Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FIN DE CARRERA

TÍTULO: Creación de un entorno para gestión avanzada de red

AUTOR: Eduardo Manrique Garcia

DIRECTOR: Mª Angélica Reyes Muñoz

FECHA: 5 de Septiembre de 2005

Título: Creación de un entorno para gestión avanzada de red

Autor: Eduardo Manrique Garcia

Director: Mª Angélica Reyes Muñoz

Fecha: 5 de Septiembre de 2005

Resumen

En este proyecto se desarrollará un entorno de gestión de políticas que utilizará como repositorio una implementación Open Source del protocolo LDAP (Lightweight Directory Access Protocol) para acceso a directorios llamado openLDAP[1]. El openLDAP requiere un backend o motor de búsqueda para almacenar y hacer búsquedas de la información. Para ello, se utiliza una herramienta también Open Source llamada BerkeleyDB (BerkeleyDataBase)[2].

La introducción y visualización de datos en el openLDAP es por consola, pero al no ser una forma muy amigable de utilización, usaremos un programa en modo visual y totalmente gratuito denominado LDAP browser/editor[3]. Con este programa podremos en cualquier momento introducir de una forma fácil las políticas a aplicar, visualizarlas, modificarlas o borrarlas.

Finalmente, se creará un programa que demuestre la utilización de la gestión basada en políticas, así como su simplicidad y escalabilidad. Este programa leerá las políticas del openLDAP y actuará primero como punto de decisión, llamado por la IETF (Internet Engineering Task Force)[4] Policy Decision Point (PDP) donde se decidirán las políticas que actuarán. Esta actuación será ejecutada por un segundo punto al que la IETF llama Policy Enforcement Point (PEP).

El programa será creado en el lenguaje JAVA ya que, aparte de ser un potente y flexible lenguaje orientado a objetos, necesitaremos la utilización del API JNDI (Java Naming and Directory Interface)[5][6] que nos permite acceder a los directorios y por tanto será el que nos permita hacer la lectura de las políticas guardadas en el openLDAP para finalmente aplicarlas.

Title: Creation of an environment for advanced network management

Author: Eduardo Manrique Garcia

Director: Mª Angelica Reyes Muñoz

Date: 5 de Septiembre de 2005

Overview

In this proyect we develop a policy managament enviroment that uses an Open Source Implementation of the LDAP (Lightweight Directory Access Protocol) called openLDAP[1] To acces directories. OpenLDAP requires a backend or search engine to store and search information; for this, we use an Open Source tool called BerkeleyDB (BerkeleyDataBase)[2].

The introduction and visualization of information in the openLDAP is mainly provided for console and due that it is not really friendly. We use a free visual program called LDAP browser/editor[3]. With this program we easily introduce the policies, which have to be applied, we also can visualize them, modify or even to erase them.

Finally, there will be created a program that shows the utilization of the Policy-Based Management, as well as his simplicity and scalability. This program read policies from the openLDAP acting first as a decision point about which policies are going to be applied. The IETF (Internet Engineering Task Force) called it policy decision point (PDP)[4], where will be decided the policies to acting. Selected actions will be executed by the second point, called by the IETF Policy Enforcement Point (PEP).

The application will be created in the JAVA language because apart from being powerful and a flexible object – oriented language, we will need his API JNDI (Java Naming and Directory Interface)[5][6] who allows us to access directories and to read policies stored in the openLDAP and to finally applying them.

ÍNDICE

INTRODUCCIÓN

En esta introducción se van a explicar cuáles han sido las razones y motivaciones para la realización de este proyecto, así como los principales objetivos que se han fijado en su desarrollo.

En el primer capítulo se presentará el marco de trabajo en el que se desarrolla este proyecto. Primero se hace una introducción de los sistemas gestores de red existentes, enfatizando en los sistemas gestores de red basadas en políticas (PBNM). Luego se hablará de los diferentes modelos de políticas, desde el Policy Core Information Model (PCIM)[7] que presenta cómo estructurar la información de políticas, hasta el Policy Core Extensions LDAP Schema (PCELS)[8] que presenta cómo introducir las políticas en el LDAP así como los esfuerzos de estandarización de los diferentes organismos como el Internet Engineering Task Force (IETF) y el Distributed Management Task Force (DMTF)[9].

En el segundo capítulo hablaremos sobre qué es una política, su diseño y edición. El capítulo estará dividido en dos partes bien diferenciadas. Primero se hablará de los sistemas o lenguajes específicos que se han creado especialmente para la edición de políticas como por ejemplo el Ponder[10]. En segundo lugar se hablará sobre los lenguajes de programación ya existentes que permiten la edición de políticas como el JNDI (Java Naming and Directory Interface).

En el tercer capítulo ya se comenzará a hablar de la aplicación del proyecto en sí. El primer punto en el cual nos centraremos es el repositorio LDAP dónde guardaremos las políticas. Hablaremos sobre la configuración y creación de los esquemas LDAP con el fin de explicar cómo se ha creado el esquema de políticas basándonos en el actual RFC 4104 (PCELS) de la IETF. También dedicaremos un apartado a la explicación de los diferentes archivos de configuración del LDAP. Por último, se hablará del software LDAP Browser/Editor empleado para la visualización de las políticas introducidas en el LDAP y como editor de políticas.

En el cuarto capítulo terminaremos de presentar la aplicación desarrollada en el proyecto. Se presentará un programa creado en JAVA, el cual será gestionado mediante polítcas.

En el quinto y último capítulo hablaremos sobre algunos de los posibles desarrollos futuros de este trabajo como es la de aumentar y crear nuevas clases y atributos que permitan considerar más opciones en los esquemas LDAP usados; la replicación de la información del LDAP para que existan varios servidores o la utilización de un servidor DSML (Directory Services Markup Language)[11][12] entre el cliente y el LDAP.

Motivación y razón del proyecto

Las redes de telecomunicaciones de hoy en día se basan en múltiples tecnologías en las que confluyen datos, telefonía, videoconferencia y otros servicios que requieren distintos niveles de calidad. Por ello los proveedores de servicios de Internet (ISP) y las empresas cada vez gestionan más tráfico y cada vez con más demanda de calidad para esos servicios. Esto incluye la capacidad de control de sus infraestructuras, garantizar el acuerdo de nivel de servicio (SLA) contratado por el usuario, etc.

Como podemos observar, estas redes pueden llegar a ser muy complejas y por tanto requerir esquemas de gestión poderosos que optimicen la ejecución de los elementos que conforman la red. Los nuevos modelos de gestión de red ven la conjunción de redes heterogéneas interconectadas como un sistema integral en el que es posible resolver aspectos que van desde la planeación de tareas a largo plazo hasta la provisión de los recursos del día a día.

Ocurre que, en las redes tradicionales, los nuevos servicios se añaden de manera lenta, añadiendo nuevo hardware, y reconfigurando la red de manera manual. Los administradores de red han de reconfigurar los elementos de la red de uno en uno. Esto implica que los sistemas actuales sean muy poco flexibles, y no permiten la gestión dinámica de los recursos de los que se dispone. Esto es un problema cuando hablamos de redes que han de ofrecer una cierta calidad de servicio ya que se ha de garantizar que se cumple el contrato adquirido con el cliente. Cuando el número de clientes y de servicios en el sistema es grande es imposible reconfigurar los elementos de red de manera manual para asegurar que en cada instante la red cumpla los permisos y derechos a la utilización de los recursos de red, sin descuidar por ello la integridad, seguridad y fiabilidad que requiera el sistema de comunicaciones.

Con la gestión de red basada en políticas (PBNM) se presenta una posible solución a este problema. Ofrece un control dinámico y coordinado de los elementos de la red ya que las decisiones se toman de manera automática basándose en reglas, peticiones de usuarios o de servicios. Esta gestión dinámica de los elementos de la red representa un cambio cualitativo, desde el punto de vista empresarial, ya que permite la gestión de los recursos de la red y de los servicios de manera más eficiente.

Este trabajo muestra un ejemplo de funcionamiento de sistema de gestión de red avanzada mediante políticas. Además el lector tendrá la oportunidad de ver con profundidad la funcionalidad del directorio LDAP pudiendo llegar a entender y manejar sus archivos de configuración, esquemas, ejecutables, etc.

Dado que este proyecto será la base de proyectos futuros, todo el software utilizado será Open Source, de esta manera se facilitará la utilización, escalabilidad y desarrollo en un futuro.

Objetivos del proyecto

El principal objetivo de este proyecto es adquirir los conocimientos necesarios sobre la gestión de red basada en políticas. En esto se incluye cómo funciona el Policy Core Information Model (PCIM) así como sus extensiones, cuáles son las clases necesarias y las relaciones entre ellas, para más tarde, ver el Policy Core Extension LDAP Schema (PCELS) que utilizaremos para introducir las políticas en el directorio LDAP.

El primer objetivo y al que se le dedicará más tiempo, consistirá en aprender el funcionamiento del LDAP[13], conociendo perfectamente sus archivos de configuración; para poder crear los suffix o raíces necesarias, una base de datos para cada suffix, controlar los accesos de lectura y escritura, administrar contraseñas, etc. También se deberá comprender perfectamente la sintaxis de los ficheros schema. Estos ficheros son la base sobre los que se apoya el LDAP para dar validez o desechar los diferentes ficheros de entrada, denominados LDIF (LDAP Data Interchange Format)[14]. En los ficheros schema crearemos los esquemas de políticas con las clases, objetos y atributos necesarios. Por último deberemos aprender cómo formar un archivo válido LDIF para que el LDAP lo pueda introducir en su esquema.

El segundo objetivo consistirá en obtener una interfaz gráfica que permita ver los datos introducidos en el LDAP, ya que esto se puede hacer por consola, pero no es una opción muy amigable.

El tercer objetivo consistirá en buscar un lenguaje específico de políticas o alguna tecnología de programación que nos permita introducir políticas en el LDAP de una forma menos costosa que por consola.

Para finalizar se pensará la creación de un programa que permita una demostración plausible de todos los objetivos anteriormente comentados sobre la gestión de red basada en políticas. Para la creación del programa se deberá estudiar el lenguaje de programación JAVA[15], así como la interface JNDI creada especialmente para la lectura, escritura y modificación de directorios; utilizando unicamente la propiedad de lectura para extraer los datos necesarios del LDAP y así poder aplicar las políticas.

CAPÍTULO 1. MARCO DE TRABAJO

Gestión de red se define como el proceso o control de la información compleja que circula por la red para maximizar su eficiencia y productividad. Las áreas fundamentales en las que actúa son la gestión de fallos, gestión de la configuración, gestión de la seguridad, gestión de la ejecución y gestión de la contabilidad.

Para comprender el problema que atañe a este proyecto es necesario explicar el estado actual en lo que se refiere a la gestión de red. Esta sección pretende dar una visión global sobre esta cuestión. Para esto, la siguiente sección presenta de manera resumida dos de los modelos de gestión de red integrada existentes, explicando sus características y problemas más relevantes. Más tarde, se presentará la gestión de red basada en políticas exponiendo los estudios teóricos y prácticos que se han realizado.

1.1 Sistemas gestores de red

La gestión de redes y servicios ha sido un campo en el que tradicionalmente se han impuesto soluciones y mecanismos propietarios de distintos fabricantes, que exigían que la gestión de sus equipos y servicios sólo se pudiera realizar con el gestor del propio fabricante. Ante este escenario, a finales de los años 80 y principios de los 90 surgieron los denominados modelos de gestión de red integrada, que definían un protocolo y unos modelos de información de gestión estándares que rompían la propietariedad en el acceso de gestión remota, permitiendo en teoría la interoperabilidad entre gestores y elementos gestionados de múltiples fabricantes. Debido a diversas razones, fueron dos los modelos propuestos: el de gestión de red de Internet (más conocido como Simple Network Management Protocol (SNMP)[16][17]) y el modelo de gestión de red de OSI (Open Systems Interconection) (conocido también con el nombre de su protocolo: Common Management Information Protocol (CMIP)[18]). Estos modelos son incompatibles entre sí, lo que ha hecho que al cabo de los años cada modelo tenga su ámbito de aplicación distinto: mientras que la gestión OSI se utiliza sobre todo para la gestión de equipos y servicios en redes de telecomunicación, la gestión de red en Internet está más orientada a la gestión de equipos y servicios sobre redes de datos.

1.1.1 SNMP

Un sistema de gestión de red basada en el protocolo SNMP (Simple Network Management Protocol) está compuesto por los siguientes cuatro elementos:

- Nodos administrativos:

Pueden ser hosts, enrutadores, puentes, impresoras u otros dispositivos capaces de comunicar información de estado al mundo exterior. Para ser administrado directamente por el SNMP, un nodo debe ser capaz de ejecutar un proceso de administración SNMP, llamado agente SNMP. Todas las computadoras cumplen este requisito al igual que una cantidad creciente de puentes, enrutadores y dispositivos periféricos diseñados para uso en redes. Cada agente mantiene una base de datos local de variables que describen su estado e historia y que afectan a su operación.

- Estaciones administrativas:

Son computadoras de propósito general que ejecutan un software de administración especial. La estación administradora contiene uno o más procesos que se comunican con los agentes a través de la red, emitiendo comandos y recibiendo respuestas. En este diseño, toda la inteligencia está en las estaciones administradoras, a fin de mantener a los agentes tan sencillos como sea posible y minimizar su impacto sobre los dispositivos en los que se ejecutan.

Muchas estaciones administradoras tienen una interfaz gráfica de usuario para que el administrador de la red pueda inspeccionar el estado de la red y emprenda acciones cuando se requieran. Muchas de las redes reales están compuestas de varios proveedores, con hosts, puentes, enrutadores, impresoras, etc. de uno o más fabricantes y compañías. Con el fin de permitir que una estación administradora hable con estos componentes diversos, la naturaleza de la información mantenida por todos los dispositivos debe especificarse rígidamente. Hacer que la estación administradora pregunte a un enrutador su tasa de pérdida de paquetes no es de utilidad si el enrutador no lleva el registro de su tasa de pérdidas. Por tanto, el SNMP describe (con riguroso detalle) la información exacta de cada tipo de agente que tiene que administrar y el formato con el que éste tiene que proporcionarle los datos.

- Información de administración:

Esta es la parte más grande del modelo SNMP. Es la definición de quién tiene que llevar el registro de qué y cómo se comunica esta información. Muy brevemente cada dispositivo mantiene una o más variables que describen su estado. En la documentación del SNMP, estas variables se llaman objetos. El conjunto de todos los objetos posibles de una red se da en la estructura de datos llamada MIB (Management Information Base).

