Defining Reusable Business-Level QoS Policies for DiffServ

André Beller, Edgard Jamhour, Marcelo Pellenz

Pontifícia Universidade Católica do Paraná – PUCPR, PPGIA
Curitiba, PR, Brazil
abeller@ig.com.br, {jamhour, marcelo}@ppgia.pucpr.br

Abstract. This paper proposes a PBNM (Policy Based Network Management) framework for automating the process of generating and distributing DiffServ configuration to network devices. The framework is based on IETF standards, and proposes a new business level policy model for simplifying the process of defining QoS policies. The framework is defined in three layers: a business level policy model (based on a IETF PCIM extension), a device independent policy model (based on a IETF QPIM extension) and a device dependent policy model (based on the IETF *diffserv* PIB definition). The paper illustrates the use of the framework by mapping the information models to XML documents. The XML mapped information model supports the reuse of rules, conditions and network information by using *XPointer* references.

1 Introduction

Policy Based Network Management (PBNM) plays an important role for managing QoS in IP-based networks. [1,2,8]. Recent IETF publications have defined the elements for building a generic, device independent framework for QoS management. An important element in this framework is QPIM (Policy QoS Information Model) [6]. QPIM is an information model that permits to describe device independent configuration policies. By defining a model that is not-device dependent, QPIM permits to "re-use" QoS configuration, i.e., configuration policy concerning similar devices can be defined only once. QPIM configuration is expressed in terms of "policies" assigned to "device interfaces", and does not take into account business level elements, such as users, applications, network topology and time constraints. The RFC 3644 that defines QPIM, points that a complete QoS management tool should include a higher level policy model that could generate the QPIM configuration based on business goals, network topology and QoS methodology (diffserv or intserv) [6].

In this context, this paper proposes a PBNM framework for automating the process of generating and distributing Differentiated Services (diffserv) configuration to network devices. The framework proposes a new business level policy model for simplifying the process of defining QoS policies. The idea of introducing a business level model for QoS management is not new [3,4,5]. However, the proposal presented in this paper differs from the similar works found in the literature because the

business level polices are fully integrated with the IETF standards. By taking advantage of the recent IETF publications concerning QoS provisioning, the framework defines all the elements required for generating and distributing *diffserv* configuration to network devices.

This paper is structured as follows. Section 2 review some related works that also proposes business level models for QoS management. Section 3 presents the overview of our proposal. Section 4 presents the business level policy model, defined as a PCIM extension and fully integrated with QPIM. Section 5 describe the QPIM based configuration model, and the process adopted for transforming the business level policies into configuration policies. Section 6 presents XML mapping strategy and examples for illustrating the use of the proposed model. Finally, the conclusion resumes the important aspects of this work and points to future developments.

2 Related Works and Discussion

This section will review some important works that address the issue of defining a business level QoS policy model. Verma [3] et al. proposes a tool for managing *diffserv* configuration in enterprise networks. The work defines the elements for building a QoS management tool, permitting to transform business level policies into device configuration information. The proposal adopts the concept of translating business level policies based on SLAs (Service Level Agreements) into device configuration. Verma [4] present an extension of this work, introducing more details concerning the business level model and a configuration distributor based on the IETF framework. The business level policy is described by statements with the syntax: “a user (or group of users) accessing an application (or group of applications) in a server (or group of servers) in a specific period of time must receive a specific service class”. The service class is defined in terms of “response time” (i.e., a round-trip delay of packets). An important concept developed in [4] refers to the strategy adopted for distributing the configuration to the network devices and servers. The strategy assumes a *diffserv* topology. For network devices (e.g., routers), a configuration policy is relevant only if the shortest-path between the source and destination IP includes the router. For servers, a configuration policy is relevant if the server IP is included in the source or destination IP ranges defined by the policy. As explained in the next sections, we adopt a similar strategy in our framework.

The Solaris Bandwidth Manager, implemented by Sun [7], proposes a business level QoS model for enterprise networks that closely follows the semantics of the IETF PCIM/PCIMe [12,13]. In the proposed model, a packet flow that satisfies some conditions receives a predefined service class defined in terms of bandwidth percentage and traffic priority. The Sun’s approach adopts the PDP/PEP implementation framework [2], extending the enforcement points to network devices (routers and switches) and servers. The communication between the PDP and the PEP is implemented through a set of proprietary APIs.

