Chapter 4

Policy Repository

There are several techniques and concepts to design and implement policies, but there is no commonly accepted terminology or notation. This Thesis suggests the use of PCIM [Moore01] and their extensions [Moore03] to design policies. However, it is not possible to store policies in a LDAP repository in that way, it is necessary to do a mapping between PCIM and PCIMe classes into the LDAP classes and attributes.

Diagram 1 provides an overview of the classes that comprise the CIM Core Policy Model, their associations to each other, and their associations to other classes in the overall CIM model.[CIM-DMTF].

Currently, the mapping of PCIM classes into an LDAP scheme is an Internet Draft of the IETF, and the mapping of the PCIM extension is part of this Thesis and forms the Internet Draft [Reyes03]. Next there is a detail about the LDAP mapping of the PCIMe.

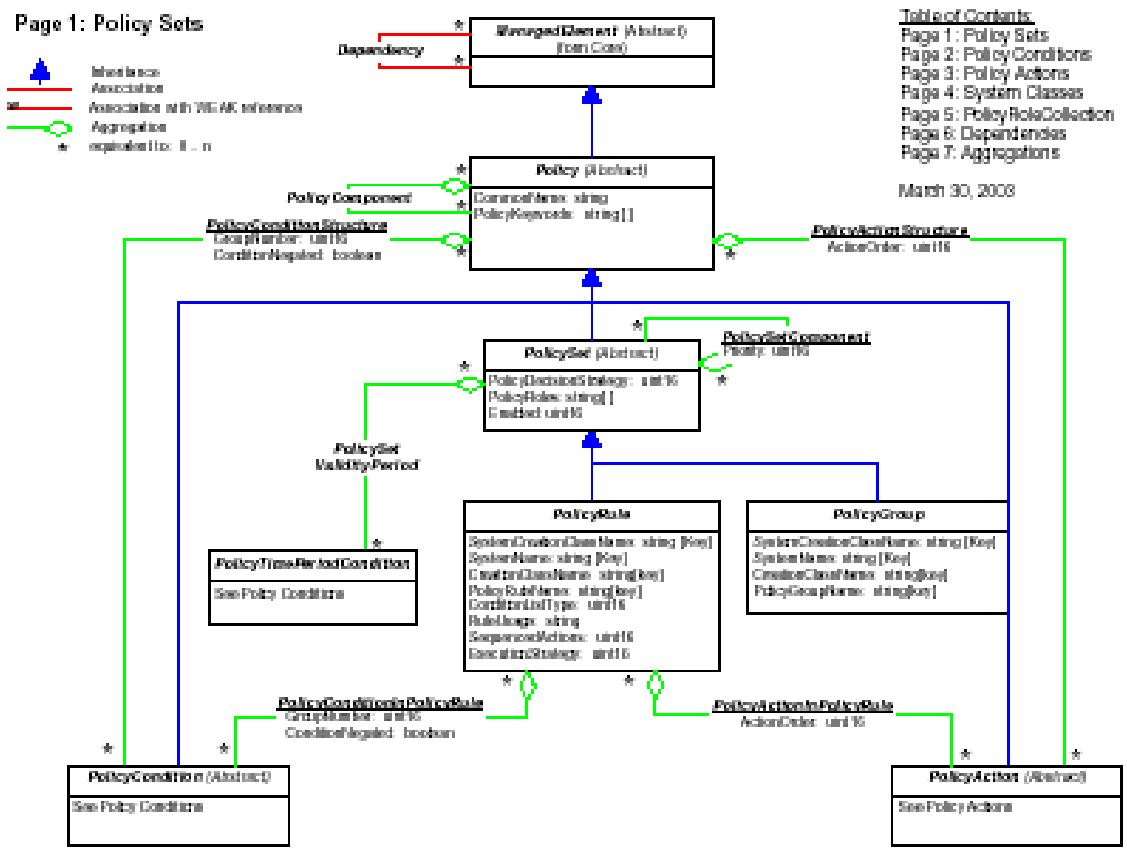

Figure 1. PCIM classes distribution

4.1 LDAP mapping of the Policy Core Information Model (PCIM) extensions to an LDAP schema

This section presents the mapping of the object-oriented information model for representing policy information to a concrete implementation using a directory that uses LDAPv3 as its access protocol. This mapping is an LDAP schema representing the classes defined in the Policy Core Information Model Extensions [Moore03]. The document [Reyes03] is an extension to [Strassner02], which defines the LDAP mapping of the Policy Core Information Model [Moore01] to an LDAP schema. The changes include additional classes previously not covered, deprecation of some object classes defined in PCLS and changes to the existing class hierarchy in PCLS.

The term 'PCELS' (Policy Core Extension LDAP Schema) is used to refer to the LDAP object class definitions proposed. The classes described in appendix I contains certain optimizations for a directory that uses LDAP as an access protocol. One example is the use of auxiliary classes to represent some of the associations defined in the information model. Note that other storage types might need to implement the association differently.

Forty-nine of the classes in the PCELS come directly from the fourty-five corresponding classes in the information model extensions. The prefix "pcime" is used to identify these LDAP classes. See next table.

Information Model (PCIM ext)	**LDAP Class(es)**
PolicySet	pcimePolicySet
PolicyRule	pcimeRule pcimeRuleAuxClass pcimeRuleInstance
SimplePolicyCondition	pcimeSimpleConditionAuxClass
CompoundPolicyCondition	pcimeCompoundConditionAuxClass
CompoundFilterCondition	pcimeCompoundFilterAuxClass
SimplePolicyAction	pcimeSimpleActionAuxClass
CompoundPolicyAction	pcimeCompoundActionAuxClass
PolicyVariable	pcimeVariable
PolicyExplicitVariable	pcimeExplicitVariableAuxClass
PolicyImplicitVariable	pcimeImplicitVariableAuxClass
PolicySourceIPv4Variable	pcimeSourceIPv4VariableAuxClass
PolicySourceIPv6Variable	pcimeSourceIPv6VariableAuxClass
PolicyDestinationIPv4Variable	pcimeDestinationIPv4VariableAuxClass
PolicyDestinationIPv6Variable	pcimeDestinationIPv6VariableAuxClass
PolicySourcePortVariable	pcimeSourcePortVariableAuxClass

