RSVP Policy Control using XACML

Emir Toktar, Edgard 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 IETF PCIM-based approach. Both approaches are compared in this paper.

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 should 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. 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, i.e., the policy control is implemented only by the application server. However, this proposal also supply the information for defining the *Tspec* and *Rspec* parameters transported in the PATH and RESV messages. Therefore, the XACML policy also provides the information 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 elements (nodes) 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 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. Fig. 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)".

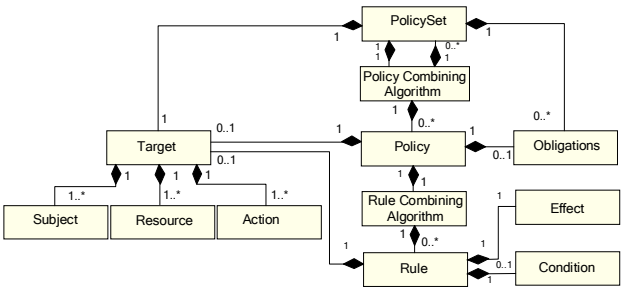

Figure 1. XACML policy scheme

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.

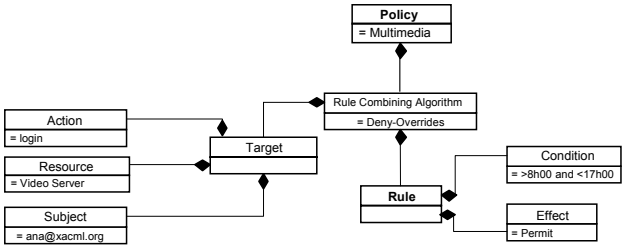

Figure 2. XACML policy example

Fig. 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 **Video Server** in the period between **08:00AM** and **05:00PM**".

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.

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, but the "concept" is flexible enough for providing "configuration information" to network nodes in several domains. 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.

```
<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>

<!— 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

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

5. Proposal

This paper proposes a XACML-based framework for distributing and enforcing access control policies to RSVP-aware application servers. Fig. 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.

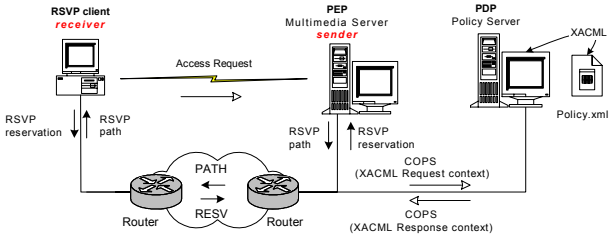

Figure 4. Policy control of RSVP with XACML

The sequence of events and messages exchanged by the elements in Fig. 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 the XACML framework features were required in order to describe and transport the QoS information.

The strategy adopted in this work for describing a RSVP policy in terms of XACML is illustrated in Figure 5.

