Capítulo 3

El servidor de directorios OpenLDAP.

Introducción a LDAP

Características

Funcionamiento

Instalación y configuración

Capítulo 3 El servidor de directorios OpenLDAP

El servidor de directorios OpenLDAP es una pieza clave en la integración que vamos a llevar a cabo. Se encargará de almacenar, entre otros, los pares nombre de usuario – clave, y de hacerlos accesibles a todos los equipos de la red local. Así, de esta manera centralizada, vamos a controlar la autenticación en toda la red, simplificando la administración y el mantenimiento de los usuarios y de las claves de éstos.

A lo largo del siguiente capítulo describiremos las principales características de OpenLDAP, comprobando que es la opción más acertada para nuestro fin. También nos centraremos en su funcionamiento y veremos algunas formas de optimizar su rendimiento.

3.1 Conceptos fundamentales

3.1.1 ¿Qué es LDAP?

LDAP es un estándar que las computadoras y los dispositivos en red pueden usar para acceder a información común sobre una red de comunicaciones. LDAP provee acceso cliente-servidor sobre una red y es además un servicio de directorio, ofreciendo también capacidades de búsqueda y recuperación de la información, definiendo operaciones para añadir, actualizar y borrar información de un directorio.

No podemos empezar a hablar de LDAP sin mirar atrás, hacia su precursor, el protocolo X.500.

X.500 nació como consecuencia de la rápida expansión de los sistemas distribuidos y las redes de telecomunicación, acuciado por la necesidad de un estándar en el ámbito de los servicios de directorio de área extensa.

Los autores de las especificaciones de X.500 diseñaron cuidadosamente un protocolo que cubriera las necesidades de los directorios, tanto presentes como futuras. Parte de su trabajo, como el modelo de nombrado o el de información, han sido adoptados por varios servicios de directorios (entre ellos LDAP).

La principal causa que hacía el uso de servidores X.500 poco atractivo era la complejidad de desarrollar clientes que se comunicaran con él. El número de opciones y variantes que permite el protocolo es muy amplio. Además era necesario ejecutarlo en un entorno OSI, cuando la mayoría de clientes corrían en un entorno TCP/IP y de hecho, sólo necesitaban un pequeño conjunto de opciones. Por ello se comenzó a trabajar en un protocolo de acceso de cliente que conllevara poca carga y que al final, culminaría en un gran éxito. De hecho todos los servidores comerciales

X.500 incluyen una pasarela LDAP, que se encarga de la traducción entre el protocolo descargado de funciones y basado en TCP/IP al protocolo X.500 del servidor, como vemos en la siguiente figura.

Figura 3.1 Esquema de acceso a un servidor X.500 usando como pasarela LDAP

Otro de los puntos a favor de LDAP son las API´S que sus diseñadores implementaron. X.500 carece de éstas, lo que obliga a los desarrolladores de aplicaciones a escribir la mayoría del código para acceder a este tipo de servidores.

3.1.2 Lo que LDAP no es

LDAP es un protocolo de acceso que se usa para compartir datos de acuerdo con un modelo particular de información. Estos datos pueden residir en una base de datos, en memoria o en cualquier otro sitio donde el servidor LDAP pueda acceder. Lo importante es que estos datos se presenten al cliente LDAP de forma que se adapten al modelo de información.

LDAP se usa para numerosas aplicaciones. Muchas de ellas son propietarias pero otras no. Para poder decidir si usamos LDAP con ellas demos tener claras sus limitaciones.

LDAP no es:

- un sustituto general para bases de datos relacionales
- un sistema de archivos para objetos muy grandes
- óptimo para objetos muy dinámicos
- útil sin aplicaciones.

3.1.2.1 LDAP no es una base de datos relacional

Se podría decir que es una base de datos especializada pero tiene bastantes características que lo distinguen de una base de datos relacional. No soporta integridad relacional, transacciones…

3.1.2.2 LDAP no es un sistema de archivos para objetos grandes

LDAP provee de una forma jerárquica de información de nombrado que se puede encontrar en muchos sistemas de archivos y podría pensarse que LDAP es ideal para guardar archivos y acceder a ellos en red. Esto es erróneo. No es la mejor forma de compartir archivos. Aunque permite almacenar y transmitir datos, no posee el bloqueo, la búsqueda ni las características avanzadas de los más modernos protocolos de compartición de archivos.