- Un protocolo de administración:

La estación administradora interactúa con los agentes usando el protocolo SNMP. Este protocolo permite a la estación administradora consultar el estado de los objetos locales de un agente y, cambiarlo de ser necesario. La mayor parte del SNMP consiste en este tipo de comunicación basada en la consulta−respuesta.

A veces ocurren sucesos no planeados. Los nodos administrativos pueden caerse y volver a activarse, las líneas pueden desactivarse y levantarse de nuevo, pueden ocurrir congestiones, etc. Cada suceso significativo se define en un módulo MIB. Cuando un agente nota que ha ocurrido un suceso significativo, de inmediato lo informa a todas las estaciones administradoras de su lista de configuración. Este informe se llama interrupción SNMP. El informe general sencillamente indica que ha ocurrido un suceso; es responsabilidad de la estación administradora emitir consultas para averiguar los detalles. Dado que la comunicación entre los nodos administrativos no es confiable, algunas estaciones administradoras sondean ocasionalmente cada nodo administrado, buscando sucesos inusuales. Este modelo supone que cada nodo administrado es capaz de ejecutar un agente SNMP internamente. Los dispositivos más viejos y los dispositivos no contemplados para usarse en redes podrían no tener esa capacidad. Para manejarlos, el SNMP define un agente apoderado, es decir un agente que supervisa uno o más dispositivos no SNMP y se comunica con la estación administradora a nombre de ellos.

Por último, la seguridad y la validación de identificación desempeñan un papel preponderante en el SNMP. Una estación administradora también tiene la capacidad de aprender mucho sobre cada nodo que está bajo su control, e incluso puede apagarlos todos. Por tanto, es de gran importancia que los agentes estén convencidos de que las consultas supuestamente originadas por la estación administradora realmente vienen de dicha estación.

1.1.2 CMIP

Tras la aparición del SNMP como protocolo de gestión de red, a finales de los 80, gobiernos y grandes corporaciones plantearon el Protocolo Común de Gestión de Información CMIP (Common Management Information Protocol) que se pensó que podría llegar a ser una realidad debido al alto presupuesto con que contaba. En cambio, problemas de implementación han retrasado su expansión de modo que solo está disponible actualmente de forma limitada y para desarrolladores.

CMIP fue diseñado teniendo en cuenta a SNMP solucionando los errores y fallos que tenía éste y volviéndose un gestor de red mayor y más detallado. Su diseño es similar a SNMP por lo que se usan PDUs (Protocol Data Unit) como variables para monitorizar la red.

En CMIP las variables de atributos son unas estructuras de datos complejas con muchos atributos, que incluyen:

- Variables de atributos: representan las características de las variables.
- Variables de comportamiento: qué acciones puede realizar.
- Notificaciones: la variable genera una indicación de evento cuando ocurre un determinado hecho.

CMIP es una arquitectura de gestión de red que provee un modo de que la información de control y de mantenimiento pueda ser intercambiada entre un gestor (manager) y un elemento remoto de red. En efecto, los procesos de aplicación llamados gestores o managers residen en las estaciones de gestión mientras que los procesos de aplicación llamados agentes residen en los elementos de la red. CMIP define una relación de igual a igual entre el gestor y el agente incluyendo lo que se refiere al establecimiento y cierre de conexión, y a la dirección de la información de gestión. Las operaciones CMIS (Common Management Information Services) se pueden originar tanto en gestores como en agentes, permitiendo relaciones simétricas o asimétricas entre los procesos de gestión. Sin embargo, la mayor parte de dispositivos contienen las aplicaciones que sólo permiten hacer de agente.

1.2 Gestión de redes basada en políticas

La gestión de red basada en políticas proporciona un modo específico de asignar recursos de red, calidad de servicio y seguridad, considerando un juego de políticas antes definido. Estos sistemas proporcionan escalabilidad y facilidad para adaptarse a los cambios de las redes en tiempo real.

El IETF Policy Framework Working Group está trabajando en cómo representar, administrar y compartir políticas y en cómo representar las reglas de políticas, para que sean independientes de los fabricantes y permitan una eficiente gestión de red.

Las funciones que debe comprender una gestión basada en políticas son:

- La toma de decisiones comparando el estado actual de la red con el estado deseado (establecido entre el cliente y el proveedor) y estudiar la forma de cómo conseguirlo, esto se desarrolla en el Punto de Decisión de Políticas (PDP).

- La aplicación de políticas que permitan conseguir una política deseada utilizando comandos que, aplicados sobre los distintos dispositivos de la red, permiten cambiar su configuración para conseguir QoS, seguridad, etc. Estas acciones son las que realiza el Punto de Aplicación de Políticas (PEP) según las decisiones del PDP.

En un sentido general, una política es una o más reglas que describen la acción que se produce cuando se da una determinada condición, pudiendo existir reglas y políticas concatenadas. Una política la podemos definir con la sintaxis *si-entonces* aunque esto será comentado en el capitulo 2 cuando hablemos sobre el diseño y edición de políticas.

A modo de ejemplo en la Fig 1.1 se muestra una posible política que daría calidad de servicio mediante la diferenciación de servicios. En la figura se aprecia que tenemos definidos tres usuarios: A, B y C y escribimos una política concatenada que asignara recursos de ancho de banda para cada usuario.

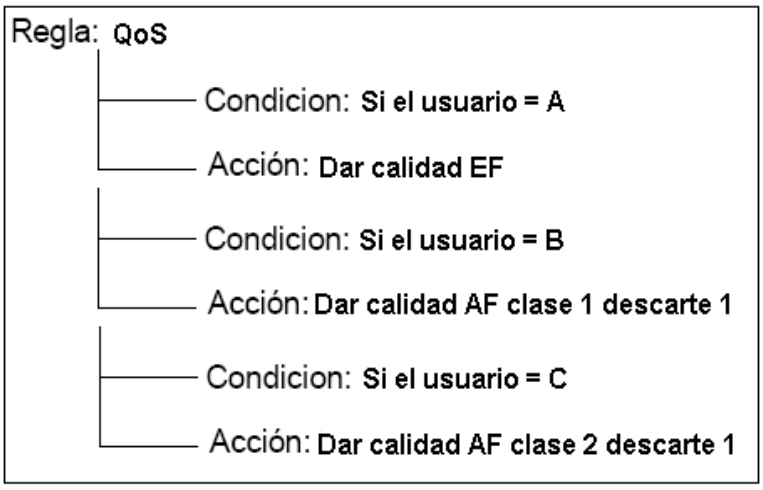

Fig. 1.1 Ejemplo de una política concatenada

1.2.1 Policy Enforcement Point y Policy Decision Point

Antes hemos comentado las funciones que debe comprender una gestión basada en políticas. En estas funciones hemos nombrado los puntos que hacen posible la ejecución de políticas. En esta sección los veremos con más detenimiento.

El Policy Enforcement Point (PEP) es siempre un componente del nodo de red. Es el punto donde se decide dónde y cómo actuará la política. Ocurre un evento en la red, el cual requiere la implicación de una política. El PEP envía la petición de este evento al PDP donde se procesa y se obtiene una respuesta que será enviada al PEP que es el que actuará.

El Policy Decision Point (PDP) es el punto de decisión de las políticas a aplicar. Al recibir una petición que requiera la implicación de una política, el PDP teniendo en cuenta el origen de la petición, obtiene todas las políticas del LDAP que tiene aplicadas el origen de esta petición, las revisa teniendo en cuenta todas la variables y soluciona, si fuese el caso, la contradicción entre una política y otra. Una vez decididas las políticas este le pasa la información al PEP. El PDP puede enviar en cualquier momento una decisión asíncrona cambiando una política.

El PDP puede implementarse en diversos lugares; puede estar junto al servidor de políticas, ser implementado en un elemento independiente que solo hiciese esta función o implementarlo junto al PEP.

Por último, hay que comentar que cuando el PDP está separado del PEP, éste último posiblemente tenga un LocalPDP (LPDP). Éste estaría implementado junto con el PEP y guardaría durante un tiempo las decisiones de las políticas aplicadas a las peticiones de tal manera que, una petición igual y poco

separada en el tiempo podría ser administrada sin necesidad de tener que enviar una petición al PDP.

Para el intercambio de información entre el PEP y el PDP se utiliza un protocolo denominado Common Open Policy Server (COPS)[19]. Este protocolo emplea un modelo de cliente/servidor donde el PEP envía peticiones y el PDP le devuelve decisiones.

COPS utiliza el protocolo TCP (Transport Control Protocol) como protocolo de transporte. Éste, al ser un protocolo de transporte orientado a conexión, reenviará los paquetes perdidos, por tanto, nos proporciona fiabilidad en la entrega de paquetes. COPS proporciona la seguridad a nivel de mensaje para la autenticación, protección y integridad del mensaje, aunque también puede utilizar los protocolos como IPSEC para la seguridad o TLS para la autenticación y para asegurar el canal entre el PEP y el PDP.

1.2.2 Modelos de Información de políticas.

Las políticas deben de ser guardadas en un directorio, la principal razón de esto es que habrá que hacer una gran cantidad de consultas al directorio y pocas modificaciones, y como veremos en el capitulo 3, esta es una de las virtudes del directorio.

Para entender como las políticas son guardadas en el directorio LDAP primero debemos ver como el directorio LDAP guarda la información en general. Después, veremos las diferencias entre el modelo de información LDAP y el Common Information Model (CIM).

El modelo de información LDAP[20] está basado en el modelo de información X.500[21]. La información está guardada en entradas que contienen atributos. Los atributos son de la forma <tipo>=<valor> en que <tipo> es definido como un objeto identificador (OID) y el <valor> tiene una sintaxis definida. Las entradas son organizadas por su distinguished name (DN).

Un esquema en el LDAP es el que define las reglas del distinguished name y qué atributos y entradas deben ser introducidas obligatoriamente o cuáles son opcionales. Para organizar las entradas de la información en el directorio LDAP, el esquema define objectclasses. Éste consiste en un juego de atributos obligados o opcionales. Cada entrada en el directorio LDAP tiene un objectclass asociado con él. Un objectclass puede ser declarado como abstract, structural o auxiliary. La colección de las entradas en el directorio forma el Directoy Information Tree (DIT).

El PCIM Policy Core Information Model es un modelo de información orientado a objetos que representa como estructurar la información de políticas. Por tanto, las políticas pueden ser implementadas en entornos de red heterogéneos. El modelo PCIM y sus extensiones definen dos tipos de clases de objetos:

- Structural classes: representan la información y control de políticas.
- Relationship classes: Indican como las clases estructurales se relacionan entre ellas.

Las clases de políticas y las relaciones de clases definidas en el modelo PCIM son genéricas. Dicho esto, el problema estaba en como almacenar el modelo de información de política en un directorio LDAP. Así que había que mapear el PCIM dentro del LDAP pero, para ello, había que superar un número significativo de diferencias como las siguientes:

- En el LDAP teníamos tres clases (abstract, auxiliary, structural), y el modelo CIM sólo tenia dos (abstract y instantiable).
- En CIM se usa el termino *property* a lo que en LDAP llamamos attribute.
- En las clases CIM no hay distinción entre las *propertys* obligatorias y las opcionales. Si no que todas las *propertys* son opcionales.
- Las clases y propiedades CIM son identificadas por un nombre, no por un OID.
- En el LDAP, los atributos definidos son globales, y pueden aparecer en diferentes clases. En CIM, una *property* es definida únicamente para una clase y no puede aparecer en otras.

Para solucionar la definición de políticas en el PCIM y cómo mapearlas en el LDAP, salieron nuevos modelos:

En Enero del 2003 surgió el Policy Core Information Model extensions (PCIMe)[22] que extendía el PCIM con diferentes cambios.

En Febrero del 2004 se aprobó el Policy Core Lightweight Directory Access Protocol Schema (PCLS)[23] basado en el PCIM que solventaba el problema antes comentado de cómo mapear el PCIM en un directorio LDAP. Al actualizarse el PCIM convirtiéndose en PCIMe el PCLS dejó de representar las nuevas clases y cambios del PCIMe.

Finalmente, en junio del 2005 fue ratificado el Policy Core Extensions Lightweight Directory Access Protocol Schema (PCELS) que estaba basado en el PCIMe y solventaba definitivamente el problema de mapear políticas en el LDAP.

En la Figura 1.2 se presentan varios modelos derivados del PCIM para solventar los problemas específicos de gestión en determinadas áreas, como por ejemplo, el QPIM propuesto para las políticas de QoS, el ICMP para las políticas de seguridad, QPLS para políticas basadas en MPLS, etc. Todas ellas ya están integradas en el PCIMe.

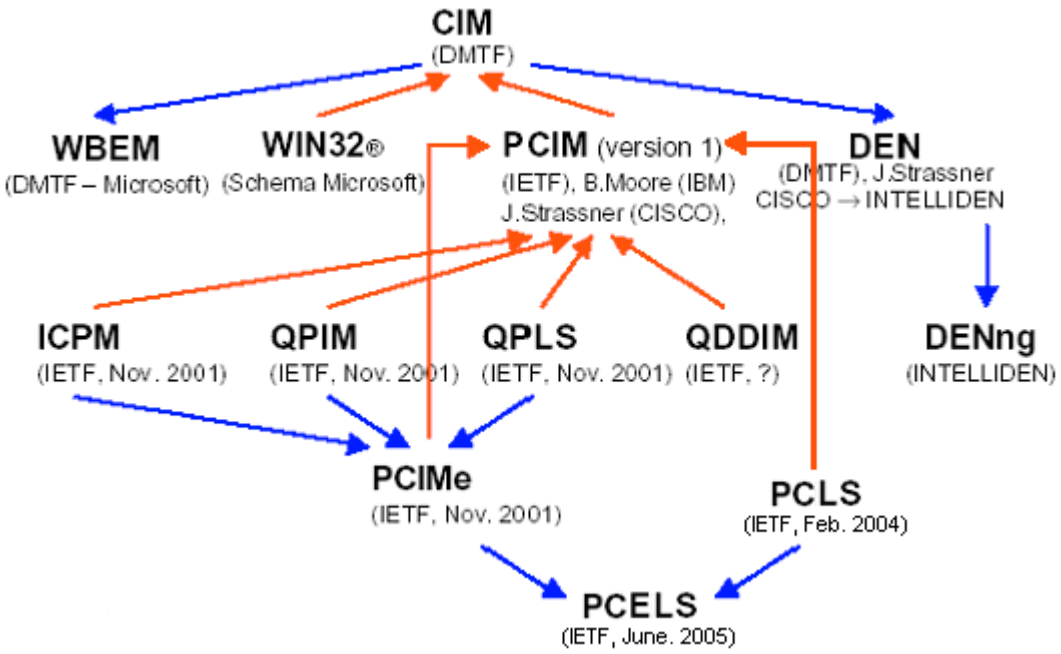

Fig. 1.2 Modelos de políticas

CAPÍTULO 2. DISEÑO Y EDICIÓN DE POLÍTICAS

Una política es un conjunto de reglas definidas en un lenguaje de alto nivel por el administrador de red, es decir, una directiva a la red que puede contener simples mandatos como asignar mayor prioridad a una aplicación que a otra o que un archivo sea encriptado antes de ser transmitido. Por lo tanto, las políticas expresan como debe ser utilizada la red por parte de los usuarios, las aplicaciones, etc.

