A XML Policy-Based Approach for RSVP

Emir Toktar, Edgar Jamhour, Carlos Maziero
Pontifical Catholic University of Paraná, PUCPR, PPGIA
{toktar, jamhour, maziero}@ppgia.pucpr.br

Abstract

This work proposes a XML-based framework for distributing and enforcing RSVP access control policies, for RSVP-aware application servers. Policies are represented by extending XACML, the general purpose access control language proposed by OASIS. Because RSVP is a specific application domain, it is not directly supported by the XACML standard. Hence, this work defines the XACML extensions required for representing and transporting the RSVP access control policy information. The XACML-based framework is proposed as an alternative to the PCIM-based approach, proposed by IETF. The work shows that, while XACML is easier to understand and deploy, it lacks the flexibility offered by PCIM. However, by properly extending XACML, the paper shows that the OASIS model is suitable for defining complex access control policies for specific domains, such as RSVP.

1. Introduction

Policy based network management (PBNM) is an important trend for IP-based networks. Recent works developed by IETF have defined a standard model for representing policies on different areas of network management. The groundwork of this model is the PCIM (Policy Core Information Model), defined by RFC 3060 [5]. PCIM is a platform independent object-oriented information model. The model defines a generic strategy for representing network policies as aggregations of rules expressed in terms of conditions and actions. PCIM is an abstract model, and it does not define sufficient elements for describing policies for particular areas of network management. To address particular areas, PCIM needs to be extended. IETF itself has already introduced PCIM extensions for representing IPsec and QoS [10] policies. Outside IETF, other works explored extensions of PCIM for the area of access control [6].

Besides IETF, others organizations are proposing standard policy models for PBNM. The OASIS (Organization for the Advancement of Structured Information Standards) proposed a language for representing access control policies, on general purpose, denominated XACML (eXtensible Access Control Markup Language). There are several differences between the PCIM and the XACML approach. While PCIM is a core model for representing policies on any area of network management, XACML is dedicated to access control. Because PCIM is an abstract model, the implementation of policies models based on PCIM is a rather complex task. The XACML, by the other hand, is simpler of being implemented and deployed. However, XACML can lack the flexibility for addressing specific application domains.

Based on this argumentation, this work proposes the use of the XACML for modeling and distributing RSVP access control policies for RSVP-aware application servers. Because RSVP is a specific application domain, it is not directly supported by the XACML standard. Hence, this work defines the XACML extensions required for representing and transporting the RSVP access control policy information. The paper compares the proposed XACML-based approach with the standard PCIM-based approach with respect to implementation and deployment. By establishing the parallels with PCIM-based approach, this work defines the futures extensions required for extending this proposal to other network elements, such as routers.

This paper is structured as follows: section 2 presents a short review of the main aspects related to RSVP policy access control. Section 3 presents an analysis of the models that can be employed for describing RSVP access control policies, and the strategies for distributing and enforcing those policies. The section 4 presents a short review of the XACML model. The section 5 describes how the XACML

can be used for describing RSVP policies, and presents the required extensions for adapting XACML to the RSVP issue. The section 6 describes how to implement the framework for distributing and enforcing the RSVP policies described in XACML. Finally, the conclusion reviews the principal aspects of this study and indicates the future works.

2. RSVP Policy Control

This section introduces a brief review of the RSVP protocol, defining the concept of RSVP policy control and presenting the important terms that will be utilized in the next sections. The RSVP signalization is composed by a set of standard messages. The most important messages are PATH and RESV. The emitter always initiates the QoS negotiation by sending the message PATH to the receiver. The PATH message has double function. It defines the QoS parameters the receiver must request for the network in order to satisfy the requisites of the application. It defines, as well, the path the other RSVP messages and the flow of data will follow between the emitter and the receiver. A flow of data on RSVP is a sequence of messages with the same origin, with same expected QoS, and one or more destinations.

The receiver, on accepting the PATH message, initiates the process of flow reservation sending the RESV message to the emitter, along the reverse way defined by the PATH message. The RESV message consists of a flow descriptor, formed by the *flowspec* and *filterspec* objects. The *filterspec*, along with the specification of the session, defines which packets of data (RSVP flow) must benefit from the QoS reservation. The QoS specification is defined by *flowspec* using two data structures: *Rspec* (Reserve Spec), that indicates the service class expected and *Tspec* (Traffic Spec) that specifies what will be transmitted.