3.1.2.3 LDAP no es óptimo para objetos muy dinámicos

Los servidores LDAP están optimizados para búsqueda, fundamentalmente a cambio de un peor comportamiento a la hora de actualizar. Incluso no da garantías que aseguren un conjunto de transacciones se produzcan en orden correcto.

3.1.2.4 LDAP no es útil sin aplicaciones

LDAP no posee un lenguaje general como podría ser SQL para bases de datos. Estos lenguajes permiten hacer búsquedas avanzadas, que en el caso de directorios no hacen falta, ya que se usan para información más general, usada por múltiples aplicaciones. Es muy importante que los servicios de directorio sean diseñados e implementados con una cooperación total de los desarrolladores de aplicaciones que utilizan dicho servicio.

Aunque carezca de un lenguaje general de generación de informes, posee bastantes APIs, y muchas de ellas están basadas en estándares y muy bien

documentadas, cuya aceptación global ha sido uno de los pilares básicos de la temprana adopción de LDAP. Al contrario que las bases de datos, los directorios que usan LDAP poseen un protocolo que puede ser usado sin necesidad de drivers especiales.

Esto hace de los directorios una pieza muy importante a la hora de compartir información entre aplicaciones que no tengan nada en común.

LDAP es hoy en día una tecnología madura con un amplio abanico de aplicación que incluye autenticación, autorización, gestión de correo y un largo etcétera. El desarrollo de muevas aplicaciones es continuo, lo que asegura que la importancia de LDAP seguirá creciendo.

3.1.3 Aplicaciones actuales

A continuación vamos a ver una serie de aplicaciones que se apoyan de manera imprescindible en los servicios de directorio.

3.1.3.1 Páginas blancas

Uno de los primeros usos de los servicios de directorios fue la provisión electrónica de libros de direcciones, llamados comúnmente páginas blancas. Esta aplicación explota la característica principal de optimización para lectura que poseen los directorios.

3.1.3.2 Autenticación y autorización

En este sentido, actualmente LDAP es un estándar de facto en cuanto al acceso a la información de identificación y credenciales para permitir la autenticación. La autenticación, como ya hemos visto, es el proceso de validar la identidad de un usuario (o cualquier otro objeto, como por ejemplo una aplicación).

La información almacenada usando LDAP es de más fácil acceso que por los medios tradicionales. Además tiene la ventaja de que podemos reutilizar la información de identificación. Por ejemplo, usando LDAP permitimos a dos servidores uno en UNIX y otro en Windows, ejecutando una aplicación particular cada uno, autenticar usuarios de la misma forma y del mismo repositorio. De hecho el tiempo de desarrollo de aplicaciones se reduce y el código para la autenticación sería relativamente estático entre ambas plataformas con lo que el coste de gestión y mantenimiento de dos repositorios de identificación se eliminaría.

Éste será el uso principal que tendrá nuestro directorio.

3.1.3.3 Personalización y perfil

Una vez que el usuario se ha identificado a través de la autenticación se podría personalizar su entorno de acuerdo con sus preferencias. Podemos encontrar en esto una gran utilidad en el caso de que estemos en un entorno complejo donde muchas de sus características han de ser configuradas al inicio. Esto tiene una gran ventaja y es que los contenidos personalizados pueden ser consistentes entre múltiples aplicaciones.

Aclarar la distinción entre personalización y perfil. La primera está más relacionada con los contenidos y la segunda con las preferencias operacionales. Manteniendo el perfil en el directorio podríamos cambiar de máquina y obtener un entorno idéntico.

Además permitiendo el perfil se podrían crear grupos con características comunes o inhabilitar ciertas operaciones que se consideren prohibidas.

3.1.3.4 Infraestructura de clave pública

La autenticación clásica usa sistemas de clave secreta. La forma de autenticar era la siguiente: se toma un mensaje, se le aplica la clave y se

mandan ambos, el mensaje en claro y codificado. El otro extremo, que conoce la clave, se la aplica el mensaje y compara. Si son iguales, la autenticación ha tenido éxito.

Este simple sistema es válido cuando tenemos pocos usuarios y se puede compartir fácilmente la clave secreta.

La tecnología de clave pública cambia todo esto y lo hace más escalable. En ella tenemos dos claves; la privada, que no se comparte con nadie, y la pública que es conocida por todos. La forma de distribución suele ser mediante un certificado digital, y estos podemos encontrarlo en un directorio. Centralizar este almacenamiento permite a las personas y aplicaciones encontrar certificados bajo demanda según el extremo con el que quiera mantener una comunicación segura.

