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

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

DRM, usage management, software architecture

1. INTRODUCTION

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse varius mi et dui consequat id faucibus massa elementum. Nunc iaculis magna nec lectus feugiat vulputate. Sed eu pretium nisl. Sed ipsum urna, vulputate pharetra tincidunt in, aliquet in magna. Nam faucibus suscipit nibh a eleifend. Nulla nisi nunc, vulputate eget sollicitudin ac, interdum quis tortor. Aenean fermentum mollis dui, vel bibendum lorem placerat non. Suspendisse eget sollicitudin orci. Pellentesque metus erat, sagittis eu laoreet sed, ultricies molestie leo. Donec pellentesque massa sed nisl feugiat tempor. Suspendisse convallis purus ac mi posuere varius. Fusce at enim id metus mattis aliquet. Donec elementum adipiscing purus quis egestas.

Quisque euismod pulvinar diam, ut porttitor turpis pharetra non. Proin eu tincidunt lectus. Aenean quis nibh vitae nulla cursus venenatis. Aliquam non dolor eu erat elementum sagittis. Vestibulum viverra tristique eros sit amet elementum. Pellentesque at tellus lorem, vitae convallis purus. Morbi at rhoncus dolor. Integer luctus elit sed elit auctor aliquam. Nulla tincidunt neque in augue adipiscing et consequat dolor fringilla. Cras vulputate dignissim lectus vel consectetur. Ut lorem enim, pulvinar commodo hendrerit vitae, dapibus sit amet turpis. Duis bibendum, turpis et porttitor molestie, magna dolor imperdiet quam, ut scelerisque ante libero sed sem.

Vivamus dictum pellentesque metus quis convallis. Quisque consequat, lacus non hendrerit posuere, nulla risus posuere purus, sit amet vestibulum urna tortor vel erat. Curabitur malesuada sodales diam quis ullamcorper. Aenean lectus lacus, volutpat eu mollis id, semper volutpat leo. Cras turpis neque, rutrum in tempor vitae, egestas ut diam. Nam luctus sagittis euismod. Fusce et lorem tortor, vel bibendum purus. Duis egestas, velit ac faucibus fringilla, magna lectus adipiscing ipsum, vitae porta metus nisi id metus. Morbi eu erat augue. Suspendisse potenti. Maecenas bibendum ultrices urna vitae porta. Aliquam feugiat neque at elit facilisis non aliquet mi adipiscing. Phasellus non lacus quis eros viverra rutrum. Phasellus suscipit suscipit dapibus. Cras quis malesuada augue. Proin hendrerit cursus lectus.

Maecenas vestibulum felis in elit interdum venenatis. Fusce consequat rhoncus justo ut blandit. Pellentesque viverra tellus eget enim cursus quis imperdiet urna consequat. Morbi nec metus sit amet arcu lacinia tempor non eu tortor. Cras ut massa sed eros porttitor aliquam. Integer justo nisi, scelerisque quis fermentum a, interdum id dui. Aliquam sed arcu eget velit malesuada rutrum. Nullam viverra lectus id tellus scelerisque ac varius metus pellentesque. Nullam leo leo, aliquam id imperdiet a, pellentesque scelerisque ipsum. Phasellus mi velit, ullamcorper ut pellentesque ut, euismod in nisi. Curabitur tincidunt, sapien condimentum congue commodo, lectus ipsum fermentum nunc, tincidunt porttitor mauris augue non massa. Aenean sollicitudin consequat arcu vel congue.

Vivamus vestibulum pulvinar quam nec consectetur. Phasellus bibendum vulputate iaculis. Nullam dictum, mauris a adipiscing porttitor, massa purus mattis magna, id porta erat nibh sed turpis. Morbi laoreet, diam ac iaculis venenatis, libero risus dignissim orci, sit amet consequat diam nisi a ante. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin feugiat ullamcorper nulla sed pellentesque. Nulla facilisi. In varius tincidunt velit, sagittis suscipit dolor pulvinar a. Pellentesque quis lorem turpis, placerat dictum orci. Pellentesque at mattis eros. Nunc luctus, elit id fermentum viverra, leo lorem mattis nisi, et faucibus magna augue dapibus ipsum. Integer pharetra felis non dui fermentum sagittis. Suspendisse sit amet justo sapien, eu aliquet nisl.

1.1 Previous Work

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur vitae dolor elit, vel sagittis justo. Nullam vehicula scelerisque fermentum. Quisque pulvinar, neque sit amet tempor sagittis, risus felis consequat massa, at euismod quam lacus sed sapien. Integer nec viverra mi. Mauris ultricies tellus non nisl tincidunt dictum. Pellentesque pretium lectus consectetur mauris tempor rutrum. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis dictum quam tellus, eu scelerisque elit. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nullam massa leo, commodo id suscipit eu, consectetur at quam. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam eu diam leo, in rutrum justo. Phasellus ut turpis at orci tempor varius. Suspendisse iaculis faucibus bibendum. Etiam eget mollis nunc. Suspendisse potenti. Pellentesque lectus leo, ornare a ultrices a, lacinia in diam. Vivamus eu justo at lacus sodales eleifend consequat id dui. Nullam laoreet rhoncus malesuada. Curabitur sit amet cursus diam.

In hac habitasse platea dictumst. Suspendisse potenti. Nunc eros libero, eleifend eu scelerisque sit amet, varius rutrum felis. Nulla facilisi. Phasellus tincidunt sapien a libero blandit blandit. Fusce rutrum aliquet lacus, non rhoncus risus facilisis at. Mauris semper varius quam et placerat. Fusce sed suscipit mi. Vivamus a ligula ante, in adipiscing velit. Nullam auctor molestie tincidunt. Duis blandit diam et ante congue vitae laoreet elit sodales. Proin eu nulla vitae est tincidunt rhoncus. Duis at ultrices odio.