The QoS is enforced for a particular data flow by a mechanism called “traffic control”. The traffic control mechanism includes: a packet classifier and a packet scheduler. The mechanism utilizes the Token Bucket algorithm for regulating the traffic of data according to the bandwidth limits specified by the *Tspec* parameters. During the resource reservation setup, two local decision modules evaluate a RSVP request: the “policy control module” and the “admission control module”. The admission control module determines whether the node (host or router) has sufficient resources available for satisfying the QoS request. The policy control module determines whether the user has administrative permission for obtaining the reservation [2]. The parameters for policy and admission control are not defined and controlled by the RSVP. The protocol merely transports the parameters to the appropriate module for interpretation.

According to the RFC2205, the sender application must specify the type of service most appropriate for its requisites of transmission by passing the related information to the RSVP daemon in the host machine [2]. The RSVP daemon after being called, query the local decision modules, verifying resources and authorization and, being allowed, initiates the exchange of RSVP messages with the nearest network element in the path to the receiver.

As explained in the next sections, the purpose of the work described in this paper consists in defining and implementing a mechanism for configuring the RSVP access control policies (“*policy control*”) for RSVP-aware application servers by using XACML. This proposal also supply the information for defining the *Tspec* parameters transported in the PATH and RESV messages. The *Tspec* information is used for “*admission control*” by the network elements along the path between the transmitter and the receiver.

3. RSVP Policy Control Strategies

In this paper, the strategy for representing, distributing and enforcing RSVP access control policies follows Policy Based Network Management (PBNM) approach. The concept of PBNM is already widely adopted by organizations that propose Internet standards, such as IETF [14] and the OASIS [7]. Although the definitions for PBNM could diverge according to the organization, the main concepts are relatively universal. The basic idea for PBNM is to offer a strategy for configuring policy on different network devices using a common management framework, composed by a policy server, denominated PDP (Policy Decision Point) and various policy clients, denominated PEPs (Policy Enforcement Points) [12]. The PDP is the entity responsible for storing and distributing the policies to the diverse nodes in the network. A PEP is, usually, a network node component responsible for interpreting and applying the

policies received from the PDP. The PBNM approach can be applied in various aspects of network management. This section will explore how this approach can be applied for managing access control policies in RSVP server (sender) applications.

The IETF explores the concept of PBNM according to two strategies, denominated outsourcing and provisioning. In the outsourcing strategy, the PEP sends a request to the PDP when it needs to make a decision. For example, considering the access problem on RSVP, the PEP would represent the server application (or more precisely, the policy component embedded in the server application). On receiving a request from a client, the PEP would send a request to the PDP in order to determine if the client has the permission for asking the reservation. The PDP then would interpret the policies and would send a final decision to the PEP, informing if the solicitation is permitted or denied. In the provisioning approach, the PEP, as being initialized, would receive from the PDP the set of policies needed for its decision. The policy information received from the PDP is locally stored by the PEP according to a locally defined scheme called PIB (Policy Information Base). On receiving a reservation request, the PEP would consult its locally stored policies and would make the decision by itself. In this approach, the communication between the PEP and the PDP is required only when there is necessity of updating the policies in the PEPs (e.g., the network administrator modifies a policy in the PDP concerning the PEP).

IETF define as well a standard protocol for supporting the communication between the PEP and the PDP. This protocol is denominated COPS (Common Open Policy Service). The basic structure of the COPS protocol is described in the RFC 2748 [1]. The COPS protocol supports both models of policy control, i.e., "outsourcing" and "provisioning". In the case of the provisioning approach, additional specifications were required and, the protocol was renamed to COPS-PR. The basic structure of the COPS-PR protocol is described in the RFC 3084 [3].

The IETF already published various works concerning the use of PBNM approach for RSVP policy control. The works cover the definition of a framework for admission control [14] and the utilization of COPS in outsourcing (COPS-RSVP) [4] and provisioning (COPS-PR) models. The provisioning approach is still under development, being necessary additional definitions for its complete specification.

The XACML proposal from OASIS also describes that its implementation could follow the approach PDP/PEP. However, OASIS does not make a distinction between the outsourcing and provisioning models, neither defines a standard protocol for supporting the communication between the PEP and the PDP. An analysis of the XACML indicates, however, that it was primarily conceived for supporting the outsourcing approach (see section 4).

An important difference between the approaches adopted by OASIS and IETF relates to how policies are represented and stored. OASIS proposes XACML as a particular model for access control, represented and stored as XML documents. On the other side, IETF defines PCIM as a generic model, independent from the way the policies will be represented and stored. The PCIM model is abstract, and needs to be extended in order to support particular areas of management, such as QoS [10]. IETF indicates strategies for mapping the information models to LDAP (Lightweight Directory Access Protocol) schemas, but this form of storage requires a supplementary effort by the developers.

