Securing Workflows with XACML, RDF and BPEL

Vijayant Dhankhar[1], Saket Kaushik[2], and Duminda Wijesekera[3]

[1]Department of Computer Science, George Mason University, Fairfax, VA 22030, U.S.A, dhankhar@gmail.com
[2]Oracle Corporation, Redwood City, CA 94065, U.S.A, saket.kaushik@oracle.com
[3]Department of Computer Science, George Mason University, Fairfax, VA 22030, U.S.A, dwijesek@gmu.edu

No Institute Given

Abstract. The XACML is the access controller of the World Wide Web (WWW). The current reference implementation has a single policy decision point and a policy enforcement point. If XACML policies are used to control workflow among cooperating web services, such as those envisioned in more contemporary languages like (BPEL), it requires coordination to be policy compliant. We propose the necessary enhancements required to do so by passing *contextual information* that are needed for the requester to evaluate an access control decision as opposed to the standard four decision values of *permit*, *deny*, *indeterminate to make a decision* and *an unforeseeable error occurred during evaluation*. Proposed contextual information is sufficient to coordinate and if necessary synchronize among coordinating policy enforcement points distributed among the WWW. We show how the contextual information can be constructed and verified using the Resource Description Framework (RDF) and the coordination implemented using BPEL.

1 Introduction

In service-oriented architectures, workflows are increasingly being used to provide a single point of access for composite services constructed from multiple sub services. In order to provide a single authority to make yes/no decisions for workflow requests, the individual mechanisms that control the components involved in authorizing the flow should cooperate, requiring distributed evaluation *and* enforcement of the access control decision. Because XACML is the access control language for web services [21], there is a need for distributed accesses controllers using XACML to coordinate in providing secure flow control.

In XACML parlance, a policy based access control decision will be evaluated by (possibly) several so called *policy decision points (PDPs)* [21] collectively; as well as enforced collectively at (possibly) several *policy enforcement points (PEPs)* [21]. This approach has additional advantages, most prominently for the requester, the single service access point provides a service specified by a single access control policy. If implemented, a single access control policy retained at the mother service and be evaluated and enforced distributively by the sub-services. However, current XACML standard and reference implementation [26] lack the desired syntax and enforcement mechanisms for such an access controller. In this paper we supply the necessary extensions to

current XACML standard and reference implementation to be able to evaluate and enforce XACML policies in a *fully-distributed* manner. We enhance our previous [11, 12] work in achieving this objective in this paper by passing sufficient contextual information between XCAML PDPs and PEPs so that the passed information contain sufficient information to control the distrbuted usage of resources such as flow control ans synhronization. Such techniques can be used in more contemporary workflow languages like BPEL.

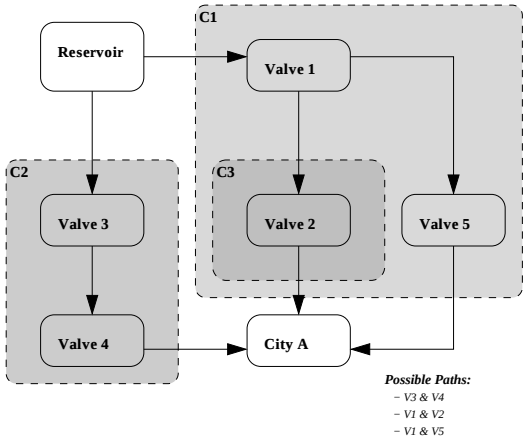

Fig. 1: *Use Case*

For example, consider a hypothesized process of transferring water from a *Reservoir* to a *City A*, enabled through Web Service invocations as shown in Figure 1. In this scenario, for water transfer to take place, the requestor must possess the required authorization *and* the action of opening a valve must be approved by the three companies (*C1*, *C2* and *C3*) that maintain the grid of water pipelines that connect the reservoir to the city A. In addition, the access controller must check if another request to transfer water is underway or not. This is a necessary environmental constraint because water pipelines have a safety limit as to the amount of pressure they can withstand. As long as there exists a continuous channel from *Reservoir* to *City A*, the water pressure is within safe limits and requestor is authorized, this transfer should be allowed.

Also once the transfer is allowed, it should be run transactionally, i.e. if the downstream valve (*Valve 2* - policy enforcement point) fails to open due to unforseen failure then the upstream valve (*Valve 1* - policy enforcement point) should close, otherwise there is a possibility for an intervening pipeline to burst – another environmental constraint to be met *during* the transfer.

We propose that coordinating access controllers share more than *permit* or *deny* decisions in requesting access to resources controlled by sub-services. Our proposal, described in detail, proposes that they enhance the *permit* or*deny* decision by providing some *context* in which the requester can evaluate the decision. Proposed context infor-

mation provides the requirements that must be ensured prior to starting the resource usage, the conditions that must be satisfied during the usage and those that must be specified when the requester relinquishes the usage, thereby providing a context for full *usage control* [27].

We propose using the *Resource Description Framework (RDF)* [6] for this purpose of describing the *context*, because RDF can be used to specify a complete *ontology* for the resource usage, including but not limited to exclusivity (i.e. such as writing privileges). We show a snippet of our implementation.

Lastly, the distributed policy enforcement points need to address the control dependencies that exists between policy enforcement points that must exist to ensure the flow. We show how this can be done using the *Business Process Execution Language (BPEL)* [19]. As of this writing we have a preliminary implementation of this.

