Managing Digital Rights Using JSON

Stephen Downes, Luc Belliveau, Saeed Samet, Md. Abdur Rahman, Rodrique Savoie

Abstract

Prior art in the expression of digital rights using XML is demonstrated to require a process of interpretation or parsing, as is characteristic with language processing, and in addition to be subject to the cross-domain scripting problem. The expression of digital rights using JSON, however, represents a novel approach, bypassing the need for language processing, and in addition, solving the cross-domain scripting problem.

1. Overview

Significant resources are expended in the tracking and registering of copyrights associated with digital resources. A person desiring to access and use a digital resource must manually confirm having the right to do so, which increases the cost of using resources, even free resources. Mechanical systems have been proposed, but these are subject to certain patent restrictions and to significant processing overhead. This again increases the cost of using resources.

In this paper, an alternative mechanism for managing copyrights is proposed. It offers a low-cost and distributed mechanism that can be maintained for little cost and effort. It is a novel solution, providing significant advantage over existing mechanisms. It takes advantage of existing capability built into web browsers and other internet reading devices.

This paper describes the use of JavaScript Object Notation – JSON – for the purpose of managing digital rights. It demonstrates a functioning system whereby the location of rights information, expressed in JSON, is stored in the metadata describing a resource, available in the search process that reveals that resource, and used to inform client-based processes that use the resource.

2. What is JSON?

JavaScript (not to be confused with Java) is an object-oriented programming language used primarily to manage web browser interactions with web content. JavaScript is typically located on a web page and is read and processed entirely within the browser accessing that web page. This is a critical security feature of JavaScript. JavaScript applications work within the browser only; files are not saved outside the browser environment, and processes inhabiting one web page are insulated from those inhabiting another page.

This makes JavaScript very different from other programming languages or environments. All other web processing is managed either by server-side scripts, such as .asp pages, Java server environments, or CGI scripts, or by browser plug-ins, such as Flash, Java applets, or QuickTime. None of these server side or plug-in applications has JavaScript's security constraints. Additionally, all of them depend on the creation and processing of compiled computer code that is run outside the browser environment. As such, none of them is a part of the web page itself, with direct access to the Document Object Model (DOM), as JavaScript is.

JSON is the notation used to store data inside JavaScript. Strictly speaking, JSON is not a programming language at all. It is a method for defining part of the Document Object Model directly. Like other parts of JavaScript, JSON may be encoded directly as a part of a web page or imported from an external file. JSON files are plain text ASCII files and can be created using any common word processor. Unlike computer code (such as found in plug-ins or server side processing) JSON is never compiled and executed.

The syntax employed in a JSON object is used to identify parts of the data. The basic structure of a JSON object is to connect labels to data. For example, an expression such as

name:Stephen

is a typical JSON construction. Such a construction is in no way processed or evaluated; it is, rather, the *subject* of processing or evaluation.

The following example demonstrates the employment of brackets and quotation marks in a JSON object:

```
{"menu": {  "id": "file",  "value": "File",
  "popup": {    "menuitem": [
     {"value": "New", "onclick":
"CreateNewDoc()"}, {"value": "Open",
"onclick": "OpenDoc()"}, {"value":
"Close", "onclick": "CloseDoc()"} ]  }}}
```

(Source: http://www.*json.org/example.html)*

As can be seen from the example, the employment of brackets and quotation marks is used to delimit names and values. They serve no *processing* function. Rather, they create a nested, hierarchical structure in JSON data. Specifically, the value of a JSON label may be either (a) a string, denoted with quotation marks (“”), (b) a set of JSON objects, denoted with curly brackets ({}), or (c) an ordered list of JSON objects, denoted with square brackets ([]}.

Again, note, a JSON object is not parsed or interpreted in any way. When it is included in a web page, it is a part of the web page, contained inside the Document Object Model.

3. Rights Expressions

A *rights expression* is a statement of the permissions and duties associated with the use of a resource. An expression may vary from fully restrictive (for example: “All rights reserved”) to fully permissive (for example: a statement declaring the resource to be in the public domain). In Canada and the United States, a rights expression in not *required*; resources are assumed upon creation to be fully copyrighted by their creators. Nor are rights expressions fully *stipulative*; a rights expression does not supersede either the expiry of copyright or the freedoms allowed under fair dealing or fair use.

