' of the work?
4. The effect of the use upon the potential market for or value of the copyrighted work
   - Is the original out of print or otherwise unavailable?
   - Is there a potential market for licensing use of the work?
   - Would the proposed use affect the sale of the original work?

As I discussed in Chapter 2, these four factors of fair use are highly subjective and vague.

A user may insist on making full use of the fair use rights as granted by the copyright law. When the user makes the rights request to the TPRS under fair use, he/she may agree to the policy of exercising only these requested rights, and other (default) rights that are granted in a license [38] by the TPRS, if it is a more convenient way of obtaining these rights under the fair use contention. The policies specified in the license should be enforced by the DRM client if required, and so, the secondary goal of FURI is to provide a mechanism to mesh together the rights request and the license granting steps for smooth translation of user's rights request into a suitable REL-based license format for enforcing these policies.

In Section 3.2, I explain the information requirements of four factors of fair use. I then devise and discuss an information model for fair use factors in Section 3.4. I use this information model to consider before-mentioned primary and secondary design issues of FURI in Section 3.5. In that section, I propose a tag language and interface design to solve these design issues. Section 3.6 analyzes some of the FURI issues with respect to the proposed design. I present some conclusions in Section 3.7.

3.2 Fair Use Considerations

It is important to understand what copyright holders look for in granting permission requests. The “Copyright Primer” [31] states the grantor might need the following information for a new work:

1. Title, author and publisher
2. Publication date
3. Edition - e.g. text or trade, and/or volume number
4. Binding - e.g. hardcover/softcover
5. Number of pages
6. Proposed selling price
7. Market - territory of distribution (e.g. US & Canada, World, etc)
8. Languages and media formats
9. Information on distribution if the process includes scanning, digitizing, or other methods that require electronic rights
10. Quantity of first printing
11. All supplementary materials that will include the requested material (teacher’s editions, etc)
12. Detailed information about the material being requested (figure, page, etc)

The above list captures the essence of what a copyright holder might look for in a permissions request, namely:

- What work of mine is used
- What is it used for
- What is the market for it

While the fair use requests that fall within the permissible requests by copyright holders can be permitted by the TPRS, the perspective of copyright holders (along with copyright users) on permissions may be used by the TPRS to make decisions on other fair use requests. So, the information supplied by FURI should try to answer the above questions.

3.3 Existing Work

Figure 3.1: Copyright Direct Rights Request Interface (from Erickson [9])

Little work is openly available on fair use request interfaces. One of the existing works on fair use request interface is by "Copyright Direct Service" from "Yankee Rights Management, Inc." [9]. It was provided to produce either an automatic response, based on certain conditions that were met, or as the result of escalation (when the rights template sent the request to *permission manager* who made a decision, and added license

details into the system). In both cases, the user went through a set of questionnaires in a browser to specify the settings of use, purpose, portion etc. as shown in Figure 3.1. It used a decision tree to make decisions, and the human mediator was part of the decision tree (for the cases where human intervention was required in decision making). The conditions for decision making were stored in the rights template.

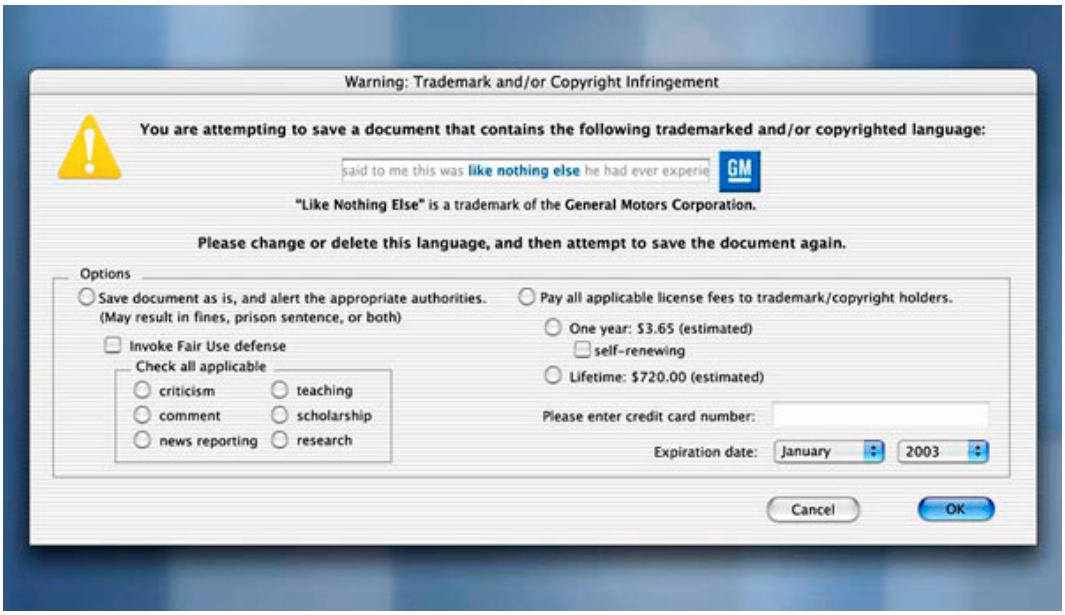


Figure 3.2: The fair use defense interface (from Hoberman [19])

Another related work that I have come across is one by Hoberman [19]. That work was done as part of his satirical take on copyright infringement, called “Infringement Series”. Figure 3.2 shows his work. I note that while his work shows an example of how a fair use defense interface will look like, the interface shown in the figure is not a real interface. It is his interpretation of copyright infringement in a satirical take where a fair use defense has to be invoked for a common phrase “like nothing else”. He created a set of Mac OS X dialog box based interfaces, one of which is shown in the figure, which model a scenario in which saving a document through the word processor forces the text through a built-in copyright scanner. If the user has been negligent so as to use a copyrighted phrase such as General Motors Corporation tagline “like nothing else”, the shown interface pops up to ask that the user either pay a fee for the use of that “copyrighted work”, or invoke the fair use defense (while alerting the appropriate authorities) using one or more of the following six use scenarios: criticism, comment, news reporting, teaching, scholarship, or research. His work, while satirical in nature, provides an example of fair use defense interface.

3.3.1 Need For a Mechanism To Generate FURI

The above Copyright Direct Service work while providing FURI for specific copyrighted materials, lacks the mechanism for generating customized FURI for any copyrighted material. The wide variety of contents such as books, journals, movies, have different requirements for determining fair uses. The factual information for determining fair use cases for a Disney movie may be different from that of a home-made movie. For example, the market for a Disney movie may cover both US and international segments, while the market for a home-made movie may be limited to the area around a local town store. While the FURI works discussed above focus on some very specific scenarios, they leave open two questions:

1. how to determine what factual information is needed (from the fair use requesters) for the fair use determination for any use of any copyrighted work
2. how to generate the appropriate FURI to ask for the above factual information from the fair use requesters for the requested fair uses

A model that models the information needs for fair use factors, and divides the information into the component that should be requested from the users, and the component that should be determined by TPRS can answer the above questions. In the next section, I design and present an information model to divide the fair use information into above two components. I will later use that model to generate FURI from the content licenses.

3.4 Information Model for Fair Use Factors

3.4.1 ODRL Rights Model

When a party makes a request for rights on a particular asset, it should be conveyed using a REL because of rights expressions involved in the request. I use the ODRL language [20] for rights expression. The relationship between the core entities of the parties, the rights and the assets is modeled in the ODRL rights expression model as shown in Figure 3.3. The parties include rights users and rights holders.

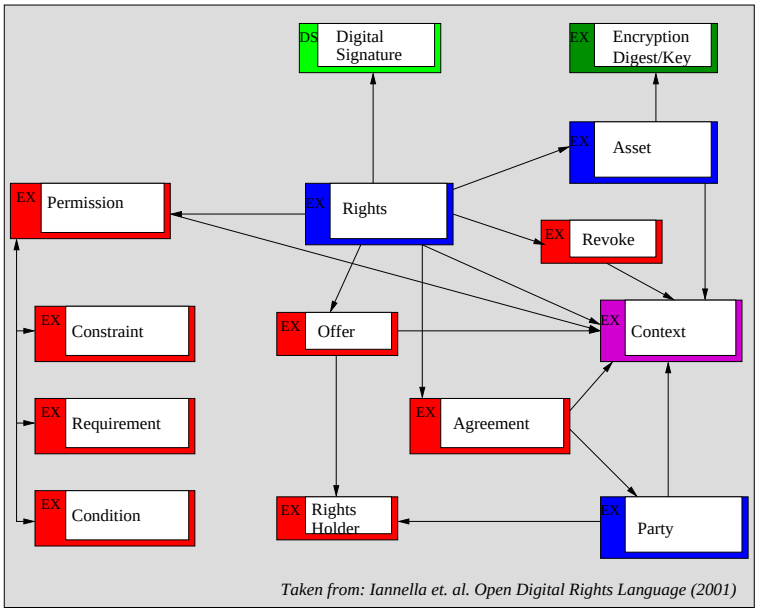


Figure 3.3: ODRL rights expression model (from *W3C ODRL specification version 1.1*)

The assets include any physical or digital content. They should be uniquely identified, and may consist of several subparts, in different formats. The rights include permissions, which can then contain constraints, requirements and conditions. Permissions are actual usages or activities allowed over the assets. Constraints are the limits to these permissions. Requirements are the obligations needed to exercise the permissions. Conditions specify exceptions that when true, expire the permissions. For example, a book (asset) may have the "view" permission with the constraint of viewing it for one year period, and a requirement to pay $5 every time the user views the contents. After one year from the date of grant, the permission to view the book expires.

Thus, in the ODRL model, the rights holder can *offer* certain rights to the end-users. In my work, I use the ODRL REL for expressing the rights and associated metadata in the licenses for the contents. In the next section, I design a fair use information model, which is based on the ODRL rights expression model discussed above.

3.4.2 Fair Use Information Model

The Fair Use information model that I present here models the information needed by the four factors of fair use in terms of assets, rights, and other components. While the information needed by these factors is broad and diverse, the information model helps us in analyzing the information requirements and the information that may be needed by the TPRS in making decision on requests. There are two main components of information: REL-based information which has the advantage of being standardized by virtue of use of a standard REL (ODRL here) vocabulary, and non-REL based information which may not submit to such standardization.

The fair use information model as shown in Figure 3.4 consists of two components:

- Component 1 is modeled on ODRL rights expression model. It supplies information for factors 1 and 3 of fair use (for a particular asset) listed in section 3.1. The *use* is modeled as a right with associated permissions, constraints and conditions. The *purpose and character of use* may be for non-profit, educational, personal, criticism, commentary, parody or other "transformative" uses, or commercial. These details can be provided in the *use details* for the rights, and *other use details* as shown in the component. For *amount and substantiality of portion*, two factors are important: the amount of portion user wants to use, and the relative amount of that portion. In the above component, the amount of portion can be determined from the rights request, e.g., copying seven pages of a published work. The information on how critical the portion is to the essence of the work requires subjective knowledge of the work [2], and it is left to the TPRS to determine and evaluate this information. *Context* information provides other context information that may be needed to evaluate the above factors, such as, publisher, territory of distribution, number of copies printed, number of pages etc.

- Component 2 models information for factors 2 and 4 of fair use. These factors are very subjective, and need *external information* together with *context information* of the asset. The information needed for these factors is usually known by rights holders. For *nature of copyrighted work*, some of the determining factors are whether the work is factual, published, a mixture of fact and imaginative work, purely imaginative or unpublished. If the work in question presents factual material and has been widely published, the balance might be in favor of a claim of fair use. On the other hand, if the work is a combination of factual materials and creative work, the case for fair use might be weak. The

effect of use on potential market for or value of copyrighted work usually weights all the previous factors in relation to the access to and market value of the work. For example, access to the work in an established market for sales or licenses might weigh heavily against fair use, and can outweigh the cumulative effect of all the other considerations. A use that competes with sales of the original or avoids licensing fees might not be fair use. Because the information involved in determining these factors requires external knowledge, such as knowledge of market, together with knowledge of the asset, it is modeled using *external knowledge* and *context* in the component.

Component 1 models the information to be supplied by the rights requesters, and is used in the rights request interface design as discussed in section 3.5. The context information is included in the ODRL license, and so, is not included in the request interface, as it is assumed that the TPRS has access to the them. On the other hand, component 2 is rightsholder specific as discussed above, and so, it is left to the TPRS (which represents the rightsholder in making rights decisions) to implement that component through the mechanisms such as specialized knowledge, information database etc. that help it determine the information for factors 2 and 4 of fair use.

In section 3.5, I explain the design of rights request interface based on the component 1. I also discuss the interface design issues in section 3.6.

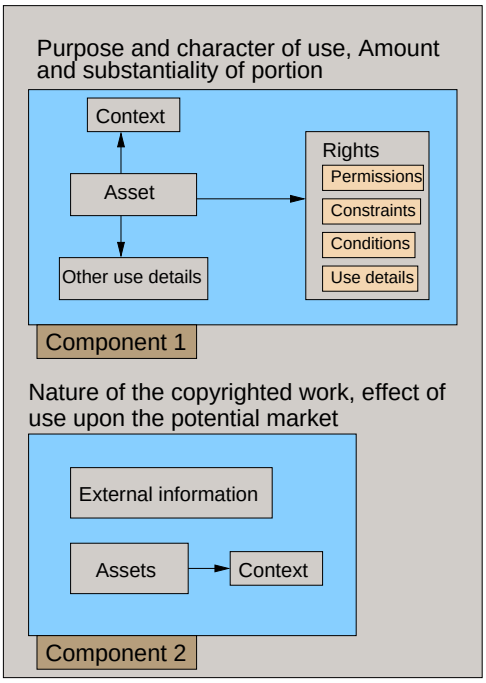


Figure 3.4: The information model consists of two components. In component 1, the factual information for the two fair use factors, namely, factors 1 and 3, for the use of an “asset” include the “Rights”, “Context” and “Other use details” as shown by the arrows. In component 2, “External information” and asset-related “Context” information is needed to determine the factors 2 and 4 of fair use.