PolicyDestinationPortVariable	pcimeDestinationPortVariableAuxClass
PolicyIPProtocolVariable	pcimeIPProtocolVariableAuxClass
PolicyIPVersionVariable	pcimeIPVersionVariableAuxClass
PolicyIPToSVariable	pcimeIPToSVariableAuxClass
PolicyDSCPVariable	pcimeDSCPVariableAuxClass
PolicyFlowIDVariable	pcimeFlowIDVariableAuxClass
PolicySourceMACVariable	pcimeSourceMACVariableAuxClass
PolicyDestinationMACVariable	pcimeDestinationMACVariableAuxClass
PolicyVLANVariable	pcimeVLANVariableAuxClass
PolicyCoSVariable	pcimeCoSVariableAuxClass
PolicyEthertypeVariable	pcimeEthertypeVariableAuxClass
PolicySourceSAPVariable	pcimeSourceSAPVariableAuxClass
PolicyDestinationSAPVariable	pcimeDestinationSAPVariableAuxClass
PolicySNAPOUIVariable	pcimeSNAPOUIVariableAuxClass
PolicySNAPTypeVariable	pcimeSNAPTypeVariableAuxClass
PolicyFlowDirectionVariable	pcimeFlowDirectionVariableAuxClass
PolicyValue	pcimeValueAuxClass
PolicyIPv4AddrValue	pcimeIPv4AddrValueAuxClass
PolicyIPv6AddrValue	pcimeIPv6AddrValueAuxClass
PolicyMACAddrValue	pcimeMACAddrValueAuxClass
PolicyStringValue	pcimeStringValueAuxClass
PolicyBitStringValue	pcimeBitStringValueAuxClass
PolicyIntegerValue	pcimeIntegerValueAuxClass
PolicyBooleanValue	pcimeBooleanValueAuxClass
PolicyRoleCollection	pcimeRoleCollection
ReusablePolicyContainer	PcimeReusableContainer PcimeReusableContainerAuxClass pcimeReusableContainerInstance
FilterEntryBase	pcimeFilterEntryBase
IPHeadersfilter	pcimeIPHeadersfilter
8021Filter	pcime8021Filter

FilterList	pcimeFilterList

Table 1.LDAP Classes

Next table shows the associations established in PCIMe and their mapping to LDAP attributes or classes.

Information Model Association	**LDAP Attribute / Class**
PolicySetComponent	pcimePolicySetComponentList in pcimePolicySet and pcimePolicySetDN in pcimePolicySetAsociation
PolicySetInSystem	DIT Containment and pcimePolicySetDN in pcimePolicySetAsociation
PolicyGroupInSystem	(same as PolicySetInSystem)
PolicyRuleInSystem	(same as PolicySetInSystem)
PolicyConditionStructure	pcimConditionDN in pcimeConditionAssociation
PolicyConditionInPolicyRule	pcimeConditionList in pcimeRule and pcimConditionDN in pcimeConditionAssociation
PolicyConditionInPolicyCondition	pcimeConditionList in pcimeCompoundConditionAuxClass and pcimConditionDN in pcimeConditionAssociation
PolicyActionStructure	pcimActionDN in pcimeActionAssociation
PolicyActionInPolicyRule	pcimeActionList in pcimeRule and pcimActionDN in pcimeActionAssociation
PolicyActionInPolicyAction	pcimeActionList in pcimeCompoundActionAuxClass and pcimActionDN in pcimeActionAssociation
PolicyVariableInSimplePolicy Condition	pcimeVariableDN in pcimeSimpleConditionAuxClass
PolicyValueInSimplePolicy Condition	pcimeValueDN in pcimeSimpleConditionAuxClass
PolicyVariableInSimplePolicy Action	pcimeVariableDN in pcimeSimpleActionAuxClass

PolicyValueInSimplePolicyAction	pcimeValueDN in pcimeSimpleActionAuxClass
ReusablePolicy	DIT containment
ExpectedPolicyValuesForVariable	DIT containment or pcimeExpectedValueList in pcimeVariable
ContainedDomain	DIT containment or pcimeReusableContainerList in pcimeReusableContainer
EntriesInFilterList	DIT containment or pcimeFilterListEntriesList in pcimeFilterList
ElementInPolicyRoleCollection	DIT containment or pcimeElementList in pcimeRoleCollection
PolicyRoleCollectionInSystem	DIT Containment

Table 2. PCIMe associations and their mapping to LDAP.

4.1.1 Attaching PolicyVariable and PolicyValues to PolicySimpleCondition and PolicySimpleAction

A PolicySimpleCondition as well as a PolicySimpleAction includes a single PolicyValue and a single PolicyVariable. Each of them can be attached or referenced by a DN. The attachment helps create compact PolicyCondition and PolicyAction definitions that can be efficiently provisioned and retrieved from the repository. On the other hand, referenced PolicyVariables and PolicyValues instances can be reused in the construction of multiple policies and permit the administrative partitioning of the data and policy definitions.

4.1.2 Aggregation of actions/conditions in PolicyRules and CompoundActions/ Conditions.

In PCIM_EXT were defined two new classes that offer the designer the capability of creating more complex conditions and actions. CompoundPolicyCondition and

CompoundPolicyActionclasses are mapped in the PCELS' CompoundConditionAuxClass and CompoundActionAuxClass classes and inherit from pcimConditionAuxClass/pcimActionAuxClass

Because of this inheritance they are stored in the same way the non-compound conditions/actions are. The compound conditions/actions defined in the PCIM_EXT are extensions of the rule capability to associate, grouping and evaluate/execute conditions/actions so the conditions/actions are associated to the compounds conditions/actions as they were associated to the rules in the PCLS.

In this section is explained how to store this classes in the directory. As a general rule, the specific conditions/actions are DIT contained under rule or compound condition/action classes and attached to the association classes. The reusable conditions/actions, compound and non-compound, are contained in reusable containers and attached to policy instances.