There are also attempts of proposing a standard model for representing business level policies. According to the IETF terminology, a SLS (Service Level Specification) represents a subset of a SLA (Service Level Agreement) that refers to

traffic characterization and treatment [8]. There was two attempts of defining a standard SLS model published by IETF as Internet drafts: TEQUILA [9] and AQUILA [10]. TEQUILA (Traffic Engineering for Quality of Service in the Internet, at Large Scale) define a SLS in terms of six main attributes: Scope, Flow Identifier, Performance, Traffic Conformance, Excess Treatment, Service Schedule and Reliability. AQUILA (Adaptative Resource Control for QoS Using an IP-based Layered Architecture) adopts the concept of predefined SLS types, based on the generic SLS definitions proposed by TEQUILA. A predefined SLS type fixes values (or range of values) for a subset of parameters in the generic SLS. According to [10], the mapping process between the generic SLS and the concrete QoS mechanisms can be very complex if the user can freely select and combine the parameters. Therefore, the use of predefined types simplifies the negotiation between customers and network administrators.

The proposal described in this paper has several similarities with the works reviewed in this section. However, the strategy for defining the policy model and the implementation framework differs in some important aspects. Considering the vendors efforts to follow the recent IETF standards, translating business level policies to a diffserv PIB [11], and distributing the configuration information using the COPS-PR [5] protocol is certainly a logical approach for a QoS management tool. None of the works reviewed in this section follows this approach altogether. In [3,4], even though some CIM and PCIM [8] concepts are mentioned, the proposal follows its own approach for representing policies, servers, clients and QoS configuration parameters. In [7], the policy model follows a closer PCIM extension, but the policy distribution and enforcement follows a proprietary approach where neither the PIB structure, nor the COPS protocol is adopted. The TEQUILA project offers some attempts of defining standard representations for SLS agreements. However, as pointed by AQUILA, the mapping between a generic SLS definition to QoS mechanisms can be very complex. AQUILA tries to solve the problem by proposing a set of predefined SLS types. This paper also follows the AQUILA strategy of adopting predefined SLS types. However, instead of using the generic TEQUILA template, our work represent SLS types as predefined actions described in terms of device-independent QPIM configuration policies. Because configurations described in terms of QPIM are easily translated to diffserv PIB instances, this strategy significantly simplifies the process of mapping the business level policies to QoS mechanisms in network devices.

3 Proposal

Fig. 1 presents an overview of our proposed framework (the explanation in this section follows the numbers in the arrows in the figure). The core of framework is the business level policy model (BLPM). The BLPM is defined as a PCIM extension and it is described in details in section 4. BLPM business rules semantics accommodates most of the elements proposed in [3,4 and 7], but all elements (group of users, group of applications and group of servers) are described in terms of standard CIM elements (1). Also, the service classes are defined are in terms of QPIM configuration, or more

precisely, QPIM actions, as explained in the next section (2). The business level policy information (3) is "compiled" to a Configuration Level Policy Model (CLPM) information (4) by the Business Level Policy Compiler (BLPC). The CLPM and the transformations implemented by the BLPC are discussed in section 5. Note that the CLPM repository is pointed as both, input and output of the BLPC module. The CLPM is defined as a combination of QPIM and PCIM/PCIMe classes. The CLPM offers classes for describing both elements in a device configuration: conditions (traffic characterization) and actions. Actions correspond to the configuration of QoS mechanisms such as congestion control and bandwidth allocation, and correspond to predefined QPIM compound actions (i.e., a manager, when creating business level policies, assigns a service level to a SLS by pointing to a predefined group of QPIM actions). The conditions, by the other hand, are generated from the business level definitions (users, applications, and servers). Therefore, a new set of CLPM configuration is created by the BLPC module during an "off-line" compilation process.

