IMPLEMENTATION OF A POLICY BASED NETWORK FRAMEWORK USING METAPOLICIES

A FINAL YEAR PROJECT BY

Alvaro Fernandez Casani

DECEMBER 2001

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Prof. Raouf Boutaba
Broadband Communications Research Group
University of Waterloo,
Canada

Prof. Josep Bernabeu
DSIC
Politechnical University of Valencia,
Spain

To my parents and sister, for all their love and support.

Table of Contents

1 INTRODUCTION

1.1 PURPOSES AND GOALS

This project is included into a bigger research, whose final objective is the creation of self-configurable networks, more independent that the ones that we know nowadays. In order to achieve that objective it is necessary to modify some issues of the actual networks.

First of all it is necessary to raise the level of abstraction of the management methods and network administration, so a higher level of integration and, above all, automation, can be allowed.

Second, intelligence must be pushed towards the managed devices, so that the control of that network is more distributed and decentralized. For this issue, it is necessary to modify the behavior of those components of the network.

These two objectives, a higher level of abstraction in combination with more intelligent decision points (that could be configured by themselves by getting or generating such configuration data); and so on, a higher level of automation, will give the necessary methods to adapt the network state and needs at each specific moment.

Policy-based networking (PBN) is a relatively new trend in the area of Network Management that accomplishes our first objective, because it raises the abstraction of Network Management by using high-level policies, from which configuration data for the network devices are automatically generated and distributed to the network devices. However, PBN doesn't achieve to address the second requirement. Although PBN is not a very centralized network-management model because it uses policy servers that can be distributed over the network; not much functionality is pushed toward the network devices, so these devices are not as intelligent and independent. Those devices need the constant attention of the policy servers to accomplish its functions.

The final objective of this project is the implementation of a PBN framework introducing a new kind of control point inside the network, enhancing the PBN by developing in the second dimension. More concretely, the implementation of a new kind of control points on the framework of the policy-based networks, so these control points (called PEP or Policy Enforcement Points) will be more independent in their decisions.

1.2 STATE OF THE ART

The area of the network management has been one of the most actives in the last years into the work done by the IETF, working in different issues that have led to many improvements to the original conception. Understanding this part is very important to understand the realized work, so we will dedicate the first chapters of this report to explain the current situation of this promising area of the networking.

1.3 DEPENDENCIES

This work has been developed according to the specifications of some documents published by the IETF. At the time of this work was conducted, some of the proposed standards like COPS and COPR-PR were RFCs, and SPPI was an Internet-draft (refer to the chapter of IETF). Hence, it must be understood that as those documents are not official standards, they can be changed in the future, so maybe some parts defined in this report could be syntactically out-of-time. Anyway, the major part of the definitions is considered stable, so the possible changes to apply to this document, to be compliant to the future standards, should be minimal.

2 NETWORK MANAGEMENT

In this second chapter we talk about Network Management, introducing the required concepts to make the reader familiar with all the concepts of this area of the networking.

2.1 INTRODUCTION

The formal definition of Network Management is:
The execution of the set of functions required for controlling, planning, allocating, deploying, coordinating, and monitoring the resources of a telecommunications network, including performing functions such as initial network planning, frequency allocation, predetermined traffic routing to support load balancing, cryptographic key distribution authorization, configuration management, fault management, security management, performance management, and accounting management.

Network Management starts with the design of the underlying network, and after this initial phase, continues with the maintenance tasks that collect and analyze data from the various network elements. This data can reveal abnormal or emergency situations as soon as – or even before – they occur. Also, these data allows the administrators to monitor the usage of the network resources, and according to it, fine-tune the network for improvements, and also plan future upgrades.

2.2 FCAPS FRAMEWORK

As we have said, Network Management groups many different issues, and may be divided into functional areas. The Organization for International Standardization (ISO) has defined the next major areas: Fault, Configuration, Accounting, Performance and Security Management.

This five areas are know as the FCAPS framework [7].

Fault Management: is in charge of detecting, isolating, fixing, and recording errors that occur inside the network.

Configuration Management: relates with the configuration of the network elements (software and Hardware), as well as the tuning and controlling parameters that relate to the normal network operations

Accounting Management: has to do mainly with the user management, as well as the accounting and billing for the used services and resources.
Performance Management: tries to maximize the network performance, and of course is mainly in charge of QoS (Quality of Service) provisioning and other factors like resource utilization, delay, jitter, and packet loss.
Security Management: is in charge of the security and safety of the network.
This project is related mainly with the area of Configuration Management, but also covers implicitly all the commented five areas, since all of them relate to the appropriate configuration of the network devices.

2.3 STANDARDIZATION

The Internet Engineering Task Force (IETF) [15] is " the protocol engineering and development arm of the Internet". Established in 1986, it is "a large open international community of network designers, operators, vendors, and researchers, concerned with the evolution of the Internet architecture and the smooth operation of the Internet".

In IETF each new specification is published as an internet-draft. These draft are widely available, have no formal status, are subject to removal at any time and evolve according to the comments and feedback that they receive from the internet community. If an internet-draft receives enough attention, becomes relatively stable and mature and is globally approved, it evolves into an RFC (Request For Comments). The RFC is an official document that describes the specification in a complete well-understood manner, and is approved by the majority of the Internet community. As with internet-drafts, RFCs do evolve, however the modifications are usually moderate. When the RFC has reached a state where no more modifications are considered necessary, it may evolve into an Internet standard.

IETF host various working groups that cover different areas (e.g. routing, transport, security, etc.). These groups identify problems in the corresponding areas and address them by developing standard protocols.

IETF is closely related to other Internet organizations, such as the Internet Engineering Steering Group, the Internet Architecture Board (IAB), the Internet Assigned Numbers Authority (IANA) and the Internet Society (ISOC).

Also IETF plays a crucial role in the evolution of the Network Management, since several of its working groups are related to it. For instance, IETF is the organization that has standardized the SNMP protocol. IETF attempts now to address the lacking issues of the SNMP through its next version (SNMP v3). However, there are serious doubts about whether SNMP will eventually manage to overcome its limitations and become the dominating protocol for Configuration Management again. This is way IETF also attempts to develop alternative management techniques that may replace or complement the existing ones.

2.4 TRADITIONAL NETWORK MANAGEMENT: SNMP

The network management of the configurable network devices (such routers and switches) has gone through different stages.

At the beginning the network administrator was in charge of manually configure each one of those devices, through the command line interfaces. In many cases each of the devices was configured independently, even when these were configured to operate similarly.

After, when the networks began to grow in number and complexity, some kind of automation became necessary. The Simple Network Management protocol gave a satisfactory solution to the problem.

SNMP is based on special databases, called Management Information Bases (MIBs), maintained by each network device. MIBs provided a standard interface to manage objects on the devices, in a less device-dependent way. This raised the level of abstraction and allowed devices to be handled in a more unified way. SNMP allowed the administrators to manage the network remotely and to automate various management tasks.

2.4.1 SNMP Technology - The Internet Management Model

SNMP is part of the Internet network management architecture. This architecture is based on the interaction of many entities, as we describe in the following paragraph.

As specified in Internet RFCs and other documents, a network management system comprises:

Network elements -- Sometimes called managed devices, network elements are hardware devices such as computers, routers, and terminal servers that are connected to networks.

Agents -- Agents are software modules that reside in network elements. They collect and store management information such as the number of error packets received by a network element.

Managed object -- A managed object is a characteristic of something that can be managed. For example, a list of currently active TCP circuits in a particular host computer is a managed object. Managed objects differ from variables, which are particular object instances. Using our example, an object instance is a single active TCP circuit in a particular host computer. Managed objects can be scalar (defining a single object instance) or tabular (defining multiple, related instances).

Management information base (MIB) -- A MIB is a collection of managed objects residing in a virtual information store. Collections of related managed objects are defined in specific MIB modules.

Syntax notation -- A syntax notation is a language used to describe a MIB's managed objects in a machine-independent format. Consistent use of a syntax notation allows different types of computers to share information. Internet management systems use a subset of the International Organization for Standardization's (ISO's) Open System Interconnection (OSI) Abstract Syntax Notation 1 (ASN.1) to define both the packets exchanged by the management protocol and the objects that are to be managed.

Structure of Management Information (SMI) -- The SMI defines the rules for describing management information. The SMI is defined using ASN.1.

Network management stations (NMSs) -- Sometimes called consoles, these devices execute management applications that monitor and control network elements. Physically, NMSs are usually engineering workstation-caliber computers with fast CPUs, megapixel color displays, substantial memory, and abundant disk space. At least one NMS must be present in each managed environment.

Parties -- Newly defined in SNMPv2, a party is a logical SNMPv2 entity that can initiate or receive SNMPv2 communication. Each SNMPv2 party comprises a single, unique party identity, a logical network location, a single authentication protocol, and a single privacy protocol. SNMPv2 messages are communicated between two parties. An SNMPv2 entity can define multiple parties, each with different parameters. For example, different parties can use different authentication and/or privacy protocols.

Management protocol -- A management protocol is used to convey management information between agents and NMSs. SNMP is the Internet community's de facto standard management protocol.

The most basic elements of the Internet management model are graphically represented in Figure 1.

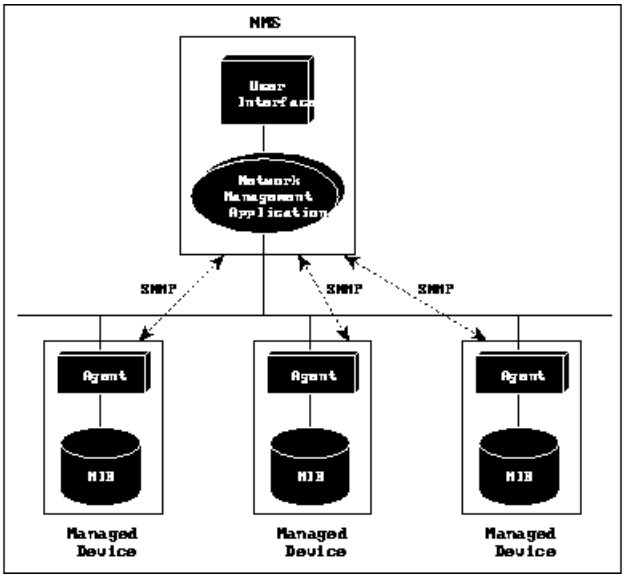

Figure 1:The Internet Management Model

Interactions between the NMS and managed devices can be any of four different types of commands:

Reads -- To monitor managed devices, NMSs read variables maintained by the devices

Writes -- To control managed devices, NMSs write variables stored within the managed devices

Traversal operations -- NMSs use these operations to determine which variables a managed device supports and to sequentially gather information from variable tables (such as IP routing tables) in managed devices

Traps -- Managed devices use traps to asynchronously report certain events to NMSs

However, SNMP (v1 and v2) was designed mainly for monitoring purposes and, although it managed to give a satisfactory solution to the problem for a while, now it seems to suffer from significant scalability and efficiency problems: SNMP is a highly centralized protocol. In fairly large networks, too many resources may be consumed just to report normal network operation, while the detection of erroneous events and the reaction to them may be too slow. Besides, although SNMP managed to raise the level of abstraction in Network Management, the operations are still device-dependent. The growth of the modern networks demands a further increase in the level of abstraction, s well as decentralization of the management centers. These issues are examined by standardization organizations (such as IETF) which guide the future of Network Management.

2.5 FUTURE OF NETWORK MANAGEMENT

The network management techniques annotated before became to be insufficient for the necessities of the modern issues, so future of network management goes through different techniques to manage the new requirements.

Some of the new techniques that would maybe be established in the future are **SNMPv3, Active Management, Directory enabled networking, and Policy based networking.**

2.5.1 SNMPv3

SNMPv3 was proposed as a draft in March of 1999 by the IESG to solve the lacks of SNMPv2. This new proposal tried to unify the different versions of SNMPv2 that were deployed until that moment. One of the biggest design requirements was the addition of security and management functionality into the previous standards.

The standardization of SNMPv3 as the only management protocol is insecure, because of this protocol does not address very well questions like scalability. However, since one of most achievements of this (and the previous related) protocols, is the extended use and its simplicity, it is believed that it will come one of the used management protocols, at least in questions of monitoring.

2.5.2 Active Management

The Active Networks program has the goal of producing a new networking platform, flexible and extensible at runtime to accommodate the rapid evolution and deployment of networking technologies and also to provide the increasingly sophisticated services demanded by modern applications.

The active networks are differentiated from the traditional ones, from the fact that the nodes (routers, switches...) can be programmed by the end-user and define the right behavior that wants to be performed by that node. This opens a great bunch of possibilities because is not necessary to follow standard protocols, that could not issue the characteristics that the end-user wants. Injecting user programs in the nodes, the way of handle day may be modified per-application or even per-user basis.

Active networks introduce radical changes, also in network management, being the main issues the next that we describe:

- Active networks enable the distribute management by agents that travel thru the network and perform management actions related with the application that introduced the agent.
- One of the most important characteristics is that the network can be managed during abnormal functioning, for example during high congestion or network partition
- The network, moving the decision taking closer to the managed devices, can distribute also monitoring centers.

Although Active networks is not directly related with this project, it has influenced in part with design decisions and ideas that have arose along the conduction of this project.

2.5.3 Directory-Enabled Networking

Directories are nothing new into networks, since nowadays we can find services based in directories as DHCP, DNS, user authentication, user directories, etc. But with directory-enabled networks what it is tried to accomplish is to unify all that services, and make user management easier and more consistent. Although the resemblance of that directory servers and common Database Management Systems (DBMS) is very

remarkable, these such differences are the ones that make different Directory-enabled networks. Hence directory servers are optimized for high quantity of information retrieval, since that is the main operation against them. Another issues like updates, or advanced characteristics of the DBMS like triggers, cascade deleting, etc., have much less performance.

2.5.4 Policy-Based Networking

One of the most promising techniques developed in the last years in the area of the network management is the Policy-Based networking. This new technology is designed from a higher level of abstraction over the traditional management techniques, and its based in policies, that is, high-level rules that determine the behavior of the nodes, and the network in consequence.

The key idea is that the network administrator creates high level policies that determine the desired goals of the networks (rather than procedures). These policies are processed by special serves, which, bind them with the current Network State, transform them into dynamic configuration data and send them to the network devices such as special router, determining his behavior.

With this high-level abstraction model we have the advantage to simplify management of large-scale networks, and the automation ensures that the consistency and integrity in the devices behavior across the network. The dynamic binding of policies at the policy servers allows net types of policies to be introduced more easily, as well.

We will dedicate next chapter to introduce deeply the policy-networking framework.

3 POLICY BASED NETWORKING

In this chapter we introduce the policy-based networking, explaining its principal characteristics and all the issues necessary to understand the concepts that we will use later in the design of the framework. Also we will briefly introduce new trends that are being developed in this area.

3.1 INTRODUCTION

Policy-Based Networking is a new trend that has been started developed in the last years and has arisen as a pledge technology for network operation and management. It is based in high-level policies that define the desired behavior of the network. This policies define *what* the network is supposed to do, rather that *how* is it going to do that. The way that the high policies are transformed in understood, we will see it later.
In PBN there are defined two main components that are named as Policy Decision Point (PDP) and Policy Enforcement Point (PEP). Basically the policies are produced in the PDPs to be distributed to the PEPs, that enforce them to obtain the desired configuration from the network.
The policy concept is nothing innovative, nevertheless, what it is new in PBN is that the policies express goals rather than procedures.
While in traditional network management, the administrators know and set the goals, and after they create procedural policies that implement that policies; in PBN the policies is set as the goal itself.
For example, in traditional networking management, if the network administrator wants to give high priority to the manager subnet, he/she created a policy similar to the following:

If ((SourceIp matches 11.22.33.0/24) or (DestinationIp matches 11.22.33.0/24)) then { remark with DSCP = 6 }

This policy has been hardcode the facts that:
1) the manager subnet is 11.22.33.0/24
2) high priority is achieved by setting the packet's DSCP[1] to 6

The new way to define the policies, using the PBN approach, would be the next example:

If ((SourceIp matches ManagerSubnet) or (DestinationIp matches ManagerSubnet)) then {give high priority}

To manage the abstractions shown in the last example, in PBN the system administrator has to provide a way to interpret all the variables as *ManagerSubnet* or how to give *high priority*. The difference is that the information is not hardcode in the policies themselves. Hence if the manager subnet is expanded to contain the subnet 11.22.34.0/24, the administrator will only need to declare this fact. All policies related to this subnet will still be valid, since they do not contain information directly related to the network topology or the devices.

If policy is to be readable, and most importantly, manageable, it must break through the complexity of traditional networking terminology such as Command Line Interfaces (CLI) syntax. Policy rules must be relatively painless to define and maintain on an ongoing basis.

Once defined, network policy is fed into a policy system. Which converts it into sets of discrete, specific, low-level networking command. This automated translation is not tied to a specific network topology. Therefore, It has the capability to adapt to even the most massive re-engineering of the network without requiring major changes to the underlying policy. Several common high level criteria used in policy expressions are User-id, User-Group, Application, Application-Type, Time/Date, and Device Type/Role.

Finally, we can say that a policy based networking system consists of tools designed to accomplish the following tasks:

- Creating rules and policies
- Checking for policy conflicts
- Storing policies
- Converting policies into device –specific command
- Distributing the commands to the network devices
- Verifying policy distribution
- Streamlining policy implementation
- Facilitating troubleshooting
- Enabling group policy deployment

1 DSCP (Differentiated Services Code Point): In differentiated services, the packets receive different treatment by the switching devices, according to the TOS field of the IP header (also named DS byte in Differentiated Services terminology). Six of its bits are used as a Differentiated services code point

3.2 COMPONENTS

In this point we describe the components of a policy based network framework.

3.2.1 Logical Components: POLICIES

A policy is a set of rules that guide and determine how to manage, allocate, and control network resources. Enforcement of policy ensures that rules are always followed. A rule is made up of a condition that leads to an action. A condition is a set of expressions or objects used to determine whether a given action should be performed. A condition answers the question, "when and where do we enforce a policy?" For example, a condition might be "all e-mail traffic." An action is a definition of what must be executed in order to enact a rule. An action answers the question, "what must be done to enforce a policy?" An action could be "encrypt e-mail using 3DES." Together, the condition and action pair result in the business rule "all e-mail traffic must be encrypted." A policy also defines how the network's resources are to be allocated among its clients. Clients can be individual users, departments, host computers, or applications. Resources can be allocated based on time of day, client authorization priorities, availability of resources, and other factors. How resources are allocated can be static or dynamic (based on variations in traffic). Policies are created by network managers and stored in a repository. During network operation, the policies are retrieved and used by network management software to make decisions.

3.2.2 Architectural Components

Defined by the Distributed Management Task Force (DMTF) and the Internet Engineering Task Force (IETF), the standard tools in a policy-based infrastructure used to accomplish the system tasks are:

- **Policy Console:** The administrative client interface to the policy system.
- **Policy Repository:** A directory service or database used to store policy data.
- **Policy Decision Points (PDP's):** Make policy decisions based on policy data.
- **Policy Enforcement Points (PEP's)**: Enforce policy decisions.

The **Policy Console** is the interface between the network administrator and the system. It's primarily used to enter and edit policies and monitor the status of the network. It also simplifies the management process by allowing network administrators to group certain items together, such as users or printers, and uses simple English commands that are later translated into device configuration commands. The policy console checks and validates the rules, making sure rules don't conflict when they are combined to make a policy. It also translates the rules to match pre-defined storage schemas in the policy repository.

The **Policy Repository** is typically a directory service that stores the rules and policies generated in the policy console. The repository typically also contains other

network information, such as user profiles and IP infrastructure data, which gives network administrators the ability to aggregate policies that encompass large groups of objects (such as devices, applications, or users) or specific groups of objects.

The **Policy Decision Point (PDP)** is responsible for accessing the policy schemas stored in the policy repository and making decisions based on policy information. For instance, the policy is set to use the strongest encryption when sending finance department documents. Therefore, the PDP would set up appropriate IPsec tunnels with the strongest encryption. The PDP can also detect policy changes and conflicts and act accordingly to correct any problems. Finally, the PDP logs events from network devices and monitors network usage. It can even be set up to create new policies based on this network state information.

Policy Enforcement Points (PEP's) are the actual network devices that implement and enforce the policies. PEP's include devices such as routers, VPN gateways, and firewalls. The PEP gathers the policy information from the PDP upon startup and stores this information in its cache. The PEP may also relay information to the PDP to keep the PDP informed of changes in the network or device conditions.

3.3 OUTSORCING AND PROVISIONING MODELS

As defined by IETF, PBN is designed for working in two different modes for policy management: *Outsourcing and Provisioning* model.

The outsourcing model the PEP receives a signaled event in that must be resolved based on policy criteria that is known as Policy Admission Control (PAC), so when the PEP is required to make a decision through this model and cannot treat this event according to the installed configuration data, it *outsource* the decision-making to an external policy decision point (PDP). The PDP will reply to the PEP sending the appropriate data that will be installed by the PEP and will treat the event. Signaling events are usually associated with end-to-end signaling protocol (such as RSVP, MPLS-LDP, Multicast Join ICMP, etc.) However, a signaled event at the PRP is decided, innocence, based upon external considerations. This mode is also known as the *'pull'* model since the PEP 'pulls' configuration data from the PDP, or the 'reactive; model, because the PDP reacts to the PEP requests.

In the provisioning model we have almost the opposite model, where the PDP predicts future configuration needs and pushes configuration information in the PEP. This occurs when the PEP connects to the PDP. The policies are stored in the pep and all incoming events are served according to them. This model is also known as the *''push''* model, since, as we have said, the PDP pushes the information to the PEP.

3.4 DIFFERENCES BETWEEN DIRECTORIES AND PBN

Although policy-based networks and directory enabled networks, share many points in its concepts, but are at the end different. Two of them are based in raising the level of abstraction of the configuration into the networks, and two of them provide the appropriate configuration to the network devices; but the main difference is the kind of this configuration, that while in the directory enabled networks is dynamic, in the PBN is dynamic.

For example in the PBN typical example, it exists a LDAP server that provides the static policies to the PDP. One of these static policies could be for example 'Staff has high priority in working hours'. Directories are very good in performance handling this kind of static data, as they work as typical SMDBs. Directories are good also for providing static data to the network devices, as the addressed of the PEP that controls that device, DNS servers, etc.

But the question is that in PDP the static data is transformed into dynamic configuration data, because for example has to check what 'Staff' is, or what kind of 'high priority' is possible with the current state of the network. Also the policy is dynamically installed when the *working hours* starts, and dynamically removed when it ends.

As we see the most complex thing is to transform the high-level configuration data, according to the Network State, into low-level configuration data understood by the network nodes.

3.5 BENEFITS OF POLICY BASED NETWORKING

PBN simplifies the task of the network administration based on the goals of the system administrator. Hence the user can express the behavior of the network in terms of high level *goals* that is desired to achieve. The system will analyze that goals or policies and will manage the translation of these terms into commands or configuration parameters that can be understood by the network nodes.

This way is possible to eliminate at the *design* phase, some of the details that will be present in the end configuration parameters, like for example, the exact bindings to user information such as IP address, application port number, ingress interface, etc. This binding, applied at packet level, is performed automatically by the system at the required point in time.

The automated coordination and dynamic binding also permits to map to configuration information such as packet handling methods, queuing mechanisms, and link capacity based on service class. This kind of *mechanisms* can be supported or not by the network nodes, so the automated generation of configuration information will be different depending on the supported methods. In addition, different devices can require different configurations to enforce the same policy, so PBN must match the required goals with the possible methods to generate the final configuration information.

Summarizing, these are the main benefits of PBN:

- **High degree of abstraction:** policies are written in high level, human-friendly terms; independent of the network topology, protocols, services and applications. Policies are explained by themselves.
- **Automation:** all the process of the network management is handled by the network elements (PDP, PEP) so the automation is complete. The system can detect changes in the network state, topology, etc. and will try to automatically achieve the goals described in the policies
- **Consistency:** as a result that all is part of an automated process, the consistency is guaranteed.
- **Dynamic policies:** for example a dynamic binding of a policy is the following: suppose that the policy "Managers have high priority" have been set. When an engineer logs on to a workstation, the PDP is informed of this fact and generates such configuration data for the network devices, that will give high priority to the specific workstation. That kind of policies is very hard to implement following traditional network management techniques.

3.6 PBN PROTOCOLS

PBN architecture uses a protocol named Common Open Policy Service (COPS) to send the appropriate configuration information from the PDP to the network devices (PEP). Whether COPS and its extensions are being developed by the Resource Allocation Protocol (RAP) working group.

This protocol allows the devices to efficiently provide feedback to the PBN system regarding the state of the network. This is an essential component for dealing with dynamic changes in the network since policy rules may need to be altered or added.

3.6.1 COPS (Common Open Policy Service) Protocol

The COPS protocol was defined by the IETF [15] as a protocol for exchanging policy information between a policy server (Policy Decision Point or PDP) and its clients (Policy Enforcement Points or PEPs).
In our project a PDP can be a computer that contains all the policies that are desirable to transmit to a PEP, and that PEP would be the router that has to enforce that policies.
The typical example of a policy client is an RSVP router that must exercise policy-based admission control over RSVP usage [23].
The basic mode of interaction between a policy server and its clients is compatible with the framework document for policy based admission control.

The characteristics of the COPS protocol are mainly:

1) The model employed for the protocol is a typical Client/Server model where the PEP sends requests, updates and deletes to the remote PDP, and the PDP is in charge of returning correct decisions to the PEP

2) The protocol uses TCP as a transport protocol for exchange of messages between policy clients and server. Therefore, as it is using a reliable protocol, no additional mechanisms are necessary for guarantee the correct communication between PDP and PEPs
3) Is a extensible protocol. It is designed to leverage off self-identifying objects and can support diverse client-specific information without requiring modifications to the COPS protocol itself, only extensions (as we will see with COPS-PR). The protocol was created for the general administration, configuration, and enforcement of policies.
4) COPS provide message level security for authentication, replay protection, and message integrity, and also can reuse existing protocols for security such as IPSec [34].The protocol is stateful in two main aspects:
   - Request / Decision state is shared between client and server: requests from the PEP are installed or remembered by the remote PDP until they are explicitly deleted by the PEP. At the same time, Decisions from the remote PDP can be generated asynchronously at any time for a currently installed request state
   - State from various events (Request / Decision pairs) may be inter-associated: the server may respond to new queries differently because of previously installed Request / Decision States that are related

Additionally, the protocol is stateful in that it allows the server to push configuration information to the client, and then allows the server to remove such state from the client when it is no longer applicable

COPS does not define the format and semantics of the exchanged configuration data: it just provides a protocol to exchange that data. Semantics and format of the data will be defined in a client-type basis (as we know that PEP may support different kinds of clients) in other documents that extend the COPS protocol.

3.6.1.1 COPS Working mode

In a typical COPS framework the PEP is responsible for initiating the communication with the PDP in the form of a TCP persistent connection. The PEP uses this TCP connection to send requests to and receive decisions from the remote PDP. Communication between the PEP and the remote PEP is mainly in the form of a stateful request / decision exchange, thought the remote PDP may occasionally send unsolicited decisions to the PEP to force changes in preciously approved request states. The PEP also has the capacity to report to the remote PDP that it has successfully completed performing the PDP's decisions locally, useful for accounting and monitoring purposes. The PEP is responsible for notifying the PDP when a request state has changed on the PEP. Finally, the PEP is responsible for the deletion of any state that is no longer applicable due to events at the client or decisions issued by the server.

When the PEP sends a configuration request (one of the first actions that it accomplish), it expects the PDP to continuously send named units of configuration data

to the PEP via decision messages as applicable for the configuration request. When a unit of named configuration data is successfully installed on the PEP, the PEP should send a report message to the PDP confirming the installation. The server may then update or remove the named configuration information via a new decision message. When the PDP sends a decision to remove named configuration data from the PEP, the PEP will deleted the specified configuration and send a report message to the PDP as confirmation.

The policy protocol is designed to communicate self identifying objects which contain the data necessary for identifying request states, establishing the context for a request, identifying the type of request referencing previously installed requests, relaying policy decisions, reporting errors, providing message integrity, and transferring client specific/ namespace information.

To distinguish between different kinds of clients, the type of client is identified in each message. Different types of clients may have different client specific data and may require different kinds of policy decisions. It is expected that each new client-type will have a corresponding usage draft, specifying the specifics of its interaction with this policy protocol.

The context of each request corresponds to the type of event that triggered it. The COPS Context Object identifies the type of request and message (if applicable) that triggered a policy event via its message type and request type fields. COPS identifies three types of outsourcing events:

- The arrival of an incoming message
- Allocation of local resources
- Forwarding of an outgoing message

Each of these events may require different decisions to be made. The content of a COPS request/ decision message depends on the context. A fourth type of request is useful for types of clients that wish to receive configuration information from the PDP. This allows a PEP to issue a configuration request for a specific named device or module that requires configuration information to be installed.

The PEP may also have the capability to make a local policy decision via its Local Policy Decision Point (LPDP) [WRK] however, the PDP remains the authoritative decision point at all times. This means that the relevant local decision information must be relayed to the PDP. That is, the PDP must be granted access to all relevant information to make a final policy decision. To facilitate this functionality, the PEP must send its local decision information to the remote PDP via an LPDP decision object. The PEP must then abide by the PDP's decision, as it is absolute.

Finally, fault tolerance is a required capability for this protocol, particularly due to the fact it is associated with the security and service management of distributed network devices. Fault tolerance can be achieved by having both the PEP and remote PDP constantly verify their connection to each other via keep-alive messages. When a failure is detected, the PEP must try to reconnect to the remote PDP or attempt to connect to a

backup/alternative PDP. While disconnected, the PEP should revert to making local decisions. Once a connection is reestablished, the PEP is expected to notify the PDP of any deleted state or new events that passed local admission control after the connection was lost. Additionally, the remote PDP may request that all the PEP's internal state be resynchronized (all previously installed requests are to be reissued). After failure and before the new connection is fully functional, disruption of service can be minimized if the PEP caches previously communicated decisions and continues to use them for some limited amount of time. We will discuss some mechanisms for achieving reliability in next sections.

3.6.1.2 Cops Defined Protocol

The Cops protocol consist in a number or objects and messages that comprise that messages in an ordered way, to achieve a semantic framework that is able to convey the policy data.

For a more deep review of the protocol you can refer to [25].

3.6.2 COPS-PR (COPS for Policy Provisioning) Protocol

As an extension of the Cops protocol, surged the Cops-Pr protocol, oriented for policy provisioning. Although, this protocol is independent of the type of policy being provisioned (QoS, Security, etc.), nevertheless we will use this protocol for conveying the policy information among the PDP and PEP entities.

Cops-Pr has been designed in a framework for conveying efficiently the policies, affording good performance in the transport of attributes, large atomic transactions of data, and efficient and flexible error reporting. It also guarantees only one server updates a particular policy configuration at any time, since it has only one connection between the PDP and the PEP, identified by the client.

3.6.2.1 Cops-Pr working mode: Interaction between PDP and PEP

As we know when a PEP boots up, it tries to get a new connection to its primary PDP, sending information about itself to the PDP in the form of configuration request. This information includes client specific information.

The PDP will download the provisioned policies that are currently relevant to that device, and the latter will map them into local mechanisms, installing them.

If the PDP notices of a change that is necessary to inform to the related PEP, it will issue the right install, updates and deleted to that device.

Also, if the PEP changes its configurations (adding new hardware, changing software...) in ways not covered by policies already known to the PEP, then the PEP asynchronously sends this unsolicited new information to the PDP in an updated configuration request. On receiving this new information, the PDP sends to the PEP any

additional provisioned policies now needed by the PEP, or removes those policies that are no longer required.

3.6.2.2 Policy Information Base (PIB)

The PEP has a named data structure called *Policy Information Base (PIB)* that maintains the policy information. The protocol assumes a named data structure, known as a Policy Information Base (PIB), to identify the type and purpose of unsolicited policy Information that is "pushed" from the PDP to the PEP for provisioning policy or sent to the PDP from the PEP as a notification. The PIB name space is common to both the PEP and the PDP and data instances within this space are unique within the scope of a given Client-Type and Request-State per TCP connection between a PEP and PDP. Since a given a device might implement multiple COPS Client-Types, a unique instance space is to be provided for each separate Client-Type. There is no sharing of instance data across the Client-Types implemented by a PEP, even if the classes being instantiated are of the same type and share the same instance identifier.

The PIB can be described as a conceptual tree namespace where the branches of the tree represent structures of data or Provisioning Classes (PRCs), while the leaves represent various instantiations of Provisioning Instances (PRIs). There may be multiple data instances (PRIs) for any given data structure (PRC). For example, if one wanted to install multiple access control filters, the PRC might represent a generic access control filter type and each PRI might represent an individual access control filter to be applied.

3.6.2.3 Cops-pr defined protocol

Cops-pr is the protocol that we will use for conveying the configuration information from the PDP to the PEP, so if its needed further information about the protocol, [30] can be reviewed.

3.7 EXAMPLE OF A FRAMEWORK

The network of our example is the network of a small company (Figure 3.5), with the following topology:

- LAN address range: X.Y.0.0/16
- Subnets X.Y.I.0/24 (public), X.Y.2.0/24 (administrators), X.Y.3.0/24 (employees)
- A central router A that routes the LAN and Internet traffic, and serves as the Internet gateway.