The significance of this proposal are three fold: Firstly, it can convey synchronization requirements beyond exclusive usage and options. Secondly, it can be customized to the operational interfaces provided by the resource, and therefore go beyond the traditional *read, write, execute* permissions. Lastly, we show how the current collection of semantic web languages and their runtimes (namely, XACML, RDF and BPEL) can be used to implement our proposal. Parts of this proposal have been implemented, and our ongoing work concentrates in making our rudimentary implementation more generic.

The rest of the paper is written as follows. Section 2 describes the current XACML reference implementation. Section 3 provides an overview of RDF. Section 4 describes the enhancements needed to fully distribute the XACML implementation. Section 5 describes some architectural enhancements we are proposing to the existing XACML reference implementation in order to achieve full distribution. Section 6 describes related work and Section 7 has our concluding comments.

2 The XACML Reference Implementation

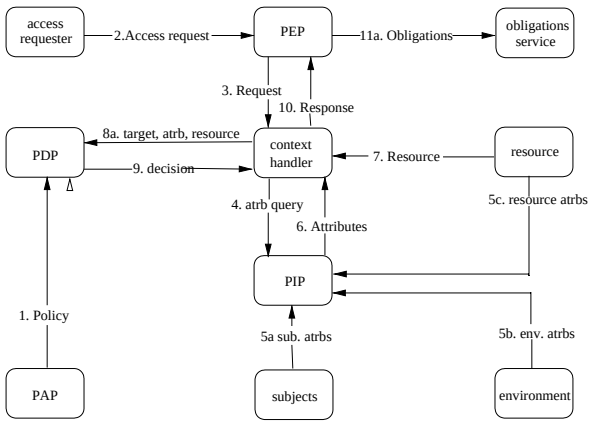

Fig. 2: *XACML Architecture*

The current XACML specification has three main entities as shown in Figure 2. As shown, it has the following main components in addition to auxiliary components.

1. **Policy Administration Point (PAP):** Entity that creates policies or policy sets.
2. **Policy Decision Points (PDP):** Entity that evaluates applicable policy and renders an authorization decision. The answer given by the PDP is one of (1) *permit*, (2) *deny*, (3) *insufficient information to decide* or (4) *error*, implying some unforseen error occurred in the execution.
3. **Policy Enforcement Point (PEP):** Entity that performs access control by enforcing authorization decisions.

Figure 2 shows the dataflow of the XACML reference implementation. First, the PAP creates a policy. At request time, an access request arrives at the PEP (flow 1), and is sent to the context handler (flow 2). The context handler determines resources to be accessed and attributes of the requester, resource and the environment, collects all required attributes and forwards them to the PDP (flows 3,4,5,6,7,8). PDP then acquires the policy from PAP (flow 1), evaluates the relevant policy and relays the decision (flows 9, 10) to the PEP through the context handler, which proceeds to enforce the authorization decision.

The policy syntax (XML) includes language constructs to identify the *resource*, the *action* (to be performed on the resource), the *subject*, and *constraints* on the access. In XACML parlance, this collection of entities is called a *target*. The request syntax (XML) identifies the resource, the action, the subject. The decision engine (PDP) *matches* the incoming request to available policies to discover all applicable policies. If more than one policy is applicable, then the PDP uses a *policy-combination algorithm* [21] to determine the evaluation result. In essence, the combination algorithm states how to combine the result of each applicable policy.

3 RDF Overview

RDF [7, 6, 15] specify meta-information about *resources, i.e.*, entities that can be uniquely identified, and *binary relations* between them so that they can be "machine processed". Such meta information about resources are specified in RDF using binary properties between resources. RDF does so by using the syntax of *triples* where the subject (the first component of the triple) is related by the property (the second component of the triple) to the object (the third component). An RDF schema can be extended further by specifying binary properties between nodes and triples. This process, carried out recursively, is referred to as *reification*. RDF(S) or RDF Schema is RDFs vocabulary description language. It has syntax to describe concepts and resources through meta-classes such as `rdfs:Class`, `rdf:type`, *etc.*, and relationships between resources through `rdf:property`. These meta classes are used to specify properties of user defined schema. Details of RDF/RDF(S) syntax and vocabulary descriptions can be found in [6]. This syntax is readily usable in XACML framework because of the inherent ability of RDF to capture *attribute-value* pairs in its syntax. Attributes are *named-properties* of nodes and their values can be (atomic) data (text, string, integer, etc.) or other nodes.

Multiple RDF triples form a graph (connected or disconnected), *i.e.*, if the object of a triple is the subject of another triple, then the two triples are merged together retaining

the common object only once (with one incoming edge and one outgoing edge) [16]. Since RDF expresses binary relationships, RDF triples and graphs can be interpreted by machine languages and *queried* using languages like RDQL, SPARQL, *etc.* [8, 18, 25, 24, 9, 22] (on the lines of SQL). In addition, due to an XML-based syntax, *XSLT Rules* can also be specified to query and interpret RDF data. In this work, we make use of these provisions to effectively utilize RDF-based information for enforcing distributed access control decisions.