2.1 Lenguajes específicos para políticas.

Los lenguajes de libre distribución para políticas definen y aplican las políticas en la red. Para ello utilizan un gestor que controla los recursos de la red mediante un conjunto de políticas predefinidas. Los lenguajes de políticas especifican reglas que son independientes de las entidades que implementan las políticas. Estos lenguajes se definen en términos de métricas de ejecución como ancho de banda y permiten la definición de eventos y condiciones de red.

Existen diversos lenguajes de libre distribución para la gestión basada en políticas, algunos de ellos dan mayor énfasis a la seguridad, otros a la gestión de recursos, etc. A continuación se presenta una breve descripción de los lenguajes.

2.1.1 Ponder

Ponder, del Imperial College of Science en Inglaterra, es uno de los lenguajes de políticas de libre distribución más antiguos y avanzados. Es un lenguaje declarativo orientado a objetos para especificar políticas de gestión y de seguridad para sistemas de objetos distribuidos. El lenguaje permite cubrir el amplio rango de los requerimientos actuales que presentan los paradigmas de los sistemas distribuidos.

Ponder utiliza dominios para facilitar la especificación de políticas relacionadas a sistemas grandes, con millones de objetos. Las políticas se especifican para colecciones de objetos almacenados en dominios en lugar de objetos individuales, lo cual proporciona escalabilidad y flexibilidad.

Las políticas que Ponder soporta incluyen: Políticas básicas, compuestas y meta-políticas.

Las políticas básicas pueden ser:

- Políticas de autorización. Definen control de acceso y uso de privilegios sobre los objetos destino.

- Políticas de obligación. Especifican políticas que deben ser ejecutadas sobre determinados objetos cuando ocurren eventos específicos.
- Políticas por delegación. Especifican acciones que un usuario puede delegar a otros.

Las políticas compuestas constan de:

- Grupos. Son políticas y restricciones que tienen relaciones semánticas similares y deben utilizarse conjuntamente
- Roles. Especifican derechos y obligaciones.
- Relaciones. Entre políticas que definen roles.
- Tipo de especialización. Permiten la extensión a nuevas definiciones de políticas utilizando la herencia.

La desventaja de Ponder es que el nivel de complejidad es muy alto debido a que tiene un gran número de constructores. Además, el compilador del Ponder solamente genera las reglas de implementación, es decir, dichas reglas aún necesitan mapearse manualmente al código de aplicación específico.

A continuación, en la Figura 2.1 se presenta la estructura de una política creada en Ponder. Dicha política es un ejemplo de política de autorización. En ella creamos una clase con el nombre de la política "*policyName*". En esta clase definimos a quién va a ser aplicada con la sentencia subject y damos la acción que ocurrirá. Opcionalmente, se puede utilizar la sentencia when para indicar cuándo la acción ha de ser aplicada.

```
inst ( auth+ | auth– ) policyName  "{"
    subject    [<type>]    domain-Scope-Expression ;
    target     [<type>]    domain-Scope-Expression ;
    action                 action-list ;
    [ when                 constraint-Expression ; ]
"}"
```

Fig. 2.1 Estructura de una política en Ponder

2.1.2 Policy Description Language (PDL)

PDL Policy Description Language[24] de la universidad de Munich, es un lenguaje que se utiliza para políticas operacionales. PDL no distingue entre políticas de autorización y políticas de obligación. Las políticas hechas en PDL son, por una parte, políticas de obligación debido a que especifican explícitamente las acciones ejecutadas después de la recepción de un evento. Por otra parte, las políticas se utilizan para configurar la infraestructura de los nodos de la red para garantizar acceso a los distintos recursos de red. Por lo

tanto, este tipo de políticas puede verse como un refinamiento de las políticas de autorización.

Una de las desventajas del lenguaje PDL es que no permite la generación de eventos mediante la aplicación de una política, sin embargo la descripción de una política no está separada de la especificación de los eventos, es decir, cada política tiene que introducir el nombre del evento que utiliza. PDL no permite definir nuevos eventos debido a que los eventos se reciben a partir de los agentes previamente definidos. PDL no contempla un lenguaje de descripción de eventos.

A continuación, en la Figura 2.2 vemos un ejemplo de una política formada en PDL. La especificación de eventos como parte de la descripción de políticas consta de un nombre de los eventos y sus atributos seguidos de la palabra clave DELIVERS. Los atributos se limitan a variables de diferentes tipos como pueden ser strings, long o booleano. Con la ayuda de las variables, es posible reutilizar los valores de los atributos dentro de la especificación de las políticas en la parte de ACTION y CONSTRAINT de la política.

Hay que tener en cuenta que PDL no hace distinciones entre los diferentes tipos de eventos, como la monitorización, los eventos que responden a una fecha u hora determinadas, etc.

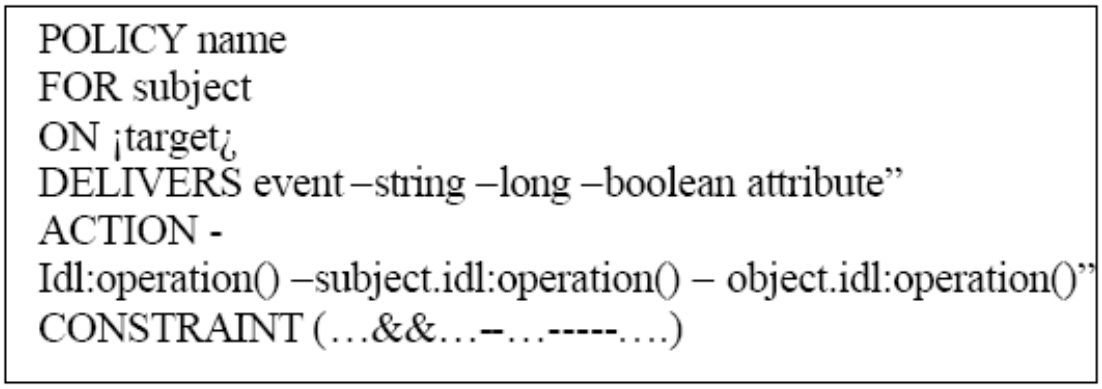

Fig 2.2 Estructura de una política en PDL

2.1.3 Security Policy Language (SPL)

SPL, Security Policy Language[25] del Instituto Mageland en Portugal, es un lenguaje de seguridad diseñado para expresar políticas que ayuden a decidir acerca de la aceptabilidad de eventos. La aceptabilidad de cada uno de los eventos depende de las propiedades de los eventos (por ejemplo el autor, destino, acción, etc.) en el contexto específico en el que suceden los eventos y depende también de las propiedades.

Una política SPL se compone de conjuntos y reglas. Los conjuntos contienen las entidades que utilizan las políticas para decidir acerca de la aceptabilidad de los eventos. Las reglas son funciones de los eventos que pueden tener tres tipos de valores: negar, permitir y no aplicar. Además, las reglas pueden ser simples o compuestas. Las reglas simples se conforman de dos expresiones binarias, una para decidir cuál es el dominio de aplicación de la regla, y la otra para definir la aceptabilidad de los eventos aplicables a la regla. Las reglas pueden estar compuestas por otras reglas y políticas a través del uso del álgebra ternaria. Cada política tiene una regla especial llamada "regla de consulta" (identificada por el signo interrogación) la cual es una composición de otras reglas y define el comportamiento de la política. La delegación se logra cuando se hace una instancia a una política y se utiliza como una regla en la composición de otras reglas.

Las desventajas del SPL son que a pesar de que SPL define un modelo de restricciones, dicho modelo fue diseñado para que lo implementara un monitor de eventos y no es adecuado para utilizarse en todas las plataformas de gestión.

2.2 Lenguajes para la edición de políticas.

En la sección anterior hemos explicado lenguajes creados para la edición de políticas. Ahora veremos cómo podemos crear nuestro propio editor de políticas con JAVA o cómo introducir políticas con el estándar XML[26].

XML eXtensible Markup Language es un conjunto de reglas para definir etiquetas semánticas que nos organizan un documento en diferentes partes. XML es un metalenguaje que define la sintaxis utilizada para definir otros lenguajes de etiquetas estructurados. Las principales características del XML, por lo que lo hacen viable para nuestro requerimiento son:

- Es una arquitectura abierta y extensible. No se necesitan versiones para que puedan funcionar en futuros navegadores. Los identificadores pueden crearse de manera simple y ser adaptados en el acto en internet/intranet por medio de un validador de documentos (parsers).
- Integración de los datos de las fuentes más dispares. Se podrá hacer el intercambio de documentos entre las aplicaciones tanto en el propio PC como en una red local o extensa.
- Datos compuestos de múltiples aplicaciones. La extensibilidad y flexibilidad de este lenguaje nos permitirá agrupar una variedad amplia de aplicaciones, desde páginas web hasta bases de datos.
- Mayor consistencia, homogeneidad y amplitud de los identificadores descriptivos del documento con XML (los RDF Resource Description FrameWork), en comparación a los atributos de la etiqueta <META> del HTML.
- Exportabilidad a otros formatos de publicación (papel, web, cd-rom, etc.). El documento maestro de la edición electrónica podría ser un documento XML que se integraría en el formato deseado de manera directa.

Mirando las características podemos observar que XML es una buena elección para nuestro proyecto, ya que es una tecnología a la que se le augura un buen futuro, no sólo en su aplicación en Internet, sino que también se propone como lenguaje de bajo nivel para el intercambio de información estructurada entre diferentes plataformas como bases de datos.

Como hemos visto en las características se necesitará un parser para introducir las políticas en el LDAP. Un parser es un módulo, biblioteca o programa que se ocupa de transformar un archivo de texto en una representación interna. En el caso de XML, como el formato siempre es el mismo, tan solo hay que crear un parser y no uno cada vez.

A continuación En la Figura 2.3 podemos ver un ejemplo de una edición de políticas en XML. Podemos observar que destaca la sencillez para entenderlo y crearlo. En el ejemplo se ha creado una política aplicable al grupo “epsc”. Este grupo tiene una regla a la que hemos llamado QoS ya que la política a aplicar es sobre ancho de banda o calidad de servicio. Esta regla contiene la condición y acción a aplicar.

```
<POLICY>
      <GROUP NAME="epsc"> </GROUP>
            <RULE NAME= "QoS">
                  <CONDITION >
                        <VARIABLE="grupo">
                        <VALUE="51">
                  </CONDITION>
                  <ACTION>
                        <VARIABLE="AF">
                        <VALUE="1">
                  </ACTION>
            </RULE>
      </GROUP>
</POLICY>
```

Fig. 2.3 Política en XML

Otra forma de introducir políticas es mediante el lenguaje Java. Este lenguaje tiene un API, el Java Naming and Directory Interface (JNDI). Éste es una especificación que permite localizar información en distintos directorios distribuidos como el directorio LDAP.

La edición de políticas mediante JNDI es relativamente sencilla y se asemeja un poco al modo en XML. Se debe crear una plantilla con los atributos y sus valores o, si estos atributos siempre fuesen los mismos se podría crear una plantilla o formulario donde sólo hubiese que introducir los valores correspondientes a cada atributo.

En la figura 4 vemos como hemos creado una política igual que la de XML pero ésta mediante JNDI. Si creásemos todo nos ocuparían varias paginas de código ya que habría que crear subcontextos para introducirlos en contextos. Por tanto, sólo vamos a mostrar cómo se crea la condición y la acción ya que los dos estarán siempre dentro del contexto rule al que suponemos creado.

```
// Creamos atributos para asociarlos al nuevo contexto
Attributes attrs = new BasicAttributes(true);
Attribute objclass = new BasicAttribute("objectclass");
objclass.add("Condition");
objclass.add("Action");
attrs.put(objclass);

// Creamos el contexto donde va a estar
Context result = ctx.createSubcontext("rule=Servicio", attrs)

// Creamos el objeto
Servicio servicio= new Servicio("QoS");

// Creamos los atributos para asociarlos con el object
Attributes attrs = new BasicAttributes(true);
Attribute objclass = new BasicAttribute("objectclass");
objclass.add("Condition");
objclass.add("Action");
attrs.put(objclass);
attrs.put("grupo", "51");
attrs.put("AF", "1");

// Lo introducimos en el directorio
ctx.bind("rule=Servicio", servicio, attrs);
```

Fig. 2.4 Política en JNDI

CAPÍTULO 3. Diseño del almacén de políticas

En este capítulo nos centraremos en todo lo relacionado sobre el Policy Repository dónde van a ser guardadas las políticas, y el Policy Management Tool que será dónde se visualizarán y editarán las políticas. Pero antes de todo esto, hagámonos una idea de todo el sistema a implementar viendo en la figura 3.1 una visión general de todo el sistema.