Suppose that the following high-level abstract access rules have been set:

#1. Internal LAN traffic is always allowed
#2. The administrator can always access the Internet, whenever and from wherever he/she is logged in.
#3. During overall congestion, traffic between the employee domain and the Internet is denied.
#4. Internet can be accessed only during working hours (Monday to Friday, 9:00-] 7:00)
(Rule #1 has the highest priority, rule #4 the lowest)

Also, suppose that the term "overall congestion" is evaluated according to whether router A is congested, i.e., based on the load of its interfaces.

Suppose that the (PEP of the) routers of the network support a PIB with a single PRC, PRIs of this PIB describe source/destination criteria that allow access to IP traffic within the network. Each PRI in this PIB is a stand-alone policy of the form:

If ((Source matches Srcaddr/Srcmask) and (Destination matches Destaddr/Destmask)) then allow.

Traffic that matches at least one PRI in the PIB is allowed. Traffic that does not match any criteria (policies in the PIB) is, by default, denied.
Suppose now that the following events take place:

08:59: No administrator logged on
09:00: start of working day
11:00: congestion detected
11:05: no congestion
15:08: congestion detected
15:11: administrator logs on at X. Y.3.7
15:20: no congestion
17:00: end of working day
17:15: administrator logs out

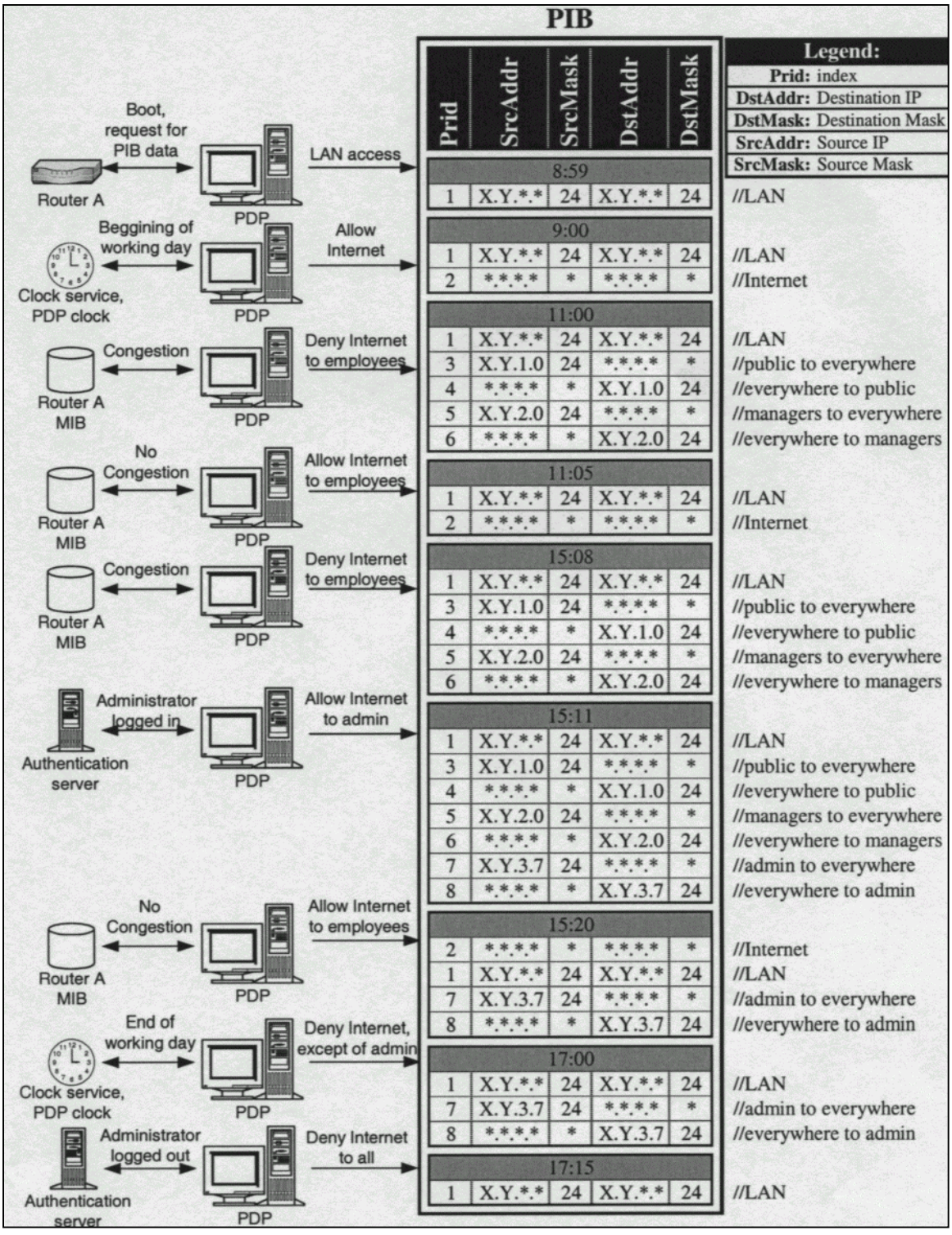

Figure 2: COPS-PR Example

Figure 2 demonstrates snapshots of the PIB of Router A during the day:

- When the router boots, the PDP sends a policy that allows all LAN traffic (PRUD #1), which implements policy #1,
- When the PDP detects the beginning of the working day (09:00), policy #4 becomes applicable, and a PRI that allows traffic to/from the Internet is added into the PIB (PRI #1 is now redundant; the PDP may decide to keep it or not; however this does not affect significantly our analysis).
- When congestion is detected (11:00), the PDP attempts to install policy #3. This policy is in conflict with the already installed policy #4; however policy #3 has higher priority, and hence the employee subnet is banned from Internet traffic.
- After a while (11:05), the network is no longer congested, and the PIB is restored to its previous state.
- When the network becomes congested again (15:08), the PIB has to be updated once more, as before.
- When the administrator logs on at the guest subnet, however (15:11), traffic to/from the Internet to his/her IP is allowed. Note that policy #2 is in conflict with policy #3, which bans traffic to the employee subnet, however the former wins since it has a higher priority.
- When the network becomes decongested (15:20), policy #3 is uninstalled, and policy #4 is installed again. At the end of the working day (17:00), policy #4 is also uninstalled, and finally, when the administrator logs out, policy #2 is uninstalled as well, denying all Internet access

4 METAPOLICIES

This chapter refers to the new idea [38] proposed to enhance the possibilities of policy based networking, and also to reduce the shortcomings that are evidenced in the traditional design of the framework.

4.1 DEFICIENCIES OF THE TRADITIONAL APPROACH

In the study cases presented in the typical study of the policy based networking, the COPS-Pr protocol is used as a method to communicate the configuration information from the server to the client. Also this protocol is used to update the required information in the PIB of the PEP, whenever the PDP decides it is necessary.

This way the high level policies are installed in each PIB´s device, with the current network state. When various events take place, the state changes and the PIB is modified. Also, when the same event occurs not always means that the same actions are taken. The occurrence of the same events does not even imply that the PDP will send exaclty the same commands to the PEP. However, there is a certain correlation between the network events and the PIB contents, which this model fails to take into consideration.

There are some special cases when the traditional approach can be improved and this are the cases that we are going to describe next. In some cases PDP has to send the same (or similar) commands, when the same event occurs. In the previous example, for instance, while the network alternates between the states "congested" and "not congested", the PDP needs to install and remove the PRIs that deny Internet access to the employee domain. In a more complex example, a big set of PRIs might need to be updated. The PEP needs to be directed about how to treat an event, even if the same event with the same consequences has occurred in the past. Hence, with these ´dumb´ devices those need to be instructed and directly directed for each change of the Network State, consuming some times many resources of the PDP for generating (the same) configuration updates, and consuming more bandwidth that necessary.

This shortcoming of COPS-PR has a great impact on its efficiency and performance, so it is a good way to try to solve this kind of problem.

Another limitation of the traditional design is located in the rigid design of the PIBs, because are too predefined structures and the high level policies cannot directly map into them, so the PDPs need to convert the high level policies into policies that can be represented in the PIB. All policies that do not precisely map to a supported policy type need to be processed at the PDP level. In the previous example, the policy “During overall congestion, traffic between the employee domain and the Internet is denied “ cannot fit in he PIB and has to be processed by the PDP. The latter, depending on the overall network state, produces the PRIs that are in conformance with the initial policy, for the given congestion status. Then the PEP implements the policies that theses PRIs describe. In this case, the high level policy has to be processed partially by the Ppd. and partially by the PEP. Obviously, the involvement of the PDP in cases like this is usually neither efficient nor desired.

In this example the PDP should ask to the PIB of the PDP for a possible congestion, and then send the appropriate policies back to the router’s PIB. It is clear that this policy could be completely processed at the PEP level, since congestion could be evaluated locally by the PEP. Similarly, for the policy “Internet can be accessed only during working hours”, the PDP is necessary in order to determine the condition “working hours”, since this condition cannot be stored in the PIB of the router. However, supposing there is a clock service that broadcasts the date and time over the network, this policy could also be evaluated entirely at the PEP level.

The rigidity of the PIBs also does not allow any other kind of policies to be evaluated by the PEP apart from these supported by the PIB, making in this way the presence of the PDP necessary, even in cases where this could be avoided. This is a significant drawback, since it makes the model very vulnerable to PDP errors or malfunctions and to network error situations, such as network congestion or network failures.

4.2 SOLUTIONS

The shortcomings of the COPS-PR protocol leading the work through possible feasible solutions that solved those issues. The first remarkable question is that the general conception of the framework seems to entrust much of the intelligence and responsibility into the PDP, so if this fails all the system will be affected.

Also the question of the repeatability of the installing and removing some PRIs when the same events occurs is a point to solve. And the rigidity of the PIB is in some way an impediment to achieve much more performance.

In general, the PEP depends much more of the necessary on the PDP, even in cases when this is not absolutely necessary.

The solution is to extend the functionality of the PIB, so the PEP could be more independent and take more decisions with the available information. The PDP would download applicable policies and directions on how to react to certain events. This would push much more functionality on the PEP, so the PDP would have fewer responsibilities and the fail of the system would be much less probable.

The role of the PDP would be downgraded mainly to communicating such events to the PEP, rather than modifying the configuration data. Also, the PEP could be programmed to monitor some of these events by itself and initiate the appropriate actions.

Assuming this conception, the PDP is able to control the PEP mainly by communicating events, rather than policies. Also, the PEP is able to take certain policing decisions by itself. In this way, intelligence is pushed toward the PEP.

More functionality is pushed towards the PEP, and in order to achieve this we use meta-policies, a new concept that is defined and discussed in the next point.

4.3 THE CONCEPT OF THE META-POLICIES

With the implementation of this new concept we will solve the shortcomings defined in the previous point. For the explanation of the concept, we will try to illustrate with an example.

In the example of the previous chapter there was the policy:

"During overall congestion, traffic between the employee domain and the Internet is denied".

Suppose that this is the only policy of a small network, consisting of two routers, A and B, where router A is the central router of the network, and B a router of a sub-domain. Also, suppose that these routers have a small filtering PIB like the one examined before, and that the condition "overall congestion" is indicated through some MIB variables of router A.

Whenever congestion is detected, the PDP sends to the PEPs of the routers some configuration data that install some PRIs and update their behavior. Since we have only one policy for this network, each router receives the same commands each time that congestion is detected. Let us call these data *DataA* and *DataB*. These PRIs are uninstalled when congestion ends.

Suppose now that the PDP could send to the two routers the following commands, which we shall call ***meta-policies:***

Router A:	**Router B:**
• *If (Congestion) then (DataA}*	• *If (Congestion) then {DataB}*

Finally, suppose that the PDP somehow directs the PEP of router A on how to evaluate the parameter "Congestion" from the appropriate variables of its MIB and informs the PEP of router B that the value of "Congestion" will be sent to it, each time that it changes. In this case, we can observe the following:

• The PDP only needs to send the meta-policies once. Then the PEPs have all the necessary information to react according to current Network State, as long as they are informed about it somehow.

• Router A can evaluate the two meta-policies locally and independently of the PDP. This means that the PDP does not need to process the original policy for router A any more. Also, the PEP will operate according to the administrative goals even in cases of high congestion (that would delay the PDP from querying the MIB of router A and update its PIB), or even while the PDP is down or unreachable.

• Router B still needs to be guided by the PDP. However, the PDP does not need to send policy commands in the form of configuration data (DataB) anymore; it must send only the value of the variable "Congestion". In this way, the PDP load is decreased,

less bandwidth is consumed, and the PDP Decision message is less likely to get lost or corrupted (since it is significantly smaller).

Although the case of a network with more than a single policy complicates the situation, based on the previous discussion, we can observe that in general, each high-level policy requires some specific PRIs to exist (or not exist) in the PIB of each device, depending on the network state. Each network event makes applicable some policies that were not applicable before and vice-versa. This means that we can associate combinations of events with PRIs that need to exist in the PIB.

Meta-policies attempt to take advantage of exactly this observation. They associate combinations of network events with PRIs that need to be installed. The event combination comprises the ***condition*** of the meta-policy; the modifications of the PIB that these events trigger are its *actions*. Meta-policies are generated by the PDP and they are sent to the PEP. The PEP processes these meta-policies and updates its PIB. The decision that the PEP takes is the same that the PDP would take, for the same network events. Of course, in order to do so, the PEP must be aware, somehow, of all the relevant network events. The PDP could be used for this purpose and inform the PEPs about network events that need a global (or at least a relatively "large") network view to be evaluated. In this case, the PEP still depends on the PDP, but less network and PDP resources are consumed. However, the PEP can be informed of network events from other sources, as well: For instance, the PEP may use the MIB of the device where it resides to evaluate local events. A network service or server (like a clock or a notification service) can also be used. Even more, mobile agents can be used to collect and provide notification of such events. The latter implies some degree of programmability and openness at the architecture of the PEP; however, such features are becoming available more and more in modem devices.

An important issue that needs to be addressed is conflicts. Valid meta-policies may be conflicting under certain circumstances. Besides, meta-policies may conflict with PRIs directly installed into the PIB by the PDP. As in COPS-PR, the PDP must resolve these conflicts before sending any commands to the PEP. Conflicts between meta-policies can also be resolved at the PEP level, as long as these policies are associated with priorities, provided by the PDP.

Finally, note that the mapping between meta-policies and high-level policies is not necessarily one to one. Some high-level policies may not be applicable for a device; some may be combined into a single meta-policy; and some others may need to be split into more that one. Besides, the PDP may still decide not to produce a meta-policy for a high-level policy, and implement it by directly installing and uninstalling PRIs into the PIB.

4.4 FORMAL DEFINITION

We define a meta-policy as a rule of the form:

if (condition) then {actions}

Where *"condition"* is a logical expression, e.g., "(C>60%) and (D==true)", and *"actions"* is a set of PIB commands that install PRIs into the PIB.

Since the actions encode a specific policy, this rule is a rule on **how** policies are enforced, on the contrary at **what** policies are enforced. This is what we call this new concept as ***meta-policies.***

Each meta-policy is generated by a PDP for a specific PEP, depending of the capabilities and limitations of the latter, so the actual information that is sent to the device is meaningful for this one only.

The way to deal with the meta-policies is to analyze them in each moment to see if the *condition* is satisfied, and if this is true to enforce the policies defined in the *actions;* installing them in the PIB of the device.

For this reason the conditions are logical expressions that are evaluated by the same PEP, and the actions just denote the PRIs that will be installed in the device when the primmer are satisfied.

Other question is that two meta-policies may conflict under certain situations. For such meta-policies, the PDP provides a relative priority between them. So if two meta-policies that are evaluated true by its conditions, two of them will be enforced except for the reason that they are in conflict among each other. If this is the case only the one with higher priority will be actually enforced in the PIB of the device.

Whether the condition and the actions may be *parametric.* This is of course clear in the case of the condition, since a not parametric one would be evaluated always the same, and the complete condition would be not necessary at all. In the case of the actions, they can contain parameters to contain variable values that we could need to install in the PIB.

The parameters are used in meta-policy conditions in order to determine when a meta-policy must be activated. Moreover, they are used by meta-policy actions in order to dynamically bind the network state within policies. For example, one meta-policy applicable to the example before could be:

CONDITION	ACTION
If (AdminLogged) then	Install (7, AdminIP, 24, *.*.*.*, 24) Install (8, *.*.*.*., 24, AdminIP, 24)

In this meta-policy we see that the condition is simply the parameter *AdminLogged.* In the PEP context, when this parameter becomes true, the device will be responsible to install the actions that are defined in the *action* field. This specific actions give acced to the network to the *administrator*.

When the *AdminLogged* parameter is false again (because the PDP sends its correct value when the administrator disconnects from the network), the PEP will remove the installed PRIs from the PIB.

The parameter must be of different classes, depending of the evaluation method to get its value. The PDP must inform in some way to the PEP the evaluation method, that could be obtaining its value from the MIB of the PEP, or getting its value from the local field that the PDP has filled in.

However the evaluation methods depend on the final capabilities of the device.

4.5 METAPIB EXAMPLE

Based in the example that we studied before, we will examine how will be affected by the addition of meta-policies.

The first policy is:

Internal LAN traffic is always allowed

And always must be enforced, so directly the PDP enforces it installing the PRI #1 into the PIB when the router boots.

The next meta-policies are:

If (worktime) then	Install (2,*.*.*.*, 24, *.*.*.*, 24)
If ((if1Util>80%) or (if2Util>80%) (if3Util>80%))	Install (3,X.Y.1.0, 24, *.*.*.*, 24) Install (4, *.*.*.*, 24, X.Y.1.0, 24) Install (5,X.Y.2.0, 24, *.*.*.*, 24) Install (6, *.*.*.*, 24, X.Y.2.0, 24)
If (AdminLogged)	Install (7,AdminIP, 24, *.*.*.*, 24) Install (8, *.*.*.*, 24, AdminIP, 24)

Also the PDP must inform to the PEP that the two first meta-policies are conflicting and the second one has higher priority. For this the PDP will send the correspondent objects that declare this statement.

The used parameters must also be reported to the PEP with its related evaluation methods. The parameters values for *worktime, AdminLogged* and *AdminIP* are sent by the PDP, and the parameters *if1Util*, *if2Util* and *if3Util* are evaluated through the appropriate MIB variables that denote the usage of the router interfaces.

The PEP will monitor the parameters, and when their value change, it reevaluated the affected conditions.

With this approach we accomplish the objective of pushing more responsibilities and intelligence towards the PEP. The PDP will download the appropriate meta-policies to the device, and afterwards it will control the device behaviour sending network events, instead of complete policies to install.

In addition many of this network events can be monitored by the PEP itself, without consuming any bandwidth at all, and accomplishing to be more independent. Moreover, parameter like *worktime* sent by the PDP could be evaluated directly by the PEP through a network clock service if it had the way to get it.

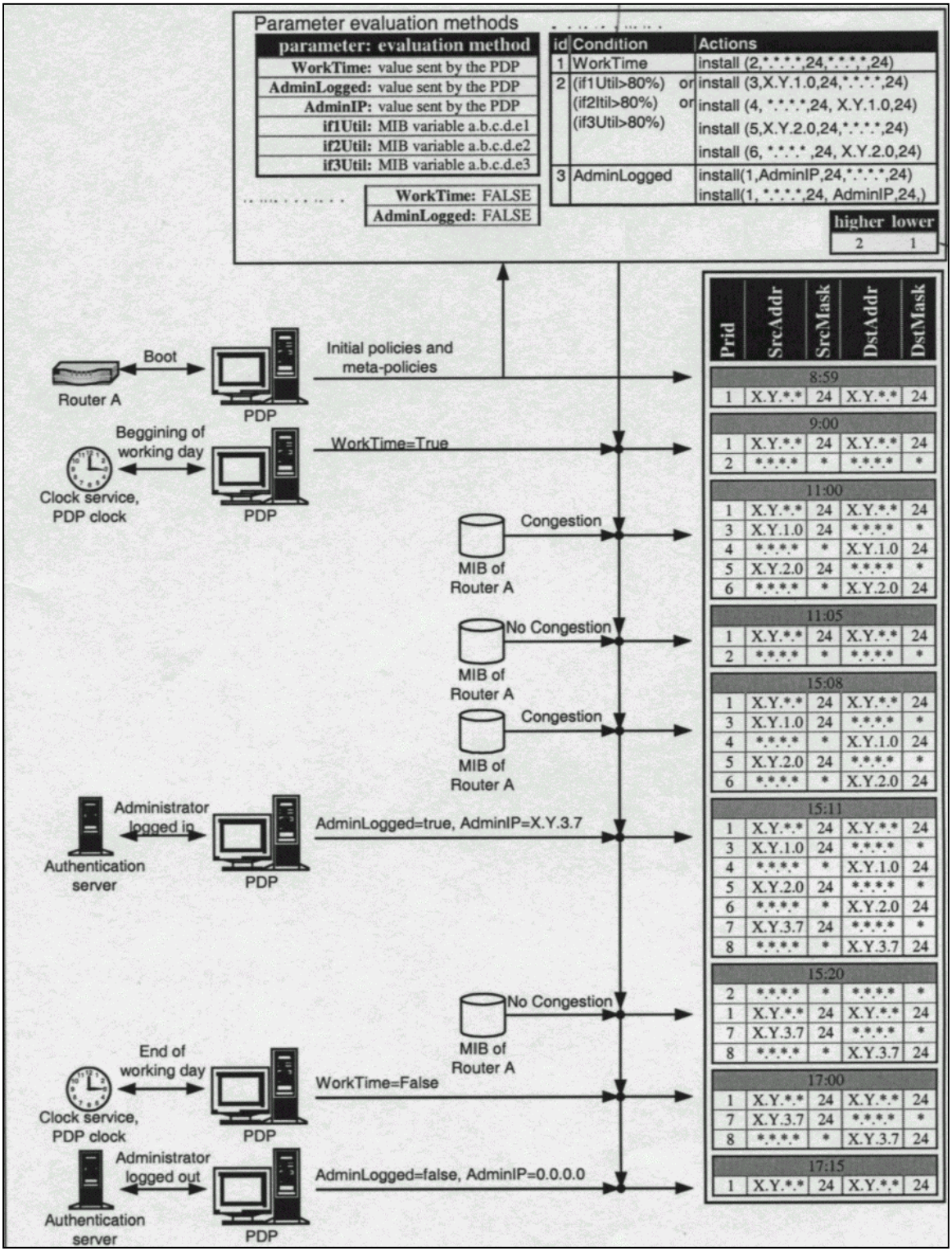

Figure 3: Example with MetaPolicies

5 ANALYSIS AND DESIGN OF THE FRAMEWORK

The design of the complete framework includes the design of the two basic entities that will make possible the implementation of a policy based network. Although the PDP could be a very complicated entity, we only need to design and implement the basic part to communicate and control the PEP.

The latter will be more complicated in our design, since it has to implement all the issues that we have previously described, including the meta-policy part

5.1 PDP

The design of this device is pretty easy for our purposes:

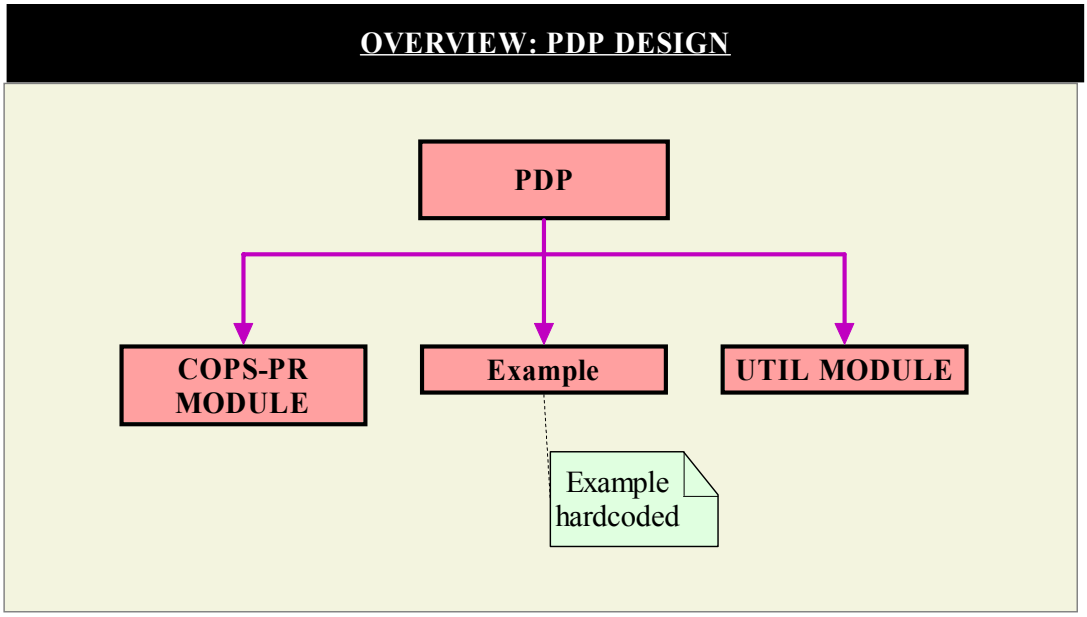

Figure 4: PDP Overview

We need one dedicated module to handle the communication with the COPS-PR protocol, and to convey all the messages to send to the PEP module.

Another part of the device will be the util methods to make easier the transformation of the high level policies to the cops-pr objects that are really sent over the network. This will be done more or less automatically in the future, but now we have also another part of the PDP that is responsible of hardcode the exact example that will be sent.

5.2 PEP

The PEP device is the final enforcement point where the policies take effect, so its role is very important in the framework, and the design of it must be carefully studied to take in consideration all the necessary issues of the environment.

The first characteristic that we must to take into consideration is the fact of the communication protocol. This device must be able to deal with the COPS-PR protocol, so an implementation of this protocol will be necessary to convey the policy (and meta-policy) information between the PDP and the PEP.

Also the purely policy information is handled in a traditional way. For this purpose the PEP must main maintain a PIB where install the policies that are directly sent by the PDP.

The new introduction of the meta-policy concept needs to deal with another class of objects that must handled separately by the PEP.

The overall design of the PEP is introduced in the next figure:

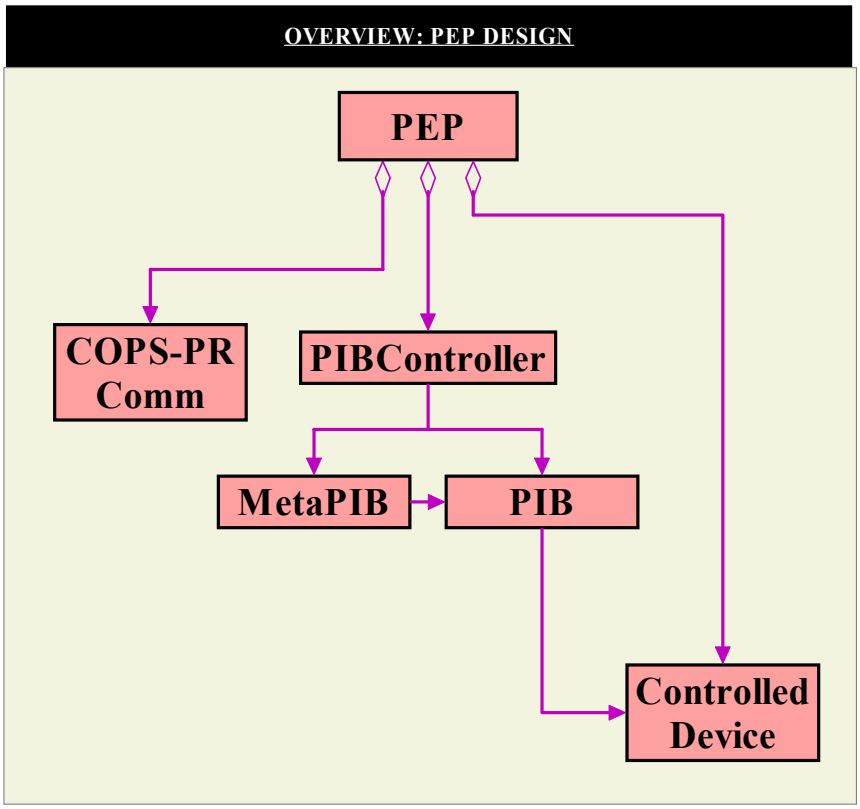

Figure 5: PEP Overview

In the next points we will discuss the requirements of the different components of the PEP and their design.

5.2.1 Cops-Pr Comm Module

This module is in charge of all the communications with the PDP so it will implement all the functionality to deal with the protocol and to convey the messages.

5.2.2 Pib

This is the module where the enforced policies are maintained, either because they have been directly sent by the PDP or because they are enforced from meta-policies whose conditions are true in a determined moment.

The concrete design and implementation of the PIB will be dependent of the designed and implemented PRCs and PRIs, and will vary depending of the necessities of each concrete PEP and final objective of the PIB. Hence, depending if we want to have a PIB to control one area (of the FCAPS framework) we will have a different PIB design.

5.2.3 Meta-Pib

This is the most complicated part of the modeled device as it has lots of requirements. This point analyses the requirements, justifies our choice to use another PIB to implement the additional functionality and discusses its design issues.

5.2.3.1 Requirements

The explained meta-policies are the main concept that restrict and guide the design of the Meta-Pib.

Each condition of a meta-policy is a Boolean expression, comprised of a number of simpler conditions. Ultimately, all conditions are decomposed into primitive logical expressions, such as arithmetic comparisons (InterfaceUtilization < 80), Boolean expressions (AdministratorLogged == true), network expressions (IpAddress matches X.Y.Z.W), etc

The actions install PRIs into the PIB. Each action identifies a single target PRI and the value that must be installed into it.

Both conditions and actions may be parametric; hence a way to communicate, store and process parameters is also necessary. Each parameter has a type, which denotes what kind of information it stores (integer, IP address, octet string, etc). Also, each parameter has a way to be evaluated. Several evaluation methods may exist. We distinguish two basic evaluation methods: First, a parameter can get its value from the MIB or PIB of the device. Second, the value can be sent by the PDP initially, and then be updated (by the PDP) whenever it changes. However, other evaluation methods may also exist, depending on the capabilities of the device. For instance, an active/programmable device may download and execute code that will evaluate this parameter. Although it is practically impossible to support any possible evaluation method, it is desirable that the

basic methods that we define can be extended with other methods (standard or vendor-specific).

5.2.3.2 Analysis

By choosing to define a PIB and use COPS-PR, all communication and storing is addressed by the protocol itself: When the PEP connects to the PDP, it reports its meta-policing capabilities and limitations. According to these capabilities and limitations, the PDP downloads all the appropriate meta-policies. These meta-policies are stored in the PIB and remain there, until they are updated by the PDP.

The meta-policy data requirements are described next:

5.2.3.2.1 Meta-Policies

Meta-policies consist of a condition and a set of actions. Since valid meta-policies may conflict under certain circumstances, the PDP must be able to declare potentially conflicting meta-policies and denote priorities between them. Also, the status of the meta-policies (whether they are active, whether they suppress a meta-policy with lower priority or whether they are suppressed) may need to be reported to the PDP.

5.2.3.2.2 Conditions

Each meta-policy must contain exactly one condition. As mentioned before, the condition is decomposed into one or more primitive expressions that need to be evaluated. Each of these primitives must contain at least one parameter (otherwise, a simpler condition without them exists, since that primitive expression always evaluates either true or false). We distinguish two categories of primitives: Boolean and generic expressions.

Boolean expressions are a subset of the generic expressions, but due to their simplicity and commonality, they are treated separately. Such primitives are evaluated according to the value of a Boolean parameter. For instance, in the expression *((X>Y) && ('.Congestion) && (WorkTime)), Congestion* and *WorkTime* are such primitives.

Generic expressions contain all the other logical expressions that cannot be decomposed into simpler Boolean primitives. Examples of such primitives are *"IP matches X.Y.Z.W" or "8:00am < time < 5:00pm).* Each PEP can only support specific types of such expressions (e.g., arithmetic), which are reported along with the other PEP capabilities to the PDP. The PDP can only send to the PEP expressions that are supported by the latter.

An important issue is that such expressions need to be standardized in order to be transmitted and stored in the PIB, However, different types of expressions require different operators (e.g., arithmetic expressions need operators like "+","-",">", while network; conditions need operators such as "matches" and "subnet"). Besides, the set of types of such expressions is infinite, since any kind of expressions may be valid; the

expression "colorl darker that color2" is a valid expression (although probably totally useless for network management). The point is that all types of possible expressions, cannot be predicted in advance, but they need to be standardized. Of course, we could choose to standardize only a few types of expressions that are most commonly used, but this would restrict the applicability of our work.

The solution given to this problem was to define an open, generic mechanism to handle such expressions. The details of this generic mechanism can be defined per expression type (arithmetic, IP expressions, etc). We have already defined common expression types, but these types can easily be extended to include other ones, as well.

More specifically, all expressions are encoded using XML. XML uses tags that give semantics to the data of the XML document. However, the semantics of these tags are defined in separate documents, called Document Type Definitions (DTDs). These DTDs specify the details of the generic mechanism, per expression type. Each PEP reports to the PDP the DTDs that it supports, through an identifier, which uniquely identifies these DTDs (which is the URL where these are published; this is the standard method adapted by the XML standard). By reporting an XML DTD, the PEP declares that it can interpret any XML document (that encodes an expression) written according to this DTD. For example, if a DTD defines tags for numerical operations (+,-,*, /,div) and comparisons (>,=,>=,<,<=,=) then the PEP should be able to understand any arithmetic expression that uses these operators. The PDP, according to the expression that it wants to encode, chooses the most appropriate DTD, encodes the condition and transmits it.
By using XML DTDs we manage to:

- Standardize the exchange of general expressions
- Accomplish a uniform way of storing them into the PIB
- Leave the PIB open to any type of expressions
- Allow each PEP to implement only the functionality that it needs, or that is appropriate, according to its resources.