A work describing the implementation and performance evaluation of a PBNM framework, using COPS in outsourcing model with RSVP (COPS-RSVP) was presented by Ponnappan [8]. The QoS policies were represented using QPIM (QoS Policy Information model), an IETF PCIM extension described by Snir [10]. The policies were represented and stored using LDAP .This work uses CORBA (Common Object Request Broker Architecture) for supporting the interaction between the application components.

4. XACML Review

The XACML (eXtensible Access Control Markup Language) is an OASIS proposal for modeling, storing and distributing descriptive access control policies [7]. XACML-based frameworks are supposed to be implemented using the PDP/PEP architecture in the outsourcing model. The XACML language is defined by two XML schemes: "*xacml context*" and "*xacml policy*". The "*xacml context*" defines how to represent policy request and policy response messages exchanged between the PEP and the PDP. The "*xacml policy*" defines how to represent the access control policies. Figure 1 shows the UML diagram of the "*xacml policy*" scheme. The figure represents the classes and associations between XACML elements, but omits its attributes. According to the XACML strategy, a policy is described in terms of a

set of access permissions (or access denials) by structures denominated Targets. A Target is expressed through the syntax: "users (Subject class) can (or cannot) apply actions (Action class) upon resources (Resource class)".

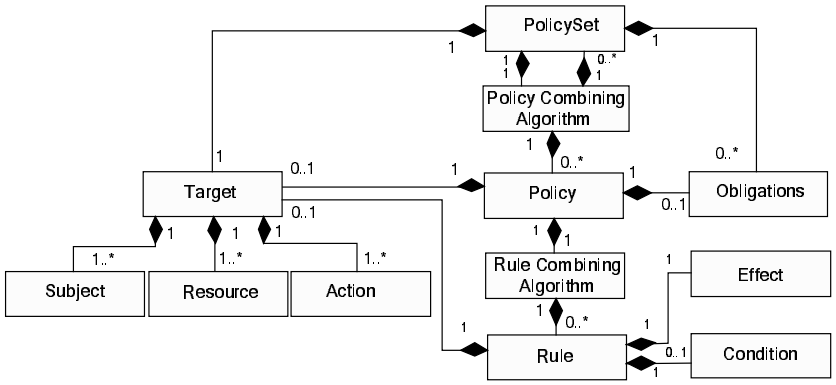

Figure 1. XACML policy scheme

Targets can be associated to a policy, to a policy set or to a rule. Targets associated to a policy or a policy set work as policy selectors, i.e., when a PEP request a decision concerning a Target, only the policies and policies sets that contain the Target elements need to be evaluated. Targets associated to rules permit to express conditional permissions (or denials). A rule is expressed by the syntax: "if the condition (Condition class) is satisfied then applies the effect (Effect class) upon the Target". The possible values for effect are: permit or deny. The effect defines the real sense of a Target as a permission or denial. Figure 2 shows a simple policy example to illustrate the use of the XACML classes. The policy represented in the figure can be described textually as follows: "the user **ana@xacml.org** can **login** on a **Multimedia Server** in the period between **08:00AM** and **17:00PM**".

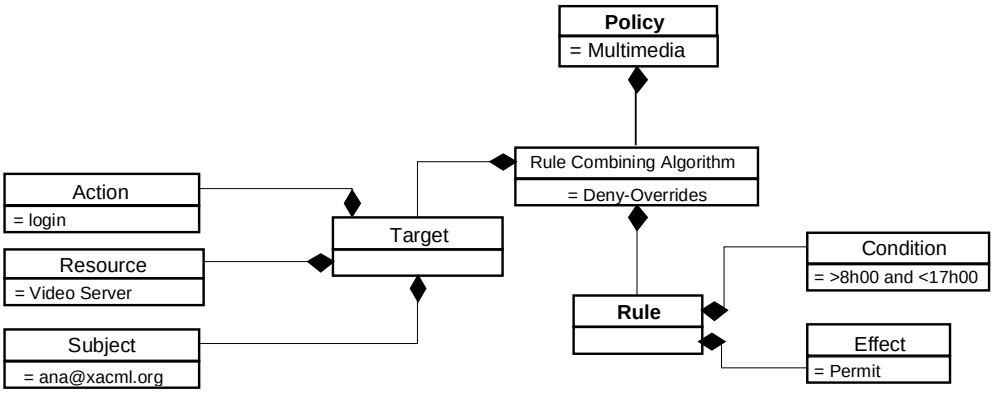

Figure 2. XACML policy example

When a PEP sends a request to the PDP, it supplies the attributes permitting to identify the elements of a Target (Subject, Resource, Action). The PDP evaluates the policy rules and determines if exists a Target with those attributes, and then returns to the PEP the corresponding effect: *Permit* or *Deny*. If it fails to find a Target in its policies that satisfy the attributes supplied by the PEP, it will return "*NotApplicable*".