2. LANGUAGE MODEL

In developing this DSL, we needed to have a clear understanding of the specific domain, and develop and 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 1, 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.

Figure 1: Conceptual Model

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.

2.1 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 that 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

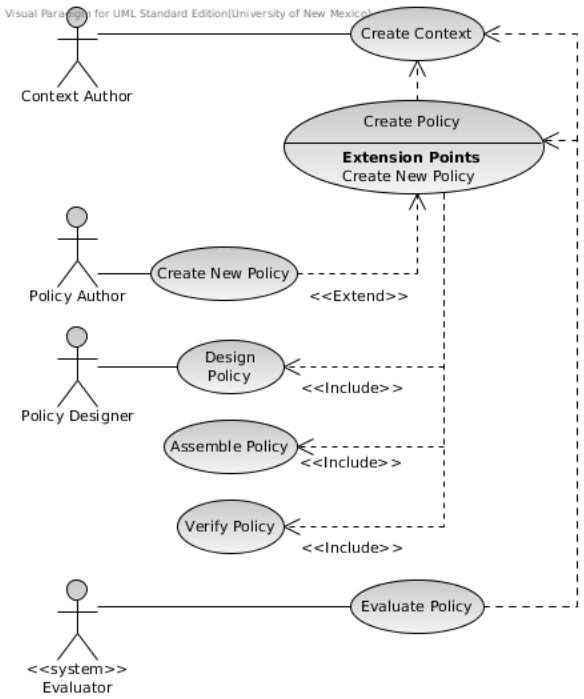

Figure 2: General DSL Use Cases

at the expense of contextual elements if necessary.

Figure 2 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.
- *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.

2.2 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 3. 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 han-

dling 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 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 4. 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 evalutators 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.

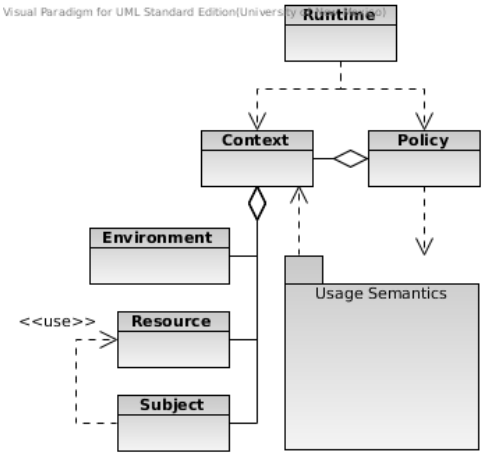

Figure 3: Basic Language Model

- *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

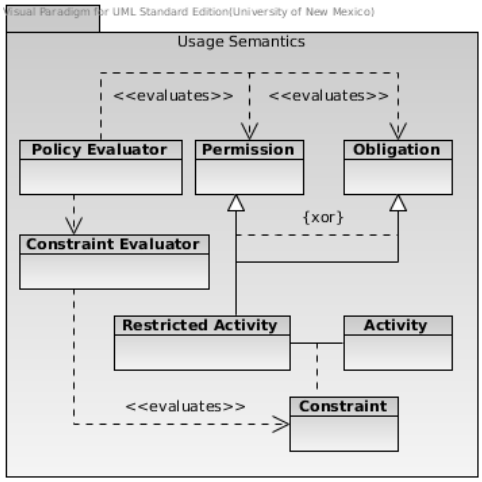

Figure 4: Current Usage Semantics

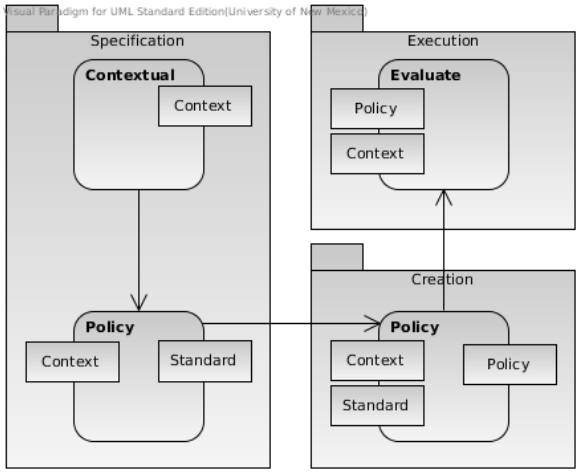

Figure 5: Policy Development Lifecycle

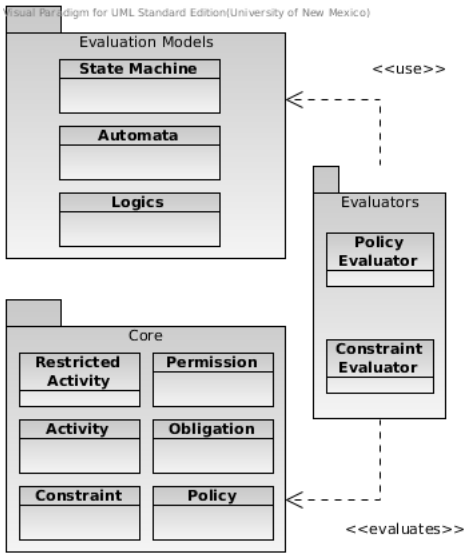

Figure 6: Language Elements

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, 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 essense, 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.

2.3 Envisioned Lifecycle

2.4 Language Components

3. LANGUAGE

Language implementation

4. APPLIED

Application section

4.1 Creative Commons

Creative commons applied

4.2 XRML

XRML applied

4.3 ODRL

ODRL applied

5. CONCLUSIONS AND FUTURE WORKS

Conclusion & future works