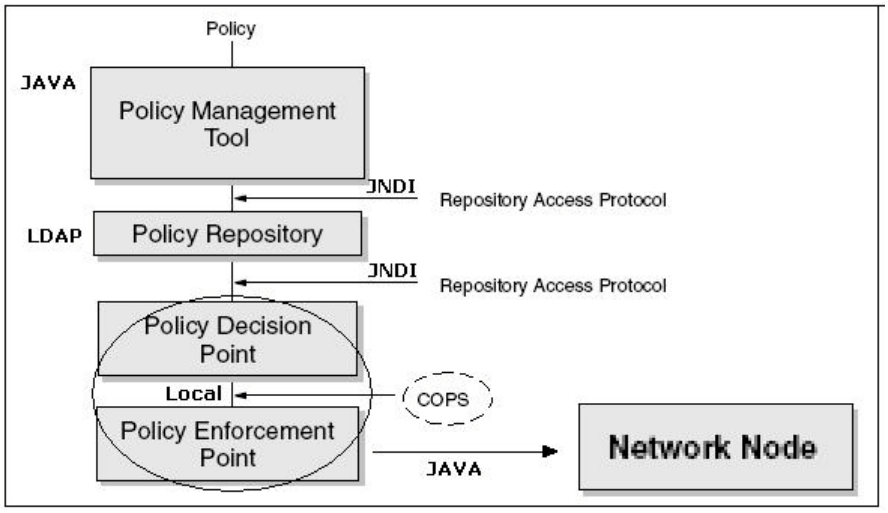

Fig. 3.1 Visión general del sistema

En el sistema tendremos:

- Un Policy Decision Point (PDP) y un Policy Enforcement Point (PEP). Éstos serán creados en JAVA utilizando JNDI y estarán en el mismo dispositivo del sistema. Por tanto, se comunicarán de una forma local y no utilizaremos el protocolo COPS.

- Tendremos el Policy Repository, que es dónde se guardarán las políticas para que el PDP las pueda leer. Como Policy Repository se utilizará el openLDAP.

- El Policy Management Tool que nos permitirá introducir, visualizar, modificar o borrar las políticas en el directorio LDAP. Como Policy Management Tool hemos utilizado un programa de libre distribución; el LDAP browser/editor al que se le han aplicado algunos cambios, necesarios para ajustarse a las necesidades del proyecto. Todos los cambios relacionados con el Policy Management Tool serán tratados en la sección 3.5.

- Por último, tendremos el Network Node que será el programa o elemento de la red de trabajo que vamos a gestionar.

Comenzaremos hablando del repositorio de políticas. Para tal fin, será utilizado el openLDAP. El proyecto openLDAP nació como la continuación de la versión 3.3 del servidor LDAP de la Universidad de Michigan cuando estos dejaron de desarrollarlo. OpenLDAP es un servidor LDAP que se distribuye bajo licencia GNU y que permite que el software se pueda usar de forma gratuita. Además, disponemos del código fuente para poder realizar nuestras propias investigaciones.

El servidor openLDAP requiere un backend o motor para almacenar y hacer búsquedas de la información. En el sistema se ha instalado BerkeleyDB que es un gestor de bases de datos disponible de forma gratuita y bajo licencia también Open Source.

El openLDAP está disponible para múltiples sistemas operativos: Apple Mac OS X, Linux (Debian, RedHat, Suse, Fedora, Mandrake...), FreeBSD, IBM AIX, Microsoft Windows 9x/2000/NT/XP[27], NetBSD y Solaris.

LDAP es un protocolo de tipo cliente-servidor para acceder a un servicio de directorio. Se usó inicialmente como un Front-end o interfaz final para X.500, pero también puede usarse con servidores de directorio únicos y con otros tipos de servidores de directorio.

Un directorio es como una base de datos, pero con importantes diferencias como:

- Propósito de acceso: En un directorio el propósito para su acceso es la lectura y búsqueda, mientras que en una base de datos el propósito general es la actualización o escritura de datos.

- Soporte de acceso: En un directorio se soportan altos volúmenes de solicitudes de lectura. Una base de datos está preparada para altas actualizaciones de datos.

- Capacidad de operación: Un directorio no es apropiado para almacenar cambios rápidos en la información que contienen. Una base de datos sí que esta preparada para tal propósito.

- Transacciones: Un directorio no soporta cierto tipo de transacciones que son mas bien sencillas. Mientras que una base de datos soporta transacciones complejas.

Los directorios permiten replicar información de forma amplia, con el fin de aumentar la disponibilidad y fiabilidad, y a la vez, reducir el tiempo de respuesta.

El openLDAP es un directorio, ya que reúne las características de éste, teniendo como principal virtud la realización de múltiples lecturas de forma muy eficiente.

El servidor openLDAP ofrece una serie de posibilidades útiles para este proyecto y futuros desarrollos como son:

- Rapidez en la lectura de registros.

- Implementación del protocolo LDAPv3: Este protocolo está definido en el RFC 3377 y permite el uso de Ipv4 e Ipv6.

- Autenticación y nivel de seguridad: Permite servicios de autenticación mediante SASL. El acceso al directorio ha de ser restringido ya que contiene información de usuarios, información de políticas y parámetros de los SLS. Por esta razón sólo el administrador del directorio tiene acceso a la información.

- Control de acceso: openLDAP permite definir una serie de filtros para el control de acceso a diferentes DITs, entradas o atributos de estas entradas.

- Distribución de información en varios servidores: openLDAP implementa el método de referrals para la distribución de los DITs en diferentes servidores.

- Escalabilidad: Muchas aplicaciones tienen interfaces de conexión a LDAP y se pueden integrar fácilmente.

- Multiplicidad: Permite la creación de múltiples directorios independientes.

- Facilidad: los servidores openLDAP son fáciles de instalar, mantener y optimizar.

LDAP utiliza estructuras de árbol para jerarquizar la información contenida. Esta estructura es muy eficaz y está arraigada en muchos medios como el DNS o la estructura de carpetas de un sistema operativo.

En la figura 3.2 podemos ver como está estructurado el LDAP como directorio de nombres, que es la función para la que está pensado. En él tenemos una raíz que tendrá como atributo dc (domain component) y como valores epsc.es. De aquí se partirá en dos atributos ou (organization unit) con los valores Alumnos por un lado y Profesores por el otro. Cada uno tendrá sus atributos cn (common name) y así continuamente, pudiendo añadir, modificar o borrar tantas ramas como se quiera.

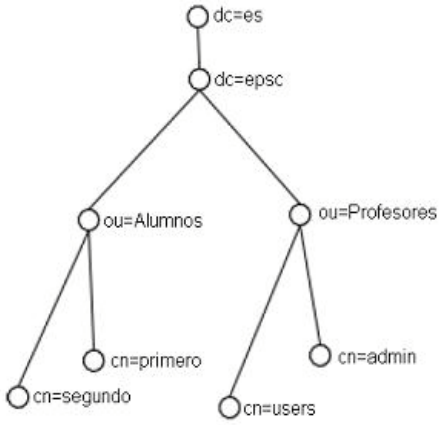

Fig. 3.2 Jerarquía LDAP

El modelo de información de LDAP está basado en entradas. Una entrada es una colección de atributos que tienen un único y global Distinguished Name (dn). El dn se utiliza para referirse a una entrada sin ambigüedades. Cada atributo de una entrada posee un tipo y uno o más valores. Como hemos visto en la Figura 3.2 podemos indicar el dn que tendrá el cn= admin de esta manera:

dn: cn=admin, ou=Profesores, dc=epsc, dc=es

Como vemos, los datos del directorio se presentan mediante pares de atributo (attributetype) y su valor. Por ejemplo, el atributo commonName o cn se usa para almacenar el nombre de una persona.

Los atributos requeridos son aquellos que deben estar presentes en las entradas que utilicen los objectclass, mientras otros pueden o no estar presentes. Por ejemplo, en la clase person, se requieren los atributos cn y sn. Los atributos description, telephoneNumber, seealso y userpassword se permiten pero no son obligatorios.

3.1 Schema LDAP

Como hemos estado viendo hasta ahora, el LDAP fue pensado principalmente como directorio de nombres, pero nosotros lo necesitamos como directorio de políticas. Para conseguir que el LDAP acepte las políticas, hay que editar el fichero schema con la estructura de políticas que necesitemos. El fichero schema es dónde se definirán las clases (objectclass) y atributos (attributetype) que aceptará el LDAP.

En nuestro LDAP vamos a continuar utilizando el esquema de personas llamado core.schema. Este esquema lo utilizaremos en un primer momento para autentificar al usuario que se quiera conectar. Luego, crearemos el

policy.schema, el cual será utilizado una vez el usuario sea autentificado, para extraer las políticas que le incumben. Mediante el gestor de base de datos BerkeleyDB que hemos instalado junto al openLDAP, y que nos sirve como motor para almacenar, crearemos dos bases de datos o espacios de almacenamiento diferentes, uno para personas y otro para políticas. Esto se especificará en el fichero de configuración slapd.conf del cual hablaremos en la sección 3.2.

Una definición de tipo de atributo especifica la sintaxis de un atributo y cómo se ordenan y comparan los atributos de ese tipo. Los tipos de atributos en el directorio forman un árbol de clases. Hay atributos obligatorios y opcionales. En la tabla 3.1 están todos los identificadores que puede tener un attributetype correspondientes a la definición de “AttrributeTypeDescription” en el RFC 2252.

Tabla 3.1 Descripción de los tipos de *attributetype*

Identificador del Atributo	**Descripción del Valor del Atributo**
NUMEROICOID (obligatorio)	Identificador de Objeto Único (OID)
NAME	Nombre del Atributo
DESC	Descripción del Atributo
OBSOLETE	“true” si es obsoleto “false” o ausente si no lo es
SUP	Nombre del tipo de atributo superior del que se deriva el tipo de atributo
EQUALITY	Nombre o OID de la regla de correspondencia si la igualdad de correspondencia está permitida; ausente si no lo está
ORDERING	Nombre o OID de la regla de correspondencia si está permitida la ordenación; ausente si no lo está
SUBSTRING	Nombre o OID de la regla de correspondencia si está permitida la correspondencia de sub-string; ausente si no lo está
SINTAX OID	Numérico de la sintaxis de los valores de este tipo
SINGLE-VALUE	“true” si el atributo no es multi-valor. “false” o ausente si lo es.
COLLECTIVE	“true” si el atributo es colectivo. “false” o ausente si no lo es.
NO-USER-MODIFICATION	“true” si el atributo no es modificable por el usuario. “false” o ausente si lo es
USAGE	Descripción del uso del atributo.

La directiva objectclass es usada para definir nuevas clases. Estas clases se utilizan para contener los atributos. En general, cada entrada contendrá una clase ABSTRACT (top o alias), al menos una clase STRUCTURAL y alguna o ninguna clase AUXILIARY. Cualquiera de estas tres descripciones de objectclass se ponen debajo del SUP y no necesitan ningún identificador delante. Por lo demás, los objectclass contienen identificadores tales como NAME, DESC, OBSOLETE,SUP descritos ya en los attributetype de la Tabla 3.1

En la tabla 3.2 vemos los identificadores que contienen los objectclass y no contienen los attributetype. Estos objectclass corresponden a la definición de "ObjectClassDescription" en el RFC 2252.

Tabal 3.2 Descripción de los tipos de *objectclass*

Identificador del objectclass	Descripción del valor del objectclass
MUST	Contendrá el nombre de los atributos obligatorios.
MAY	Contendrá el nombre de los atributos opcionales.

Si se desea consultar algo más sobre los attributetype o objectclass o saber cómo crear los tuyos propios y que OID habría que poner,etc. se recomienda consultar la página del openLDAP o un tutorial muy completo sobre éste[28].

3.1.1 Creación del esquema de usuarios

Como ya hemos dicho anteriormente también utilizaremos el directorio como repositorio de personas para la autentificación. El esquema relacionado al directorio de personas se llama core.schema el cual no lo explicaremos con detenimiento ya que esto se puede encontrar en la muchas paginas web como la del openLDAP.

Sobre el core.schema tan sólo diremos los atributos y objectclass que utilizaremos para la autentificación. Éstos son los siguientes:

- El objectclass organization con su attributetype obligatorio organizationName, llamado "o" en modo abreviado.

- El objectclass uidObject con su attributetype obligatorio userid, llamado "uid" en modo abreviado.

- Los objectclass person con su attributetype obligatorio common name (cn) y su atributo opcional userPassword.

Para finalizar la explicación, veremos en la figura 3.3 cómo va a estar compuesta la jerarquía en árbol del directorio de usuarios. En ella hemos creado a partir de una raíz “people” un grupo “profesores”. En este grupo hemos introducido el nombre de una persona, con un identificador y un password. Con este ejemplo ya se puede ver la idea de un directorio de personas dónde, a partir de la raíz se pueden crear múltiples grupos, dentro de ellos poner más personas o introducir grupos dentro otros grupos que contengan personas, etc.

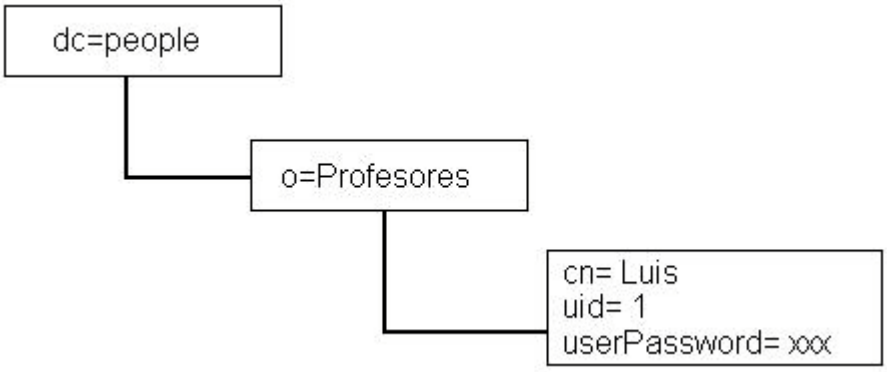

Fig. 3.3 Ejemplo de directorio de usuarios del proyecto

3.1.2 Creación del esquema de políticas

En esta sección veremos como crear el fichero policy.schema, el cual tendrá los objectclass y attributetype necesarios para la creación de políticas. Este esquema está basado en el RFC 4104 (PCELS).

El fichero schema se puede crear fácilmente con cualquier editor de textos, tan solo hay que ponerle la extensión schema para que el LDAP lo acepte. El esquema de políticas es muy extenso y conlleva muchas dependencias entre objectclass. Para este proyecto se ha estudiado y se ha acortado a fin de cubrir las necesidades básicas para su presentación. El esquema de políticas final que crearemos está representado en la Figura 3.4.