3.1.3.5 Entrega de mensajes

Una dirección de correo válida (aunque ficticia) podría ser juanlopez@micorreo.es.

Un mensaje mandado a la dirección anterior se encaminaría gracias a la parte a la derecha de @, es decir gracias al dominio. Esto se realiza mediante el DNS.

Una vez que el mensaje ha llegado a la máquina correcta, se entrega en esa máquina usando el nombre de usuario, que es la parte de la dirección que se encuentra a la izquierda de @.

Muchos sistemas de correo soportan el uso de LDAP para determinar como entregar el mensaje.

El proceso de entrega puede incluir operaciones avanzadas, aunque por el momento, se usa principalmente sólo para alias de nombres enteros e implementar listas de correo.

Otra característica es la facilidad para crear grupos ya sea una lista de usuarios o una especificación dinámica (como podría ser usuarios en un departamento).

Esta será otra de las facilidades que usaremos al integrar el agente de transferencia de correo qmail con LDAP.

3.2 Los modelos de LDAP

Los modelos de LDAP representan los servicios que proveen los servidores, tal como los ve el cliente. Son modelos abstractos que describen los diferentes aspectos de un directorio LDAP. Existen 4 modelos.

- Modelo de información
- Modelo de nombrado
- Modelo funcional
- Modelo de seguridad

3.2.1 Modelo de información

Este modelo provee de las estructuras y tipos de datos necesarios para construir un árbol de directorios LDAP. La unidad básica en un directorio LDAP es la entrada. Una entrada se puede ver como un nodo en el árbol de información de directorio (DIT). Una entrada contiene información sobre una instancia de uno o más objectClass. Estos objectClass son unos objetos que tienen ciertos atributos, algunos opcionales y otros obligatorios. Los atributos pueden ser de distintos tipos y cada tipo lleva asociado reglas de codificación y de coincidencia que tienen en cuenta cosas como qué tipo de dato puede tomar este atributo o como compararlo en una búsqueda. Veamos como sería una entrada simple.

```
dn: cn=Jose Martin, o=caos, c=sp
objectClass: person
cn: Jose Martin
sn: Martin
```

En la primera línea encontramos el nombre distinguido (dn) y en las siguientes atributos. Los atributos están compuestos por pares tipo-valor (es). El tipo de atributo se separa del valor mediante dos puntos (:). En este ejemplo el atributo dn no es ningún atributo, es el nombre distinguido de la entrada. Este nombre es único en todo el directorio y es una manera muy

útil de referenciar una entrada individual. Por último resaltaremos el atributo objectClass, que es un atributo especial que determina que atributos pueden (o deben) ser almacenados en una entrada particular.

3.2.1.1 El esquema

El esquema soportado por un servidor LDAP determina el tipo de información que puede ser almacenada en un directorio particular. Un esquema consiste en un conjunto de definiciones de tipos de atributos y de clases de objetos.

Antes de crear un esquema conviene tener en cuenta el esquema estándar. Todo el mundo tiene su propia idea de cómo nombrar un tipo de atributo. Con los esquemas estándar nos aseguramos de que todos los usen los mismos nombres cuando hablamos de las mismas cosas. De la misma forma las aplicaciones no solo trabajarían con un directorio sino con todos los que soporten el mismo esquema anterior.

Las clases de objetos y tipos de atributos estándar se definen en muchos sitios. Por ejemplo el IETF en los RFC 2252 y RFC 2256 definieron algunas de las clases de objetos que mas se utilizan, como son person, organizationalPerson u organization.

3.2.1.2 Tipos de atributos

A continuación vamos a ver como se componen y definen los tipos de atributos. La definición de los tipos de atributos incluyen los siguientes componentes.

- Nombre
- ID del objeto (OID)
- Sintaxis
- Reglas de concordancia
- Herencia

Nombre

Los nombres de tipos son cadenas que solo contienen letras, números, guiones (-) y punto y coma (;). Además no se distinguen mayúsculas, minúsculas o combinación de estas, todos los casos son el mismo. El (;) tiene un uso especial que se verá más adelante.