```
<PolicySet PolicySetId="RSVP_Aware_Server_Application">
   <Target> <!—Defines the services (resources) to which the policy applies →
   </Target>
   <Policy PolicyId="Service Level 1"> <!—e.g. GOLD →
      <Rule>
         <Target> <!—Subjects to which the policy applies → </Target>
         <Condition> <!-- Time and client's IP addresses restrictions -->
         </Condition>
      </Rule>
      <Obligations> <!—TSpec specification for service level 1 →
      </Obligations>
   </Policy>
   <Policy PolicyId="Service Level 2"> … </Policy> <!—e.g. SILVER →
   <Policy PolicyId="Service Level N"> … </Policy> <!—e.g. BRONZE →
   <Policy PolicyId="Default Policy"> <!—usually denies all → </Policy>
</PolicySet>
```

Figure 5. RSVP XACML Policy Structure

In the proposed strategy each "RSVP-aware" server application (or group of applications) is mapped to a XACML <PolicySet>. Server applications can be described by the same policy set only if they offer the same "QoS Service Levels" for the same set of users, under the same restrictions. For example, distinct video streaming servers in a university campus that offer a "GOLD" service for registered students and "SILVER" service for visitors (with the same Tspec definitions) can be represented by a single policy set. The policies are mapped to services through the <Target> element in the <PolicySet> structure (see the example in section 6). The <Policy> elements in the <PolicySet> are used for defining distinct QoS service levels offered by the same application. For example, "GOLD", "SILVER", etc. The <Rule> defines the users (subjects) that have authorization to receive the service level and the <Obligations> element describes the *Tspec* parameters.

The reason for defining a RSVP policy in terms of a <PolicySet> and not in terms of a single <Policy> element is related to the XACML definition. One observes in Figure 1 that the <Obligations> element is mapped to <Policy> or <PolicySet>, but it can't be mapped to Rules, i.e., all <Rules> in a policy defines the same <Obligations>. Therefore, distinct service levels can't be represented in a single policy.

Another important point is to define where the users and services information is located. If we consider the PCIM approach, defined by IETF, a logical approach would consist in representing users and services through CIM[1] objects. Because CIM is supposed to be supported by an important set of hardware and software vendors, it is an interesting choice for sharing the same information among heterogeneous systems. Both CIM and PCIM information can be stored in LDAP servers.

The XACML definition permits to define all the information concerning the policy (subjects, resources and actions) in the same XML document, as defined by the "*xacml policy scheme*". However, OASIS points that it will be possible to write XACML policies that refers to information elements stored in a LDAP repository. The 1.0 specification does not define how it can be done. However, because *xacml* is based on standard xml definitions, a possible solution would be create references in a policy to external documents using the *XML Pointer Language (XPointer)* strategy [15]. There are some references about the use of *XPointer* in the 1.0 OASIS specification, however, its use is limited to request documents (i.e., context scheme) and its use in policy documents is not supported. However, using *XPointer* for creating policies with reusable subjects and services information is a logical extension for future XACML versions.

Hence, this work adopts the use of XPointer for defining policies with reusable subjects (users) and services information. In the future, this approach can be replaced by the LDAP approach without modifying the policy strategy.

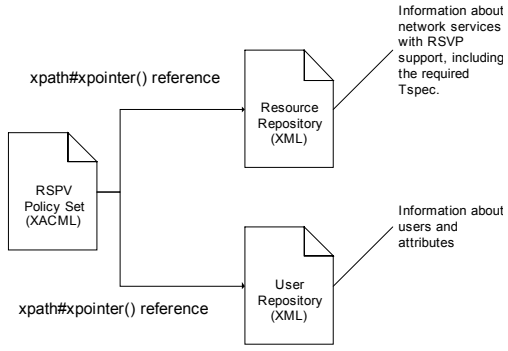

Figure 6. Policy, Resources and Users documents.

[1] CIM (Common Information Model), proposed by the DMTF (Distributed Management Task Force) is a information model compatible with PCIM that defines classes and associations for representing users, network elements and servies.

Figure 6 illustrates the relationship between the XML documents used for describing a policy. An example of how these documents can be defined are presented in case study in the next section (see Figures 8 and 9). The structure of the resource and user documents described in this work was chosen intentionally simple for didactical purposes. The QoS information is described in the resource document. A resource is defined as a network service that supports RSVP negotiations. Hence, the resource document 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].

Fig. 7 illustrates the XML scheme corresponding to the RSVP parameters. Our proposal assumes that a single service can offer different service levels. 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*). The class enumerated in the scheme are define by ITU-T, and are included in the scheme for illustration purposes only. Observe in Fig. 7, that the RSVP resource 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.

```
<xs:schema>
  <xs:element name="ResourceRsvp" type="xacml:ResourceRsvpType"/>
  <xs:complexType name="ResourceRsvpType">
    <xs:sequence>
      <xs:element ref="xacml:TspecBucketRate_r"/>
      <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="1">
        <xs:element ref="xacml:RsvpStyle"/>
      </xs:choice>
      <xs:choice minOccurs="0" maxOccurs="1">
        <xs:element ref="xacml:RsvpService"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
    <xs:attribute name="RsvpClass" type="xacml:RsvpClassType"
  use="required"/>
  </xs:complexType>
  <xs:simpleType name="RsvpClassType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="G711"/>
      <xs:enumeration value="G729"/>
      <xs:enumeration value="H263CIF"/>
      <xs:enumeration value="H261QCIF"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="RsvpService">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="Null"/>
        <xs:enumeration value="Guaranteed"/>
        <xs:enumeration value="Controlled-load"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="RsvpStyle">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="SE"/>
        <xs:enumeration value="WF"/>
        <xs:enumeration value="FF"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="TspecBucketRate_r">
    <xs:simpleType>
      <xs:restriction base="xs:double">
        <xs:minInclusive value="1"/>
        <xs:maxInclusive value="40000000000000"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <!—definitions for other elements: TspecBucketSize_b , etc-->
  </xs:schema>
```

Figure 7. Tspec Scheme Definition

6. Case Study and Implementation

6.1. Case Study

In order to illustrated the use of the XACML approach for describing RSVP policies, the following scenario was considered: A set of "video streaming" servers in a university campus offers "tutorials" to registered and unregistered students (visitors). The policy adopted for having access to the video streaming is defined as follows:

a) Registered students have permission to access any server in the campus offering a "*TutorialVideoStreaming*" service without time restrictions. If a student connects to a server using a client host from inside the campus, he will receive a "GOLD" or "SILVER" service level. Otherwise, it will receive a "BRONZE" service level.

b)Unregistered students can have access to the "*TutorialVideoStreaming*" service only from the internal network and not in business-time. They can receive only the "BRONZE" service level.