The examples below illustrate the four possible cases combining specific/reusable compound/non-compound condition/action. The rule has two compound conditions; each one has two different conditions. The schemes can be extended in order to store actions.

The mapping of compound conditions/actions and the schemas below are based on the section 4.4 of the PCLS [Strassner02] and how conditions and actions are associated to rules and repositories.

First case: specific compound condition/action with specific conditions/actions.

Because the compound conditions/actions are specific to Rule, the auxiliary classes that represent them are attached to, structural classes pcimeConditionAssociation or pcimeActionAssociation. These structural classes represent the association between the rule and the compound condition and compound action. The rule's specific condition/action are DIT contained in rule entry.

The conditions/actions have to be tied to compound conditions/actions in the same way as compound conditions/actions are tied to rules, but association classes do the association between them compound conditions/actions and its specific conditions/actions.

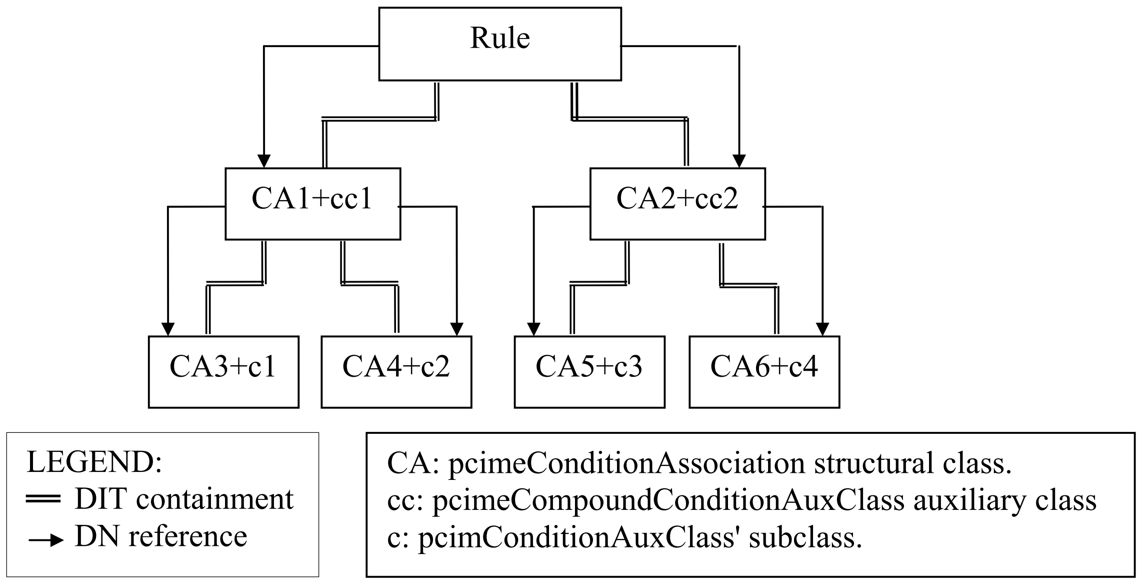
Rule
CA1+cc1
CA2+cc2
CA3+c1
CA4+c2
CA5+c3
CA6+c4
LEGEND:
═ DIT containment
→ DN reference
CA: pcimeConditionAssociation structural class.
cc: pcimeCompoundConditionAuxClass auxiliary class
c: pcimConditionAuxClass' subclass.

Figure 2. Specific compound condition/action with specific conditions/actions

Second case: Rule's specific compound conditions/actions whit reusable conditions/actions.

This case is similar to the first one. The conditions/actions are reusable so they are not attached to the association classes but they are attached to structural classes in the reusable container. It's needed that the association classes tie the conditions/actions in the reusable container using DN references.

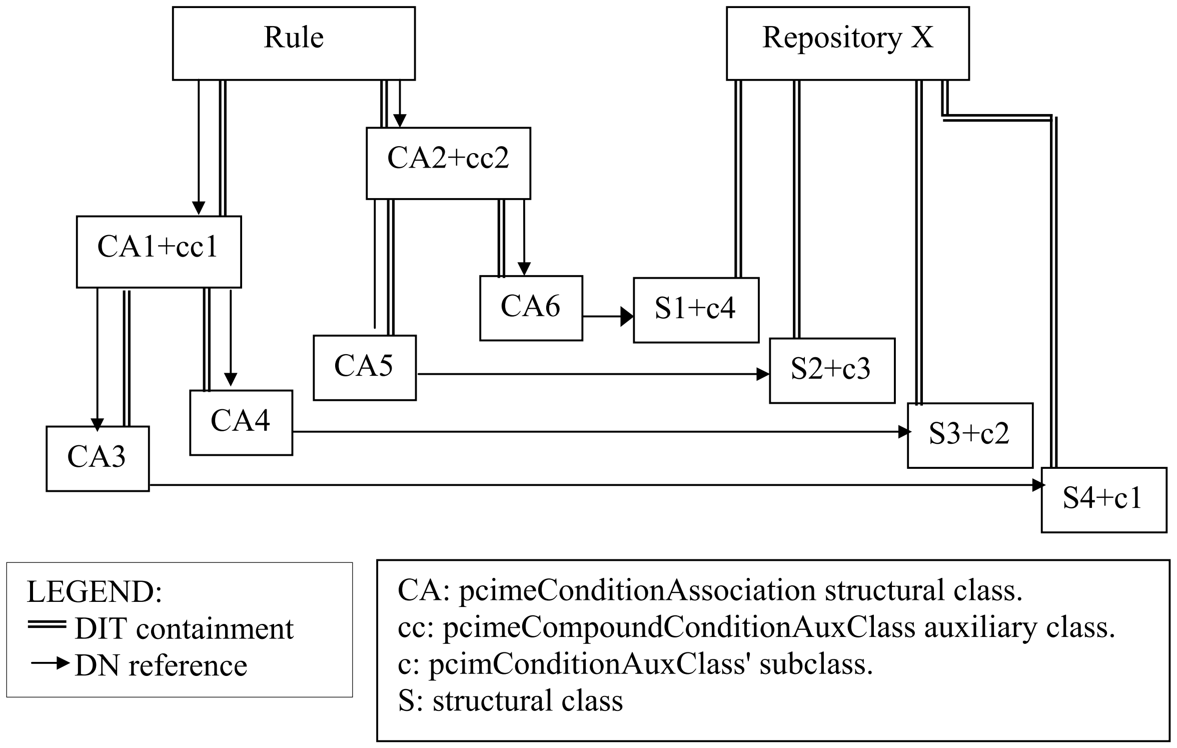
Rule
Repository X
CA2+cc2
CA1+cc1
CA6
S1+c4
CA5
S2+c3
CA4
S3+c2
CA3
S4+c1
LEGEND:
DIT containment
DN reference
CA: pcimeConditionAssociation structural class.
cc: pcimeCompoundConditionAuxClass auxiliary class.
c: pcimConditionAuxClass' subclass.
S: structural class