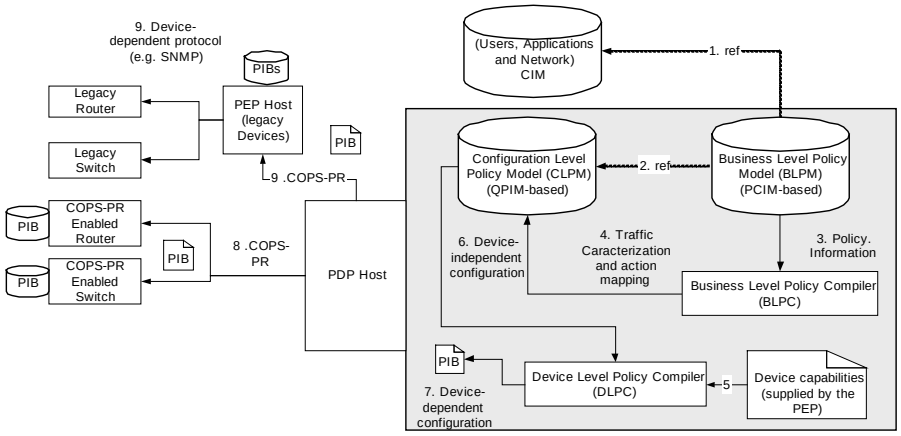

Fig. 1. Framework overview.

The CLPM device-independent configuration (6) is transformed into a device-specific configuration (7) by the Device Level Policy Compiler (DLPC). The DLPC "existence" is conceptually defined by the IETF framework, in the provisioning approach. The device-dependent configuration is expressed in terms of a diffserv PIB, which general structure is defined by the IETF [11]. Because network devices can support different mechanisms for implementing diffserv actions, the DLPC must also receive the "device capabilities" as an input parameter. Device capabilities can be "optionally" transmitted by the PEP through the COPS-PR protocol [5] when the provisioning information is requested to the PDP. The process of configuring network devices consists in transmitting the PIB using the COPS-PR protocol. Two situations can be considered. (i) COPS-PR enabled Network devices capable of directly accepting the PIB information as configuration (i.e., all necessary translation from the PIB to vendor-specific commands are implemented internally by the device). (ii) Legacy devices, where a programmable host is required to act as PEP, converting the

PIB information to vendor-specific commands using a configuration protocol, such as SNMP. The DLPC module and the PIB generation is not discussed in this paper.

4 Business Level QoS Policy Model

The strategy used for describing the business level policies can be expressed as: “user (or group of users) accessing an application (or group of applications) in a server (or group of servers), in a given period of time, must receive a predefined service level”. Fig. 2 presents the UML diagram of the proposed business level policy model. The policy model is derived from the PCIM/PCIMe model [12,13] by creating a new set of specialized classes. Basically, the PCIM/PCIMe model permits to create policies as a group of rules. Each rule has conditions and actions. If the conditions are satisfied, then the corresponding actions must be executed. There are many details concerning how conditions are grouped and evaluated. For a more detailed discussion about extending PCIM model, please, refer to [14].

In our proposal, the *PredefinedSLSAction* refers to a predefined QPIM compound policy action (see Fig. 3). For example, a QoS specialist can create predefined QPIM compound actions defining a Gold, Silver and Bronze service levels (this example is illustrated in the section 6). Then, in the business level policy model, the administrator only makes a reference to the predefined service description using the *PredefinedSLSName* attribute of the *PredefinedSLSAction* class. The conditions of the *SLSPolicyRule* permit to define “who” will receive the service level and “when” the service will be available. Considering the diffserv approach, the “who” policy information must be used for defining: (i) the filtering rules used by the device for classifying the traffic. This information is used for completing the QPIM configuration (as explained next). (ii) which devices must receive the pre-defined service level configuration. This information is used by the PDP for selecting which policies must be provisioned in a given device.

In the business level policy model the “who” information is represented by the *CompoundTargetPolicyCondition* class. This class defines users/applications/servers semantic and it is composed by three *CompoundPolicyCondition* extensions: *CompoundServerPolicyCondition*, *CompoundApplicationPolicyCondition* and *CompoundUserPolicyCondition*. In our model, compound conditions have been choosen for supporting information reuse. A compound condition permits defining objects in terms of logical expressions. These logical expressions are formed by *SimplePolicyConditions*, which follow the semantics “variable” match “value”, defined by PCIMe. The variables refer to already defined CIM objects (*PolicyExplicitVariable*), permitting to create policies that reuse CIM information. Therefore, compound conditions can be used for representing group of users, group of applications and group of servers that can be reused in several business policies.

