

A Domain Specific Language for Usage Management

Christopher C. Lamb, Pramod A. Jamkhedkar, Viswanath Nandina, Mathew P. Bohnsack, Gregory L. Heileman

University of New Mexico
Department of Electrical and Computer Engineering
Albuquerque, NM 87131-0001

{cclamb, pramod54, vishu, mbohnsack, heileman}@ece.unm.edu

ABSTRACT

In the course of this paper we will develop a domain specific language (DSL) for expressing usage management policies and associating those policies with managed artifacts. We begin by framing a use model for the language, including generalized use cases, a loose ontology, an general supported lifecycle, and specific extension requirements. We then develop the language from that model, demonstrating key syntactic elements and highlighting the technology behind the language while tracing features back to the initial model. We also compare and contrast this DSL with others developed for rights management (e.g. Ponder). We then demonstrate how the DSL supports common usage management and DRM-centric environments, including creative commons, the extensible rights markup language (XRML), and the open digital rights language (ODRL).

Categories and Subject Descriptors

D.2.12 [Software Engineering]: Interoperability—*Distributed Objects*; F.4.3 [Mathematical Logic and Formal Languages]: Formal Languages—*Security Policy*

General Terms

Design, Languages, Theory, Security, Policy Languages

Keywords

Access Control, Interoperability, DRM

1. INTRODUCTION

For the purposes of this paper, we current define usage management as the management of the usage of resources (and data) across and within computing environments. More than access control or digital rights management, usage management concerns itself with fine-grained control of all aspects of how a given digital resource is used. As digital environments become more open over time, the need for usage management for resources that span utility computational

environments (e.g. cloud provider systems) will become increasingly important.

With the advent and widespread use of cloud computing, those responsible for a given usage managed resource are almost never those responsible for the computing systems, except at edge devices like mobile phones or other small profile computing devices. Resources are regularly moved across national boundaries and regional areas without either the content owner's or creator's knowledge. Furthermore, this kind of transfer is generally according to pre-established algorithms or data routing protocols over which users of all stripes have no control. Managing these kinds of issues requires new usage management capabilities that can run on platforms ranging from small, hand-held devices to nodes in large data centers.

Historically research in this area has been focused on developing more expressive policy languages via either different type of mathematical logics or formalisms with greater reasoning capabilities [6, 7, 13, 18, 19, 34, 39]. These approaches however fail to address interoperability challenges posed by new commercially available distributed computing environments. Interoperability efforts have resorted to translation mechanisms, where the policy is translated in its entirety to a different language [20, 33, 37]; it has been shown recently however that such techniques are infeasible and hard to perform for most policy languages [28, 36]. Other approaches have led to complex policy specification languages that have tried to establish themselves as the universal standard [1, 2, 38, 40]. This unfortunately tends to stifle both innovation and flexibility [20, 25, 26, 27].

To address these issues, we first applied the principles of system design to develop a framework for usage management in open, distributed environments that supports interoperability. These principles have been used by researchers in large network design create a balance between interoperability and open, flexible architectures [5, 10, 14], allowing for computing scale and power without sacrificing innovation. Initially we standardized certain features of the framework operational semantics, and left free of standards features that necessitate choice and innovation.

We have implemented this framework, including a usage management calculus providing a platform usage management, within a Domain Specific Language (DSL) and associated evaluation environment. The DSL and its environ-

ment implements our previously defined framework, separating various roles needed for distributed policy creation and management, provides the capability to develop executable licenses, and is extensible from both a policy and constraint definition perspective.

In this paper, we will first review the problems in usage management in more detail. Then, in Section 3 we will first review the model we developed to guide the DSL’s syntactic and semantic development. Then, in the next section, we will cover the language itself, how it was developed, and its supporting evaluation environment. We will then close the paper with three specific implementation examples showing how the language and its runtime support usage management scenarios from three different environments — creative commons (CC), the extensible rights markup language (XRML), and the open digital rights language (ODRL).

2. MOTIVATION

Here, we lay out the historical development of usage management and define its scope and constituent elements, tracing its origins to access control and digital rights management. Following this, we explain the challenges that result from the evolving nature in which information is being used across systems.

2.1 Access Control

Access control mechanisms are systems that manage controlled access to resources. The central idea behind access control is that access to a resource is granted depending upon subject attributes, object attributes, and system attributes. The central component of any access control system, is an access control language (ACL) that is used to express rules for granting access to different resources in the system.

Access control policies can be categorized into two types, namely, discretionary access control (DAC) and mandatory access control (MAC) [22]. DAC policies are the policies that are specified by the owner of the resource, based on the users’ attributes. On the other hand MAC policies are made by a central authority and apply to the whole system. A number of access control models have been developed that allow different types of access control in these two modes. The most successful ones being the role-based access control model (RBAC), and the Bell and LaPadula model [8, 9]. The focus of access control models is to capture the different types of relationships between and among a set of resources and a set of users, and express access rules based on those relationships.