Figure 3 Rule's specific compound conditions/actions whit reusable conditions/actions

Third case: Reusable compound condition/action with specific conditions/actions.

Because of the re-usability of the compound compound condition/action they are attached to structural classes and stored in the reusable container. They are related to the rule through the DN reference between the association classes and the compound condition/action. The specific conditions/actions are DIT contained in the compound condition/action entries.

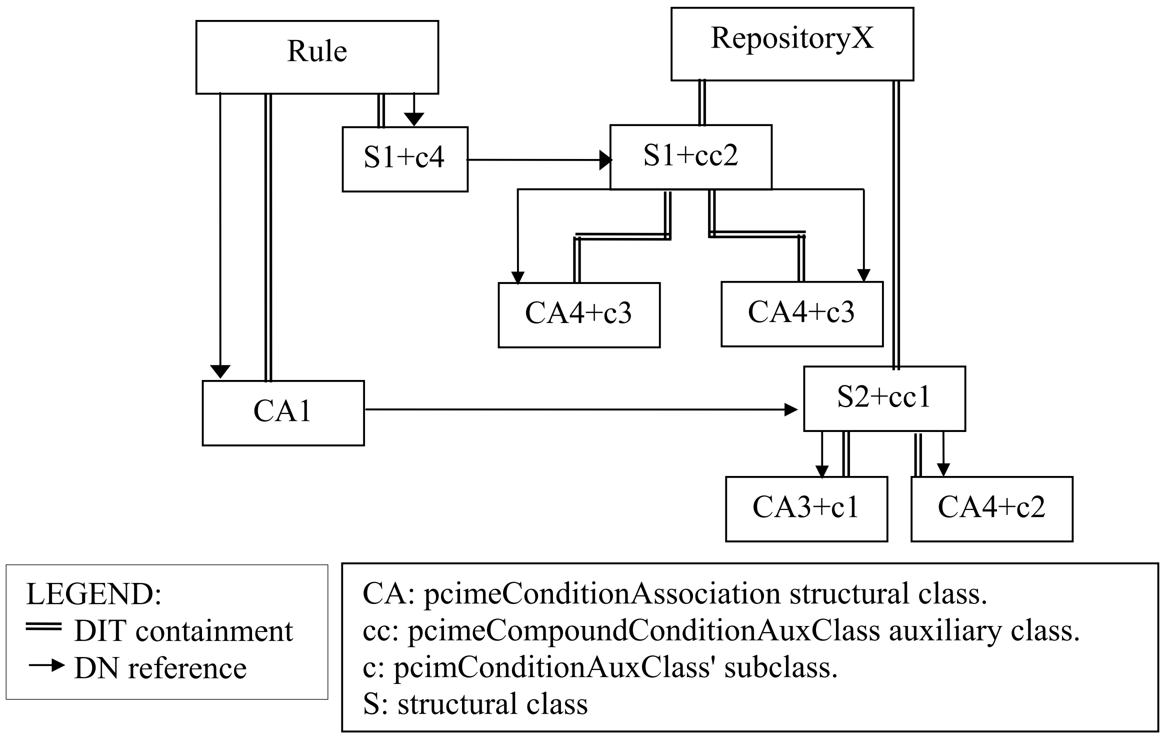
Rule
RepositoryX
S1+c4
S1+cc2
CA4+c3
CA4+c3
CA1
S2+cc1
CA3+c1
CA4+c2
LEGEND:
DIT containment
DN reference
CA: pcimeConditionAssociation structural class.
cc: pcimeCompoundConditionAuxClass auxiliary class.
c: pcimConditionAuxClass' subclass.
S: structural class

Figure 4. Reusable compound condition/action with specific conditions/actions

Fourth case: Reusable conditions/actions and compound conditions/actions.

All the conditions/actions are reusable so they are stored in reusable containers. Figure 4 illustrates two different repositories or reusable containers but the number of containers in the system depends on the policy administrator so the conditions/actions could be stored in the same container or each condition/action could be stored in a different container.