3.5 Interface Implementation

3.5.1 Shortcomings of ODRL License

The Nokia content publishing tool is a stand-alone, off-line tool that can produce a DRM package consisting of contents and ODRL license for the content files. I use the Nokia tool to generate the ODRL licenses in my work.

Though an ODRL license may be used for the purpose of interface generation, some factors motivate the use of separate tags within ODRL license for specifying the request interface as explained below.

The ODRL license as generated by the Nokia tool consists of only those rights that have been granted on the content. So, if the printing rights haven’t been granted on the contents, it doesn’t appear in the license. Figure 3.5 illustrates the rights in a sample license for pdf contents for which only “play” rights has been granted. “print”, “save” and “annotate” rights haven’t been granted for that license, and as can be seen in Figure 3.5,

```
<rights>
<permission>
<play>
<select_count>1</select_count>
<start_time>2003 04 27 12 14 07</start_time>
<end_time>2004 04 27 12 14 07</end_time>
<interval>
<year>1</year>
<month>1</month>
<day>7</day>
<hour>4</hour>
<minute>5</minute>
<second>9</second> </interval>
</play>
</permission>
</rights>
```

Figure 3.5: Part of the ODRL License for a pdf document

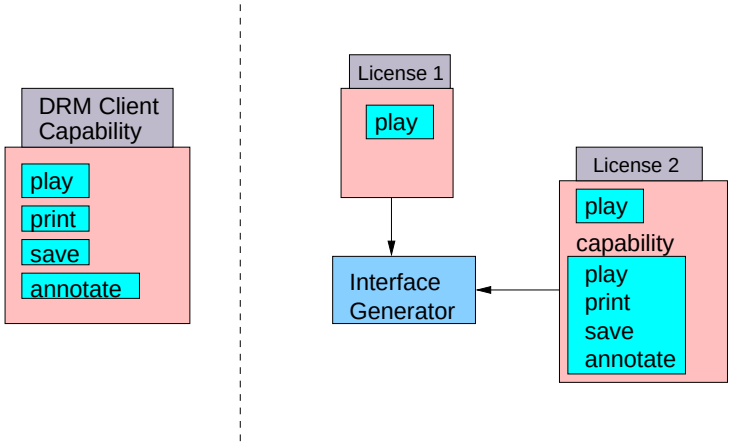


Figure 3.6: The DRM client can handle various rights. The license 1 has only *Play* right as only this right has been granted in the license. The dashed line indicates that the interface generator is unaware of the DRM client's capability. License 2 is same as license 1, but enumerates the capabilities of DRM client, which the interface generator can use to enumerate the possible rights, and present them to the users for making fair use requests

tags corresponding to “print”, “save” and “annotate” rights are absent from the license. Figure 3.6 illustrates it graphically. In that figure, the above license is shown as license 1. If we were to use this license for the request interface, the interface generator must have a mechanism for discovering the rights that are absent in the license, such as print rights, and which can be recognized by the DRM client. These rights may be needed by the users for requests to the TPRS. If the interface generator were to use license 1, then it may generate an interface with request for “play” right only, because no other rights are enumerated in the license, and the interface generator doesn’t have the knowledge of capabilities of the DRM client. On the other hand, if it were to use license 2, then it can enumerate all possible rights in the request interface, because of the enumeration of the DRM client capabilities in license 2.

Suppose the request interface has “Other Rights” field to make request for the other possible rights that aren’t listed in the interface. Then, the enumeration of DRM client capabilities is important for the cases when a user makes the request for the rights under the “Other Rights” field. Suppose the user Bob makes a request to the TPRS for “adding comments” to a content, and the TPRS decides to grant the license for that request. But, the rights granted in the license have to be recognized by Bob’s DRM client. Given the enumeration as before, the TPRS may determine that “adding comments” is equivalent to “annotate” right and grant it as “annotate” right in the license. On the other hand, if the TPRS determines that it is not one of the enumerated rights, then it knows that such a right can’t be granted in the license, as the DRM client is not capable of recognizing any right other than the enumerated rights.

Thus, if all the rights recognized by a DRM client are enumerated in the request interface, then when a request for a right is made to the TPRS, and if it was one of the enumerated rights, the TPRS will know that a license can be generated for this right and that it will be recognized by the DRM client. Those rights that are not part of the enumerated rights are not recognized by the DRM client, and can’t be granted through a license unless a different DRM client recognizing those rights is available. Thus, enumerating all possible rights in the interface is a simple mechanism to mesh rights request and license granting steps for translation of user’s right requests into license grant.

If the capabilities of the DRM client are listed in the license as shown in license 2 in Figure 3.6, then the interface generator can use this list to enumerate in the request interface all the rights recognized by that DRM client. This simple solution is based on the fact that in general, the DRM client capabilities remain relatively static, and so, if they have the capability to recognize the policies in a license at time *A*, they are

```
<usage_permission_types>
<permission type value value-type>
<constraints type value value-type> ... </constraints>
</permission>
</usage_permission_types>
```

Figure 3.7: The tag language for the interface

likely to retain the capability to recognize it at time B where $B > A$.

3.5.2 Tag Language for Interface Generation

I provide a small tag language for interface specification as shown in Figure 3.7. The tag language may be used to enumerate all possible rights recognized by the DRM client, so that they can be requested by the content users if they want to. The request interface can be specified within the license using the tag language, as explained below.

The `<usage_permission_types>` tag and the other tags within it are parsed by a parser and a rights request interface is generated. Within `<usage_permission_types>`, there are tags for possible permissions and constraints on the asset. Except for `<usage_permission_types>` tag, other tag names must correspond to *rights data dictionary (rdd)* definitions in ODRL for reasons that will be explained in section 3.6.1. The constraints can have either numerical values or string values (e.g., geographical location). I focus on numerical constraints here. The permissions and constraints constitute the possible rights that are recognized by the DRM client corresponding to an asset. The users can phrase their license request using the rights that have been embedded in these tags. The tags might have the attributes, *type, val,* and/or *val_type*. The *type* attribute has following possible values: *perms* or *constraint*. The *perms* attribute value tells the parser that it is a rights permission tag. The value of *val* attribute is the name the parser should put for that right in the request interface. The *constraint* attribute value is for constraints. The value of *val_type* attribute tells the parser about the values that constraint can take. When the value is *none*, then the constraint doesn't have a value, i.e. it is used to group other constraints, e.g. "interval" constraint in Figure 3.8. For constraints which have numerical values (I focus on numerical values only in this work), such as year, month, day, print count, the *val_type* attribute has *num* value.

```
<usage_permission_types>
<play type="perms" val="Play Rights">
<interval type="constraint" val="Duration" val_type= "none" >
<year type="constraint" val = "Number of Years" val_type="num"></year>
<month type="constraint" val= "Number of Months" val_type="num"></month>
<day type = "constraint" val="Number of Days" val_type="num"></day>
<hour type="constraint" val="Number of Hours" val_type="num"></hour>
<minute type="constraint" val="Number of Minutes" val_type="num"></minute>
<second type="constraint" val="Number of Seconds" val_type="num"></second>
</interval>
</play>
</usage_permission_types>
```

Figure 3.8: An interface specification example

Parser Implementation for Generating Interface

I have implemented the interface generator as a parser in Java. It takes a XML encoded license (including ODRL) containing the interface specification as input, and generates a HTML request interface as output. Figure 3.8 shows an example interface specification. The interface generator has two components. One component parses the tags, and the other component adds standard questions as explained later. The output of the interface generator for the example specification is shown in Figure 3.9. As seen in that Figure, for the *val_type* attributes with *none* value, the quantifications are left out. So, the `<interval>` tag doesn't have any quantifying box in the generator output, and is expressed as *Duration* as given by *val* attribute. The other tags following `<interval>` tag are grouped under it, and have quantifying boxes next to them. These constraints are grouped together under permission (which is "Play Rights" in this example), as specified in the Figure 3.8. Thus given interface specification in ODRL license in the form of permissions, constraints, and possible constraint values, the generator can generate the rights that can be requested by the users.

Addition of Standard Questions to Interface

The second component of the interface generator adds standard questions to the interface. Two of them are based on a schema outlined by Morris et. al. [31] for description of rights objects. In the schema described by them, *the purpose of use* of an object can be:

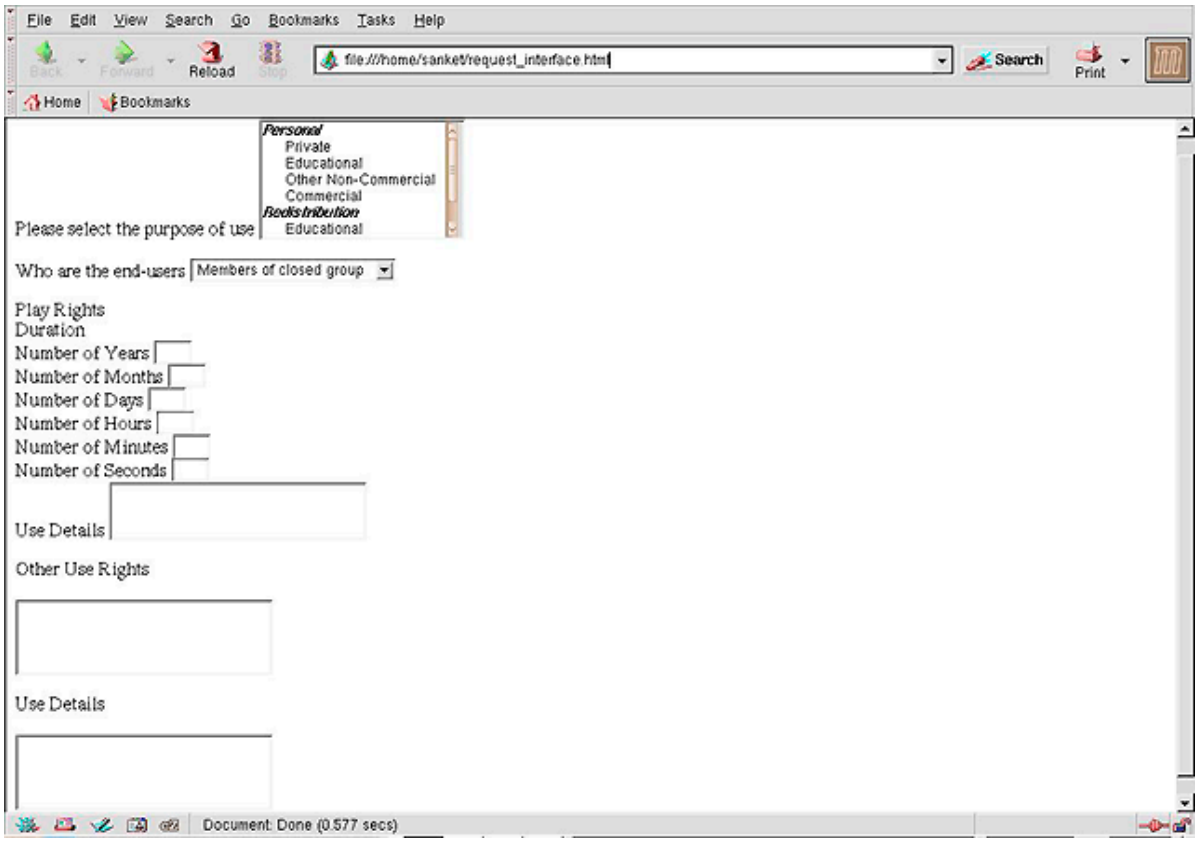


Figure 3.9: The Fair Use Request Interface

- personal (e.g. reading, printing, saving) subdivided into:
  - Private
  - Educational
  - Other non-commercial (e.g. public library)
  - Commercial (use in pursuit of commercial ends)
- Redistribution
  - Educational - e.g. course packs, electronic reserve
  - Other non-commercial - e.g. individual sending a copy to a colleague
  - Commercial - systematic or non-systematic use in pursuit of commercial ends

In order to evaluate purpose and character of use, it is also important to know who the end-users are, as they will be making use of the objects. According to Morris et. al., the *end-users* can be:

- Member of closed user group (e.g. authorized user within university of company)
- Member of open user group (e.g. customer of licensee, unidentified walk-in library users)

The above two factors are clearly very important in determining purpose and character of use (factor 1 of fair use), and are involved in all rights transactions on objects. I use the two questions based on these factors in the Fair Use Request Interface as seen in Figure 3.9. By including these questions in the interface, we also standardize some of the information that is provided by the users, and provide the users with some standard and familiar (that is, after repeated use of the interface) questions.

The users may also want to request the rights that are not recognized by the DRM client for the contents on which these rights are requested. For such cases, they can request these rights by using the *Other Use Rights* section of the request interface. It should be noted that if no DRM client capable of processing these rights is available, and if the rights are granted to the users, the contents might have to be provided to them in an unsecured format to allow the users to exercise these rights over the contents. The resulting issue of whether they misuse the freedom of usage of that content or not is left to the TPRS.

The *Use Details* section is provided to allow the users to provide other use details (separate from "rights use details") that might be needed for determination of fair use factors. For example, a user copying portion of a scientific article for converting it into a accessible format for disabled people might explain it in this section, and also explain how the use of the contents is for "non-commercial redistribution" purpose.