Access control mechanisms are generally tightly coupled with the system in which they are deployed. Situations where users identities are not known apriori requires system agnostic authentication mechanism like public key infrastructure.

2.2 Digital Rights Management

Digital rights management (DRM) consists of mechanisms that manage controlled usage of digital resources. The central idea behind DRM is that usage rules for a given resource are specified for a particular user or group of users, and the use of that resource is then managed for a finite period of time. The usage rules generally include a set

of permissions and obligations, along with rules specifying how the permissions may be exercised over a period of time and under what circumstances. DRM also includes mechanisms such as trusted computing that ensure the enforcement of rights on various user systems [4]. The most well known, but unsuccessful, approaches to address this problem are IBM’s Cryptolopes and Microsoft’s Palladium technologies [12, 15, 29]. Given the problem of enforcing DRM, researchers have proposed incentive based, game-theoretic approaches for DRM [21, 42]. The central component of DRM systems is a rights expression language (REL) that is used to express usage rules (or rights) in the form of a license that is generated by a resource owner for a user or a group of users that will use a resource.

Some of the earliest attempts at DRM date back to 1980’s, and involve the development a formal language for legal discourse [11, 31, 30]. At present, creative commons, along with two XML-based RELs, namely, XrML and ODRL, are most commonly used [41, 24]. Both XrML and ODRL have formed alliances with major players in the industry and standards bodies. Semantics of these XML-based languages are informal. A number of approaches using various types of formalisms and approaches have been used to develop formal RELs. Gunter et al. [17] and Pucella et al. [34] have used trace-based semantics to develop formal RELs. Other formalisms such as first-order logic and CafeOBJ have been used to develop RELs [6, 13, 39]. Many researchers have attempted to provide formal semantics for existing XML based RELs [18, 35, 19]. It is however the general agreement that popular XML-based RELs are difficult to formalize in their entirety [18, 19, 27].

More recently, researchers have tried to expand the the concept of DRM to propose concepts including usage rights management and usage control. The idea of usage rights management is developing along the lines of making users aware of how a resource is supposed to be used [23]. A more formal framework, *UCON_{ABC}* encompassing usage rules and access control has been proposed by Park et al. [32].

2.3 Scope of Usage Management

Before we discuss the scope and components of usage management, it is important to note the difference between ACLs and RELs. Even though the goals of these two types of languages overlap, the focus of research in ACLs and RELs is significantly different. ACLs focus on defining access rules in terms of relationships between sets of resources and sets of users. In DRM systems, once a user obtains a license for a resource, access to that resource is implicit, and what matters is how that resource is used from that point onwards. Therefore, RELs focus on defining different types of usage rules for a given user (or group of users) over a given resource (or group of resources).

We introduce the concept of usage management that is built upon the term *usage control* introduced by Park et al. [32]. The usage control model, called *UCON_{ABC}*, is based on *Authorizations, obligations and Conditions* [32]. *UCON_{ABC}* combines access control and permissions and obligations in a single model.

We now describe the components of usage management as

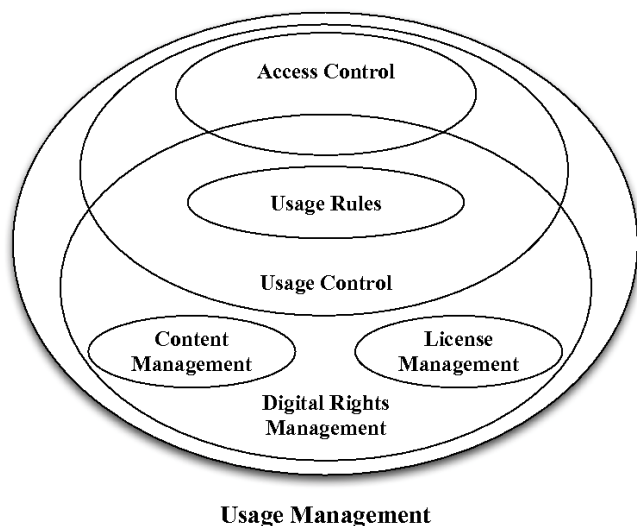


Figure 1: The primary elements in a usage management system.

shown in Figure 1, and how they relate to each other. Usage management is a combination of usage control and DRM. Usage control is a combination of access control and usage rules. Digital rights management includes content management, license management, specification of usage rules and a simplified subset of access control. Content and license management include processes that manage how content and license are bundled, encrypted and distributed or managed across multiple clients. These processes include encryption mechanisms, trust management, trusted computing platforms and other such management techniques. Many RELs, including XrML and ODRL, have tried to incorporate these functionalities.

Next, we discuss challenges in implementing usage management systems, and the need for an actionable framework for addressing the challenges.

2.4 Challenges in Usage Management

Unlike classical access control systems, information is increasingly used across highly networked, distributed computing environments that are not a part of a single centrally managed system. In addition, digital information is increasingly used in innovative ways in which it is transformed, processed or merged with other information while being used across computing environments. This necessitates usage management policies to be tightly coupled with the resource it controls rather than systems upon which that resource may reside. The policy is then interpreted and enforced as the resource moves across different computing environments.