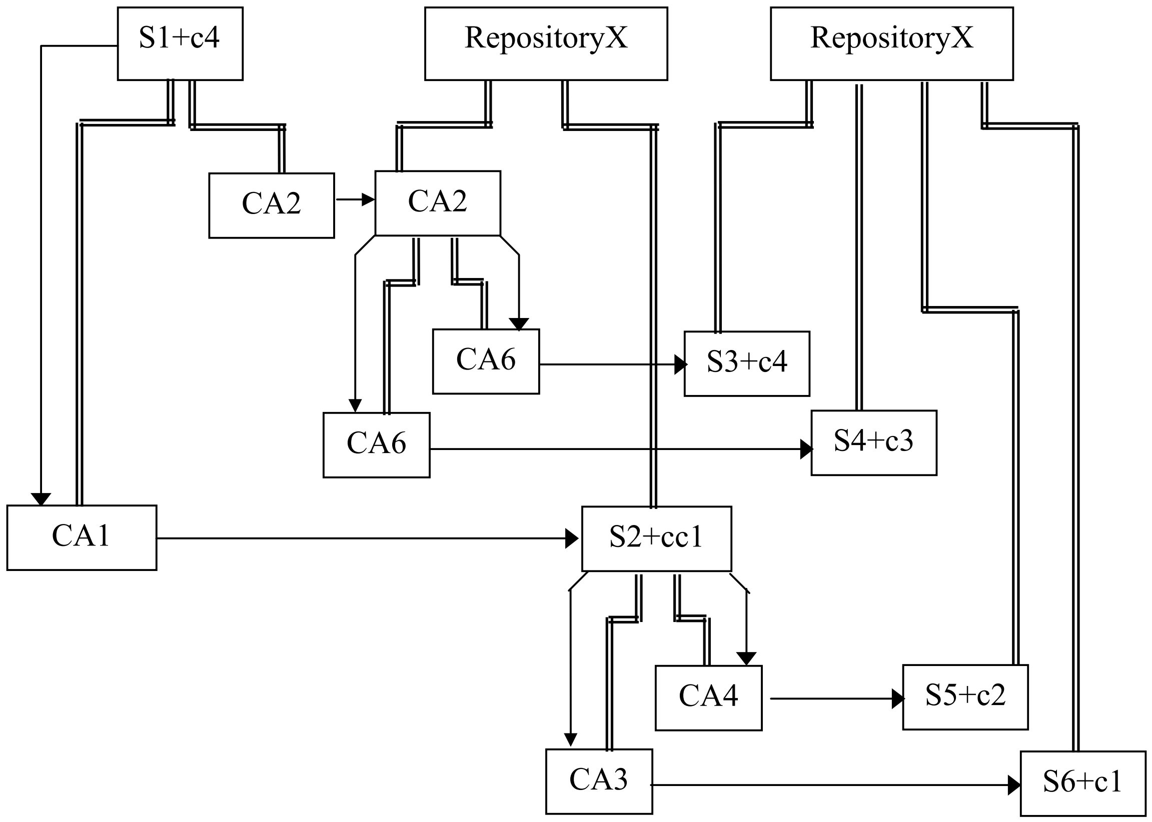

Figure 5. *Reusable conditions/actions and compound conditions/actions*

4.2 Applications, examples.

This section shows some policy examples based on the methodology of the Policy Core Information Model [Moore01] in the first case and from the point of view of the maagement application in the second case. .

4.2.1 Example 1

We present some policies examples that follow a set of steps to get an easy and adequate design that is going to be useful for an implementation in an object-oriented platform, for example Java or c++. In order having a good design, we follow four steps. First, it is necessary to describe the policy goal. Second step establishes a definition of the variables needed by the policy. Third step involves the definition of the set of conditions associated with a policy rule specifying when the policy rule is applicable. The notation is based on the Conjunctive Normal Form (CNF) and on the Disjunctive Normal Form (DNF) [Moore01]. The last step consists of defining the actions.

We use the BasicPolicyCondition defined in the Policy Core Information Model extensions (PCIMe) [Moore03]. This class models elementary boolean expressions of the form (<variable> match <value>). The variable specifies the attribute that should be matched when evaluating the condition. After the matching the value produce the boolean result to decide if the policy is going to be applied or not. PCIMe defines two types of PolicyVariables: Explicit Variables and Implicit Variables.

The last step performs the corresponding action or actions. The elementary action operation is (Set <variable> to <value>). We use some of them, but also self-defined actions, derived from the abstract class PolicyAction.

In order to facilitate the explanation of this section, next we present four policies examples: two service-level policies and two network level policies [Reyes02-2].

Policies from Service Level

Description of policy 1 goal. This policy compares bandwidth available on the link (BW_l) with the bandwidth required by the service (BW_r), if the network cannot guarantee this requirement then the network reject the service.

Definition of the variables needed by policy 1. The BW_r is in the SLS database as an ExplicitVariable (classname = SLS, attribute name = BW_r).

The available BW_l can be calculated in several ways, for example one possibility is when the network accepts to provide a service request, the requested parameters are stored in a database. There is also a record for indicating which requests achieve their destination. In that way we will have an approximation of current transport services in the system; summing the bandwidth assigned of each service, we obtain occupied bandwidth. After that it is necessary to subtract occupied bandwidth from total bandwidth in order to get bandwidth available for new transport services. Note that due to the lack of a variable-to-variable comparison feature in PCIMe, we need to define one by ourselves.

Definition of Conditions and Actions of policy 1

PolicyRule: IF $BW_l < BW_r$ then reject request

- BasicPolicyCondition
  - ExplicitVariable (BW_l = available)
  - Operator (Less than)
  - ExplicitVariable (class name =SLS, attribute name = bandwidth BW_r)
- Action (RejectRequest)

Description of policy 2 goal. This policy assigns Virtual Wire PDB (VWPDB) when the users or the services have high priority.

Definition of the variables needed by policy 2. User and service priorities are specified in SLA database. Available bandwidth in the VWPDB can be calculated with a similar procedure as in the policy example 1, it means that we have to sum the bandwidth of all transport services with VWPDB and later subtract it from the total assigned bandwidth to this PDB.

Definition of Conditions and Actions of policy 2

PolicyRule: If ((user priority = high) or (service priority = high)) and (VW PDB = available) then use VW PDB.

```
│
├── SimplePolicyCondition
│       ├─ImplicitVariable (VWPDB  exist)
│       └─BooleanValue (true)
├── SimplePolicyCondition
│       ├─ExplicitVariable (user priority)
│       ├─BooleanValue (true)
│       └─IntegerValue (high)
├── SimplePolicyCondition
│       ├─ExplicitVariable (service priority)
│       └─IntegerValue (high)
└── Action (Use VW PDB)
```

Policies from Network Level

Description of policy 3 goal. This policy determines if the network has an adequate technology to offer the requested service. This policy is also used to establish the range of parameters that the ISP is interested in offering. For example the ISP could be not interested in offering services with a very slow delay or with a very high rate.