4 Requirements for Distributing XACML

As described in Section 2, the PDP issues one of four decisions of permit, deny, inapplicable policy or an error condition. Although these may adequately reflect the case of a centralized policy evaluation and enforcement, it it inadequate in the case of distributed flow control. For example, consider the three regions C1, C2 and C3 controlled by three independent XACML engines that individually emit one of *permit*, *deny*, *inapplicable policy* or *error*, and these decisions are collected by a centralized PDP to give the final decision. The last two decisions from either component should result in denial of permissions for the transfer request of water. For example, if the request to transfer 2000 gallons of water per minute to city A from the reservoir as drawn in Figure 1, and the master PDP request the slave PDP's governing regions C1 and C2 for a subsequent request to transfer water, then they must come back with a reply saying the amount of water they are willing to transfer and what other conditions must be satisfied in order to grant this request. For example, PDP governing region C1 may say that it can grant this request by opening *Valve 1* and *Valve 2*, and closing *Valve 5* provided that *Valve 3* is closed. Simultaneously, the PDP governing region C2 may say that it is willing to grant the request provided that *Valve 3* and *Valve 4* are open and *valve 2* is closed. Consequently, the master PDP must now decide on which path it chooses, provide that all pre-requisite conditions can be met. Otherwise, it has to *Deny* this request. Consequently, there is a need for the secondary PDPs to convey to the master PDP the pre-requisite state information for the former to grant the request, the state that it expects the concerned resources to maintain while the granted resources are being used and the post-requisite state of the resources that it expects the resources to be in when the requestor relinquishes the use of the resource. Because the granted permission or denial are conditional upon these state information, we call it the *decision context*. Consequently, the decision is valid only if the context is satisfied during the enforcement process, consequently providing support for usage control [27]. Furthermore, this *decision context* of how PDP reached a policy decision has to be made available to *PEP's* for the correct enforcement of the *PDP's Decision*. As stated in Section 2, the preferred syntax to state properties of resources on the WEB is RDF, we provide a preliminary definition for the *decision context* using RDF as follows, and refine it in Section 5.

Definition 1 (preliminary definition of decision context) *The decision context consist of a triple of RDF statements, referred to as pre-context, during-context and post-context.*

The *decision context* is different than *obligations* as the latter imposes future requirements on that the PEP must adhere to. The XACML specification specifies an

obligation as an action performed by the PEP in conjunction with the enforcement of an authorization decision. This definition separates the enforcement action from obligation processing. Such a system where the enforcement is distributed should be able to maintain transactional semantics based on the *decision context*.

Another important aspect of the decision context is that if the decision is not going to be valid during the enforcement of the decision, the actions of the PEP should be rolled back. In the previous example if Valve 1 fails to enforce the policy decision (fails to open), the decision context (both Valve 1 and Valve 2 should open) is invalid and hence Valve 2 must also be closed (rolled back). This transactional semantics based on the decision context should be adhered to by *PEP's* execution.

5 Architecture and Design

In order to enforce distributed access control using XACML, we propose having a separate PDP at every *site* that need to evaluate a local access control policy. Then these local policies communicate their decisions with the encompassing context to the master PDP that collects all such decisions and renders the final access control decision to the external requestor. This arrangement can be repeated recursively, creating a hiearchy of PDPs that are arranged in a tree structure. Consequently, the arrangement applicable to our example scenario is given in the left hand side of Figure 3. For the example given in Figure 1, because the web services are arranged so that the top level service depends upon sub-services C1, and C2, and C1 depends on C2. Consequently, the PDPs given in the left hand side of Figure 3 inherits the same hierarchical structure. Consequently, the corresponding policy enforcement points of these services C1, C2 and C3 should be coordinated in the same hierarchical manner. The right hand side Figure 3 is there to show that the latter coordination can be specified and enforced using the *Business Process execution Language (BPEL)* and will be explained shortly. Accordingly, these PEPs need to agree to enforce the decision within the decision context that all PDPs will pass along with the decision.

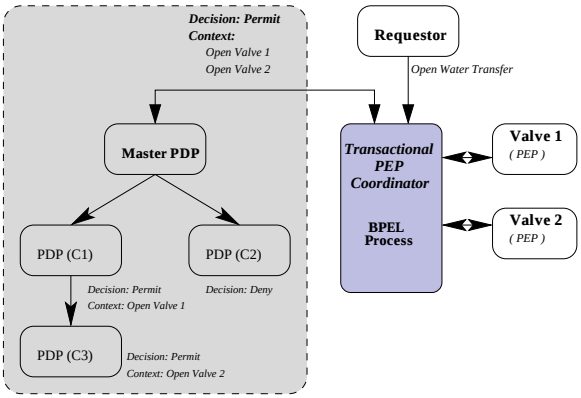

Fig. 3: *Architecture and Design*

5.1 Context

Because the decision context expresses constraints under which permissions can be granted to acquire resources, we express them as RDF statements. Because the request to the resource has three constraints, these are a triple of RDF constraints, referred to as the pre-context, during-context and the post-context, as defined in Definition 1. (We provide an example, soon after we show how the *decision context* is used).

In order to use this context elements, we alter the access control policies and make the *decision context* conditionals in the access control policies. We do so by using the ⟨*Conditional*⟩ in XACML policies. Furthermore, in order to express and evaluate the enriched policies, we have enriched the XACML runtime. In order to minimize the alteration, we pass the *design context* as a *MetaBoolean* type in XACML, so that it the XACML runtime allows us to use our own evaluator, for which we use and RDF evaluation engine. We formally define the *MetaBoolean* type in XACML as follows.

```
1<xs:element name="MetaBoolean" type="xacml:MetaBoolean"/>
2<xs:complexType name="MetaBooleanType">
3    <xs:sequence>
4        <xs:element ref="xacml:Context" minOccurs="0" maxOccurs="unbounded"/>
5    </xs:sequence>
6    <xs:attribute name="eval" type="http://www.w3.org/2001/XMLSchema#boolean" use="required"/>
7</xs:complexType>
```

Type 1: MetaBoolean Type

In order to give examples of the *decision context* and its embedding as a conditional in the access control policy as a *MetaBoolean* type, consider the case where a request to transfer 3000 gallons of water is received by the master PDP in Figure 1. In response to the request, the PDP retrieves the appropriate policy, given as `Policy 1`.