CompoundServerPolicyCondition refers to one or more CIM *UnitaryComputerSystem* objects, permitting to retrieve the correponding server IP addresses through the associated *RemoteServiceAccessPoint* objects. *CompoundUserPolicyCondition* refers to one or more CIM *Person* objects, permitting to retrieve the correponding user’s host IP addresses or host names also through the

associated *RemoteServiceAccessPoint* objects. Finally, *CompoundApplicationPolicyCondition* points to one or more CIM *ApplicationSystem or InstalledProduct* objects permitting to retrieve the application's protocol and port information trough the associated *SoftwareFeatures* and *ServiceAccessPoint* objects.

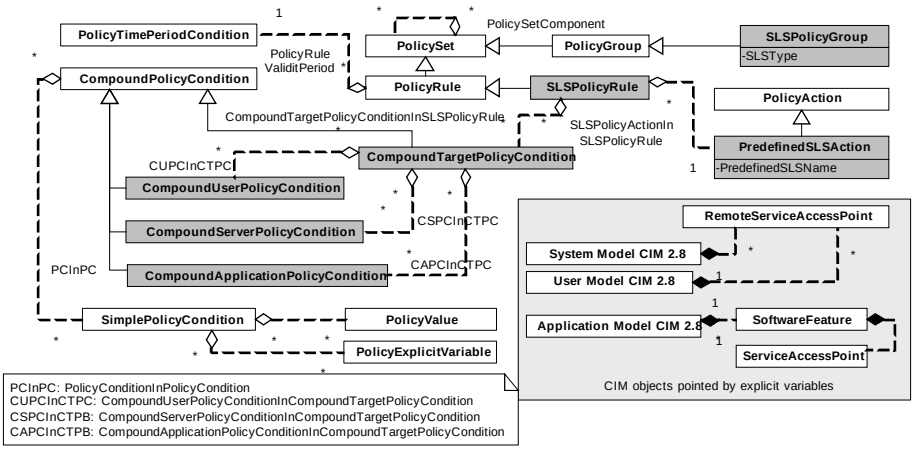

Fig. 2. The PCIM/PCIMe-based business level QoS Policy Model (extended classes are shown in gray). In the proposed model, a policy is represented by a *SLSPolicyGroup* instance. A *SLSPolicyGroup* contains one or more *SLSPolicyRule* instances (associated by the *PolicySetComponent*). When the conditions of a *SLSPolicyRule* are satisfied, then the corresponding *PredefinedSLSActions* must be executed.

5 Configuration Level QoS Policy Model

Our proposal adopts the strategy of representing SLS predefined actions using the QPIM model. The QPIM model is a PCIM/PCIMe extension, and aims to offer a device independent approach for modeling the configuration of *intserv* and *diffserv* devices. Because our work addresses only the *diffserv* methodology, only the *diffserv* elements of QPIM will be presented and discussed. For *diffserv*, QPIM should offer elements for representing both, traffic profile, used by QoS mechanisms to classify the traffic, and QoS actions, used by the QoS mechanisms to adequate the output traffic to the specified levels. In fact, the RFC 3644 [6] does not present the complete model. Instead, it presents only the new classes that are related to QoS actions. The RFC merely suggests that developers must combine the QPIM elements with PCIM/PCIMe for creating a complete configuration model. Fig. 3 presents our approach for using the QPIM extensions.

A device configuration is expressed by a *ConfigPolicyGroup* instance. Note in Fig. 3 that this class is associated to a *PolicyRole* collection. This association permits to assign "roles" for the configuration. According to IETF, roles are used by the PDP to decide which configuration must be transmitted to a given PEP (i.e., a network device

interface). During the provisioning initialization, a PEP informs the roles assigned to the device interfaces, and the PDP will consider all the *ConfigPolicyGroup* instances that match these roles. In our approach a *ConfigPolicyGroup* instance is dynamically created as a result of the Business Policy Level (BPL) compilation. Therefore, the BPL compiler must also determine which roles are assigned to the configuration. This is determined by the association between the *PolicyRoleCollection* and the CIM Network class. The BPL compiler assures that all business policies including users or servers with IP addresses belonging to the network subnet associated to a given *PolicyRoleCollection* will generate configuration policies with the same roles of this collection.

