Constraint language data types and operations.

This write-up summarizes my suggestions on various aspects of the data type usage and constraint logical functions in XACML. Initially, I prepared a single schema, but realized that it contains certain elements, in particular aimed at support of policy management and analysis, that are outside the scope of the XACML and may be hard to integrate with the rest of the current proposal. It would have made it hard to vote on other features, that I consider more important to resolve. In this proposal I go over the main points that may be discussed separately. First, I list my assumptions, then a proposed implementation.

Assumptions.

I start from the ideas that were discussed at L.A. F2F meeting. My suggestions are highlighted.

PDP implementation will need to satisfy following criteria, which imply certain requirement on the data type and operations usage:

1) Reliability and security of policy evaluation
    a. **All data used in authorization decision evaluation should be strongly typed and explicitly declared**. This avoids ambiguity in data format interpretation. It also means that data type should not be determined at the runtime, implicitly from the data itself, as it is unsafe.
    b. **All data types used in a particular policy should be declared as part of the policy**. I assume that most implementations will have to declare and use custom data types and conversions. Even within the same implementation, policy exchanged between PDPs need to keep track and enforce all data formats used.
    c. **All permissible data type conversions should be listed as part of the policy**. By "listed as part of the policy" I mean - included into the declarative section of the policy file, in a form of XML formatted statements. PDP should provide all the declared data types and conversion algorithms to be able to enforce the policy.
    d. **All functions/data operations should have unambiguous argument and return type, and be declared, including versioning information, as part of the policy data.** I consider it important that all elements of the policy that affect the outcome of each authorization decision (for example, version of a particular custom data function used) are available to the PDP, in a readable and auditable format, not as a separate document or standard.
2) High performance
    a. **All data types should be resolved, and appropriate conversions assigned before the actual constraint evaluation time**. That

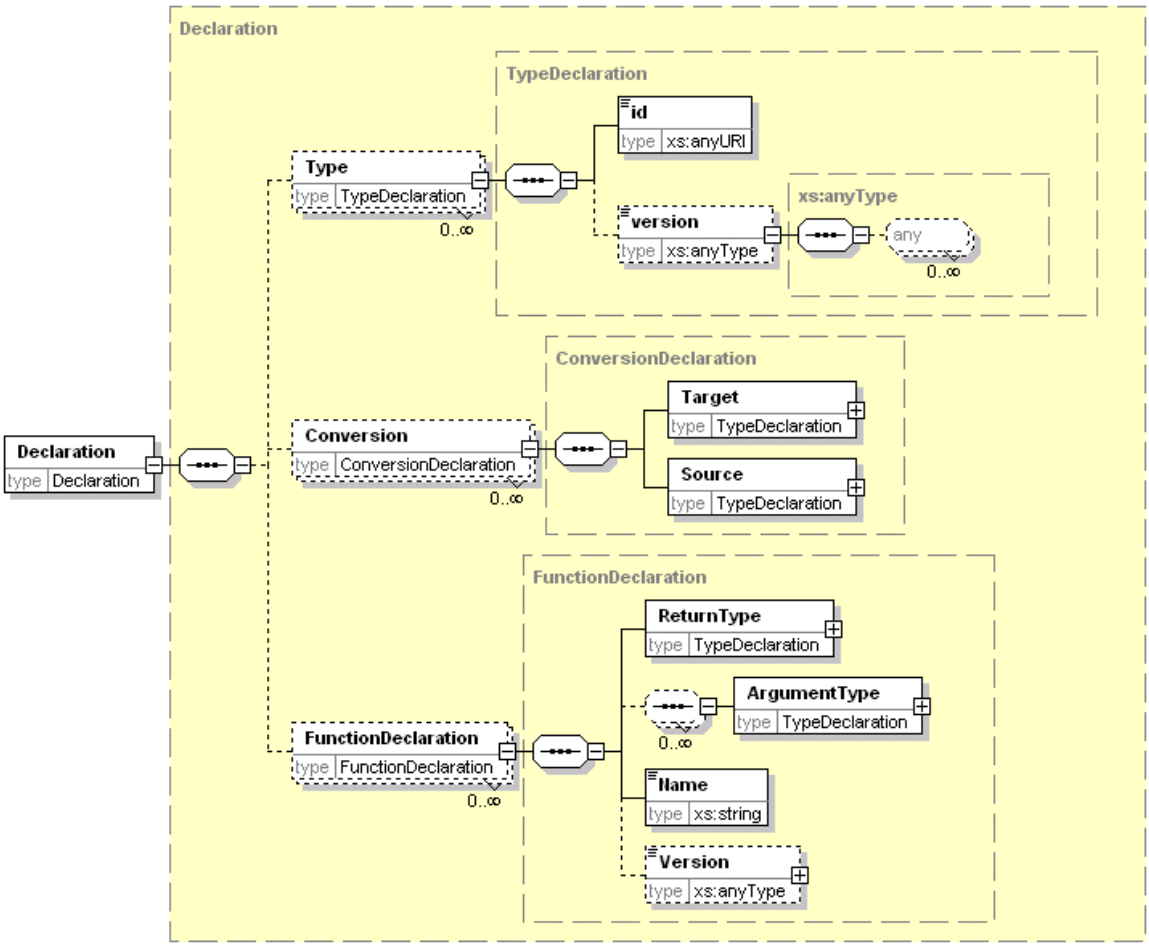
implies that for all function calls, argument types and conversions should be determined in the "compile", or policy import time. This is also a requirement related to the above-mentioned reasons.