As shown, the top level policy expressing Company 1's (i.e. the one that governs region c1) in line 1 says that it consists of a policy set. Although the target of this policy is omitted for brevity, it has a set of rules starting in line 8. For brevity we show only one rule, the one starting in line 8 and ends in line 14. Line 1 says that these rules are to be applied in the *denial override*, meaning that if any component rule evaluates to a denial, then the resulting decision returned to the calling PDP will be a denial. The reason being that a synchronous delivery succeeds only if all of its requirements succeed.

In line 8 of the described rule, Policy 1 calls for an evaluation of (*c1:policy-check-valve-1*) and in line 17 has a Policy reference to Company 2's (c2) policy for opening Valve 2 (*c2:policy-check-valve-2*). The *c2:policy-check-valve-2* listing is shown in `Policy 2`.

The intended effect of Policy 1 and Policy 2 taken together is that it will ask the PDP governing C2 to evaluate Policy 2 and send back the resulting decision along with its decision context. Then our enhanced XACML runtime will combine the decisions by using the *policy-combining-algorithm:deny-overrides* algorithm before returning the Decision to the PEP Coordinator.

```
1<PolicySet PolicySetId="c1:policyset-check-reservoir-city:a" PolicyCombiningAlgId="policy-combining-algorithm:deny-
      overrides">
2  <Target>
3    ....
4  </Target>
5  <Policy PolicyId="c1:policy-check-valve-1" RuleCombiningAlgId="rule-combining-algorithm:deny-overrides">
6    <Target/>
7    <Rule RuleId="c1:valve-1-scheduling-check" Effect="Permit">
8      ....
```

```
9      <Condition>
10      <Apply FunctionId="c1:function:schedeule-valve-1">
11      </Apply>
12     </Condition>
13    </Rule>
14    ....
15  </Policy>
16  <PolicyIdReference>c2:policy-check-valve-2</PolicyIdReference>
17</PolicySet>
```

Policy 1: Top-Level C1 PolicySet

```
1 <Policy PolicyId="c2:policy-check-valve-2" RuleCombiningAlgId="rule-combining-algorithm:deny-overrides">
2  <Target/>
3  <Rule RuleId="c2:valve-2-scheduling-check" Effect="Permit">
4    ....
5   <Condition>
6    <Apply FunctionId="c2:function:schedule-valve2">
7    </Apply>
8   </Condition>
9  </Rule>
10  ....
11 </Policy>
```

Policy 2: C2's Check for Valve-2

We now provide examples of *decision contexts* returned by the evaluators of these two policies.

```
1<MetaBoolean eval="true">
2  <Context>
3    <PreContext>
4      <rdf:RDF xmlns:rdfs="http://www.gmu.edu/xacml/owl/ontology/#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
             syntax-ns#" xmlns:gmu="http://www.gmu.edu/xacml/rdf#">
5       <rdf:Description rdf:about="http://gmu/valve1">
6        <gmu:state xml:lang="http://www.w3.org/2001/XMLSchema#string">close</gmu:state>
7        <gmu:ID>Valve1</gmu:ID>
8       </rdf:Description>
9      </rdf:RDF>
10   </PreContext>
11   <DuringContext>
12     <rdf:RDF xmlns:rdfs="http://www.gmu.edu/xacml/owl/ontology/#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
             syntax-ns#" xmlns:gmu="http://www.gmu.edu/xacml/rdf#">
13      <rdf:Description rdf:about="http://gmu/valve1">
14       <gmu:capacity xml:lang="http://www.w3.org/2001/XMLSchema#int">1000</gmu:capacity>
15       <gmu:state xml:lang="http://www.w3.org/2001/XMLSchema#string">open</gmu:state>
16       <gmu:ID>Valve1</gmu:ID>
17      </rdf:Description>
18     </rdf:RDF>
19   </DuringContext>
20   <PostContext>
21     <rdf:RDF xmlns:rdfs="http://www.gmu.edu/xacml/owl/ontology/#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
             syntax-ns#" xmlns:gmu="http://www.gmu.edu/xacml/rdf#">
22      <rdf:Description rdf:about="http://gmu/valve1">
23       <gmu:state xml:lang="http://www.w3.org/2001/XMLSchema#string">close</gmu:state>
24       <gmu:ID>Valve1</gmu:ID>
25      </rdf:Description>
26     </rdf:RDF>
27   </PostContext>
28  </Context>
29</MetaBoolean>
```

MetaBoolean 1: Generated by valve-1-scheduling-check

The *decision context* returned by the evaluation of Policy 1 is given in *Metaboolean 1*. As the listing shows, the evaluation returned is *true* conditional upon the the pre-context, during-context and post-context given between lines (3-13), (14-25) and (26-36). As shown, all three of them refer to an RDF descriptions in lines 8, 19 and 31. The three lines immediately following the references to RDF descriptions, namely (9-10), (20-22) and (32-33) describe the three *decision context*s. The pre-context in lines 9-10

says that the *state* property of *valve 1* must be closed. The during-context in lines (20-21) says that the the *state property* of *valve 1* must be open and further state that the *capacity* property of the valve must be set to 1000 gallons per minute. It is conceivable that the RDF semantics of a valve is such that other properties such as the time duration that it remains open should also be specified. Note that these properties can be specified using RDF polices for the individual resources that are pertinent for the requests to be completely specified so that the PEP can enforce the permission. The post context contained in lines (32-33) say that *close* property of the *valve* must be closed.