```
<service serviceId="TutorialVideoStreaming">
   <description> tutorial videos in the campus university
   </description>
   <sap>
      <inetaddress> 192.168.200.10 </inetaddress>
      <inetaddress >192.168.5.3 </ inetaddress >
      <protocol>tcp</protocol>
      <port>8976</port>
   </sap>
   <serviceLevel serviceId="Gold">
      <ResourceRsvp AttributeId="qosG711" 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>
         <RsvpStyle>FF</RsvpStyle>
      </ResourceRsvp>
   </serviceLevel>
   <serviceLevel serviceId="Silver">
      <ResourceRsvp AttributeId="qosH261Q"
RsvpClass="H261QCIF">
         <TspecBucketRate_r>12000</TspecBucketRate_r>
         <TspecBucketSize_b>6000</TspecBucketSize_b>
         <TspecPeakRate_p>12000</TspecPeakRate_p>
         <TspecMinPoliceUnit_m>80</TspecMinPoliceUnit_m>
         <TspecMaxPacketSize_M>2500</TspecMaxPacketSize_M>
         <RsvpService>Controlled-load</RsvpService>
         <RsvpStyle>SE</RsvpStyle>
      </ResourceRsvp>
   </serviceLevel>
   <serviceLevel serviceId="Bronze">
      <ResourceRsvp AttributeId="qosH263C"
RsvpClass="H263CIF">
         <TspecBucketRate_r>16000</TspecBucketRate_r>
         <TspecBucketSize_b>8192</TspecBucketSize_b>
         <TspecPeakRate_p>16000</TspecPeakRate_p>
         <TspecMinPoliceUnit_m>80</TspecMinPoliceUnit_m>
         <TspecMaxPacketSize_M>8192</TspecMaxPacketSize_M>
         <RsvpService>Controlled-load</RsvpService>
         <RsvpStyle>WF</RsvpStyle>
      </ResourceRsvp>
   </serviceLevel>
</service>
```

Figure 8. Service Information

The service information is represented in the document illustrated in Figure 8. Note that the <SAP> structure defines the services in the campus that are subjected to the policy. The *Tspec* information concerning the <GOLD>, <SILVER> and <BRONZE> service levels are also defined in the file.

A XACML request from the video server (i.e., a PEP) will usually identify the user by its login (uid). However, the policy in the PDP will be described in terms of the student status (registered or unregistered). The mapping between the user id and the corresponding student status is represented by the XML document illustrated in Figure 9.

```
<subjects>
   <user>
      <cn>Emir Toktar</cn>
      <uid>etoktar</uid>
      <mail>toktar@ppgia.pucpr.br</mail>
      <businessCategory>RegisteredStudent</businessCategory>
   </user>
   <user>
      <cn>Luis Cezar</cn>
      <uid>lcezar</uid>
      <mail>ortega@ppgia.pucpr.br</mail>
      <businessCategory>RegisteredStudent</businessCategory>
   </user>
   <user>
      <cn>guest</cn>
      <uid>guest</uid>
      <businessCategory>UnregisteredStudent</businessCategory>
   </user>
</subjects>
```

Figure 9. User Information

The RSVP policy is defined in terms of a XACML <PolicySet>, as described in section 5. The <PolicySet> structure for the case study defines four policies, as shown in Figure 10. The Target structure in the <PolicySet> defines the resources to which the policy applies. Note <ResouceMatch> elements in the <Target> structure defined conditions that compares the information supplied by the PEP in the Request message (resource-id, and ip-address:sender) with the information described in the XML Service Information File (see Figure 8) .

Figure 11 illustrates the structure of "Policy 1" in the <PoliceSet>. This policy defines the conditions applied to the access of registered students from inside the campus. The <Subject> element in the <Rule> defines that the policy applies only to registered students. The <Condition> of the rule determines that the policy applies only to requests where the client host is located inside the university campus.

A typical policy request from a PEP to the PDP is illustrated in Figure 12. The <Subject> element will supply the information about the receiver, i.e., user id and IP address of its host. The <Resource> element supplies the information about the server (i.e., the sender), including its name (resource id) and IP address. The type

of action requested is defined as “getResorceQoS”, in this case, the only action supported by the policy.