A logical approach to this problem is to build an interpreter for the policy language that runs on the computing environment (or the client), and enforces the policy on the client. This approach has been very successful for access control systems, because access control policies are tightly coupled with computing environments whose nature is known a priori.

However, in usage management scenarios where resources move across environments that are not known a priori, such an approach is infeasible. To address such a situation, each of the computing environments must incorporate an interpreter and enforcement mechanism that is custom built for different policy languages.

One of the solutions to this situation is the use of a standard usage policy language for all types of information management ecosystems. Rights expression languages such as XrML and ODRL have been adopted separately as standards by different industry alliances. The semantics of these XML-based languages are informal, and researches have demonstrated the difficulty involved in providing formal semantics for these languages [19].

At the same time, numerous formal logic-based rights expression languages have been developed by academicians. These languages use different types of mathematical logics to express and reason over various types of usage semantics. Both XrML and ODRL have developed and continue to develop independent of these formal languages, and have not been able to incorporate their expression and reasoning power. Hence, unlike formal access control models, these formal languages continue to remain outside the remit of industry alliances.

As a result, none of these languages are likely to become the *de facto* industry standard in the future. Different information ecosystems will continue to use different policy languages according to the policy expression requirements. Such a fragmented use of policy languages will remain the biggest obstacle to achieving usage management along with unhindered flow of information across highly distributed computing environments. This implied existence of multiple policy languages in such scenarios leads to two clear problems — difficulty supporting multiple languages and lack of interoperability.

Multiple Language Support. If a given computing platform intends to be a part of multiple information ecosystems, it must support the policy languages used by each of these ecosystems. This means that policy language interpreters for each of these policy languages need to be custom built for that particular computing platform. Furthermore, any advances or changes that are made in these policy languages will require corresponding updates in the interpreters used in all the computing platform. This implies that infrastructure as a service providers like Amazon or Rackspace would need to coordinate the installation of support for the same policy languages in virtual machine instances or lock potential clients out of their systems.

Interoperability. The second disadvantage is that even though a given computing platform may support multiple information ecosystems, each of these information ecosystems still operate in complete isolation from one another. Since different ecosystems use different policy languages, licenses expressed within one ecosystem cannot be interpreted in another ecosystem. This prevents resources from moving freely across different ecosystems.

A number of recent papers have provided interesting solu-




Figure 2: Conceptual Model

tions addressing interoperability at this level [3, 16, 28, 36, 37]. The most common approach to interoperability has been translation mechanisms that translate a policy from one language to another language. It is, however, extremely difficult to effect this translation [20]. The Coral and Marlin initiatives have provided architectural solutions to DRM interoperability [3, 16]. In the Coral approach, different licenses for different DRM systems are generated from a common token in accordance with a common ecosystem [16]. In the Marlin approach, licenses are expressed programmatically in the form of control objects to prevent dependence on any one particular REL [3]. Both Coral and Marlin architectures focus on the management of licenses across systems.

The DSL we describe, in tandem with its runtime, embodies an interoperable framework for usage management, unlike Coral and Marlin architectures, that implements a formal calculus to reason about the relationship between a license, a computing environment, and interoperability between them. It incorporates concepts such as programmable licenses and common ecosystems used by Coral and Marlin architectures respectively. The usage management framework design is based on the principles of *design for choice*, eloquently described by Clarke et al. with reference to “tussles” in cyberspace [14]. They explain the importance of identify the locations in the architecture where standards need to be introduced to enable interoperability, and locations where they should *not* be applied to enable innovation and differentiation. This DSL supports that kind of differentiation by supporting for pluggable evaluators that allow users to extend the basic language syntax, semantics, and runtime arbitrarily.

3. LANGUAGE MODEL

In developing this DSL, we needed to have a clear understanding of the specific domain, and develop an appropriate domain model to guide our efforts. Admittedly, there exist many possible models that can describe this area of policy and policy management, and the model that we chose to initially use is purposefully simple to help ease development and implementation efforts. We did however provide arbitrary language-level extensibility to support future extension into more demanding policy implementation areas.

Our expectation, reflected in Figure 2, is that we have created a system through which three primary actors work in tandem to create executable policies that can be converted into licenses in order to manage the use of defined content. The process starts with a context designer, who specifies the environment in which a resource can be used by a specified subject. This context is then applied by a policy designer to create specific policy language, which is then used by a policy author to create a policy. The policy itself is manipulated

behind a defined interface that requires an instantiation of the defined context from a usage management system.

We developed this model to help us understand how policy-centric DSLs would be used, to visualize how the various elements are inter-related, and to clarify important areas upon which to focus effort. Through this model, we were able to conceptualize the initial language structure and generate performance hierarchies, as well as to tailor expected DSL use.