That said, rights expressions are widely used in order to modify the presumptions ordinarily made under law, and in particular, to clarify ownership of copyright (which may have been assigned by the creator) and to allow specific uses not automatically granted under copyright. Instances of rights expressions include *licenses* and *assignments* of copyright or ownership. Rights expressions have elements in common, and these elements are replicated digitally in the form of rights expression languages.

1. *Rights holder* – the identity and coordinates of the person or entity holding copyright over a resource
2. *Resource* – the unique identity of the resource over which copyright is held
3. *Action* – the specific use to which the resource will be put (for example, ‘displayed’ or ‘printed’ or ‘resold’)
4. *Condition* – the constraint or duty applied to a user who desires to perform the action with the specified resource

In rights expressions, the assignee is implicit, and is presumed to be the reader of the rights expression. A rights *contract* may exist when a document specifies both the rights holder and the assignee.

Examples of rights expressions may be found in both the Open Digital Rights Language (ODRL) and the MPEG Rights Expression Language (MPEG-REL, formerly XrML). In ODRL, they are known as ‘party’, ‘asset’, ‘permission’ and ‘prohibition’ respectively. (Ianella & Guth, 2007) And in MPEG-REL they are called ‘issuer’, ‘resource’, ‘right’ and ‘condition’ respectively (in MPEG-REL, ‘right’ and ‘condition’ taken together are called a ‘grant’). (Wang, deMartini, Barney, Paramasivam, & Barlas, 2005)

Probably the most widely used rights expression today is the Creative Commons rights expression. At least 130 million works are licensed under Creative Commons. (Creative Commons, 2009) Creative Commons is a set of licenses that express different sets of permissions on resources. Licenses may be varied by allowing or not allowing derivatives or commercial usage, and by requiring or not allowing attribution and sharing using the same license. (Creative Commons, 2009) These correspond, respectively, to actions and constraints. Creative Commons licenses are represented using both plain text and legal text descriptions. Additionally, the Creative Commons Rights Expression Language (ccREL) is now a W3C submission. (World Wide Web Consortium, 2008) The rights expression model identifies two major elements: work properties, and license properties. License properties express the

actions and constraints; work properties express the work and the rights holder. (Cover, 2008)

4. Prior Art of Rights Expression

Numerous patents and patent applications exist that describe mechanisms for expressing and enforcing digital rights. Various licensing schemes exist. For example, in a European patent application filed by Lucent, EP20060744803, "a system checks for content rights and if a license is required, then the license is obtained and stored as a persistent file. (Cookson & Furlong, 2006) And ContentGuard has a series of patents related to licensing systems and access controls. Patent US 7290699, for example, describes a centralized rights repository that issues licenses to qualified clients to enable them to open content stored on trusted clients. (Reddy, Lao, & Budo-marek, 2005)

Various applications propose the employment of a plug-in or some other support software to run alongside the web browser. For example, another ContentGuard patent, US 7237125, describes "a client device, having a standard application program for accomplishing a task related to the product [a browser or viewer of some sort] and a rights management module operatively coupled to the server and said client device and configured, upon a request to access the content, to determine if security components are coupled to the application program." (Raley, Chen, Wu, & Ta, 2003) Similarly, a patent application filed in 2001 by IBM contemplates the use of a Java Virtual Machine (JVM) alongside the browser application in order to manage access. (Koved, Mourad, Munson, Pacifici, Pistoia, & Youssef, 2001)

Probably the most significant of the patents extant are held by ContentGuard, including one in which "a rights expression system and method for facilitating creation and/or modification of rights expressions in a rights expression language based on one or more schemas are provided." (Nguyen, Fung, Yee, & Tran, 2002) The patent describes a mechanism for representing "rights expressions in a rights expression language (REL)." The extent of this patent is not clear; there is no extant legal action from ContentGuard, either with respect to the use of ODRL (as for example) by the Open Mobile Alliance, or with respect to Creative Commons licenses, which have both made use of rights expressions. In the documentation, the application asserts that "a rights expression is a syntactically and semantically correct language construct, based on a defined grammar that conveys rights information". It is clear that the authors have the use of XML in mind, as the example provided is of XrML, and as the languages are informed by XML schema.