```
<PolicySet PolicySetId="TutorialVideo"
PolicyCombiningAlgId=":policy-combining-algorithm:first-
applicable">
    <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId=":function:string-equal">
          <AttributeValue DataType="#string">
            TutorialVideo</AttributeValue>
            <ResourceAttributeDesignator DataType="#string"
            AttributeId=":resource:resource-id"/>
        </ResourceMatch>
        <ResourceMatch MatchId=":function:xpath-node-match">
          <AttributeValue DataType="#string">
http://pdp/resources.xml#xpointer(//service[@serviceId=”TutorialVide
oStreaming”]/sap/inetaddress/text())
          </AttributeValue>
          <ResourceAttributeDesignator DataType="#string"
AttributeId=":resource:authn-locality:ip-address:sender"/>
        </ResourceMatch>
      </Resource>
    </Resources>
  </Target>
  <!-- Policy 1: Registered Students from inside the campus -->
  <Policy PolicyId=":policy:TutorialRegStudentsInternal"
RuleCombiningAlgId=":rule-combining-algorithm:first-applicable">
  </Policy>
  <!-- Policy 02: Registered Studens from outside the campus -->
  <Policy PolicyId=":policy:TutorialRegStudentsExternal"
RuleCombiningAlgId=":rule-combining-algorithm:first-applicable">
  </Policy>
  <!-- Policy 03: Unregiestered Students -->
  <Policy PolicyId=":policy:TutorialRegStudentsGuest"
RuleCombiningAlgId=":rule-combining-algorithm:first-applicable">
  <!-- Policy 04 - Deny for All -->
  <Policy PolicyId=":policy:TutorialDenyForOthers"
RuleCombiningAlgId=":rule-combining-algorithm:first-applicable">
    <Rule RuleId=":Tutorial_Deny_Rule_For_Others"
Effect="Deny"/>
  </Policy>
</PolicySet>
```

Figure 10. Policy Set Structure

Finally, Figure 13 illustrates the response from the PDP to the PEP. The “Permit” information informs to the PDP that there are services to be offered to the client. The services are described in the <Obligations> structure. In this example, two *Tspec* specifications are returned to the PEP. These specification correspond to the service level “GOLD” and “SILVER” offered by the VideoStreaming server. An alternate approach could be return only the highest service level. The structure presented in the <Obligations> section is defined by the XACML context-schema.

```
<Policy PolicyId=":policy:TutorialRegStudentsInternal"
RuleCombiningAlgId=":rule-combining-algorithm:first-applicable">
<Rule RuleId="Reg_Students_Internal_Get_Gold_Silver" Effect="Permit">
 <Target>
  <Subjects><Subject>
    <SubjectMatch MatchId=":function:xpath-node-match">
     <!-- return of a Bag attributes of elements 'uid' that are 'RegisteredStudent' -->
     <AttributeValue DataType="#string">
       http://pdp/subjects.xml#xpointer(//subjects/user
       [businessCategory='RegisteredStudent']/uid/text())
     </AttributeValue>
     <SubjectAttributeDesignator AttributeId=":subject:subject-id"
       DataType="#string"/>
     </SubjectMatch>
  </Subject></Subjects>
  <Actions><Action>
    <ActionMatch MatchId=":function:string-equal">
     <AttributeValue DataType="#string">getResourceQoS</AttributeValue>
     <ActionAttributeDesignator DataType="#string"
     AttributeId=":action:action-id:ServerAction"/>
    </ActionMatch>
  </Action></Actions>
 </Target>
 <Condition FunctionId=":function:or">
  <Apply FunctionId=":function:any-of">
   <Function FunctionId=":function:regexp-string-match"/>
   <AttributeValue DataType="#string">192.168.0.*</AttributeValue>
    <SubjectAttributeDesignator
      AttributeId=":subject:authn-locality:ip-address:receiver"
      DataType="#string"/>
  </Apply>
 </Condition>
</Rule>
<Obligations>
 <Obligation ObligationId="GoldSilverStudentsInternal" FulfillOn="Permit">
  <AttributeAssignment AttributeId="qosG711" DataType="#string">
     http://pdp/resources.xml#xpointer(//service/serviceLevel
     [@serviceId='Gold']/ResourceRsvp/*)
 </AttributeAssignment>
 <AttributeAssignment AttributeId="qosH261Q" DataType="#string">
    hdp://pdp/resources.xml#xpointer(//service/serviceLevel
    [@serviceId='Silver']/ResourceRsvp/*)
</AttributeAssignment>
 </Obligation>
</Obligations>
```

Figure 11. Policy Structure for Registered Students in Internal Network