3.6 Design Issues

3.6.1 Phrasing and Parsing Rights Request

The request interface differs from the existing work as it is generated from the specification embedded in the ODRL license. The specification tags in the ODRL license define the interface as discussed in the previous

Usage Permissions (pertaining to the end use of an asset)			
Name	Identifier	Definition	Comment
Display	display	The act of rendering the asset onto a visual device	
Print	print	The act of rendering the asset onto paper or hard copy form	
Play	play	The act of rendering the asset into audio/video form	
Execute	execute	The act of executing the asset	For example, machine executable code or Java

Table 3.1: ODRL usage permission rdd elements

section. The `<play>` and `<render>` tags in two different licenses for a print content might refer to the same right, but they might be confusing for the rights request service unless the service has the knowledge of their meaning. Here is where the rights data dictionary [40] helps. The rights data dictionary (referred as *rdd*) provides a comprehensive framework and standard vocabulary for the expression of rights. ODRL uses a data dictionary [20] to define the elements within the ODRL language. The ODRL data dictionary elements form the basis of the language and can be extended by additional elements. The data dictionary uses five logical data models (Figure 3.3): permission, constraints, requirements, rightsholder, context. For illustrative purpose, some of the permission elements are defined in Table 3.1.

By using ODRL right tags for permissions and constraints within the specification, we can ensure that the rights request service can discover their semantics through ODRL rdd if needed.

The fair use request interfaces that we discussed in Section 3.3 are static, and so are the rights tags used in these interfaces. These static tags can be hard coded in the rights request service that process the requests from these interfaces. On the other hand, in our case, the request interface is generated in real-time from the specification coded in the ODRL licenses. The specifications in different licenses might contain different tags for same use, such as `<play>` and `<render>` tags in the two licenses in the example before. By ensuring that the rights (i.e., the permissions and constraints that make up rights) tags are derived from the ODRL data dictionary, we ensure standard vocabulary for parsing and processing of these tags by third party rights service. The tag semantics can be looked up the in the ODRL data dictionary by TPRS if needed, and new tags can be introduced by extending the ODRL data dictionary [20].

3.6.2 Discovery of Third Party Rights Service

The users need to know the location of the TPRS to be able to contact it and send their requests to it. I provide for a `<tprs_location>` tag to record the location of the TPRS. This tag is encoded within the `<usage_permission_types>` tag, and it contains the location in the web URL format, such as `http://www.dartmouth.edu/tsrp/submit_request`. Given the tag, the location of the TPRS can be determined, and the rights request encoded in the interface can be sent to it.

3.6.3 Approximating Vagueness of Fair Use

The fair use can be vague in real world. For example, Betsy might want to print certain pages from a political journal for her political education class. She would determine which pages she wants to print, after she has other course information. She might want to print five pages now, but later when she has more course information, she might want to print other pages from the journal. She doesn't have the information yet, and so doesn't know which pages she might need to print. But, if she were to make fair use request to rights request service for that journal, she will have to specify precisely in her request which pages she wants to print.

The rights requests are quantified precisely, as seen in Figure 3.9 before. For a fair use which is vague at the time of request, the vagueness might necessitate imprecise rights requests. But, the rights that are requested using the interface (other than *other rights*) have to be precise because the DRM client requires precise rights specification in the license, for controlling the content use.

The accommodation of imprecision in rights requests may be made by making multiple precise rights requests to the rights service. When the information whose absence led to a vagueness becomes available, the vagueness may be resolved and a precise rights request(s) may be specified. Thus, in Betsy's case, she can make multiple rights requests for printing, asking for permission each time she receives the information that enables her to determine the pages that are needed for the class.

Thus, while in the fair use interface, the vagueness in fair use could be accommodated through multiple (precise) rights requests to the decision service, the solution is cumbersome, and can be inconvenient for the users, chilling the spontaneity. The above discussion illustrates the issue of the gap between the intentions of

the users which can be imprecise, and the policy enforcement by the DRM systems which are precise. A better solution is needed to accommodate the unanticipated needs of the users that lead to the vagueness. Providing such a solution might also need a mechanism to provide a mapping between the purpose and intentions of the users, and authorization for future, undetermined uses. In Betsy's case, it may be a mapping between her reasons for printing pages from the journal and the limited multi-page printing capability to accommodate future printing needs.

3.6.4 Gap Between the Functions at DRM Client and the Needs of Users

The range of usage functions supported by a DRM client can be limited, and may not accommodate all the usage needs of the users. For example, a user wants to print a few excerpts from the non-consecutive pages of a journal. Suppose his DRM client doesn't provide the capability to selectively print only the excerpts, but does provide the capability to print pages. Then, the user will need to translate his usage need in the form of "print page" request with list of the specific pages containing the excerpts he wants to print. This illustrates the gap between the needs of the users, and the usage functions at the DRM clients. The user also may have to determine the suitable capability that approximates his needs, as in the above example where the user determines that "printing pages" is a suitable approximation for "printing excerpts". So, the conceptual problem of providing a translation engine at the FURI that takes the usage needs of the users as input, and translates it to appropriate rights requests also exists, and it needs to be solved.

3.7 Conclusion

In this chapter, I presented an information model for the fair use factors, and specified the tags for generating FURI from the licenses. The Fair Use Information model is derived from the ODRL model, and it models the information involved in determining the four factors of fair use. Since the ODRL licenses may not enumerate the capabilities of the DRM clients, FURI tag specifications are provided to embed FURI tags within the licenses. They can be used to specify the information, including capabilities of the DRM client, for generating the FURI. The Fair Use Information model can be used to design the FURI interface for

requesting the factual information for fair uses from the users. Some questions are also added to FURI to provide some standardization in the interfaces presented to the users.

Chapter 4

Load Balancing in TPRS

4.1 Introduction

While the FURI discussed in the previous chapter provides a mechanism to compose and send the requests to the TPRS, the request processing role of the TPRS too is very important. The TPRS and FURI are closely dependent on each other for successful composition and processing of the requests. The TPRS depends on the FURI for the expression of requests in the form it can understand, and the FURI depends on the TPRS for the timely processing of requests, so that the users can get results in the reasonable time frames. Thus, the TPRS has to not only handle and process the requests, but also ensure that it does so in a way that meets the spontaneity needs of fair use. If it takes too long in processing the requests, it may chill the use of the FURI for the fair use requests.

In the TPRS, the rights requests are processed by the humans. Once the requests arrive at the server in the TPRS, how these requests are dispatched for processing is a moot question. Given the requirement that these requests should be processed in minimum amount of time, the system of dispatching them to the request handlers becomes critical. While these requests can be fetched by the request handlers self-consciously from a centralized location (i.e. server), the performance of this system depends on the consciousness of the request handlers. If the involved handlers actively and self-consciously fetch the next request after they finish the current one, the system performance might be good. On the other hand, if such act is not performed,

some handlers may be very busy while others may be idle, leading to uncertain delays in processing. Fetching of requests by handlers with mismatched capabilities may lead to delays even when handlers with required capabilities are available.

The processing of fair use requests can depend on the organization structure of the processing organization (e.g. libraries), and may involve multiple issues such as describing the role of task handlers, calculating their load, dispatching a request, calculating the workload of handlers, tracking their work history etc. For example, if a certain fair use request was handled by a particular librarian, and many similar requests arrive in a short period of time after that, then that librarian might be in best position to process such requests quickly, having processed them before.

The handling of tasks also depends on the organization model. An organization model consists of the organization structure and the organization rules. The task handling depends on this model, and also the optimization goals which may consist of multiple goals such as task type, handling capability, workload of task handlers etc. For example, in a multi-library organization such as Dartmouth College Library Services, fair use requests related to humanities work may be handled only by the librarians at Humanities library, while the requests related to biological work may be handled only by the librarians at the Biological Sciences library. Even within each library, each librarian may have his own specialization that may give him an edge over others in processing the requests pertaining to that specialization. Further, these librarians may have multiple responsibilities, and so, the optimization goals of the libraries may be to process the fair use requests as soon as possible while devoting certain amount of time and "priority" to other responsibilities, such as answering email inquiries from patrons.

With the above issues in mind, I investigate a task assignment algorithm here using dynamic load-balancing to balance the loads among the task handlers while ensuring that the requests are assigned to the task handlers who can process them. I discuss the load-balancing problem in Section 4.2 and then examine a dynamic algorithm with respect to the problem in Section 4.3. In Section 4.4, I show the application of the algorithm to fair use request processing, and in Section 4.5, I discuss the simulation result for the algorithm. I present the details of process flow for fair use request process based on the load-balancing and FURI components in Section 4.6. Section 4.7 presents some conclusions.

4.2 Load-Balancing Problem

The problem of load-balancing of the fair use requests may be stated as follows: Given the task handlers, assign the request to the task handler who is "competent" to process it while maintaining stable and balanced load on all task handlers who are "competent" to process that request.

Given that the capabilities of task handlers can vary depending on various criteria such as their roles, responsibilities, skill level etc., a good load-balancing algorithm should be able to take into account the following factors:

- Organization structure and organization rules - which may determine which task handlers may be able to process a request, as in multi-library example above.
- Properties of the fair use requests - which may be defined by the type, purpose and character of rights requested, the amount of use, the person requesting it (such as student or faculty).
- Multiple goals of the request processing service - assigning priorities to request processing while handling other organizational work responsibilities

Traditional load-balancing algorithms [30, 28, 4, 6] have considered load-balancing in the cases involving computing units. In case where the task handling units are humans, the load-balancing factors are much more complex and varied as the examples above show. Song et. al. [42] have proposed a load-balancing algorithm (referred as *org_load_balance* in this discussion) for human task handlers in workflow systems, with dynamic load-balancing policy. *To the best of my knowledge, their algorithm is the first one to deal with human task handling in load-balancing architectures.* Their load-balancing algorithm however assumes that the task handlers are devoted to the tasks assigned by the algorithm, and doesn't take into account the load associated with other responsibilities that task handlers may have. In fact, the task handlers may have other work responsibilities, and ignoring it in calculating the total load of task handlers is a serious flaw in the load-balancing algorithm because its performance depends on the accuracy of the actual load of task handlers. In section 4.3, I correct this flaw by modifying the *org_load_balance* algorithm to accommodate the load associated with other responsibilities.

4.3 Dynamic Algorithm for Load-Balancing

I discuss the *org_load_balance* algorithm by Song et. al. in this section. That algorithm has three component sub-algorithms: *Task Assignment, Workload Prediction, Actor Assignment*. I modify *org_load_balance* algorithm by introducing the concept of *other workload* in the discussion below, and use this workload in the *Actor Assignment* sub-algorithm. The other sub-algorithms, namely *Task Assignment* and *Workload Prediction* are identical as in *org_load_balance* algorithm.

The task assignment to a person depends on his role and responsibility in the organization, and the organization structure. The organization structure can be defined in terms of role, actor and group as below:

- **Role:** An abstract representation that describes the characteristics of task handlers, denoted by R . The set of all roles in the organization forms a *RoleSet* . Each role differs from others by some characteristic attributes such as name, responsibility etc.. Role R can be described as an *n- tuple*, e.g.:

$$R =< Name, Responsibility, \ldots >$$

- **Actor:** A member of an organization, denoted by *A*. *ActorSet* represents the set of all actors in the organization. Actors may differ from each other by attributes such as name, job title, specialty etc. Actors can assume multiple roles, and can be described by an *n-tuple*, e.g.:

$$A =< Name, Job\_Title, Speciality, Department, \ldots >$$

- **Group:** Consists of several actors who work together for a user task and is denoted by G . *GroupSet* represents the set of all groups in the organization. The groups may differ from each other in terms of attributes such as name of group, task function of the group etc. A group can act in multiple roles, and can be described by an *n- tuple*, e.g.:

$$G= < Name, Task, NumberOfActors, \ldots >$$

Thus, actor is the basic unit of the organization. A group is used to group together actors, and role organizes the groups and actors.

There are two basic organization rules to check for the membership of an actor in a role and a group. The first rule $\in$ checks for the set membership. Thus, given a person P and role R, $P \in R$ checks if P belongs to role R. Given group G, $P \in G$ checks if the person P belongs to group G. The second rule combines the first rule expressions using set operations $\{\bigcap, \bigcup, -\}$ to perform more complex operations.

For example, the problem of choosing actors who are qualified for the role of *Humanities Librarian* can be phrased as:

$$Humanities\_Actor\_Set=\{a|a \in A \bigwedge a \in r \bigwedge r \in R \bigwedge r[name]=\text{`}humanities\_librarian\text{'}\}$$

The actors who are specialized to process the *Archiving requests* are given by,

$$Archive\_Request\_Actor\_Set=\{a|a \in A \bigwedge Archiving \in a[specialities]\}$$

The Humanities librarians who can process the archiving requests are given by,

$$Humanities\_Actor\_Set \bigcap Archive\_Request\_Actor\_Set$$

Dynamic Task Assignment Algorithm

The task assignment algorithm first finds the *ActorSet* for the task to be assigned, and if more than one actor are available, it assigns task to one of them according to the *Actor Assignment* algorithm which is discussed later.

Task Assignment Algorithm

1. Find all the actors who can act in the role (with associated conditions) required by the task T, and put them in the set *AvailableActorSet*.
2. If *Cardinality(AvailableActorSet)* $= 1$ go to step 4.