Another ContentGuard patent describes "a computer implemented method for processing a rights expression for association with an item for use in a digital rights management system." (Ta & et.al., 2003) This patent, like the other ContentGuard patents, is specific to "grammar-based language wherein said rights expression specifies a manner of use of said item for enforcement on a device, and said rights expression is encoded with a grammar-based expression language," which is what it understands XML to be.

Though the patents are intended to be general, it is clear that they were developed with XML in mind and were written so as to define the term "markup language" as generally as possible. However, the patents cover specific actions based on the nature and structure of XML, not the use of XML itself. This is seen by the continued filing of patents related to the use of XML in rights expression; for example, a European patent application filed by Vodaphone, EP1638292, refers to a 'rights object', "for example, an XML document expressing permissions and constraints associated with a piece of DRM content." (Irwin, Wright, & Mulligan, 2005)

5. Using XML Rights Expressions

All three of XrML, ODRL and ccRML are expressed in a form of XML. XML, which stands for eXtensible Markup Language, is a World Wide Web Consortium recommendation. (World Wide Web Consortium, 2008)

XML may be used in a web environment in one of two major ways:

1. Translation – the XML data is directly translated, using XSL Transformations (XSLT), into a viewer-friendly format, such as HTML, which can be used

by web browsers. (World Wide Web Consortium, 2009) The idea of XSLT is that data contained in a single XML document may be presented to readers in a variety of document formats

2. Parsing – the XML data is indirectly translated, using an XML parsing function, into data that may be managed by computer programs, such as database tables. An example of an XML parser is the Universal XML parser, authored by Mark Pilgrim. (Pilgrim, 2006) A parser translates the XML into a data structure; this data structure is then used by the computer code for subsequent processing or display. Parsing functions may use XSLT documents to manage parsing, or they may use native parsing functions, such as the Simple API for XML (SAX), for custom parsing applications. (Megginson, 2004)

XML may be used in one of two major processing environments:

1. Server side – XML is processed by a web data provider, such as a web site or web service. The web data provider acts as an intermediary between the ultimate data user (the 'client') and the XML data. The XML data is retrieved by the web provider, parsed or translated, and then processed or presented to the client.

2. Client side – XML is processed by the client interface program. This is typically, but not necessarily, the web browser. The XML and (if necessary) XSLT is accessed directly by the client software, translated or parsed, and then processed or presented to the client. Client side processing may be accomplished in one of two ways:

a. Natively – that is, the XML is processed by the web browser or client software directly, without access to external software. A 'native' web browser is definable as an application that instantiates all or part, and only, the specifications described by the World Wide Web Consortium. Examples of native XML processing would be processing using XSLT or processing using JavaScript. (World Wide Web Consortium, 2009)

b. With Support – a supplementary program, known variously as a 'helper application' or a 'plug-in', is used by the web browser to translate or parse XML. Examples of plug-in or helper applications include those authored in Microsoft Silverlight, Adobe Flash or Java, or readers such as the Adobe Reader. These applications form separate processes outside the web browser process, and bypass or ignore web browser limitations.

There are two major web browser limitations. These limitations are imposed by the World Wide Web Consortium in order to maintain browser security:

1. Beyond very narrowly defined 'cookie' files, web browsers may not alter the file system of the client computer.

2. Web application technologies apply origin restrictions to network requests. Specifically, these limitations prevent data from one origin from being used by processes or another origin. (World Wide Web Consortium, 2008)

These constraints create what is commonly known as the 'cross domain scripting problem'. (Kraan, 2003) In essence, either XML and native processing (such as XSLT or JavaScript) must originate from the same domain, or non-native parsing of the XML file must be performed, either by a plug-in or by a server side process. There is, according to the cross domain scripting problem, no way to process XML files using only the web browser.

In the sections that follow, the response to the cross-domain scripting problem is described. It proceeds in two major steps: first, the employment of JSON, rather than XML, to express digital rights, and second, the employment of the 'tag hack' to manage the transfer of rights information across domains.

6. The 'Tag Hack

The 'tag hack' is a mechanism for placing JavaScript data from one domain into a web page that originates in another domain. It is essentially a direct response to the cross domain scripting problem. (Crockford, 2009)

JavaScript scripts may be embedded into an HTML web page using the <script> tag. Specifically, the 'src' attribute may be used to specify a JavaScript to be run on the page: <script src="url" /> where 'url' specifies the web address of the script to be placed

onto the page. This script is *not* subject to cross-domain limitations, which means that the script may originate on one domain and the web page on a second domain.