Note that the PDP is always able to find a way to send an expression, even if this is not optimal: Even if the appropriate DTD is not supported, the expression may be transformed to a supported one. In the worst case, the entire expression is represented as a Boolean parameter, and the PDP sends the value for this parameter

5.2.3.2.3 Actions

Each meta-policy can contain 1 or more actions that represent the policies to enforce when the associated condition is satisfied. It is acceptable that no action is specified, but normally this is not very common. Each action is a binding of a PRID pointing to a single PRI, and the value that will be installed in that. We also know that since *actions* can contain parameters, this can be dynamically evaluated (just before being installed).

5.2.3.2.4 Parameters

Parameters are present into conditions and actions of the meta-policies. Each parameter has an evaluation method communicated someway by the PDP. The mandatory evaluation methods are evaluated through the MIB or the PIB of the device, or through the PDP. Specific Vendors could specify another evaluation methods.

Parameters that are evaluated through the MIB need to specify the polling frequency to update the value

5.2.3.2.5 Pib approach

The proposed enhancements require meta-policing information to be exchanged between the PDP and the PEP, and be stored and processed by the latter. Hence, a crucial question that must be tackled in the early design phase is what protocols and data structures will be used. We decided to use COPS-PR to communicate such data and define a PIB to store them at the PEP (as opposed to defining another protocol and/or storage structure, or extending the existing ones). This decision was based on a number of reasons:

• Meta-policies need to be sent to the PEP in a provisioning style, and COPS-PR is a protocol defined for policy provisioning.

• Our work is in line with the work conducted in IETF. No new protocols need to be developed, and the proposed PIB can easily be adapted by the Internet community (researchers and vendors). Even legacy devices can support the proposed PIB (e.g., with software updates).

• By using a PIB to store meta-policies, meta-policing data are treated as any PIB data. Consequently, meta-policies on meta-policies could also be defined

• Finally, by using COPS-PR and PIBs, the design and the implementation is simplified: the definition of a PIB is much simpler that defining a new protocol. Meta-policy exchange and storage is already handled by the protocol and does not need to be addressed by us. The implementation is based on existing, tested tools. The reuse of knowledge and code makes the design, implementation and testing safer and easier, and minimizes the chance for errors.

In general, although the choice of using a PIB and COPS-PR introduces some further requirements, it does not prevent or hinder us from meeting any of our goals.

Our decision to define a PIB and use COPS-PR to implement our proposal implies that the SPPI specification must be used to define the PIB. SPPI [37] demands all data to be placed in tabular format (each table is a PRC, and the rows of the tables the PRIs). SPPI also demands strong typing of the attributes of the PRIs. However, the SPPI is very flexible in defining new types; this feature is exploited in order to overcome the previous restriction.

5.2.3.3 Design

In this section we will define the Meta-Policy PIB classes. The full review of this part can be found in [38].

5.2.3.3.1 Meta-Pib

According to the previously presented analysis, we define the classes (tables) or PRC that comprise the Meta-Pib. These classes are defined using the SPPI.

The PRCs are grouped into five groups:

- The **Capabilities group** contains the provisioning classes (PRCs) that store the capabilities and limitations of the PEP. The PRIs of these classes are reported to the PEP into the REQ message when the PEP connects. This groups contains the XML DTDs that the PEP supports, for encoding espressions. Each row of the xmlDTDTable consists of an identifier and the DTD URL.

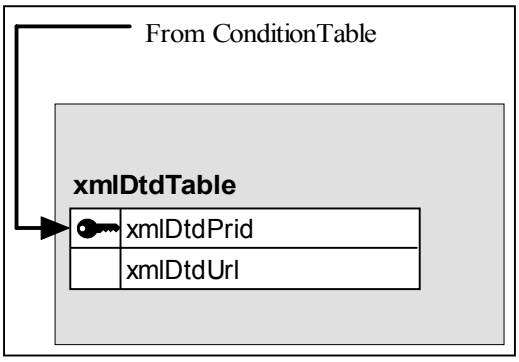

Figure 6: Capabilities Group

- The **Parameter group** contains the classes: *parameter, mibPibParameter,* and *pdpParameter*.
  The *parameter* class is the base class for the representation of parameters. Each PRI contains of an identifier (PRID), and a name of the parameter. Also each PRI must be associated with a PRI of a class that EXTENDS this one.
  MibPibParameter represents a parameter whose value is extracted from the MIB or the PIB of the devices. The class contains the PRID of the PRI that the parameter extracts its value from. Since the MIB and the PIB have different name spaces, the identifier in meaningful by itself alone. This class contains also a value that represents the frequency of polling to obtain its value
  The *PdpParameter* contains the last value that the PDP has sent so it contains the actualized value.

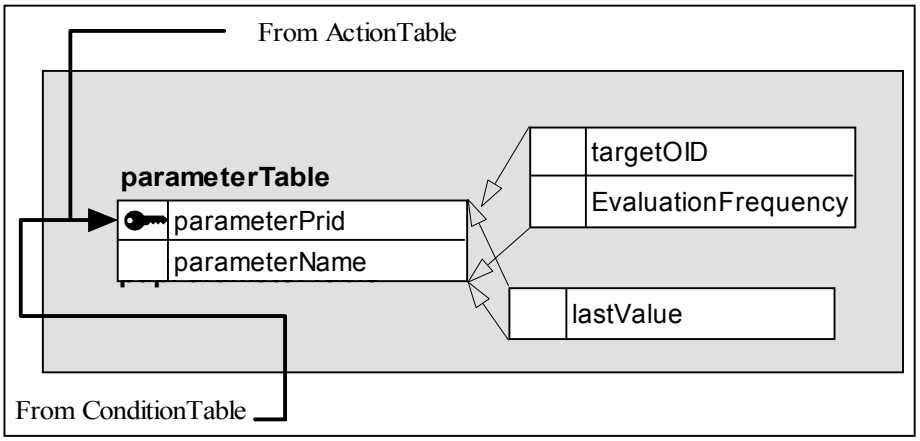

Figure 7: Parameter Group

- The base **Meta-Policy group** contains three classes direcly related with the meta-policy handling: the *metaPolicy,* the *metaPolicyStatus* and the metaPolicyPriority classes.
  The *metaPolicy* class is the base PRC of the meta-policies. Each instance of this class represents one meta-policy, and comprises an identifier (PRID), the name of the meta-policy, a reference to a condition and an action tag that references a group of actions.
  The *metaPolicyStatus* is a PRC that *AUGMENTS* the base class. This means that one instance of this class is directly related with the class that it augments, and only exists in the context of this one. Each instance (PRI) of this PRC reports the status (whether it is active or not) of the installed meta-policy, and the reason of this status (if it is being suppressed by another active meta-policy with higher priority, etc. These instances can also serve as parameter to other meta-policies, so as to construct conditions that are based on wheter installed meta-policies are active, inactive or suppressed.
  The metaPolicyPriority class is the PRC that represents the priorities between the metapolicies, if any. Each entry of the table contains two fields that refer to metapolicies, in a way that the first entry represents a meta-policy that has more priority that the second entry.

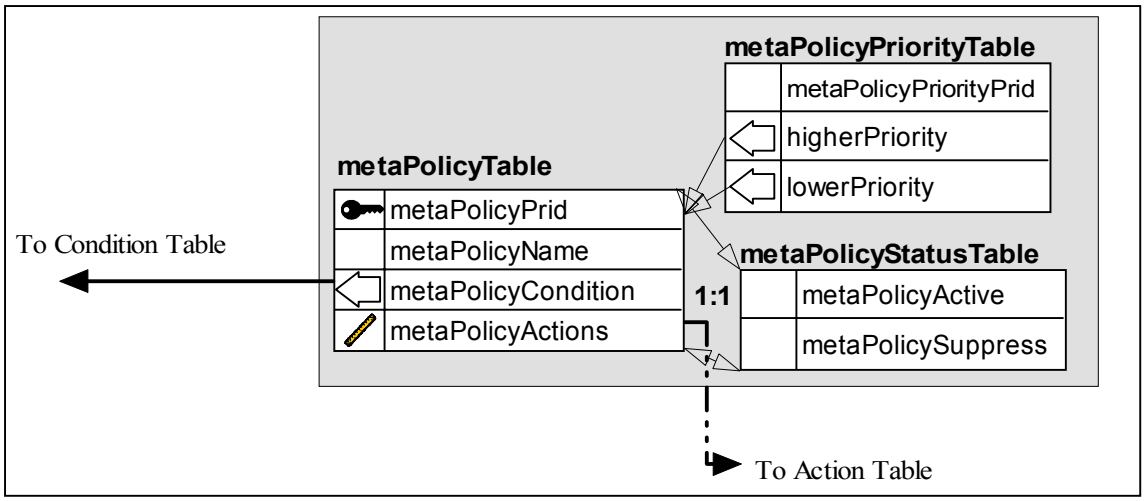

Figure 8: MetaPolicy Group

- The **Condition group** contains the classes related with the condition of a meta-policy. This group is formed by a parent class *condition* that represents the base condition and comprises an identifier (PRI), and a flag that indicates that the result of the evaluation of the condition must be negated. Nevertheless, this base class must always be associated with PRIs of another class that EXTENDS this one. So this class will never directly have instances, but its descendants.
  The classes that EXTENDS this one are *complexCondition,booleanCondition* and *generalCondition.*
  The first one, *complexCondition*, is used when decomposing a bigger condition breaking it into smaller components. This object permits construct binary trees of conditions, so it contains a *leftTerm* and a *rightTerm* that references other conditions that constitute the left and right branches. Also has an *operator* that defines a logical operation between these two terms that both represent logical conditions.
  The *booleanCondition* contains a reference to a parameter, with must be of type Boolean. The value of the condition will be evaluated according to the value of the parameter
  Finally, the *generalCondition* class is used to represent conditions that cannot fit in the ones described above. This class contains a XML formatted string that represents the condition. The formatting will be according a DTD, so this class contains a reference to a xmlDTD class, in addition to the condition string itself.

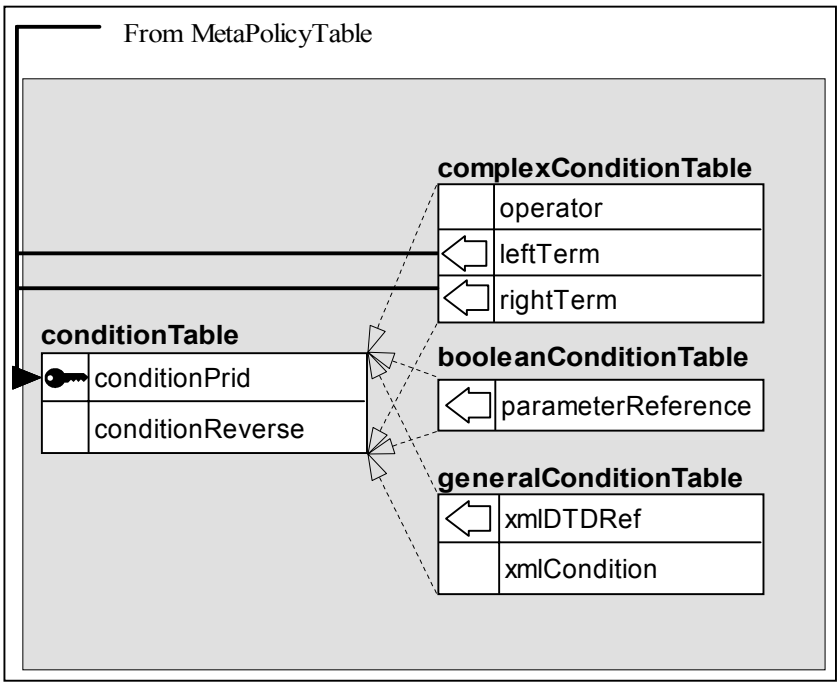

Figure 9 : Condition Group

- The **Actions group** consists of three classes: *action, actionValue* and the *actionParametricValue* classes.
  The base class, *action*, contains the identifier of the instance (PRID) and a tag that groups actions that will be installed together. Also contains the PRID of the concrete PRI that will be the container of the installed policy. Of course the PRID will refer to an entry of the PIB.
  This class has no instances by itself, but its descendants that EXTEND this one, have.
  The descendants provide the value that will be installed in the PRI, and depends on the class that the instance is.
  The *actionValue* class contains directly the value that will be installed in the PIR.
  The *actionParametricValue* contains a reference to a parameter that will be dynamically evaluated to obtain the value to install.

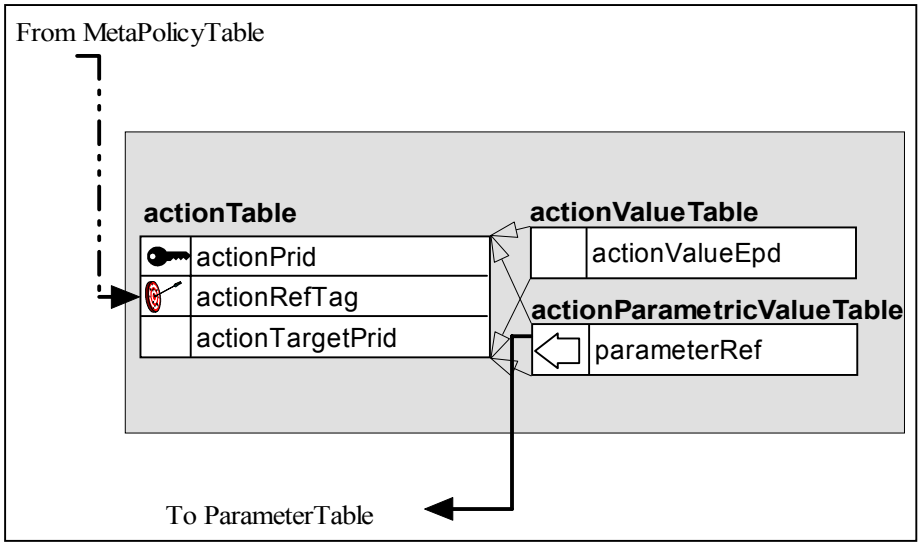

Figure 10: Action Group

5.2.4 Controlled Device

This is the interface with the controlled device whose behavior will be controlled by the policies enforced in the PIB in each moment. As we know these policies can be directly installed or through meta-policies. This is a very simple interface with the controlled router, switcher, or general network device. The code to control the device will be specific of each vendor hardware and will be inserted into the interface that we define.

This is the complete design of the MetaPib:

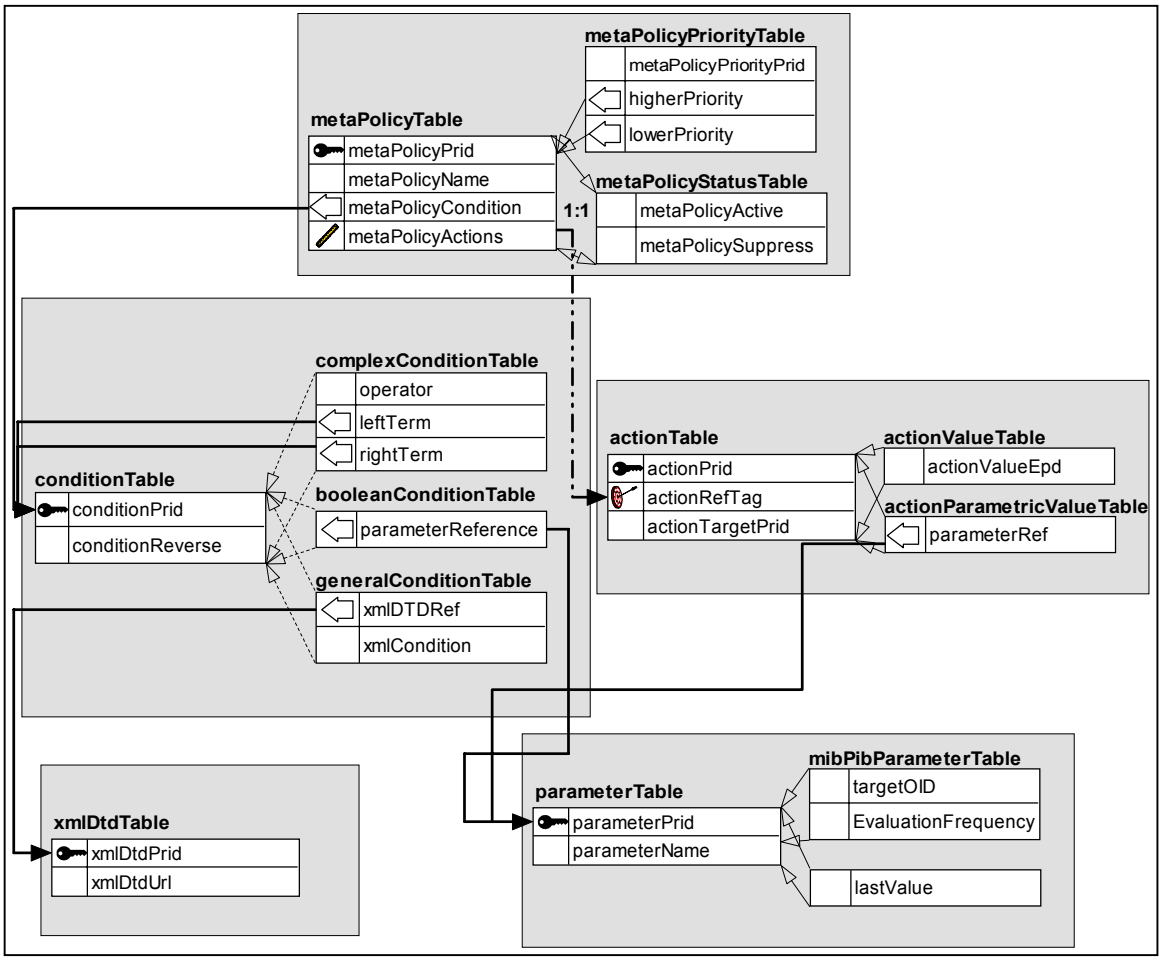

Figure 11: Complete MetaPib

6 IMPLEMENTATION

In this chapter we will introduce all the details of the implementation, including some classes schemas and also the specification of the main methods for the comprehension of the framewotk

6.1 PDP

In this point we will present the description of the implementation details of the PDP device.

6.1.1 PDP Working mode

The function of the PDP is, as we know, coordinate the right functioning of the policy-based network, sending the configuration data to the PEP and updating this configuration data when necessary.

The implementation of the PDP is centralized in the class ***Server***. This class has references to the objects of the classes that it needs to perform the communication with the PEP.

For example it will have a ***ServerSocket*** for the incoming requests. Also it has a reference to a ***COPSmessageReceiver*** object in charge of receiving and decoding the incoming COPS messages, and a ***CopsMessage*** object that will hold the last message received (the one that the ***COPSMessageReceiver*** has decoded).

The PDP can be very complicated because much intelligence can be conducted by this entity. Nevertheless a complete PDP implementation is out of this scope, so only the necessary functionality for testing the framework and, in consequence the METAPIB, has been implemented. The existing implementation, consider issues like COPS and COPS-PR communication making possible the communication with the PEP. The concrete sent commands must be hardcode, so this is what we should do for any example that we want to test. Functionality to convert high level policies to the specific commands was out of the first approach for its inherent complexity, but can be studied to be added in the future.

The Server class contains the hardcoded rules that permit to test the proposed example. It also contains a reference to another object that contains another hardcoded

commands to test another example, but the code can be modified to reference all the examples that we need.

6.1.2 PDP architecture

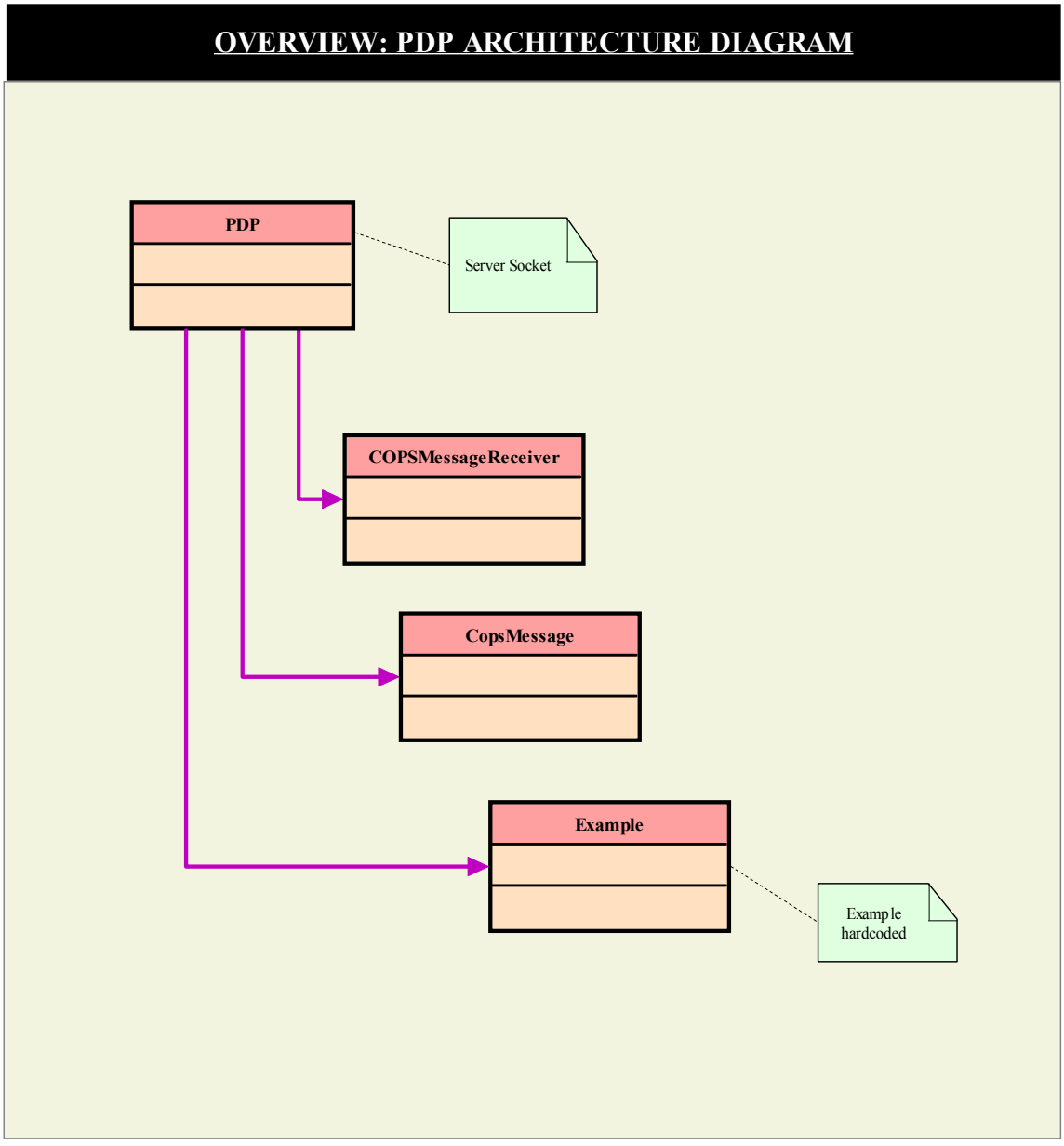

Figure 12 : PDP Concrete Architecture

6.1.3 PDP Classes

Class Server
thesis.pdp.Server

Description

This is the main class of the PDP server. It contains all the code necessary to serve the PEP clients, and also contains some hardcode rules in order to test the environment.

Fields

Type	*Name*	*Description*
ServerSocket	serverSocket	this is the server socket that serves the incoming requests from the PEP
DataOutputStream	out	stream for sending *outgoing* messages
DataInputStream	in	strean for obtaining *incoming* messages
CopsMessageReceiver	rcv	to receive and decode incoming messages
CopsMessage	msg	to hold the last decoded COPS message
BufferedReader	stdin	only for convenient purposes, this buffer is for entry by keyboard from the PDP Console.

Methods

main

Invocation	public static void **main**(String[] args)
Description	This is the main function when the PDP is executed. It basically opens a ServerSocket and waits connections. When it receives an incoming connection from a PEP, it will send the configuration information that is coded, and will wait for the messages of confirmation that the PEP will issue.
Parameters	args - arguments to the function for a parametric execution of the server.
Returns	none
Throws	IOException - where there is an error in the sockets or in the

other methods

Other methods included in the implementation correspond to the hardcoded example.

Also the example class *Example2* contains a reference to an object that is defined in the copspr package and that helps to *hardcode* the objects that will be sent.

6.2 PEP

In the next point we will introduce the details of the implementation of the PEP and the description of the methods of each implied class.

6.2.1 PEP working mode

As we know for the design of policy context, the PEP is basically in charge of opening a communication session with the PDP, and waiting the configuration and update messages from the server to work correctly. It will maintain a PIB when it stores the policies that are enforced, and will handle the incoming and outgoing packets in consequence.

Also, with the new concepts introduced in the METAPOLICY framework, the PEP will have more intelligence to solve situations when the PDP cannot pilot constantly the behavior of the PEP. For this reason the PEP will maintain also the meta-policy information base METAPIB, where it stores the metapolicies and all the other objects of the framework.

In the implementation, the most important class of the PEP is the class named **PEP.class**. This class is responsible for coordinating all the other related classes, and is the departure point for the understanding of the PEP behavior.

Every PEP contains a reference to a **COPSComm** class, in charge of the basic communication over the network with the COPS protocol.

Basically the **COPSComm** class contains a socket (actually contains a reference to a socket object that implements the functionality of the socket) that serves as entry point of the incoming requests and communications. Also this class contains all the code regarding the COPS protocol and serves the other classes with the COPS data.

The PEP class also contains references to each client that it is able to deal with. In fact it contains a reference to a **ClientHandler** class for each client.

Thus it handles the corresponding references to the types of the former, in form of a references vector.

Each **ClientHandler** is able to handle different client-specific issues. In our case, we only need that it be able to deal with COPS-PR objects, so it will contain a reference to an object of the class **COPSPRClient**.

The **COPSPRClient** class contains the code related with the COPS-PR protocol, and therefore is able to handle all the objects, and messages, of that protocol. It will serve the modules that depend of him with the correct data for the right working of those modules. In this point, the PEP is able to extract the content of the messages that will be installed or removed in the framework.

Likewise, the **COPSPRClient** has a reference to an object of the **PIBController** class, which it will call with the data that it unpacks from the received messages. This class is in charge of discerning the address space of the object to install or update, so it will call the right method. If the object to install/remove is in the address space of the PIB, it will call the matching method of the **PIB** class.
If it is in the address space of the METAPIB, it will do the same with the **METAPIB** class.

These two classes are defined to deal specifically with the restrictions of the incoming objects, because although the PIB could not have any consistency restrictions, the METAPIB objects have a lot. For example a **MetaPolicy** object has to refer a valid installed **Condition** object, so if this does not exist, it will issue an error. All this consistency and integrity checking is performed in the context of the **METAPIB** class.

The **METAPIB** class is the most important and complex class of the entire framework and plays an important role in maintaining the framework consistency, and in the enforcing of the correct metapolicies as well. Although the concrete description of all the methods that perform the functionality of the MetaPib will be explained later, we will present an overview of the PEP architecture, and some of the important related classes.

6.2.2 PEP architecture

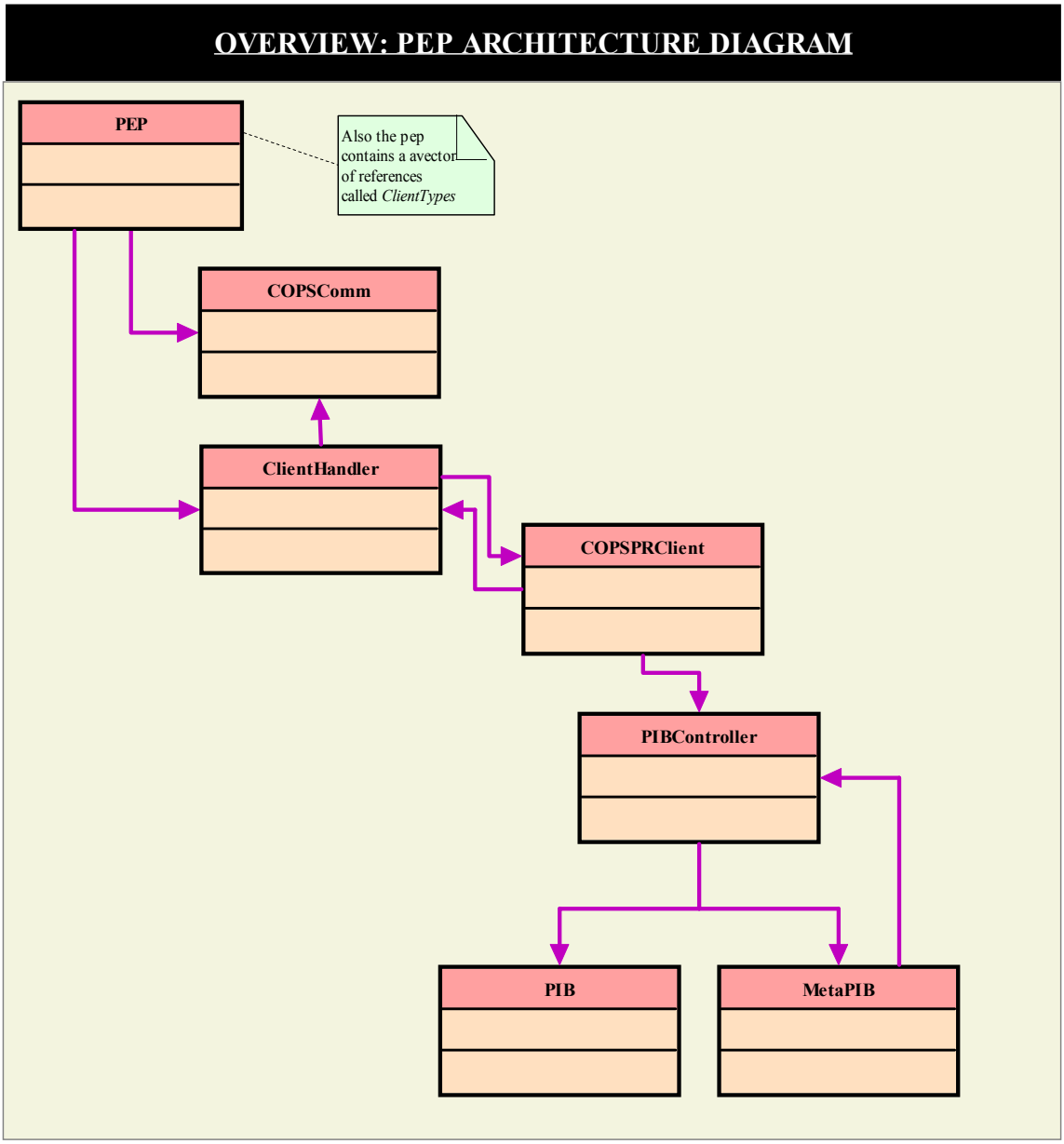

Figure 13: PEP Concrete Architecture

6.2.3 PEP Classes

PEP PACKAGE

Class PEP
thesis.pep.PEP

Description

This class is the main one of the PEP and is responsible of the correct working mode of the PEP. It also contains references to another classes to handle with all messages and data. The working mode of the PRP has been previously discussed.

Fields

Type	*Name*	*Description*
Private COPSComm	comm	Link to the COPSComm object that serves as communication point.
Private final int	NUMBER_OF_CLIENTS	Number of clients that the PEP is capable to del with
Private final String	PDPAddr	Internet Address of the PDP
Private final int	PDPPort	Port of the PDP
Private short[]	ClientTypes	Vector of NUMBER_OF_CLIENTS components, that contains the types of the corresponding ClientHandler
Private ClientHandler[]	ClientHandler	Vector of NUMBER_OF_CLIENTS + 1 components that containts the references to the correspondingClientHanles
Protected String	PEPName	Name of the PEP that will be used to identify this PEP in the communication with the PDP

Methods

(Constructor)

Invocation	**PEP()**
Description	Constructor. Initialices the fields of the object
Parameters	none
Returns	none
Throws	none

run

Invocation	public void **run** (String PepName)
Description	begins the PEP: 1) Connects to the PDP 2) Initizalices the ClientHandler and ClientTypes structures depending of the PEP/PDP capabilities 3) waits for PDP messages
Parameters	PepName - name of the PEP
Returns	none
Throws	none

OPN

Invocation	public void **OPN** (short ClientHandler)
Description	Sends a open (OPN) message thru the COPSComm
Parameters	ClientHandler - number of the Client that sends the OPN message
Returns	none
Throws	none

Class COPSComm

thesis.pep.COPSComm

Description

This class maintains the code to communicate with a PDP. It is the lowest layer of the protocol.