3.1 Expected Use

In order to develop the appropriate DSL giving users the power and expressivity they need to easily express usage management concepts, we begin by developing a model describing how we expect it to be used, and by whom, identifying key functional and non-functional characteristics. We use roles codified as actors to identify the primary user base, and link those roles to specific use cases we expect to be common in day to day DSL use. We also identify common inputs and outputs from expected activities, and show how those input and output elements are related. We finally specify the essential core structure of the DSL, as well as extension points and default implementations of those points.

In general day to day use, we expect that certain activities will be much more common than others. For example, each *policy* requires a *context* in order to both be developed and to run. That *context* describes the actors using an artifact protected by a policy, the artifact itself, and the environment in which the artifact is both expected to be used (during policy design) and is being used (at evaluation). That said, the expectation is that the number of policies is much greater than the number of contexts associated with those policies.

Likewise, we expect that the number of times a policy is evaluated is much greater than the number of times that policy is designed and created. Policies should be read, evaluated, or combined with other policies frequently. This gives us a magnitude ordering for these activities, where the number of supported contexts is much less than the number of created policies, which is in turn much less than the number of times that policy is evaluated or otherwise used.

This has specific implications on both the DSL syntax and performance profile. For example, as it is much more common for policies to be evaluated than contexts to be created, our efforts and tuning the system and increasing performance are best focused on policy evaluation rather than contextual activities. In a similar vein, the language itself should be as simple to comprehend as possible for policies at the expense of contextual elements if necessary.

Figure 3 shows the primary system actors we have identified as well as the use cases with which they will be involved. Actors include:

- *Context Author*. The context author is responsible for defining the context in which a policy will be applied to a given resource. The context itself defines the environment in which the policy executes, the resource to which the policy is applied, and the subject that attempts to use the resource.

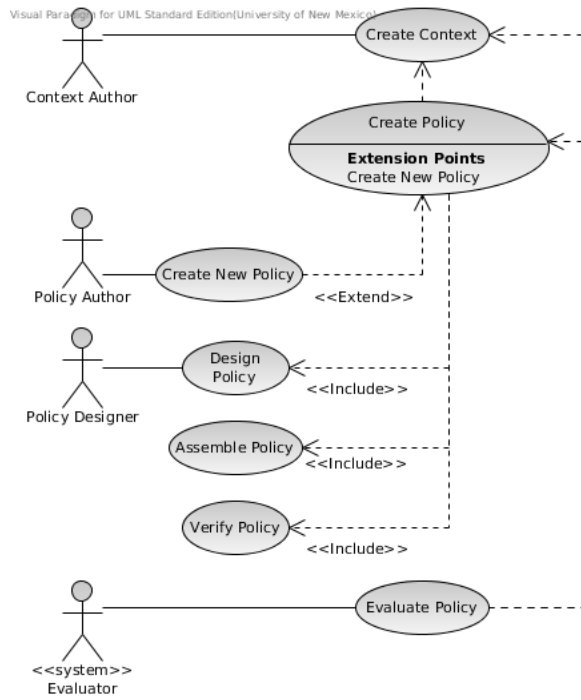


Figure 3: General DSL Use Cases

- *Policy Designer*. The policy designer is responsible for putting together a specification for a given policy that a policy author can use to build an instance of a policy. The person in this role is also responsible for building various policy evaluation components for the DSL if needed.
- *Policy Author*. The policy author creates a policy to control the use of a subject defined in the policy context.
- *Evaluator*. Another system element, an evaluator evaluates a given policy with a specific context.

When a context author creates a context, that author compiles the elements of that context for use both at policy creation and policy execution. When the policy is initially created, the resource is the only defined element. Generally both the subject, representing the eventual user, and the environment, containing information describing the evaluation environment, are only defined at the classifier level. That is to say, they both are defined, but individual properties have yet to be assigned.

Creating a policy is an activity undertaken by a policy author. This step requires a *declared* (but not *defined*) context. This is also undertaken in tandem with some kind of policy specification that describes roughly what the policy should manage and how it should be managed. In the ontology we have defined, this is the step at which the policy designer defines the various constraints, activities, restricted activities, and obligations and develops the appropriate DSL components required to evaluate that policy. Creating a new policy is precisely what it describes - creating a brand new policy applied to a context.

The included cases define policy, assemble policy, and verify policy are common development steps through which the policy is essentially designed, developed, and then tested against a context.

Finally, a policy is evaluated by an evaluator, a system actor, after creation and association with a resource. At this point, the context has a fully instantiated context, with defined resource, subject, and environmental elements.

Now we have a general understanding of the expected use of a given policy, and have defined the expected roles. With this in place, we begin to look at the elements the DSL should have to allow it to express the use cases we expect we need to support as the next step in refining our understanding of what this DSL should look like.

3.2 Domain Model

Our domain model will be the foundation of our DSL. It will allow us to begin to understand the various language elements and how they are related, leading us to an eventual syntax to represent these classifiers and relationships. Not understanding this structure well, or developing a structure that does not support our defined use cases will lead us to develop a DSL that inadequately supports our expected use.