```
<Request>
  <Subject>
    <Attribute AttributeId=":subject:subject-id" DataType="#string">
      <AttributeValue>etoktar</AttributeValue>
    </Attribute>
    <Attribute AttributeId=":subject:authn-locality:ip-address:receiver"
DataType="#string">
      <AttributeValue>192.168.0.1</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId=":resource:resource-id" DataType="#string">
      <AttributeValue>TutorialVideoStreaming</AttributeValue>
    </Attribute>
    <Attribute AttributeId=":resource:authn-locality:ip-address:sender"
DataType="#string">
      <AttributeValue>192.168.200.10</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId=":action:action-id:ServerAction" DataType="#string">
      <AttributeValue>getResourceQoS</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

Figure 12. Example of Policy Request

```
<Response>
  <Result>
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value=":status:ok"/>
    </Status>
          <Obligations xmlns="urn:oasis:names:tc:xacml:1.0:policy">
      <Obligation ObligationId=":GoldSilverStudentsInternal"
FulfillOn="Permit">
        <AttributeAssignment AttributeId="RsvpClass#1"
          DataType="#string"> G711</AttributeAssignment>
        <AttributeAssignment AttributeId="TokenBucketRate_r#1"
          DataType="#double"> 9250.0</AttributeAssignment>
        <AttributeAssignment AttributeId="TokenBucketSize_b#1"
          DataType="#double"> 680.0</AttributeAssignment>
        <AttributeAssignment AttributeId="PeakRate_p#1"
          DataType="#double">13875.0</AttributeAssignment>
        <AttributeAssignment AttributeId="MinimumPoliceUnit_m#1"
          DataType="#integer">13875</AttributeAssignment>
        <AttributeAssignment AttributeId="MaximumPacketSize_M#1"
          DataType="#integer">13875</AttributeAssignment>
        <AttributeAssignment AttributeId="RsvpService#1"
          DataType="#string">Guaranteed</AttributeAssignment>
        <AttributeAssignment AttributeId="ServiceQoS#1"
          DataType="#string">FF</AttributeAssignment>

        <AttributeAssignment AttributeId="RsvpClass#2"
          DataType="#string">H261QCIF</AttributeAssignment>
        <AttributeAssignment AttributeId="TokenBucketRate_r#2"
          DataType="#double">12000.0</AttributeAssignment>
        <AttributeAssignment AttributeId="TokenBucketSize_b#2"
          DataType="#double">6000.0</AttributeAssignment>
        <AttributeAssignment AttributeId="PeakRate_p#2"
          DataType="#double">12000.0</AttributeAssignment>
        <AttributeAssignment AttributeId="MinimumPoliceUnit_m#2"
          DataType="#integer">80</AttributeAssignment>
        <AttributeAssignment AttributeId="MaximumPacketSize_M#2"
          DataType="#integer"> 2500</AttributeAssignment>
        <AttributeAssignment AttributeId="RsvpService#2"
          DataType="#string">Controlled-load</AttributeAssignment>
        <AttributeAssignment AttributeId="ServiceQoS#2"
          DataType="#string"> SE</AttributeAssignment>
      </Obligation>
    </Obligations>
  </Result>
```

Figure 13. Example of Policy Response

6.2. 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 extensions are compatible with existing implementation packages. The strategy adopted consisted in adding new functionalities to the XACML framework without modifying the scheme. We observed that it was not necessary to modify the package code, except in the case of treatment of the *<Obligations>* structure and the use of *XPointer* references to external files. 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);
```

7. Conclusion

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. The 1.0 XACML specification and the corresponding packet implementation are deficient in returning results that are not simple deny or permit decisions. In the proposed work, some features have been added to the XACML framework without modifying its scheme: <Obligations> are dynamically processed and XPointer references to external documents are used for creating policies with reusable resources and subjects.

Some modifications on XACML scheme, however, would be useful. First, we suggest a more flexible way of mapping conditional <Obligations> to policies. Mapping <Obligations> to <Rules> would permit to define different service levels in a single policy. This modification would certainly be useful for other application domains. Another suggested modification is to formalize the use of XPointer references in the XACML scheme.

9. 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.

[15] W3C, XPointer Framework, W3C Recommendation, 25 March 2003.