Fields

Type	*Name*	*Description*
private PEP	pep	Reference to the linked PEP
private Socket	sock	Socket
private DataOutputStream	out	Stream for outgoing data
private DataInputStream	in	Stream for ingoing data

Methods

(Constructor)

Invocation	public **COPSComm**(String HOST, int PORT, PEP pep)
Description	This constructor creates a new Socket with the related parameters to communicate with the PDP and obtains the DataOutputStream (out) and DataInputStream (in)
Parameters	HOST - Internet Address of the Host (PDP) PORT - Port of the Host PEP - Linked PEP
Returns	none
Throws	COPSException - if there is an error in the COPS protocol

getNextMessage

Invocation	public CopsMessage **getNextMessage**()
Description	creates a new CopsMessageReceiver with the **input** data
Parameters	none
Returns	the CopsMessage that has received
Throws	COPSException - if there is an error in the COPS protocol

OPN

Invocation	public void **OPN**(short ClientType)
Description	creates and sends a new open (OPN) message
Parameters	ClientType - type of the client that it is willing to open
Returns	none
Throws	COPSException - if there is an error in the COPS protocol

CC

Invocation	public void **CC**(short ClientType)
Description	creates and sends a new Close (CC) message
Parameters	ClientType - type of the client that it is willing to close
Returns	none
Throws	COPSException - if there is an error in the COPS protocol

REQ

Invocation	public void **REQ**(short ClientType, int ClientHandle, short rtype, short mtype, int size, Vector S, Vector N)
Description	creates and sends a new Request (REQ) message
Parameters	ClientType - type of the client that it is sending the request CientHandle - identifier of the Client (ClientHandle) rtype - Rtype of the Context (refer to Context Object) mtype - Mtype of the Context (refer to Context Object) size - size of the vectors of dataInputStream S - Ctype vector (ctype of each ClientSI object) N - ClientSI vector
Returns	none
Throws	COPSException - if there is an error in the COPS protocol

RPT

Invocation	public void **RPT**(boolean solicited, short ClientType, int ClientHandle, short rtype, int size, Vector CTypeV, Vector ClientSIV)
Description	creates and sends a new Report(RPT) message
Parameters	solicited - indicates if the report has been solitited by the PDP ClientType - type of the client that it is sending the request CientHandle - identifier of the Client (ClientHandle) rtype - Rtype of the Context (refer to Context Object) size - size of the vectors of dataInputStream CTypeV - Ctype vector (ctype of each ClientSI object) ClientSIV - ClientSI vector
Returns	none
Throws	COPSException - if there is an error in the COPS protocol

finallize

Invocation	void finallice()
Description	closes the data streams, and the socket
Parameters	none
Returns	none
Throws	IOException - there has been an error closing the data streams and the socket

Class ClientHandler
thesis.pep.ClientHandler

Description

This class represents a determined kind of client and it is responsible for handling correctly the messages that it receives, corresponding with the semantics of that client.

The clientHandle implemented corresponds with a COPS-PR client, since it is necessary for our goals. Nevertheless this class could implement other kind of client, only respecting the interface with the rest of the objects.

Fields

Type	*Name*	*Description*
private COPSComm	comm	Link to the associated class COPSComm, necessary to reply to some messages
int	handle	handle of the client (unique in a PEP)
short	clientType	type of the client
COPSPRClient	Client	Reference to the associated client. In this case, and how commented before, it is a COPSPRClient (field clientType: COPSPRClient)
boolean	expectingDec	flag to know if it is expecting a decission from the PDP

Methods

(Constructor)

Invocation	public ClientHandler(int handle, COPSComm comm)
Description	This constructor initialices and fills all the necessary flag fields
Parameters	handle - handle of the client comm - reference to the associated COPSComm object
Returns	none
Throws	none

DEC

Invocation	public void **DEC**(CopsMessage msg)
Description	this method is in charge of handle an incoming Decision (DEC) message. It extracts all the idecision objects and calls the same method for the associated Client.
Parameters	msg - the incoming decision message
Returns	none
Throws	COPSPRException - error processing the message

CAT

Invocation	public boolean **CAT**(short ClientType)
Description	this method is in charge of handling the Client Accept message. It calls the 'start' method of the associated client, that also will send a REQ msg to the PEP
Parameters	ClientType - type of the accepted client
Returns	true - if the client starts OK false - it there is any error
Throws	none

CC

Invocation	public boolean **CC**(short ClientType)
Description	this method is in charge of handling the Client Close message. It calls the method 'accept' setting false the accepted flag of the associated client.
Parameters	ClientType - type of the client that is being closed
Returns	true - if the client closes OK false - it there is any error
Throws	none

REQ

Invocation	public void **REQ**(short rtype, short mtype, int size, Vector S, Vector N)
Description	this method is in charge of send a Request (REQ) message to the PDP. It calls the even method of the linked COPSComm object, and it is used at the beginning of the boot phase.
Parameters	rtype - Rtype of the Context (refer to Context Object) mtype - Mtype of the Context (refer to Context Object) size - size of the vectors of dataInputStream CTypeV - Ctype vector (ctype of each ClientSI object) ClientSIV - ClientSI vector
Returns	none
Throws	none

RPT

Invocation	public void **RPT**(boolean solicited, short reportType, int size, Vector S, Vector N)
Description	this method is in charge of send a Report(RPT) message to the PDP. It calls the even method of the linked COPSComm object.
Parameters	solicited - if this report was solicited by the PDP size - size of the vectors of dataInputStream CTypeV - Ctype vector (ctype of each ClientSI object) ClientSIV - ClientSI vector
Returns	none
Throws	none

checkClientType

Invocation	public void **checkClientType**(short clientType)
Description	this method checks wheter the clientType as parameter is the same of the actual client , or not.
Parameters	ClientType - type of the client that is being checked
Returns	none
Throws	COPSException - if the client type is not the same of the implemented client.

COPSPR PACKAGE

Class COPSPRClient

thesis.copsprclient.COPSPRClient,

Description

This class implement the code related of a client of the PEP. In this case is a client for policy provisioning with COPS-PR protocol

Fields

Type	*Name*	*Description*
private static short	ClientType	Type of the client (COPSPRClient = 1)
private static final boolean	EXTND	Currently not used
private int	Handle	handle of the client (unique identifier in a PEP)
private PIBController	pibc	Associated Pib Controller
boolean	accepted	Flag to indicate wheter the client is already accepted or not (At the beginning equal to False)
private ClientHandler	Controller	Link to the Client Handler of this client

Methods

(Constructor)

Invocation	public **COPSPRClient**(int handle, ClientHandler c)
Description	this constructor fills an initialices all the class fields.
Parameters	handle - handle of the client c - reference to the associated ClientHandler object
Returns	none
Throws	none

accepted

Invocation	public void **accepted**(boolean acc)
Description	method called when the client is accepted by the PDP (with acc == True) of then the client is closed by the PDP (acc == False)
Parameters	acc - tells if it has been accepted or not
Returns	none
Throws	none

start

Invocation	public void **start**()
Description	method called when the client is accepted by the PDP (receives the CAT message). 1) starts the associated It calls the method with the same name of the associated PIBController and afterwards, 2) Sends the corresponfing REQ message to the PDP, thru the associated Controller. Data regarding configuration, is filled in (1) and sended in (2)
Parameters	none
Returns	none
Throws	none

DEC

Invocation	public void DEC(short rtype, short mtype, short code, byte[] Data)
Description	this method is in charge of handle a single decision object
Parameters	rtype - Rtype of the Context object that was in the associated decision message mtype - Mtype of the Context object that was in the associated decision message code - Install / Uninstall decision Data - bytes of data of the rest of the message
Returns	none
Throws	COPSPRException - error processing the object

COPSPRCLIENT PACKAGE

Class PIBController

thesis.copsprclient.PIBController

Description

This class is responsible of controlling the actions taken in the data structures that maintain the policies (and meta-policies) in the PEP. Actually it deals with COPSPR Objects and depending on the kind of object it decides its destination in the following way:

If it is a PIB object it sends the object to the PIB module. If it is a METAPIB object, it will send it to the METAPIB module.

The PiBController class is a descendant of the Thread class, because it is responsible of constantly check if there are changes in the parameters (for example the MibPibParameters) that change the value of any condition. So in that case it will have to enforce the policies into the PIB.

Fields

Type	*Name*	*Description*
PIB	pib	PIB module (object) that deals with the pib objects
METAPIB	metapib	METAPIB module (object) that deals with metapib objects.

Methods

(Constructor)

Invocation	public **PIBController**()
Description	This constructor initialices and fills all the necessary fields. Also starts a new thread in charge of the checking of the MibPibParameters.
Parameters	none
Returns	none
Throws	none

run

Invocation	public void **run**()
Description	this method overrides the run() method of the Thread (parent) class. It constanly checks the MibPibParameters.
Parameters	none
Returns	none
Throws	none

getPibType

Invocation	private int **getPibType**(String pridString)
Description	private method to decide the module (at present PIB or METAPIB) that will correspong to the prid sent as parameter
Parameters	pridString - prid (as string)
Returns	integer identifier of the type of the prid.
Throws	none

install

Invocation	public void **install**(CopsPRObjSet objs)
Description	install the set of COPSPR objects in the pertinent place (in the PIB or the METAPIB)
Parameters	objs - set of objects to install
Returns	none
Throws	COPSPRException - if there is an error in the messages construcion, or if there is an error installing that object (for example for integrity constraints)

remove

Invocation	public void **remove**(CopsPRObjSet objs)
Description	removes the set of COPSPR objects (of the PIB or the METAPIB)
Parameters	objs - set of objects to remove
Returns	none
Throws	COPSPRException - if there is an error in the messages construcion, or if there is an error removing that object (for example for integrity constraints)

selectInstall

Invocation	private void **selectInstall**(OIDObj oid, EPDObj epd)
Description	called from 'install'; selects where to install the object (PIB or METAPIB)
Parameters	oid - oid of the object to install epd - epd of the object to install
Returns	none
Throws	COPSPRException - if there is an error in the messages construcion, or if there is an error installing that object (for example for integrity constraints)

selectRemove

Invocation	private void **selectRemove**(OIDObj oid, EPDObj epd)
Description	called from 'install': selects where to remove from the object (PIB or METAPIB) and calls the corresponding remove function of the PIB or the METAPIB.
Parameters	oid - oid of the object to install epd - epd of the object to install
Returns	none
Throws	COPSPRException - if there is an error in the messages construcion, or if there is an error removing that object (for example for integrity constraints)

start

Invocation	public int **start**(Vector Ctype, Vector ClientSI)
Description	this function is called when the PIBControlled is initialized, and is in charge of sending the initial information to the PDP regarding the status of the PEP, its capabilities, etc ...
Parameters	Ctype - vector of the type of objects sended ClientSI - vector of the specific client information
Returns	int - number of objects sended
Throws	COPSPRException - if there is an error in the messages construcion

enforcePolicy

Invocation	public void **enforcePolicy**(String oid, byte[] data)
Description	this function is called when it is necessary to enforce a policy
Parameters	oid - oid of the policy to enforce in the PIBControlled data - data bytes of the policy to being enforced
Returns	none
Throws	COPSPRException - if there is an error enforcing the policy in the PIB

unenforcePolicy

Invocation	public void **unenforcePolicy**(String oid)
Description	this function is called to unenforce a Policy from the PIB
Parameters	oid - oid of the policy to unenforce in the PIBControlled

Returns	none
Throws	COPSPRException - if there is an error enforcing the policy in the PIB

updatePolicy

Invocation	public void **updatePolicy**(String oid, byte[] data)
Description	this function is called to update a Policy in the PIB. In fact it calls the same function to install a new one, so that fuction will be the one in charge of check if it is installing a new one (no previous exists) or is updating.
Parameters	oid - oid of the policy to unenforce in the PIBControlled data - data bytes to update with
Returns	none
Throws	COPSPRException - if there is an error updating the policy in the PIB

METAPIB PACKAGE

The objects in this package represent all the Metapib related objects. They all are necessary to implement the added functionality to the PEP and as we will see, all of them are part of a class tree that inherits from the base class called MetaObject.

This package also contains the class METAPIB, that is referenced from the PIBController and is responsible for mantaining the consistency of the related PIB that contains the metapolicies (the MetaPib itself). It checks all the constraints when installing and removing new objects into the metapib, and is also in charge of looking which are the metapolicies that are evaluated true in each moment, and which are the policies that must be installed (or removed).

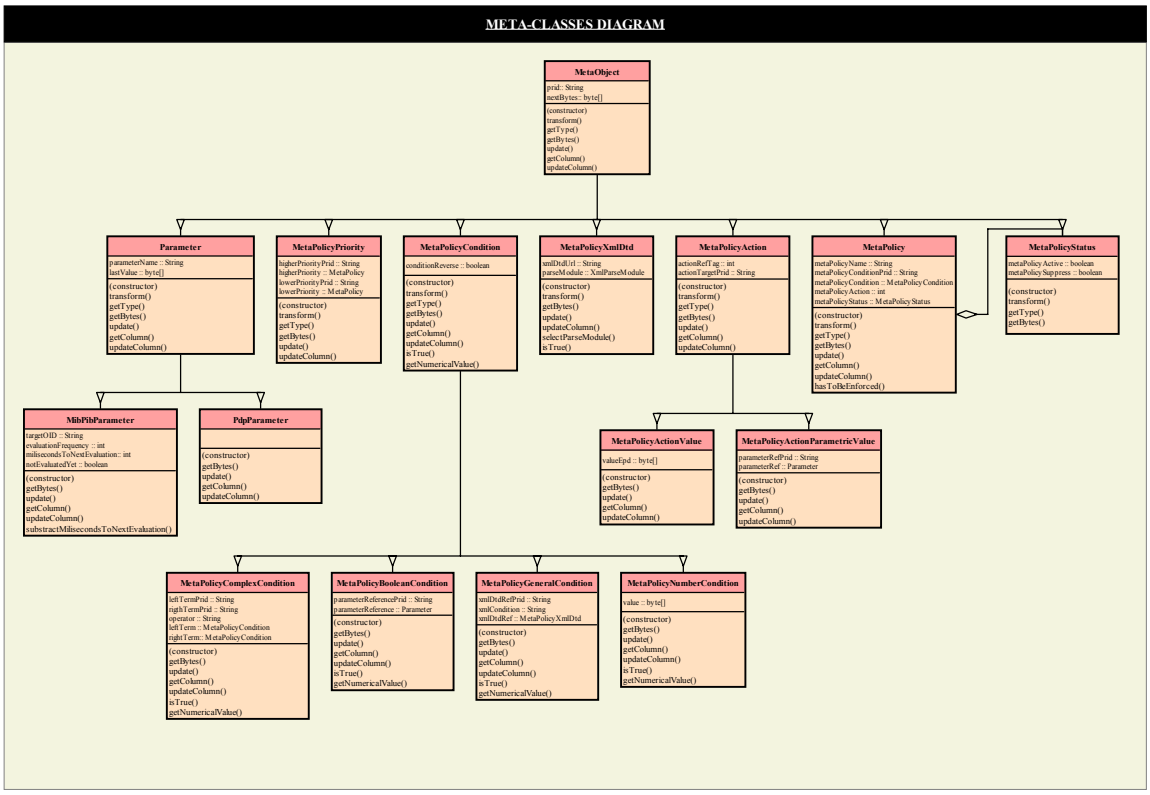

Figure 14 : MetaClasses

Class MetaObject

thesis.metapib.MetaObject

Description

This is the base class for all the other MetaObjects as Parameter, Condition, Action, MetaPolicy, etc. Contains the fields and methods that are common for all of them, and we will see that this approximation as a tree of classes is very well designed as

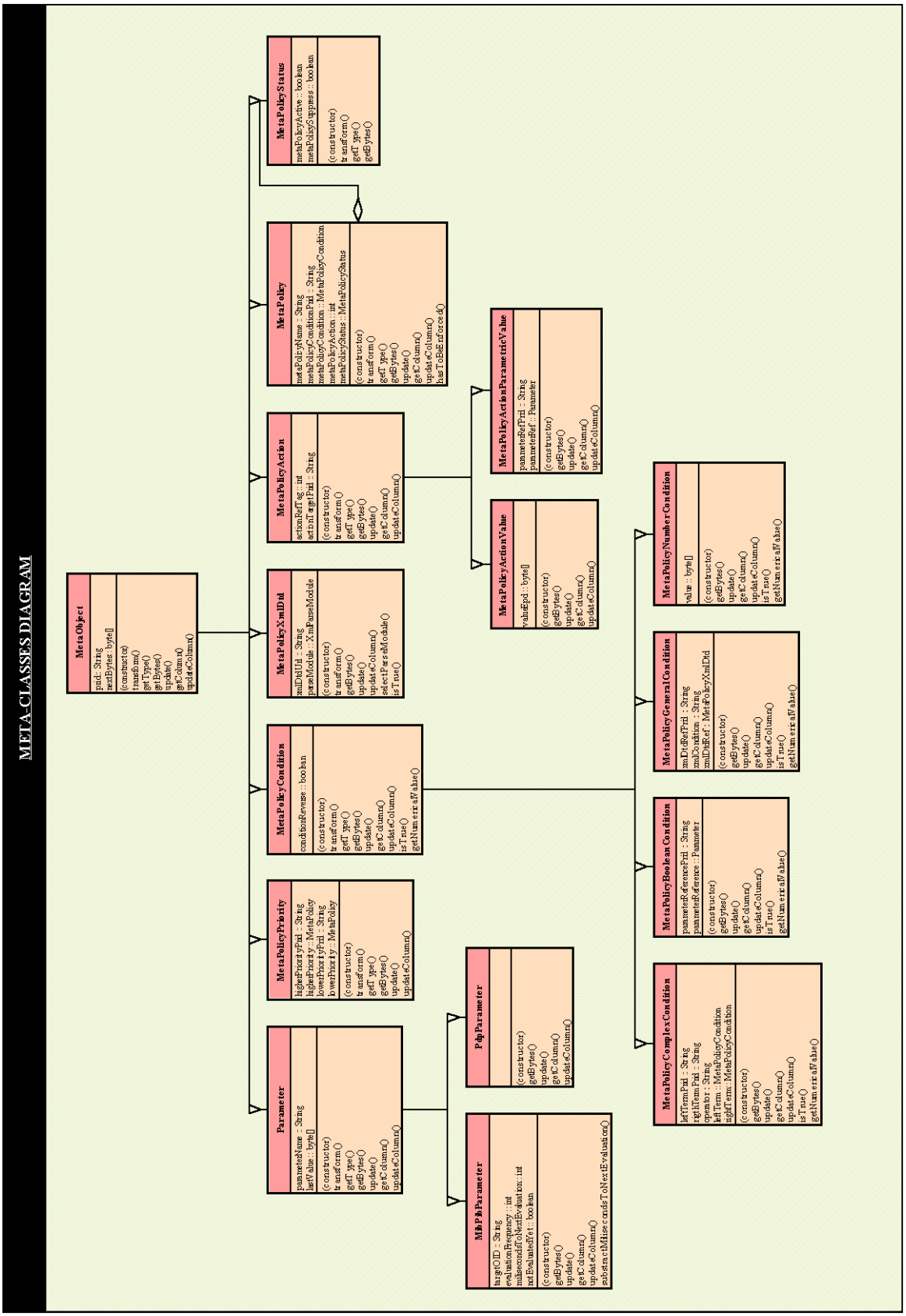

Figure 15 : MetaClasses (Extended)

the reusing of the code (of each class parent) improves wheter the performance regarding speed, and the code size as well.

Fields

Type	*Name*	*Description*
Protected String	prid	prid of the object (each object has a pri identifier inside a the tree of PRC)
Protected byte[]	nextBytes	is a field used when the object is created from a byte sequence (For example when received from the PEP in a stream) Represents the rest of the bytes that have not been analized and are part of the descendants of this class.

Methods

(Constructor)

Invocation	public **MetaObject**(String prid)
Description	This constructor creates a MetaObject initializing the prid to the one passed as parameter. the field *nextBytes* will be null.
Parameters	prid - the prid of the metaObject
Returns	none
Throws	none

(Constructor)

Invocation	public **MetaObject**(MetaObject object)
Description	This constructor creates a MetaObject from another MetaObject. It copies all the fields one by one, an will be used when creating a new MetaObject from one of its descendants.
Parameters	prid - the prid of the metaObject
Returns	none
Throws	none

(Constructor)

Invocation	public **MetaObject**(byte[] dataBytes)
Description	This constructor creates a MetaObject from a stream of bytes. It will obtain in order the prid, and will put subsequent bytes in the *nextBytes* field
Parameters	dataBytes - byte atream from where the object will be constructed

Returns	none
Throws	none

transform

Invocation	public **transform**()
Description	This method will transform this general MetaObject in a more correct, specific one, one of its descendants, depending of the prid that it has.
Parameters	none
Returns	MetaObject - the correct descendant of MetaObject that corresponds to the prid.
Throws	none

getType

Invocation	public static int **getType**(String pridString)
Description	returns the type that corresponds to that prid
Parameters	pridString - the prid to analize
Returns	int - identifier (as definded in the *Constant* class) that corresponds to the prid.
Throws	none

getBytes

Invocation	public byte[] getBytes()
Description	This method obtains the object as a byte stream, and it is specific of each descendant, so it will be overriden by each descendant. Basically it transforms in an ordered way each field, coding it into bytes with the BER protocol. It will be used when conveying the object from the PDP to the PEP.
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

update

Invocation	public void update(MetaObject newMetaObject)
Description	method to update a MetaObject with another one, basically copying all the fields. This method must be overriden for its descendants to accomplish the semantics.
Parameters	newMetaObject - the metaObject to update with
Returns	none
Throws	MetaPibException - when there is an error updating

getColumn

Invocation	public byte[] getColumn(int column)
Description	method to get a column of a specific MetaObject.
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column
Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void updateColumn(int column , byte[] newValue)
Description	method to get update a column of a specific MetaObject.
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

Class Parameter

thesis.metapib.Parameter

Description

This class extends the base class MetaObject and represents the general Parameter of the metapolicy framework. Basically it inherits the field and methods of its parent (we will see that overrides some methods to manaint the semantics)

This class will be a superclass of the PdpParameter and the MibPibParameter.

Fields

Type	***Name***	***Description***
protected String	parameterName	mantains a string with the parameter name with identifier proposals.
protected byte[]	lastValue	this is a field that contains the lastValue of the parameter, and it is in this class for optimization purposes (see subclasses MibPibParameter and PdpParameter. It will contain the last value sended by the PDP (in case of it is a PdpParameter) or the last value obtained by the PEP (MibPibParameter)

Methods

(Constructor)

Invocation	public **Parameter**(String parameterPrid_, String parameterName_)
Description	This constructor creates a MetaObject calling the super constructor with the prid sent as first parameter, and also filling the parameterName field of the class with the second parameter. the field *nextBytes* will be null.
Parameters	parameterprid - the prid of the parameter parameterName - name of the parameter
Returns	none
Throws	none

(Constructor)

Invocation	public **Parameter**(Parameter son)
Description	This constructor creates a Parameter from another Parameter. It first calls the analog method of the parent (to copy the fields related to the parent) and after copies all the fields of this class one by one (in this case parameterName and nextBytes). It will be used when creating a new Parameter from one of its descendants, useful in some case as we will see.
Parameters	son - the parameter to construct from
Returns	none
Throws	MetaPibException - when there is an error creating the parameter

(Constructor)

Invocation	public **Parameter**(MetaObject parent)
Description	This constructor creates a Parameter from an object of the superclass MetaObject. It first calls the analog method of the parent (to copy the fields related to the parent) and after extracts or creates the specific object fields from the bytes of the *nextBytes* field. It will be used when creating a new Parameter from an object of its parent class, when it is creating the right object from a stream of bytes.
Parameters	parent - MetaObject to create the Parameter from
Returns	none
Throws	MetaPibException - when there is an error creating the parameter or because the bytes not correspond directly to the established fields.

transform

Invocation	public **transform**()
Description	This method overrides the homonym method of its parent class and has the same semantic adapted to a Parameter: will transform this general Parameter in a more correct, specific one, one of its descendants, depending of the prid that it has.
Parameters	none
Returns	MetaObject - the correct descendant of MetaObject (Parameter in this case) that corresponds to the prid.
Throws	none

getType

Invocation	public static int **getType**(String pridString)
Description	Overrides the homonym parent method. returns the type (restricted to the *Parameter* types) that corresponds to that prid
Parameters	pridString - the prid to analize
Returns	int - identifier (as definded in the *Constant* class) that corresponds to the prid.
Throws	none

getBytes

Invocation	public byte[] getBytes()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the object (in this case only *parameterName,* not *lastValue* because it is only transmited for PdpParameters)
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

update

Invocation	public void update(MetaObject parObject)
Description	overrides the parent one. First, the method checks that the metaObjects sent as parameter is instance of a Parameter. After checks that the prid of the object to update, and the prid of the object to update from, are equal, to maintain the consistency. After the checking, it can copy the fields from the new object to this one.
Parameters	parObject - the metaObject (that must be instance of Parameter) to update with.
Returns	none
Throws	MetaPibException - when there is an error updating or when the parameter *parObject* is not from the correct class (Parameter)

getColumn

Invocation	public byte[] getColumn(int column)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be obtained in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to obtain the column).
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column
Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void updateColumn(int column , byte[] newValue)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be updated in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to update the column).
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

Class PdpParameter

thesis.metapib.PdpParameter

Description

This class extends the class Parameter and represents the general PdpParameter of the metapolicy framework. This class maintains the last Value sent by the PDP regarding the value of the parameter

Fields

This class has no specific fields in the implementation. The field *lastValue* has been 'uploaded' to the parent class for optimization purposes, but it will handle that field as if it was local.

Methods

(Constructor)

Invocation	public **PdpParameter**(String parameterPrid_, String parameterName_, byte[] lastValue)
Description	This constructor creates a PdpParameter calling the super constructor with the prid sent as first parameter, and parameterName as second one. Before that, it checks that the prid corresponds to a PdpParameter. After the creation of the parent, also fills the *lastValue* field of the class with the third parameter. the field *nextBytes* will be null.
Parameters	parameterprid - the prid of the parameter parameterName - name of the parameter lastValue - BER encoded last value of the parameter
Returns	none
Throws	MetaPibException - thrown when the prid that is being sent to create a Pdp Parameter does not corresponds in fact with a PdpParameter object

(Constructor)

Invocation	public **PdpParameter**(String parameterPrid_, String parameterName_, int lastValue)
Description	Wrapper to the base constructor, for convenience purposes, when the last value is a **integer.**
Parameters	parameterprid - the prid of the parameter parameterName - name of the parameter lastValue - BER/INTEGER encoded last value of the parameter
Returns	none
Throws	MetaPibException - thrown when the prid that is being sent to create a Pdp Parameter does not corresponds in fact with a PdpParameter object

(Constructor)

Invocation	public **PdpParameter**(String parameterPrid_, String parameterName_, String lastValue)
Description	Wrapper to the base constructor, for convenience purposes, when the last value is a **string.**
Parameters	parameterprid - the prid of the parameter parameterName - name of the parameter lastValue - BER/STRING encoded last value of the parameter
Returns	none
Throws	MetaPibException - thrown when the prid that is being sent to create a Pdp Parameter does not corresponds in fact with a PdpParameter object

(Constructor)

Invocation	public **PdpParameter**(String parameterPrid_, String parameterName_, InetAddress lastValue)
Description	Wrapper to the base constructor, for convenience purposes, when the last value is a **InetAddress.**
Parameters	parameterprid - the prid of the parameter parameterName - name of the parameter lastValue - BER/IPV4ADDRESS encoded last value of the parameter
Returns	none

Throws	MetaPibException - thrown when the prid that is being sent to create a Pdp Parameter does not corresponds in fact with a PdpParameter object

(Constructor)

Invocation	public **PdpParameter**(String parameterPrid_, String parameterName_, boolean lastValue)
Description	Wrapper to the base constructor, for convenience purposes, when the last value is a **boolean.**
Parameters	parameterprid - the prid of the parameter parameterName - name of the parameter lastValue - BER/BOOLEAN encoded last value of the parameter
Returns	none
Throws	MetaPibException - thrown when the prid that is being sent to create a Pdp Parameter does not corresponds in fact with a PdpParameter object

(Constructor)

Does not exist a constructor from a Parameter son because in fact there are no subclasses from this one. If there would be those subclasses, there would be necessary to implement that Constructor.

(Constructor)

Invocation	public **PdpParameter**(Parameter parent)
Description	This constructor creates a PdpParameter from an object of the superclass Parameter. It first calls the analog method of the parent (to copy the fields related to the parent) and after extracts or creates the specific object fields from the bytes of the *nextBytes* field. It will be used when creating a new Parameter from an object of its parent class, when it is creating the right object from a stream of bytes.
Parameters	parent - Parameter to create the PdpParameter from
Returns	none
Throws	MetaPibException - when there is an error creating the parameter or because the bytes not correspond directly to the established fields.

GetBytes

Invocation	public byte[] **getBytes**()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the object (in this case *lastValue*)
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

update

Invocation	public void **update**(MetaObject parObject)
Description	overrides the parent one. First, the method checks that the metaObject sent as parameter is instance of a PdpParameter. After checks that the prid of the object to update, and the prid of the object to update from, are equal, to maintain the consistency. After the checking, it can copy the fields from the new object to this one.
Parameters	parObject - the metaObject (that must be instance of PdpParameter) to update with.
Returns	none
Throws	MetaPibException - when there is an error updating or when the parameter *parObject* is not from the correct class (PdpParameter)

getColumn

Description	overrides the parent method but mantaining it: If the *column* is not possible to be obtained in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to obtain the column).
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column
Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void updateColumn(int column , byte[] newValue)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be updated in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to update the column).
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

Class MibPibParameter

thesis.metapib.MibPibParameter

Description

This class extends the class Parameter and represents the general MibPibParameter of the metapolicy framework. This class maintains the targetOID from where the parameter is evaluated and Evaluation Frequency that it is used for obtain that parameter value.

Fields

Type	*Name*	*Description*
protected String	targetOID	mantains the prid form where the parameter is evaluated
protected int	evaluationFrequency	evaluation frequency in miliseconds
protected int	milisecondsToNextEvaluation	field for optimization purposes that represents the milisecods left for the next evaluation.
protected boolean	notEvaluatedYet	flag to maintain if the MibPibParameter has been evaluated or not

Methods

(Constructor)

Invocation	public **MibPibParameter**(String parameterPrid_, String parameterName_, String targetOID_, int evaluationFrequency_)
Description	This constructor creates a MibPibParameter calling the super constructor with the prid sent as first parameter, and parameterName as second one. Before that, it checks that the prid corresponds to a PdpParameter. After the creation of the parent, also fills the *targetOID* field of the class with the third parameter, and *evaluationFrequency* with the fourth parameter. The field *nextBytes* will be null, and the *milisecondsToNextEvaluation* equal to the evaluationFrequency.
Parameters	parameterprid - the prid of the parameter parameterName - name of the parameter targetOID - OID prid to evaluate from evaluationFrequency - evaluation frequency in miliseconds.
Returns	none

Throws	MetaPibException - thrown when the prid that is being sent to create a Pdp Parameter does not corresponds in fact with a PdpParameter object

(Constructor)

Does not exist a constructor from a Parameter son because in fact there are no subclasses from this one. If there would be that subclasses, there would be necessary to implement that Constructor.

(Constructor)

Invocation	public **MibPibParameter**(Parameter parent)
Description	This constructor creates a MibPibParameter from an object of the superclass Parameter. It first calls the analog method of the parent (to copy the fields related to the parent) and after extracts or creates the specific object fields from the bytes of the *nextBytes* field. It will be used when creating a new MibPibParameter from an object of its parent class, when it is creating the right object from a stream of bytes.
Parameters	parent - Parameter to create the MibPibParameter from
Returns	none
Throws	MetaPibException - when there is an error creating the parameter or because the bytes not correspond directly to the established fields.

getBytes

Invocation	public byte[] getBytes()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the object (in this case *targetOID* and *evaluationFrequency*)
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

update

Invocation	public void update(MetaObject parObject)
Description	overrides the parent one. First, the method checks that the metaObject sent as parameter is instance of a MibPibParameter. After checks that the prid of the object to update, and the prid of the object to update from, are equal, to maintain the consistency. After the checking, it can copy the fields from the new object to this one.
Parameters	parObject - the metaObject (that must be instance of MibPibParameter) to update with.
Returns	none
Throws	MetaPibException - when there is an error updating or when the parameter *parObject* is not from the correct class (MibPidParameter)

getColumn

Invocation	public byte[] getColumn(int column)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be obtained in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to obtain the column).
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column
Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void updateColumn(int column , byte[] newValue)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be updated in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to update the column).
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

substractMilisecondsToNextEvaluation

Invocation	public void **substractMilisecondsToNextEvaluation** (int miliseconds2substract)
Description	This method subtracts the parameter value from the field *milisecondsToNextEvaluatio.* It is called from the evaluation method for MibPibParameters, to indicate the number of miliseconds awaited since the last time (that was evaluated any MibPibParameter - see method *checkMibPibParameter* in the METAPIB class for details)
Parameters	miliseconds2substract - number of miliseconds awaited
Returns	none
Throws	none

ClassMetaPolicyCondition

thesis.metapib.MetaPolicyCondition

Description

This class extends the base class MetaObject and represents the general Condition of the metapolicy framework. Basically it inherits the field and methods of its parent (we will see that overrides some methods to maintain the semantics)

This class will be superclass of the *MetaPolicyComplexCondition*, *MetaPolicyBooleanCondition*, *MetaPolicyGeneralCondition*, and a new class not defined in the metapib framework but created for optimization purposes, the *MetaPolicyNumberCondition*

Fields

Type	***Name***	***Description***
protected boolean	conditionReverse	flag to identity if the condition must be evaluated as a negative condition or not.