Based on our use cases, we know the model contains a *context*, some kind of policy-specific sub-ontology, and a logic engine that can act over that sub-ontology. Based on our current understanding of our needs, the sub-ontology contains *obligations* and *constraints* applied to *activities*. We use simple first-order logic to reason over the policy elements.

This understanding leads us to the model view in Figure 4. Of special note, the specific policy sub-ontology is contained in the usage semantics package, as is all policy evaluators. Both of these can change within this model to allow for inclusion of more complex policy systems and powerful reasoning capabilities.

Primary elements within this ontology are:

- *Runtime*. This is the system that manages use of a given *resource* by a *subject* in accordance with a *policy*. It is responsible for providing and managing context elements, controlling policies and licenses, and handling requests from subjects. Realizations of this system must be cross platform to support distributed use as well.
- *Context*. The context describes the operating environment of the policy. This information must be available at runtime, and parts of it must be understood when the policy is initially designed. In order to effectively control use of a given artifact, the parameters that artifact can be used under must be understood when the policy is created and must be read when that policy is evaluated.
 - *Environment*. The environment in which a given policy is evaluated. This must be understood in order to constrain the conditions where a policy

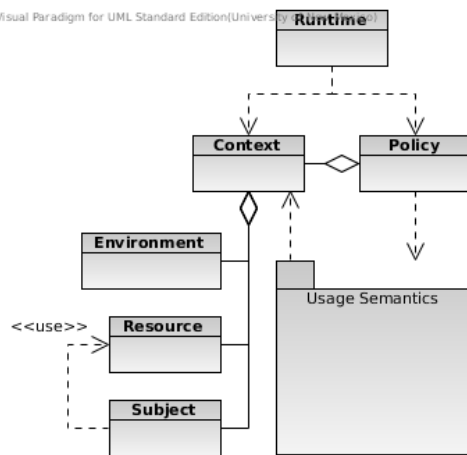


Figure 4: Basic Language Model

will allow or disallow artifact access. This is essentially an associative array, where the keys are specific expected properties of a given environment.

- *Resource*. A resource is the artifact over which the policy controls use. This can be any type of artifact whatsoever, ranging from documents to media files to streaming data. A resource may also have arbitrary properties like an associated URI, a canonical name, a MIME type, or creation metadata.
- *Subject*. Subjects use a given resource. Acceptable use is described by the policy.
- *Policy*. A policy describes the conditions of use for a given resource. In our example, this includes information on acceptable contexts and subjects, as well as obligations and constraints. Policies can be configured in this DSL to use arbitrary evaluators. This allows users to implement specific policy semantics tailored to their domain if needed, though they are free to use packaged syntax evaluators if those evaluators fit their needs.
- *Usage Semantics*. As policies can implement arbitrary semantics, they can be based on an specific models tailored to the needs for the particular policy system. For example, this DSL currently implements obligations and constraints restricting defined activities. Other domains may need to use more descriptive semantics, perhaps addressing causality or ordering.

Current usage semantics package various evaluators (specifically constraint and policy evaluators) and related entities as shown in Figure 5. Flexibility in policy evaluation is provided by pluggable *evaluators*. These evaluators essentially execute a given policy in tandem with a supplied context.

- *Evaluators.* Evaluators are pluggable components that provide flexibility in policy capability. We currently use two — one for evaluating policies, and one for

evaluating constraints. Policy designers can also specify multiple evaluators to enable combinations of semantics, as long as those semantics are not in conflict. Evaluators can use any kind of computational model or logic as well. For example, one evaluator may use a simple state machine model to evaluate a policy, while another may use a more complex turing-complete model.

- *Policy Evaluator*. This evaluator is responsible for evaluating the policy as a whole. To do so, it evaluates the relationships between permissions and obligations, and uses the constraint evaluator to examine restricted activities.
 - *Constraint Evaluator*. Constraint evaluators specifically examine constraints defined within restricted activities. They can do so using arbitrary evaluation rules ranging from simple conjunctive evaluation (where all constraints must be true) to more complex schemes where, for example, a certain number of constraints must be true while one of a set of other constraints must be false.
- *Obligation*. An obligation describes a restricted activity that must have occurred or must occur in the future for a restricted activity to be performed. For example, a media stream may wish to obligate users to purchase access to that stream on the third access.
 - *Permission*. In this context, a permission is a restricted activity that a subject can perform under the condition that certain specified obligations are met.
 - *Constraint*. A constraint generally constrains a restricted activity. This could be as simple as limiting use to a single identifiable subject or as complex as limiting use based on time and date, user identity, and geographic location.
 - *Activity*. A general activity is something a subject would wish to do in association with an artifact. It describes how a *subject* would use a *resource* in an unrestricted way, or some action that *subject* performs.
 - *Restricted Activity*. When an activity is embellished with constraints or obligation, it becomes restricted.