3) Ease of interoperability
   a. Policy data should include all necessary information needed for PDP to determine if it can effectively enforce the policy. I propose having policy data as **two separate files**, both conforming to XACML schema: declaration file and policy data file. Declaration file contains data type, data type conversion and data operations (functions) and constants declarations. Core XACML standard provide a mandatory declaration file, partially described below. It may be extended in a particular policy.
4) Ease of implementation
   a. **Functions should be the only element of the constraint**. Top most function(argument) should return a type that can be converted to xacml:boolean type. There is no need for any other distinct constructs, such as predicates.
   b. **Policy consistency and data typing enforcement is handled by PDP implementation, not XML schema mechanism**. PDP has to validate policy consistency anyway, based on the policy data – ensuring that all data types, functions are supported and constraints are properly formed and can be evaluated. Schema can only provide limited support for verification. It will be a waste of effort to try to predict all data type combinations that may be used.
   c. Declaration of functions and types is provided as an **XML formatted document**, not schema extension. It is hard to argue the ease of implementation issue. I wrote a mock up of the PDP evaluation code having to perform following functions: read declaration of XACML standard and several custom functions and type conversions, validate that they are supported, parse a constraint expression and assign runtime evaluation code for fast data conversion and processing. Dealing with objects generated by commonly used XML processing libraries seems to produce by far more compact and maintainable code. It is also make more logical sense, given the above-mentioned suggestions. Of course – logically, same concepts may be expressed both ways.
   d. **Functions with undefined number of arguments are not supported**. If such functionality is necessary, it can be implemented using array or list data types, as arguments. All other cases can be covered by function nesting.

Declaration schema.

I would ask XML experts to advise on the preferable way to present this data structure. In particular – Type declarations should be probably made global elements, referenced in

conversion and Function declaration, so that only defined types can be used there.. I have not found the best way to express that. Exact way to present it is not important – the core idea is that all elements of the constraints (predicates, core and custom functions) all share the same simple syntax, and are declared, as part of the policy data. Here is a draft, representing essential elements:



Declaration – root element
TypeDeclaration – id is URI of a type that must be supported. Contains optional "version" attribute, which must be understood by PDP, if present.
Conversion – contains a pair of references to TypeDecloration elements describing permissible type conversion, that must be supported.
ArgumentTypeDeclaration and **ReturnTypeDeclaration** – contain reference to TypeDeclarations, used in FunctionDeclaration.
FunctionDeclaration – includes ReturnType declaration and a sequence of ArgumentType declaration. Includes attributes – unique "name" and optional "version".

XACML core data types and operations.

Core data types and functions can be defined using the aforementioned schema.