JavaScript enables the direct writing of parts of web pages. One way it accomplishes this is with the document.write() function. The command: document.write("This is some text"); will write "This is some text" to a web page.

This becomes key to the cross domain scripting problem. Under certain circumstances, data written to the web page using a cross-domain JavaScript script will be considered native to the domain of the *web page*, and not the remote script. This allows data from one web site to be processed by a web page from a second web site.

In particular, data structures in JSON defined by a JavaScript script will be considered native to the web page by the web browser. Hence, the JavaScript script, located on one domain, is able to insert data, in the form of a JSON data object, into a web page located on another domain.

If data is encoded using JSON instead of XML, the cross domain scripting problem is solved. Rights expression metadata may be incorporated into a web page, as a part of a web page, using *no processing whatsoever*, and in particular, without using either server-side processing or browser plug-ins or helper applications.

The tag hack is said to raise security concerns, and that these concerns may mitigate in favor of the use of XML rather than JSON. This risk resides, however, not in the data, but in the JavaScript used to read the data. A similar risk exists with XML parsers, and in each case, clients are recommended to run software only from trusted sources.

7. Rights Models

Rights may be associated with digital resources in one of two major ways: either the rights expression accompanies the resource as a part of the resource itself or of a content package containing the resource or the rights expression resides in some other location and is pointed to or referenced by the resource or a package containing the resource.

Moreover, the rights expression associated with a given resource may be one of two types: it may be either an *offer*, which is to say, a description of the conditions under which, if accepted, the resource may be used, or it may be an *agreement*, which is to say, an assertion that permission has been granted to use the resource in the described manner. (Ianella & Guth, 2007)

In an offer, certain parts of the rights expression remain blank. Because the client, who would accept the offer, is not yet identified, is not named. Thus while an agreement may say, in essence: " Fred grants Tom the right to read but not copy document 4323," an offer will leave one part of the expression undefined: "Fred grants *x* the right to read but not copy document 4323."

In the same way, a resource owner may want to apply an offer to any of a number of resources. In such a circumstance, another part of the rights expression will remain undefined, the specification of the resource itself:

Fred grants *x* the right to read but not copy document *d*.

Expressions of this form are known as *rights models*. A rights model identifies the owner of a resource and a set of zero or more licenses (specifically, statements defining actions and conditions) that are on offer with respect to the resource.

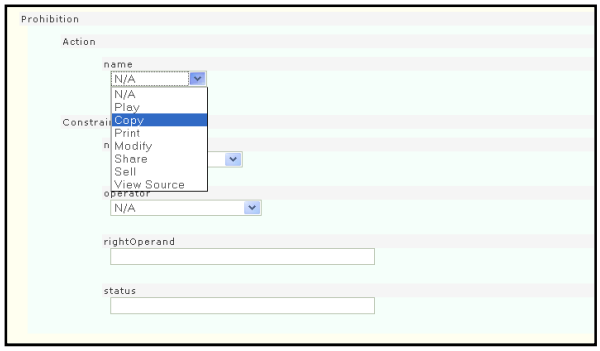

Figure 1. Excerpt of ODRL Rights-Model Creation Dialogue. Full dialogue may be viewed at http://dev-synergic.elg.ca/drm/prototype/rightsmodel.php

8. Expressing of Digital Rights Using JSON

In our implementation of digital rights expression using JSON, we use a heavily modified ODRL profile, attached as Appendix 1 (below). This schema is used to define a data entry form in a web page; by entering values in the form, a user creates a new digital rights model.

Rights models that have been created appear as selection options in the learning object metadata (LOM) creation form. The URL of the rights model is placed into the rights.description LOM element.