OID

Un tipo de atributo se asocia siempre con un OID. Este OID es una ristra de números separados por puntos, que siempre es único. Así el nombre puede tomar diferentes formas según el idioma en el que estemos trabajando, pero el OID hará que sea el mismo atributo.

Los OIDs más altos los asigna el ANSI. Para conseguir uno hay que registrarse en él (http://www.ansi.org).

Definición de sintaxis

El protocolo LDAP intercambia toda la información como cadenas de caracteres de 8 bits. La sintaxis te dice como tratar esos bits (si son enteros, cadenas…).

Las definiciones de sintaxis también tienen OIDs, así se prevén conflictos y se reduce la dependencia de los atributos con nombres anglosajones.

Un tipo de atributo también puede definir su sintaxis mediante la especificación de su sintaxis y un tamaño. Así podemos limitar los tipos y prevenir posibles desbordamientos en las aplicaciones o en los servidores.

Reglas de concordancia

Los tipos de atributos llevan asociados estas reglas que indican como se van a comportar los atributos ante una búsqueda. A un cliente LDAP no se

le permite especificar la regla a usar en una búsqueda. Las reglas son totalmente dependientes de la definición del atributo en el servidor. Se pueden usar 4 tipos de reglas:

- Igualdad. Se determina la igualdad entre el valor de este atributo y el valor dado en la búsqueda.
- Mayor o menor que. Estos se usan para determinar si un valor es mayor/menor que otro.
- Subcadena. Se determina si un valor contiene a otro.
- Subesquema. Se encuentra información del esquema soportado por un directorio.

Herencia

Muchos atributos comparten características. En lugar de redefinirlos cada vez y crear un nuevo tipo, se puede crear un tipo generalizado que contenga los elementos comunes. Así podríamos crear tipos especializados que heredarán todas las características del tipo generalizado.

3.2.1.3 Clases de objetos

Las clases de objetos en LDAP muestran que atributos se requieren y cuales son opcionales en una entrada en particular. Las entradas se encuadran en una clase mediante el uso de un atributo especial llamado objectClass.

Como tipo de atributo que es, incluye los siguientes componentes:

- Nombre
- OID
- Herencia

Además incluye información que define los contenidos permitidos de las entradas que usen esta clase de objeto:

- Tipo de clase
- Lista de atributos requeridos
- Lista de atributos permitidos

En cuanto a la herencia de una clase de objeto sólo pueden tener una clase superior. La herencia múltiple no se soporta, pero una entrada puede ser miembro de varias clases de objetos, aunque estas clases no estén relacionadas. Una entrada definida de este modo adoptará los atributos requeridos y permitidos de la unión de las dos clases de objetos de las que son miembros.

Existen 3 tipos de clases de objetos y se deben a que no todos se crean de la misma forma.

- Abstractas
- Estructurales
- Auxiliares

Abstractas

Una clase definida como abstracta nunca es el objeto primario de una entrada. Más bien, lo que contiene es una lista de tipos de atributos requeridos y permitidos que son comunes a una gran variedad de clases de objetos. Una clase abstracta se puede usar como superclase para otros tipos de clases de objetos.

La clase de objeto `top` es un ejemplo de clase abstracta. Está definida para requerir el atributo objectClass. Cada clase de objeto LDAP extiende siempre la clase `top`, por lo que cada entrada debe contener un atributo objectClass.

```
top object class
Description       The root object class
Type              Abstract
OID               2.5.6.0
Required          objectClass
```

```
Allowed          dITStructureRules, nameForms, ditContentRules,
                 objectClasses, attributteTypes, matchingRules,
                 matchingRuleUse
Defined          RFC 2256
```

Estructurales

Cada entrada pertenece exactamente a una clase de objetos estructural. Un ejemplo de clase estructural es la clase organizationalPerson, que extiende la clase person, que también es de clase estructural. Debe quedar claro que cada objeto orgazationalPerson es uno person, pero un objeto person no es organizationalPerson.

```
organizationalPerson object class
Description      A person belonging to an organization
Type             Structural
Superior         person
OID              2.5.6.7
Required         sn,cn
Allowed          userPassword,telephoneNumber,seealso,description
Defined          RFC 2256
```

3.2.1.4 Diseño del esquema mediante el modelado de objetos

A la hora de diseñar clases y relaciones es muy usual utilizar el modelado de objetos para implementarlo después en un lenguaje orientado a objetos.