Similarly, *Metaboolean 2* given below states the *decision context* returned by the PDP that governs region C2 to the master PDP. As previously explained, if Rule *c2:valve-2-scheduling-check* evaluates to *permit* the function *c2:function-schedule-valve-2* within the conditional returns a *decision context* that requires *valve 2* to be closed prior to granting the requests and it be opened during the usage at the rate of 1000 gallons per minute.

```
1 <MetaBoolean eval="true">
2   <Context>
3     <PreContext>
4       <rdf:RDF xmlns:rdfs="http://www.gmu.edu/xacml/owl/ontology/#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
              syntax-ns#" xmlns:gmu="http://www.gmu.edu/xacml/rdf#">
5        <rdf:Description rdf:about="http://gmu/valve2">
6         <gmu:state xml:lang="http://www.w3.org/2001/XMLSchema#string">close</gmu:state>
7         <gmu:ID>Valve2</gmu:ID>
8        </rdf:Description>
9       </rdf:RDF>
10    </PreContext>
11    <DuringContext>
12      <rdf:RDF xmlns:rdfs="http://www.gmu.edu/xacml/owl/ontology/#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
              syntax-ns#" xmlns:gmu="http://www.gmu.edu/xacml/rdf#">
13       <rdf:Description rdf:about="http://gmu/valve2">
14        <gmu:capacity xml:lang="http://www.w3.org/2001/XMLSchema#int">1000</gmu:capacity>
15        <gmu:state xml:lang="http://www.w3.org/2001/XMLSchema#string">open</gmu:state>
16        <gmu:ID>Valve2</gmu:ID>
17       </rdf:Description>
18       <rdf:Description rdf:about="http://gmu/valve5">
19        <gmu:state xml:lang="http://www.w3.org/2001/XMLSchema#string">close</gmu:state>
20        <gmu:ID>Valve5</gmu:ID>
21      </rdf:Description>
22      </rdf:RDF>
23    </DuringContext>
24    <PostContext>
25      <rdf:RDF xmlns:rdfs="http://www.gmu.edu/xacml/owl/ontology/#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
              syntax-ns#" xmlns:gmu="http://www.gmu.edu/xacml/rdf#">
26       <rdf:Description rdf:about="http://gmu/valve2">
27        <gmu:state xml:lang="http://www.w3.org/2001/XMLSchema#string">close</gmu:state>
28        <gmu:ID>Valve2</gmu:ID>
29       </rdf:Description>
30      </rdf:RDF>
31    </PostContext>
32  </Context>
33 </MetaBoolean>
```

MetaBoolean 2: Generated by Rule valve-2-scheduling-check

There are two pertinent issues here. The first is that either C1 or C2 can consult its own policy and decide how to internally schedule water flow and control their own rates. Secondly they can consult their own scheduling rates so that the resource usage adheres to its own semantics of operations. Our ongoing work addresses these two issues. We now describe the Policy evaluation process used by a master PDP to evaluate an XACML request, upon receipt of *decision context*s.

5.2 Evaluating the Decision Context

As shown the decision context consists of pre-context, during-context and post-context. Consequently, every child PDP sends its decision context to its parents PDP as a response to a distributed request.

The parent PDP then collects all decision context of its children and combine them to determine if the collected decision context ate consistent. In order to do so, the PDP evaluation collects all *pre context*s, *during context*s and *post context*s separately, and if all of them are determined to be consistent, then evaluate the design context to be consistent. Conversely, if either of them is found to be inconsistent, then the design context is determined to be inconsistent. Conversely, if any of the design context are determined to be `Indeterminate`, then the design context is said to be indeterminate.

The consistency of the decision context are determined using an RDF rule evaluation engine. The rules supplied to this engine define which contexts are consistent. The RDF rules state which combination of RDF property instances imply falsehood. This process, we refer to as *decision context unification* is performed in a hierarchical manner as shown in Figure 4

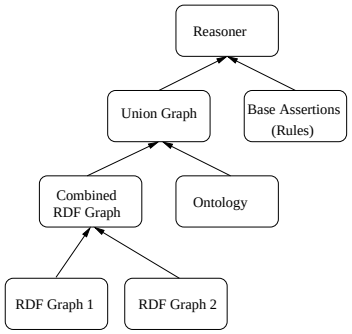

Fig. 4: *Context Unification*

In order for the process to work, we construct an OWL Ontology and feed it to an RDF reasoner. The Ontology defines the model and specifies the restrictions on the model. A brief snippet of the ontology used to reason about the water system is shown in the listing `Valve Ontology`

```
<owl:Class rdf:ID="Valve">
 <owl:Restriction>
  <owl:onProperty rdf:resource="state"/>
  <owl:cardinality>1</owl:cardinality>
 </owl:Restriction>
</owl:Class>
....
<owl:DatatypeProperty rdf:ID="capacity">
 <rdfs:range>
   <rdfs:Datatype>
     <xsp:base rdf:resource="&xsd;int"/>
    <xsp:minInclusive rdf:datatype="&xsd;int">0</xsp:minInclusive>
    <xsp:maxInclusive rdf:datatype="&xsd;int">4000</xsp:maxInclusive>
   </rdfs:Datatype>
 </rdfs:range>
</owl:DatatypeProperty>
```

Valve Ontology

The important facts stated in the *valve ontology* are as follows:

1. Cardinality of state of a valve is one.
2. for valve 1 the capacity can not be more than 4000 gallons/min.

```
[ rule-conflict:
 ( ?v1 <http://www.gmu.edu/xacml/rdf#MUST_VALVE_CLOSE> ?bag1 )
 ( ?bag1 ?m ?v2 )
 ( ?v3 <http://www.gmu.edu/xacml/rdf#MUST_VALVE_OPEN> ?bag2 )
 ( ?bag2 ?m ?v2 )
 (?m rdf:type rdfs:ContainerMembershipProperty)
->
 ( ?v1 <http://www.gmu.edu/xacml/rdf#conflict> ?v3 )
]
```

Valve Conflict Rule

In additon to the ontology, there can be user specified rules that must not be violated. These rules could capture other business/state requirements not captured by the ontology. The listing `Valve Conflict Rule` states that the same valve can not be open and closed at the same time.

If the *Context Unification* leads to a conflict (based on the OWL/RDF Rules) the policy evaluation results in an `Indeterminate` result. For example, if the `MetaBoolean` 1 specified in the < *DuringContext* > that Valve 5 must be open then this would be in conflict with < *DuringContext* > of `MetaBoolean 2` because the Ontology has cardinality restriction on the *state* resource of Valve. In the running example, there are not conflicts after the *Context Unification*. The Master PDP thus evaluates the request and the decision is sent back to the *PEP Coordinator*. This is shown in the listing `Decision` 1.

```
<Response>
 <Result ResourceId="water:3000">
  <Decision>Permit</Decision>
  <Status>
   <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
  </Status>
  <Context>
   <PreContext>
     <rdf:RDF xmlns:rdfs="http://www.gmu.edu/xacml/owl/ontology/#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
           syntax-ns#" xmlns:gmu="http://www.gmu.edu/xacml/rdf#">
      <rdf:Description rdf:about="http://gmu/valve1">
       <gmu:state xml:lang="http://www.w3.org/2001/XMLSchema#string">close</gmu:state>
       <gmu:ID>Valve1</gmu:ID>
      </rdf:Description>
      <rdf:Description rdf:about="http://gmu/valve2">
       <gmu:state xml:lang="http://www.w3.org/2001/XMLSchema#string">close</gmu:state>
       <gmu:ID>Valve2</gmu:ID>
      </rdf:Description>
   </rdf:RDF>
   </PreContext>
   <DuringContext>
     <rdf:RDF xmlns:rdfs="http://www.gmu.edu/xacml/owl/ontology/#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
           syntax-ns#" xmlns:gmu="http://www.gmu.edu/xacml/rdf#">
      <rdf:Description rdf:about="http://gmu/valve1">
       <gmu:capacity xml:lang="http://www.w3.org/2001/XMLSchema#int">1000</gmu:capacity>
       <gmu:state xml:lang="http://www.w3.org/2001/XMLSchema#string">open</gmu:state>
       <gmu:ID>Valve1</gmu:ID>
      </rdf:Description>
      <rdf:Description rdf:about="http://gmu/valve2">
       <gmu:capacity xml:lang="http://www.w3.org/2001/XMLSchema#int">1000</gmu:capacity>
       <gmu:state xml:lang="http://www.w3.org/2001/XMLSchema#string">open</gmu:state>
       <gmu:ID>Valve2</gmu:ID>
      </rdf:Description>
      <rdf:Description rdf:about="http://gmu/valve5">
       <gmu:state xml:lang="http://www.w3.org/2001/XMLSchema#string">close</gmu:state>
       <gmu:ID>Valve5</gmu:ID>
     </rdf:Description>
   </rdf:RDF>
   </DuringContext>
   <PostContext>
     <rdf:RDF xmlns:rdfs="http://www.gmu.edu/xacml/owl/ontology/#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
           syntax-ns#" xmlns:gmu="http://www.gmu.edu/xacml/rdf#">
      <rdf:Description rdf:about="http://gmu/valve1">
       <gmu:state xml:lang="http://www.w3.org/2001/XMLSchema#string">close</gmu:state>
```

```
42         <gmu:ID>Valve1</gmu:ID>
43        </rdf:Description>
44        <rdf:Description rdf:about="http://gmu/valve2">
45         <gmu:state xml:lang="http://www.w3.org/2001/XMLSchema#string">close</gmu:state>
46         <gmu:ID>Valve2</gmu:ID>
47        </rdf:Description>
48      </rdf:RDF>
49      </PostContext>
50    </Context>
51  </Result>
52 </Response>
```

Decision 1: Permit under Top-Level PolicySet

5.3 The Decision enforcement process

The XACML specification places no restrictions on the policy enforcement point (PEP). Our additions require PEP to be able to execute PDP decisions transactionally. In order to do that we have the PEP associated with master PDP, referred to as the PEP coordinator, that communicates and with slave PEPs (i.e. PEP that correspond to slave PDPs).