Definition of the variables needed by policy 3. It is necessary a comparison between requested parameters ($delay_r$, BW_r, $lost_r$) and the highest and the lowest network parameters to determine if the technology is adequate. We have to add more conditions to compare the requested parameters with the established values by the ISP.

Notation: Virtual Wire PDB has the highest network parameters allowed: Bandwidth = BW VWPDB, delay = delay VWPDB, lost = lost VWPDB.
Bulk handling PDB has the lowest network parameters allowed: Bandwidth = BW BHPDB, delay = delay BHPDB, lost = lost BHPDB

If we want that the policy considers too the ISP values, then conditions should use a maximum value allowed ($BW_{isp-maximum}$, $delay_{isp-maximum}$, $lost_{isp-maximum}$) and a minimum value allowed ($BW_{isp-minimum}$, $delay_{isp-minimum}$, $lost_{isp-minimum}$)

Definition of Conditions and Actions of policy 3

PolicyRule: If (($delay_r$ > delay VWPDB) or ($delay_r$<delay BHPDB) or ($delay_r$ < $delay_{isp-minimum}$) or ($delay_r$ > $delay_{isp-maximum}$) or (BW_r > BW VWPDB) or (BW_r < BW BHPDB) or (BW_r < $BW_{isp-minimum}$) or (BW_r> $BW_{isp-maximum}$) or ($lost_r$ > lost VWPDB) or ($lost_r$< lost BHPDB) or ($lost_r$ < $lost_{isp-minimum}$) or ($lost_r$>$lost_{isp-maximum}$)) then reject request

- SimplePolicyCondition
  - ExplicitVariable (BW_r)
  - IntegerValue ((> BW VWPDB) or (< BW BHPDB) or (> $BW_{isp-maximum}$) or (< $BW_{isp-minimum}$))
- SimplePolicyCondition
  - ExplicitVariable ($delay_r$)
  - IntegerValue ((> delay VWPDB) or (< delay BHPDB) or (> $delay_{isp-maximum}$) or (< $delay_{isp-minimum}$)
- SimplePolicyCondition
  - ExplicitVariable ($lost_r$)
  - IntegerValue ((> lost VWPDB) or (< lost BHPDB) or (> $lost_{isp-maximum}$) or (< $lost_{isp-minimum}$)
- Action (RejectRequest)

Description of policy 4 goal. This policy calculates the number of rejections of Virtual Wire PDB and if it is bigger than a predefined value then the bandwidth assigned to Virtual Wire PDB is increased in 30% in the whole network in order to allow for more services be admitted in this class.

In order to increase the bandwidth of the VWPDB it is necessary to decrease the bandwidth of the Bulk handling PDB to maintain the proportion on capacity of the link.

Definition of the variables needed by policy 4. Rejections number of Virtual Wire PDB = rejections VW PDB. The predefined value to do the comparison is based on the network technology and can be specified only for the network operator.

Definition of Conditions and Actions of policy 4
PolicyRule: If (rejections VW PDB > x) then ((decrease 30% to BH PDB) and (add 30% to BW VWPDB))

```
│
├── SimplePolicyCondition
│       ├─ExplicitVariable (Rejection VW-PDB )
│       └─IntegerValue (> X)
│
├── Action (Decrease 30% BW assigned to
│              BH-PDB)
└── Action (Increase 30% BW assigned to
               VW-PDB)
```

In some cases, the performance of a Policy action generates new policies to solve specific situations. In that way, new policies can also generate more policies. This situation can produce several conflicts between policies or SLA/SLS inconsistencies or problems related to indefinite policy creation. .

Another kind of problem is when comes up situations that can not be solved with predefined policies, for example policy 4 action indicates an increment in the bandwidth assigned to Virtual Wire PDB. This action has as main consequence a decrement in the BH PDB bandwidth or/and in the AR PDB bandwidth. One problem could be for example, if it is not possible to decrease the bandwidth of any PDB, in this case the policy can not apply the action and it originates an inconsistency in the system. This is related with error handling in policy evaluation. Plus another problem lies in transactional relationship between the two actions. If one fails, the other does not need to be performed as well.

One way to avoid inconsistencies is doing previous test, for example in policy case number four, some tests help to know if it is already full assigned the bandwidth in the PDBs. If the bandwidth is not full assigned then it is possible to decrease the bandwidth in a PDB in order to increase the bandwidth in other PDB.

There are several inconsistencies that may come up in a system, some of them only can be solved by the network operator. In future work we are going to try to solve these system inconsistencies with dynamic policies that will depend on network state monitoring.

4.2.2 Example 2

In this case, both the information referring to a network's users and the different service policies that the service offers are stored in the following LDAP directory. Several QoS levels are defined, each one takes a specific rank of parameters.

QoSType	BW_Guaranteed	BW (Kbps)	CTDmax	CDVT	CLR
1	TRUE	2000			
2	FALSE	364			
...					

Table 3. Possible relation amon QoS and associarted parameters

This table can own as many columns as necessary. The Qos type column is the reference on which all the operations referring to QoS are based. A client's information is a structure formed by a series of alphanumeric fields.

Reference	Name	Password	Contact person	email	Account Number
CltReference	CltName	CltPassword	CltContact	CltEmail	CltAccount
...					

Table 4.Accounting parametrs

Row number 2 makes reference to the nomenclature used in IDL terminology. SLA agreed between the ISP and the different clients can be seen in the following table.

User Reference	QoS Agreed	SLA Status
CltReference(Tabla 2)	QoSType (Table 3)	Boolean On/Off

Table 5.Relations between user and SLAs and QOS

Every row in this table associates a service level (QoSType) to every client (CltReference). In the status column we can see the information about whether the account is active or the administrator has blocked it.