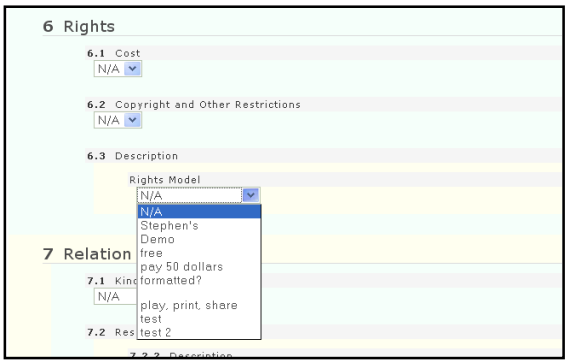

Figure 2. Selection of rights model in LOM metadata creation dialogue. Full dialogue may be viewed at http://dev-synergic.elg.ca/drm/prototype/lom.php

The resulting LOM XML reads as follows:

```
<rights><cost></cost>
  <copyrightAndOtherRestrictions/>
  <description><string>
      http://dev-
synergic.elg.ca/drm/prototype/rights/6_rightsM
odel.js
    </string></description></rights>
```

Figure 5. Except of LOM metadata containing JSON URL.

When the URL is dereferenced (opened by a browser or client application) the following JSON file is accessed. The JSON file will look like the following:

```
drm({"ODRL": {
"Rights": {"uid":"http://dev-
synergic.elg.ca/drm/prototype/rights/2_rightsModel.j
s","rightsModelName":"Stephen's","type":"offer","Pe
rmission": [{
  "Action": {"name":"Play"},
  "Constraint": [{"name":"","operator":"",
"rightOperand":"","status":""}],
  "Duty": [{"relax":"false","Constraint":
[{"name":"Number of Usages","operator":"greater
than", "rightOperand":"","status":""}],
  "Object": {"measure":"","value":""},"Action":
{"name":"CC SourceCode"}}]}],
  "Prohibition": [{"Action":
{"name":""},"Constraint": {"name":"","operator":"",
"rightOperand":"","status":""}}]}})
```

(The original may be viewed at http://dev-synergic.elg.ca/drm/prototype/rights/6_rightsModel.js)

9. Recognizing JSON-Encoded Digital Rights

When a client user desires to view a learning resource, the client first obtains information about the object in the form of learning object metadata (, for example, IEEE-LOM). This metadata has been distributed by the resource publisher to be made available by learning resource search services.

When the resource is selected, the client software then dereferences the JSON file and incorporates it as a part of the JavaScript script functioning in the client web page. The following information (for example) is now available to the client software:

Figure 3. Rights model. See http://dev-synergic.elg.ca/drm/prototype/lom_screen.php?xml=5_learningObject.xml

In this example, the client software is able to detect that permission to print the resource is offered (other permissions, which are not offered, are grayed out).

When the user attempts to view the object, the duties and constraints defined in the JSON rights expression may be applied. In this example, they are displayed as an alert box.

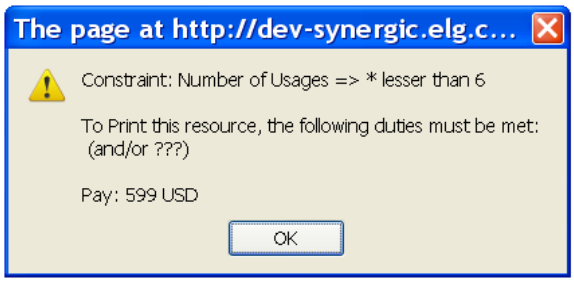

Figure 4. Display of rights duties and constraints

The JavaScript script encoding is attached as Appendix 2.

In an actual client environment, these conditions would be used to initiate other processes. For example, a request for financial compensation may initiate a purchasing service on behalf of the client. The number of usages may be stored as a browser cookie value or (using a web service) recorded on a third party application.

It should be emphasized that these are not digital rights enforcement mechanisms; they are digital rights compliance mechanisms. The distinction is that control of the mechanism is managed by the client. It is possible to circumvent the conditions encoded in the JSON rights expression. The client architecture described here is a mechanism that helps the client comply with the rights as expressed, nothing more.

That said, such a mechanism would be consistent with a more robust application that supports access controls. Because the client has rights information (JSON) and publisher information (contained in the LOM), the client could contact a web service to negotiate access to the resource. In an earlier prototype of this model, the eduSource Distributed Digital Rights system (Downes, 2004) the client would contact a broker who would, after executing a financial transaction, supply the client with a key, which would provide access to the resource through a proxy server hosted by the publisher.