Now we have rigorously defined the domain elements our DSL will address. Keep in mind, this domain model allows us to dynamically replace ontology elements in that all *policy ontology* and *policy logic* elements can be replaced on per-policy basis. This would allow us to create multiple policies described using disparate ontologies and related evaluation logics if needed to more fully describe restrictions in a specific evaluation domain.

We have also separated the definition of *activities* from *restricted activities*. This separation of concerns allows policy developers to define a single activity which can then be reused across a large number of restricted activities based on specific varying constraints. For example, if I have a write activity, I can constrain that activity in slightly different ways to create a relatively large number of related restricted activities. I could restrict write by geographic area.

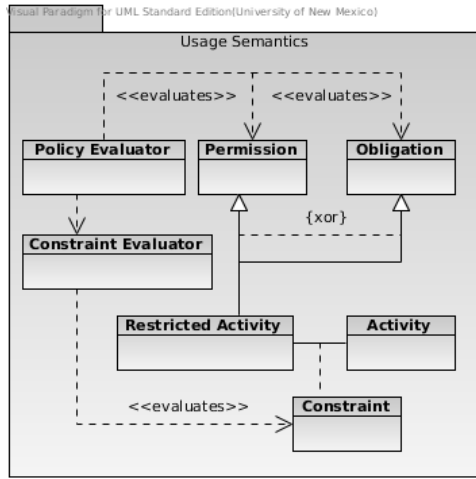


Figure 5: Current Usage Semantics

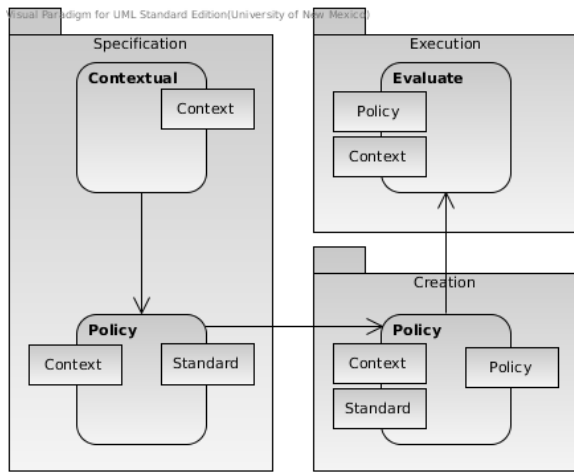


Figure 6: Policy Development Lifecycle

by subject identification, by date and time, or by having contributed to some political cause, creating four restricted activities from the same base activity.

The base domain model however, based on contextual elements, policies, and licenses does not change. This stability allows for ease of runtime integration as it hides any policy evaluation-specific changes. As long as a given logic and policy ontology is delivered with a given policy, the policy evaluation runtime will be able to evaluate that policy against resources and subjects in a given environment. In essence, this is a layered model where the context and policy elements compose the user interoperability layer while policies and policy logics form a more dynamic policy expression layer. This way, resource users have a stable runtime view of policies and policy managed resources, while policy and context developers have the flexibility to implement policies at arbitrary levels of expressibility.

3.3 Envisioned Lifecycle

3.4 Language Components

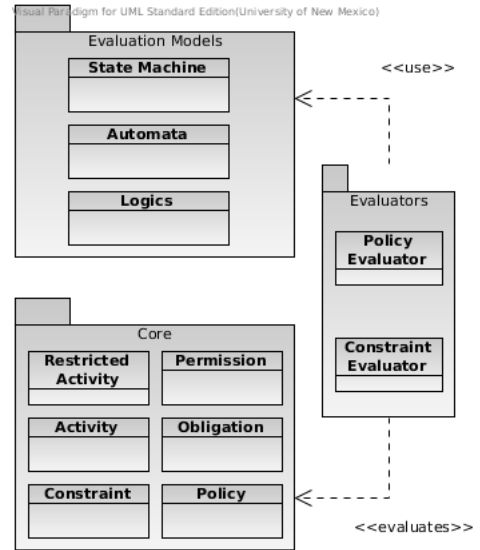


Figure 7: Language Elements

4. LANGUAGE

Language implementation

5. APPLIED

Application section

5.1 Creative Commons

Creative commons applied

5.2 Creative Commons

The Creative Commons is a nonprofit organization that works to increase the amount of creativity available in “the commons”. It provides free, easy-to-use legal tools that provide a simple, standardized way to pre-clear usage rights to creative work for copyright owners. It lets copyright holders easily change from default of “all rights reserved” to “some rights reserved”.