The Obligations class, when defined, is returned to the PEP in conjunction with the decision. The Obligations class is supposed to inform a set of actions that must be performed by the PEP, concerning the decision. The XACML version (1.0) used in our study [7] does not specify the type of actions described in Obligations. The specification only defines the PEP must be capable of interpreting any information passed through the Obligations class. As will be explained further, our proposal uses the Obligations class to pass QoS parameters to a RSVP node.

As shown in figure 1, a XACML policy can include several rules. The "Rule Combining Algorithm" class determines the strategy used to evaluate the set of rules associated to the same policy. The following strategies are defined by the XACML: Deny-overrides; Permit-overrides and the First-applicable. In Deny-overrides, if the conditions of a rule with effect "*Deny"* are satisfied, then the decision for the policy will be to deny, regardless the other rules; Permit-overrides defines a similar approach for the effect "*Allow"*. In First-applicable, the first rule satisfied defines the effect of the policy. Observe, also in figure 1, that policies can be aggregated through the class *PolicySet*. Similarly as to the rules, the policies are interpreted according to the class "Policy Combining Algorithm", which defines the same strategies utilized to combine rules, adding still the only-one-applicable strategy. In this case, if more than one policy is satisfied within a *PolicySet*, then the result of the policy set evaluation will be "*Indeterminate*". In addition, XACML defines that developers can also add their own strategies for policy and rule combining.

Figure 3 illustrates how the UML model shown in figure 2 is represented in a XML document. The XML document "format" is formally described by the "*xacml policy*" scheme.

```
<Policy PolicyId=" " RuleCombiningAlgId=" ">
  <Target>
      <Subjects>...</Subjects>
      <Resources>...</Resources>
      <Actions>...</Actions>
  </Target>
  <Rule RuleId=" " Effect=" ">
      <Target>...</Target>
      <Condition FunctionId=" ">...</Condition>
  </Rule>
  <Obligations>
      <Obligation ObligationId=" " FulfillOn=" "> </Obligation>
  </Obligations>
</Policy>

<!-- The elements of a Target: Subjects, Resources and Actions,  are defined by
attributes included within the corresponding <TAGS> -->

<!— In Obligations, the attribute FulfillOn indicates if the obligation must be
executed when the resulting effect is Permit or Deny -->
```

Figure 3. A XACML Policy document

Though the Obligations class offers an alternative for implementing some sort of policy "provisioning", we observe that XACML is primarily supposed to be implemented using the outsourcing approach, because the PDP basically returns decisions of type "*Permit*" or "*Deny*" to the PEPs. As it will be explained in the next section, the Obligations approach, as defined in XACML version 1.0, is rather limited, because the XACML framework offers no facilities for pre-processing Obligations before returning them to the PEPs. Other limitations of the present XACML specifications concern the lack of definitions regarding the communication protocol for supporting the exchange of messages between the PDP and the PEPs, as well as definitions about the strategy for storing the XACML documents that represent the network policies.

5. Proposal

This paper proposes a XACML-based framework for distributing and enforcing access control policies to RSVP-aware application servers. Figure 4 illustrates a typical scenario for this framework. The PEP element represents a component of the server application, responsible for requesting policy decisions to the PDP and interacting with the RSVP daemon in the host computer. The code of the PEP must be integrated with the application server, as explained in section 6. In our proposal, the PEP is responsible for all interaction with the RSVP daemon, releasing the application from the task of any QoS negotiation. This interaction includes retrieving the traffic information for building PATH messages and granting or not the reservation request on receiving the RESV message. This approach can be implemented in any system that supports the RSVP APIs described in the RFC 2205.