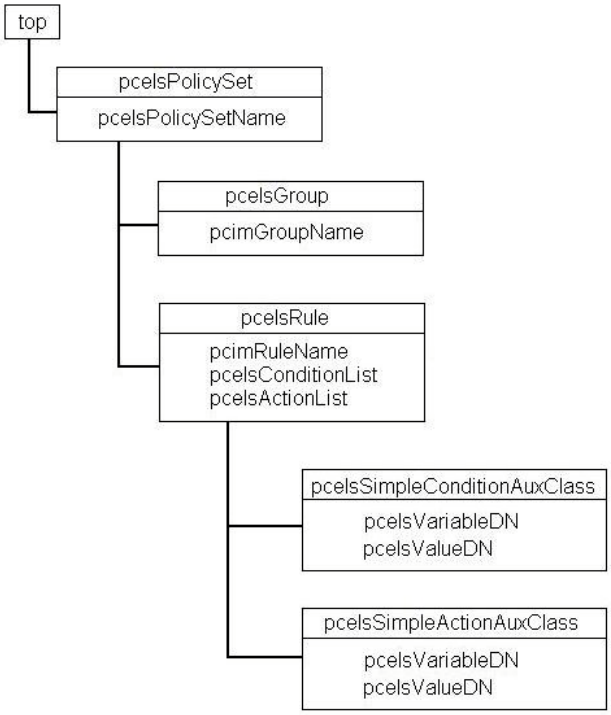

Fig. 3.4 Esquema de políticas

A continuación veremos la definición de cada objectclass y attributetype mencionados anteriormente para luego pasar a explicar un ejemplo que aclare la utilización de estos.

- Top: Este elemento no es ni un objectclass ni un attributetype. Se puede definir como una palabra clave que acepta el LDAP, la cual no hay que definir en el fichero schema. Cuando definimos un objectclass éste tiene el identificador SUP, a este identificador se le coloca un atributo que es el nombre de otro objectclass del cual cuelga, formando así la mencionada estructura en árbol. Pero cuando el objectclass es el elemento superior del árbol hay que poner que éste cuelga del top, que es la palabra clave para indicar que éste no cuelga de nadie, sino que es el objectclass superior. Podríamos decir que es la raíz.

- pcelsPolicySet: Esta clase representa el juego de políticas que tiene una estrategia común. Proporciona la suficiente información para permitir el orden entre los distintos juegos de políticas. Hay que incluir el objectclass dcObject que lo podemos encontrar en el core.shema y que es necesario para la el domain component (dc).

  - pcelsPolicySetName: Este atributo opcional es usado para darle nombre a los juegos de políticas de las entradas del pcelsPolicySet. Está definido como string y tan solo puede tener un valor.

- pcelsGroup: Esta clase es la base para representar una política de grupo. Proporciona la suficiente información para crear grupos de políticas.

    - pcimGroupName: Este atributo opcional es usado para darle nombre al grupo de políticas de las entradas del pcelsGroup. Está definido como string y tan solo puede tener un valor.

- pcelsRule: Esta clase es la base para representar la regla de políticas que tienen una estrategia común tanto en condición como en acción. Actuará bajo un pcelsGroup definido anteriormente

    - pcimRuleName: Este atributo opcional está definido en el PCLS y es usado para darle nombre a las entradas del pcelsRule. Está definido como string y tan solo puede tener un valor.

    - pcelsConditionList: Este atributo opcional es usado para darle nombre a las entradas de condición del pcelsSimpleConditionAuxClass. Definido como string y de único valor.

    - pcelsActionList: Este atributo opcional es usado para darle nombre a las entradas de acción del pcelsSimpleActionAuxClass. Definido como string y de único valor.

- pcelsSimpleConditionAuxClass: Esta clase es la base para representar la condición de la política. Proporciona la suficiente información para crear una condición. Cuelga del objectclass pcelsRule.

- pcelsSimpleActionAuxClass: Esta clase es la base para representar la acción de la política. Proporciona la suficiente información para crear una acción. Cuelga del objectclass pcelsRule.

    - pcelsVariableDN: Este atributo opcional es usado tanto para el objectclass pcelsSimpleConditionAuxClass como para el pcelsSimpleActionAuxClass. Indica la variable de la condición y de la acción.

    - pcelsValueDN: Este atributo opcional es usado tanto para el objectclass pcelsSimpleConditionAuxClass como para el pcelsSimpleActionAuxClass. Indica el valor de la condición y de la acción.

Para editar los objectclass y attributetype que utilizaremos tan solo hay que copiarlos del RFC 4104 en un editor de textos como word en windows o el editor vi en linux.

Un apunte, que es muy importante sobre la edición del fichero schema, es que al inicializar el LDAP este hace una revisión de los esquemas que va a utilizar. Si el esquema no lo considera correcto, el LDAP indicaría un mensaje de error y abortaría su inicialización. Un error en la edición del policy.schema fue poner

los attributetype debajo de los objectclass ignorando que el LDAP al revisar el schema tenia como regla que los attributetype tenían que editarse encima de los objectclass que los contenían para considerar el schema correcto.

Después de haber hablado de las características de los objectclass y attributetype y cuáles vamos introducir, veamos un ejemplo en la Figura 3.5 donde tenemos el objectclass pcelsGroup y su attributetype pcimGroupName.

```
objectclass ( 1.3.6.1.1.9.1.3
   NAME 'pcelsGroup'
   DESC 'Base class for representing a policy group'
   SUP pcelsPolicySet
   ABSTRACT
   MAY ( pcimGroupName )
)

attributetype ( 1.3.6.1.1.6.2.4 NAME 'pcimGroupName'
        DESC 'The user-friendly name of this policy group.'
        EQUALITY caseIgnoreMatch
        ORDERING caseIgnoreOrderingMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
        SINGLE-VALUE
)
```

Fig. 3.5 Ejemplo de un objectclass y un attributetype

Una vez definidos los objectclass y attributetype vamos a explicar un ejemplo de cómo crear una política, cómo estaría definida en el LDAP y cómo se aplicaría.

Un administrador desea crear una política de conexión a un grupo de una empresa. El grupo al que se le quiere aplicar la política es el de finanzas. Pero finanzas está dividido en 5 secciones y solo se quiere aplicar a la sección 2. La política pretende que este grupo no pueda hacer peticiones http (bloquear el puerto destino 80).

Lo primero que hará el administrador será darle un nombre a esa política, por ejemplo le pondrá el nombre “navegar”. Por tanto utilizará el objectclass pcelsPolicySet y su attributetype pcelsPolicySetName el cual contendrá “navegar”.

Del pcelsPolicySet le haremos colgar el objectclass pcelsGroup que aportará el attributetype pcimGroupName donde pondremos el nombre del grupo al que se le aplica esta política. El grupo será finanzas, así que de momento estamos creando una política que se aplicará a todos el grupo de finanzas.

Luego colgaremos del pcelsGroup el pcelsRule. Este objectclass es el que contendrá los objectclass de la condición y de la acción a realizar. El pcelsRule tiene tres attributetype: el pcelsRuleName que servirá para darle nombre a la regla, el pcelsConditionList y pcelsActionList que dará nombre a la condición y a la acción. Tanto a la regla, como a la condición y acción le pondremos el mismo nombre: http. El ponerles el mismo nombre es una forma cómoda para distinguir cada grupo de regla → condición/acción. Ahora mismo tenemos creada una política con una regla (llamada http) a un grupo (finanzas), a parte de tener ya definidos el nombre que le pondremos a la condición y a la acción.

Ahora introduciremos los attributetype de la condición y de la acción, como son el pcelsVariableDN y el pcelsValueDN. El primero indicará la variable y el segundo aportará el valor de esta variable.

En la condición editaremos como pcelsVariableDN “finanzas” y como pcelsValueDN “2” (refiriéndose a la sección 2).

En la acción indicaremos como pcelsVariableDN “bloquear” y como pcelsValueDN “80” (refiriéndose al puerto destino http).

En la figura 3.6 vemos el esquema de la política, igual que vimos en la figura 3.4, pero ahora tiene definidos los valores.

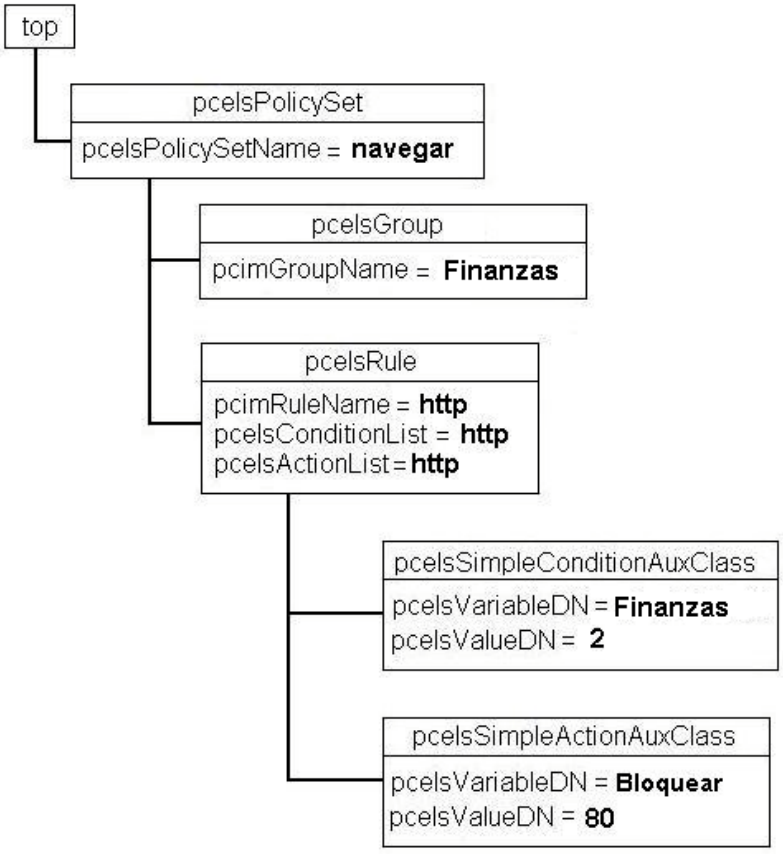

Fig. 3.6 Implementación de una política.

Como ya tenemos la política implementada veamos los pasos que seguiría el PDP cuando le llegase un evento desde PEP. Este evento es una petición de una página web de un usuario de finanzas de la sección 2.

- En el paso 1 tenemos que el grupo del usuario es finanzas, entonces busca si hay pcelsGroupName con el nombre finanzas. En este caso sí que lo hay, así que seguimos con el paso 2.

- En el paso dos cogemos todas las reglas que tiene aplicadas el grupo de finanzas. En este caso sólo tiene una regla aplicada: http (pcelsRuleName).

- En el paso tres, cogemos la condición del LDAP y los datos del usuario ("grupo" y "uid") y los comparamos: *Si* finanzas(pcelsVariableDN)= "grupo" y 2(pcelsValueDN)= "uid", seguimos con el paso 4.

- En el paso cuatro tenemos que la condición se ha cumplido por tanto se produce la acción: *entonces* bloquear(pcelsVariableDN) 80(pcelsValueDN).

- El quinto y último paso es pasar esta acción al PEP para que éste indique al usuario que su petición ha sido denegada.

Por ultimo imaginamos que ahora se quiere crear una política al grupo de finanzas sección 3 que no les permita hacer peticiones al dns (cerrar el puerto destino 53).

En este caso no habría que comenzar desde el principio, ya que el nombre del grupo lo tenemos definido en la política anterior. Así que tan solo se tendría que crear una rama desde el grupo finanzas que contuviese una regla con su condición y acción. En la figura 3.7 vemos esta nueva rama que colgaría del pcelsGroup.

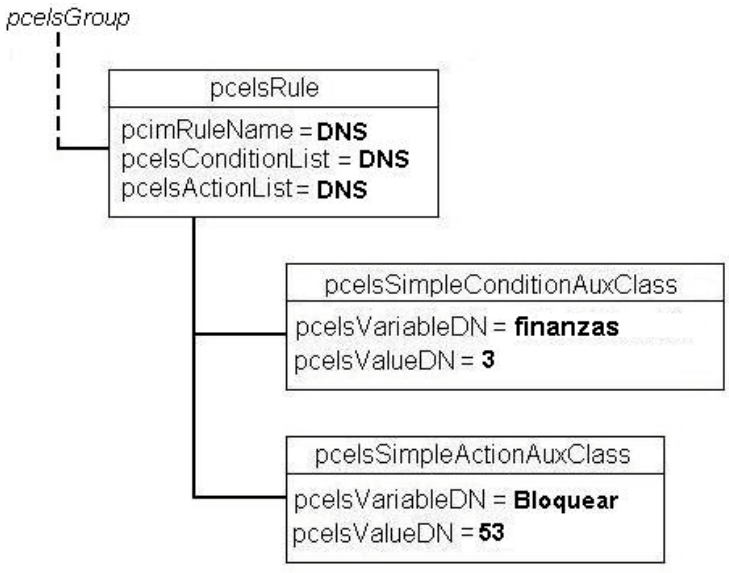

Fig. 3.7 Ampliación de una política

3.2 Configuración del slapd.conf

Una vez terminado el esquema de políticas, se deberá modificar el fichero slapd.conf,. Éste es el fichero de configuración del LDAP. A continuación veremos este fichero y comentaremos cada línea.

Lo primero que hay que definir en el fichero son los esquemas que va a utilizar el openLDAP con la sentencia include. En el proyecto se han introducido el core.shema y el policy.schema. Las líneas correspondientes a la introducción del esquema son:

```
include          C:/openldap/etc/schema/core.schema
include          C:/openldap/etc/schema/policy.schema
```

Podríamos haber editado los dos esquemas en un mismo fichero sin ningún problema. Tan solo habría que tener en cuenta lo comentado anteriormente y que es muy importante, escribir los attributetype encima de los objectclass que los contienen.

Después de esto, hay que definir el control de accesos que nos permite configurar quién puede acceder a la información y quién puede borrarla o modificarla. El usuario que entra como root tiene todos los privilegios. El control de accesos sólo es para usuarios. La sintaxis del control de accesos es de la siguiente manera:

```
access      to<what>
    [       by<who><access> [<control>] ]+
```

En el próximo ejemplo vemos un control de accesos. Éste permite a los usuarios autentificados realizar cualquier tipo de cambio en sus entradas (by self write) y leer otras entradas de otros usuarios (by users read). Los usuarios no autentificados no tienen acceso al directorio (by * none). Notar que cuando se pone la sigla “*” , ésta significa todo.

```
access      to *
    by self         write
    by users        read
    by *            none
```

Por último, definiremos las bases de datos. En nuestro caso hemos creado dos bases de datos, uno para el directorio de personas (Figura 3.8) y el otro para el directorio de políticas (Figura 3.9).