3. Choose optimal actor *a* from *AvailableActorSet* according to *Actor Assignment* algorithm.

4. Dispatch *T* to *a*.

Determining the processing times of tasks is crucial for load-balancing as it depends on the processing times of the tasks. By dividing the tasks in different types, we can use the task types to assess their processing times. A task requires a minimum organizational designation to handle it. For example, certain requests from Humanities faculty may be handled only by a person of assistant humanities librarian or higher designation. Let $Rank^q$ represent organizational rank *q*. I assume a hierarchical organization as shown in Figure 4.1, with rank assignment to the people in the organization. The figure also shows the role types and ranks play in the assignment of tasks in the organization. As seen in that figure, task types also determine which part of the organization they are assigned to, and for a particular type, the rank of the task determines which task handlers can handle it. Thus, in Figure 4.1, a task of type *a* and rank 2 can be handled by the nodes of rank 2 or higher in Humanities library. That task can't be handled by the node of rank 2 in Sciences library because that node can't handle type *a* tasks. Thus, the task type not only determines the processing time, but also the task handlers from the set of task handlers who have the rank equal or higher than the rank requirement of the task. I use the terms "actor" and "task handler" interchangeably here.

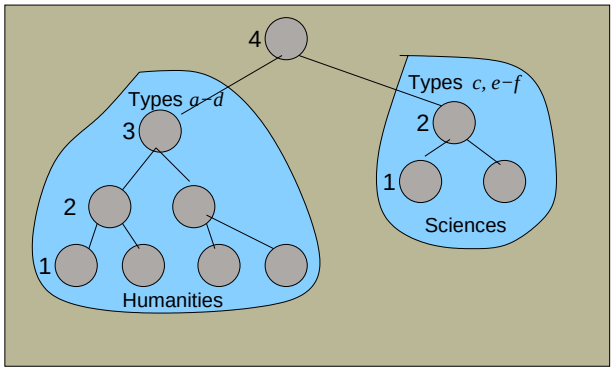


Figure 4.1: The two blobs show Humanities and Sciences division of a library. The nodes in the blobs represent the task handlers, and the number next to the nodes at each level represent its rank in the organization, with nodes at higher level having higher rank. The humanities librarians can process tasks of types *a-d*, while the sciences librarians can process tasks of types *c, e-f*. Tasks of type *c* are common to both humanities and sciences. The root node with rank 4 can handle tasks of all types, i.e., *a-f*, if it satisfies the rank requirement for the tasks.

So, the tasks are classified according to types and rank requirements. Thus, task *T* with type *p* and requiring $Rank^q$ or higher is classified as $T^{p,q}$. The estimated processing time of a task of type *p* is given by twl^p. Thus, $T^{p,q}$ has estimated processing time of twl^p. Suppose that the organization has *n* actors, denoted by

a_i, where $i = \{1, \ldots, n\}$. Each actor a_i has a_i*[rank]* attribute, denoting its rank in the organization. If an actor a_i is handling the task $T^{p,q}$ for the first time, and has the required or higher rank in the organization, i.e., a_i*[rank]* $\geq$ $Rank^q$, then a_i can handle $T^{p,q}$, and the processing time of a_i for that task is the estimated processing time for that task, i.e., $wl_i^p = twl^p$. On the other hand, if a_i has handled tasks of this category before, then his processing time wl_i^p for that task $T^{p,q}$ is calculated as the average value of all successful execution times for these tasks.

Processing times of one task type can differ from another task type. For example, processing a fair use request for mass printing may take significantly more time than a request for one-time printing. Thus,

- Tasks are classified according to types and rank requirements.
- For task $T^{p,q}$, processing time by each actor is calculated.
- The past work history of the actor on $T^{p,q}$ may be used to predict the capability of the actor for task execution. The capability of an actor in executing task $T^{p,q}$ is defined by number of successful past executions s on $T^{p,q}$ divided by total execution time t on $T^{p,q}$ (assuming $t \geq 1$), that is, $\frac{s}{t}$. It may be determined by keeping a working log for each actor which includes the success flag, execution time for each execution instance, and type of tasks executed.
- An actor is chosen whose load is minimum and is qualified for task $T^{p,q}$. The qualifications are judged on the basis of two criteria: capability and rank. An actor a_i is qualified for task $T^{p,q}$ if a_i*[rank]* $\geq$ $Rank^q$, and for the required capability, $capability^q$ defined for that task, capability of that actor (given by $\frac{s}{t}$ above) equals or exceeds it. $capability^q$ can range from 0 to 1 (as the range of $\frac{s}{t}$ is *[0,1]*), with 0 being least important and 1 being most important.
- **My contribution:** The task assignment algorithm calculates the load of task handlers only for the tasks that are being assigned by the algorithm. In our case, it will be used to handle fair use requests, but the task handlers may have multiple responsibilities, such as handling the library paperwork, answering the email queries etc., and the tasks associated with these responsibilities will not be passed to the algorithm because they are not fair use requests. They are handled separately by the task handlers, and they add to the load of the task handlers. But the load associated with them won't be known to the *org_load_balance* algorithm. To reflect their load in the algorithm, I introduce *other workload* to reflect the load associated with such tasks. The task handlers provide *other workload* information to the task

assignment algorithm. Thus, if the task handler a_i has m pending other workloads, they are denoted by $owl_{i,1}, \ldots, owl_{i,m}$. Then, the total load associated with other responsibilities is $\sum_{l=1}^{m} owl_{i,l}$.

The processing time wl_i^p of actor a_i on task $T^{p,q}$ is calculated by the **Workload Prediction Algorithm** as follows.

Workload Prediction Algorithm

1. Calculate total execution time t of a_i for past instances of $T^{p,q}$
2. If $t = 0$, goto step 5.
3. Calculate s, the number of time a_i has executed $T^{p,q}$ successfully, and calculate $\frac{s}{t}$.
4. If $\frac{s}{t} \geq capability^q$, then put $\lambda = 1$ and calculate processing time $wl_i^p = \frac{(\sum_1^s wl_{i,j}^p)}{s}$ where $wl_{i,j}^p$ is the processing time for j-th successful execution of $T^{p,q}$ by a_i. Goto step 6.
5. If $a_i[rank] \geq Rank^q$ then $\lambda = 1$, $wl_i^p = twl^p$. Else $\lambda = 0$, $wl_i^p = 0$.
6. End of the algorithm

In the above algorithm, λ encodes the qualification of a task handler for handling the task $T^{p,q}$. Thus, if the task handler has the required capability and rank, then $\lambda = 1$, else $\lambda = 0$. When $\lambda = 0$, the actor doesn't qualify for handling the task and won't be considered for task assignment in the *Actor Assignment* algorithm below.

In step 2, if $t = 0$, then it is the first time the task handler is processing $T^{p,q}$, and the algorithm jumps to step 5, where if the task handler has the required rank or higher, his processing time wl_i^p for that task is calculated as the estimated processing time for that task, twl^p. The capability of task handler is undefined due to the absence of work history for that task category and so, is not taken into account the first time. On the other hand, if $t \neq 0$, the task handler has processed that task before. Then, the algorithm calculates his capability for that task in step 3, and if it exceeds the required capability for the task, sets λ to 1 and calculates the processing time as the average value of all successful execution times for that task. Also, since the task handler processed this category of task before, he has the required rank (assuming that the ranks of task handlers don't change with time), and so, the algorithm skips step 5.

The above workload prediction algorithm is used to predict the processing time wl_i^p of the actor a_i for the task $T^{p,q}$ in the *Actor Assignment* algorithm. Let *m* be the number of tasks that are pending with actor a_i, and let $pwl_{i,j}$ represent the processing time for *j*-th task. Let *r* be the number of other responsibilities that are pending with actor a_i, and let $owl_{i,j}$ represent the processing time for *j*-th *other workload* of a_i. Then, the load of a_i for these tasks is $\sum_{j=1}^{r} owl_{i,j} + \sum_{j=1}^{m} pwl_{i,j}$. Then, the following algorithm finds the actor to whom the task $T^{p,q}$ should be assigned.

Actor Assignment Algorithm

1. $i = 1$
2. Repeat steps 3 through 6 until $i = n$
3. Calculate wl_i^p and λ through *Workload Prediction* algorithm
4. Calculate $TotalLoad = \sum_{j=1}^{r} owl_{i,j} + \sum_{j=1}^{m} pwl_{i,j}$
5. $wl_i = \lambda(wl_i^p + TotalLoad)$
6. $i = i + 1$
7. Select a_i where $wl_i = \min wl_i, i = 1, \ldots n, wl_i > 0$
8. End of the algorithm

The above algorithm calculates the load $wl_1, \ldots, wl_n$ of actors $a_1, \ldots, a_n$ respectively given *n* actors. In step 3, it predicts the processing time of actor a_i, wl_i^p for task $T^{p,q}$ that it wants to assign. In step 4, it calculates the pending load, and in step 5, adds it to the predicted processing time from step 3. If the actor qualified for the task handling (in step 3), $\lambda = 1$, and so, $wl_i > 0$, else $wl_i = 0$. In step 7, the algorithm picks the actor that has the minimum load of all actors who have non-zero positive load (i.e. $\lambda = 1$).

Step 4 of the above algorithm is different from the corresponding *Actor Assignment* component in *org_load_balance* algorithm in that it adds $owl_{i,j}$ in the calculation of *TotalLoad*.

The algorithm takes into account an actor's capability derived from his past success, and organizational rank, and the load associated with other responsibilities. If more than one actors have identical minimum load, then

the tie might be broken between them based on some criteria, such as the lapsed time when the last task was assigned.

4.4 Application to Fair Use Request Processing

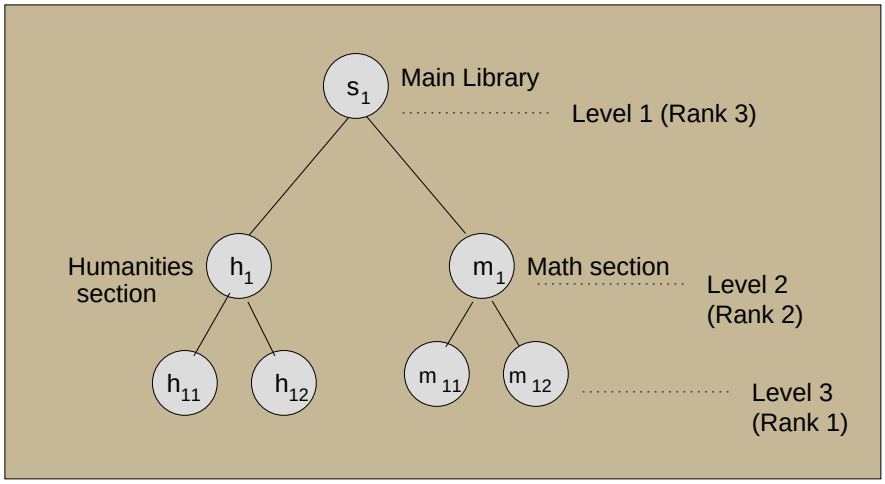


Figure 4.2: A multi-library university library structure. The nodes represent the librarians. Librarian s_1 can process both humanities and math related requests, and is ranked highest in the structure. The nodes in Humanities and Math sections can process only humanities related and math related requests respectively.

I illustrate the application of above algorithm in the context of a fair use request processing scenario involving a multi-library library organization at a university. Each node represents a librarian. The requests are categorized based on the type of requests and the discipline. Let `type` denote the type of requests and `type=<purpose_of_use,end_users,discipline,user>`, where `purpose_of_use` and `end_user` are determined from the selection the users make in the FURI in the previous section. So, a request r_1 with `<Redistribution-educational,closed-group,endusers,humanities,faculty>` attributes is of different type than a request r_2 with `<Redistribution-educational,closed-group,endusers,math,student>` attributes. Also, the organization structure of the fair use processing service is shown in Figure 4.2. Level 1 and Level 2 nodes have the requisite rank, skills and capabilities to process the requests from both faculty (which may be considered more important than non-faculty requests) and non-faculty in their respective disciplines (level 1 can process both disciplines) , while level 3 nodes can process the requests from non-faculty only. Thus, r_1 will be processed by either h_1 or s_1 depending on their load. So, if say, s_1 has high priority responsibilities associated with library administration,

the load of s_1 may show up as higher than load of h_1 even though s_1 has no pending fair use requests. On the other hand, if s_1 has relatively easy workday, then his load will be lower than h_1, and r_1 may be assigned to him. Similarly, r_2 will be processed by either s_1, m_1, m_{11} or m_{12} depending on who has the minimum load.

As the above example demonstrates, the algorithm takes into account the organization structure associated with task handling, and uses multi-goal load-balancing based on rank and capability to handle the requests. Assigning priority weight to the tasks associated with other work responsibilities allows the algorithm to take into account the multiple roles of actors in the organization.

4.5 Simulation

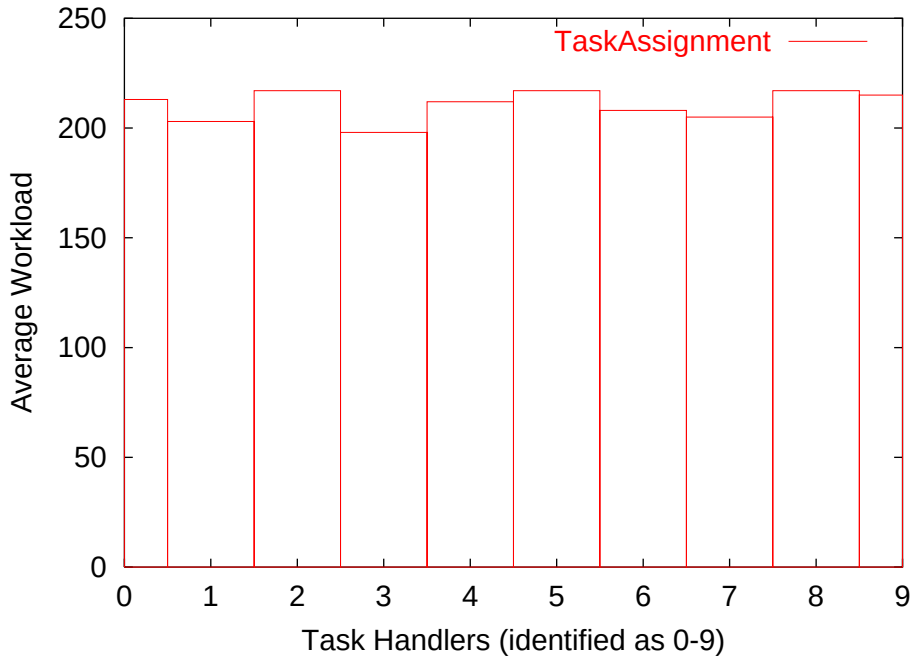


Figure 4.3: The simulation output of the load-balancing algorithm for ten task handlers is shown here. It shows the load of ten task handlers averaged over 100 days. The algorithm distributes the load uniformly among all the task handlers, as seen here.