Methods

(Constructor)

Invocation	public **MetaPolicyCondition**(String parameterPrid_, boolean conditionReverse_)
Description	This constructor creates a Condition calling the super constructor with the prid sent as first parameter, and also filling the conditionReverse field of the class with the second parameter. the field *nextBytes* will be null.
Parameters	parameterprid - the prid of the parameter conditionReverse - flag to evaluate in a reverse way (if necessary)
Returns	none
Throws	none

(Constructor)

Invocation	public **MetaPolicyCondition**(MetaPolicyCondition son)
Description	This constructor creates a MetaPolicyCondition from another MetaPolicyCondition. It first calls the analog method of the parent (to copy the fields related to the parent) and after copies all the fields of this class one by one (in this case *conditionReverse* and *nextBytes*).

	It will be used when creating a new Parameter from one of its descendants, useful in some case as we will see.
Parameters	son - the Condition to construct from
Returns	none
Throws	MetaPibException - when there is an error creating the parameter

(Constructor)

Invocation	public **MetaPolicyCondition**(MetaObject parent)
Description	This constructor creates a MetaPolicyCondition from an object of the superclass MetaObject. It first calls the analog method of the parent (to copy the fields related to the parent) and after extracts or creates the specific object fields from the bytes of the *nextBytes* field. It will be used when creating a new MetaPolicyCondition from an object of its parent class, when it is creating the right object from a stream of bytes.
Parameters	parent - MetaObject to create the Parameter from
Returns	none
Throws	MetaPibException - when there is an error creating the parameter or because the bytes not correspond directly to the established fields.

transform

Invocation	public **transform**()
Description	This method overrides the homonym method of its parent class and has the same semantic adapted to a MetaPolicyCondition: will transform this general MetaPolicyCondition in a more correct, specific one, one of its descendants, depending of the prid that it has.
Parameters	none
Returns	MetaObject - the correct descendant of MetaObject (Parameter in this case) that corresponds to the prid.
Throws	none

getType

Invocation	public static int **getType**(String pridString)
Description	Overrides the homonym parent method. returns the type (restricted to the *MetaPolicyCondition* types) that corresponds to that prid
Parameters	pridString - the prid to analize
Returns	int - identifier (as definded in the *Constant* class) that corresponds to the prid.
Throws	none

getBytes

Invocation	public byte[] **getBytes**()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the object (in this case only *conditionReverse*)
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

update

Invocation	public void **update**(MetaObject newMetaObject)
Description	overrides the parent one. First, the method checks that the metaObject sent as parameter is instance of a MetaPolicyCondition. After, it checks that the prid of the object to update, and the prid of the object to update from, are equal, to maintain the consistency. After the checking, it can copy the fields from the new object to this one.
Parameters	parObject - the metaObject (that must be instance of MetaPolicyCondition) to update with.
Returns	none
Throws	MetaPibException - when there is an error updating or when the parameter *parObject* is not from the correct class (MetaPolicyCondition)

getColumn

Invocation	public byte[] **getColumn**(int column)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be obtained in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to obtain the column).
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column

Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void **updateColumn**(int column , byte[] newValue)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be updated in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to update the column).
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

isTrue

Invocation	public boolean **isTrue**()
Description	method called to evaluate whether a condition is true or not. This method must be overridden for the sub classes that inherit from this one.
Parameters	none
Returns	true - if the condition is evaluated true in a determined moment. false - otherwise
Throws	MetaPibException - if there is an error during the evaluation of the conditions tree. ImpossibleEvaluationException - if the conditions tree contains a MibPibParameter that has not been evaluated yet.

getNumericalValue

Invocation	public byte[] **getNumericalValue**()
Description	gets a numerical value of the underlying condition tree, if possible. It is useful when evaluating arithmetic conditions like (parameter1 == parameter2) This method must be overridden for the sub classes that inherit from this one.
Parameters	none
Returns	BER encoded value that represents the arithmetic value of the evaluation

Throws	MetaPibException - if there is an error during the evaluation of the conditions tree. ImpossibleEvaluationException - if the conditions tree contains a MibPibParameter that has not been evaluated yet.

ClassMetaPolicyComplexCondition

thesis.metapib.MetaPolicyComplexCondition

Description

This class extends the class MetaPolicyCondition and represents the Complex Condition of the metapolicy framework, that is used for construct more complex conditions trees.

It represents a binary condition, so it maintains a left and a right term, and also an operator to apply to these two terms.

Fields

Type	*Name*	*Description*
protected String	leftTermPrid	prid of the left term
protected String	rightTermPrid	prid of the right term
protected String	operator	operator between the two tems
protected MetaPolicyCondition	leftTerm	reference to the left term condition (for optimization purposes)
protected MetaPolicyCondition	rightTerm	reference to the right term condition (for optimization purposes)

Methods

(Constructor)

Invocation	public **MetaPolicyComplexCondition**(String conditionPrid_, boolean conditionReverse_, MetaPolicyCondition leftTerm_, String operator_, MetaPolicyCondition rightTerm_)
Description	This basic constructor creates a Condition calling the super constructor with the prid sent as first parameter, and the conditionReverse as second one. After it will fill the *operator* (calling 'checkOperator' to check that it is a valid one), and the *leftTerm*, and *rightTerm*. it will fill the *leftTermPrid* and *rightTermPrid,* obtaining the values from the *leftTerm* and *rightTerm,* respectively. the field *nextBytes* will be null.
Parameters	conditionPrid_- the prid of the condition conditionReverse_ - flag to evaluate in a reverse way (if necessary) leftTerm_ - reference to the left term condition operator_ - operator as a string rightTerm_ - reference to the right term condition

Returns	none
Throws	MetaPibException - when there is an error (for example an integrity error would be that the leftTerm or the rightTerm references the condition itself)

(Constructor)

Invocation	public **MetaPolicyComplexCondition**(String conditionPrid_, MetaPolicyCondition leftTerm_, String operator_, MetaPolicyCondition rightTerm_)
Description	this calls the basic constructor putting *false* to the *conditionReverse* parameter
Parameters	conditionPrid_- the prid of the condition leftTerm_ - reference to the left term condition operator_ - operator as a string rightTerm_ - reference to the right term condition
Returns	none
Throws	MetaPibException - when there is an error (for example an integrity error would be that the leftTerm or the rightTerm references the condition itself)

(Constructor)

Invocation	public **MetaPolicyComplexCondition**(MetaPolicyCondition leftTerm_, String operator_, MetaPolicyCondition rightTerm_)
Description	This constructor is used when we don't care about the prid in particular that it has, for example when it is needed to create automatic conditions (as we will see very useful) this calls the basic constructor with an automatically generated prid (unique), and putting *false* to the *conditionReverse* parameter.
Parameters	leftTerm_ - reference to the left term condition operator_ - operator as a string rightTerm_ - reference to the right term condition
Returns	none
Throws	MetaPibException - when there is an error (for example an integrity error would be that the leftTerm or the rightTerm references the condition itself)

(Constructor)

Invocation	public **MetaPolicyComplexCondition**(boolean conditionReverse_, MetaPolicyCondition leftTerm_, String operator_, MetaPolicyCondition rightTerm_)
Description	The same type of (auto) constructor than the previous one, but with the conditionReverse parameter too.
Parameters	conditionReverse_ - flag to evaluate in a reverse way (if necessary) leftTerm_ - reference to the left term condition operator_ - operator as a string rightTerm_ - reference to the right term condition
Returns	none
Throws	MetaPibException - when there is an error (for example an integrity error would be that the leftTerm or the rightTerm references the condition itself)

(Constructor)

Invocation	public **MetaPolicyComplexCondition**(String conditionPrid_, boolean conditionReverse_, String leftTermPrid_, String operator_, String rightTermPrid_)
Description	another constructor, but now instead of sending the term references, the parameters regarding the terms are only the prids. This constructor is used when creating a Complex Condition in the PEP from sent bytes from the PDP. We have to realize that the PDP will transmit a complex condition with the term prids established, but not the terms themselves (because that terms reference objects in the PDP context, not the PEP context). For this reason after creating a complex condition with this constructor, the term references are null, so it should be filled to maintain the consistency of the metapib framework (this will be responsibility of the METAPIB class, as we will see)
Parameters	conditionPrid - prid of the condition conditionReverse_ - flag to evaluate in a reverse way (if necessary) leftTerPrid_ - prid of the left term condition operator_ - operator as a string rightTerm_ - prid of the right term condition
Returns	none

Throws	MetaPibException - when there is an error (for example an integrity error would be that the leftTerm or the rightTerm references the condition itself)

(Constructor)

Invocation	public **MetaPolicyComplexCondition**(MetaPolicyCondition parent)
Description	This constructor creates a MetaPolicyComplexCondition from an object of the superclass MetaPolicyCondition. It first calls the analog method of the parent (to copy the fields related to the parent) and after extracts or creates the specific object fields from the bytes of the *nextBytes* field. It will be used when creating a new MetaPolicyComplexCondition from an object of its parent class, when it is creating the right object from a stream of bytes.
Parameters	parent - MetaPolicyCondition to create the Complex Condition from
Returns	none
Throws	MetaPibException - when there is an error creating the condition or because the bytes does not correspond directly to the established fields.

getBytes

Invocation	public byte[] getBytes()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the object (in this case *leftTermPrid, operator, rightTermPrid*)
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

checkOperator

Invocation	private String **checkOperator**(String operator_)
Description	this method calls the *isOperator* method of the *Constant* class to decide if the string sent as parameter is a valid operator or not.
Parameters	operator_ - String of the operator
Returns	the same operator sent as parameter
Throws	MetaPibException - if the operator is not a valid one

update

Invocation	public void update(MetaObject newMetaObject)
Description	overrides the parent one. First, the method checks that the metaObject sent as parameter is instance of a MetaPolicyComplexCondition. After, it checks that the prid of the object to update, and the prid of the object to update from, are equal, to maintain the consistency. After the checking, it can copy the fields from the new object to this one.
Parameters	parObject - the metaObject (that must be instance of MetaPolicyCondition) to update with.
Returns	none
Throws	MetaPibException - when there is an error updating or when the parameter *newMetaObject* is not from the correct class (MetaPolicyComplexCondition)

getColumn

Invocation	public byte[] getColumn(int column)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be obtained in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to obtain the column).
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column
Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void updateColumn(int column , byte[] newValue)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be updated in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to update the column).
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

isTrue

Invocation	public boolean **isTrue**()
Description	method called to evaluate whether a condition is true or not. If the operator is a logical operator (*and, or*) it will return the logical evaluation of the two terms (evaluated recursively). If is a comparison operator (==, >, <, *etc...*) it will return the correct evaluation, trying to get the values of the left and right term (recursively) and comparing them with the correct operator. This last part will be performed calling the correct comparison method of the BER class. In case of other operator (+, -), it will throw a MetaPibException, since boolean evaluation of conditions like (3 + 6) are not permitted. In case of an unknown operator it will throw a MetaPibException as well.
Parameters	none
Returns	true - if the condition is evaluated true in a determined moment. false - otherwise
Throws	MetaPibException - if there is an error during the evaluation of the conditions tree (right or left term equal to null, not known operator, etc...) ImpossibleEvaluationException - if the conditions tree contains a MibPibParameter that has not been evaluated yet.

getNumericalValue

Invocation	public byte[] getNumericalValue()
Description	gets a numerical value of the underlying condition tree, if possible. If the operator is an arithmetic one (+, -) it will return the correct value, trying to get the values of the left and right term (recursively) and operating with the correct method of the BER class. In any other case it will throw a MetaPibException
Parameters	none
Returns	BER encoded value that represents the arithmetic value of the evaluation
Throws	MetaPibException - if there is an error during the evaluation of the conditions tree, or is an unknow or not valid operator. ImpossibleEvaluationException - if the conditions tree contains a MibPibParameter that has not been evaluated yet.

ClassMetaPolicyBooleanCondition

thesis.metapib.MetaPolicyBooleanCondition

Description

This class extends the class MetaPolicyCondition and represents the Boolean Condition of the metapolicy framework, that is used for include parameters in a condition tree.

It includes a reference to a Parameter of the parameterTable, so it can be a MibPibParameter or a PdpParameter.

Fields

Type	*Name*	*Description*
protected String	parameterReferencePrid	prid of the parameter referenced
protected Parameter	parameterReference	reference to the parameter

Methods

(Constructor)

Invocation	public **MetaPolicyBooleanCondition**(String conditionPrid_, boolean conditionReverse_, String parameterReferencePrid)
Description	This basic constructor creates a Condition calling the super constructor with the prid sent as first parameter, and the conditionReverse as second one. It fills the parameterReferencePrid with the third parameter. the field *nextBytes* will be null. This constructor is used when creating a Complex Condition in the PEP from sent bytes from the PDP. It is necessary to take into consideration that this constructor will not find and fill the correct parameter reference for the parameterReferencePrid. To set that correct reference is a duty of the method that calls this constructor. (method of the METAPIB class)
Parameters	conditionPrid_- the prid of the condition conditionReverse_ - flag to evaluate in a reverse way (if necessary) parameterReferencePrid - prid of the referenced parameter
Returns	none
Throws	MetaPibException - when there is an error

(Constructor)

Invocation	public **MetaPolicyBooleanCondition**(String conditionPrid_, String parameterReferencePrid_)
Description	this calls the basic constructor putting *false* to the *conditionReverse* parameter
Parameters	conditionPrid_ - the prid of the condition parameterReferencePrid - prid of the referenced parameter
Returns	none
Throws	MetaPibException - when there is an error

(Constructor)

Invocation	public **MetaPolicyBooleanCondition**(String parameterReferencePrid_)
Description	This constructor is used when we don't care about the prid in particular that it has, for example when it is needed to create automatic conditions (as we will see very useful) this calls the basic constructor with an automatically generated prid (unique), and putting *false* to the *conditionReverse* parameter.
Parameters	parameterReferencePrid - prid of the referenced parameter
Returns	none
Throws	MetaPibException - when there is an error

(Constructor)

Invocation	public **MetaPolicyBooleanCondition**(MetaPolicyCondition parent)
Description	This constructor creates a MetaPolicyBooleanCondition from an object of the superclass MetaPolicyCondition. It first calls the analog method of the parent (to copy the fields related to the parent) and after extracts or creates the specific object fields from the bytes of the *nextBytes* field. It will be used when creating a new MetaPolicyBooleanCondition from an object of its parent class, when it is creating the right object from a stream of bytes.
Parameters	parent - MetaPolicyCondition to create the boolean Condition from
Returns	none
Throws	MetaPibException - when there is an error creating the condition or because the bytes does not correspond directly to the established fields.

getBytes

Invocation	public byte[] getBytes()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the object (in this case *parameterReferencePrid*)
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

update

Invocation	public void update(MetaObject newMetaObject)
Description	overrides the parent one. First, the method checks that the metaObject sent as parameter is instance of a MetaPolicyBooleanCondition. After, it checks that the prid of the object to update, and the prid of the object to update from, are equal, to maintain the consistency. After the checking, it can copy the fields from the new object to this one.
Parameters	parObject - the metaObject (that must be instance of MetaPolicyCondition) to update with.
Returns	none
Throws	MetaPibException - when there is an error updating or when the parameter *newMetaObject* is not from the correct class (MetaPolicyBooleanCondition)

getColumn

Invocation	public byte[] getColumn(int column)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be obtained in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to obtain the column).
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column
Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void updateColumn(int column , byte[] newValue)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be updated in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to update the column).
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

isTrue

Invocation	public boolean **isTrue**()
Description	method called to evaluate whether a condition is true or not. This method will obtain the value of the parameter and will work dependig the type of value (ber encoded): - BOOLEAN: it will return true or false depending the value - INTEGER: it will return true if value > 0 (false otherwise) - DEFAULT: it will throw a MetaPibException because it does not now how to evaluate true or false
Parameters	none
Returns	true - if the condition is evaluated true in a determined moment. false - otherwise (see description of the method)
Throws	MetaPibException - if there is an error during the evaluation of the conditions tree (parameterReference is null, error getting value) ImpossibleEvaluationException - if the conditions references a MibPibParameter that has not been evaluated yet.

getNumericalValue

Invocation	public byte[] getNumericalValue()
Description	gets the value of the parameter
Parameters	none
Returns	BER encoded value that represents the arithmetic value of the parameter
Throws	MetaPibException - if the parameter reference is null ImpossibleEvaluationException - if references a MibPibParameter that has not been evaluated yet.

ClassMetaPolicyGeneralCondition

thesis.metapib.MetaPolicyGeneralCondition

Description

This class extends the class MetaPolicyCondition and represents the General Condition of the metapolicy framework, that is used for representing xml conditions.

The general condition contains the xml condition as a xml formatted string. The way to format that condition is based on a xmlDTD, so therefore this general condition contains a reference to the object of the Xml Dtd table that is able to encode and analize that kind of formatting.

Fields

Type	*Name*	*Description*
private String	xmlDtdRefPrid	prid of the xmlDtd referenced
private String	xmlCondition	string that contains the xml formatted condition
private MetaPolicyXmlDtd	xmlDtdRef	reference to the xmlDtd object (for optimization purposes)

Methods

(Constructor)

Invocation	public **MetaPolicyGeneralCondition**(String conditionPrid_, boolean conditionReverse_, String xmlDtdRefPrid_, String xmlCondition)
Description	This basic constructor creates a Condition calling the super constructor with the prid sent as first parameter, and the conditionReverse as second one. After that It fills the xmlDtdRefPrid and xmlCondition, with the other parameters. the field *nextBytes* will be null. This constructor is used when creating a Boolean Condition in the PEP from sent bytes by the PDP. It is necessary to take into consideration that this constructor will not find and fill the correct xmlDtd reference for the xmlDtdRefPrid. To set that correct reference is a duty of the method that calls this constructor. (method of the METAPIB class)
Parameters	xmlDtdRefPrid_ - reference to the xmlDtd xmlCondition - formatted xml condition
Returns	none

Throws	MetaPibException - when there is an error

(Constructor)

Invocation	public **MetaPolicyGeneralCondition**(MetaPolicyCondition parent)
Description	This constructor creates a MetaPolicyGeneralCondition from an object of the superclass MetaPolicyCondition. It first calls the analog method of the parent (to copy the fields related to the parent) and after extracts or creates the specific object fields from the bytes of the *nextBytes* field. It will be used when creating a new MetaPolicyGeneralCondition from an object of its parent class, when it is creating the right object from a stream of bytes.
Parameters	parent - MetaPolicyCondition to create the General Condition from
Returns	none
Throws	MetaPibException - when there is an error creating the condition or because the bytes does not correspond directly to the established fields.

getBytes

Invocation	public byte[] **getBytes**()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the object (in this case *xmlDtdRefPrid, xmlCondition*)
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

update

Invocation	public void **update**(MetaObject newMetaObject)
Description	overrides the parent one. First, the method checks that the metaObject sent as parameter is instance of a MetaPolicyGeneralCondition. After, it checks that the prid of the object to update, and the prid of the object to update from, are equal, to maintain the consistency. After the checking, it can copy the fields from the new object to this one.
Parameters	newMetaObject - the metaObject (that must be instance of

	MetaPolicyGeneralCondition) to update with.
Returns	none
Throws	MetaPibException - when there is an error updating or when the parameter *newMetaObject* is not from the correct class (MetaPolicyGeneralCondition)

getColumn

Invocation	public byte[] **getColumn**(int column)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be obtained in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to obtain the column).
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column
Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void updateColumn(int column , byte[] newValue)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be updated in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to update the column).
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

isTrue

Invocation	public boolean **isTrue**()
Description	method called to evaluate whether a condition is true or not. This method actually calls the method *isTrue* of the referenced xmlDtd object, passing the xmlCondition as parameter.
Parameters	none

Returns	true - if the condition is evaluated true in a determined moment. false - otherwise
Throws	MetaPibException - if there is an error during the evaluation of the conditions tree (xmlDtd is null, error analizing xml string)

getNumericalValue

Invocation	public byte[] **getNumericalValue**()
Description	At the moment it always throws an exception, because it is not possible to obtain a numerical value from a condition formatted as a xml string.
Parameters	none
Returns	BER encoded value that represents the arithmetic value of the evaluated condition.
Throws	MetaPibException - if the parameter reference is null

ClassMetaPolicyNumberCondition

thesis.metapib.MetaPolicyNumberCondition

Description

This class extends the class MetaPolicyCondition and although it was not designed in the metapib framework, it is necessary to maintain numbers in conditions trees. For example, a condition that uses a number (i.e. 'Parameter1 > 50'), can be represented by an xmlString (and does not need this class to be maintainted in the framework); or can be represented by a tree of condition pri's, so one of the leafs of the tree will be '50', and that is the reason for this class to exist.

The number condition maintains the value of the number that represents.

Fields

Type	*Name*	*Description*
private byte[]	value	Ber encoded numeric value of the object

Methods

(Constructor)

Invocation	public **MetaPolicyNumberCondition**(String conditionPrid_, byte[] value)
Description	This basic constructor creates a Condition calling the super constructor with the prid sent as first parameter, and *false* conditionReverse. This is because a number can't be evaluated false or true, it has always a numeric value The value as parameter will be the *value* of the object. the field *nextBytes* will be null.
Parameters	conditionPrid - prid of the condition value - ber encoded value to assign
Returns	none
Throws	MetaPibException - when there is an error

(Constructor)

Invocation	public **MetaPolicyNumberCondition**(String conditionPrid_, int value)
Description	Constructor when the value of the object is an **integer**. Calls the basic constructor

Parameters	conditionPrid - prid of the condition value - ber encoded value to assign
Returns	none
Throws	MetaPibException - when there is an error

(Constructor)

Invocation	public **MetaPolicyNumberCondition**(String conditionPrid_, Stringvalue)
Description	Constructor when the value of the object is an **String** , useful for conditions like (Parameter2 == "level 0") Calls the basic constructor.
Parameters	conditionPrid - prid of the condition value - ber encoded value to assign
Returns	none
Throws	MetaPibException - when there is an error

(Constructor)

Invocation	public **MetaPolicyNumberCondition**(MetaPolicyCondition parent)
Description	This constructor creates a MetaPolicyNumberCondition from an object of the superclass MetaPolicyCondition. It first calls the analog method of the parent (to copy the fields related to the parent) and after extracts or creates the specific object fields from the bytes of the *nextBytes* field. It will be used when creating a new MetaPolicyNumberCondition from an object of its parent class, when it is creating the right object from a stream of bytes.
Parameters	parent - MetaPolicyCondition to create the General Condition from
Returns	none
Throws	MetaPibException - when there is an error creating the condition or because the bytes does not correspond directly to the established fields.

getBytes

Invocation	public byte[] **getBytes**()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the

	object (in this case *localValue*)
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

update

Invocation	public void **update**(MetaObject newMetaObject)
Description	overrides the parent one. First, the method checks that the metaObject sent as parameter is instance of a MetaPolicyNumberCondition. After, it checks that the prid of the object to update, and the prid of the object to update from, are equal, to maintain the consistency. After the checking, it can copy the fields from the new object to this one.
Parameters	newMetaObject - the metaObject (that must be instance of MetaPolicyNumberCondition) to update with.
Returns	none
Throws	MetaPibException - when there is an error updating or when the parameter *newMetaObject* is not from the correct class (MetaPolicyNumberCondition)

getColumn

Invocation	public byte[] **getColumn**(int column)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be obtained in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to obtain the column).
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column
Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void updateColumn(int column , byte[] newValue)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be updated in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it

	does not know how to update the column).
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

isTrue

Invocation	public boolean **isTrue**()
Description	method called to evaluate whether a condition is true or not. Always throws a MetaPibException because it is supposed that it is not possible to apply boolean evaluation to a Number. (This behavior can be modified)
Parameters	none
Returns	true - if the condition is evaluated true in a determined moment. false - otherwise
Throws	MetaPibException - if there is an error during the evaluation of the conditions tree (xmlDtd is null, error analizing xml string)

getNumericalValue

Invocation	public byte[] getNumericalValue()
Description	Returns directly the field *value,* that represents the value of the object
Parameters	none
Returns	BER encoded value that represents the arithmetic value of the object.
Throws	MetaPibException - if the parameter reference is null

ClassMetaPolicyAction

thesis.metapib.MetaPolicyAction

Description

This class extends the base class MetaObject and represents the general Action of the metapolicy framework. Basically it inherits the field and methods of its parent (we will see that overrides some methods to maintain the semantics)

This class will be superclass of the *MetaPolicyActionValue*, and *MetaPolicyActionParametricValue.*

Fields

Type	***Name***	***Description***
protected int	actionRefTag	tag that represents the group of actions in which this particular action is included.
protected String	actionTargetPrid	prid (of the PIB) where will be installed this action

Methods

(Constructor)

Invocation	public **MetaPolicyAction**(String actionPrid_, int actionRefTag_, String actionTargetPrid_)
Description	This constructor creates a Action calling the super constructor (of the MetaObject class) with the prid sent as first parameter, and also filling the rest of the fields (*actionRefTag, actionTargetPrid)* the field *nextBytes* will be null.
Parameters	actionPrid - prid of the action object actionRefTag - tag of the group of actions in which this one is included actionTargetPrid - prid (of the PIB) where this action will be installed
Returns	none
Throws	none

(Constructor)

Invocation	public **MetaPolicyAction**(MetaPolicyAction son)
Description	This constructor creates a MetaPolicyAction from another MetaPolicyAction. It first calls the analog method of the parent (to copy the fields

	related to the parent) and after copies all the fields of this class one by one (in this case *actionRefTag, actionTargetPrid* and *nextBytes*). It will be used when creating a new Parameter from one of its descendants, useful in some cases as we will see.
Parameters	son - the Condition to construct from
Returns	none
Throws	MetaPibException - when there is an error creating the Action

(Constructor)

Invocation	public **MetaPolicyAction**(MetaObject parent)
Description	This constructor creates a MetaPolicyCondition from an object of the superclass MetaObject. It first calls the analog method of the parent (to copy the fields related to the parent) and after extracts or creates the specific object fields from the bytes of the *nextBytes* field. It will be used when creating a new MetaPolicyAction from an object of its parent class, when it is creating the right object from a stream of bytes.
Parameters	parent - MetaObject to create the Action from
Returns	none
Throws	MetaPibException - when there is an error creating the parameter or because the bytes not correspond directly to the established fields.

transform

Invocation	public **transform**()
Description	This method overrides the homonym method of its parent class and has the same semantic adapted to a MetaPolicyAction: will transform this general MetaPolicyAction in a more correct, specific one, one of its descendants, depending of the prid that it has.
Parameters	none
Returns	MetaObject - the correct descendant of MetaObject (Parameter in this case) that corresponds to the prid.
Throws	none

getType

Invocation	public static int **getType**(String pridString)
Description	Overrides the homonym parent method. returns the type (restricted to the *MetaPolicyAction* types) that corresponds to that prid
Parameters	pridString - the prid to analize
Returns	int - identifier (as defined in the *Constant* class) that corresponds to the prid.
Throws	none

getBytes

Invocation	public byte[] **getBytes**()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the object (in this case *actionRefTag,* and *actionTargetPrid*)
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

update

Invocation	public void **update**(MetaObject newMetaObject)
Description	overrides the parent one. First, the method checks that the metaObject sent as parameter is instance of a MetaPolicyAction. After, it checks that the prid of the object to update, and the prid of the object to update from, are equal, to maintain the consistency. After the checking, it can copy the fields from the new object to this one.
Parameters	parObject - the metaObject (that must be instance of MetaPolicyAction) to update with.
Returns	none
Throws	MetaPibException - when there is an error updating or when the parameter *newMetaObject* is not from the correct class (MetaPolicyAction)

getColumn

Invocation	public byte[] **getColumn**(int column)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be obtained in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to obtain the column).
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column

Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void **updateColumn**(int column , byte[] newValue)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be updated in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to update the column).
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

Class MetaPolicyActionValue

thesis.metapib.MetaPolicyActionValue

Description

This class extends the class MetaPolicyAction and represents the Action Value of the metapolicy framework.

It contains the value that will be inserted direcly into the actionTargetPrid (field that inherits from its parent)

Fields

Type	*Name*	*Description*
protected byte[]	valueEpd	byte value of the pri that will be enforced when corresponds

Methods

(Constructor)

Invocation	public **MetaPolicyActionValue**(String actionPrid_, int actionRefTag_, String actionTargetPrid_, byte[] valueEpd_)
Description	This constructor creates a Action calling the super constructor (of the MetaObject class) with the prid sent as first parameter, and also filling the rest of the fields (*actionRefTag, actionTargetPrid, valueEpd)* the field *nextBytes* will be null.
Parameters	actionPrid - prid of the action object actionRefTag - tag of the group of actions in which this one is included actionTargetPrid - prid (of the PIB) where this action will be installed valueEpd - byte value of the pri to install
Returns	none
Throws	none

(Constructor)

Invocation	public **MetaPolicyActionValue**(MetaPolicyAction parent)
Description	This constructor creates a MetaPolicyActionValue from an object of

	the superclass MetaPolicyAction. It first calls the analog method of the parent (to copy the fields related to the parent) and after extracts or creates the specific object fields from the bytes of the *nextBytes* field. It will be used when creating a new MetaPolicyActionValue from an object of its parent class, when it is creating the right object from a stream of bytes.
Parameters	parent - MetaObject to create the Action from
Returns	none
Throws	MetaPibException - when there is an error creating the parameter or because the bytes not correspond directly to the established fields.

getBytes

Invocation	public byte[] **getBytes**()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the object (in this case *valueEpd*)
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

update

Invocation	public void **update**(MetaObject newMetaObject)
Description	overrides the parent one. First, the method checks that the metaObject sent as parameter is instance of a MetaPolicyActionValue. After, it checks that the prid of the object to update, and the prid of the object to update from, are equal, to maintain the consistency. After the checking, it can copy the fields from the new object to this one.
Parameters	parObject - the metaObject (that must be instance of MetaPolicyActionValue) to update with.
Returns	none
Throws	MetaPibException - when there is an error updating or when the parameter *newMetaObject* is not from the correct class (MetaPolicyActionValue)

getColumn

Invocation	public byte[] **getColumn**(int column)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be obtained in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to obtain the column).
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column
Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void updateColumn(int column , byte[] newValue)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be updated in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to update the column).
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

Class MetaPolicyActionParametricValue

thesis.metapib.MetaPolicyActionParametricValue

Description

This class extends the class MetaPolicyAction and represents the Action Parametric Value of the metapolicy framework.

It contains a reference to a parameter from wich the value that will be inserted directly into the actionTargetPrid will be extracted

Fields

Type	*Name*	*Description*
protected String	parameterRefPrid	prid of the parameter from wich the value of the policy will be extracted
protected Parameter	parameterRef	reference to the parameter (for optimization purposes)

Methods

(Constructor)

Invocation	public **MetaPolicyActionParametricValue**(String actionPrid_, int actionRefTag_, String actionTargetPrid_, String parameterRefPrid_)
Description	This constructor creates a Action calling the super constructor (of the MetaObject class) with the prid sent as first parameter, and also filling the rest of the fields (*actionRefTag, actionTargetPrid, parameterRefPrid)* the field *nextBytes* will be null. Also the parameterRef itself will be null when using this constructor, so it will be necessary to be filled in by the method that calls this constructor (method of the METAPIB)
Parameters	actionPrid - prid of the action object actionRefTag - tag of the group of actions in which this one is included actionTargetPrid - prid (of the PIB) where this action will be installed parameterRefPrid - prid of the parameter from which the value to be inserted in the pri of the PIB, will be inserted
Returns	none
Throws	none

(Constructor)

Invocation	public **MetaPolicyActionParametricValue**(MetaPolicyAction parent)
Description	This constructor creates a MetaPolicyActionParametricValue from an object of the superclass MetaPolicyAction. It first calls the analog method of the parent (to copy the fields related to the parent) and after extracts or creates the specific object fields from the bytes of the *nextBytes* field. It will be used when creating a new MetaPolicyActionParametricValue from an object of its parent class, when it is creating the right object from a stream of bytes.
Parameters	parent - MetaPolicyAction to create the Action from
Returns	none
Throws	MetaPibException - when there is an error creating the parameter or because the bytes not correspond directly to the established fields.

getBytes

Invocation	public byte[] **getBytes**()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the object (in this case *valueEpd*)
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

update

Invocation	public void **update**(MetaObject newMetaObject)
Description	overrides the parent one. First, the method checks that the metaObject sent as parameter is instance of a MetaPolicyActionParametricValue. After, it checks that the prid of the object to update, and the prid of the object to update from, are equal, to maintain the consistency. After the checking, it can copy the fields from the new object to this one.
Parameters	parObject - the metaObject (that must be instance of MetaPolicyActionValue) to update with.
Returns	none

Throws	MetaPibException - when there is an error updating or when the parameter *newMetaObject* is not from the correct class (MetaPolicyActionParametricValue)

getColumn

Invocation	public byte[] **getColumn**(int column)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be obtained in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to obtain the column).
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column
Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void **updateColumn**(int column , byte[] newValue)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be updated in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to update the column).
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

ClassMetaPolicy

thesis.metapib.MetaPolicy

Description

This class extends the base class MetaObject and represents the general MetaPolicy of the metapolicy framework. Basically it inherits the field and methods of its parent (we will see that overrides some methods to maintain the semantics)

We will also see that the *MetaPolicyStatus* augments this class to represent the current status of each meta-policy.