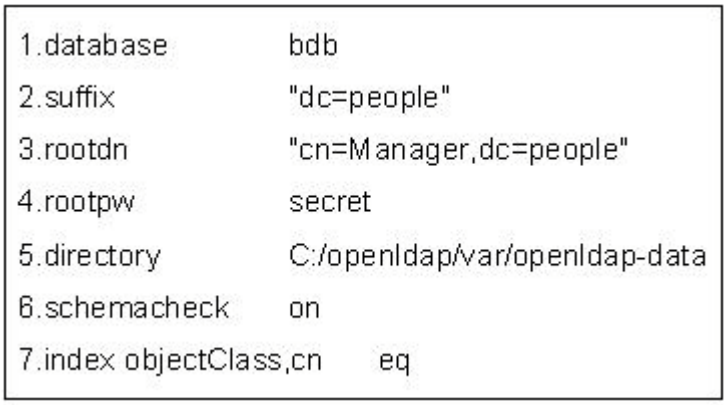

Fig. 3.8 Base de datos de usuarios

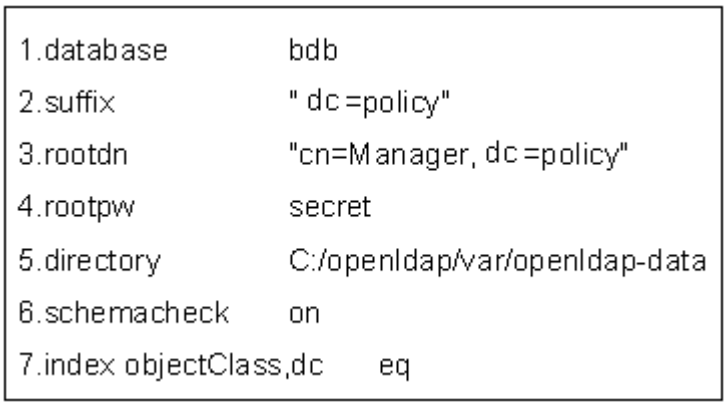

Fig. 3.9 Bases de datos de políticas

En la primera línea indicamos el inicio de la nueva base de datos que almacena información. Se pueden crear tantas bases de datos como se quiera, con la consiguiente carga de memoria que conlleva eso. Para el proyecto se han creado dos base de datos, una para usuarios y otra para políticas. Esta base de datos ha de utilizar el gestor de base de datos BerkeleyDB.

En la línea 2 tenemos la directiva suffix, la cual indica que todas las peticiones que tengan como sufijo o raíz ‘dc=people’ o ‘dc=policy’ sean enviadas a la base de datos correspondiente.

En la línea 3 indicamos cuál es la entrada de administración del directorio. Esta entrada es necesaria ya que sólo le está permitido al administrador añadir entradas al directorio, por lo que es necesario autentificarse cuando se desea realizar esta operación.

En la línea 4 tenemos la contraseña para acceder como administrador. Aquí se ha puesto en claro, pero con el comando “slappasswd” que viene con el ldap podemos poner esta contraseña codificada en el fichero de configuración. La directiva por consola es:

$slappasswd –s secret (o la contraseña que queramos).

Esto nos dará como resultado la contraseña secret codificada:

<SSHA> aWLGmT2yjrtEqNI+bv5M0er8jw8cmt2q

En la línea 5 indicamos dónde los ficheros de la base de datos se han de almacenar. Este directorio ha de tener los permisos de lectura y escritura habilitados a todos los usuarios de la máquina.

En la línea 6 tenemos la directiva 'schemacheck on' que indica qué ha de revisar el esquema cuando se inicie el directorio para detectar posibles fallos en su estructura.

Por último, en la línea 7 tenemos la directiva 'index' que indica para qué atributos se han de almacenar índices. En nuestro caso para 'objectClass', 'cn' y 'dc' ya que son los atributos que identifican a las entradas de usuarios y políticas.

3.3 Comandos del openLDAP

El openldap tiene varios comandos para su utilización. Como ya dijimos en capítulos anteriores estos comandos son poco amigables de utilizar, por ello, en esta sección nombraremos algunos y más adelante veremos la solución para que sean fáciles de manejar.

- ldapsearch: Se usa para realizar búsquedas dentro del LDAP. Su sintaxis es la siguiente: ldapsearch [opciones] filter [parametros].

- ldapadd: Esta herramienta permite añadir entradas. La sintaxis es: ldapadd [opciones] –f "archivo"

- ldapdelete: Esta herramienta permite borrar entradas. La sintaxis es: ldapdelete [opciones] "DN_de_la_entrada".

- ldapmodify: Esta herramienta permite cambiar, añadir o borrar atributos. La sintaxis es: ldapmodify [opciones] -f "archivo"

Se entiende por filtro la condición que se debe cumplir para la búsqueda de entradas. Se puede usar el comodín * para las búsquedas pero no es aconsejable si se espera un número muy alto de atributos.

A continuación en la Tabla 3.3 vemos los parámetros obligatorios y en la Tabla 3.4 los parámetros opcionales.

Tabla 3.3 Parámetros obligatorios

Parámetros obligatorios	Descripción
-b basedn	Especifica el DN base para las búsquedas
-s scope	Alcance de la búsqueda: base, one o sub

Tabla 3.4 Parámetros opcionales

Parámetros opcionales	Descripción
-A	Sólo muestra los nombres de los atributos (no los valores)
-a deref	Referencias a los alias: never, always, search, or find
-B	Permite imprimir valores no-ASCII
-D binddn	Cuando se autentifica con un directorio, permite especificar la entrada binddn. Se usa con la opción –w password.
-d debug level	Nivel de debug
-E “character_set”	Especifica la página de codificación de caracteres
-f file	Ejecuta la sentencia de búsquedas archivadas en el archivo file
-h ldaphost	Conecta al servidor LDAP en la dirección ldaphost. El valor por defecto es localhost
-L	Muestra las entradas en formato LDIF
-l timelimit	Timeout en segundos antes de abandonar una búsqueda
-p ldapport	Conecta al servidor en el puerto TCP especificado en ldapport. Por defecto conecta en el puerto 389.
-S attr	Ordena los resultados por el atributo attr
-v	Modo extendido
-w passwd	Especifica la contraseña para hacer el bind (para autentificación simple)
-z sizelimit	Especifica el número máximo de entradas que pueden ser mostradas.

A continuación, tenemos un ejemplo para añadir una entrada dónde se puede ver lo laborioso que sería cada vez que se quisiese añadir, modificar o borrar entradas.

ldapadd –p 389 –D “cn=Manager, dc=policy” –w secret –f politica.ldif

3.4 Ficheros LDIF

Para importar y exportar información de directorio entre servidores de directorios basados en LDAP, o para describir una serie de cambios que han

de aplicarse al directorio, se usa en general el fichero de formato conocido como LDIF (formato de intercambio de LDAP).

Un fichero LDIF almacena información en jerarquías de entradas orientadas a objeto. Todos los servidores LDAP incluyen una utilidad para convertir ficheros LDIF a formatos orientados a objetos ya que un fichero LDIF es un fichero ASCII.

Igual que comentábamos con la introducción de LDIF mediante comandos, la creación de un LDIF si es muy largo puede llegar a ser muy costoso. Esto también lo solucionaremos más adelante. En la Figura 3.10 un ejemplo de LDIF dónde introduciremos políticas en forma de árbol de directorio.

```
dn: dc=politica
dc: politica
objectClass: dcObject
objectClass: pcelsPolicySet

dn: pcelsPolicySetName=Prioridad, dc=politica
objectClass: pcelsPolicySet
pcelsPolicySetName: Prioridad

dn: pcelsGroupName=uni, pcelsPolicySetName=Prioridad, dc=politica
objectClass: pcelsGroup
pcelsGroupName: uni

dn:pcimRuleName=conexion,pcelsGroupName=uni,pcelsPolicySetName=Prioridad, dc=politica
objectClass: pcelsRule
pcimRuleName: conexion

dn: pcelsConditionList=grupo,pcimRuleName=conexion,pcelsGroupName=uni,pcelsPolicySetName= Prioridad, dc=politica
objectClass: pcelsConditionAuxClass
Variable: uni
Value: 1
pcelsConditionList: grupo

dn: pcelsActionList=grupo, pcimRuleName=conexion, pcelsGroupName=uni,pcelsPolicySetName= Prioridad,dc=politica
objectClass: pcelsActionAuxClass
Variable: prioridad
Value: 2
pcelsActionList: grupo
```

Fig. 3.10 Fichero LDIF

3.5 LDAP Browser/Editor

Como hemos comentado a lo largo del proyecto, la opción por consola para visualizar, insertar, borrar o modificar datos no es muy amigable. Por esta razón utilizaremos el software LDAP Browser/Editor. Éste permite a los

usuarios ver los elementos almacenados en un directorio LDAP de una manera jerárquica y añadir, borrar o modificar los elementos si el usuario se ha conectado como administrador

En el LDAP Browser los objetos LDAP son presentados en forma de árbol y todos los atributos de las entradas en forma de tablas. El estado del browser se ve en la barra de estado. En esta barra se muestran mensajes al seleccionar una entrada o al añadirla, modificarla o borrarla. Los mensajes de estado son de color negro, las advertencias son en amarillo y los mensajes de error en rojo.

En la figura 3.10 vimos un LDIF con políticas. Ahora en la figura 3.11 veremos como el LDAP Browser nos va a mostrar el mismo LDIF introducido ya en el LDAP y de una forma visual.

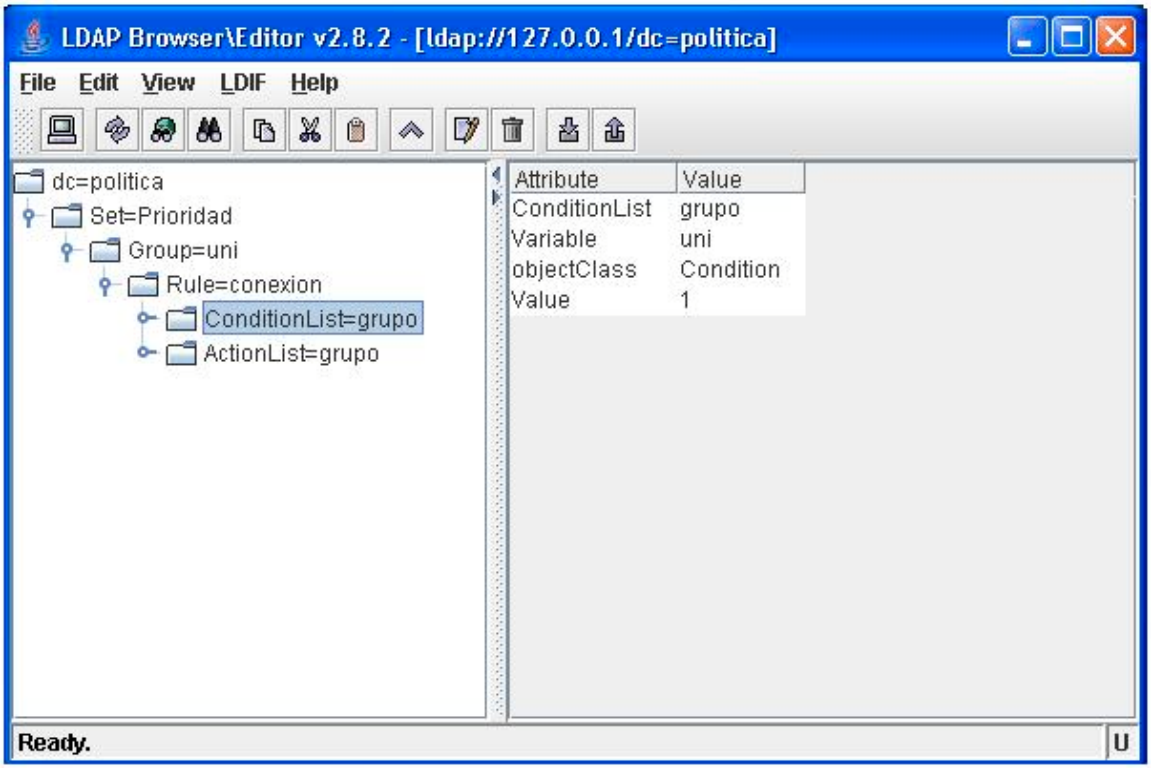

Fig. 3.11 LDAP Browser/Editor

En la figura 3.11 observamos que podemos ver la jerarquía en árbol de las entradas de una forma visual, que es mucho más amigable de verlo que en el LDIF o haciendo un ldapsearch. Hay que comentar una diferencia de los attributetype de la captura del LDAP Browser respecto al LDIF mostrado en la figura 3.8. Los attributetype como por ejemplo pcelsPolicySetName o pcelsGroupName y los objectclass como el pcelsConditonAuxClass , han sido cambiados por un nombre más corto como Set o Group y Condition respectivamente. Este cambio lo hemos aplicado para una mejor lectura. Eso

sí, todos estos cambios se tienen que haber definido en el schema LDAP cambiando nombre por nombre, ya que el LDAP sólo acepta los atributos o clases que tiene en el schema, no hay que equivocarse y pensar que en el browser se puede escribir o cambiar lo que quieras, éste tan solo visualiza lo que contiene el LDAP.

Al lado de la estructura en árbol se visualiza en forma de tabla los attributetype y objectclass con los respectivos valores que contiene cada entrada seleccionada. En el ejemplo hemos seleccionado la entrada “ConditionList”, la cual nos muestra los attributetype ConditionList, Variable y Value y el objectclass Condition. Todo esto es lo mismo que podriamo ver en el LDIF.

Por último, observar lo mencionado anteriormente sobre los mensajes de estado y el color de los mismos.

Observando la figura 3.11 podemos ver que ésta tiene unos desplegables con diferentes opciones. A continuación, se explicaran rápidamente algunas de las opciones, para finalmente centrarnos en la opción Edit → Add que es la realmente interesante. Esta opción será la que nos permitirá editar políticas en el LDAP mediante unas pequeñas modificaciones. La sencillez de estas modificaciones es la que nos ha hecho decidirnos por usar este browser LDAP frente a otros de libre distribución como el softerra LDAP Browser, LDAP explorer Tool o phpLDAPadmin.