The algorithm partitions the task handlers in several groups based on the types of requests they can handle, other workload, capability and rank. A good algorithm will distribute the load evenly among the task handlers in each group, as they all have similar capabilities (and assuming they have equal amount of *other workload*). To determine the performance of algorithm, I used ten task handlers with 0 units of "other workload" to give

all task handlers equal weight in task handling, and the "capability" requirement of all tasks was set as 0 units, to allow all task handlers to be considered in the assignment of all tasks. I derive some assumptions about the request arrival rate and processing time. Poisson request arrival rates have been found to hold for distributed systems and real world processes [39]. Under the assumption that fair use requests are random real world processes, I use Poisson distribution to model their arrival rate. Erickson [9] have observed that the processing times for majority of the fair use requests are clustered together, leading to my assumption of a Gaussian bell-shaped distribution for the task processing times. The tasks were simulated using assumed Poisson arrival rate of 140 and Gaussian processing time distribution (with each different processing time corresponding to a task type) of 30 with standard deviation of 5. The simulation result over 100 cycles is shown in Figure 4.3. As seen in that Figure, the load is distributed uniformly among all actors.

While the simulation results show good load distribution among the task handlers under uniform capabilities, we need to validate the results in actual settings where the capabilities and skill levels of task handlers are different, diverse and may fluctuate with time. The task arrival rates too can fluctuate with time. For example, larger number of fair use requests may be made during an academic term deadline than during the beginning of the academic term. The impact of other responsibilities and priorities of task handlers on the distribution and processing of requests too needs to be assessed. The unavailability of a working fair use request service implementation and the lack of data associated with such service precludes such validation through simulation.

4.6 Fair Use Request Process

In this section, I use the FURI and load-balancing components to design the process flow for fair use request process for requests to the TPRS. The process flow is shown in Figure 4.4, and is explained below:

1. The user receives a license through the conventional request process. The license also contains the tag specification for the FURI.

2. Later, the user attempts to make a use that is not allowed by the license. The DRM controller detects it, and asks the user that it wants to send a rights request to the TPRS under the fair use contention. If the user decides to make rights request to the TPRS, the DRM controller generates the FURI from the

license using the FURI generator.

3. The user encodes his request for the desired rights using the FURI, and the request is sent to the TPRS.

4. The server at the TPRS receives the request, and invokes the load-balancing module to pass the request to a request handler. In this example, the server is located at the license generator for illustrative purpose.

5. The request handler processes the request, and invokes the license generator to generate appropriate license if needed. The new license would possibly be an expansion of the original license to include those rights as requested by the user in step 3 , and granted by the request handler.

6. The result is sent back to the user.

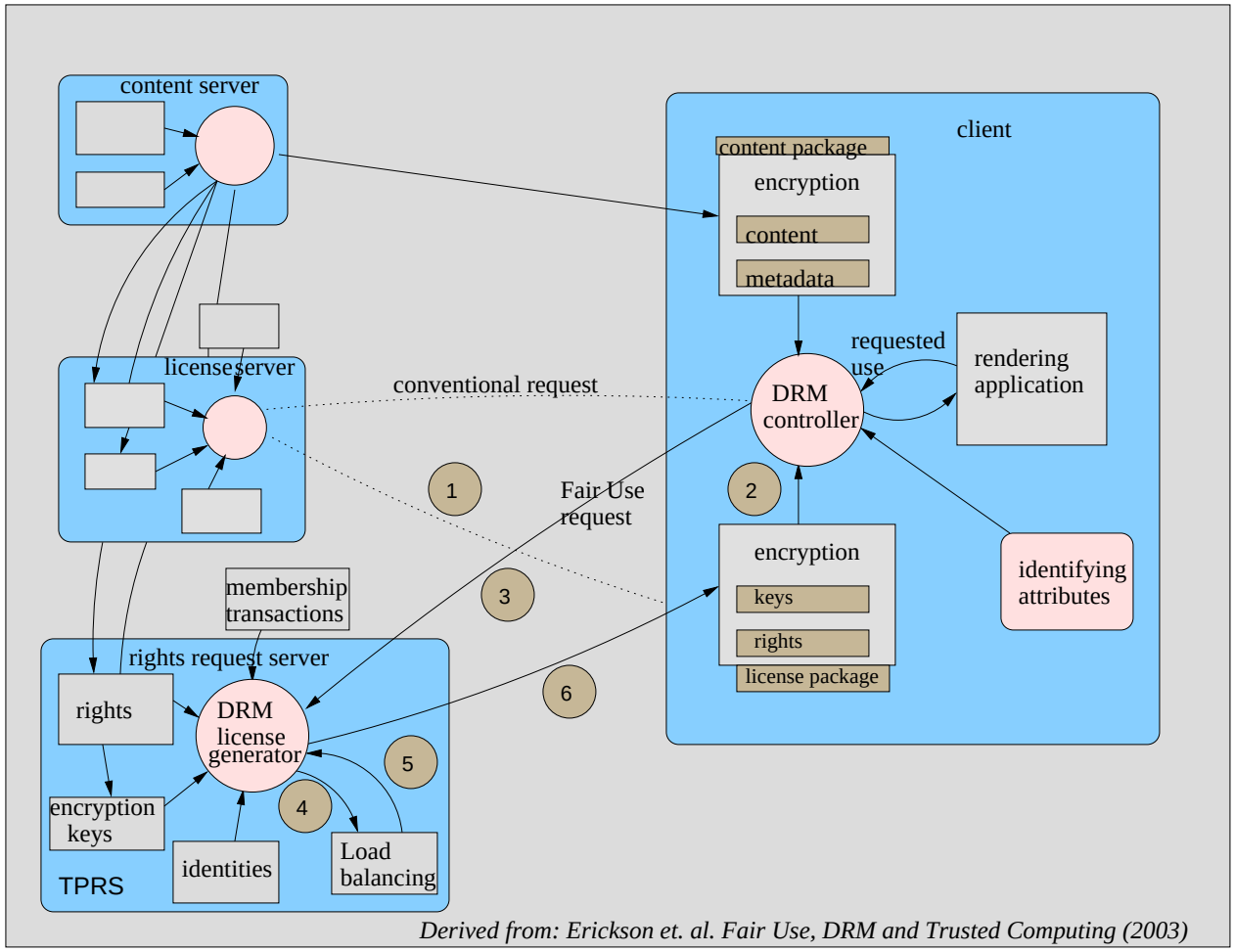


Figure 4.4: An illustration of process flow for fair use requests using TPRS

4.7 Conclusion

The *Task Assignment* algorithm takes into account the organization structure of TPRS, and allows for the task assignment on the basis of the nature of fair use requests, and the abilities of the request handlers who can process them. The capabilities of request handlers are determined from their success rate in handling such requests before, and their rank in the organization. Further, the request handlers may have other responsibilities, such as handling paperwork. The tasks associated with these responsibilities are usually assigned outside the task assignment algorithm. To handle such cases, the concept of *other workload* was introduced to calculate the load associated with these responsibilities. This load is considered by the algorithm when making the task assignment.

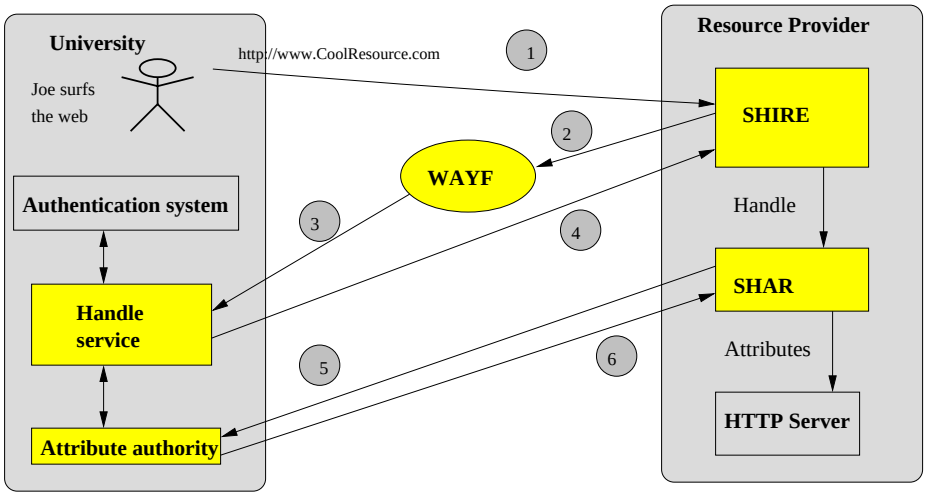
The simulation shows that the algorithm distributes the tasks evenly in case of the handlers with same capabilities, and same load associated with other responsibilities. The performance of the algorithm needs to be validated in the real-world settings where the capabilities, task processing times, and task arrival rates may show wide fluctuations.

Chapter 5

Modifications in Shibboleth Architecture

5.1 Introduction

I discussed the two major components of the Third Party Rights Service, namely, the Fair Use Request Interface, and the load-balancing component for the fair use request processing, in the previous two chapters. I also discussed the fair use issues in Chapter 2, and illustrated the need for a flexible licensing infrastructure in academic settings in that chapter. Shibboleth is an architecture for accessing resources in the academic settings, and in this chapter, I will illustrate the shortcomings of the Shibboleth architecture in the context of the fair use issues, and the Third Party Rights Service. After discussing the shortcomings in Section 5.2, I will propose changes in Shibboleth architecture in Section 5.3, and analyze the resulting architecture. Section 5.4 summarizes the work.



Taken from: Shibboleth−Architecture Draft v05 (2002)

Figure 5.1: An illustration of components and message flows in Shibboleth architecture

5.2 Shibboleth Architecture

5.2.1 Original Architecture

Shibboleth [8] is an academic project to develop a DRM solution to support research and education in academic community. Shibboleth aims to provide electronic resource sharing across the organizations by "federating" the administration. In federated administration, the resource provider leaves the administration of user identities and attributes to the user's origin site, and it relies on the origin site to provide attributes of the users that it can then decide in making an access control decision when the user attempts to use a resource.

Figure 5.1 shows the Shibboleth architecture components and the message flows between them that take place when a user attempts to access a shibbolized resource:

1. The user enters a URL in the web browser to access a shibbolized resource from a HTTP web server.
2. The Shibboleth Handle Indexical Reference Estabilisher (SHIRE) receives the user's request and sends the user's desired target URL and SHIRE's URL to the "Where Are you from" (WAYF) server.
3. The WAYF server redirects the user to the Handle Service (HS). The user authenticates with the authentication system at the origin site, and after the HS receives an affirmation from the authentication

system, it interacts with the Attribute Authority (AA) to create an opaque handle for the user. This handle will later be used by the AA to provide the handle attribute requests for that user.

4. The HS returns the handle package to the SHIRE. The package includes the opaque handle and the address of the user's local AA server (UAA). The SHIRE passes this handle package to the Shibboleth Attribute Requester (SHAR).

5. The SHAR constructs an Attribute Query Message (AQM) and submits it to the UAA defined in the handle package. The AQM includes the opaque handle, the target URL and the SHAR name.

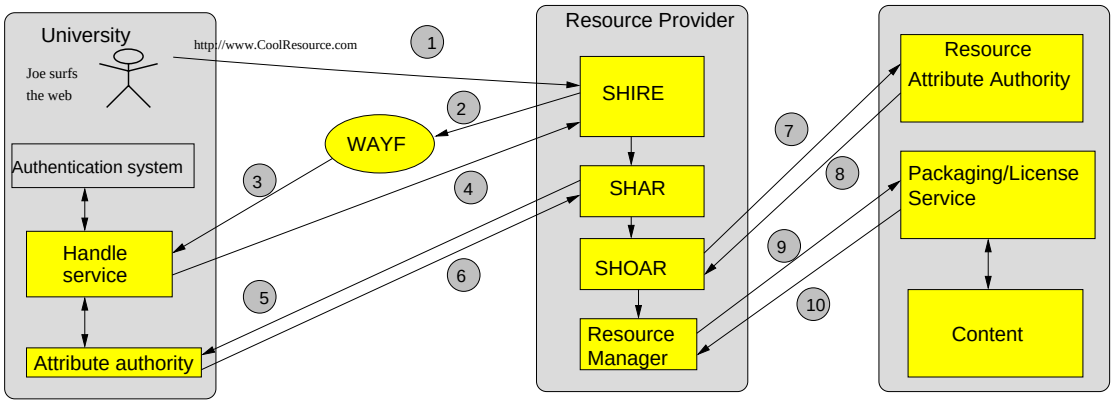
6. The UAA processes the AQM and responds with an Attribute Request Message (ARM) which includes the SHAR name, the target URL and the user attributes as allowed by the user's Attribute Release Policy (ARP). The SHAR passes the attributes to the Resource Manager (RM) of the resource that the user originally requested. The resource manager then processes the attributes to decide whether to allow the user access to the requested resource.