I propose to define XACML own data types and define core functions in terms of these data types. All XML schema data types should be convertible to this data types. Implementation of this data types is not specified – they should be just broad enough. This allows an implementation to choose a data type that covers some other implementations (for example GPS time for time..). Syntax for arrays may be left to the implementation or defined in the standard (? – how).

Types:

xacml:boolean
xacml:integer
xacml:float
xacml:date
xacml:time
xacml:string
xacml:list (list of strings)
xacml:integerArray
xacml:floatArray

and all XML schema data types..

Integer should be declared convertible to float, but conversion of float to integer done, using one of the standard functions (below). Duration and other arithmetic operations on time should be probably omitted from the first version of the standard.

All appropriate conversions from XML data types to XACML data type should be defined. It should be suggested that all policies use XML schema data type wherever possible.

Any data type should be convertible to and from string (conversion from string may result in runtime error).

Empty list or array converts to boolean value "false".

Functions declared (data types are the xacml types above)

boolean OR(boolean, boolean)
boolean AND(boolean, boolean)
boolean NOT (boolean)

integer FLOOR (float)

boolean EQUAL (X, X) - for all data types, including list and array, where equality should be defined as set equality (independent of order)

integer PLUS, MINUS, DIVIDE, TIMES, MOD (integer, integer) – and for float

boolean GT, LT, GEQ, LEQ, NEQ (X, X) for all appropriate data types

list UNION (list), list INTERSECT (list) – same for integer and float arrays.

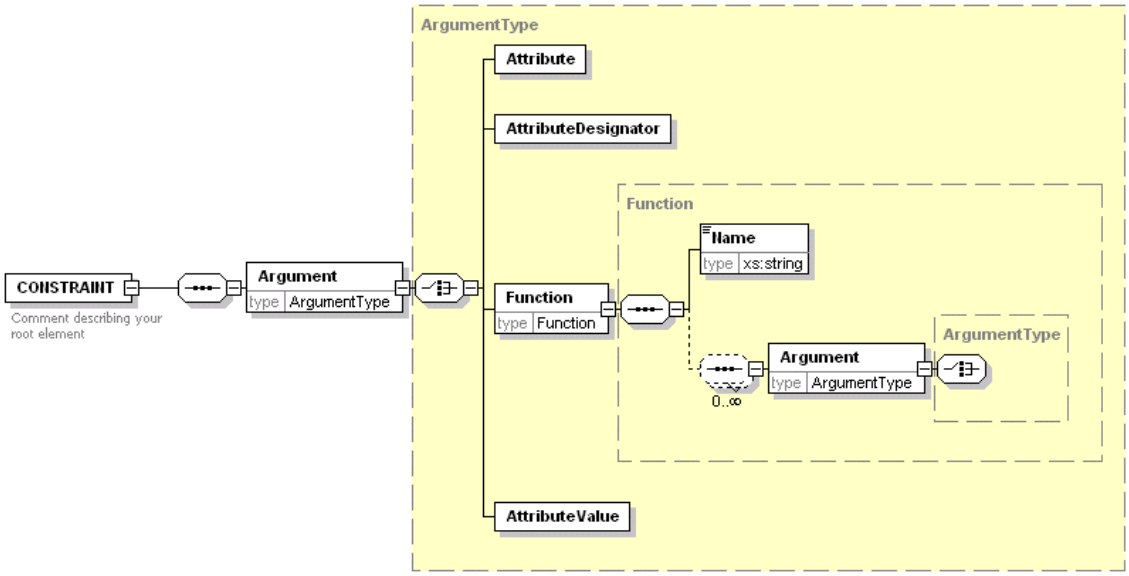
boolean SUBSET (list, list) – same for integer and float arrays.

boolean IN, NOTIN (string, list) – same for integer, integerArray and float and floatArray…

integer MAX, MIN (integerArray) – same for float

Constraint element

The rule element in the policy definition should be expressed in the following way. The only limitation, enforced by implementation – Top most argument should be of data type that was declared to be convertible to xacml:boolean.



Resolving function name, and enforcing proper types of arguments is left for the implementation.
I assume that Attribute, AttributeDesignator does include proper URI of a declared data type and AttributeValue is interpreted as string – any data type can be converted to and from string.