10. Expressing Rights Using JSON: A Novel Approach

Although JSON is a fully enclosed subset of the JavaScript programming language, it should be clear that the notation – consisting entirely of commas, colons and brackets – is not itself a *language*, but rather, a shell or enclosure used to store data. It bears the same relation to JavaScript that the use of tabular data bears to the English language. JSON cannot be understood as having a semantics *per se*; it is simply a container for holding data, and is neutral regarding the expressive impact of that data.

Even if JSON is understood as a language, however, more significant is the fact that it is not *used* as a language in the digital rights expression system described here. In the prior art described by the various patents, and in the expression of digital rights using XML as described immediately thereafter, it is clear that the rights expression must, as is common for expressions in a language, be the subject of *translation* or *parsing*. This is what characterizes expressions in XML as a *language* and not merely as *data stacks*. And this is what constitutes the 'art' in the prior art: the processing required to interpret expressions in the language as statements about digital rights.

No such processing is required in the system using JSON. In addition to offering a more efficient mechanism for displaying and using rights data, and in addition to providing a unique solution to the cross-domain scripting problem, the JSON approach eliminates the need for any translation or processing. The need for the mechanisms described in the prior art are simply eliminated. Digital rights models are not, technically, *expressions* of digital rights; rather, they are bits of computer code data which become part of the executable JavaScript, the selection of which is tantamount to the expression of digital rights, but which is not.

The proof of this assertion lies in the comparative computability of the two systems. A rights expression in a language in which the terms were replaced with meaningless strings of 1 and 0 could not be parsed or translated without a data dictionary, and hence, would be inoperative in a rights expression system. This expression: <11>010</11> would be meaningless as a rights expression and unusable to a piece of computer software. However, the same data, expressed in JSON: 11:00 may be directly applied in a piece of JavaScript code and could function with no interpretation, parsing or translation. This is the

fundamental difference between language and data, and represents the reason the expression of digital rights in JSON is a novel approach, distinct from the prior art.

This work is part of the SynergiC3 project (www.synergic3.com), which is partially funded by the Atlantic Canada Opportunities Agency's Atlantic Innovation Fund (AIF), a program designed to encourage partnerships among private sector firms, universities, colleges and other research institutions to develop new products and services. The SynergiC3 project is a Research and Development collaboration between the National Research Council of Canada, Université de Moncton and Desire2Learn Incorporated.

Cookson, R. L., & Furlong, J. (2006). *Patent No. EP20060744803.* Europe.

Cover, R. (2008, August 22). *W3C Member Submission for Creative Commons Rights Expression Language (ccREL)*. Retrieved February 22, 2009, from Cover Pages: http://xml.coverpages.org/ni2008-08-22-a.html

Creative Commons. (2009). *Licenses*. Retrieved February 20, 2009, from Creative Commons: http://creativecommons.org/about/licenses/

Creative Commons. (2009). *Metrics*. Retrieved February 22, 2009, from Creative Commons: http://wiki.creativecommons.org/Metrics

Crockford, D. (2009). *JavaScript and HTML Script Tags*. Retrieved February 20, 2009, from www.crockford.com: http://javascript.crockford.com/script.html

Downes, S. e. (2004). Distributed Digital Rights Management: The EduSource Approach to DRM. *Open Digital Rights Language Proceedings* (p. http://www.downes.ca/files/DDRM_14April2004.pdf). Vienna: ODRL.

Ianella, R., & Guth, S. (2007, January 13). *ODRL V2.0 - Model Semantics: Working Draft: 13th January 2007.* Retrieved January 14, 2009, from ODRL: http://odrl.net/2.0/WD-ODRL-Model-20070113.html

Irwin, J., Wright, T., & Mulligan, C. (2005). *Patent No. 1638292.* Europe.

Koved, L., Mourad, M. M., Munson, J. P., Pacifici, G., Pistoia, M., & Youssef, A. S. (2001). *Patent No. 7308717.* United States.

Kraan, W. (2003, June 21). *A feature or a bug; SCORM and cross domain scripting*. Retrieved February 20, 2009, from CETIS: http://zope.cetis.ac.uk/content/20030622203659

Megginson, D. (2004, April 27). *About SAX*. Retrieved February 20, 2009, from SAX: http://www.saxproject.org/

Nguyen, M., Fung, Yee, J. Z., & Tran, D. (2002). *Patent No. US 7,177,843.* United States.

