Managing with XACML

Introduction

A broad range of languages have been defined for expressing policy such that it can be enforced by a machine. In most cases, these languages are tailored to a particular application domain, such as network management, access-control, privacy or digital rights. Few attempts have been made to define a language that can serve more than one domain.

The first major division in the taxonomy of policy languages is that between authorization and management policies. Authorization policy specifies the circumstances under which an action may be allowed, whereas management policy specifies the actions that should be taken when specified circumstances arise.

Ponder is the only language known to the author that was designed to address the requirements of both of these policy categories.

XACML was defined primarily for expressing authorization policy. But, in attempting to deal with the side-effects of authorization, its designers gave XACML some of the features necessary for expressing management policy.

In this paper, we attempt to identify deficiencies in the current definition of XACML in relation to its use as a language for expressing management policy.

Management policy

Management policy statements commonly contain three main clauses: ***event***, ***condition*** and ***action***. The semantics of such statements are that, when the specified ***event*** occurs and if the specified ***condition*** holds, then the specified ***action*** should be taken. Management policies find application in the fields of network (and other resource) management and work-flow applications (amongst others).

Management policy in XACML

The proposed way of mapping the constructs of management policy to XACML constructs is shown in Table 1.

Management policy	XACML construct
Event	Target
Condition	Condition
Action	Obligation

Table 1 - Correspondence between management policy and XACML constructs

XACML Constructs

Primary constructs

Target

An XACML "Target" is a declarative-style construct for expressing a simple predicate over attributes of the subject, resource, action and environment. It was designed to express part of the authorization post-condition and to allow efficient indexing of policy statements. However, it is readily capable of expressing an ***event*** definition, such as a connection made to port 23:

> Resource equals "port 23" and Action equals "connect"

Condition

An XACML "Condition" is a declarative-style construct for expressing an arbitrarily complex predicate over attributes of the subject, resource, action and environment. It was designed to express the authorization pre-condition. However, it is readily capable of expressing a management policy ***condition***.

Obligation

An XACML "Obligation" is an imperative-style expression, consisting of an instruction with parameters. Obligations represent the second part of the authorization post-condition. The instructions represented by the obligations are to be executed independently. There is no implied sequence or interdependence between the individual instructions.

Obligations may express the ***action*** part of a management policy.

Secondary constructs

Effect

The XACML "Effect" is the result of evaluating a condition. Ignoring fault conditions, it is a Boolean with a value of either "Permit" or "Deny".

The XACML Effect has a secondary function: it triggers obligations. Each obligation is tagged with the Effect value for which it must be executed. When used in authorization policy, the Effect can cause different instructions to be executed depending upon the authorization decision.

When used in management policy, Effect can identify management ***actions*** associated with particular ***conditions***.

Combining algorithm

An XACML "combining algorithm" operates on the individual Effect values in a set of conditions to arrive at an overall Effect for the set. For all the combining algorithms

defined by XACML, the overall Effect may be determined without evaluating the complete set of conditions, and XACML allows this partial evaluation. For instance, the "deny-overrides" combining algorithm can terminate as soon as it encounters the first true condition whose Effect value is "deny", thereby leaving any remaining conditions unevaluated, even though one or more of those conditions may hold.

The impact of this is that unevaluated conditions may contain obligations that don't get triggered, because their Effect values were not calculated.

XACML enhancements

The primary output of a management policy is the set of management ***actions***, expressed in XACML, using obligations. Effect, which is the primary output of an authorization policy, serves merely to trigger obligations. Expressing management policy requires combining algorithms that evaluate all conditions without terminating prematurely. This would not require a change to core XACML, only the definition of a new combining algorithm.

XACML defines Effect to be a Boolean with values "Permit" and "Deny". These values are entirely appropriate for authorization policy, but are not appropriate for management policy. Values such as Triggered and NotTriggered would be more suitable. Such a change would be entirely cosmetic.

Procedural constructs, such as "sequence" and "choice" for sets of obligations would be a helpful addition.

Conclusions

XACML has many of the features required to express management policy. Some trivial extensions are required, such as the definition of a new combining algorithm. Some cosmetic changes are also desirable, such as the ability to assign new values to the Effect variable. More significantly, it is necessary to be able to express sequences and choices of obligations.