El uso de este modelado permite asegurar que las clases creadas serán reutilizables y extensibles. Debido a que LDAP soporta muchos conceptos de la orientación a objetos, se pueden usar algunas técnicas de modelado para diseñar nuevos objetos de directorio antes de su creación en algún servidor particular.

El modelado de la información de directorio se escapa del ámbito del proyecto ya que utilizaremos siempre un esquema estándar.

3.2.2 Modelo de nombrado

Las entradas en LDAP tal como ocurre en X.500 se organizan en árbol. En la parte superior encontramos con uno o varios nodos raíz, que se llaman sufijos o contextos de nombrado. Bajo cada nodo raíz puede haber un subárbol con nodos adicionales. La siguiente figura ilustra este concepto.

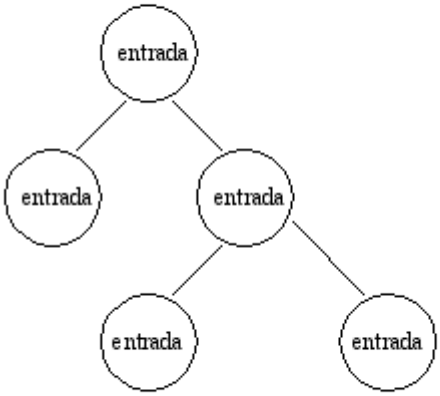

Figura 3.2 Jerarquía de datos

Cada hijo de un nodo particular se distingue del resto de ramas por su nombre distinguido relativo (RDN). El RDN consiste en el nombre de uno de los atributos de la entrada, seguido por el signo igual (=), y seguido por uno de los valores que puede tomar, por ejemplo uid=jmartin.

Podemos componer el nombre distinguido (DN) de una entrada tomando los RDN de cada nodo en el camino que lo llevan hasta el nodo raíz, separados por comas. Así en nuestro ejemplo tendríamos uid=jmartin,ou=people, o=universidad.

Por último señalar que los nombres de atributos no tienen en cuentan si se encuentran en mayúsculas o minúsculas. Tanto el espacio entre comas como los espacios que pueden aparecen junto al signo igual (=) se ignoran.

3.2.3 Modelo funcional

El modelo funcional es el mismo protocolo LDAP. El protocolo provee de mecanismos para acceder a los datos en el árbol de directorio. El acceso se implementa mediante operaciones de autenticación (bindings), de peticiones (búsqueda y lecturas) y actualizaciones.

3.2.4 Modelo de seguridad

El modelo de seguridad provee de un mecanismo que permite a los clientes probar su identidad (autenticación) y al servidor controlar el acceso a los datos (autorización). LDAPv3 ya trae algunos métodos de autenticación, cosa que las versiones anteriores no venían haciendo. Algunas características, como las listas de control de acceso (ACL) todavía no han sido estandarizadas.

3.3 Criterios de búsqueda

Como ya hemos señalado la principal característica de un directorio LDAP es su capacidad para devolver resultados muy rápidamente ante una búsqueda.

Cuando un cliente LDAP le pide a un servidor información, le pasa un pequeño conjunto de criterios de búsqueda que consiste en:

- qué parte del árbol de directorios será examinada
- qué contienen las entradas que estamos buscando
- qué información de las entradas coincidentes deben ser devueltas

El servidor tomará estos criterios, determinará si el cliente tiene los privilegios necesarios para realizar la búsqueda y devolverá los resultados que el cliente esté autorizado a recibir.

3.3.1 ¿Dónde buscar?

El criterio usado para determinar la porción de árbol de directorios donde aplicar la búsqueda consta de 2 factores.

- base
- alcance

La base que es simplemente el DN de una entrada en el directorio que será la entrada más alta asociada con la búsqueda.

Especificar la base de una búsqueda trata fundamentalmente de confinar la búsqueda en algunas ramas de árbol. Las búsquedas nunca trepan por el árbol. Se disponen desde la base hacia abajo.

El otro factor es el alcance. El servidor usa el alcance para determinar cuanto se tendrá que adentrar en el árbol para buscar entradas que coincide con el resto del criterio de búsqueda. El protocolo LDAP define 3 alcances:

- base
- un nivel
- subárbol

Alcance base

Evalúa solo una entrada individual. El protocolo no ofrece ninguna operación para leer una entrada del directorio por un nombre. Se puede leer una entrada directamente especificando una búsqueda con base igual al nombre distinguido que se quiere leer y con alcance base.