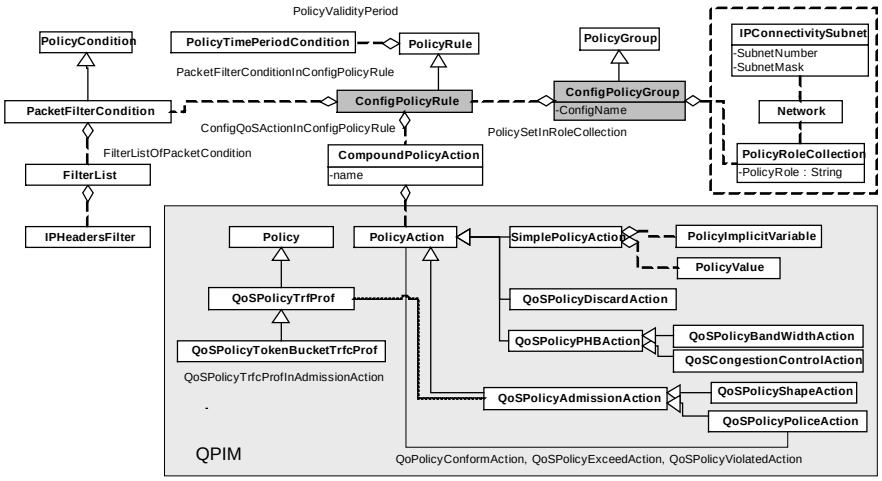

Fig. 3. The configuration policy model, including PCIM/PCIMe and QPIM classes. The QPIM classes are highlighted in the figure by a grey rectangle. We have introduced two new classes: *ConfigPolicyGroup* and *ConfigPolicyRule*. The other classes are defined by PCIM/PCIMe, CIM Policy and CIM Network.

A *ConfigPolicyGroup* instance aggregates one or more *ConfigPolicyRule* instances. In our approach, each *ConfigPolicyRule* instance is associated to *PacketFilterCondition* instances and to *CompoundPolicyAction* instances. *PacketFilterConditions* are used for defining the rules classifying the traffic that will benefit from the QoS service level defined by the *CompoundPolicyAction*. The *PacketFilterConditions* are defined by the BPL compiler considering the "who" information in the BPL model. The *CompoundPolicyAction* instance is a pre-defined SLS QoS action, which is simply pointed by the BPL compiler by matching the attribute *PredefinedSLSName* in the BPL model with the name attribute of the *CompoundPolicyAction*. The actions included in the *CompoundPolicyAction* are defined by QPIM [6]. An example of QPIM configuration is presented in the section 6.

6 XML Mapping and Examples

The proposed framework have been implemented using XML for mapping all information model related to the business level policy model, configuration policy model and CIM information. The strategy adopted for mapping the information models into XML is inspired by the LDAP mapping guidelines proposed by IETF and DTMF, and can be summarized as follows: (i) for the structural classes the mapping is one-for-one, information model classes and their properties map to XML elements and their attributes. (ii) for the relationship classes two different mappings are used: If the relationship does not involve information reuse, a superior-subordinate relationship is established by XML parent-child relationship, the association class is not represented and its attributes are included in the child element. If the relationship involves reusable information, the association class maps to a XML child node, which includes a *XPointer* reference [15] attribute that points to a specific reusable object. In this case, if the relationship is an association, the parent node corresponds to the antecedent class and the child node points to the dependent class. If the relationship is an aggregation, the parent node corresponds to the group component and the child node points to the part component class.

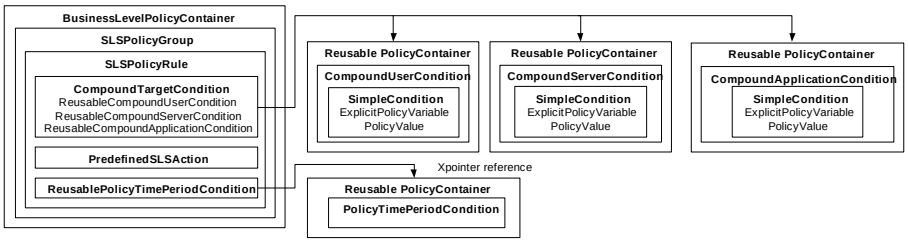

Fig. 4. Business level XML mapping structure. In the <*SLSPolicyRule*> element the conditions are defined by <*CompoundTargetPolicyCondition*> elements that point to user, application and server compositions stored in a <*ReusablePolicyContainer*>. The mapping supports the reuse of *CompoundPolicyConditions* and *PolicyTimePeriodConditions.* The simple conditions are based on the *ExplicitPolicyVariable* semantics, which permits to make references to elements described in terms of CIM objects. In our approach, simple conditions are not reusable.