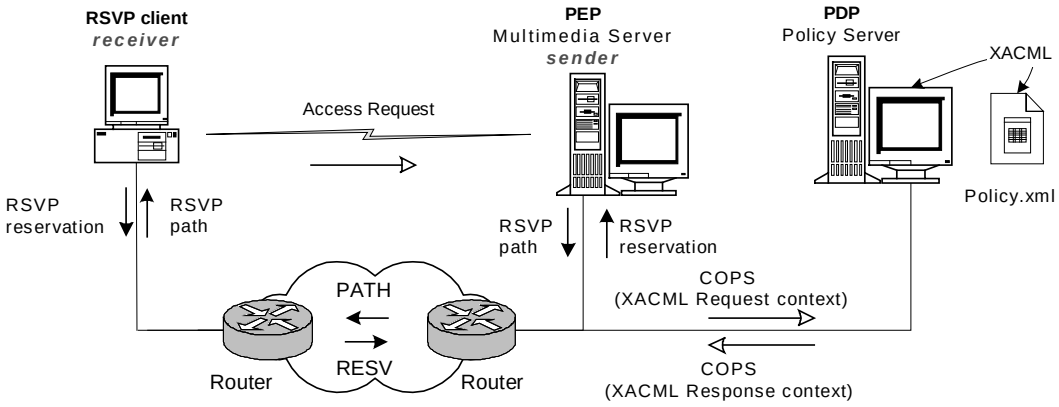

Figure 4. Policy control of RSVP with XACML

The sequence of events and messages exchanged by the elements in figure 4 during the establishment of a RSVP reservation, using the proposed framework, is described as follows:

1. A RSVP client requests a connection to a multimedia server for obtaining services with QoS.
2. In the multimedia server, the application calls the PEP for evaluating the request. Then, the PEP sends to the PDP a XACML request context message informing a "Target" containing its IP address (Resource), the IP address of the client (Subject) and the requested operation (Action).
3. The PDP evaluates the policy defined in XACML for the supplied target, and returns to the PEP a XACML response context message having, besides the result (permit or deny), the information of traffic specification (*Tspec*, supplied through the Obligations structure).
4. In case of positive decision, the PEP calls its RSVP daemon, informing the *Tspec* parameters. The RSVP daemon, then, sends a RSVP PATH message to the receiver (i.e., the RSVP client). The *Tspec* parameters are stored in the PEP for further analysis (see step 6).
5. The RSVP client, on receiving a RSVP PATH message, calls its RSVP daemon, which obtains the traffic parameters from the PATH message and formats a RESV RSVP message, returning it to the sender (i.e., the PEP).
6. On receiving the RESV message from the client, the RSVP daemon of the server triggers an event to the PEP forwarding the *Tspec* information. The PEP compares the *Tspec* information received from the client with the *Tspec* information saved in step 4. If the *Tspec* parameters are identical or smaller than those saved in step 4, the PEP confirms the reservation to the RSVP daemon. In this step, the RSVP daemon also verifies if it has enough resources to satisfy the request (admission control).

The steps 1 to 6 refer to a well-succeeded scenario of reservation, and exception treatment was omitted. A RSVP access solicitation differs from a conventional access solicitation (e.g., access to a file or directory) because the PDP needs to return the information necessary for the PEP building the PATH message. For this reason, extensions to XACML language were required in order to accommodate the transport of QoS information. As the complete description of the extensions in the XACML policy and context schemes is rather extensive, this paper will describe only the elements needed for understanding the main aspects of our proposal. For a complete description of work, please refer to [11].

In the XACML policy scheme, the *Resource* class was extended and called *ResourceRsvp*. The extended class accommodates the description of RSVP parameters required for building the PATH message, i.e., *Tspec* {r,b,p,m,M}, type of service (GS – guaranteed service or controlled load – CL) and reservation style as described in the RFC 2210 [13] and RFC 2215 [9]. Figure 5 illustrates the XACML scheme extension.

```
<xs:element    name=”ResourceRsvp”  type=”xacml:ResourceRsvpType”/>
  <xs:complexType name="ResourceRsvpType">
    <xs:sequence>
       <xs:element ref="xacml:TspecBucketRate_r"/>  <!-- Tspec {r,b,p,m,M} -->
       <xs:element ref="xacml:TspecBucketSize_b"/>
       <xs:element ref="xacml:TspecPeakRate_p"/>
       <xs:element ref="xacml:TspecMinPoliceUnit_m"/>
       <xs:element ref="xacml:TspecMaxPacketSize_M"/>
       <xs:choice minOccurs="0" maxOccurs="unbounded">
       <xs:element ref="xacml:RsvpService"/> <!-- CL, GS, Null -->
       <xs:element ref="xacml:RsvpStyle"/>  <!-- SE, WF, FF -->
    </xs:choice>
  </xs:sequence>
 <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
 <xs:attribute name="RsvpClass"  type="xacml:RsvpClassType" use="required"/>
</xs:complexType>
```