Alcance de un nivel

Permite buscar aquellas entradas que se encuentran directamente por debajo de la base de la búsqueda actual.

Este alcance no incluye la entrada que sirve de base para la búsqueda. Una búsqueda que incluye el atributo objectClass tendrá el efecto de listar todos las entradas que se encuentran debajo de la base de la búsqueda.

Este alcance es importante cuando se trata de aplicaciones gráficas y tenemos que desplegar las ramas del árbol nivel a nivel.

Alcance de subárbol

Este alcance llega desde la base hasta el final del subárbol especificado. Al contrario que el alcance de un nivel, si tiene en cuenta la entrada que sirve como base.

La mayoría de aplicaciones que no tienen requerimientos muy detallados del árbol de directorios, usan búsqueda con este alcance para encontrar entradas cuyo DN no conocen.

3.3.2 Filtros

El filtro de búsqueda te dice las condiciones que debe cumplir una entrada para que se a devuelta como resultado. Los filtros LDAP pueden ser tan simples como pedir que se cumpla un atributo en particular. Podemos crear filtros más complejos que incluyan diferentes conjuntos de operaciones. Para ello, tenemos 7 tipos básicos de filtros LDAP.

- de presencia
- de igualdad
- de subcadena
- mayor o igual que
- menor o igual que
- aproximado
- extensible

Es posible combinar estos filtros usando operadores AND, OR y NOT. El filtro o conjunto de filtros provistos por el cliente lo aplica el servidor a cada entrada, en la localización que marquen base y alcance. Si la aplicación del filtro es satisfactoria la entrada se devuelve, sino se ignora.

Filtro de presencia

Un filtro de presencia solo requiere que un atributo en particular tenga un valor concreto cualquiera, esto es, que exista.

Por ejemplo el filtro `sn=*` que devolvería todas las entras que tengan algún valor asociado al atributo sn.

Filtro de igualdad exacta

Limita las entradas devueltas a aquellas cuyos atributos y valores especificados coinciden con los que posee. Exige un valor concreto a diferencia del filtro de presencia, que sólo pide la existencia del atributo demandado. Ej: `sn=jimenez`.

La mayoría de los servidores de directorio ofrecen su mejor comportamiento en este tipo de búsquedas. De hecho es el filtro que se utiliza más comúnmente.

Filtro de subcadena

Muchas aplicaciones necesitan buscar información de la que sólo conocen una parte. Se utilizan estos filtros para buscar las ocurrencias de un patrón determinado y así obtener una selección de la cual poder escoger el deseado.

El filtro de subcadena se usa mucho y es más complejo de lo que pueda parecer. Veamos algunos ejemplos:

```
cn=*mar*
cn=mar*
cn=*mar*e*ez
```

El primer filtro buscará los caracteres `mar` en cualquier posición dentro del atributo cn. El segundo buscará todas las entradas que comiencen por `mar` y el último es un ejemplo de filtro no válido.

No debemos confundirnos y pensar que cualquier carácter que normalmente usamos en búsquedas en nuestro sistema operativo habitual, puede usarse en la composición de estos filtros. De hecho las búsquedas con subcadenas se limitan a los ejemplos propuestos y poco más.

Por último indicar que estos filtros son más lentos que los de presencia y los exactos. Sin embargo las búsquedas que implican subcadenas al principio o al final son más rápidas que aquellas que incluyen la ocurrencia en medio de la cadena.

Coincidencia ordenada

Se llama de coincidencia ordenada porque requieren que el servidor conozca el orden de los valores del atributo en todas las entradas del directorio. Veamos un ejemplo: sn>=smith

La operación de mayor o igual se designa por el identificador >=. Análogamente la operación de menor o igual usa el identificador <=.

Un valor se determina si es mayor o igual que otro de acuerdo con la sintaxis del atributo y las reglas de concordancia.

Filtros de aproximación

LDAP soporta un tipo de filtro para ocurrencias aproximadas. Éste se suele usar cuando a menudo en aplicaciones de páginas blancas cuando una persona intenta buscar un nombre pero no esta segura de cómo se escribe. Ej: sn~=jimenez.

Este último filtro pediría al servidor que buscase entradas en las que el atributo sn presente ocurrencias parecidas a jimenez (podría tratarse de gimenez, jimenes u otro caso parecido).