In our implementation, XML was preferred as an alternative to LDAP, due to the considerable availability of development tools and recent support introduced in commercial relational databases. However, the information model discussed in this paper can also be mapped to LDAP or to a hybrid combination between LDAP and XML. Fig. 4 illustrates XML mapping structure, and the strategy adopted for supporting information reuse in the business level policy repository. Fig. 5 presents and example of a business level policy model (BLPM) mapped in XML. Fig. 6 illustrates the compound conditions representing users, applications and servers.

```
<PolicyContainer Name="BusinessLevelPolicy">
  <SLSPolicyGroup SLSType="Olimpic" PolicyDecisionStrategy="2">
    <!—Silver Rule -->
    <SLSPolicyRule Name="SilverRule" Enabled="1" ConditionListType="1" ExecutionStrategy="2" Priority="2">
      <CompoundTargetPolicyCondition ConditionListType="1"  GroupNumber="1" ConditionNegated="false">
        <CompoundUserPolicyConditionInCompoundTargetPolicyCondition GroupNumber="1
        ConditionNegated="false"  PartComponent="./CompoundConditions.xml#
        xpointer(//CompoundUserPolicyCondition[@Name='CommercialManager']) " />
        <CompoundApplicationPolicyConditionInCompoundTargetPolicyCondition .../>
        <CompoundServerPolicyConditionInCompoundTargetPolicyCondition … />
      </CompoundTargetCondition>
      <PredefinedSLSPolicyAction PredefinedSLSName="Silver" />
      <PolicyRuleValidityPeriod PartComponent="./Validity.xml#
      xpointer(//PolicyTimePeriodCondition[@Name='Period1'])" />
    </SLSPolicyRule>
    <!-- Gold Rule and Bronze Rule ....  -->
  </SLSPolicyGroup>
</PolicyContainer>
```

Fig. 5. Example of business level policy in XML. The *SLSType* attribute in the <*SLSPolicyGroup*> indicates the predefined set of reusable service types adopted in the model. In this case, the "Olimpic" indicates three service levels (SLS), named "Bronze", "Silver" and "Gold". Only the service level corresponding to "Silver" is detailed in the figure by the corresponding <*SLSPolicyRule*> element. The <*CompoundTargetPolicyCondition*> defines the conditions for receiving the "Silver" pre-defined SLS action. The *XPointer* expression assigned to the *PartComponent* attributes follows the syntax "reusable-info-repository URI"#xpointer("XPath expression for selected nodes in the repository").

```
<!– CompoundConditions.xml -->
<ReusablePolicyContainer Name="CompoundUserCondition">
  <CompoundUserPolicyCondition Name="CommercialManager" ConditionListType="1">
    <SimplePolicyCondition GroupNumber="1" ConditionNegated="false">
      <PolicyExplicitVariable ModelClass="Person" ModelProperty="BusinessCategory" />
      <PolicyStringValue StringList="Manager" />
    </SimplePolicyCondition>
    <SimplePolicyCondition GroupNumber="1" ConditionNegated="false">
      <PolicyExplicitVariable ModelClass="Person" ModelProperty="OU" />
      <PolicyStringValue StringList="CommercialDepartment" />
    </SimplePolicyCondition>
  </CompoundUserPolicyCondition>
</ReusablePolicyContainer>
<ReusablePolicyContainer Name="CompoundApplicationCondition"> ...
</ReusablePolicyContainer>
<ReusablePolicyContainer Name="CompoundServerCondition"> ...
</ReusablePolicyContainer>
```

Fig. 6. Example of reusable compound conditions. The "*CommercialManager*" <*CompoundUserCondition*> selects the users matching "BusinessCategory = Manager" AND "OU = *CommercialDepartment*" .

Fig. 7 illustrates the strategy adopted for mapping the configuration level information model. Fig. 8 illustrates an example of configuration policy generated by the BLPC. The corresponding predefined SLS compound action is illustrated in Fig. 9, and the reusable QPIM actions and associations are illustrated in Fig. 10.