Figure 5. XACML RSVP Scheme Extension

The *ResourceRsvp* class was defined as an optional element inside *Resource*, and it can be declared more than once. Several occurrences of *ResourceRsvp* objects for the same *Resource* permit to describe several modes a service can be offered by a given application. For example, a multimedia server can define various QoS modes for streaming video in order to support different resolutions. In this case, each QoS mode must receive a distinct class specification (attribute *RsvpClass*). Observe in figure 5, that the RSVP policy scheme does not include the *Rspec* parameters. In this work, we suggest the PEP could reject the proposal received on the RESV message if the *Rspec* parameters are much larger than those specified by *Tspec*, not being necessary to consult the PDP again for validating the RESV message.

Figure 6 shows an example of RSVP policy. The main elements were highlighted, and most attributes and references were suppressed. The policy describes the access to a resource called of “*Multimedia Server*”, with a QoS defined by the element <*ResourceRsvp*>. A rule is used for restricting the access to the server for a limited range of IP addresses and a specific period of time. An action named *getResourceQoS* was used for identifying the operation requested by the client. The policy also specifies the element <*Obligations*> that will be returned to the PEP with the *Permit* or *Deny* decision.

```
<Policy PolicyId="MultimidiaPolicy">
 <Target>
  <Subjects> <AnySubject/>  </Subjects>
  <Actions> <AnyAction/> </Actions>
  <Resources>
   <Resource>
    <ResourceMatch MatchId="string-equal">
     <AttributeValue>MultimediaServer</AttributeValue>
     <ResourceAttributeDesignator AttributeId=" resource-id"/>
    </ResourceMatch>
    <ResourceRsvp  AttributeId=" multimediaserver" RsvpClass="G711">
     <TspecBucketRate_r>9250</TspecBucketRate_r>
     <TspecBucketSize_b>680</TspecBucketSize_b>
     <TspecPeakRate_p>13875</TspecPeakRate_p>
     <TspecMinPoliceUnit_m>340</TspecMinPoliceUnit_m>
     <TspecMaxPacketSize_M>340</TspecMaxPacketSize_M>
     <RsvpService>Guaranteed</RsvpService>
    </ResourceRsvp>
   </Resource>
  </Resources>
 </Target>
<Rule RuleId=" ResourceQoS" Effect="Permit">
 <Target>
   <Subjects> <!-- … receiver=192.168.200.0/24  …sender=192.168.200.1 …  --> </Subjects>
  <Resources> <AnyResource/> </Resources>
  <Actions> <!-- … getResourceQoS → </Actions>
 </Target>
 <Condition> <!-- … schedule restrictions … → </Condition>
</Rule>
<Obligations>
 <Obligation FulfillOn="Permit" ObligationId=”G711”>
  <AttributeAssignment AttributeId="TspecBucketRate_r">
   <AttributeSelectorRsvp PolicyPath="ResourceRsvp[@RsvpClass=’G711´]/TspecBucketRate_r/text()">
   </AttributeSelectorRsvp>
  </AttributeAssignment>
  <!-- … this structure repeats for: TspecBucketSize_b,TspecPeakRate_p,
       TspecMinPoliceUnit_m, TspecMaxPacketSize_M and RsvpService … -->
   </Obligation>
  </Obligations>
</Policy>
```

Figure 6. Example of RSVP policy

In our work, the <*Obligations*> structure is used for supplying the *Tspec* parameters to the server application. This utilization of <Obligations> is a proposal of our work, once XACML does not specify this type of action. The <*AttributeSelectorRSVP*> element was also introduced in our proposal in order to allow the <Obligations> structure to make references to the traffic information already defined in the <Target> by <ResourceRSVP>. When the PEP sends a request for decision to the PDP (i.e., a *xacml Request context* message), it specifies a <Target> with the Subject, Resource and Action elements, as show in figure 7.

```
<Request ...>
   <Subject>
     <Attribute AttributeId="subject:authn-locality:ip-address:receiver">
      <AttributeValue>IP_Address_RECEIVER</AttributeValue>
     </Attribute>
     <Attribute AttributeId="subject:authn-locality:ip-address:sender">
      <AttributeValue>IP_Address_SENDER</AttributeValue>
     </Attribute>
   </Subject>
   <Resource>
     <Attribute AttributeId="resource:resource-id">
      <AttributeValue>MultimediaServer</AttributeValue>
     </Attribute>
   </Resource>
   <Action>
     <Attribute AttributeId="action:action-id:ServerAction">
      <AttributeValue>getResourceQoS</AttributeValue>
     </Attribute>
   </Action>
</Request>
```