Para hacer esto se usan algoritmos especiales. Muchos servidores soportan el algoritmo Soundex. Aún sabiendo esto, es difícil para un desarrollador saber qué va a devolver el servidor.

Usualmente se utiliza para ofrecer una lista con las opciones que más se aproximen a las requeridas y que el usuario elija el que estaba buscando.

Filtros múltiples: operadores AND y OR

Podemos combinar varios filtros mediante estos operadores para conseguir resultados más precisos. Ej: (&(sn=jimenez) (cn=j*)).

Filtros negativos: el operador NOT

Utilizamos este filtro para eliminar valores que no queremos obtener en los resultados. Ej: (! (sn=jimenez)).

Debemos recordar que un filtro negativo no es lo mismo que la operación != que aparecen comúnmente en los lenguajes de programación. El filtro NOT da como resultado todas las entradas cuyos atributos no casen con el indicado, incluso aquellas que ni siquiera contengan ese atributo. En caso de que pidiésemos la entrada en las que el atributo sn exista deberíamos de crear un filtro complejo de esta forma (&(sn=*)(!(sn=jimenez))).

Búsquedas extendidas y reglas de coincidencia

Podemos usar las reglas de coincidencia para realizar búsquedas extendidas. Aparecen raramente pero cuando se usan correctamente ofrecen una búsqueda muy potente.

Pongamos un ejemplo de búsqueda extendida sn:1.2.3.4.5=jimenez . Este filtro indica que se use la regla de coincidencia 1.2.3.4.5 cuando devuelva entradas con el atributo sn igual a jimenez. Si el servidor define esa regla de forma que tenga en cuenta mayúsculas/minúsculas, la entrada sn=Jiménez no casará y por tanto no será devuelta.

3.3.3 ¿Qué se obtiene como resultado tras una búsqueda?

Se obtiene una lista de atributos. Si esta lista está vacía se devuelve todos los atributos no operacionales. Los atributos operacionales son típicamente aquellos que usa el servidor de forma interna y que raramente devuelve. Estos atributos varían según el fabricante pero suelen estar relacionados con las listas de control de acceso, marcas de tiempo y otro tipo de información. Los tipos de atributos que no existen, ni en el esquema del servidor ni en las entradas devueltas se ignoran sin mostrar ningún tipo de error.

Se devuelve siempre el nombre distinguido. El dn no es un atributo así que es imposible usarlo en una entrada ni eliminarlo de la lista de atributos devuelta.

Eliminar atributos de la lista devuelta es interesante debido al hecho de que muchas aplicaciones necesitan pocos atributos. Por ejemplo el caso de una aplicación que use un directorio para autenticación solo necesita dos atributos, identificación y clave. Si la entrada también contiene otro tipo de información como puede ser fotografía o teléfono es evidentemente ineficiente que el servidor devuelva la entrada entera.

3.4 Intercambio de información de directorio

A menudo es necesario compartir o usar la información que reside en el directorio fuera del propio servidor de directorio. Una forma de hacerlo sería escribir la información en archivos de texto siguiendo unas especificaciones establecidas. En los directorios LDAP estas especificaciones son las siguientes:

- LDIF
- DSML

3.4.1 Representación de la información fuera del directorio

Cuando recuperamos la información de un repositorio es usual que queramos almacenarla para uso futuro. Esto lo podemos hacer siguiendo un formato específico.

El primero y más usado es LDIF. Es muy simple e intuitivo, de hecho es el que hemos usado hasta ahora. La desventaja que tiene es que es muy específico de directorios LDAP.

```
dn: cn=pepe garcia,dc=mi casa,c=us
cn: pepe garcia
sn: garcia
```

El otro formato es DSML que se usa para representar la información en XML. Aunque este formato parece más complejo a primera vista, el hecho de estar basado en DML hace que existan más y mejores APIs para leer y escribir este tipo de ficheros.

En la versión 1 se podían representar el esquema y las entradas. En la 2 se añaden algunas operaciones de protocolo. Veamos un ejemplo.

```
<dsml:dsml xmlns:dsml="http://www.dsml.org/DSML">
<dsml:directory-entries>
```

```
<dsml:entry dn="cn=Sam Smith,dc=xyz,dc=com">
<dsml:objectclass>
<dsml:oc-value>person</dsml:oc-value>
<dsml:oc-value>organizationalPerson</dsml:oc-value>
</dsml:objectclass>
<dsml:attr name="cn">
<dsml:value>Sam Smith</dsml:value>
</dsml:attr>
<dsml:attr name="sn">
<dsml:value>Smith</dsml:value>
</dsml:attr>
</dsml:entry>
</dsml:directory-entries>
</dsml:dsml>
```

3.4.2 LDIF

Es un formato muy simple de representación de los datos del directorio como texto. Casi todos los servidores lo soportan y la mayoría de las APIs escriben y leen este formato.