In order to achieve the application implementation, a node was defined as an undetermined whole of ports, where every port is represented by means of a data structure formed by a queue couple Capacity-Policy. Capacity makes reference to the port speed and the policy makes reference to the class of policy followed in the node (FIFO, etc.). The following IDL code represents that relation

```
typedef unsigned short QueuePolicy;
typedef unsigned short Capacity;        // Port speed, in Kbps
struct Port{
       QueuePolicy queue;
       Capacity    speed;
             };
typedef sequence<Port> Node;
```

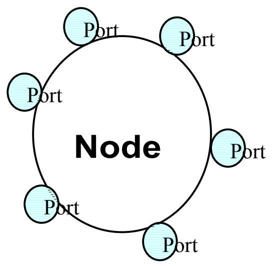

Figure 6. Node ports structure

Implementation allows the ORB to receive 6 events. However, the system is scalable when more events appear. The events used are described as follows:

```
const Evento NODE_DOWN = 0;
const Evento NODE_UP = 1;
const Evento PORT_DOWN = 2;
const Evento PORT_UP = 3;
const Evento PORT_CONGESTION = 4;
const Evento PORT_DECONGESTION = 5;
```

A session stays as defined by the following parameters:

- SessionId
- Host originating the connection
- Host destination
- QoS assigned. It coincides with one of the values in table 3.
- Path. It is a routers sequence in which the first item is the edge router.
- MPLabel. It is a label assigned to the connection by the MPLS protocol.
- Client. It is the reference of the user using the service. It must be one of the values of the CltReference column in table 3.

There is a special administration session, which stays as open by the system administrator. Therefore, the username and the password must be validated. From this session it is possible to manage all service policies and all user's data. ADM_SESSION is the identifier for this session. The functioning scheme is presented as shows figure 7.

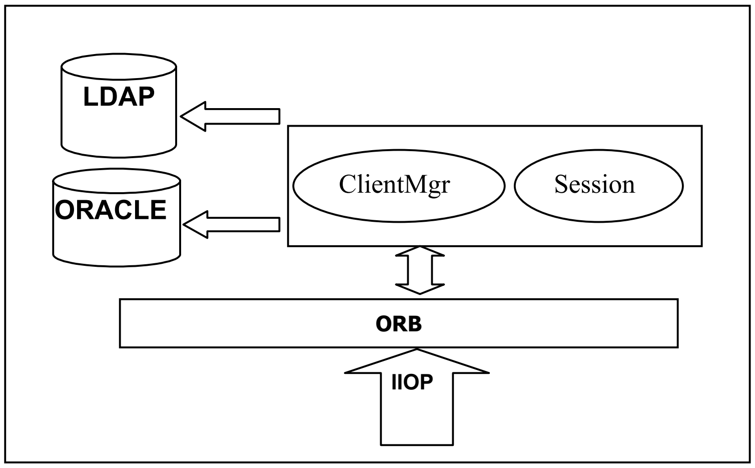

Figure 7. Corba management application

Interface Definition IDL File example

```
// -----------------------------------------------------------------------------------
//                      Management Application
// -----------------------------------------------------------------------------------
module mas {
// -----------------------------------------------------------------------------------
// Interfaces, types and excepcions
// -----------------------------------------------------------------------------------
   interface ClientMgr;
   interface Session;
   typedef unsigned short QoSType;
   typedef string PropertyName;
   typedef any PropertyValue;

   struct Property {
      PropertyName name;
      PropertyValue value;
   };

   typedef sequence<Property> PropertySeq;
   exception UnsupportedQoS{QoSType denied;};
   exception NoPrivileges{};
   exception UserBlocked{};
   exception NoSuchUser{};
   exception BadPassword{};
// -----------------------------------------------------------------------------------
// QoS issues
//   - Guaranteed Bandwidth (Boolean)
//   - Bandwidth (Kbps)
//   - Maximum delay peer to peer (CTDmax)
//   - Jitter (CDVT)
//   - Constant Losses Rate (CLR)
//
//    QosType: Takes integer values (0,1,2,3,4... ). It is the QoS identification offered to a specific service in
function of the Traffic parameters.
//
//   It is possible to add more features
// -----------------------------------------------------------------------------------
   struct QoS{
      QoSType Tipo;
      PropertySeq TrafficPar;
      };
// -----------------------------------------------------------------------------------
// User Issues.
// -----------------------------------------------------------------------------------
   typedef string CltName;
   typedef string CltPassword;
   typedef string CltContact;
   typedef string CltEmail;
   typedef string CltAccount;                  // Accounting Data
   typedef unsigned short CltReference;        // Customer Reference
   typedef string Router;          // Identification of routers via their IP address.
   typedef string Host;            // Identification of host via their IP address.
   typedef short Label;            // MPLS Label
```

```
    typedef sequence<Router> Route; // First, a route can have only one hop

    interface ClientMgr{
// ---------------------------------------------------------------------------------
// In OpenSession Invocation, the edge router information goes in the route parameter.
// This parameter only contains an edge router
// This invocation is an interface ClientMgr, which suppose the calculus of the path.
// The best path between the Host Source and Host target. This path has stored in the
// Session instance that come back the invocation to the function.
// ---------------------------------------------------------------------------------
        Session OpenSession(
            in CltName Nombre,
            in CltPassword Clave,
            in Host Source,
            in Host Destino,
            in Route Hops) raises (NoSuchUser,BadPassword,UserBlocked);
         };
// ---------------------------------------------------------------------------------
// Issues related to connections and Routing
// ---------------------------------------------------------------------------------
    typedef unsigned short QueuePolicy;
    typedef unsigned short Capacity;        // Port Velocity in Kbps
    struct Port{
        QueuePolicy queue;
        Capacity    speed;
        };
    typedef sequence<Port> Node;
// ---------------------------------------------------------------------------------
// There is a special management session that does not use any resource.
// ---------------------------------------------------------------------------------
    interface Session{
        typedef unsigned short SessionId;
        const SessionId ADM_SESSION = 0;
        typedef unsigned short Evento;  // Events that trigger an optimum path.
        const Evento NODE_DOWN = 0;
        const Evento NODE_UP = 1;
        const Evento PORT_DOWN = 2;
        const Evento PORT_UP = 3;
        const Evento PORT_CONGESTION = 4;
        const Evento PORT_DECONGESTION = 5;
        exception NoSuchEvent{};        // The event is unknown
        readonly attribute SessionId Id;
        readonly attribute Host Source;
        readonly attribute Host Destination;
        readonly attribute QoSType QoS ;
        readonly attribute Route   Hops;
        readonly attribute Label  MPLabel;
        readonly attribute CltReference Reference;
        Session Reconfigure(
            in Evento Alarma,
            in Node Enlace) raises (NoSuchEvent);
        void CloseSession();
         exception QoSFactoryError{};    // Error value for
                                // traffic descriptor
        void QoSCreate(in QoSType Tipo,in PropertySeq TrafficPar)
            raises (QoSFactoryError,NoPrivileges);
        void QoSDelete(in QoSType Tipo)
            raises (QoSFactoryError,NoPrivileges);
        void QoSModify(in QoSType Tipo,in PropertySeq TrafficPar)
            raises (QoSFactoryError,NoPrivileges);
        CltReference ClientCreate(
            in CltName Nombre,
            in CltPassword Clave,
            in CltContact Contacto,
            in CltEmail Correo,
            in CltAccount Cuenta) raises (NoPrivileges);
        void SetClientStat(
            in CltReference User,
            in QoSType QoSContratado,
            in boolean Active) raises (NoPrivileges,NoSuchUser,UnsupportedQoS);
        };
};
```