In this section, we use our DSL to implement a policy that can be used with Creative Commons licensed content, covering all seven of the main Creative Commons license types:

- *Public Domain.*
- *Attribution.*
- *Attribution, Share-Alike.*
- *Attribution, No-Derivatives.*
- *Attribution, Non-Commercial.*
- *Attribution, Non-Commercial, Share-Alike.*
- *Attribution, Non-Commercial, No-Derivatives.*

First, we create an activity. This activity represents sharing a Creative Commons licensed asset.

```
share = activity(:share) do
  true
end
```

Next, we define constraints on this share activity. There are four such constraints in Creative Commons: attribution, commercial use, derivative work, and share-alike.

```
attribution = constraint(:share) do
  true
end

non_commercial_use = constraint(:share) do
  true
end

non_derivative_work = constraint(:share) do
  true
end

share_alike = constraint(:share) do
  true
end
```

Finally, we define the restricted activities, covering all 7 basic license types.

```
# 1) Public Domain
share_pd_work = restrict share do
  # No constraints
end

# 2) Attribution
share_by_work = restrict share do
  with attribution_constraint
end

# 3) Attribution, Share-Alike
share_by_sa_work = restrict share do
  with attribution_constraint,
    share_alike_constraint
end

# 4) Attribution, No-Derivatives
share_by_nd_work = restrict share do
  with attribution_constraint,
    non_derivative_work_constraint
end

# 5) Attribution, Non-Commercial
share_by_nc_work = restrict share do
  with attribution_constraint,
    non_commercial_use_constraint
end

# 6) Attribution, Non-Commercial, Share-Alike
share_by_nc_sa_work = restrict share do
  with attribution_constraint,
    non_commercial_use_constraint,
    share_alike_constraint
end

# 7) Attribution, Non-Commercial, No-Derivatives
share_by_nc_nd_work = restrict share do
  with attribution_constraint,
    non_commercial_use_constraint,
    non_derivative_work_constraint
end
```

5.3 XRML

XRML applied

5.4 ODRL

ODRL applied

6. CONCLUSIONS AND FUTURE WORKS

Conclusion & future works

7. REFERENCES

- [1] Enabler release definition for DRM V2.0. Technical report, Open Mobile Alliance, 2003.
xml.coverpages.org/OMA-ERELD_DRM-V2_0_0-20040401-D.pdf.
- [2] Open digital rights language ODRL version 2 requirements. ODRL, Feb. 2005.
odrl.net/2.0/v2req.html.
- [3] Marlin architecture overview. Technical report, 2006.
<http://www.marlin-community.com>.
- [4] S. Ahmad-Reza and C. Stubble. Taming trusted platforms by operating system design. *Springer-Verlag*, pages 286–302, 2004.
- [5] H. Alverstrand. The role of the standards process in shaping the internet. *Proceeding of the IEEE*, 92(9):1371–1374, 2004.
- [6] A. Arnab and A. Hutchison. Persistent access control: A formal model for drm. In *DRM '07: Proceedings of the 2007 ACM workshop on Digital Rights Management*, pages 41–53, New York, NY, USA, 2007. ACM.
- [7] A. Barth and J. C. Mitchell. Managing digital rights using linear logic. In *LICS '06: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, pages 127–136, Washington, DC, USA, 2006. IEEE Computer Society.
- [8] D. E. Bell and L. J. L. Padula. Secure computer systems: Mathematical foundations, MTR-2547, Vol. I. Technical report, The MITRE Corporation, Bedford, MA, March 1, 1973. ESD-TR-73-278-I.
- [9] D. E. Bell and L. J. L. Padula. Secure computer system: Unified exposition and multics interpretation, MTR-2997, Rev. 1. Technical report, The MITRE Corporation, Bedford, MA, March 1976. ESD-TR-75-306.
- [10] M. S. Blumenthal and D. D. Clark. Rethinking the design of the Internet: The end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology*, 1(1):70–109, Aug. 2001.
- [11] A. J. Bonner. A prolog framework for reasoning about permissions and obligations, with applications to contract law. Technical report, Rutgers University, 1988.
- [12] A. Carroll, M. Juarez, J. Polk, and T. Leininger. Microsoft palladium: A business overview. Technical report, Microsoft Content Security Business Unit, June 2002.
- [13] C. N. Chong, R. Corin, S. Etalle, P. Hartel, W. Jonker, and Y. W. Law. LicenseScript: A novel digital rights language and its semantics. In *Third International Conference on the Web Delivery of Music*, pages 122–129, Los Alamitos, CA, Sept. 2003.
- [14] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: Defining tomorrow's internet. In *SIGCOMM*, pages 347–356, Pittsburgh, Pennsylvania, USA, Aug. 2002.
- [15] T. Clark. IBM closes unit, 2002.
<http://news.com.com/2100-1001-206465.html?legacy=cnet>.
- [16] Coral consortium whitepaper. Technical report, Feb.