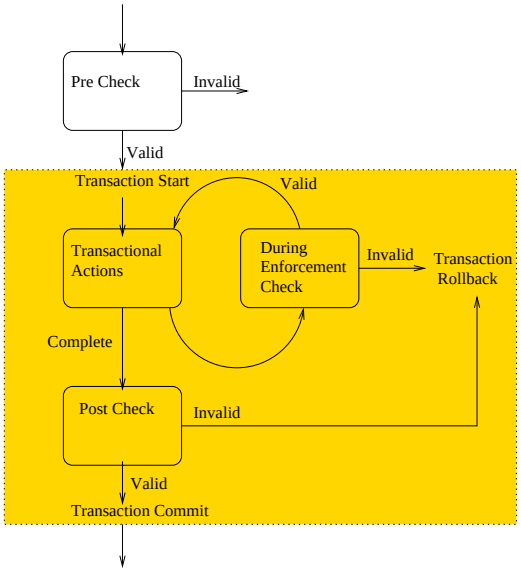

Fig. 5: *Context Evaluation within PEP*

Finally, because the decision was permitted only if the *decision context* is valid, it becomes the responsibility of the PEP to ensure that the policy decisions are enforced in the *decision context*. In order to do so, The PEP coordinator also needs to be able to understand the RDF model specified by the PDP Decision and follow the state transition specified in Figure 5. This implies that it should be able to monitor the state of the

control system and abort if global state is in conflict with the *decision context*. The algorithm used by our PEP coordinator is given in Algorithm **??** in the Appendix Section. As shown in the Figure 5, the

1. **PreContext:** Is checked for consistency before the start of the Transactional actions of PEP. Only when the pre check is valid that the decision holds.
2. **DuringContext:** If the decision holds (pre check passes), the PEP makes sure that during enforcement context of the decision is valid throughout the execution of the transactional actions of PEP. If at any point during enforcement context becomes invalid then PEP needs to roll back the transactional actions.
3. **PostContext:** Finally post check context verifies that post conditions of the decision hold. If it isn't valid then PEP needs to roll back the transactional actions.

The BPEL workflow engine is adequate to provide the above mentioned behavior of coordinated PEPs. Because BPEL natively does not understand RDF, we use a RDF Interpreter Web Service for that domain to synthesize the RDF generated by the PDP.

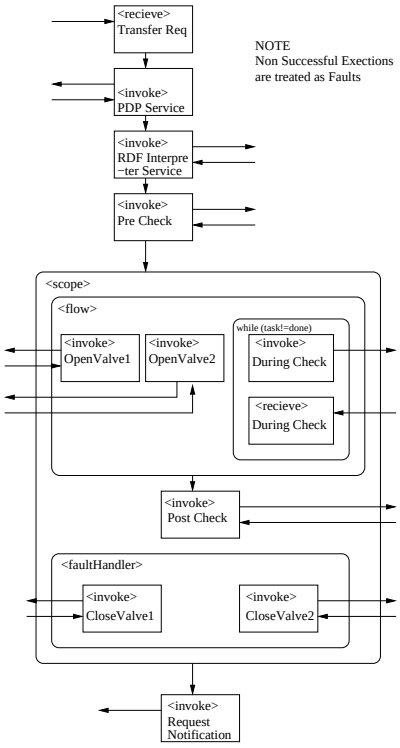

Fig. 6: *BPEL PEP Coordinator*

The construction of BPEL workflow is shown in Figure 6.

6 Related Work

Bertino *et al.* [4] propose an extension to BPEL [19] for expressing RBAC-like [23] authorization requirements (called RBAC_WS_BPEL) for BPEL workflows. The authors propose to encode RBAC requirements in XACML syntax (much like the XACML-RBAC profile [20]) and depend upon the BPEL engine to enforce this specification. However, the authors do not consider a use case for distributive access control requirements as introduced in this paper and consequently they cannot support practical access control requirements, such as those requiring separation duty principle etc, because as far as we know existing XACML runtimes do not support them. We enhance XACML syntax with contextual information and implement a fully-distributed access controller to enforce practical access control use-cases.

FlexFlow [10] is another general purpose modeling language for capturing workflow representations in tree structure and expressing/enforcing access control requirements on the work-flow. Flex flow is based syntactically on *FAF [14]* (locally stratified Horn-clause programs). However, the work is semantic-web agnostic, and consequently authors ignore real-life scenarios such as those programmable using BPEL. The main disadvantage of using FlexFlow for BPEL security is the simplifying assumptions made in the design that do not take into account the *distributed-ness* of the access decision evaluation and enforcement.

Atluri, Huang *et al.* [2] consider a related security problem of multi-level secure work-flow systems, where work-flows at a higher security label should not be dependent upon work-flows at a lower security label. The authors identify dependencies in the work-flows into different categories, thus identifying security issues (*i.e.*, high to low dependency) and prevent them. This work is orthogonal to our domain, where we aim to secure any workflow based on the security policy – enforcing it throughout the workflow, including at distributed Web Services that are the part of the secured workflow.

Bertino, Ferrari and Atluri in [5] present a logical language for generic workflows that can be broken down to a sequence of tasks. The main aim of this work is to be able to express RBAC-like authorization constraints and enforce them for workflows. However, this work does not consider the runtime issues like policy enforcement distribution, exclusive usage etc that require more complex control algorithms between enforcement points. In this aspect, our work provides a complete end to end security cover for workflows. Because we use XACML based policies for expressing security requirements, we can utilize earlier extensions like the *lock manager enhancements* by Dhankhar *et al.* [11] to enforce RBAC-like authorization constraints.

Several query languages have been proposed for querying RDF meta-data [8, 22, 24, 9, 25], *etc.* These query languages have been used to implement RDF reference implementations like Jena [17], Redland [3], ICSForth Suite [1], *etc.* In this work we use Jena RDF API and reference implementation for integrating XACML with BPEL. The choice is purely due to the free availability of this API and reference implementation.

Dhankhar *et al.* have extended reference XACML implementation [26] with extensions to enforce exclusive use [11] and distributed policy evaluation [12] within a nested transaction tree framework. This paper extends their work to *fully distribute* evaluation and enforcement of XACML policies. In that sense we extend their work to include

distributed policy enforcement, including conflict management during policy enforcement.