Thus, in Shibboleth, once the user has access to the resources, he/she can make unrestricted use of them because of absence of a DRM client to control usage of resources, and unsecured nature of downloaded resources. Shibboleth belongs to the EN category of the DRM taxonomy discussed in Chapter 2 because of the absence of the DRM client and the repository nature of the resource access.

The lack of the DRM client in Shibboleth gives the academic users control over the use of resources, and while it certainly allows for the fair use of resources, it does nothing to stop other uses that might be serious infringement of the copyright law. It restricts the shibbolized resources primarily to those resources that are open-source in spirit, with little restrictions on their use. The exclusion of the DRM client from Shibboleth is partly due to the recognition that the current DRM solutions with the DRM client tip the balance between the rights owner and the user too much in favor of the rights owner, which undermines the fair use and the first sale doctrines, two principles of critical importance to research and education communities [5, 17, 18, 37].

5.2.2 FDRM Proposal

Martin et. al. [14] have recognized the importance of including the DRM client in Shibboleth to increase the range of shibbolized resources, and have proposed a *Federated Digital Rights Management* (FDRM)



Taken from: Martin et. al. Federated Digital Rights Management (2002)

Figure 5.2: The process flow in the FDRM architecture

architecture based on the Shibboleth architecture. I discuss the FDRM architecture to illustrate how the DRM client functionality can be added to Shibboleth. Figure 5.2 shows the FDRM architecture.

The FDRM proposal extends the Shibboleth architecture to include four components, as discussed below.

Resource Attribute Authority (RAA): This is a database of metadata containing the rights records of rights, permissions and constraints associated with the resources. The records are written in an XML-based REL.

Shibboleth Object Attribute Resolver (SHOAR): This component interacts with the RAA to obtain the rights metadata associated with the requested resource.

Packaging/License Services (PLS): It is the component that dynamically packages content for delivery. It may create licenses as part of the packaging, specifying the rights the user is allowed to exercise on the resource.

Resource Manager (RM): This resolves the user's attributes and resource attributes such as rights, permissions, and constraints, and forwards the details of the package request to the PLS. The RM is equivalent of the DRM controller in the DRM reference architecture.

In the FDRM, the first six steps are same as discussed in Section 5.2.1. For the next four steps, the following take place:

7. The SHAR passes the result of the ARM to the SHOAR in step 6. The SHOAR constructs a Resource

Attribute Query (RAQ) and submits it to the Resource Attribute Authority (RAA) associated with the requested resource. The location of the RAA and the requested resource is contained in the URL accessed by the user.

8. The RAA returns a Resource Attribute Response (RAR) detailing the supporting services and the access rights associated with the requested resource to the SHOAR. After processing the assertions from the UAA and the RAA, the SHOAR sends a package request to the RM.
9. The RM forwards this request to the PLS.
10. The PLS creates the requested package and sends it back to the RM. The RM passes it to the user.

The FDRM proposal proposes that the RM be implemented on the server side using the CGI prototyping or Apache module. The RM interacts with the Shibboleth components through the SHOAR as seen in Figure 5.2, and is not directly involved in the Shibboleth protocol. I note that while the RM is equivalent of the DRM client in the FDRM proposal, the DRM client could instead be implemented at the user's side in user's domain, and the RM may act as a package dispatcher instead, sending the package to the user through his web browser. The user may then access the package through the DRM client.

The FDRM architecture while providing for the DRM client and the Packaging/Licensing services, lacks a good mechanism for expanding the licenses to include the fair uses. A user may make requests for expanded licenses to the resource provider, but it is up to the resource provider to decide whether to grant such requests or not (besides exposing the fair use usage patterns to the resource provider). The resource provider has the power to decide on the requests, and can decline the requests (such as criticism or parody use) that may be detrimental to his interests, but would qualify as the fair uses. Thus, while the FDRM proposal may meet the goal of protecting intellectual property, it lacks a mechanism for flexible access to resources to meet the fair use needs of the community, and realizing the rich potential of networked collaboration and flexible resource access that could be possible through the federated architecture of Shibboleth.

5.3 Third Party Rights Service in Shibboleth

5.3.1 Issues To Consider in Design

In Shibboleth, the users are grouped in the same domain based on their affiliation with same academic organization (such as Dartmouth College) and so, introducing the TPRS seems a good choice . Thus, in Shibboleth, a TPRS can be introduced to deal with the user requests as opposed to the non-Shibboleth cases where the users may have affiliation to different organizations, preventing a meeting of interests between the TPRS and the users.

The issues involved in the introduction of the TPRS in Shibboleth are:

Procurement of default licenses by the users: As discussed in Chapter 3, the users need default licenses to access the FURI which is generated from the interface specification coded in these licenses. The first step in making requests to the TPRS is to have access to the fair use interface for composing requests. So, the mechanism of procuring default licenses should be established as part of the design process.

Location of the TPRS: I refer to the domain of academic institution the users belong to as academic domain. The TPRS should be located in the academic domain in Shibboleth architecture, because it belongs to the user's academic domain by the nature of the service which offers an authorization authority to the users while trying to strike a balance between the interests of the rights-holders and the users. The users may not trust a TPRS that belongs to the resource provider's domain. There are three components in that domain: Authentication system, Handle service and Attribute authority. The TPRS may interact with these components to aid in the decision making process, and this interaction should be figured out.

Mechanism of contacting the TPRS: When the users contact the TPRS, they may submit their requests directly to the TPRS through the FURI or they may first submit to authentication by the Authentication system in their domain before submitting the requests.

Mechanism of communication between the TPRS and users: After the users submit their requests to the TPRS, the TPRS may need to contact them during the request processing if more information is needed. Establishing such a mechanism of contact is complex because of complicated legal, political and social issues

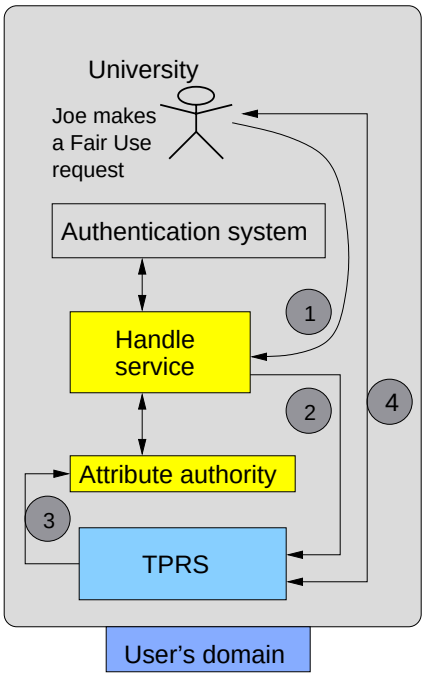


Figure 5.3: User's domain in Shibboleth with the addition of TPRS component

involved [3, 2]. Anonymity is one such issue that was discussed in Chapter 2.

Location of Packaging/Licensing services: Location of Packaging/Licensing services (referred also as P/LS) is important in the TPRS. Whether they should duplicate the rights-holder's P/LS at their site, or whether they should send the license requests to the P/LS of the rights-holder is an issue that should be resolved during the TPRS design in Shibboleth. I note that the packaging and licensing services are usually distinct components in the DRM architectures. In the P/LS module (based on the PLS module in the FDRM proposal) considered here, these components can be different entities, and they interact with each other to provide the packaging and licensing services as represented by the module.

5.3.2 Modification of Shibboleth Architecture

Before modifying the Shibboleth architecture to introduce the TPRS, I assume the presence of a DRM client-based licensing in Shibboleth such as the FDRM. Though Shibboleth lacks a working implementation of such a licensing service, I believe that presence of a TPRS that serves to balance the interests of users and copyright holders could trigger the momentum for such an implementation. By designing the TPRS such that

it is independent of the DRM implementations as long as the licenses generated by such implementations are FURI-compatible through inclusion of specifications as discussed in Chapter 3, and use of the ODRL as the REL to express the licenses, I leave the DRM designers flexibility in designing and implementing the DRM services to take advantage of the TPRS component in the modified architecture.

While the Shibboleth protocol mentioned in section 5.2.1 is meant for access to web resources by the users, I note that these web resources might include the secured contents and the licenses which the users may download (assuming availability of the suitable DRM client on user's side). An enhanced P/LS based architecture such as FDRM can also be used to access the contents and the licenses. The following steps take place for the fair use requests by the users:

1. After the user has the license, he/she can use the Fair Use interface (from the interface specifications in the license) to compose the requests for the fair use exceptions. These requests are submitted to the HS which accepts the requests after authenticating the users (e.g. via Kerberos at Dartmouth College).
2. The HS generates the handle for the user, packages it with the request and sends it to the TPRS.
3. The TPRS receives the user's request, and uses the handle to look up user's attributes that it might need in the decision making process. It also looks up the user's contact address with the AA.
4. The TPRS contacts the user if it needs more information. After processing the request, it communicates the decision to the user.

If the TPRS grants the license to the user, the user can then use the license with his DRM client to make use of the resources as granted by the license.

Let us investigate this design here.

The TPRS protocol doesn't interfere with the Shibboleth protocol. Given the license containing the tag specifications for the FURI, and the support at the AA to respond to the TPRS requests, users can make requests to the TPRS after they have received the license through the Shibboleth protocol. The addition of the TPRS can be done without changing the existing Shibboleth protocol, as will be discussed in Section 5.3.3, and can be utilized by the users if the support for the FURI is provided in the licenses.

I assume that the TPRS is honest (which is a good assumption since it is usually controlled by the authorities such as librarians in user's domain), and will abide by the policies as set by user's AA. Having the user authenticate to the local authentication authority, and then sending the request to the appropriate TPRS by the HS along with user's handle is good in the situations where:

- It is undesirable to carry the changes in the P/LS services (especially, when a number of resource providers are involved, each with its own P/LS service) to include the authentication handle in the licenses so that users can contact the TPRS directly instead of going through the HS. Also, such an authentication handle may get stale should the authentication requirements change later. By having the user authenticate to the HS, the HS can ensure that the authentication requirements are met before the request is submitted to the TPRS.
- There are multiple Third Party Right services. Then, it might be hard for the user to resolve which service to contact. Through the use of the HS in authentication step, this decision can be shifted to the HS which can decide about the service to contact, perhaps based on the user's attributes at the AA. For example, Arts and Science faculty requests may be handled by a separate, dedicated service.
- The TPRS location changes with time. The HS would be aware of location of the TPRS and could direct the requests to them. This is better than the scenario where a user attempts to contact a outdated TPRS, possibly when he/she has a license that has old URL.

There is also the question of whether the TPRS should be allowed to know the user's identity for the purpose of contact. It is a critical question because the TPRS will be dealing with the usage requests from the users, and if it also knows the user's identities, then it will probably know who made which request for what and the purpose behind the use. Consider for example, Bob's request to use extract excerpts from a professor's work for criticism in an anonymous whistle-blower article on tenure-process that Bob is writing. If Bob knows that the TPRS will potentially know his identity during the request process, then he may hesitate to make the request, in fact may not make the request unless he has a guarantee that the TPRS won't reveal his identity even to the college authorities who run the TPRS, and thus, compromise his anonymity.

If instead, such work were available say in print format in one of the college libraries, then such problem won't arise for Bob because no one will know that he browsed and used the extract from the book in his article. If he checked out the book, then the record linking the book to him may be deleted after he returns it without

any fine. For example, at Dartmouth College, under “Patron Record Privacy”, a library registration record is created from each patron, and it contains the borrower’s name, address and other identifying information that is used for circulation purposes. The library’s circulation function temporarily links a patron with the library material he/she has checked out and if the item is returned in time without any fine or exceptions, then the link between the patron and the library material is severed. The library staff follows a policy of non-disclosure, under which it won’t reveal the link between the checked out material and the patrons, personal information of any patron (such as address, phone number, email), or identify a patron who has checked out a particular item or describe them in any way.

As I explained before, I assume that the TPRS is honest, and that it follows the requirements (such as anonymity) as specified by the user’s AA. Having the TPRS know user’s identities provides a simple mechanism of communication between the TPRS and the users in cases where the TPRS needs more information from the users, and under the assumption that all components in user’s domain are equally trusted, doesn’t impact the anonymity of the user if an intermediary such as the HS is used for communication between the TPRS and the user, as the intermediary is equally likely to divulge the information as the TPRS. The privacy work on secure coprocessors [22, 21] is of interest here because of the capability of secure coprocessors to enforce policies as set by the TPRS.

However honest and scrupulous TPRS may be in maintaining the anonymity of users, there is a certain development that should be kept in mind. Under the “USA Patriot Act” [1] passed by Congress, the state library confidentiality records protecting the library records are overridden, and the libraries are required to comply with the act. While a library’s policy relating to privacy and confidentiality of information (such as, severing the links between the patrons and records) may not change, they may be served with a search warrant under the act to reveal the information on patrons, and further under the “gag” provision , they may not disclose, under penalty of law, the existence of warrant or the fact that records were produced as a result of the warrant. The staff can not reveal to a patron that he/she is the subject of the FBI investigation, that his/her records were given to the FBI, and can not reveal the inquiry to co-workers, media or other government officials, other than reporting it to a higher authority within the library.