2006.
www.coral-interop.org/main/news/Coral.whitepaper.pdf.
- [17] C. Gunter, S. Weeks, and A. Wright. Models and languages for digital rights. In *HICSS '01: Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 9*, page 9076, Washington, DC, USA, 2001. IEEE Computer Society.
- [18] J. Y. Halpern and V. Weissman. A formal foundation for XrML licenses. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, pages 251–265, Asilomar, CA, June 2004.
- [19] J. Y. Halpern and V. Weissman. A formal foundation for XrML. *J. ACM*, 55(1):1–42, 2008.
- [20] G. L. Heileman and P. A. Jamkhedkar. DRM interoperability analysis from the perspective of a layered framework. In *Proceedings of the Fifth ACM Workshop on Digital Rights Management*, pages 17–26, Alexandria, VA, Nov. 2005.
- [21] G. L. Heileman, P. A. Jamkhedkar, J. Khoury, and C. J. Hrnecir. DRM game. In *Proceedings of the Seventh ACM Workshop on Digital Rights Management*, pages 54–62, Alexandria, VA, Oct. 2007.
- [22] V. Hu, D. Ferraiolo, and R. Kuhn. Assessment of Access Control Systems. Technical report, National Institute of Standards and Technology, Sep. 2006.
csrc.nist.gov/publications/nistir/7316/NISTIR-7316.pdf.
- [23] H. Hundacker, D. Pahler, and R. Grimm. URM - usage rights management. In *Proceedings of the 7th International Workshop for Technical, Economic and Legal Aspects of Business Models for Virtual Goods*, pages 125–138, Nancy, France, Sep. 2009.
- [24] R. Iannella. Open digital rights language (ODRL), Version 0.5, Aug. 2000. odrl.net/ODRL-05.pdf.
- [25] P. A. Jamkhedkar and G. L. Heileman. DRM as a layered system. In *Proceedings of the Fourth ACM Workshop on Digital Rights Management*, pages 11–21, Washington, DC, Oct. 2004.
- [26] P. A. Jamkhedkar and G. L. Heileman. *Handbook of Research on Secure Multimedia Distribution*, chapter Rights Expression Languages. IGI Publishing, 2008.
- [27] P. A. Jamkhedkar, G. L. Heileman, and I. Martinez-Ortiz. The problem with rights expression languages. In *Proceedings of the Sixth ACM Workshop on Digital Rights Management*, pages 59–67, Alexandria, VA, Nov. 2006.
- [28] R. H. Koenen, J. Lacy, M. MacKay, and S. Mitchell. The long march to interoperable digital rights management. *Proceedings of the IEEE*, 92(6):883–897, 2004.
- [29] U. Kohl, J. Lotspiech, and M. A. Kaplan. Safeguarding digital library contents and users: Protecting documents rather than channels. *D-Lib Magazine*, 3(9), Sept. 1997.
- [30] L. T. McCarty. A language for legal discourse I. basic features. In *ICAIL '89: Proceedings of the 2nd international conference on Artificial intelligence and law*, pages 180–189, New York, NY, USA, 1989. ACM.
- [31] D. L. McGuinness. Reasoning with permissions and obligations in contract law. Technical report, Rutgers University, 1986.
- [32] J. Park and R. Sandhu. The $UCON_{ABC}$ usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, 2004.
- [33] J. Polo, J. Prados, and J. Delgado. Interoperability between ODRL and MPEG-21 REL. In *Proceedings of the first international ODRL workshop*, Vienna, Austria, Apr. 2004.
- [34] R. Pucella and V. Weissman. A logic for reasoning about digital rights. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 282–294, Nova Scotia, Canada, June 2002.
- [35] R. Pucella and V. Weissman. A formal foundation for ODRL, Jan. 2006. arxiv.org/abs/cs/0601085.
- [36] R. Safavi-Naini, N. P. Sheppard, and T. Uehara. Import/export in digital rights management. In *Proceedings of the Fourth ACM Workshop on Digital Rights Management*, pages 99–110, Washington, DC, Oct. 2004.
- [37] A. U. Schmidt, O. Tafreschi, and R. Wolf. Interoperability challenges for DRM systems. In *IFIP/GI Workshop on Virtual Goods*, Ilmenau, Germany, 2004.
<http://virtualgoods.tu-ilmenau.de/2004/program.html>.
- [38] X. Wang. MPEG-21 rights expression language: Enabling interoperable digital rights management. *IEEE Multimedia*, 11(4):84–87, Oct./Dec. 2004.
- [39] J. Xiang, D. Bjorner, and K. Futatsugi. Formal digital license language with OTS/CafeOBJ method. In *Proceedings of the sixth ACS/IEEE International Conference on Computer Systems and Applications*, Doha, Qatar, Apr. 2008.
- [40] eXtensible Rights Markup Language (XrML) 2.0 Specification, November 2001. www.xrml.org.
- [41] XrML 2.0 technical overview, version 1.0, March 2002. www.xrml.org/reference/XrMLTechnicalOverviewV1.pdf.
- [42] Z. Zhang, Q. Pei, J. Ma, L. Yang, and K. Fan. Cooperative and non-cooperative game-theoretic analyses of adoptions of security policies for DRM. In *Proceedings of the 6th Annual IEEE Consumer Communications and Networking Conference*, pages 1–5, Las Vegas, NV, Jan. 2009.