Fields

Type	***Name***	***Description***
protected String	metaPolicyName	String that identifies the policy with a name, for legibility purposes.
protected String	metaPolicyConditionPrid	prid of the condition of this MetaPolicy
protected MetaPolicyCondition	metaPolicyCondition	reference to the condition itself (for optimization purposes)
protected int	metaPolicyAction	tag that references the group of actions (policies) that will be installed when this meta-policy becomes active.
protected MetaPolicyStatus	metaPolicyStatus	reference to the Status associated to this meta-policy

Methods

(Constructor)

Invocation	public **MetaPolicy**(String metaPolicyPrid_, String metaPolicyName_, String metaPolicyConditionPrid_, int metaPolicyAction_, MetaPolicyStatus metaPolicyStatus_)
Description	This basic constructor creates a MetaPolicy calling the super constructor (of the MetaObject class) with the prid sent as first parameter, and also filling the rest of the fields (*MetaPolicyName, metaPolicyConditionPrid, metaPolicyAction, metaPolicyStatus)* It is necessary to specify a reference to a *MetaPolicyStatus,* so if we don't want to explicitly indicate this, we can use the next constructor that creates one automatically. It will be also mandatory to fill the *metaPolicyCondition* reference to the one that refers the parameter *metaPolicyConditionPrid_* (this must be done by the method that calls this method., in this case a methos of the METAPIB class) the field *nextBytes* will be null.

Parameters	metaPolicyPrid_ - prid of the meta-policy object metaPolicyName_ - desired name of the meta-policy metaPolicyConditionPrid_ - prid of the associated condition metaPolicyAction_ - tag of the associated actions metaPolicyStatus_ - reference to the associated status
Returns	none
Throws	MetaPibException - when the referenced Status cannot be associated with this metapolicy (do not have the same prid)

(Constructor)

Invocation	public **MetaPolicy**(String metaPolicyPrid_, String metaPolicyName_, String metaPolicyConditionPrid_, int metaPolicyAction_)
Description	This constructor is used when creating a meta-policy with default status, for example when is received from a PDP. It calls the basic constructor.
Parameters	metaPolicyPrid_ - prid of the meta-policy object metaPolicyName_ - desired name of the meta-policy metaPolicyConditionPrid_ - prid of the associated condition metaPolicyAction_ - tag of the associated actions
Returns	none
Throws	MetaPibException - when there is an error creating the meta-policy (see basic constructor)

(Constructor)

Invocation	public **MetaPolicy**(MetaObject parent)
Description	This constructor creates a MetaPolicy from an object of the superclass MetaObject. It first calls the analog method of the parent (to copy the fields related to the parent) and after extracts or creates the specific object fields from the bytes of the *nextBytes* field. It will be used when creating a new MetaPolicy from an object of its parent class, when it is creating the right object from a stream of bytes.
Parameters	parent - MetaObject to create the Action from
Returns	none
Throws	MetaPibException - when there is an error creating the meta-policy or because the bytes not correspond directly to the established fields.

transform

Invocation	public **transform**()
Description	This method overrides the homonym method of its parent class and has the same semantic adapted to a MetaPolicy: returns the same object, since the MetaPolicy class has no descendants
Parameters	none
Returns	MetaObject - the same object (MetaPolicy)
Throws	none

getType

Invocation	public static int **getType**(String pridString)
Description	Overrides the homonym parent method. returns the type (restricted to the *MetaPolicy* types) that corresponds to that prid. Actually it will be the type of MetaPolicy, or Unknow if does not correspond to a meta-policy
Parameters	pridString - the prid to analize
Returns	int - identifier (as defined in the *Constant* class) that corresponds to the prid.
Throws	none

getBytes

Invocation	public byte[] **getBytes**()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the object (in this case m*etaPolicyName, metaPolicyConditionPrid, metaPolicyAction,* and *metaPolicyStatus*). It sends the bytes of the metaPolicyStatus object associated (calling the method *getBytes* of that object*)*
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

update

Invocation	public void **update**(MetaObject newMetaObject)
Description	overrides the parent one. First, the method checks that the metaObject sent as parameter is instance of a MetaPolicy. After, it checks that the prid of the object to

	update, and the prid of the object to update from, are equal, to maintain the consistency. After the checking, it can copy the fields from the new object to this one.
Parameters	newMetaObject - the metaObject (that must be instance of MetaPolicy) to update with.
Returns	none
Throws	MetaPibException - when there is an error updating or when the parameter *newMetaObject* is not from the correct class (MetaPolicyAction)

getColumn

Invocation	public byte[] **getColumn**(int column)
Description	overrides the parent method but maintaining it: If the *column* is not possible to be obtained in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to obtain the column).
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column
Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void **updateColumn**(int column , byte[] newValue)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be updated in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to update the column).
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

hasToBeEnforced

Invocation	public boolean ***hasToBeEnforced***()
Description	This method calls the methos *isTrue()* of the associated condition, to evaluate if it has to be enforced or not.
Parameters	none
Returns	*True* if its associated condition is true *False* otherwise
Throws	MetaPibException - if trying to update a column that does not exist in this object

ClassMetaPolicyStatus

thesis.metapib.MetaPolicyStatus

Description

This class extends the base class MetaObject and represents the general Status of the metapolicy framework. In the design it *augments* the class MetaPolicy, so it is only semantically correct that one object of this class exists when it is associated with an object of the meta-policy class.

Declaring it as a object that descends from the MetaObject Class, we can handle it as another object, with the same inherited methods, so it is easier. Also we can reference a status as a object itself, so for example we can convey that object from PEP to PDP to communicate the status of a meta-policy,

Fields

Type	***Name***	***Description***
protected boolean	metaPolicyActive	flag to indicate that a meta-policy is active and enforced
protected boolean	metaPolicySuppress	flag to indicate that a meta-policy is suppressing or is being suppressed by another one

Depending of the values of these fields, we have the following possibilities:

metaPolicyActive Value	***metaPolicySuppress Value***	***Meaning***
TRUE	TRUE	meta-policy active, suppresses another one
TRUE	FALSE	meta-policy active, does not suppress another one
FALSE	TRUE	meta-policy inactive, suppressed by another
FALSE	FALSE	meta-policy inactive, because conditions not met

Methods

(Constructor)

Invocation	public **MetaPolicyStatus**(String metaPolicyPrid_)
Description	This basic constructor creates a MetaPolicy calling the super constructor (of the MetaObject class) with the prid sent as first parameter, and also setting the rest of the fields (*metaPolicyActive, metaPolicySuppress)* to the default values (*false*)

	the field *nextBytes* will be null.
Parameters	metaPolicyPrid_ - prid of the meta-policy object that this object represents its status
Returns	none
Throws	none

(Constructor)

Invocation	public ***MetaPolicyStatus***(MetaPolicyStatus son)
Description	This constructor creates a MetaPolicy from an object of the same class It first calls the analog method of the parent (to copy the fields related to the parent) and after copies the fields object-specific
Parameters	parent - MetaObject to create the Action from
Returns	none
Throws	MetaPibException - when there is an error creating the meta-policy or because the bytes not correspond directly to the established fields.

(Constructor)

Invocation	public ***MetaPolicyStatus***(MetaObject parent)
Description	This constructor creates a MetaPolicy from a parent object. As usual, it first calls the constructor of the parent, and after tries to get the field values form the field *netxtBytes*.
Parameters	parent - MetaObject to create the MetaObjectStatus
Returns	none
Throws	MetaPibException - when there is an error creating the meta-policyStatus or because the bytes not correspond directly to the established fields.

(Constructor)

Invocation	public ***MetaPolicyStatus***(byte bytes[])
Description	This constructor creates a MetaPolicy from an stream of bytes. It is called from the constructor of the MetaPolicy Class It first calls the analog method of the parent (to copy the fields related to the parent) and after copies the fields object-specific
Parameters	bytes - bytes from where the object will be created
Returns	none
Throws	MetaPibException - when there is an error creating the meta-policyStatus or because the bytes not correspond directly to the

	established fields.

transform

Invocation	public **transform**()
Description	This method overrides the homonym method of its parent class and has the same semantic adapted to a MetaPolicyStatus: returns the same object, since the MetaPolicyStatus class has no descendants
Parameters	none
Returns	MetaObject - the same object (MetaPolicy)
Throws	none

getType

Invocation	public static int **getType**(String pridString)
Description	Overrides the homonym parent method. returns the type (restricted to the *MetaPolicyStatus* types) that corresponds to that prid. Actually it will be the type of MetaPolicyStatus, or Unknow if does not correspond to a meta-policy
Parameters	pridString - the prid to analize
Returns	int - identifier (as defined in the *Constant* class) that corresponds to the prid.
Throws	none

getBytes

Invocation	public byte[] getBytes()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the object (in this case *metaPolicyActive* and *metaPolicySuppress*). It sends the bytes of the metaPolicyStatus object associated (calling the method *getBytes* of that object*)*
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

Note: Since this object will not be in a table/PRC of the framework, it does not need the methods update, update column, etc. Nevertheless, since one status is related with a MetaPolicy object, all this procedures can be done through this class.

ClassMetaPolicyPriority

thesis.metapib.MetaPolicyPriority

Description

This class extends the base class MetaObject and represents the MetaPolicyPriority of the metapolicy framework. Basically it inherits the field and methods of its parent (we will see that overrides some methods to maintain the semantics)

Fields

Type	*Name*	*Description*
protected String	higherPriorityPrid	prid of the metapolicy with higher priority
protected MetaPolicy	higherPriority	reference to the metapolicy with higher priority
protected String	lowerPriorityPrid	prid of the metapolicy with lower priority
protected MetaPolicy	lowerPriority	reference to the metapolicy with lower priority

Methods

(Constructor)

Invocation	public **MetaPolicyPriority**(String metaPolicyPriorityPrid_, MetaPolicy higherPriority_ , MetaPolicy lowerPriority_)
Description	This basic constructor creates a MetaPolicyPriority calling the super constructor (of the MetaObject class) with the prid sent as first parameter, and also filling the rest of the fields (*higherPriority, lowerPriority*) with the references to the objects. It also gets the prids of that objects and fills in the corresponding fields (*higherPriorityPrid, lowerPriorityPrid*) This constructor is used when we know the references to the objects
Parameters	metaPolicyPriorityPrid_ - prid of the meta-policy object higherPriority_ - reference to the MetaPolicy with higher priority lowerPriority_ - reference to the MetaPolicy with lower priority
Returns	none
Throws	none

(Constructor)

Invocation	public **MetaPolicyPriority**(String metaPolicyPriorityPrid_, String higherPriorityPrid_ , String lowerPriorityPrid_)
Description	Constructor used when we only know the prid's of the metapolicies the have higher / lower priority. It will be mandatory to fill the reference fields (*higherPriority, lowerPriority)* to ensure correct function. (This will be done by the method that calls this constructor)
Parameters	metaPolicyPriorityPrid_ - prid of the meta-policy object higherPriorityPrid_ - prid of the MetaPolicy with higher priority lowerPriorityPrid_ - prid of the MetaPolicy with lower priority
Returns	none
Throws	MetaPibException - when there is an error creating the meta-policyPriority (see basic constructor)

(Constructor)

Invocation	public **MetaPolicyPriority**(MetaObject parent)
Description	This constructor creates a MetaPolicy from an object of the superclass MetaObject. It first calls the analog method of the parent (to copy the fields related to the parent) and after extracts or creates the specific object fields from the bytes of the *nextBytes* field. It will be used when creating a new MetaPolicy from an object of its parent class, when it is creating the right object from a stream of bytes.
Parameters	parent - MetaObject to create the Priority from
Returns	none
Throws	MetaPibException - when there is an error creating the meta-policy or because the bytes not correspond directly to the established fields.

transform

Invocation	public **transform**()
Description	This method overrides the homonym method of its parent class and has the same semantic adapted to a MetaPolicyPriority: returns the same object, since the MetaPolicyPriority class has no descendants
Parameters	none
Returns	MetaObject - the same object (MetaPolicy)
Throws	none

getType

Invocation	public static int **getType**(String pridString)
Description	Overrides the homonym parent method. returns the type (restricted to the *MetaPolicyPriority* types) that corresponds to that prid. Actually it will be the type of MetaPolicyPriority, or Unknow if does not correspond to a meta-policy
Parameters	pridString - the prid to analize
Returns	int - identifier (as defined in the *Constant* class) that corresponds to the prid.
Throws	none

getBytes

Invocation	public byte[] **getBytes**()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the object (in this case *higherPriorityPrid,* and *lowerPriorityPrid*).
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

update

Invocation	public void **update**(MetaObject newMetaObject)
Description	overrides the parent one. First, the method checks that the metaObject sent as parameter is instance of a MetaPolicyPriority. After, it checks that the prid of the object to update, and the prid of the object to update from, are equal, to maintain the consistency. After the checking, it can copy the fields from the new object to this one.
Parameters	newMetaObject - the metaObject (that must be instance of MetaPolicyPriority) to update with.
Returns	none
Throws	MetaPibException - when there is an error updating or when the parameter *newMetaObject* is not from the correct class (MetaPolicyPriority)

getColumn

Invocation	public byte[] getColumn(int column)
Description	overrides the parent method but maintaining it: If the *column* is not possible to be obtained in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to obtain the column).
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column
Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void updateColumn(int column , byte[] newValue)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be updated in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to update the column).
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

ClassMetaPolicyXmlDtd

thesis.metapib.MetaPolicyXmlDtd

Description

This class extends the base class MetaObject and represents the MetaPolicyXmlDtd of the metapolicy framework. It is in charge of maintain the DTDs that the PEP is able to deal with, and the methods to analize the xml formatted strings relatef with each DTD.

Fields

Type	*Name*	*Description*
protected String	xmlDtdUrl	String that represents the URL that maintains the DTD.
XmlParseModule	parseModule	module that will parse the speciffic formatting that is related to this DTD

Methods

(Constructor)

Invocation	public **MetaPolicyXmlDtd**(String dtdPrid_, String xmlDtdUrl_)
Description	This is the basic constructor of the class. Depending of the url will try to get the correct *parseModule* (calling the method *selectParseModule)*
Parameters	dtdPrid_ - prid of the object xmlDtdUrl - url that contains the DTD
Returns	none
Throws	none

(Constructor)

Invocation	public **MetaPolicyXmlDtd**(MetaObject parent)
Description	This constructor creates a MetaPolicy from an object of the superclass MetaObject. It first calls the analog method of the parent (to copy the fields related to the parent) and after extracts or creates the specific object fields from the bytes of the *nextBytes* field. It will be used when creating a new MetaPolicyXmlDtd from an object of its parent class, when it is creating the right object from a stream of bytes.

Parameters	parent - MetaObject to create the Priority from
Returns	none
Throws	MetaPibException - when there is an error creating the metapolicyXmlDtd or because the bytes not correspond directly to the established fields.

selectParseModule

Invocation	public XmlParseModule selectParseModule(String url)
Description	depending of the *url,* this method will obtain
Parameters	url - url of the DTD
Returns	A XmlParseModule object that is able to analyze the xml formatted strings according to the *url* sent as parameter.
Throws	none

transform

Invocation	public **transform**()
Description	This method overrides the homonym method of its parent class and has the same semantic adapted to a MetaPolicyPriority: returns the same object, since the MetaPolicyXmlDtd class has no descendants
Parameters	none
Returns	MetaObject - the same object (MetaPolicy)
Throws	none

getBytes

Invocation	public byte[] **getBytes**()
Description	This method overrides the parent method and has the same semantic. It first calls the homonym parent method to get the bytes of the parent fields, and after adds the bytes corresponding to the fields of the object (in this case x*mlDtdUrl*).
Parameters	none
Returns	stream of bytes that represent the object
Throws	none

update

Invocation	public void **update**(MetaObject newMetaObject)
Description	overrides the parent one. First, the method checks that the metaObject sent as parameter is instance of a MetaPolicyXmlDtd. After, it checks that the prid of the object to update, and the prid of the object to update from, are equal, to maintain the consistency. After the checking, it can copy the fields from the new object to this one.
Parameters	newMetaObject - the metaObject (that must be instance of MetaPolicyXmlDtd) to update with.
Returns	none
Throws	MetaPibException - when there is an error updating or when the parameter *newMetaObject* is not from the correct class (MetaPolicyXmlDtd)

getColumn

Invocation	public byte[] **getColumn**(int column)
Description	overrides the parent method but maintaining it: If the *column* is not possible to be obtained in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to obtain the column).
Parameters	column- the index of the column to obtain
Returns	BER encoded value (as a byte stream) of the column
Throws	MetaPibException - if trying to get a column that does not exist in this object

updateColumn

Invocation	public void **updateColumn**(int column , byte[] newValue)
Description	overrides the parent method but mantaining it: If the *column* is not possible to be updated in this class, it will call the homonym super method (that will do the same until reaching the MetaObject class, and finally will throw a MetaPibException if it does not know how to update the column).
Parameters	column- the index of the column to update newValue - BER encoded new value of the column
Returns	none
Throws	MetaPibException - if trying to update a column that does not exist in this object

isTrue

Invocation	public boolean **isTrue**(String xmlCondition)
Description	This method is called to evaluate a *xmlCondition* thru the methods of the associated *parseModule*. In fact, it actually calls the method *isTrue* of that module.
Parameters	xmlCondition
Returns	true - if the *xmlCondition* is evaluated true false - otherwise
Throws	MetaPibException - if there is an error evaluating the xmlCondition

Class METAPIB

thesis.metapib.METAPIB

Description

This is the most important class of the package, since it coordinates all the other objects of the framework. It is responsible of the correct work of the metapib, checking its integrity and consistency. The METAPIB class is also responsible for the functionality that it is established in the design of the framework; hence, it checks the possible meta-policies that has to be enforced and also calls the corresponding methods to actually enforce the correct policies

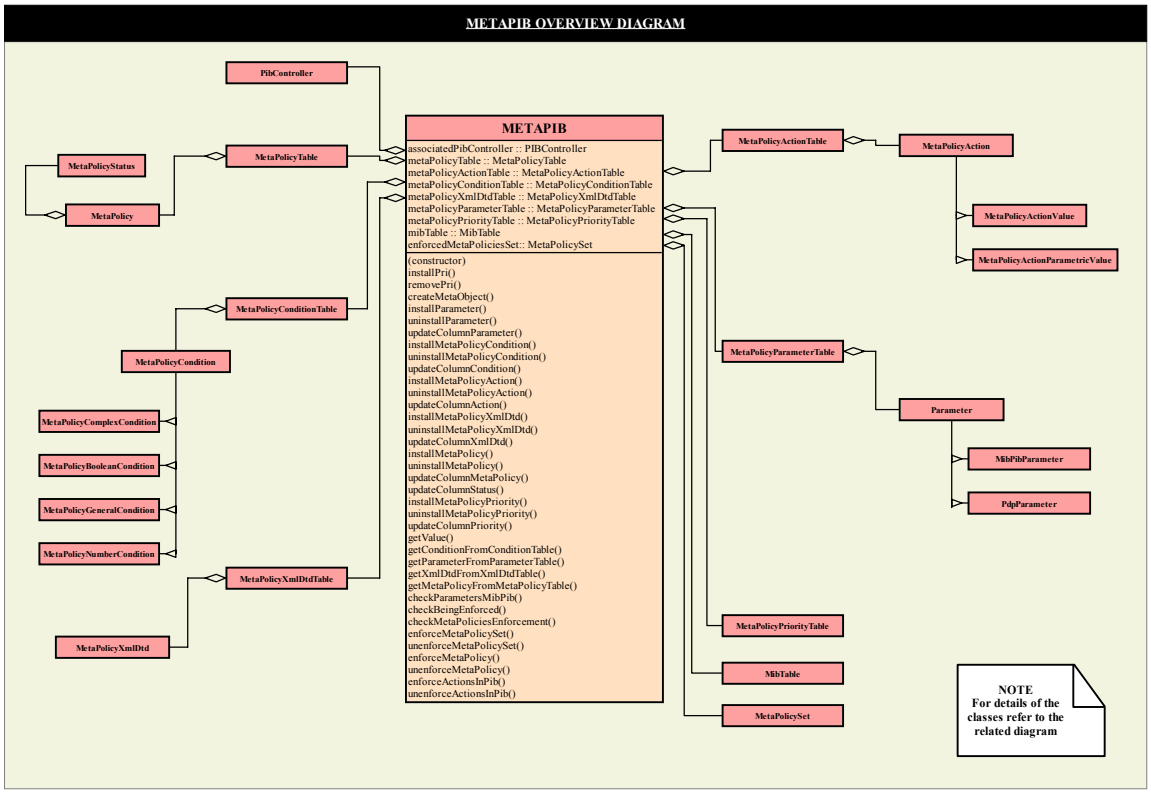

Figure 16 : MetaPib Classes Diagram

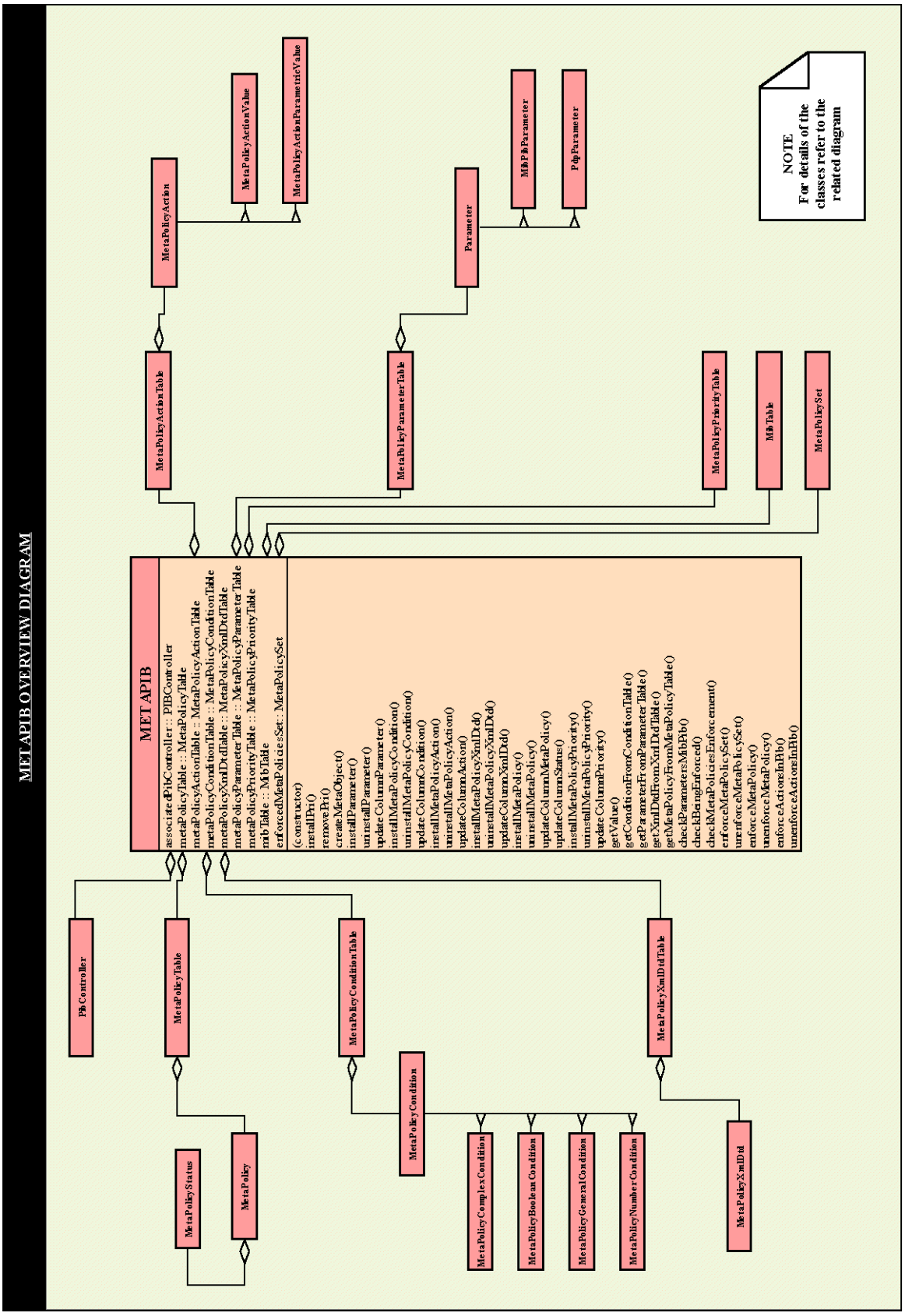

Figure 17 : MetaPib Classes Diagram (Extended)

Fields

Type	*Name*	*Description*
protected PIBController	associatedPibController	reference to the PIBController that if fact controls the METAPIB. It is necessary to send the orders to enforce the policies
protected static MetaPolicyTable	metaPolicyTable	reference to the table that contains the meta-policies.
Protected MetaPolicyActionTable	metaPolicyActionTable	reference to the table that contains the actions
protected static MetaPolicyConditionTable	metaPolicyConditionTable	reference to the table that contains the conditions
protected static MetaPolicyXmlDtdTable	metaPolicyXmlDtdTable	reference to the table that contains the XmlDtd's
protected static MetaPolicyParameterTable	metaPolicyParameterTable	reference to the table that contains the parameters
protected MetaPolicyPriorityTable	metaPolicyPriorityTable	reference to the table that contains the priorities
protected MetaPolicySet	enforcedMetaPoliciesSet	set of MetaPolicies that are enforced each moment
MibTable	mibTable	the MIB is simulated as another table of the framework. this is its reference

Methods

(Constructor)

Invocation	public **METAPIB**(PIBController pibController_)
Description	This is the constructor of the metapib. It will be called from a PibController, so the reference is passed as parameter. It is responsible for creating all the tables of the framework.
Parameters	pibController_- reference to the pibControlled that actually controls this metapib
Returns	none
Throws	MetaPibException - if there is any error creating the tables

installPri

Invocation	public void **installPri**(String prid, byte[] priData)
Description	this is the main *entry point* when installing/updating a meta-policy into the metapib. This method is called from the pibController, and this are its characteristics: It is responsible of creating the correct MetaObject from the *priData* calling the method *createMetaObject*, and also to choose its destination (the correct table for that object). For that purpose it first creates a new PriDescriptor that will contain the correct destination table, the row, type of pri, etc ... (refer to the object PriDescriptor for details) The descriptor can refer to an entry or a column of an entry, so depending of that it will call the correct method to install/update a entire entry, or update a column.
Parameters	prid - prid of the object to install-update priData- stream of bytes that represents the object
Returns	none.
Throws	MetaPibException - if the prid is incorrect, or if there is an error installing/updating the object

removePri

Invocation	public void **removePri**(String prid)
Description	this is the main *entry point* when removing a meta-policy from the metapib. This method is called from the pibController, and it is, like the installing method, responsible of creating the correct MetaObject from the *priData* calling the method *createMetaObject*, and also to choose its destination (the correct table for that object). For that purpose also creates s a new PriDescriptor (as the method *installPri)* depending of the descriptor it will call the correct method to remove the metapolicy from the METAPIB. It is not permitted to delete a row of an entry.
Parameters	prid - prid of the object to remove from the METAPIB
Returns	none
Throws	MetaPibException - if the prid is incorrect, or if there is an error removing the object, or when trying to remove only a row, as well.

createMetaObject

Invocation	private MetaObject **createMetaObject**(String prid, byte[] data)
Description	creates the correct object (Parameter, Condition, ...) from the bytes passed as parameter.
Parameters	prid - prid of the object to contruct data - stream of bytes that represent the object
Returns	the correct MetaObject created from the bytes of *data*
Throws	MetaPibException - when there is an error creating the object

installParameter

Invocation	private void **installParameter**(Parameter parObject)
Description	private method called from the *installPri* method, when the object to install/update is a Parameter
Parameters	parObject - parameter to install/update
Returns	none
Throws	MetaPibException - when there is an error installing/updating

uninstallParameter

Invocation	private void **uninstallParameter**(String parameterPrid)
Description	private method called from the remove*Pri* method, when the object to remove is a Parameter
Parameters	removePri - prid of the parameter to remove
Returns	none
Throws	MetaPibException - when there is an error removing the object, or when it was being referenced by another object

updateColumnParameter

Invocation	private void **updateColumnParameter**(String prid, byte[] priData, int row, int column)
Description	this private method is called from *installPri,* whenever it has to update a column of a Parameter. It is actually a wrapper method for the *updateColum* of the *MetaPolicyParameterTable*
Parameters	prid - prid of the object to update priData- stream of bytes that represents the object

	row - row of the object inside the table column- column to update
Returns	none
Throws	MetaPibException - if there is an error updating

installMetaPolicyCondition

Invocation	private void **installMetaPolicyCondition**(MetaPolicyCondition condition)
Description	private method called from the *installPri* method, when the object to install/update is a Condition
Parameters	condition - condition to install/update
Returns	none
Throws	MetaPibException - when there is an error installing/updating

uninstallMetaPolicyCondition

Invocation	private void **uninstallMetaPolicyCondition**(String prid)
Description	private method called from the remove*Pri* method, when the object to remove is a Condition
Parameters	prid - prid of the condition to remove
Returns	none
Throws	MetaPibException - when there is an error removing the object, or when it was being referenced by another object

updateColumnCondition

Invocation	private void **updateColumnCondition**(String prid, byte[] priData, int row, int column)
Description	this private method is called from *installPri,* whenever it has to update a column of a Parameter. It is a wrapper method for the *updateColum* of the *MetaPolicyConditionTable*
Parameters	prid - prid of the object to update priData- stream of bytes that represents the object row - row of the object inside the table column- column to update
Returns	none

Throws	MetaPibException - if there is an error updating

installMetaPolicyAction

Invocation	private void **installMetaPolicyAction**(MetaPolicyAction action)
Description	private method called from the *installPri* method, when the object to install/update is a Action
Parameters	action - action to install/update
Returns	none
Throws	MetaPibException - when there is an error installing/updating

uninstallMetaPolicyAction

Invocation	private void **uninstallMetaPolicyAction**(String prid)
Description	private method called from the remove*Pri* method, when the object to remove is a Action
Parameters	prid - prid of the condition to remove
Returns	none
Throws	MetaPibException - when there is an error removing the object, or when it was being referenced by another object

updateColumnAction

Invocation	private void **updateColumnAction**(String prid, byte[] priData, int row, int column)
Description	this private method is called from *installPri,* whenever it has to update a column of a Condition. It is a wrapper method for the *updateColum* of the *MetaPolicyActionTable*
Parameters	prid - prid of the object to update priData- stream of bytes that represents the object row - row of the object inside the table column- column to update
Returns	none
Throws	MetaPibException - if there is an error updating

installMetaPolicyXmlDtd

Invocation	private void **installMetaPolicyXmlDtd**(MetaPolicyXmlDtd xmlDtdObject)
Description	private method called frrom *installPri* , when the object to install/update is a XmlDtd
Parameters	xmlDtdObject - action to install/update
Returns	none
Throws	MetaPibException - when there is an error installing/updating

uninstallMetaPolicyXmlDtd

Invocation	private void **uninstallMetaPolicyXmlDtd**(String prid)
Description	private method called from the remove*Pri* method, when the object to remove is a XmlDtd
Parameters	prid - prid of the xmlDtd to remove
Returns	none
Throws	MetaPibException - when there is an error removing the object, or when it was being referenced by another object

updateColumnXmlDtd

Invocation	private void **updateColumnXmlDtd**(String prid, byte[] priData, int row, int column)
Description	this private method is called from *installPri,* whenever it has to update a column of a XmlDtd. It is a wrapper method for the *updateColum* of the *MetaPolicyXmlDtdTable*
Parameters	prid - prid of the object to update priData- stream of bytes that represents the object row - row of the object inside the table column- column to update
Returns	none
Throws	MetaPibException - if there is an error updating

installMetaPolicy

Invocation	private void **installMetaPolicy**(MetaPolicy metaPolicy)
Description	private method called from the *installPri* method, when the object to install/update is a MetaPolicy
Parameters	metaPolicy - action to install/update
Returns	none
Throws	MetaPibException - when there is an error installing/updating

uninstallMetaPolicy

Invocation	private void **uninstallMetaPolicy**(String prid)
Description	private method called from the remove*Pri* method, when the object to remove is a MetaPolicy
Parameters	prid - prid of the xmlDtd to remove
Returns	none
Throws	MetaPibException - when there is an error removing the object, or when it was being referenced by another object

updateColumnMetaPolicy

Invocation	private void **updateColumnMetaPolicy**(String prid, byte[] priData, int row, int column)
Description	this private method is called from *installPri,* whenever it has to update a column of a MetaPolicy. It is a wrapper method for the *updateColum* of the *MetaPolicyTable*
Parameters	prid - prid of the object to update priData- stream of bytes that represents the object row - row of the object inside the table column- column to update
Returns	none
Throws	MetaPibException - if there is an error updating

updateColumnStatus

Invocation	private void **updateColumnStatus**(String prid, byte[] priData, int row, int column)

Description	this private method is called from *installPri,* whenever it has to update a column of a Status. It actually always raises an Exception, because is not semantically correct to update the Status of a MetaPolicy from a PDP
Parameters	prid - prid of the object to update priData- stream of bytes that represents the object row - row of the object inside the table column- column to update
Returns	none
Throws	MetaPibException - always raised, because it is not semantically correct to update the Status of a MetaPolicy from a PDP

installMetaPolicyPriority

Invocation	private void **installMetaPolicyPriority**(MetaPolicyPriority priority)
Description	private method called from the *installPri* method, when the object to install/update is a Priority
Parameters	prioriry - priority to install/update
Returns	none
Throws	MetaPibException - when there is an error installing/updating

uninstallMetaPolicyPriority

Invocation	private void **uninstallMetaPolicyPriority**(String prid)
Description	private method called from the remove*Pri* method, when the object to remove is a Priority
Parameters	prid - prid of the priority to remove
Returns	none
Throws	MetaPibException - when there is an error removing the object, or when it was being referenced by another object

updateColumnPriority

Invocation	private void **updateColumnPriority**(String prid, byte[] priData, int row, int column)
Description	this private method is called from *installPri,* whenever it has to update a column of a Status.