The issue of location of the P/LS is also important. Ideally, the TPRS should duplicate the P/LS of the resource providers. If they are to use the P/LS provided by the resource providers, then they invariably may give away the information about the pattern of usage and requests to the resource providers. This may be

undesirable, and may subject them to disputes and infringement claims. On the other hand, by having its own P/LS, the licenses can be generated within the user's domain, and the information about request and usage patterns can be limited to their domain if allowed by the agreement between the domain and resource providers. The nature of licenses also dictates that the DRM architectures deployed by the resource providers (and the users) should belong to either the MX or EX categories of the DRM taxonomy. The reason is that licenses (and the specifications within it) needed for generation of the FURI should be separate from the contents. The separation between the licenses and the contents allow for separation of policies and contents. After the users receive new licenses from the TPRS, they should be able to use it to access the contents because of the separation between the contents and the policies on content use.

5.3.3 Accommodating Design Changes and Effect of the TPRS Addition on Shibboleth Protocol

Let us call the modified Shibboleth architecture based on the TPRS *Shibbot* (**Shibbo**leth with **T**PRS). Further design changes may be made in Shibbot to accommodate the EX architecture (to take advantage of the TPRS). This would enable the addition of new resources to take advantage of the TPRS functionality in Shibbot.

The Shibboleth protocol discussed in section 5.2 is based on the exchange of attributes between the user and resource provider domain. It assumes presence of a web browser at user's side, and a web server on the resource provider's side to establish resource related communication between the users and the resource providers. The five components of Shibboleth, namely, HS, AA, WAYF, SHIRE and SHAR are involved in the communication between the user and the resource providers, and the exchange of attribute information. The FDRM proposal builds on this communication capability by adding other components in resource provider's domain which communicate with user domain through the SHAR, as seen in Figure 5.2 . In Shibbot, the TPRS resides in user's domain, and interacts only with the HS and the AA in that domain. As we saw in section 5.3.2, the interaction of the TPRS with the HS and the AA is separate from the interaction between the HS, the AA and the resource providers. Thus, even if Shibboleth architecture was modified to Shibbot, the existing shibbolized resources would be available to users, with the TPRS playing no role in the communication process between the user and resource provider for those web resources. Figure 5.4 shows the interaction of the TPRS, shibbotized (explained below) and shibbolized resource servers with the HS and the AA. As seen in that figure, interactions of the TPRS and the shibbolized resources are independent. The

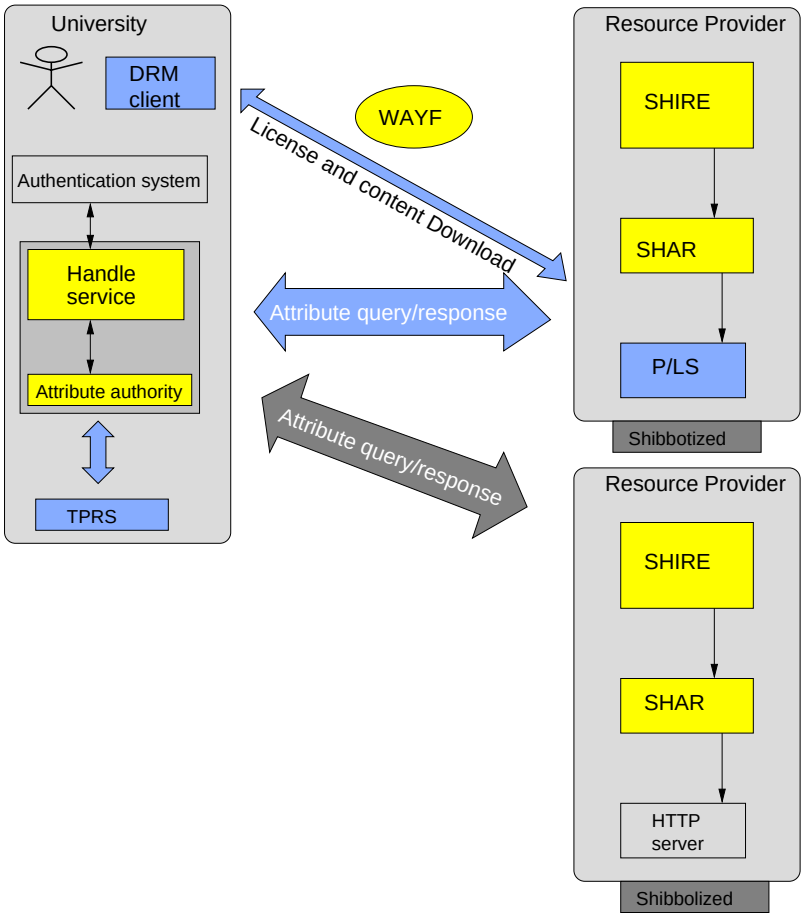


Figure 5.4: Interaction between the TPRS enabled user domain, shibbotized and shibbolized resources: "TPRS", "DRM client", and "P/LS" are the new components. The TPRS component is added to user's domain, and doesn't affect the interaction of existing shibbolized resources (shown by the slanted *Attribute query/response* arrow) with the user's domain. To take advantage of the TPRS service, DRM client and P/LS can be added in user's and resource provider's domain respectively, and the resulting shibbotized resources can be introduced in parallel with the existing shibbolized resources. The *License and content Download* arrow indicates the interaction between the DRM client and the P/LS to transfer the license and content packages from the P/LS to the DRM client.

users don't need the TPRS functionality for shibbolized web servers, and further, all the TPRS interaction with the HS and the AA are limited to within user's domain, hiding it from the interactions between the user and resource provider's domain.

Providing the shibbotized resources in Shibbot will primarily involve two additions. First, a DRM client has to be provided to the user for controlling the use of resources as specified by the license. Second, a P/LS has to be added in the resource provider's domain to provide packaging of license and content in appropriate format for the DRM client. The only dependency on the TPRS for these services is that in order for the users to make

requests to the TPRS, the licenses should contain tag specifications for the FURI, and the DRM client should have the FURI generator that users can use to generate the FURI and compose the requests to the TPRS. The TPRS will also need to duplicate the P/LS in the user's domain to create the licenses for the user requests. The DRM client and P/LS components should also use existing Shibboleth communication protocol between the user and resource provider domains to communicate with each other. Thus, the DRM client may have a web browser plug-in through which user may access the P/LS with HTTP front-end. Alternatively, the user may download the content and license package for the DRM client separately by accessing the P/LS resources through a web browser. The P/LS would have a HTTP front-end so that users can access the packaging and licensing resources through web interface. The P/LS may communicate with user's domain to find user attributes, for packaging and licensing decisions. It may communicate with user domain through the SHAR or may have the functionality to communicate directly with the UAA in user's domain through the handle provided by the SHAR.

For example, Figure 5.4 shows a shibbotized resource provider with P/LS (with HTTP-based web interface) added to it. As shown in Figure 5.4, DRM client is added in user's domain. The packaging and licensing service, i.e., P/LS is added in the resource provider's domain and it has a HTTP front-end which the user may access after authentication and exchange of attributes through Shibboleth protocol as discussed in Section 5.2. After the user makes the request for content and license through the HTTP front-end, the P/LS may obtain attributes from user's domain, either through SHAR or directly, using the handle. These attribute exchanges by PL/S are shown by the **Attribute query/response** arrow in the figure. After the required attributes have been obtained, PL/S would package the content and license in appropriate format, and make it available for the user to download, illustrated by the **License and content download** arrow in the figure. Once the content and license are downloaded, they may be unlocked by the user's DRM client, and rendered. Since the license contains the FURI specification, if the user later wants to make the fair use request to the TPRS, the FURI can be generated by the DRM client to compose and send request to the TPRS.

Thus, the addition of the TPRS functionality to Shibboleth can be done while keeping the existing shibbolized web resources. The shibbotized web resources can be added in phases through the addition of the appropriate DRM client and P/LS in the user and resource provider domains respectively. This is an advantage in the academic library settings where changes may be carried out in phases and incrementally.

5.4 Conclusion

Addition of the TPRS to Shibboleth is a desirable functionality that can be done in the user's domain, without affecting the shibbolized resources. The TPRS design proposed in this chapter provides high flexibility to the designers for addition of packaging and licensing functionality to make use of the TPRS facility. It allows for an incremental approach to Shibboleth modification where the TPRS, P/LS and DRM client components can be added in different phases. This is very useful in academic environment where the libraries tend to be very conservative in adding new functionalities, and may want to validate the TPRS design before deciding on the addition of the P/LS and DRM client (on the user's side).

While Shibboleth is a design for providing access to the resources with strong bias in the favor of the interests of the academic users, adding the TPRS functionality to Shibboleth can possibly provide a mechanism for managing rights in accordance with the copyright law and its exceptions, especially fair use, and bring in more resources in Shibboleth which would not otherwise be available. The development of a successful DRM model will require active involvement of the academic community and resource providers - content creators, publishers and distributors, and Shibboleth provides an attractive opportunity to try out the TPRS model because of such wide involvement.

Chapter 6

Other Issues and Future Work

6.1 Other Issues and Relevant Directions for Future Work

I conclude the thesis with the discussion of some important issues and the directions for the future work in this chapter.

6.2 Fair Use Request Interface

The FURI while being simple, involves the issue of spontaneity in expressing the fair use requests. The users may find it cumbersome to provide the factual information for their fair use requests in the present FURI design. The underlying fair use information model on which the FURI is based needs more fine-tuning. Such fine-tuning may require deeper knowledge of the range of possible fair use requests, and their classification based on the factual information that may be needed for deciding on these requests. The resulting information model may be used to generate the interfaces that ask for the required factual information more precisely than is currently possible.

The issues of unanticipated requests, and the gap between the drm use functions and the use needs of the users were discussed in Sections 3.6.3 and 3.6.4 respectively. In the first case, a mechanism is needed to map

the purpose and intentions of the users (in making the rights requests) to future, undetermined uses. In the second case, a mechanism is needed to bridge the gap between the user's desired uses, and the use functions provided by the DRM client. Both mechanisms need to be devised, and are part of the future work.

6.3 Load-Balancing Component

The load-balancing algorithm discussed in the load-balancing component should be tested and validated in the actual settings of a working fair use request service. While the simulation shows good load-balancing performance, these results should be validated in actual settings where the capabilities, other responsibilities and the skills of the task handlers may be diverse and fluctuate with time. Further, the task arrival rates can fluctuate with time. For example, at Dartmouth college, a large number of fair use requests may be made during the academic term deadlines than in between the academic terms. The impact of such fluctuations on the performance of the algorithm should be assessed.

6.4 Shibboleth Design Changes

I make the following recommendations for the TPRS addition in Shibboleth:

Change of Shibboleth architecture to EN-EX combination

EN and EX refer to the categories of the DRM taxonomy discussed in Chapter 2. The Shibboleth architecture should be modified to add the EX functionality besides the EN functionality that is already present. Adding the EX functionality will allow inclusion of resources whose usage is desired to be controlled by the DRM client. With the inclusion of the FURI tag specification in the licenses for these resources, the users can generate the FURI to compose rights requests to the TPRS under the fair use exception, and request additional rights. Since the EX architecture allows for the separation of content and policies, the new licenses can be used by the users to access the contents as allowed by the policies specified in the licenses.

Changes in Shibboleth Components

The HS and AA in user's domain should be designed to handle the TPRS. Functionality should be added to the HS to authenticate users making fair use requests, to package the request with a suitable handle, and send it to the TPRS. The AA attribute specification should be developed for specifying anonymity and privacy policies, and for communicating these policies with the TPRS.

Single licensing structure and REL

Having a single licensing structure and REL for specifying policies in the licenses will simplify the implementation of the P/LS at the TPRS. Use of different licensing structures by the resource providers will require duplication of each of these services separately at the TPRS, which could become infeasible if there are too many variations. It might not be desirable for the TPRS to contact the P/LS of resource providers because of usage information that the resource providers may gain, and also, resistance of academic users to such an arrangement due to privacy and anonymity fears.

Address Privacy and Anonymity Concerns in TPRS

The TPRS can be aware of the users and their resource usage history, which raises significant privacy concerns, as discussed in Chapter 5. The development of privacy and anonymity policies in the TPRS is further complicated by the "USA Patriot Act" that overrides the state library laws, and the library policies on anonymity and privacy. The reconciliation between the requirements of Patriot act, state library laws and library policies should be determined, and specified in the form of the policies. The secure coprocessor based privacy work by Illiev et. al. [21, 22] may be useful in the enforcement of the policies. The TPRS is vulnerable to being used by the administrator to glean some information, either covertly (if the policies are against such use) or overtly. Using a secure coprocessor based policy enforcement may provide users with some assurance of policy-enforcement and ease the fears of leakage of usage information. This could go a long way in preventing the chilling of uses with potential (incriminating) consequences for the users should their usage information be revealed.

Recommendation for Implementation

The TPRS implementation in Shibboleth should be done in phases to simplify the implementation process. The implementation process will work as follows:

- In phase 1, the TPRS should be implemented and integrated with the HS and AA in the user's domain in Shibboleth. The handle and the attribute specifications for the handle and attribute query/response respectively should be specified for the interaction between the TPRS, HS and AA. The existing co-processor based privacy work should be investigated to handle privacy and anonymity issues arising from the interaction between the TPRS and users through the HS and AA. The load-balancing technique discussed before should be implemented for task handling in the TPRS, and its effectiveness in load-balancing should be evaluated during the running of the TPRS service.

- In phase 2, a test resource provider should be identified, and its resources should be shibbotized through the addition of a DRM client and P/LS in the user and resource domains respectively. The licenses generated by the P/LS services should contain the FURI specifications, and the DRM client should have a FURI generator to generate the FURI for the users. The P/LS should be duplicated at the TPRS for the license generation. The test runs by the users may be used to refine the FURI specifications further, and to improve the FURI for more effective communication of the factual information needed by the TPRS. During this phase, the issues of authentication by the users making the fair use requests may be investigated, and the mechanisms devised to identify the users correctly, and to link them with the requests.