Fox in [13] provides several examples where contextual information is necessary for decision evaluation. Though we don't consider context at the level of granularity as described by Fox, but, it is a validation of our claim that decisions are arrived in a particular context and valid only within a related context.

7 Conclusion

Following our previous work, we are in the process of fully decentralizing the XACML reference implementation. That is, we would like to have the XACML reference implementation be able to evaluate and enforce policies that refers to resources available anywhere on the world-wide web. That entails the policy decision point (PDP) to be able reach the appropriate policy and access governing authority of the referenced resource and be able to seek and obtain the permissions for the requestor.

During our research and development process, we realized that in such a decentralized system, the resource owners may impose condition that the requestor has to adhere to in order to use the resource as requested. They have been named *decision context* because the PDPs decisions are to be evaluated under these conditions. They are passed on to the request originators PDP and are passed back to the policy enforcement points.

The *decision context* we designed have been specified using RDF and OWL, that specify how the resource can be used. In addition, other rules that constitute consistent use is also passed to a PDP. The PDP then evaluates if the request is permitted, and if so under which amalgamated request and passes that information to a master PEP, that distributes them over to all other PEPs.

We have also realized that our PEP coordination can be specified and enforced using BPEL. Our initial experiments in implementing the stated examples have resulted a reasonable performance. Our ongoing work addresses the process of auto-generating all *decision context* and passing and enforcing them using a BPEL process in a more general context.

References

1. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ICS-FORTH RDFSuite: Managing voluminous rdf description bases. In *Second International Workshop on the Semantic Web (SemWeb'2001)*, May 2001.
2. V. Atluri, W. Huang, and E. Bertino. A semantic-based execution model for multilevel secure workflows. *Journal of Computer Security*, 8(1), 2000.
3. D. Beckett. The design and implementation of the Redland RDF application framework. In *Tenth International World Wide Web Conference, (WWW10)*, May 2001.
4. E. Bertino, J. Crampton, and F. Paci. Access control and authorization constraints for WS-BPEL. In *IEEE International Conference on Web Services (ICWS 2006)*, pages 275–284, 2006.
5. E. Bertino, E. Ferrari, and V. Atluri. A flexible model supporting the specification and enforcement of role-based authorization in workflow management systems. In *ACM Workshop on Role-Based Access Control*, pages 1–12, 1997.
6. D. Brickley and R. Guha. Resource Description Framework (RDF) Schema Specification 1.0: RDF schema. W3C workding Draft, 2003.

7. D. Brickley, R. Guha, and B. McBride. RDF vocabulary description language 1.0: RDF schema. W3C Recommendation, February 2004.
8. J. Broekstra and A. Kampman. SeRQL, a second generation RDF query language. In *SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, Amsterdam, Nov 2004.
9. L. Chen, A. Gupta, and M. E. Kurul. A semantic-aware RDF query algebra. In *12th International Conference on Management of Data (COMAD)*, Hyderabad, Dec 2005.
10. S. Chen, D. Wijesekera, and S. Jajodia. Flexflow: A flexible flow control policy specification framework. In *17th Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec 2003)*, pages 358–371, 2003.
11. V. Dhankhar, S. Kaushik, and D. Wijesekera. XACML policies for exclusive resource usage. In *21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec 07)*, 2007.
12. V. Dhankhar, S. Kaushik, D. Wijesekera, and A. Nerode. Evaluating distributed XACML policies. In *2007 ACM Workshop on Secure Web Services (SWS 2007)*, November 2007.
13. M. S. Fox. *Knowledge Representation for Decision Support Systems*. Elsevier, 1985.
14. S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, 26(2):214–260, 2001.
15. S. Kaushik, C. Farkas, D. Wijesekera, and P. Ammann. An algebra for composing ontologies. In *International Conference on Formal Ontology in Information Systems (FOIS'06)*, November 2006.
16. G. Klyne, J. J. Carroll, and B. McBride. Resource description framework (RDF): Concepts and abstract syntax. W3C Recommendation, 2004.
17. B. McBride. Jena: Implementing the rdf model and syntax specification. In *Second International Workshop on the Semantic Web (SemWeb'2001)*, May 2001.
18. L. Miller, A. Seaborne, and A. Reggiori. Three implementations of SquishQL, a simple RDF query language. In *International Semantic Web Conference (ISWC)*, pages 399–403, 2002.
19. OASIS. Business process execution language for web services, May 2003.
20. OASIS. XACML profile for role based access control (rbac). http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf, Feb 2004.
21. OASIS. Extensible access control markup language, Feb 2005.
22. E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. http:/www.w3.org/TR/rdf-sparql-query, Apr, 2005.
23. R. Sandhu, D. Ferraiolo, and R. K. D. The NIST model for role based access control: Towards a unified standard. In *5th ACM Workshop on Role Based Access Control*, July 2000.
24. A. Seaborne. A query language for RDF. http:/www.w3.org/Submission/2004/SUBM-RDQL-20040109, 2004.
25. M. Sintek and S. Decker. Triple, an RDF query, inference and transformation language. In *Deductive databases and knowledge management (DDLP)*, 2001.
26. Sun Microsystems. Sun's XACML implementation. http://sunxacml.sourceforge.net/index.html, July 2004.
27. X. Zhang, J. Park, F. Parisi-Presicce, and R. Sandhu. A logical specification for usage control. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 1–10, New York, NY, USA, 2004. ACM.