	It is a wrapper method for the *updateColum* of the *MetaPolicyPriorityTable*
Parameters	prid - prid of the object to update priData- stream of bytes that represents the object row - row of the object inside the table column- column to update
Returns	none
Throws	MetaPibException - when there is an error updating

getValue

Invocation	public byte[] **getValue**(String prid)
Description	This method obtains the value associated with the *prid*. Nowadays it can get values for the PIB, METAPIB and MIB (the interface with the MIB is simulated with another table). If the prid refers to an entry, it returns the bytes of that entry. If the prid refers to a column of an entry, it return that concrete column.
Parameters	prid - prid of the object-column that we want to get its value
Returns	value of the object that the parameter *prid* refers to.
Throws	MetaPibException - when there is an error getting the value

getConditionFromConditionTable

Invocation	public static MetaPolicyCondition **getConditionFromConditionTable**(String conditionPrid_)
Description	this static method gets the reference to the MetaPolicyCondition that has the *conditionPrid_*, from the metaPoliciyConditionTable. It is called from some methods that has to obtain the correct reference to maintain the consistency of the metapib (for example when updating a *MetaPolicy* with a new *MetaPolicyConditionPrid,* it has to look for the installed Condition that has that prid, and emit an error if it not installed)
Parameters	conditionPrid_- the prid of the MetaPolicyCondition that we are looking for
Returns	the reference to the MetaPolicyCondition that has the indicated prid.
Throws	MetaPibException - if the condition is not installed in the MetaPolicyConditionTable

getParameterFromParameterTable

Invocation	public static Parameter **getParameterFromParameterTable**(String parameterPrid_)
Description	the same that the method above, but for Parameters.
Parameters	parameterPrid_ - the prid of the Parameter that we are looking for
Returns	the reference to the MetaPolicyParameter that has the indicated prid.
Throws	MetaPibException - if the parameter is not installed in the MetaPolicyParameterTable

getXmlDtdFromXmlDtdTable

Invocation	public static MetaPolicyXmlDtd **getXmlDtdFromXmlDtdTable**(String xmlDtdPrid_)
Description	the same that the method above, but for XmlDtd's.
Parameters	xmlDtdPrid_ - the prid of the xmlDtd that we are looking for
Returns	the reference to the XmlDtd that has the indicated prid.
Throws	MetaPibException - if the xmlDtd is not installed in the MetaPolicyXmlDtdTable

getMetaPolicyFromMetaPolicyTable

Invocation	public static MetaPolicy **getMetaPolicyFromMetaPolicyTable**(String metaPolicyPrid_)
Description	the same that the method above, but for meta-policies
Parameters	metaPolicyPrid_ - the prid of the meta-policy that we are looking for
Returns	the reference to the MetaPolicy that has the indicated prid.
Throws	MetaPibException - if the meta-policy is not installed in the MetaPolicyTable

checkParametersMibPib

Invocation	public synchronized int **checkParametersMibPib**(int milisecondsSlept)
Description	this method checks and updates the parameters that refer to a objects in the PIB or in the MIB (*MibPibParamters*). It is called continually from the PibControlled, and if any value has changed since last time, it will call the method

	checkMetaPoliciesEnforcement to see if any condition have changed, and accordingly if has to enforce/unenforce any metapolicy. Moreover it will call *checkBeingEnforced* for each updated parameter, to see if it was referenced by an enforced metapolicy, and to update to its new value.
Parameters	milisecondsSlept - miliseconds slept since this function was called last time
Returns	the number of miliseconds to sleep until next time that is necessary to call this method again.
Throws	MetaPibException - If there is an error updating a parameter calue

checkBeingEnforced

Invocation	public synchronized void **checkBeingEnforced**(String parameterPrid)
Description	With this method it is possible to check if a parameter was already in as enforced policy, and then it will update that parameter into the PIB
Parameters	parameterPrid - prid of the parameter to check
Returns	none
Throws	MetaPibException - if there is an error updating the parameter (it it was enforced)

checkMetaPoliciesEnforcement

Invocation	public synchronized void **checkMetaPoliciesEnforcemen**t()
Description	This is the main method to check and enforce (and unenforce) meta-policies, and it is called from the constantly from the PibController to ensure that the correct policies are enforced. It is also called from the same METAPIB when, for example, installs a new meta-policy, to check immediately if it has to be enforced. The functionality of this method is the next: It obtains the set of MetaPolicies susceptibles to enforce (the ones that its conditions are evaluated TRUE). First, checks if there is any enforced policy that has to be unenforced (Policies that were previosly enforced and now are NOT evaluatedTrue), calling the method *unenforceMetaPolicySet* with those that has to be unenforced. Second, it check ti enforce new policies(policies whose conditions evaluated true and are not already active), calling *enforceMetaPolicySet.*
Parameters	none
Returns	none
Throws	MetaPibException - When there is an error (see called methods like *unenforceMetaPolicySet or enforceMetaPolicySet)*

enforceMetaPolicySet

Invocation	private synchronized void **enforceMetaPolicySet**(MetaPolicySet metaPolicySet, MetaPolicySet metaPoliciesEvaluatedTrue)
Description	This general to enforce policies obtains the intersection between the two parameters (obtaining the policies that have to be enforced AND that are evaluated true); and calls the method *enforceMetaPolicy* for each meta-policy that is in the intersection.
Parameters	metaPolicySet - Set of metaPolicies that we want to enforce metaPoliciesEvaluatedTrue - Set of metaPolicies that have the condition=true in a determined moment
Returns	none
Throws	MetaPibException - If there is an error enforcing a meta-policy

unenforceMetaPolicySet

Invocation	private synchronized void **unenforceMetaPolicySet**(MetaPolicySet metaPolicySet, MetaPolicySet metaPoliciesEvaluatedTrue boolean isSuppressed by another)
Description	makes the contrary of the previous method, 'unenforcing' a set of metapolicies, calling for each one the method *unenforceMetaPolicySet*
Parameters	metaPolicySet - Set of metaPolicies that we want to un-enforce metaPoliciesEvaluatedTrue - Set of metaPolicies that have the condition=true in a determined moment isSuppressed by another - flag to determined if the reason to unenforce the policies is because it is being suppressed by another (and not because its conditions=false)
Returns	none
Throws	MetaPibException - If there is an error un-enforcing a meta-policy

enforceMetaPolicy

Invocation	private synchronized void **enforceMetaPolicy**(MetaPolicy metaPolicy, MetaPolicySet metaPoliciesEvaluatedTrue)

Description	this method checks if a condition that is able to be enforced/active (condition=true), is possible to be actually enforced (not higher policies in conflict); and puts the fields ACTIVE and SUPPRESS in its corrects values. This private method will be called from the more general method *enforceMetaPolicySet.*
Parameters	metaPolicy - metaPolicy to be enforced metaPoliciesEvaluatedTrue - Set of metaPolicies that have the condition=true in a determined moment
Returns	none
Throws	MetaPibException - If the metapolicy was already enforced

unenforceMetaPolicy

Invocation	private synchronized void **unenforceMetaPolicy**(MetaPolicy metaPolicy, MetaPolicySet metaPoliciesEvaluatedTrue, boolean isSuppressed by another)
Description	this method 'unenforces' a metapolicy (because of its condition has become false, or because it is being suppressed by another with higher priority) and puts the fields ACTIVE and SUPPRESS in its corrects values. first it retires the enforced actions/policies from the PIB (*unenforceActionsInPib)*, and after that it extracts the policy from the enforced-policies vector. Afterwards, it checks if it was suppressing another (or others), so if this is true, it will call *enforceMetaPolicySet* with each of these suppressed meta-policies, to check if each meta-policy has to be enforced, or not. This private method will be called from the more general method *unenforceMetaPolicySet.*
Parameters	metaPolicy - metaPolicy to be enforced metaPoliciesEvaluatedTrue - Set of metaPolicies that have the condition=true in a determined moment isSuppressedByAnother - flag to indicate that the unenforcing is because it is suppresed by another (and not because condition=false)
Returns	none
Throws	MetaPibException - If the metapolicy was not enforced, or there is an error deinstalling policies.

enforceActionsInPib

Invocation	private void **enforceActionsInPib**(MetaPolicy metaPolicy)
Description	this method gets the Actions of a metaPolicy and enforces them in the Pib (calling pibController)
Parameters	metaPolicy - the metapolicy to get the actions to enforce in the PIB
Returns	none
Throws	MetaPibException - if there is an error installing a determined action in the PIB

unenforceActionsInPib

Invocation	private void **unenforceActionsInPib**(MetaPolicy metaPolicy)
Description	this method works in a reverse way that *enforceActionsInPib*. It unenforces the actions from the PIB
Parameters	metaPolicy - the metapolicy to get the actions to unenforce from the PIB
Returns	none
Throws	MetaPibException - if there is an error removing a determined action in the PIB

Class ActionSet

thesis.metapib.METAPIB

Description

This class is one of the util classes that are not direcly defined in the METAPIB, but it helps in the implementation.

It represents a set of Actions, and has a number of methods to ease the deal

Fields

Type	*Name*	*Description*
Vector	set	set that contains the actions

Methods

(Constructor)

Invocation	public **ActionSet()**
Description	basic constructor that creates the set of actions
Parameters	none
Returns	none
Throws	none

addAction

Invocation	public void **addAction**(MetaPolicyAction newAction_)
Description	Adds an Action to the set
Parameters	newAction_ - action to add
Returns	none
Throws	none

size

Invocation	public int **size**()
Description	returns the number of actions in the set
Parameters	newAction_ - action to add
Returns	number of elements of the set
Throws	none

elementAt

Invocation	public MetaPolicyAction **elementAt**(int index)
Description	rmethod to obtain the Action at the indicated index
Parameters	index - index of the Action to obtain
Returns	returns a MetaPolicyAction
Throws	none

getAction

Invocation	public MetaPolicyAction**getAction**(int index)
Description	the same functionality of the above method
Parameters	index - index of the Action to obtain
Returns	returns a MetaPolicyAction
Throws	none

Class ConditionSet

thesis.metapib.METAPIB

Description

This class is another of the util classes that are not direcly defined in the METAPIB, but it helps in the implementation.

It represents a set of Conditions, and has a number of methods to ease the deal

Fields

Type	*Name*	*Description*
Vector	set	set that contains the Conditions

Methods

(Constructor)

Invocation	public **ConditionSet()**
Description	basic constructor that creates the set of Conditions
Parameters	none
Returns	none
Throws	none

addCondition

Invocation	public void **addCondition**(MetaPolicyAction newCondition_)
Description	Adds a Condition to the set
Parameters	newCondition_ - Condition to add
Returns	none
Throws	none

size

Invocation	public int **size**()
Description	returns the number of conditions in the set
Parameters	none
Returns	number of elements of the set
Throws	none

elementAt

Invocation	public MetaPolicyAction **elementAt**(int index)
Description	rmethod to obtain the Condition at the indicated index
Parameters	index - index of the Condition to obtain
Returns	returns a MetaPolicyCondition
Throws	none

getMetaPolicyCondition

Invocation	public MetaPolicyAction **getMetaPolicyCondition**(int index)
Description	the same functionality of the above method
Parameters	index - index of the Condition to obtain
Returns	returns a MetaPolicyCondition
Throws	none

Class MetaPolicySet

thesis.metapib.METAPIB

Description

This class is another of the util classes that are not direcly defined in the METAPIB, but it helps in the implementation.

It represents a set of MetaPolicies, and has a number of methods to ease the deal.

Fields

Type	*Name*	*Description*
Vector	set	set that contains the Conditions

Methods

(Constructor)

Invocation	public **MetaPolicySet**()
Description	basic constructor that creates the set of MetaPolicies
Parameters	none
Returns	none
Throws	none

addMetaPolicy

Invocation	public void **addMetaPolicy**(MetaPolicy newMetaPolicy_)
Description	Adds a MetaPolicy to the set
Parameters	newMetaPolicy_ - MetaPolicy to add
Returns	none
Throws	none

size

Invocation	public int **size**()
Description	returns the number of MetaPolicies in the set
Parameters	none
Returns	number of elements of the set
Throws	none

elementAt

Invocation	public MetaPolicy **elementAt**(int index)
Description	rmethod to obtain the MetaPolicy at the indicated index
Parameters	index - index of the MetaPolicy to obtain
Returns	returns a MetaPolicy
Throws	none

getMetaPolicy

Invocation	public MetaPolicy **getMetaPolicyCondition**(int index)
Description	the same functionality of the above method
Parameters	index - index of the MetaPolicy to obtain
Returns	returns a MetaPolicy
Throws	none

getSet

Invocation	public Vector **getSet()**
Description	returns the complete set of MetaPolicies
Parameters	none
Returns	a Vector that contains all the MetaPolicies
Throws	none

contains

Invocation	public boolean **contains**(MetaPolicy object)
Description	method that tells if a MetaPolicy is in the set
Parameters	object - MetaPolicy that is checked that is in the set
Returns	true - if the metaPolicy is in the set false - otherwise
Throws	none

removeMetaPolicy

Invocation	public boolean **removeMetaPolicy**(MetaPolicy object)
Description	removes the indicated metapolicy from the set
Parameters	object - metapolicy to remove
Returns	none
Throws	none

removeAll

Invocation	public boolean **removeAll**(MetaPolicySet removeSet)
Description	removes all the methods that are in *removeSet* from the set of MetaPolicies
Parameters	removeSet - set of policies to remove
Returns	true - if everithing is removed ok false - otherwise
Throws	none

clone

Invocation	public Object **clone()**
Description	clone interface to obtain another set exactly to this one
Parameters	none
Returns	a copy of the set
Throws	none

substraction

Invocation	public MetaPolicySet **substraction**(MetaPolicySet substractSet)
Description	substracts the *substractSet* from the set of metapolicies. It asctually calls the method *removeAll* to remove the policies, and after returns the result.
Parameters	substractSet - set of metapolicies to substract
Returns	the set of metapolicies, after substracting the ones that are in the parameter *substractSet*
Throws	MetaPibException - If there is any error in the substraction procedure

intersection

Invocation	public MetaPolicySet **intersection**(MetaPolicySet intersectSet)
Description	method to obtain the intersection between the metaPolicySet, and the set sent as paramenter
Parameters	intersectSet - set of metapolicies to intersect
Returns	a set of metapolicies that contains the intersection between the ones that are in the paramenter, and the set of metaPolicies
Throws	none

getActives

Invocation	public MetaPolicySet **getActives**()
Description	returns the metapolicies that are actives from the ones that are in the set
Parameters	none
Returns	a new set that contains the metapolicies that are actives (in this moment)
Throws	MetaPibException - If there is any error

getNotActives

Invocation	public MetaPolicySet **getActives**()
Description	The reverse method of the previous one.
Parameters	none
Returns	a new set that contains the metapolicies that are NOT actives (in this moment)
Throws	MetaPibException - If there is any error

containsActionRef

Invocation	public boolean **containsActionRef**(int actionTag)
Description	method to check if the actionTag sent as paramenter is being referenced by any of the metapolicies of the set
Parameters	actionTag - tag to check
Returns	true - if at least one metapolicy of the set contains the actionTag
Throws	none

getActionsEnforced

Invocation	public ActionSet **getActionsEnforced**(ActionSet actionSet)
Description	method to obtain the actions that are enforced in one moment, calling the method *ContainsActionRef* to check if the action is being referenced (and adding to the resultSet, if so)
Parameters	actionSet - actions that we want to check
Returns	a new Action set that contains the Actions that are enforced in a determined moment
Throws	none

Class XmlDtdSet

thesis.metapib.XmlDtdSet

Description

This class is another of the util classes that are not direcly defined in the METAPIB, but it helps in the implementation.

It represents a set of XmlDtd objects, and has a number of methods to ease the deal

Fields

Type	*Name*	*Description*
Vector	set	set that contains the XmlDtd objects

Methods

(Constructor)

Invocation	public **XmlDtdSet()**
Description	basic constructor that creates the set of XmlDtd objects
Parameters	none
Returns	none
Throws	none

addXmlDtd

Invocation	public void **addXmlDtd**(MetaPolicyAction newXmlDtd_)
Description	Adds a Condition to the set
Parameters	newXmlDtd_ - XmlDtd object to add
Returns	none
Throws	none

size

Invocation	public int **size**()
Description	returns the number of XmlDtd objects in the set
Parameters	none
Returns	number of elements of the set
Throws	none

elementAt

Invocation	public MetaPolicyXmlDtd **elementAt**(int index)
Description	rmethod to obtain the Condition at the indicated index
Parameters	index - index of the Condition to obtain
Returns	returns a MetaPolicyCondition
Throws	none

getMetaPolicyCondition

Invocation	public MetaPolicyXmlDtd **getXmlDtd**(int index)
Description	the same functionality of the above method
Parameters	index - index of the Condition to obtain
Returns	returns a MetaPolicyCondition
Throws	none

Class PriDescriptor

thesis.metapib.PriDescriptor

Description

This class is not defined in the design of the Metapib framework, but it is useful in the implementation, so it is declared as a util class.

It represents a prid and its characteristics, so we can use it to obtain information about the object that is referring a particular prid.

All the identifiers that we refer to, are defined in the CONTANT class.

Fields

Type	***Name***	***Description***
protected String	effectivePrid	Prid of the accesed entry (if applicable)
protected int	addessSpace	addressSpace that it is refering: METAPIB, PIB, OTHER (MIB)
protected int	priType	type of the pri, inside its *addressSpace*: COLUMN, ENTRY, ALL (THE ENTRIES OF A TABLE)
protected int	tableId	Identifier of the table (if applicable)
protected int	row	row that it is refering (if aplicable)
protected int	column	column that it is refering (if aplicable)

Methods

(Constructor)

Invocation	public **PriDescriptor**(String prid)
Description	basic constructor that creates a PriDescriptor from a prid. It will analize the prid to obtain as much information as possible, filling in all the applicable fields
Parameters	prid - prid to analize
Returns	none
Throws	MetaPibException - error analizing prid

checkPriTypeMib

Invocation	private void **checkPriTypeMib**(String prid)
Description	private method called form the constructor when the pri address space corresponds to the MIB. It will continue the analizing and fill the applicable fields (*addressSpace).* Afterwards it will call the method *checkRestPrid* with the part of the prid not analized yet, to fill the rest of the fields
Parameters	prid - prid to analize
Returns	none
Throws	MetaPibException - error analizing prid

checkPriTypePib

Invocation	private void **checkPriTypePib**(String prid)
Description	private method called form the constructor when the pri address space corresponds to the PIB. It will continue the analizing and fill the applicable fields (*addressSpace).* Afterwards it will call the method *checkRestPrid* with the part of the prid not analized yet, to fill the rest of the fields
Parameters	prid - prid to analize
Returns	none
Throws	MetaPibException - error analizing prid

checkPriTypePib

Invocation	private void **checkPriTypePib**(String prid)
Description	private method called form the constructor when the pri address space corresponds to the METAPIB. It will continue the analizing and fill the applicable fields (*addressSpace).*Also it will try to discern the concrete table of the METAPIB that the prid is refering to Afterwards it will call the method *checkRestPrid* with the part of the prid not analized yet, to fill the rest of the fields
Parameters	prid - prid to analize
Returns	none
Throws	MetaPibException - error analizing prid

checkRestPrid

Invocation	private int **checkRestPrid**(String restPrid)
Description	checks the rest of the prid looking for applicable information. This method will try to identify if it is refering to a COLUMN, and ENTRY, or ALL the table. With the result it will fill the field *priType*. It will also obtain the correct *effectivePrid* to access the object (usually the prid of the table that it is accessing.
Parameters	restPrid - rest of the prid to analize what type (*priType)* is.
Returns	the type (according to the *priType value)* of the prid.
Throws	MetaPibException - error analyzing

chopPrid

Invocation	public static String **chopPrid**(String PRID, String prefix)
Description	method to chop the *prefix* from the PRID
Parameters	PRID - prid value prefix - prefix to chop
Returns	the result from the chopping.
Throws	MetaPibException - error chopping the prid

TABLES PACKAGE

The objects in this package represent the related tables of the Metapib and its associated methods. Each table is descendant of the basic class *Table*, which contains the common methods.

Each descendant object contains particular methods that serve with any specific functionality.

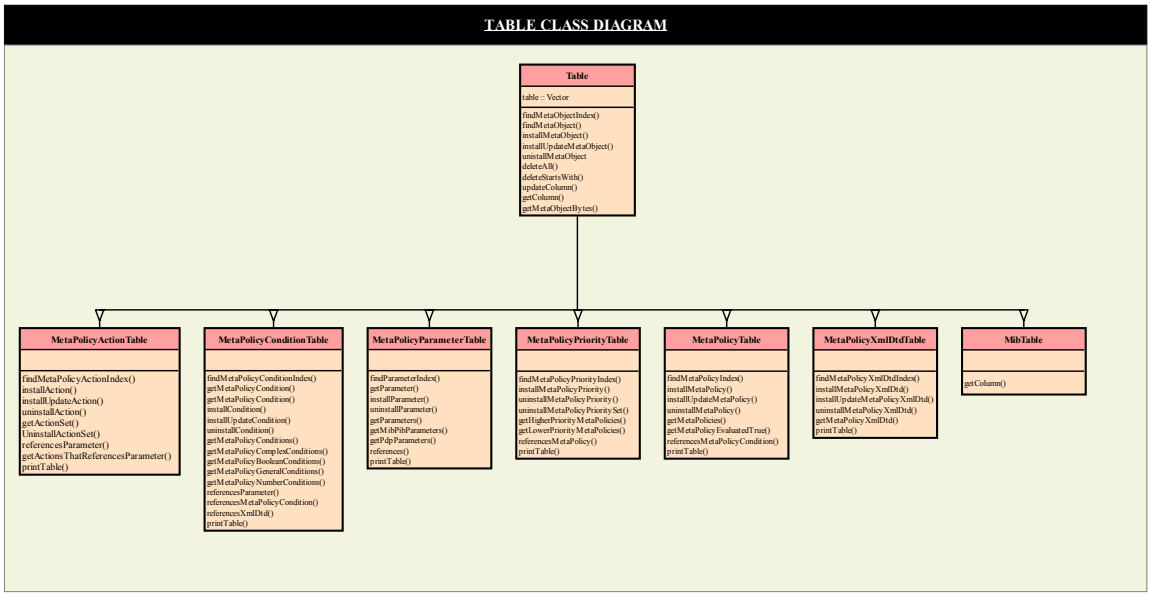

Figure 18 : Table Class Diagram

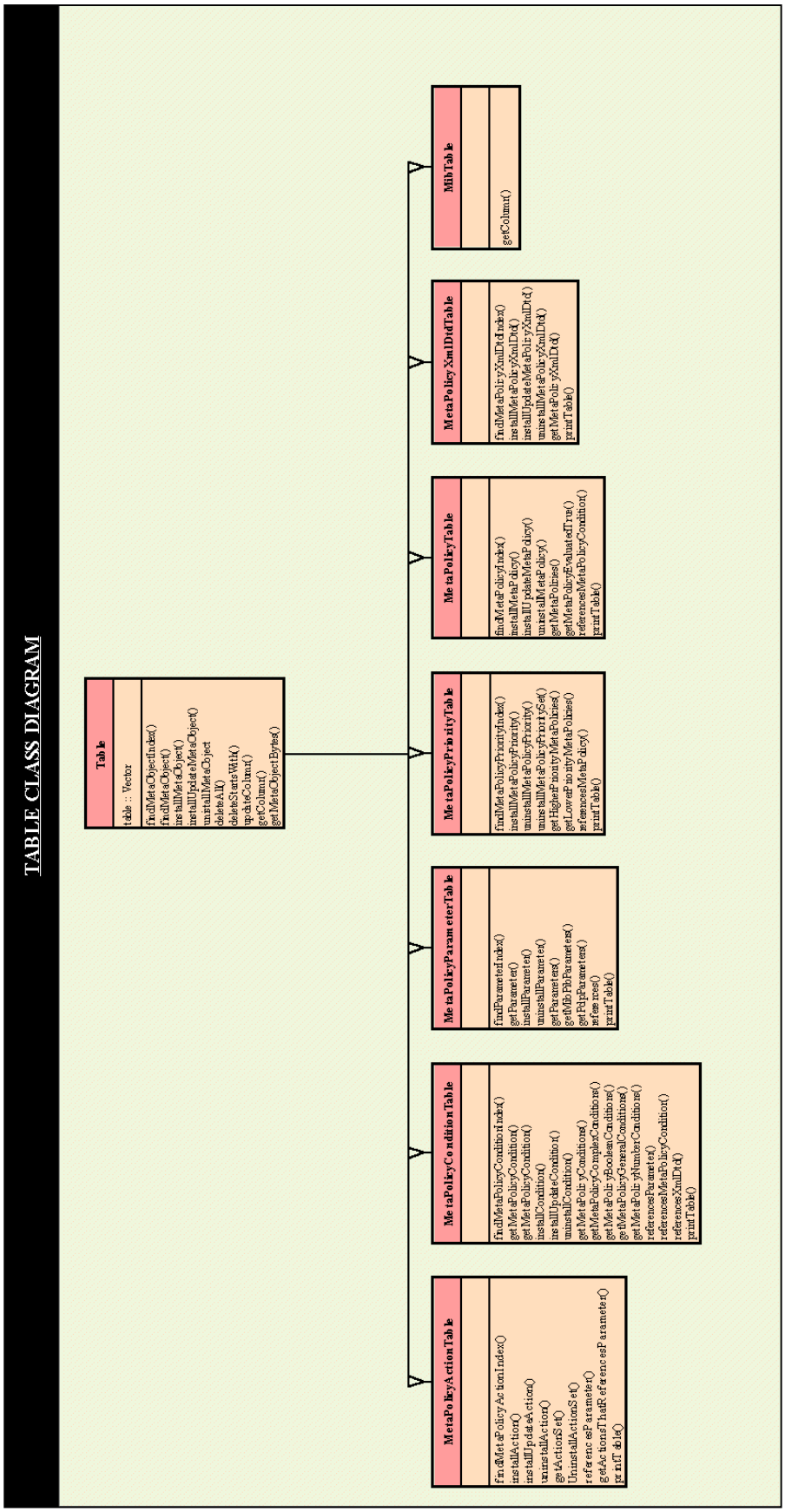

Figure 19 Table Class Diagram (Extended)

Class Table

thesis.metapib.tables.Table

Description

The Table class refers to the basic table that maintains MetaObjects. It can be considered as a PRC that will be parent of all the other PRCs (ConditionTable, ActionTable..) of the METAPIB framework.

Actually it is implemented as a vector where we can allocate all the objects, but it can be modified to be implemented as a Hash Table, etc ...

Fields

Type	*Name*	*Description*
Vector	table	table that maintains all the MetaObjects

Methods

(Constructor)

Invocation	public public **Table()**
Description	this is the basic constructor that sets the table (Vector)
Parameters	none
Returns	none
Throws	none

findMetaObjectIndex

Invocation	public int **findMetaObjectIndex**(String searchedPrid_)
Description	method to get the index of a looked MetaObject (by its prid)
Parameters	searchedPrid - prid of the MetaObject to look for
Returns	integer >= 0 - index (inside the table) of the object found -1 - object not found
Throws	none

findMetaObject

Invocation	public int **findMetaObject**(int index)
Description	method to get the a reference to a looked MetaObject (by its index in the table)
Parameters	index - index of the looked Object inside the table
Returns	the searched MetaObject d, or null if is not in the table

Throws	none

findMetaObject

Invocation	public MetaObject **findMetaObject**(String searchedPrid_)
Description	method to get the a reference to a looked MetaObject (by its prid)
Parameters	searchedPrid - prid of the MetaObject to look for
Returns	the searched MetaObject , or null if is not in the table
Throws	none

installMetaObject

Invocation	public void **installMetaObject**(MetaObject metaObject_)
Description	method to install a MetaObject in the table. This method does not updates a previously installed one (see *intallUptateMetaObject* instead.
Parameters	metaObject - object to install in the table
Returns	none
Throws	MetaPibException - if there is any error installing (for example if the object was already in the table)

installUpdateMetaObject

Invocation	public void **installUpdateMetaObject**(MetaObject metaObject_)
Description	method to install or update a MetaObject in the table.
Parameters	metaObject - object to install/update in the table
Returns	none
Throws	MetaPibException - if there is any error installing or updating

uninstallMetaObject

Invocation	public void **uninstallMetaObject**(string searchedPrid_)
Description	method to uninstall a MetaObject from the table.
Parameters	searchedPrid - prid of the object to remove.
Returns	none
Throws	MetaPibException - if the object was not in the table

deleteAll

Invocation	public void **deleteAll**()
Description	method to delete all the object of the table
Parameters	none
Returns	none
Throws	none

deleteStartsWith

Invocation	public void **deleteStartsWith**(String prefix)
Description	method to delete all the object of the table that its prid starts with the *prefix*
Parameters	prefix - prefix of the objects that we want to delete
Returns	none
Throws	none

updateColumn

Invocation	public void **updateColumn**(String prid, byte[] priData, int row, int column)
Description	method to update a determined column, from a determined object of the table.
Parameters	prid - prid of the object to update row - row of the object in the table (actually not used in the final implementation) column- column of the object to update pridData - data to update with
Returns	none
Throws	MetaPibException - if there is any error updating

getColumn

Invocation	public byte[] **getColumn**(String prid, int row, int column)
Description	method to get the value of a determined column, of a determined object
Parameters	prid - prid of the object to get the column row - row of the object in the table (actually not used in the final implementation column- column to get
Returns	stream of bytes with the data of the requested column
Throws	MetaPibException - if there is any error obtaining the column

getMetaObjectBytes

Invocation	public byte[] getMetaObjectBytes(String prid)
Description	with this method we can obtain the representation of the object with the requested *prid*, as a byte stream. It actually first find the object (*findMetaObject*), and afterwards it calls its particular *getBytes* method.
Parameters	prid - prid of the object that we want its bytes
Returns	stream of bytes that represents the object.
Throws	none

Class MetaPolicyActionTable

thesis.metapib.tables.MetaPolicyActionTable

Description

This class represents the table where all the MetaPolicyActions are maintained, and of course, is a descendant of the class *Table*.

It inherits the majority of the methods of its parent class, and also defines new that deal with specific issues of the MetaPolicyActions. Also, in some cases, there are some wrappers to parent functions, to maintain consistency with the syntax of the class.

Fields

No specific defined fields

Methods

(Constructor)

Invocation	public public **MetaPolicyActionTable()**
Description	this is the basic constructor that sets the table (Vector). It actually calls the constructor of the parent
Parameters	none
Returns	none
Throws	none

findMetaPolicyActionIndex

Invocation	public int **findMetaPolicyActionIndex**(String searchedPrid_)	
Description	method to get the index of a looked MetaPolicyAction (by its prid). It is actually a wrapper for the parent method *findMetaObjectIndex*	
Parameters	searchedPrid	- prid of the MetaObject to look for
Returns	integer >= 0 -1	- index (inside the table) of the object found - object not found
Throws	none	

installAction

Invocation	public void **installAction**(MetaPolicyAction action_)
Description	Wrapper to the parent function *installMetaObject.* The functionality is the same: install a MetaObject in the table. This method does not update a previously installed one (see

	intallUptateAction instead.
Parameters	action - MetaPolicyAction to install in the table
Returns	none
Throws	MetaPibException - if there is any error installing (for example if the object was already in the table)

installUpdateAction

Invocation	public void **installUpdateAction**(MetaPolicyAction action_)
Description	Wrapper to the parent function *installUpdateMetaObject* Functionality: install or update a MetaPolicyAction in the table.
Parameters	action - MetaPolicyAction to install/update in the table
Returns	none
Throws	MetaPibException - if there is any error installing or updating

uninstallAction

Invocation	public void **uninstallAction**(String actionPrid_)
Description	Wrapper to the parent function *uninstallMetaObject* method to uninstall a MetaObject from the table.
Parameters	actionPrid_ - prid of the MetaPolicyAction to remove.
Returns	none
Throws	MetaPibException - if the MetaPolicyAction was not in the table

getActionSet

Invocation	public ActionSet **getActionSet**(int actionTag_)
Description	Method to get the actions from the table that have the indicated actionTag
Parameters	actionTag - actionTag that we are looking for
Returns	a new ActionSet that contains the MetaPolicyActions from the table that has the *actionTag*.
Throws	MetaPibException - if *actionTag* is not a positive number

uninstallActionSet

Invocation	public void **uninstallActionSet**(int actionTag_)
Description	Method to uninstall the actions from the table that have the indicated actionTag
Parameters	actionTag - actionTag that we are looking for
Returns	none
Throws	MetaPibException - if *actionTag* is not a positive number

referencesParameter

Invocation	public boolean **referencesParameter**(String parameterPrid_)
Description	Method that returns true if there is any Action (MetaPolicyActionParametricValue) hat references the *parameterPrid_*
Parameters	parameterPrid - prid that we want to check
Returns	none
Throws	none

referencesParameter

Invocation	public boolean **referencesParameter**(ParameterSet parameterSet)
Description	Method that returns true if there is any Action (MetaPolicyActionParametricValue) that references ANY of the Parameters contained in *parameterSet*
Parameters	parameterSet - set of parameters that we want to check whether they are referenced or not
Returns	none
Throws	none

getActionsThatReferencesParameter

Invocation	public ActionSet **getActionsThatReferenceParameter**(String parameterPrid_)
Description	Method that returns all the Actions (MetaPolicyActionParametricValue) that reference the *parameterPrid_*
Parameters	parameterPrid - prid of the parameter that we want to check
Returns	a new ActionSet with all the Actions that reference the parameter
Throws	none

printTable

Invocation	public void **printTable()**
Description	This is a method to show (print on the screen) all the objects that are installed in a determined moment. It will format correctly according with the fields of the table
Parameters	none
Returns	none
Throws	none

Class MetaPolicyConditionTable

thesis.metapib.tables.MetaPolicyConditionTable

Description

This class represents the table where all the MetaPolicyConditions are maintained, and is again descendant of the class *Table*.