Figure 7. Example of RSVP policy request

Figure 8 illustrates the answer returned from the PDP to the PEP. In the example, the policy defines only one mode of operation for the multimedia server (defined by the *RsvpClass* attribute with value G711, in the <*ResourceRsvp*> element). In case of multiples operation modes, all *Tspec* definitions supported by the multimedia server would be returned to the PEP through the structure <*Obligations*>. It is up to the PEP the responsibility of choosing the operation mode for the client. This approach was adopted because XACML specification does not define any mechanism for pre-processing the <*Obligations*> structure before returning it to the PEP (i.e., there is no way of returning only a part of the <Obligations> structure). This limitation can be observed in the UML class model in the figure 1 that shows how the <*Obligations*> instances are associated to a policy.

```
<Response …>
 <Result>
  <Decision>Permit</Decision>
   <Status>
    <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
   </Status>
   <Obligations xmlns="urn:oasis:names:tc:xacml:1.0:policy">
    <Obligation ObligationId="obligation:MultimidiaServer"  FulfillOn="Permit">
     <AttributeAssignment AttributeId="TspecBucketRate_r">9250</AttributeAssignment>
     <AttributeAssignment AttributeId="TspecBucketSize_b">680</AttributeAssignment>
     <AttributeAssignment AttributeId="TspecPeakRate_p">13875</AttributeAssignment>
     <AttributeAssignment AttributeId="TspecMinPoliceUnit_m">340</AttributeAssignment>
     <AttributeAssignment AttributeId="TspecMaxPacketSize_M">340</AttributeAssignment>
     <AttributeAssignment AttributeId="RsvpService"> Guaranteed </AttributeAssignment>
    </Obligation>
   </Obligations>
  </Result>
</Response>
```

Figure 8. Example of RSVP policy response

7. Implementation

On important advantage of the XACML approach with respect to PCIM refers to its implementation. Because it is defined in terms of XML, a XACML implementation benefits from the existing tools for developing XML applications. There are free packages for supporting XACML in Java language (Sun XACML project) and on C++ (by Jiffy Software).

The framework described in this paper was implemented using the Java™ 2 SDK, Standard Edition 1.4.2, and the Sun XACML package. The Sun XACML package includes the modules: "*com.sun.xacml*. *PolicySchema*" and "*com.sun.xacml.ContextSchema"*. The first module supports the interpretation of XACML policies (required for implemented a PDP) and the second, the exchange of messages between the PDP and the PEP.

The implementation permitted to evaluate if the proposed XACML schema extensions are compatible with existing implementation packages. We observed that it was not necessary to modify the package code, except in the case of treatment of the <*Obligations*>structure. The scheme developed in this work, denominated "cs-xacml-schema-policy-01-rsvp.xsd", is described in details in Toktar [11]. The possibility of extending the XACML schemas and even then, reusing existing development packages is an important advantage in the OASIS approach. The packet significantly simplifies the process of developing a PDP and embedding PEPs in existent applications.

Next, one presents some examples of utilization of the Sun XACML package for developing a PDP. The following code fragment illustrates the sequence of steps for creating a PDP instance, initialized with a policies file defined by "PolicyQoS.xml". The "policyModule.addPolicy" method permits to validate the policy with respect to the XACML policy schema. This method was used for validating the syntax of the schema extensions proposed in this work.

```
FilePolicyModule policyModule = new FilePolicyModule();
policyModule.addPolicy("Path/PolicyQoS.xml");
```

The XACML package offers classes that, through the Hash tables, simplify the process of searching policies (PolicyFinder) and attributes (AttributeFinder). The fragment of typical code for the creation of an instance of PDP is illustrated following.

```
PolicyFinder polFinder = new PolicyFinder();
Seth policyModules = new HashSet();
policyModules.add(policyModule);
policyFinder.setModules(policyModules);
AttributeFinder attrFinder = new AttributeFinder();
List attrModules = new ArrayList();
attrFinder.setModules(attrModules);
PDP pdp = new PDP(new PDPConfig(attrFinder, polFinder, null));
```

The next fragment of code illustrates the creation of a PEP. The RequestCtx class implements a PEP requests to a PDP. The attributes passed in the class constructor refers to the Target elements <Subject>, <Resource> and <Action>. The Environment attributed is used for passing other relevant information, concerning time, for example.

```
RequestCtx request = new RequestCtx(AttribSubjects, AttribResource, AttribAction,
AttribEnvironment);
```

The *ResponseCtx* class is used for receiving the PDP response. A *ResponseCtx* object encapsulates the decision, status code and the <*Obligations*> structure. The code fragment is presented next:

```
ResponseCtx response = pdp.evaluate(request);
```

8. Conclusion

This paper evaluated the utilization of the XACML language for describing RSVP access control policies. The XACML is still under development and, although limited in a few aspects by lack of standardization, it can be considered a flexible model for the description of the control access policies in different application domains.