La opción File nos permite abrir, cerrar, reconectarte o salvar una sesión. En new window se puede tener mas de un browser abierto con conexiones a diferentes database del mismo o distinto LDAP.

La opción Edit permite añadir y borrar atributos y entradas o editarlos si ya están creados. También puedes copiar una entrada o moverla a otra rama del árbol.

La opción View permite ver las características de las entradas o atributos seleccionados. También permite buscar un DN, refrescar una entrada, ordenar la visualización del árbol o de la tabla en modo ascendente o descendente.

La entrada LDIF permite importar o exportar un fichero LDIF. Si se exporta se puede elegir entre exportar sólo la entrada seleccionada, sólo la entrada con sus hijos intermedios o la entrada con todos los hijos.

Después de ver toda la visualización del directorio LDAP (browser), veamos la edición de entradas en el LDAP (editor).

Para la edición de las políticas utilizaremos la opción Edit → Add Entry. Esta opción coge las entradas de un fichero llamado templates con extensión “web Configuration File”. En este fichero escribiremos el nombre que se quiera poner a la entrada, de tal manera que tendremos Edit → Add Entry → “Nombre que queramos”. Debajo del nombre, se pondrán todos los objectclass necesarios para esa entrada. Para finalizar, se pondrán los dos puntos.

A la izquierda de la figura 3.10 tenemos dos entradas creadas en el fichero templates. A una entrada le hemos dado el nombre de Set para indicar el attributetype pcelsPolicySetName, y hemos introducido el objectclass que lo contiene (pcelsPolicySet). A la otra entrada lo mismo con el nombre Group para indicar el attributetype pcelsGroupName y su objectclass pcelsGroup.

A la derecha de la figura 3.12 tenemos como saldrían las dos entradas creadas. Se pueden crear cuantas se quiera y con el nombre que se desee.

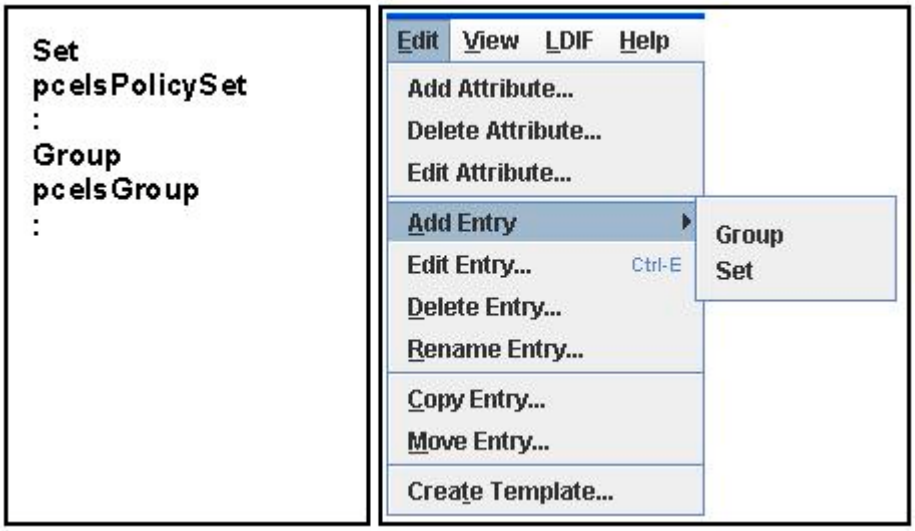

Fig. 3.12 Creación de entradas

Después de esto, hay que crear un fichero del mismo nombre que la entrada "inventada" en el fichero templates y con extensión TEMPLATE, por ejemplo necesitaremos un fichero Set.TEMPLATE.

En el fichero Set.TEMPLATE escribiremos el PREFIX, éste será el nombre del *attributetype* que se desea para esa entrada. Este atributo tiene que estar contenido en uno de los *objectclass* puestos en el fichero templates (pcelsPolicySet) y su nombre tiene que coincidir con el nombre del *attributetype* que hay en el fichero *schema*. Luego siguiendo el mismo criterio que en el fichero s*chema* pondremos los *attributetype* obligatorios (REQUIERED ATTRIBUTES) y los *attributetype* opcionales (OPTIONAL ATTRIBUTES) de cada *objectclass.*

En la figura 3.13 podemos ver los ficheros Set.TEMPLATE y Group.TEMPLATE.

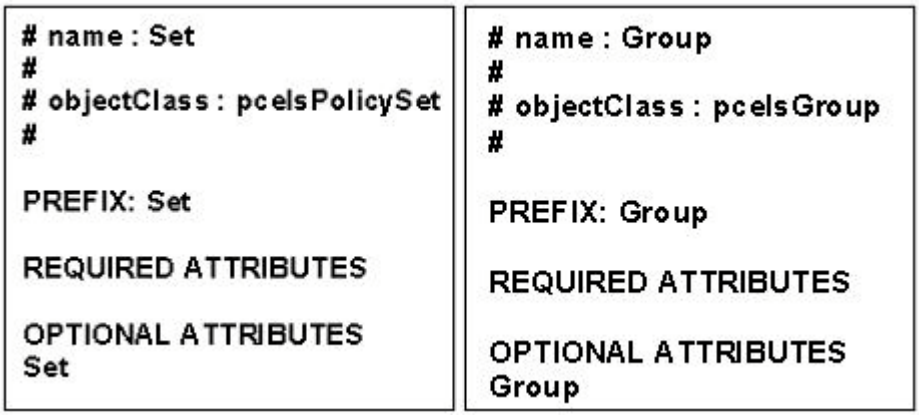

Fig. 3.13 Ejemplos Ficheros TEMPLATE

Como vimos en el tema diseño y edición de políticas, hemos creado las políticas mediante el lenguaje de programación JAVA (JNDI). Este programa lo que hace es leer la información de los ficheros comentados y actuar en consecuencia.

CAPÍTULO 4. ESCENARIO DE PRUEBAS

El escenario de pruebas que se ha creado es un programa realizado en JAVA que gestionará mediante políticas el acceso a los datos de las personas de una empresa guardados en una base de datos SQL[29][30]. En la figura 4.1 podemos ver el esquema del escenario de pruebas.

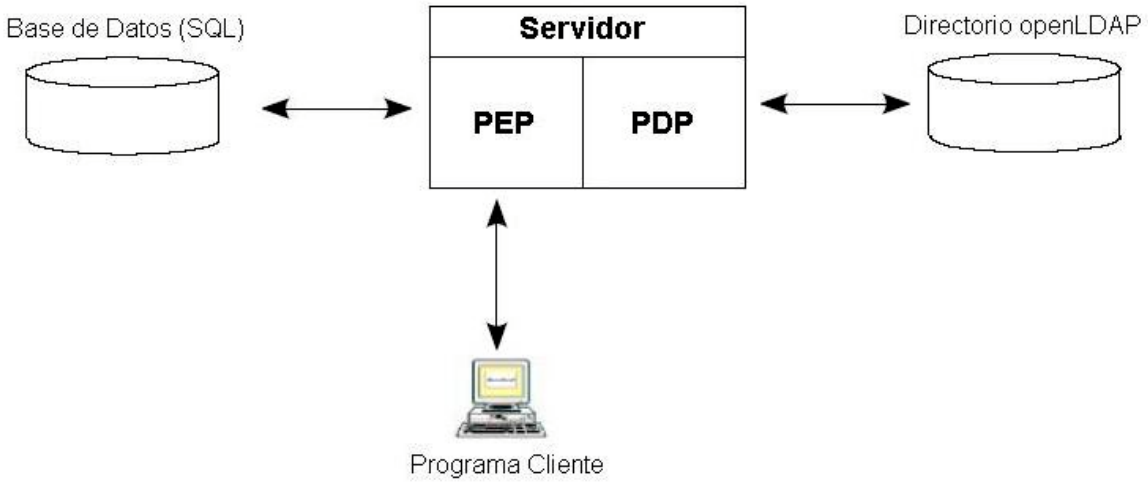

Fig. 4.1 Escenario de pruebas

El programa cliente, es dónde nos identificaremos como usuarios, introduciendo el password, grupo y número del grupo. También introduciremos la dirección IP del servidor al que nos conectemos. Una vez identificados se nos aplicarán las políticas que gestionan el acceso a los diferentes directorios de ficheros y datos que hay en ellos.

Una vez aplicadas las políticas tendremos acceso a la lectura de todos, algunos o ningun directorio de ficheros. Al hacer click en una carpeta determinada podrán ocurrir dos cosas: que se nos permita el acceso y por tanto podamos ver las fichas administrativas de las personas que hay en las carpetas seleccionadas, o que se nos deniege ver la ficha. En la figura 4.2 podemos ver el programa cliente.

Fig. 4.2 Programa Cliente

Para la creación del programa cliente ha sido necesaria la utilización del AWT (Abstract Windows Toolkit)[15]. El AWT es la parte de JAVA que se ocupa de construir interfaces gráficas de usuario. También necesitaremos el JFC (Java Foundation Classes)[36] que comprende un grupo de características para ayudar a construir interfaces gráficos de usuario (GUIs), la versión JFC 1.1 también es conocido como el API Swing.

El servidor será el que contendrá tanto el PEP como el PDP. El PEP será el que reciba las peticiones del programa cliente y quien tenga acceso a la base de datos SQL. El PDP será el que reciba las peticiones de los clientes que le pase el PEP y el que consulte el LDAP para decidir las políticas a aplicar.

Para las consultas que realice el PDP al LDAP, se utilizará el Java Naming and Directory Interface (JNDI). Éste es una interfaz de programación (API) que proporciona funcionalidades de nombrado y directorio a las aplicaciones escritas usando Java. Está definido para ser independiente de cualquier implementación de servicio de directorio. Así se puede acceder a una gran variedad de directorios (nuevos, emergentes, y ya desarrollados) de una forma fácil.

En el servidor se han implementado threads o hilos, de esta manera podrá soportar múltiples peticiones de clientes.

Los pasos que sigue el programa son los siguientes:

1. Un cliente introduce sus datos y envia una petición de autentificación al servidor.

2. El PEP recibe esta petición y la pasa al PDP.

3. El PDP consulta la base de datos “people” creada en el LDAP para ver si el cliente existe.

4. El PDP responde al PEP y éste al cliente, indicándole mediante un mensaje si esta autentificado o no.

5. Si esta autentificado, el PDP consultara la base de datos “policy” creada en el LDAP y mirara las políticas aplicadas que le daran permiso a la visualización de las fichas técnicas.

6. El PEP recibe del PDP las políticas que tiene aplicadas el usuario. Éstas políticas otorgarán derechos para ver ciertas fichas técnicas. Éstas fichas seran leidas de una base de datos SQL.

7. El PEP enviara al cliente las fichas técnicas a las que tiene acceso.

CAPÍTULO 5. Mejoras y ampliaciones futuras

Una vez finalizado el proyecto presentaremos una sección con diferentes ideas para desarrollos futuros. Estos desarrollos no han sido factibles ya sea por tiempo o por recursos. Por ello hay ideas explicadas de una forma teórica, como la ampliación del esquema LDAP, o tratadas con más detalle, como la replicación de la información.

5.1 Ampliación del esquema LDAP

Para la realización del esquema se utilizó el RFC 4104 (PCELS). Este esquema es muy extenso en cuanto a clases y atributos. En el proyecto se optó por crear un esquema reducido, el cual fuese suficiente para realizar una gestión basada en políticas.

Un desarrollo futuro sobre el proyecto puede ser la ampliación del esquema creado. Para ello, primero hay que definir que se quiere gestionar mediante la utilización de las políticas. Una vez hecho esto, tan sólo habrá que buscar en el PCELS los objectclass que contengan los atributos que creamos necesarios y copiarlos en el policy.schema, teniendo en cuenta la dependencia de cada objectclass.

A modo de ejemplo, podríamos decir que queremos aplicar una política sólo en cierto espacio de tiempo. Podríamos no dejar conectarse a la red los domingos. Para esta política sería muy útil la clase pcimRuleValidityAssociation ya que con esta clase se puede representar la activación o desactivación de una regla en un periodo de tiempo. Esta clase contiene los atributos pcimValidityConditionName y pcimTimePeriodConditionDN. El primero sirve para darle nombre a la clase y el segundo para indicar el periodo de tiempo en que es útil la condición.

5.2 Replicación de directorios

Una de las mejoras que se puede realizar en el proyecto es la replicación de la información del LDAP, teniendo así varios servidores a consultar. Esto nos será muy útil para solventar fallos por si algún servidor se cae o para repartir la carga si tenemos muchos clientes.

Para replicar un directorio con OpenLDAP debemos elegir un servidor master y uno o más servidores slave. En el servidor master se configura el *daemon* slurp que se encargará de las replicaciones.

En la figura 5.1 se muestran las líneas que hay que añadir en el slapd.conf del servidor maestro.

```
replogfile    /opt/openldap/var/openldap-slurp/replica/slurpd.replog
replica host = "IP o host de la maquina a replicar (slave)"
bindmethod = simple
binddn = "cn=root,dc=epsc,dc=com"
credentials=secret
```

Fig. 5.1 Configuración slapd.conf para replicar información (master)

En la primera línea (replogfile) indicamos donde está el archivo que la máquina master pasará a la máquina slave.

Luego en la segunda línea indicamos la IP o host donde será enviado el fichero. Con la sentencia bindmethod decimos como queremos pasar el fichero, si lo queremos con algún tipo de codificación o no. En el ejemplo se pone simple, así que se transmite sin codificar.

Por último, en la cuarta y quinta línea vemos el "login" para autentificarse a la máquina master. Éste está compuesto por el binddn dónde ponemos el dn de root y por el credentials donde se introduce el password.

A partir de ahora si queremos añadir mas servidores esclavo tan solo habrá que volver a poner las líneas de la 2 a la 5. Notar que la única línea a cambiar obligatoriamente será la 2. La línea 3 puede cambiarse o ser igual, mientras que la línea cuarta y quinta tendrán que ser iguales.