As the previous defined table for the Actions, this class inherits the majority of the methods of its parent class, and defines new that deal with specific issues of the MetaPolicyCondition as well. Also, in some cases, there are some wrappers to parent functions, to maintain consistency with the syntax of the class.

Fields

No specific defined fields

Methods

(Constructor)

Invocation	public public **MetaPolicyConditionTable()**
Description	this is the basic constructor that sets the table (Vector). It actually calls the constructor of the parent
Parameters	none
Returns	none
Throws	none

findMetaPolicyConditionIndex

Invocation	public int **findMetaPolicyConditionIndex**(String searchedPrid_)	
Description	method to get the index of a looked MetaPolicyCondition (by its prid). It is actually a wrapper for the parent method *findMetaObjectIndex*	
Parameters	searchedPrid	- prid of the MetaObject to look for
Returns	integer >= 0 -1	- index (inside the table) of the object found - object not found
Throws	none	

getMetaPolicyCondition

Invocation	public iMetaPolicyCondition **getMetaPolicyCondition**(String searchedPrid_)
Description	method to get a MetaPolicyCondition (by its prid). It is actually a

	wrapper for the parent method *findMetaObjectIndex*
Parameters	searchedPrid - prid of the MetaPolicyCondition to look for
Returns	a MetaPolicy Condition, or null
Throws	none

getMetaPolicyCondition

Invocation	public MetaPolicyCondition **getMetaPolicyCondition**(int index)
Description	method to get a MetaPolicyCondition (by its prid). It is actually a wrapper for the parent method *findMetaObjectIndex*
Parameters	index - index of the MetaPolicyCondition in the table
Returns	a MetaPolicy Condition, or null
Throws	none

installCondition

Invocation	public void **installCondition**(MetaPolicyCondition action_)
Description	Wrapper to the parent function *installMetaObject.* The functionality is the same: install a MetaPolicyCondition in the table. This method does not update a previously installed one (see *intallUptateAction* instead.
Parameters	action - MetaPolicyCondition to install in the table
Returns	none
Throws	MetaPibException - if there is any error installing (for example if the object was already in the table)

installUpdateCondition

Invocation	public void **installUpdateCondition**(MetaPolicyCondition action_)
Description	Wrapper to the parent function *installUpdateMetaObject* Functionality: install or update a MetaPolicyCondition in the table.
Parameters	action - MetaPolicyAction to install/update in the table
Returns	none
Throws	MetaPibException - if there is any error installing or updating

uninstallCondition

Invocation	public void **uninstallCondition**(String actionPrid_)
Description	Wrapper to the parent function *uninstallMetaObject* method to uninstall a MetaObject from the table.
Parameters	actionPrid_ - prid of the MetaPolicyCondition to remove.
Returns	none

Throws	MetaPibException - if the MetaPolicyCondition was not in the table

getMetaPolicyConditions

Invocation	public ActionSet **getMetaPolicyConditions**()
Description	returns all the installed MetaPolicyConditions
Parameters	none
Returns	a new ConditionSet with all the MetaPolicyConditions of the table
Throws	none

getMetaPolicyComplexConditions

Invocation	public ActionSet **getMetaPolicyConditions**()
Description	returns all the installed MetaPolicyComplexConditions
Parameters	none
Returns	a new ConditionSet with all the MetaPolicyComplexConditions of the table
Throws	none

getMetaPolicyBooleanConditions

Invocation	public ActionSet **getMetaPolicyBooleanConditions**()
Description	returns all the installed MetaPolicyBooleanConditions
Parameters	none
Returns	a new ConditionSet with all the MetaPolicyBooleanConditions of the table
Throws	none

getMetaPolicyGeneralConditions

Invocation	public ActionSet **getMetaPolicyGeneralConditions**()
Description	returns all the installed MetaPolicyGeneralConditions
Parameters	none
Returns	a new ConditionSet with all the MetaPolicyGeneralConditions of the table
Throws	none

getMetaPolicyNumberConditions

Invocation	public ActionSet **getMetaPolicyGeneralConditions**()
Description	returns all the installed MetaPolicyNumberConditions
Parameters	none
Returns	a new ConditionSet with all the MetaPolicyNumberConditions of the table

Throws	none

referencesParameter

Invocation	public boolean **referencesParameter**(String parameterPrid_)
Description	Method that returns true if there is any Condition (MetaPolicyBooleanCondition) that references the *parameterPrid_*
Parameters	parameterPrid - prid that we want to check
Returns	none
Throws	none

referencesParameter

Invocation	public boolean **referencesParameter**(ParameterSet parameterSet)
Description	Method that returns true if there is any Condition (MetaPolicyBooleanCondition) that references ANY of the Parameters contained in *parameterSet*
Parameters	parameterSet - set of parameters that we want to check whether they are referenced or not
Returns	none
Throws	none

referencesMetaPolicyCondition

Invocation	public boolean **referencesMetaPolicyCondition**(String prid_)
Description	Method that returns true if there is any Condition (complexCondition) that references the Condition with the *prid*
Parameters	prid_ - prid of the Condition that we want to check
Returns	true - if any ComplexCondition referenced the condition with the *prid.*
Throws	none

referencesXmlDtd

Invocation	public boolean **referencesXmlDtd**(String xmlDtdPrid)
Description	Method that returns true if there is any Condition (generalCondition) that references the XmlDtd with the *xmlDtdPrid*
Parameters	prid_ - prid of the XmlDtd that we want to check
Returns	true - if any generalCondition referenced the xmlDtd with the *prid.*
Throws	none

referencesXmlDtd

Invocation	public boolean **referencesXmlDtd**(XmlDtdSet xmlDtdSet)
Description	Method that returns true if there is any Condition (generalCondition) that references any of the XmlDtd objects contained in the *xmlDtdSet*
Parameters	xmlDtdSet - Set of XmlDtd objects that we want to check
Returns	true - if any generalCondition referenced the xmlDtd with the *prid*.
Throws	none

printTable

Invocation	public void **printTable()**
Description	This is a method to show (print on the screen) all the objects that are installed in a determined moment. It will format correctly according with the fields of the table
Parameters	none
Returns	none
Throws	none

Class MetaPolicyParameterTable

thesis.metapib.tables.MetaPolicyActionTable

Description

This class represents the table where all the MetaPolicyParameters are maintained, and of course, is a descendant of the class *Table*.

It inherits the majority of the methods of its parent class, and also defines new that deal with specific issues of the Parameters. Also, in some cases, there are some wrappers to parent functions, to maintain consistency with the syntax of the class.

Fields

No specific defined fields

Methods

(Constructor)

Invocation	public public **MetaPolicyParameterTable()**
Description	this is the basic constructor that sets the table (Vector). It actually calls the constructor of the parent
Parameters	none
Returns	none
Throws	none

findParameterIndex

Invocation	public int **findParameterIndex**(String searchedPrid_)	
Description	method to get the index of a looked MetaPolicyParameter(by its prid). It is actually a wrapper for the parent method *findMetaObjectIndex*	
Parameters	searchedPrid	- prid of the Parameter to look for
Returns	integer >= 0 -1	- index (inside the table) of the object found - object not found
Throws	none	

getParameter

Invocation	public Parameter **getParameter**(String searchedPrid_)
Description	method to get a Parameter (by its prid). It is actually a wrapper for the

	parent method *findMetaObjectIndex*
Parameters	searchedPrid - prid of the Parameter to look for
Returns	a Parameter reference, or null
Throws	none

getParameter

Invocation	public Parameter **getParameter**(int index)
Description	method to get a Parameter (by its index). It is actually a wrapper for the parent method *findMetaObjectIndex*
Parameters	index - index of the Parameter in the table
Returns	a Parameter reference, or null
Throws	none

installParameter

Invocation	public void **installParameter**(MetaPolicyParameter parameter_)
Description	Wrapper to the parent function *installMetaObject.* The functionality is the same: install a MetaObject (in this case its descendant class, Parameter) in the table. This method does not update a previously installed one.
Parameters	parameter - Parameter to install in the table
Returns	none
Throws	MetaPibException - if there is any error installing (for example if the object was already in the table)

uninstallParameter

Invocation	public void **uninstallParameter**(String actionPrid_)
Description	Wrapper to the parent function *uninstallMetaObject* method to uninstall a Paramter from the table, by its prid
Parameters	parameter_ - prid of the Parameter to remove.
Returns	none
Throws	MetaPibException - if the Parameter was not in the table

uninstallParameter

Invocation	public void **uninstallParameter**(Parameter parameter_)
Description	Wrapper to the parent function *uninstallMetaObject* method to uninstall a Paramter from the table, by the reference.

Parameters	parameter_ - reference to the Parameter to remove.
Returns	none
Throws	MetaPibException - if the Parameter was not in the table

getParameters

Invocation	public ParameterSet **getParameters**()
Description	Method to get all the installed parameters.
Parameters	none
Returns	a new ParameterSet that contains the Parameters.
Throws	none

getMibPibParameters

Invocation	public ParameterSet **getMibPibParameters**()
Description	Method to get all the installed MibPibParameters.
Parameters	none
Returns	a new ParameterSet that contains the Parameters.
Throws	none

getPdpParameters

Invocation	public ParameterSet **getPdpParameters**()
Description	Method to get all the installed PdpParameters.
Parameters	none
Returns	a new ParameterSet that contains the Parameters.
Throws	none

references

Invocation	public boolean **referencesParameter**(String prid_)
Description	Method that returns true if there is at least one MibPib in the table thath references *prid_*
Parameters	prid_ - prid that we want to check
Returns	none
Throws	none

printTable

Invocation	public void **printTable()**
Description	This is a method to show (print on the screen) all the objects that are

	installed in a determined moment. It will format correctly according with the fields of the table
Parameters	none
Returns	none
Throws	none

Class MetaPolicyPriorityTable

thesis.metapib.tables.MetaPolicyPriorityTable

Description

This class represents the table where all the MetaPolicyPriorities are maintained, and of course, is a descendant of the class *Table*.

It inherits the majority of the methods of its parent class, and also defines new that deal with specific issues of the MetaPolicyPriorities. Also, in some cases, there are some wrappers to parent functions, to maintain consistency with the syntax of the class.

Fields

No specific defined fields

Methods

(Constructor)

Invocation	public public **MetaPolicyPriorityTable()**
Description	this is the basic constructor that sets the table (Vector). It actually calls the constructor of the parent
Parameters	none
Returns	none
Throws	none

findMetaPolicyPriorityIndex

Invocation	public int **findMetaPolicyPriorityIndex**(String searchedPrid_)
Description	method to get the index of a looked MetaPolicyPriority (by its prid). It is actually a wrapper for the parent method *findMetaObjectIndex*
Parameters	searchedPrid - prid of the Priority to look for
Returns	integer >= 0 - index (inside the table) of the object found -1 - object not found
Throws	none

installMetaPolicyPriority

Invocation	public void **installMetaPolicyPriority**(MetaPolicyPriority priority_)
Description	Wrapper to the parent function *installMetaObject.* The functionality is the same: install a MetaObject in the table. This

	method does not update a previously installed one
Parameters	action - MetaPolicyPriority to install in the table
Returns	none
Throws	MetaPibException - if there is any error installing (for example if the object was already in the table)

uninstallMetaPolicyPriority

Invocation	public void **uninstallMetaPolicyPriority**(String metaPolicyPriorityPrid_)
Description	Wrapper to the parent function *uninstallMetaObject* method to uninstall a MetaPolicyPriority from the table.
Parameters	metaPolicyPriorityPrid - prid of the MetaPolicyPriority to remove
Returns	none
Throws	MetaPibException - if the MetaPolicyAction was not in the table

uninstallMetaPolicyPrioritySet

Invocation	public void **uninstallMetaPolicyPrioritySet**(String metaPolicy2RemovePrid_)
Description	this method removes the MetaPolicyPriority objects that refer (as higher or lower priority) to the MetaPolicy with prid equal to *metaPolicy2RemovePrid_*
Parameters	metaPolicy2RemovePrid_ - prid of the MetaPolicy that is refering
Returns	none
Throws	MetaPibException - if the MetaPolicyAction was not in the table

getHigherPriorityMetaPolicies

Invocation	public MetaPolicySet **getHigherPriorityMetaPolicies**(MetaPolicy referenceMetaPolicy)
Description	This is a method to obtain the Metapolicies that higher defined priority that the one that we sent as parameter (*referenceMetaPolicy)*
Parameters	referenceMetaPolicy - the prid of the MetaPolicy that we want to obtain the higher Priority MetaPolicies
Returns	A new MetaPolicySet that contains the MetaPolicies with higher priority that the reference one
Throws	none

getLowerPriorityMetaPolicies

Invocation	public MetaPolicySet **getLowerPriorityMetaPolicies**(

	MetaPolicy referenceMetaPolicy)
Description	This is a method to obtain the Metapolicies that lower defined priority that the one that we sent as parameter (*referenceMetaPolicy)*
Parameters	referenceMetaPolicy - the prid of the MetaPolicy that we want to obtain the lower Priority MetaPolicies
Returns	A new MetaPolicySet that contains the MetaPolicies with lower priority that the reference one
Throws	none

referencesMetaPolicy

Invocation	public boolean **referencesMetaPolicy**(String metaPolicyPrid_)
Description	Method that returns true if there is any *MetaPolicyPriority* object that refers (as higher, or lower) to the MetaPolicy with prid equal to metaPolicyPrid_
Parameters	metaPolicyPrid - prid that we want to check
Returns	true - if there is at least one object that refers to that metaPolicy
Throws	none

referencesMetaPolicy

Invocation	public boolean **referencesMetaPolicy**(MetaPolicySet metaPolicySet)
Description	Method that returns true if there is any MetaPolicyPriority object that refers (as higher, or lower) any of the MetaPolicy objects contained in the set *metaPolicySet*
Parameters	metaPolicySet - set of MetaPolicies that we want to check
Returns	true - if there is at least one object that refers to that metaPolicy
Throws	none

printTable

Invocation	public void **printTable()**
Description	This is a method to show (print on the screen) all the objects that are installed in a determined moment. It will format correctly according with the fields of the table
Parameters	none
Returns	none
Throws	none

Class MetaPolicyTable

thesis.metapib.tables.MetaPolicy

Description

This class represents the table where all the MetaPolicy are maintained, and of course, is a descendant of the class *Table*.

This is one of the most important tables because it maintains the MetaPolicies that will be checked for possible enforcement. For this purpose, this class will implement some methods that facilitate this.

Fields

No specific defined fields

Methods

(Constructor)

Invocation	public public **MetaPolicyXmlDtdTable()**
Description	this is the basic constructor that sets the table (Vector). It actually calls the constructor of the parent
Parameters	none
Returns	none
Throws	none

findMetaPolicyIndex

Invocation	public int **findMetaPolicyIndex**(String searchedPrid_)	
Description	method to get the index of a looked MetaPolicy (by its prid). It is actually a wrapper for the parent method *findMetaObjectIndex*	
Parameters	searchedPrid	- prid of the Priority to look for
Returns	integer >= 0 -1	- index (inside the table) of the object found - object not found
Throws	none	

installMetaPolicy

Invocation	public void **installMetaPolicy**(MetaPolicy metaPolicy_)
Description	Wrapper to the parent function *installMetaObject.* The functionality is the same: install a MetaObject in the table. This

	method does not update a previously installed one (see method *installUpdateMetaPolicy*)
Parameters	metaPolicy_ - MetaPolicy to install in the table
Returns	none
Throws	MetaPibException - if there is any error installing (for example if the object was already in the table)

installUpdateMetaPolicy

Invocation	public void **installUpdateMetaPolicy**(MetaPolicy metaPolicy_)
Description	this method install or updates a MetaPolicy
Parameters	metaPolicy_ - MetaPolicy to install in the table
Returns	none
Throws	MetaPibException - if there is any error installing

uninstallMetaPolicy

Invocation	public void **uninstallMetaPolicy**(String metaPolicyPrid_)
Description	Wrapper to the parent function *uninstallMetaObject* method to uninstall a MetaPolicy from the table.
Parameters	metaPolicyPriority - prid of the MetaPolicy to remove
Returns	none
Throws	MetaPibException - if the MetaPolicy was not in the table

getMetaPolicies

Invocation	public MetaPolicySet **getMetaPolicies**()
Description	This is a method to obtain all the installed Metapolicies
Parameters	none
Returns	A new MetaPolicySet that contains all the MetaPolicies of the table
Throws	none

getMetaPolicyEvaluatedTrue

Invocation	public MetaPolicySet **getMetaPolicyEvaluatedTrue**()
Description	This is a method to obtain the Metapolicies susceptibles to be enforce. For each MetaPolicy of the table, it checks if its condition is true, and if this is the case, it adds to the result set.
Parameters	none
Returns	A new MetaPolicySet that contains the MetaPolicies that are

	evaluated true.
Throws	none

referencesMetaPolicyCondition

Invocation	public boolean **referencesMetaPolicyCondition**(String prid_)
Description	Method that returns true if there is any Condition, of the installed MetaPolicies, that is the same as the sent *prid*
Parameters	prid - the prid of the Condition that we want to check
Returns	true - if there is at least one MetaPolicy whose condition is the one that we are checking false - otherwise
Throws	MetaPibException - If there is any error (for example an integrity error when a MetaPolicy has not any Condition)

referencesMetaPolicyCondition

Invocation	public boolean **referencesMetaPolicyCondition**(ConditionSet conditionSet_)
Description	Method that returns true if there is any Condition, of the installed MetaPolicies, that is equal to any of the Conditions of the set.
Parameters	prid - the prid of the Condition that we want to check
Returns	true - if there is at least one condition that refers to that condition false - otherwise
Throws	MetaPibException - If there is any error (for example an integrity error when a MetaPolicy has not any Condition)

printTable

Invocation	public void **printTable()**
Description	This is a method to show (print on the screen) all the objects that are installed in a determined moment. It will format correctly according with the fields of the table
Parameters	none
Returns	none
Throws	none

Class MetaPolicyXmlDtdTable

thesis.metapib.tables.MetaPolicyXmlDtdTable

Description

This class represents the table where all the MetaPolicyXmlDtd objects are maintained, and of course, is a descendant of the class *Table*.

Fields

No specific defined fields

Methods

(Constructor)

Invocation	public **MetaPolicyXmlDtdTable()**
Description	this is the basic constructor that sets the table (Vector). It actually calls the constructor of the parent
Parameters	none
Returns	none
Throws	none

findMetaPolicyIndex

Invocation	public int **findMetaPolicyXmlDtdIndex**(String searchedPrid_)
Description	method to get the index of a looked MetaPolicyXmlDtd (by its prid). It is actually a wrapper for the parent method *findMetaObjectIndex*
Parameters	searchedPrid - prid of the Priority to look for
Returns	integer >= 0 - index (inside the table) of the object found -1 - object not found
Throws	none

installMetaPolicy

Invocation	public void **installMetaPolicyXmlDtd**(MetaPolicyXmlDtd metaPolicyXmlDtd_)
Description	Wrapper to the parent function *installMetaObject.* The functionality is the same: install a MetaObject (XmlDtd objects) in the table. This method does not update a previously installed one (see method *installUpdateMetaPolicy*)

Parameters	metaPolicy_ - MetaPolicy to install in the table
Returns	none
Throws	MetaPibException - if there is any error installing (for example if the object was already in the table)

installUpdateMetaPolicyXmlDtd

Invocation	public void **installUpdateMetaPolicyxmlDtd**(MetaPolicy metaPolicy_)
Description	this method install or updates a MetaPolicyXmlDtd object
Parameters	metaPolicy_ - MetaPolicy to install in the table
Returns	none
Throws	MetaPibException - if there is any error installing

uninstallMetaPolicyXmlDtd

Invocation	public void **uninstallMetaPolicyXmlDtd**(String searchedPrid_)
Description	Wrapper to the parent function *uninstallMetaObject* method to uninstall a MetaPolicy from the table.
Parameters	searchedPrid - prid of the MetaPolicyXmlDtd to remove
Returns	none
Throws	MetaPibException - if the MetaPolicyXmlDtd was not in the table

getMetaPolicyXmlDtd

Invocation	public MetaPolicySet **getMetaPoliciesXmlDtd**(String searchedPrid_)
Description	This is a (another) method to obtain a MetaPolicyXmlDtd by its prid
Parameters	searchedPrid_ - the prid of the MetaPolicyXmlDtd that we are looking for
Returns	The MetaPolicy that has the prid that we are looking.
Throws	MetaPibException - error getting the MetaPolicyXmlDtd

printTable

Invocation	public void **printTable()**
Description	This is a method to show (print on the screen) all the objects that are installed in a determined moment. It will format correctly according with the fields of the table
Parameters	none
Returns	none

Throws	none

Class MibTable

thesis.metapib.tables.MibTable

Description

This class represents an API to access the MIB of the PEP.
The access is exactly the same as another table, so it is easier to implement. Nowadays, and because we need to simulate, the MIB is represented as (physical) files that contains the individual elements of the MIB.

That is, if for example the MIB contains three parameters, we will have three different files that contain the value of those parameters. The access to that files, is through the method *getColumn. D*epending of the column that we access, it will get the value from the real file.

This class could be superseded by the real API to the MIB on the PIB, when it be executed in the real PEP.

Fields

No specific defined fields

Methods

(Constructor)

Invocation	public **MibTable()**
Description	this is the basic constructor that sets the table (Vector). It actually calls the constructor of the parent, and also installs all the (artificial) objects that represent the parameters of the Mib that we can access.
Parameters	none
Returns	none
Throws	MetaPibException - if there is any error when installing the (artificial) objects

getColumn

Invocation	public byte[] **getColumn**(String prid, int row, int column)
Description	This method overrides the parent one, and depending of the column that we are accessing it will provide access to the file that corresponds.

Parameters	prid - prid that we want to access row - row that we want to access column- column of the object that we will access
Returns	stream of bytes that represent the value of the column
Throws	none

7 CONCLUSIONS AND FUTURE WORK

This chapter complete the work described in this final project describing the final results achieved the end of the project. We also introduce some possible future guidelines to work in the same line that the present work.

7.1 CONCLUSIONS

This work presented the current situation in some aspects of network management and also introduced some new ideas that were only some actuation guidelines when I began to be interested in this area.

The idea of the MetaPolicies is a new trend that has been recently presented and that was only a design idea before the position of this project.

With the conclusion of this project, It has been demonstrated that the area of policy based networking is very interesting and still lots of work can be directed in this domain.

At the end of this work, a complete framework of policy based networking based in MetaPolicies has been completely developed, implemented and tested. The functioning of this framework is of high quality, and the obtained results are very promising for the future of the Internet.

7.2 FUTURE WORK

Although the work is complete in the meaning of the a functioning framework and it has been tested with some simulations, a testing environment with real routers and big networks would be necessary to obtain the real performance measurement of the introduced approach.

Some utilities like a graphic console to control and check the framework would be desirable, and in fact is in the schedule for the next immediate work in the Broadband Communications Research Group of the University Of Waterloo.

References

[1]. **"Introduction to Policy-based Networking and Quality of Service";** IPHighway, White paper, January 2001

[2]. **"Policy Standards and IETF Terminology";** IPHighway, White paper, January 2001.

[3]. **"Policy Based Networking Products, Design and Architecture";** IPHighway, White paper, February 2001.

[4]. **"Target Market and Case Studies";** IPHighway, White paper, February 2001.

[5]. **"Policy Based Networking"**; Smartpipes Inc., White paper,

[6]. D. Kakadia, **"Enterprise Qos Policy Based Sytems & Network Management";** Sun Microsystems, White Paper.

[7]. **"FCAPS OVERVIEW";** http://www.fore.com/products/fv/fc_fcaps_wp.html **(September 2000)**

[8]. J. Young, **"A Framework for Providing Differential Services in Policy-Based Networking";** Thesis, October 1999.

[9]. **"Delivering End-to-End Security in Policy-Based Networks",** Cisco Systems, WhitePaper, 1998.

[10]. **"A Primer on Policy-based Network Management";**Hewlett-Packard, White Paper, September 1999.

[11]. **"Introduction to QoS policies"** Stardust.com, Whitepaper, July 1999.

[12]. Nichols, K., Blake, S., Baker, F., Black, D., **"Definition of the Differentiated Services Field (DS Field)in the IPv4 and IPv6 Headers"**, IETF RFC 2474, Proposed Standard, December 1998.

[13]. Bernet, Y., Yavatkar R., Ford, P., Baker, F., Nichols, K., Speer, M**., "A Framework for End-to-End QoS Combining RSVP/Intserv and Differentiated Services"**, IETF , November 1998.

[14]. A. Westerinen, J. Schnizlein,J. Strassner, Mark Scherling,Bob Quinn,Jay Perry, Shai Herzog, An-Ni Huynh, Mark Carlson, Steve Waldbusser; **"Terminology";** *IETF, Internet-Draft, draft-ietf-policy-terminology-02.txt, November 2000 (*http://www.ietf.org/internet-drafts/draft-ietf-policy-terminology-02.txt*)*

[15]. **"Internet Engineering Task Force";** *http://www.ietf.org/*

[16]. **"Policy-Powered Networking and the Role of Directories";** *3COM, White paper, July 1998.*

[17]. Susan J. Shepard; **"Policy-based networks: hype and hope";** *IT Professional, Vol. 2, No. 1, January-February 2000, pp.12 –16*

[18]. **"Introduction to Policy-based Networking and Quality of Service";** *IPHighway, White paper, January 2000*

[19]. R. Boutaba, K. El-Guemhioui, P. Dini; **"An Outlook on Intranet Management";** *IEEE Communications Magazine, Special issue on Intranet Services and Communication Management, October 1997, pp.92-97*

[20]. R. Boutaba, S. Znaty, **"An Architectural Approach for Integrated Networks and Systems Management";** *ACM-SIGCOM Computer Communication Review, Vol. 25, No 5, October 1995, pp. 13-39*

[21]. M. Sloman; **"Policy Driven Management For Distributed Systems";** *International Journal of Network and Systems Management, Vol. 2, No. 4, December 1994, pp. 333-360*

[22]. **"Policy Standards and IETF Terminology";** *IPHighway, White paper, January 2000.*

[23]. S. Herzog, Ed.J.Boyle, R.Cohen, D.Durhan, R.Rajan, A.Sastry, **"COPS usage for RSVP"** IETF, RFC 2749, January 2000

[24]. **"Policy Based Networking Products, Design and Architecture";** *IPHighway, White paper, January 2000.*

[25]. D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry; **"The COPS (Common Open Policy Service) Protocol";** IETF*, RFC 2748, January 2000; (*http://www.ietf.org/rfc/rfc2748.txt*)*

[26]. **"Resource Allocation Protocol (rap)";** http://www.ietf.org/html.charters/rap-charter.html

[27]. **"Intel COPS client Software Development Kit";** http://www.intel.com/ial/cops/

[28]. **"IPHighway – COPS open source";** http://www.iphighway.com/opensource1.htm

[29]. **"COPS Download Page";** http://www.vovida.org/protocols/downloads/cops/

[30]. K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, A. Smith; **"COPS Usage for Policy Provisioning";** IETF*, RFC 3084, March 2001 (*http://www.ietf.org/rfc/rfc3084.txt*)*

[31]. M. Fine, K. McCloghrie, J. Seligson, K. Chan, S. Hahn, A. Smith, F. Reichmeyer; **"Differentiated Services Quality of Service Policy Information Base";** IETF*, Internet-Draft, draft-ietf-diffserv-pib-03.txt, March 2001 (*http://www.ietf.org/internet-drafts/draft-ietf-diffserv-pib-03.txt*)*

[32]. D. Rawlins, A. Kulkarni, K. Ho Chan, D. Dutt, **"Framework of COPS-PR Policy Information Base for Accounting Usage";** IETF*, Internet-Draft, draft-ietf-rap-acct-fr-pib-01.txt, July 2000 (*http://www.ietf.org/internet-drafts/draft-ietf-rap-acct-fr-pib-01.txt*)*

[33]. J. Ottensmeyer, M. Bokaemper, K. Roeber; **"A Filtering Policy Information Base (PIB) for Edge Router Filtering Services and Provisioning via COPS-PR";** IETF*, Internet-Draft, draft-otty-cops-pr-filter-pib-00.txt, November 2000 (*http://www.ietf.org/internet-drafts/draft-otty-cops-pr-filter-pib-00.txt*)*

[34]. M. Li, D. Arneson, A. Doria, J. Jason, C. Wang; **"IPSec Policy Information Base";** *IETF, Internet-Draft, draft-ietf-ipsp-ipsecpib-02.txt, March 2001 (*http://www.ietf.org/internet-drafts/draft-ietf-ipsp-ipsecpib-02.txt*)*

[35]. Harsha Hegde, Brad Stone; **"Load Balancing Policy Information Base";** *IETF, Internet-Draft, draft-hegde-load-balancing-pib-00.txt, February 2001 (http://www.ietf.org/internet-drafts/ draft-hegde-load-balancing-pib-00.txt)*

[36]. M. Fine, K. McCloghrie, J. Seligson, K. Chan; S. Hahn, R. Sahita, A. Smith, F. Reichmeyer; **"Framework Policy Information Base",** IETF*, Internet-Draft, draft-ietf-rap-frameworkpib-04.txt, November 2000 (*http://www.ietf.org/internet-drafts/draft-ietf-rap-frameworkpib-04.txt*)*

[37]. K. McCloghrie, M. Fine, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith, F. Reichmeyer; **"Structure of Policy Provisioning Information (SPPI)";** IETF, *Internet-Draft, draft-ietf-rap-frameworkpib-06.txt, February 2001 (http://www.ietf.org/internet-drafts/ draft-ietf-rap-frameworkpib-06.txt)*

[38]. A. Polyrakys, R.Boutaba: **"The Meta Policy Information Base ", IETF, draft-ietf-rap-mpib-00.txt**

ARTICLES AND PRESENTATIONS:

[39]. S. Hezog, **"Policy Based Networking for QoS";** TF-TANT, Presentation, April 2000.

[40]. J. Strassner, **"Policy-Based Networking";** Cisco System, Presentation.

[41]. J. Strassner, **"Policy-Based Networking Standards Report";** Cisco System, Presentation.

[42]. J. Fukuda, K. Iseda, M. Minoura, H. Ueno, T. Chujo, **"Policy-based Networking Service over Heterogeneus Public IP Networks (DynaServ)";** Fujitsu Laboratories LTD., Presentacion.

[43]. J. Conover, **"Policy-Based Network Management",** Article, (http://www.networkcomputing.com/1024/1024f1.html), November 1999.

[44]. J. Conover, **"Policy-Based Network Management Solution Features",** Comparison table, (http://www.networkcomputing.com/1024/1024f1.html), November 1999.

LINKS TO PBN-RELATED-PRODUCTS:

[45]. **"Intel COPS Client SDK";** (http://developer.intel.com/ial/cops/index.htm)

[46]. **"CiscoAssure Policy Manager" ;** (http://www.cisco.com/warp/public/cc/pd/nemnsw/cap/index.shtml)

[47]. **"Cisco COPS Qos Policy Manager Data Sheet, version 2.0" ;** (http://www.cisco.com/warp/public/cc/pd/nemnsw/cap/index.shtml)

[48]. **"Cisco Qos Policy Manager"**; (http://www.cisco.com/warp/public/cc/pd/nemnsw/cap/index.shtml)

[49]. **"Jade Communications Products – Policy Based Networking";** (http://www.cisco.com/warp/public/cc/pd/nemnsw/cap/index.shtml)

[50]. **"Allot – bandwidth management, QoS, service level agreement';** (http://www.allot.com)

[51]. **"Hewlett Packard – policyExpert";** (http://www.openview.hp.com/products/policyexpert/index.asp)

[52]. **"Check Point Flood Gate";** **(**http://www.openview.hp.com/products/policyexpert/index.asp)

[53]. **"Solaris Bandwidth Manager";** (http://www.openview.hp.com/products/policyexpert/index.asp)

BOOKS :

[54]. D. Kosiur, S. Herzog, **"Understanding Policy-Based Networking";** Book ISBN: 0471388041, Wiley Computer Publishing, January 2001.

[55]. D. C. Verma, "**Policy-Based Networking: Architecture and Algorithms";** Book ISBN: 1578702267, New Riders Publishing, November 2000.