Pilgrim, M. (2006, January 10). *Universal Feed Parser*. Retrieved february 20, 2009, from Dive Into Mark: http://www.feedparser.org/

Raley, M., Chen, D., Wu, H.-c., & Ta, T. (2003). *Patent No. 7237125.* United States.

Reddy, K. H., Lao, G., & Budo-marek, A. (2005). *Patent No. 7290699.* United States.

Ta, T., & et.al. (2003). *Patent No. US 7,359,884.* United States.

Wang, X., deMartini, T., Barney, W., Paramasivam, M., & Barlas, C. (2005, June). *The MPEG-21 Rights Expression Language and Rights Data Dictionary*. Retrieved January 14, 2009, from IEEE Transactions on Multimedia. Volume 7, Number 3: http://www.contentguard.com/whitepapers/MPEG-21_REL_and_RDD.pdf

World Wide Web Consortium. (2008, September 12). *Access Control for Cross-Site Requests*. Retrieved February 22, 2009, from World Wide Web Consortium: http://www.w3.org/TR/access-control/

World Wide Web Consortium. (2008, May 1). *ccREL: The Creative Commons Rights Expression Language*. Retrieved February 22, 2009, from World Wide Web Consortium: http://www.w3.org/Submission/2008/SUBM-ccREL-20080501/

World Wide Web Consortium. (2008, October 14). *Extensible Markup Language (XML)*. Retrieved February 20, 2009, from World Wide Web Consortium: http://www.w3.org/XML/

World Wide Web Consortium. (2009). *JavaScript Tutorial*. Retrieved February 20, 2009, from W3C Schools: http://www.w3schools.com/JS/default.asp

World Wide Web Consortium. (2009, February 3). *The Extensible Stylesheet Language Family (XSL)*. Retrieved February 20, 2009, from World Wide Web Consortium: http://www.w3.org/Style/XSL/

Appendices

Appendix 1

ODRL-20.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tagger="http://dev-synergic.elg.ca/drm/tagger"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xsd:element name="ODRL">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Rights">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Permission"
minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element ref="Action"
minOccurs="1" />
                    <xsd:element ref="Constraint"
minOccurs="0" maxOccurs="unbounded"/>
                    <xsd:element ref="Duty" minOccurs="0"
maxOccurs="unbounded"/>
                    <!--<xsd:element ref="Party"
minOccurs="0" maxOccurs="unbounded"/>-->
                    <!--<xsd:element ref="Asset"/>-->
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="Prohibition"
minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element ref="Action"/>
                    <xsd:element ref="Constraint"/>
                    <!--<xsd:element ref="Asset"/>-->
                    <!--<xsd:element ref="Party"
minOccurs="0" maxOccurs="unbounded"/>-->
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="uid" type="xsd:string">
            </xsd:attribute>
            <xsd:attribute name="rightsModelName"
type="xsd:string">
            </xsd:attribute>
            <xsd:attribute name="type" type="restrictType"
/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Constraint">
    <xsd:complexType>
      <xsd:attribute name="name"
type="restrictConstraintName" />
      <xsd:attribute name="operator" type="restrictOperator"
/>
      <xsd:attribute name="rightOperand" type="xsd:string"
/>
      <xsd:attribute name="status" type="xsd:string" />
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Action">
    <xsd:complexType>
      <xsd:attribute name="name" type="restrictAction" />
    </xsd:complexType>
  </xsd:element>
  <!--<xsd:element name="Asset">
    <xsd:complexType>
      <xsd:attribute name="uid" type="xsd:string" />
      <xsd:attribute name="inherit" type="xsd:string" />
    </xsd:complexType>
  </xsd:element>-->
  <xsd:element name="Duty">
    <xsd:complexType>
      <xsd:sequence>
        <!--<xsd:element ref="Constraint" minOccurs="0"
maxOccurs="unbounded"/>-->
        <xsd:element name="Action">
          <xsd:complexType>
            <xsd:attribute name="name"
type="restrictDutyAction" />
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Object">
          <xsd:complexType>
            <!--<xsd:sequence>
              <xsd:element ref="Asset" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>-->
            <xsd:attribute name="measure"
type="restrictObjectMeasure" />
            <xsd:attribute name="value" type="xsd:string" />
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <!--<xsd:attribute name="relax" type="xsd:boolean" />-
->
    </xsd:complexType>
  </xsd:element>
  <!--
  <xsd:element name="Party">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Duty"/>
        <xsd:element ref="Asset"/>
      </xsd:sequence>
      <xsd:attribute name="uid" type="xsd:string" />
    </xsd:complexType>
  </xsd:element>-->
```