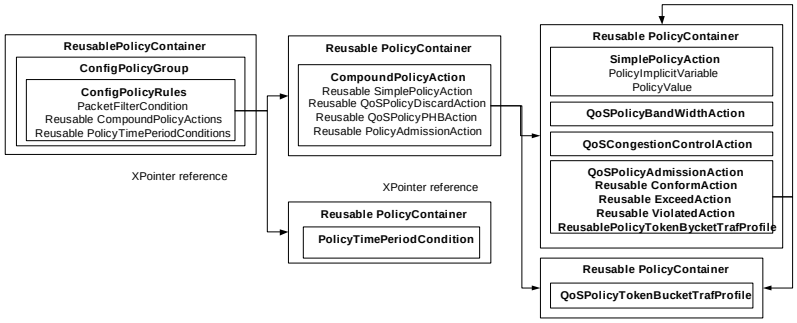

Fig. 7. . Configuration Level XML Mapping Structure. A <*ConfigPolicyGroup*> groups the <*ConfigPolicyRules*> corresponding to the configuration of devices with "similar role" in the network. The *PacketFilerCondition* is generated by the BLPC, and it is not reusable. The <*CompoundPolicyActions*> and <*PolicyTimePeriodConditions*>, however, are reusable information pointed by *XPointer* references. Note the <*CompoundPolicyAction*> also points to reusable QPIM actions.

```
<PolicyContainer Name="ConfigPolicy">
  <ConfigPolicyGroup ConfigName="OlimpicConfigQoSCommercial" PolicyDecisionStrategy="1">
    <ConfigPolicyRule Enabled="1" ConditionList Type="1" Priority="2">
      <PacketFilterCondition FilterEvaluation="4" GroupNumber="2" ConditionNegated="false">
        <IPHeadersFilter IsNegated="False" HdrIPVersion="4" HdrSrcAddress="0.0.0.0" HdrSrcMask="0"
          HdrDestAddress="10.0.4.1" HdrDestMask="24" Direction="3"/>
      </PacketFilterCondition>
      <!-- ... other PacketFilterConditions -->
      <PolicyRuleValidityPeriod PartComponent="./Time.xml#
        xpointer(//PolicyTimePeriodCondition[@Name='Period1'])" />
      <ConfigQoSActionInConfigPolicyRule PartComponent="./QoSOlimpic.xml#
        xpointer(//CompoundPolicyAction[@name='SilverAction'])" />
    </ConfigPolicyRule>
  </ConfigPolicyGroup>
  <!-- ... other ConfigPolicyGroups -->
</PolicyContainer>
```

Fig. 8. Configuration policy generated by the BPL compiler. In this example, each <*ConfigPolicyGroup*> represents the configuration of the devices in a specific subnet in a enterprise *diffserv* network. Only the configuration policy corresponding to the Silver service level in the Commercial subnet is detailed in the figure.

```
<ReusablePolicyContainer Name="OlimpicQoSSpecification">
  <CompoundPolicyAction Name="BronzeAction" SequencedActions="1"
    ExecutionStrategy="2"> … </CompoundPolicyAction>
  <CompoundPolicyAction Name="SilverAction" SequencedActions="1" ExecutionStrategy="2">
    <PolicyActionInPolicyAction ActionOrder="1" PartComponent=
      "./QPIMAction.xml#xpointer(//QoSPolicyPoliceAction[@Name=' PoliceSilverFlow '])"/>
      <PolicyActionInPolicyAction ActionOrder="2" PartComponent=
      "./QPIMAction.xml#xpointer(//QoSPolicyCongestionControlAction[@Name='SilverQueueClass'])" />
      <PolicyActionInPolicyAction ActionOrder="3" PartComponent=
      "./ QPIMAction.xml#xpointer(//QoSPolicyBandwidthAction[@Name='SilverBWClass'])" />
  </CompoundPolicyAction>
  <CompoundPolicyAction name="GoldAction" SequencedActions="1" ExecutionStrategy="2">…
  </CompoundPolicyAction>
</ReusablePolicyContainer>
```