In this work, XACML use was extended beyond the access control functionalities, because the decisions generated by the PDP include the *Tspec* parameters necessary for building the PATH messages. The capacity of returning configuration parameters through PDP decisions is an important feature for many PBNM scenarios. This feature, easily supported in IETF PCIM-based models, is quite difficult to implement in XACML. To support the RSVP scenario, modifications in the <Obligations> structure were required, including some features not supported by the XACML Sun package. We conclude that the XACML model is deficient in returning results that are not simple deny or permit decisions. Further specifications of the <*Obligations*>, as well a more flexible way of mapping conditional <Obligations> to policies, are suggested developments for the XACML model.

Our work requires some additional specifications concerning the use of the <*Obligations*> structure for representing multiples operation modes supported by the same application (i.e., distinct *Tspec*). Other future work consists in extending the XACML model and existing packages for supporting the provisioning model. The provisioning model is a promising approach for extending the proposed framework for configuring RSVP policies in network devices (e.g. routers), once the present approach is restricted to application servers. In order to support the provisioning model, as defined by IETF, several extensions are required. First, the XACML model must be extended in order to provide the elements for mapping the policies to "interface roles". Second, one must define the algorithms for interpreting and translating the XACML-policy information to a PIB. Third, because next-generation of policy-aware network devices are supposed to understand COPS-PR, in the provisioning model, the policy information should be directly encapsulated in the COPS-PR protocol, rather than being transported as "XACML-context messages", as defined for the outsourcing approach.

10. References

[1] Boyle, J.; Cohen, R.; Durham, D.; Herzog, S.; Rajan, R.; Sastry, A. The COPS (Common Open Policy Service) Protocol, RFC2748, Jan. 2000.

[2] Braden, R.; Zhang, L.; Berson, S.; Herzog, S.; Jamin, S. Resource Reservation Protocol (RSVP) Version 1 Functional Specification, RFC2205, Sep. 1997.

[3] Chan K.; Seligson, J.; Durham, D.; Gai, S.; McCloghrie, K.; Herzog, S.; Reichmeyer, F.; Yavatkar, R.; Smith, A. COPS Usage for Policy Provisioning (COPS-PR), RFC3084, Mar. 2001.

[4] Herzog, S.; Rajan, R.; Sastry, A. COPS usage for RSVP, RFC2749, Jan. 2000.

[5] Moore, B.; Ellesson, E.; Strassner, J.; Westerinen, A. Policy Core Information Model - Version 1 Specification, RFC3060, Feb. 2001.

[6] Nabhen, R., Jamhour, E., Maziero C. "Policy-Based Framework for RBAC", Proceedings for the fourteenth IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, October, Germany, Feb. 2003.

[7] OASIS, eXtensible Access Control Markup Language (XACML) Version 1.0. OASIS, Feb. 2003.

[8]Ponnappan, A.; Yang, L.; Pillai, R.; Braun, P. "A Policy Based QoS Management System for the IntServ/DiffServ Based Internet". Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks (POLICY.02). IEEE, 2002 .

[9] Shenker, S.; Wroclawski, J. General Characteri-zation Parameters for Integrated Service Network Elements, RFC 2215, Sep. 1997.

[10] Snir, Y.; Ramberg, Y.; Strassner, J.; Cohen, R. "Policy QoS Information Model, work in progress, draft-ietf-policy-qos-info-model-05.txt". IETF, May. 2003.

[11] Toktar, E. Controle de Admissão de RSVP utilizando XACML. Dissertação de Mestrado, PPGIA, PUCPR. Aug. 2003.

[12] Westerinen, A. et. al. Terminology for Policy Based Management. RFC3198, Nov. 2001.

[13] Wroclawski, J. RSVP with INTSERV, RFC 2210, Sep. 1997.

[14] Yavatkar, R., Pendarakis, D.; Guerin, R. A Framework for Policy-Based Admission Control, RFC2753, Jan. 2000.