Server Side

A privileged user is defined to manage the system. This user is the only one that can carry out creation and destruction operations, and user’s and service policy modifications. The privileged user corresponds to the LDAP directory administrator and owns enough rights to read and write on the ORACLE database.

The main program creates an initial instance of the ClientMgr interface. The ClientMgr on receiving and processing an OpenSession operation successfully creates session interface instances. The CORBA Session object owns a defined CloseSession operation to release the resources used in the network.

After the Broker and initial Servant creation, the server waits for new invocations about all registered objects in it. These operations are the following ones:

In the ClientMgr interface

OpenSession. If parameters are right, a Session interface instance is returned. In addition to this, on being invocated, as all connection data are known, a first optimum path computation is carried out. There are three exceptions that can happen:

There is no user

The password is wrong

The client is disabled administratively

In the Session interface

If the administrator opens an administration session, this will own the ADM_SESSION identifier. Only in this case, invocations about the ClientCreate, SetClientStat, QoSCreate, QoSDelete and QoSModify operations will be able to be carried out. Otherwise, an invocation to these operations will generate a NoPrivileges{} exception.

ClientCreate. A row is added to Table 2. The only exception that can happen is the fact that the user invocating the operation does not own any privileges.

SetClientStat. It originates the values modification in Table 3, in which every user has a service quality and an administrative status (activated or deactivated) assigned. Exceptions happening can be the following ones: There is no user, the Session owner user does not own any privileges, there is no QoS specified in table 3.
QoSCreate. A row is added in table 3. Some possible exceptions happening can be either the absence of privileges or the fact that the specified traffic parameters are not adequate.

QoSDelete. It deletes a row in table 3. Some possible exceptions happening can be either the absence of privileges or the fact that the specified traffic parameters are not adequate.

QosModify. It modifies a row in table 3, that is to say, the traffic parameters of a specific service policy.

Reconfigure. A Session interface can receive an invocation of its reconfigure operation when there is some event in the network. This operation takes as a parameter the kind of event originated and the network node where it happened. As a result, the optimum path is computed again with the new network configuration and the Route and Label attributes are updates (if it is necessary according to the MPLS). The event must also be communicated to the edge router in order to renew its information. (Route and label for this session). The only exception we contemplate here is the fact of observing and notifying any other different event from the six ones we have defined until now.

CloseSession. It implies the destruction of this instance. Its implementation must contemplate the database ORACLE update that keeps the information about the network status.

4.3 Contribution in this chapter

In this chapter we show the implementation in a real system of the PCIM policy model using a LDAP repository. The use of a LDAP directory with a structure hierarchically distributed allows to extend the manage architecture to great networks without reducing their capabilities.

On the other hand, the fact of storing the network and services management information in directories and databases specified by ITU (LDAP/X.500) rules allows to extend the PBMS system easily, which was designed as an alternative planning to the one specified in the TMN network. This aspect has to do with the application for the management of other types of network based on ITU signals, as for example, mobile communication networks, B-RDSI, etc.

In this paragraph we want to mention the cooperation between the author of this Thesis and its director, Antonio Barba, together with other authors: M.Brunner, M.Pana and D.Morón in the specification of Draft [Reyes03] corresponding to the IETF, which especially supports all contributions in this chapter.

References

[LPDL] Yongxin Li Ming Chen Xuping Jiang Lihua Song. A Logic-based Policy Definition Language for Network Management. O. Festor and A. Pras (Eds.). 12th International Worshop on Distributed Systems: Operations and Management DSOM'2001 France, 2001.

[2-LPDL] G.Stone, B.Lundy, G.Xie, Network policy languages: a survey and a new approach, IEEE Network, January/February, 2001

[Reyes 03] Reyes, A., Barba A., Moron, D., Brunner, M., Pana M. *Policy Core Extension LDAP Schema (PCELS)*, IETF Internet Draft. February 2003.

[Reyes02-2] Angélica Reyes, Marcus Brunner, Antoni Barba. Controlling IP Network Management Systems via Policies CIIT **2002 IASTED**- IEEE St Thomas, USA.2002

[Strassner02] J. Strassner, B. Moore, R. Moats E. Ellesson. *Policy Core LDAP Schema* IETF Internet-Draft. October 2002.

Standards

[CIM-DMTF] CIM Core Policy Model White Paper. March 14, 2003 http://www.dmtf.org/standards/documents/CIM/DSP0108.pdf

[Moore01] Moore, E. Ellesson, J. Strassner, A. Westerinen. *Policy Core Information Model-- Version 1 Specification.* IETF Request for Comment (RFC) 3060. February 2001.

[Moore03] B. Moore, Ed.. *Policy Core Information Model (PCIM) Extensions.* IETF Request for Comment (RFC) 3460. January 200