Después de haber modificado el slapd.conf del master habrá que introducir unas líneas en el slapd.conf del slave. Estas líneas están recogidas en la figura 5.2.

```
Updateref ldap://"host o IP de la maquina master":389
Updatedn "cn=root,dc=epsc,dc=com"
```

Fig. 5.2 Configuración slapd.conf para replicar información (slave)

En la primera línea indicamos el host o IP de la máquina master y a continuación, después de los dos puntos se pone el puerto por el que se va a transmitir. El puerto 389 es el puerto well-known para LDAP. La transmisión de la información por este puerto se hace de forma simple, así que como vemos cuadra con el bindmethod de la configuración del master. El LDAP permite transmitir la información mediante SSL. Si éste hubiese sido el caso, se tendría que haber introducido el puerto 636.

En la segunda línea tan sólo nos indica el nombre de la entrada del administrador del directorio.

A partir de este momento cada cambio que se realice en el servidor master será replicado a todos los servidores slave.

5.3 DSML

Otra de las mejoras estudiadas para este proyecto es la creación de un servidor DSML que haga de intermediario entre las peticiones y respuestas de los clientes al LDAP y viceversa.

El Directory Services Markup Language (DSML) es un XML para representar la información de directorios. Es un formato genérico para importar o exportar información de un directorio. La información de directorio en DSML puede ser compartida entre dispositivos DSML sin exponer el protocolo LDAP.

XML proporciona un modo eficaz para representar y transferir datos. DSML ha sido diseñado como una vía estándar para acceder a los directorios como el LDAP. Mientras que LDAP proporciona una forma de extraer datos de los directorios, DSML describe esos datos de forma genérica, convirtiéndolos en texto plano que los desarrolladores podrán utilizar para procesarlo en aplicaciones basadas en Web.

DSML nació gracias a la unión de seis grandes fabricantes que se unieron para promover una vía de estandarización que permitiera a sus respectivos productos intercomunicarse. Estas Empresas fueron IBM, Microsoft, Oracle, Sun, Netscape y la firma Bowstreet Software. Estas seis compañías remitieron el borrador de la especificación DSML 1.0 a OASIS (Organization for Advancement of Structured Information Standards)[31], un consorcio internacional sin ánimo de lucro considerado por muchos como la organización independiente líder en la estandarización de aplicaciones XML para el comercio.

DSML v1.0 se creó a finales de 1999, pero éste sólo proporcionaba una expresión meta del modelo de datos de directorio y su estructura y tenia fallos en operaciones de preguntas y actualizaciones hacia los directorios.

En 2001 surgió DSML v2.0, éste proporcionaba un método para expresar preguntas, actualizaciones y representar los resultados en XML, y de esta manera ser más útil para los programadores.

A continuación presentamos algunas diferencias entre DSML v2.0 y el directorio LDAP:

- Autentificación: La petición LDAP contiene autentificación, mientras que la petición DSMLv2 no la usa. Esta es la causa por la que el documento DSMLv2 puede ser transportado a una gran variedad de mecanismos. Pero esto no significa que DSMLv2 no pueda ser usado para la autenticación, de hecho, DSMLv2 incluye una solicitud “Auth” que puede ser usada para dar seguridad a una colección de operaciones DSMLv2.

- Agrupación de operaciones: LDAP no tiene implementada una operación para agrupar peticiones en una sola. DSMLv2 puede agrupar múltiples operaciones LDAP para ser expresadas en una única petición. DSMLv2 especifica una correspondencia entre peticiones y respuestas individuales.

En la figura 5.3 vemos una típica transacción DSML, entre una aplicación XML y un directorio LDAP.

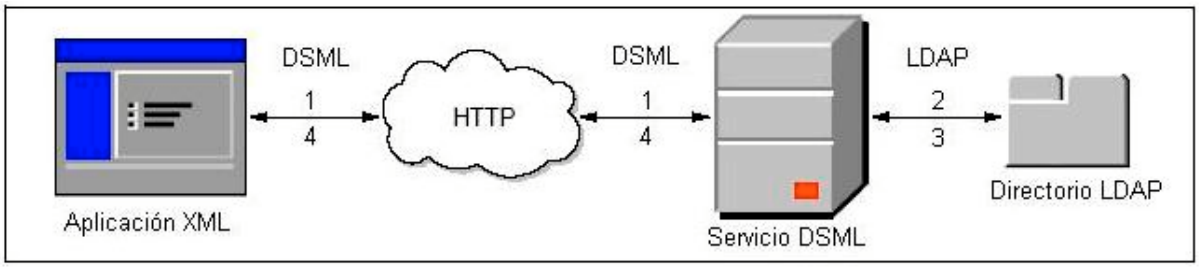

Fig. 5.3 Transacción DSML

Los pasos que sigue esta transacción son los siguientes:

1. La aplicación XML envía una petición DSML a través de un HTTP.

2. El servicio DSML recibe la petición y traslada esta petición al LDAP.

3. El servicio DSML recibe los datos del directorio LDAP y los transforma al formato DSML.

4. El servico DSML envía el resultado de la petición a la aplicación XML a través de un HTTP.

Para que la transacción entre el servicio DSML y el LDAP sea correcta, hay que modelar el schema DSML al schema LDAP. En capítulos anteriores hablamos ampliamente sobre el schema LDAP, por tanto no nos será difícil entender el schema DSML.

En la figura 3.5 vimos un ejemplo de objectclass como era el pcelsGroup y su attributetype pcimGroupName. Estos estaban mostrados tal y como estaban representados en el LDAP schema. Ahora en la figura 5.4 tenemos la misma parte del schema, pero esta vez representado en un schema DSML.

```
<!--<xml:batchRequest xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core'>-->
<dsml:dsml xmlns:dsml="http://www/dsml.org/dsml'>

<!--Objectclass-->
<dsml:directory-schema>
<dsml:class id="pcelsGroup" superior="#pcelsPolicySet" type="abstract'>
<dsml:name>pcelsGroup</dsml:name>
<dsml:description>RFC4104</dsml:description>
<dsml:object-identifier>1.3.6.1.1.9.1.3</dsml:object-identifier>
<dsml:attribute ref="#pcimGroup" required="false"/>
</dsml:class>
</dsml:directory-schema>
</dsml:dsml>

<!--attributetype-->
<dsml:directory-schema>
<dsml:attribute-type id="pcimGroupName">
<dsml:name>pcimGroupName</dsml:name>
<dsml:description>Name</dsml:description>
<dsml:object-identifier>1.3.6.1.1.6.2.4</dsml:object-identifier>
<dsml:syntax bound="32">1.3.6.1.4.1.1466.115.121.1.15</dsml:syntax>
<dsml:equality>caseIgnoreMatch</dsml:equality>
 <dsml:ordering>caseIgnoreOrderingMatch</dsml:ordering>
<dsml:substring>caseIgnoreSubstringsMatch</dsml:substring>
</dsml:attribute-type>
</dsml:directory-schema>
```

Fig. 5.4 Esquema DSML

En la figura podemos ver que la información está muy bien estructurada en etiquetas, ya que el DSML es un XML. También vemos que los campos se asemejan mucho a los campos del schema LDAP.

A parte de mapear el *schema* LDAP correspondiente al servidor DSML hay que mapear todas las demás funciones del LDAP (entradas y modificaciones al directorio, todos los errores que puede dar el LDAP, etc). La explicación de todo esto es muy extensa, por tanto para seguir desarrollando esta idea se recomienda la lectura del Standard DSMLv2 en OASIS.

Bibliografía y referencias.

[1] openLDAP (http://www.openldap.org).

[2] BerkeleyDB (www.sleepycat.com).

[3] LDAP Browser/Editor (http://www-unix.mcs.anl.gov/~gawor/ldap/).

[4] IETF – Internet Engineering Task Force (www.ietf.org).

[5] Rosana Lee “The JNDI Tutorial” (http://java.sun.com/products/jndi/tutorial/).

[6] LDAP Naming Service Provider for the Java Naming and Directory Interface™ (JNDI) (http://java.sun.com/j2se/1.3/docs/guide/jndi/jndi-ldap.html).

[7] B.Moore, E. Ellesson, J. Strassner, A. Westerinen “Policy Core Information Model -- Version 1 Specification” RFC 3060.

[8] M. Pana, A. Reyes, A. Barba, D. Moron, M. Brunner “Policy Core Extension Lightweight Directory Access Protocol Schema (PCELS)” RFC 4104.

[9] DMTF – Distributed Management Task Force (www.dmtf.org).

[10] Ponder: A Policy Language for Distributed Systems Management (http://www-dse.doc.ic.ac.uk/Research/policies/ponder.shtml).

[11] DSML (http://www.dsmltools.org).

[12] Tutorial DSML – OASIS (http://www.oasis-open.org/committees/dsml/docs/DSMLv2.doc).

[13] M. Wahl, T. Howes S. “Lightweight Directory Access Protocol (v3)” RFC 2251

[14] G. Good. “The LDAP Data Interchange Format (LDIF)” RFC 2849

[15] Javier García de Jalón, José Ignacio Rodríguez, Iñigo Mingo, Aitor Imaz, Alfonso Brazalez, Alberto Larzabal, Jesús Calleja, Jon García. “Aprenda Java como si estuviera en primero” Campus Tecnológico de la Universidad de Navarra, San Sebastián, 2000. (http://www.unav.es/cti/manuales/)

[16] J. Case, M. Fedor, M. Schoffstall, J. Davin. “A Simple Network Management Protocol (SNMP)” RFC 1157

[17] SNMP (http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/snmp.htm)

[18] CMIP (http://www.arrakis.es/~gepetto/redes/rog08p7.html)

[19] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry. “The COPS (Common Open Policy Service) Protocol “ RFC 2748.

[20] S. Kille, M. Wahl, A. Grimstad, R. Huber, S. Sataluri “Using Domains in LDAP/X.500 Distinguished Names” RFC 2247.

[21] M. Wahl. “A Summary of the X.500(96) User Schema for use with LDAPv3” RFC 2256

[22] B.Moore “Policy Core Information Model (PCIM) Extensions” RFC 3460.

[23] J. Strassner, B. Moore, R. Moats, E. Ellesson “Policy Core Lightweight Directory Access Protocol (LDAP) Schema” RFC 3703.

[24] J. Lobo, R. Bathia, S. Naqvi. “A policy Description Language”. (http://citeseer.ist.psu.edu/lobo99policy.html)

[25] C.Ribeiro, A. Zúquete, P. Ferreira, P. Guedes. “SPL: An access control language for security policies with complex constraints” (www.inesc-id.pt/pt/indicadores/Ficheiros/1162.pdf).

[26] Brett McLauglin “Java y XML” Ediciones Anaya Multimedia, 2001.

[27] L. Bergman, OpenLDAP for Win32 (http://lucas.bergmans.us/hacks/openldap/)

[28] Tutorial LDAP – YoLinux (http://www.yolinux.com/TUTORIALS/LinuxTutorialLDAP.html).

[29] SQL-Manual (http://www.desarrolloweb.com/manuales/9/)

[30] SQL-Software (www.apachefriends.org/en/xampp.html)

[31] OASIS (www.oasis-open.org/)

[32] John C. Strassner, “Policy-Based Network Management” Morgan Kaufmann Publishers, San Francisco, 2004.

[33] Jonathan Follows, Detlef Straeten “Application-Driven Networking: Concepts and Architecture for Policy-Based Systems” (http://www.redbooks.ibm.com).

[34] Sotware Eclipse (http://www.eclipse.org).

[35] C.bonfill, A.Barba. Proyecto de final de carrera: “Implementación de un sistema de gestión basado en políticas: Configuración de red, monitorización y control JDMK” Mayo 2003

[36] Swing y JFC (http://www.programacion.com/java/tutorial/swing/)

ANEXO I. Instalación y configuración del openLDAP.

Se instaló el servidor openLDAP versión estable 2.2.26 en dos ordenadores diferentes.

Primero tuvimos el Pc1, que fue utilizado hasta la mitad de proyecto ya que sus características eran insuficientes para seguir avanzando. Entonces se optó por utilizar el Pc2, con capacidad y sistema operativo suficiente.

El LDAP Browser/Editor y el programa en Java que creamos con el software eclipse[34] funcionaban en el Pc1, pero no lo hacía con fluidez.

Características de los PC´s:

- Pc1:

Procesador: AMD Athlon 1000.
Memoria RAM: 128 Mb.
Disco Duro: 20 Gb
Sistema operativo: Linux Fedora Core 1.0

- Pc2:

Procesador: Pentium V 2,4 GHz.
Memoria RAM: 256 Mb.
Disco Duro: 60 Gb
Sistema operativo: Windows XP

a. Instalación en Linux

El primer paso es descargar la ultima versión estable del database Berkeley en la dirección http:// www.sleepycat.com/, obteniendo así el fichero db-4.3.28.tar.gz.

Descomprimimos el fichero y lo instalamos:

```
$ tar xvzf db-4.3.28.tar.gz
$ cd db-4.3.28
$ cd build_unix
$ ../dist/configure –prefix=/usr/local/BerkeleyDB-4.3.28/
$make
```

Ahora como root:

```
$make install
```

El segundo paso es descargarse la última versión del servidor openLDAP de www.openldap.org. Obtenemos el fichero openldap-stable-2.2.26.tar.gz.

Descomprimimos el fichero y lo instalamos:

$ tar xvzf openldap-stable-xxxxx.tar.gz
$ cd openldap-2.2.26
$ ./configure --enable-threads --enable-tls --prefix=/usr/local/etc/openldap-2.2.26
--enable-bdb --enable-crypt --enable-referrals
$ make depend
$ make test

Ahora como root:

$ make install

Ahora tenemos instalado:

- En /usr/local/etc/openldap-2.2.26/ tenemos el slapd.conf y los schemas
- En /usr/local/libexec tenemos los daemons slapd y slurp

Para inicializar el openLDAP lo hacemos de esta manera como root:

$.\slapd –d 1

Para inicializarlo hay diferentes flags. Ponemos los flags -d para indicar que se inicialice en forma de debug y el 1 indica el nivel de debug. De esta manera vemos cuándo se hacen transacciones o, si ocurren errores, poder ver en qué momento de la ejecución da error.

b. Instalación en Windows

Lo único que hay que hacer es descargar el paquete openLDAP para W32 de http://lucas.bergmans.us/hacks/openldap/. En este paquete viene incluido todo el software necesario. Tenemos la versión openldap-2.2.19, el backend BerkeleyDB-4.3.21 y el openssl-0.9.7e para transmisiones seguras.

Una vez hecho esto, iremos a la consola de windows e iremos al directorio dónde esta el openldap y escribiremos el mismo comando que en linux:

$.\slapd –d 1