Fig. 9. Example of reusable pre-defined QPIM compound actions. The compound "*SilverAction*" points to a set of reusable QPIM actions, which must be executed in a predefined order.

```
<ReusablePolicyContainer name="QPIMAction">
  <QoSPolicyPoliceAction Name="PoliceSilverFlow" qpAdmissionScope="0">
    <QoSPolicyTrfcProfInAdmissionAction Dependent="./QPIMAction.xml#
      xpointer(//QoSPolicyTokenBucketTrfcProf[@Name='SilverTBFlow'])" />
    <PolicyConformAction Dependent="./ QPIMAction.xml #
      xpointer(//SimplePolicyAction[@Name='SilverDSCPFlowConform'])" />
    <PolicyExceedAction … />
    <PolicyViolateAction … />
  </QoSPolicyPoliceAction>
  <QoSPolicyCongestionControlAction Name="SilverQueueClass" qpQueueSizeUnits="1" qpQueueSize="15"
    qpDropMethod="3" qpDropThresholdUnits="0" qpMinThresholdValue="30" qpMaxThresholdValue="45" />
  <QoSPolicyBandwidthAction Name="SilverBWClass" qpBandwidthUnits="1" qpMinBandwidth="25" />
  <SimplePolicyAction Name="SilverDSCPFlowConform">
    <PolicyDSCPVariable Name="PolicyDSCPVariable" />
    <PolicyIntegerValue IntegerList="AF21" />
  </SimplePolicyAction>
  <SimplePolicyAction Name="SilverDSCPFlowExceed">…  </SimplePolicyAction>
  <SimplePolicyAction Name="SilverDSCPFlowViolate">…  </SimplePolicyAction>
  …
</ReusablePolicyContainer>

<ReusablePolicyContainer name="TokenBucket">
  <QoSPolicyTokenBucketTrfcProf  Name="SilverTBFlow"
    qpTBRate="256" qpTBNormalBurst="64" qpTBExcessBurst="32" />
  <!-- other traffic profiles -->
</ReusablePolicyContainer>
```

Fig. 10. Example of reusable pre-defined QPIM actions.

Conclusion

This work contributes for defining a complete framework for QoS diffserv management that is in according with recent IETF standards. This work proposes a new business level model and completes the QPIM model with classes required for defining filtering conditions for diffserv configuration. An important point with

respect to the implementation of CIM/PCIM-based frameworks concerns the strategy adopted for mapping class associations to XML or LDAP. Because the directives published by IETF and DTMF offers several possibilities for mapping the information model classes, retrieving information from a repository requires a previous knowledge of how the information classes have been mapped to a specific repository schema. That poses an important obstacle for building “out-of-the box” frameworks that could reuse existent CIM/PCIM information. This is certainly a point that should be addressed by IETF and DMTF. Future works includes extending the business level policy model for supporting more elaborated policies rules and the development of a graphical tool for generating the business level policies.

References

1. 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 .
2. Yavatkar, R., Pendarakis, D.; Guerin, R. A Framework for Policy-Based Admission Control, RFC2753, Jan. 2000.
3. D. Verma, M. Beigi and R. Jennings, "Policy Based SLA Management in Enterprise Networks", Proceedings of Policy WorkShop 2001.
4. D. Verma, "Simplifying Network Administration using Policy based Management", IEEE Network Magazine, March 2002.
5. 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)”, IETF RFC 3084, Mar. 2001.
6. Snir, Y.; Ramberg, Y.; Strassner, J.; Cohen, R.; Moore, B.; “Policy Quality of Service (QoS) Information Model”, IETF RFC 3644, Nov. 2003.
7. Kakadia, D.; “Enterprise QoS Based Systems & Network Management”, Sun Microsystems White Paper, Article #8934, Volume 60, Issue 1, SysAdmin Section, February 4, 2003.
8. J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, S. Waldbusser; “Terminology for Policy-Based Management”, IETF RFC 3198, Nov. 2001.
9. D. Goderis, D. Griffin, C. Jacquenet, G. Pavlou; “Attributes of a Service Level Specification (SLS) Template”, IETF draft, October 2003.
10. S. Salsano, F. Ricciato, M. Winter, G. Eichler, A. Thomas, F. Fuenfstueck, T. Ziegler, C. Brandauer; “Definition and usage of SLSs in the AQUILA consortium”, IETF draft, Nov. 2000 (expired).
11. K. Chan, R. Sahita, S. Hahn, K. McCloghrie, “Differentiated Services Quality of Service Policy Information Base”, IETF RFC 3317, Mar. 2003.
12. B. Moore, E. Elleson, J. Strasser, A. Weterinen: Policy Core Information Model. IETF RFC 3060, February 2001.
13. B. Moore, E. Elleson, J. Strasser, A. Weterinen: Policy Core Information Model Extensions. IETF RFC 3460, February 2001.
14. 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, pg. 181-193.
15. W3C, XPointer Framework, W3C Recommendation, 25 March 2003.