```
  <xsd:simpleType name="restrictObjectMeasure">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="CAD" />
      <xsd:enumeration value="USD" />
      <xsd:enumeration value="EUR" />
      <xsd:enumeration value="AUD" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="restrictOperator">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="greater than"/>
      <xsd:enumeration value="lesser than"/>
      <xsd:enumeration value="equals"/>
      <xsd:enumeration value="greater than or equals"/>
      <xsd:enumeration value="lesser than or equals"/>
      <xsd:enumeration value="not equals"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="restrictType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="offer" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="restrictAction">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Play" />
      <xsd:enumeration value="Copy" />
      <xsd:enumeration value="Print" />
      <xsd:enumeration value="Modify" />
      <xsd:enumeration value="Share" />
      <xsd:enumeration value="Sell" />
      <xsd:enumeration value="View Source" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="restrictConstraintName">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Number of Usages" />
      <xsd:enumeration value="Expiration Date" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="restrictDutyAction">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Pay" />
      <xsd:enumeration value="Accept" />
      <xsd:enumeration value="Register" />
      <xsd:enumeration value="Attribution" />
      <xsd:enumeration value="Tracked" />
      <xsd:enumeration value="Share Alike" />
      <xsd:enumeration value="Notice" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Appendix 2. JavaScript

```
    <script type="text/javascript">
      <!--
        function drm(json) {
          var objectODRL = json;
          var canPlay = false;
          var can
          var found = false;
          var proceed = false;
          var url = '<? echo textJS($objectURI); ?>';
          for (var i=0;
i<objectODRL.ODRL.Rights.Permission.length; i++) {
            var actionName =
objectODRL.ODRL.Rights.Permission[i].Action.name;
            var btn = document.getElementById('btn' +
actionName);
            if (btn) {
              btn.disabled = false;
              var msg = '';
              if
(objectODRL.ODRL.Rights.Permission[i].Constraint.length !=
0) {
                for (var x=0;
x<objectODRL.ODRL.Rights.Permission[i].Constraint.length;
x++) {
                  var constraintName =
objectODRL.ODRL.Rights.Permission[i].Constraint[x].name;
                  var constraintOperator =
objectODRL.ODRL.Rights.Permission[i].Constraint[x].operat
or;
                  var constraintRightOperand =
objectODRL.ODRL.Rights.Permission[i].Constraint[x].rightO
perand;
                  var constraintStatus =
objectODRL.ODRL.Rights.Permission[i].Constraint[x].status;
                  if (constraintStatus == '') constraintStatus
= '*';
                  if (constraintName != '') msg +=
'Constraint: ' + constraintName + ' => ' + constraintStatus + ' '
+ constraintOperator + ' ' + constraintRightOperand + '\\n';
                }
              }
              if (msg != '') msg += '\\n';
              msg += 'To ' + actionName + ' this resource,
the following duties must be met: \\n (and/or ???)\\n\\n';
              var output = false;
              for (var x=0;
x<objectODRL.ODRL.Rights.Permission[i].Duty.length; x++)
{
                var dutyObjectValue =
parseInt(objectODRL.ODRL.Rights.Permission[i].Duty[x].Obj
ect.value);
                var dutyObjectMeasure =
objectODRL.ODRL.Rights.Permission[i].Duty[x].Object.meas
ure;
                var dutyActionName =
objectODRL.ODRL.Rights.Permission[i].Duty[x].Action.name
;
                msg += dutyActionName;
                if (dutyObjectValue >= 0) {
                  msg += ': ' + dutyObjectValue + ' ' +
dutyObjectMeasure + '\\n';
                  output = true;
                }
              }
              if (output == false) msg += 'None.\\n';

              eval("btn.onclick = function() { alert('" + msg +
"');  }");
            }
          }
        }
      -->
    </script>
```