- In phase 3, the results from the trial runs of phase 2 should be used to expand the shibbotized resources to include the resources provided by the other resource providers.

  Introducing the TPRS functionality in Shibboleth involves several unknown factors, such as form and purpose of the typical user requests, the ideal time frame for processing the requests, the anonymity issues, the organizational set-up of request handling service etc. By implementing the functionality in phases, the knowledge gained in earlier phases can be used for the design changes and implementation in the later phases. The incremental introduction of the TPRS, and the shibbotized resources can be done while keeping the existing shibbolized resources.

6.5 Incorporating Safe Harbor Rights

Not all fair use requests may need to be processed by the humans in TPRS. For example, some fair use requests may be very similar or even identical, and so, using human intervention to process them may be overkill, especially when the results are almost certain. Having a system in place that could screen the fair use requests before passing them to task handlers, screening out the requests where automatic, algorithm-based decisions could be rendered, and process them according to a deterministic algorithm, could improve the handling of the fair use requests. Such system could also build on the experience of the fair use task handlers by constructing a knowledge database encoding their knowledge, and use it to process those requests which are eligible.

Building such a system may require provision of *Safe Harbor* [16] where the system designer would be protected from the charges of any contributory infringement. The human mediators in TPRS may not be subject to such charges as long as they make their decisions on the fair use requests by following the fair use guidelines of the copyright law. On the other hand, a system that automatically evaluates the fair use requests is subject to infringement claims because of the algorithms used in the evaluation. Such algorithms may be open to scrutiny, and subject to disputes as to whether they accurately reflect the fair use given the vagueness of the fair use factors. Even a fuzzy algorithm that tries to model such vagueness may be disputed on the ground whether the fuzziness specified in the algorithm is reflective of the fair use judgment process. Safe harbor provisions protect the DRM systems from such infringement claims by establishing boundaries of *safe* fair use rights. The two issues in establishing a safe harbor, as discussed by Fox et. al. [16] are:

- creating machine-interpretable expressions that adequately model a set (or subset) of the fair use rights;
- getting the stake holders (content owners, DRM system builders, and Congress, as the representative of the people's interest in the social bargain of copyright) to work together on defining the boundaries of a subset of the fair use rights that would be safe to implement.

We need a safe harbor for the DRM system designers where they could implement the fair use features that are declared non-infringing, thus protecting the designers from the charges of contributory infringement with respect to any action grounded in the safe harbor rights. The safe harbor could provide the opportunity to bring the DRM systems one step closer in the efforts to implement and enforce the delicate balance inherent

in the copyright law.

A on-going collaboration between the copyright stake holders to create safe harbor could go a long way in the form of a series of expanding safe harbor rights to model larger and larger subsets of the fair use rights in the DRM systems.

6.6 Summary of Contribution

I conclude this thesis with a summary of my contribution:

- I designed a *Fair Use Information Model* to model the information requirements for the four factors of fair use. A *Fair Use Request Interface* generator was designed, based on that model.
- I analyzed a load-balancing algorithm by Song et. al. [42], for assigning the fair use requests among the request handlers. The algorithm was modified to introduce the concept of "other responsibilities", and a simulator was implemented to simulate and analyze the performance of the algorithm.
- The FURI and the load-balancing components above were used in the design of the process flow for the *Fair Use Request Process* between the TPRS and the users.
- I designed the *Shibbot* architecture based on the introduction of the TPRS in the Shibboleth architecture. I analyzed the issues in the resulting architecture, and showed how new shibbotized resources may be added while keeping the existing shibbolized resources.

Bibliography

[1] American Library Association. Information on USA Patriot Act. In *http://www.ala.org*, 2001.

[2] Dan L. Burk and Julie E. Cohen. Fair Use Infrastructure for Rights Management Systems. *Harvard Journal of Law & Technology*, 15(1):41–83, Fall 2001.

[3] Julie E. Cohen. DRM and Privacy. In *Communications of the ACM*, pages 47–49, April 2003.

[4] M. Colajanni, P.S. Yu, and V. Cardellini. Dynamic Load Balancing on Web-Server Systems. In *IEEE Internet Computing*, pages 28–39, May 1999.

[5] National Research Council et al. *The Digital Dilemma: Intellectual Property in the Information Age*. National Academy Press, 2101 Constitution Ave., NW Box 285, Washington, DC 20055, 2000.

[6] D. Eager, E. Lazowska, and J. Zahorjan. Adaptive Load Sharing in Homogeneous Distributed Systems. In *IEEE Transactions on Software Engineering*, pages 662–675, May 1986.

[7] Paul England, John DeTreville, and Butler Lampson. Patent #6330670: Digital Rights Management Operating System, December 2001.

[8] Marlena Erdos and Scott Cantor. Shibboleth-Architecture DRAFT v05. In *http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-arch-v05.pdf*. Internet2/MACE, May 2002.

[9] John S. Erickson. Invited talk: Tools and Services for Web-based Rights Management. In *WWW8 Workshop W7: Managing Intellectual Content on the Web: Use of the Digital Object Identifier (DOI)*, Toronto, Canada, May 1999.

[10] John S. Erickson. A Digital Object Approach to Interoperable Rights Management Fine-grained Policy Enforcement Enabled by a Digital Object Infrastructure. *D-Lib Magazine*, 7(6), June 2001.

[11] John S. Erickson. OpenDRM: A Standards Framework for Digital Rights Expression, Messaging and Enforcement. In *NSF Middleware Initiative and Digital Rights Management Workshop*, September 2002.

[12] John S. Erickson. Fair Use, DRM, and Trusted Computing. In *Communications of the ACM*, pages 34–39, April 2003.

[13] Karl L. Ginter et al. Patent #6185683: Trusted and Secure Techniques, Systems and Methods for Item Delivery and Execution, February 2001.

[14] Mairead Martin et al. Federated Digital Rights Management - A Proposed DRM Solution for Research and Education. *D-Lib Magazine*, 8(7/8), July/August 2002.

[15] Edward W. Felten. A Skeptical View of DRM and Fair Use. In *Communications of the ACM*, pages 57–59, April 2003.

[16] Barbara L. Fox and Brian A. LaMacchia. Encouraging Recognition of Fair Uses in DRM Systems. In *Communications of the ACM*, pages 61–63, April 2003.

[17] Jean-Claude Guedon. In Oldenburg's Long Shadow: Librarians, Research Scientists, Publishers, and the Control of Scientific Publishing. *Association of Research Libraries Proceedings of the 138th Annual Meeting*, May 2001.

[18] Carol C. Henderson. Libraries as Creatures of Copyright: Why Librarians Care about Intellectual Property Law and Policy. *http://www.ala.org/washoff/copylib.html*, November 1998.

[19] Perry Hoberman. Infringement Series. In *http://www.perryhoberman.com/accept/html/infringement.html*, February 2003.

[20] Renato Iannella. *Open Digital Rights Language (ODRL) Version 1.1*. IPR Systems, http://www.w3.org/TR/2002/NOTE-odrl-20020919, November 2001.

[21] Alex Illiev and Sean Smith. Privacy-Enhanced Directory Services. In *2nd Annual PKI Research Workshop*, April 2003.

[22] Alex Illiev and Sean W. Smith. Prototyping an Armored Data Vault: Rights Management on Big Brother's Computer. In *Privacy-Enhancing Technology*, April 2002.

[23] ContentGuard Holding Inc. *eXtensible Rights Markup Language (XrML) 2.0 Specification*. http://www.xrml.org, November 2001.

[24] ContentGuard Holding Inc. *XrML 2.0 Technical Overview*. http://www.xrml.org/reference/XrMLTechnicalOverviewV1.pdf, March 2002.

[25] D. Mulligan J. Erickson and A. Burstein. *Supporting Limits on Copyright Exclusivity in a Rights Expression Language Standard*. A Requirements Submission To the OASIS Rights Language TC on Behalf of The Samuelson Law, Technology & Public Policy Clinic and the Electronic Privacy Information Center, Berkeley, CA, August 2002.

[26] Marc A. Kaplan. *IBM Cryptolopes^{TM}, SuperDistribution and Digital Rights Management*. http://www.research.ibm.com/people/k/kaplan/cryptolope-docs/crypap.html.

[27] Charlie Kaufman, Radia Perlman, and Mike Speciner. *Network Security: Private Communication in a Public World*. Prentice Hall, Inc., Eaglewood Cliffs, NJ, First edition, 1995.

[28] C. Kim and H. Kameda. An Algorithm for Optimal Static Load Balancing in Distributed Computer Systems. In *IEEE Transactions on Computers*, volume 41, pages 381–384, 1992.

[29] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. Number 0-8493-8523-7. CRC Press, 2000 NW Corporate Blvd, Boca Raton, FL, October 1996.

[30] M. Mitzenmacher. The Power of Two Choices in Randomized Load Balancing. *Ph.D. Thesis, UC Berkeley*, September 1996.

[31] Sally Morris et al. "BIC/EDItEUR/NISO Rights Metadata Working Party - Interim Report". Technical report, http://www.bic.org.uk/rightsmd.rtf, June 1998.

[32] Association of American Publishers Inc. *Digital Rights Management for Ebooks: Publisher Requirements version 1.0*. http://www.publishers.org/digital/drm.pdf.

[33] Jaehong Park and Ravi Sandhu. Originator Control in Usage Control. In *Proceedings of* 3^{rd} *International Workshop on Policies for Distributed Systems and Networks*, pages 60–66, June 2002.

[34] Jaehong Park and Ravi Sandhu. Towards Usage Control Models: Beyond Traditional Access Control. In *ACM Symposium on Access Control Models and Technologies*, pages 57–64, June 2002.

[35] Jaehong Park, Ravi Sandhu, and James Schifalacqua. Security Architectures for Controlled Digital Information Dissemination. In *Proceedings of the Sixteenth Annual Computer Security Applications Conference*, pages 224–235, December 2000.

[36] Russell Perry and Matthew M. Williamson. A Licensing and Payment System for Distribution of Digital Content. Technical Report HPL-2002-167, HP Labs, 2002.

[37] Thomas A. Peters. Gutterdammerung (Twilight of the Gutter Margins): e-books and Libraries. *Library Hi Tech*, 19(1), 2001.

[38] Bill Rosenblatt. *Digital Rights Management Business and Technology*. Hungry Minds Inc., New York, NY, first edition, 2002.

[39] Mema Roussopoulos and Mary Baker. Practical Load Balancing for Content Requests in Peer-to-Peer Networks. Technical Report cs.NI/0209023, Stanford University, 2003.

[40] Godfrey Rust. `<indecs>2rdd`: Specification and Model. 2rdd Submission to MPEG's CfP for RDD/REL (M7610/B).

[41] Pamela Samuelson. DRM {and, or, vs.} the Law. In *Communications of the ACM*, pages 41–45, April 2003.

[42] Baoyan Song, Ge Yu, Dan Wang, Derong Shen, and Guoren Wang. An Efficient User Task Handling Mechanism Based on Dynamic Load-Balance for Workflow Systems. In *5th Asia-Pacific Web Conference (APWeb 2003)*, pages 483–494, Xian, China, April 2003.

[43] Mark J. Stefik. Patent #5715403: System for Controlling the Distribution and Use of Digital Works having Attached Usage Rights Where the Usage Rights are Defined by a Usage Rights Grammar, February 1998.

Glossary of Terms

AA - Attribute Authority (p. 54)

AQM - Attribute Query Message (p. 54)

ARM - Attribute Request Message (p. 54)

ARP - Attribute Release Policy (p. 54)

CS - Control Set (p. 6)

DRM - Digital Rights Management (p. 1)

EE - The architecture in the DRM taxonomy (p. 5) with external repository and embedded control set (p. 5)

EF - The architecture in the DRM taxonomy with external repository and fixed control set (p. 5)

EN - The architecture in the DRM taxonomy with external repository and no control set (p. 6)

ER - External Repository (p. 6)

EX - The architecture in the DRM taxonomy with external repository and external control set (p. 6)

FDRM - Federated Digital Rights Management (p. 54)

FURI - Fair Use Request Interface (p. 18)

HS - Handle Service (p. 53)

ME - The architecture in the DRM taxonomy with message push and embedded control set (p. 5)

MF - The architecture in the DRM taxonomy with message push and fixed control set (p. 5)

MN - The architecture in the DRM taxonomy with message push and no control set (p. 6)

MP - Message Push (p. 6)

MX - The architecture in the DRM taxonomy with message push and external control set (p. 6)

ODRL - Open Digital Rights Language (p. 23)

P/LS - Packaging and Licensing Services (p. 58)

PLS - Packaging and Licensing Services module in FDRM architecture (p. 55)

RAA - Resource Attribute Authority (p. 55)

rdd - Rights Data Dictionary (p. 30)

REL - Rights Expression Language (p. 5)

RM - Resource Manager (p. 54)

RMP - Rights Messaging Protocol (p. 5)

SHAR - Shibboleth Attribute Requester (p. 54)

Shibboleth - An architecture to support inter-institutional sharing of resources that are subject to access control (p. 1)

Shibbolized - A Shibboleth term used to describe the resources in resource provider's domain in Shibboleth (p. 53)

Shibbot - Shibboleth architecture with Third Party Rights Service (p. 62)

Shibbotized - A term used to describe the resources in resource provider's domain in Shibbot (p. 62)

SHIRE - Shibboleth Handle Indexical Reference Estabilisher (p. 53)

SHOAR - Shibboleth Object Attribute Resolver (p. 55)

TPRS - Third Party Rights Service (p. 15)

VM - Virtual Machine (p. 6)

WAYF - "Where Are You From" module in Shibboleth (p. 53)