Además de representar las entradas completas, LDIF, se usa para establecer cambios en entradas o esquemas.

LDIF solo permite texto imprimible así que usar el estándar Base64 para codificar los valores binarios.

3.4.2.1 Representación de las entradas

Veamos una entrada. Lo primero que aparece es esta línea.

```
dn: cn=jose martin,dc=ldap,dc=com
```

Como otras, está dividida en 2 partes por dos puntos (:). El lado izquierdo suele ser el tipo de atributo, solo hay una excepción y es esta primera línea, que es el nombre distinguido.

Una vez que tenemos nombrada la entrada se especifican todos los atributos del mismo modo que hemos especificado el DN. La parte a la izquierda es el nombre del atributo a la de la derecha, el valor que toma. Seguimos.

```
cn:jose martin
sn: martin
```

Los atributos que poseen más de un valor se expresan en varias líneas.

```
objectClass: person
objectClass: organizationalperson
```

Por último vamos a ver como se añade una entrada en este formato usando una herramienta de línea de comandos. Esta herramienta la incluyen la mayoría de servidores de directorio.

```
ldapmodify -D cn=admin -w manager -a
dn: cn=jose martin,dc=ldap,dc=com
cn:jose martin
sn: martin
objectClass: person
objectClass: organizationalperson
```

También podemos representar cambios. Para ello se añade el campo changetype, que puede tomar los valores add, y entonces equivaldría al ejemplo anterior y modify. A continuación se muestran algunos ejemplos donde se muestre su uso.

```
dn: cn=jose martin,dc=ldap,dc=com
changetype: add
cn:jose martin
sn: martin
objectClass: person
objectClass: organizationalperson
```

Este sería el caso que acabamos de comentar. También podemos guardar lo anterior en un fichero de texto y redirigirlo a ldapmodify de esta forma:

```
ldapmodify -D cn=admin -w manager < cambio.ldif
```

En el próximo ejemplo indicamos el tipo de cambio que vamos a realizar.

```
dn: cn=jose martin,dc=ldap,dc=com
changetype: modify
add: description
description: aqui va la descripcion
cn:jose martin
sn: martin
objectClass: person
objectClass: organizationalperson
```

Las operaciones de borrado y reemplazo también se ejecutan fácilmente.

```
dn: cn=jose martin,dc=ldap,dc=com
changetype: modify
delete: description
replace: cn
cn: pepe martin
```

3.4.2.2 Representación de esquemas

En cuanto a los esquemas se suele usar LDIF para hacer cambios en el esquema del servidor. Para realizar esto se usa un conjunto de atributos y sintaxis definidas en la RFC 2252. A continuación mostramos un ejemplo.

```
objectclasses: (2.5.6.6 NAME 'person' sup top
MUST (sn $ cn) MAY (password $ phonenumber $ description)
```

Veamos ahora una definición de un tipo de atributo.

```
attributetype: (2.5.4.35      NAME      `password`    EQUALITY
octectStringMatch  SYNTAX     1.3.6.1.4.1.1466.115.121.1.40
USAGE       userApplications)
```

Se está definiendo el atributo password con una sintaxis particular, usando la regla de concordancia `octectStringMatch`. Por último, para añadir el atributo al esquema, se crearía el siguiente fichero que se redirigiría al comando ldapmodify.

```
dn: cn=schema
changetype:modify
Add:attributetypes
Attributetypes: (1.2.3.4 NAME 'password' …
```

3.4.3 Lenguaje de marcas para servicios de directorio (DSML)

DSML es una porción de XML que se usa para representa la información de un directorio. Al estar basado en XML se puede usar DSML como formato genérico para los datos de directorios. Existen ocasiones a veces en las que la información de directorio tiene que salir más allá de los límites de la red o la compañía. DSML puede almacenar tanto esquemas como datos, haciendo mucho mas fácil el uso de las entradas exportadas fuera de su contexto original.

Otra razón para usar DSML es que las nuevas aplicaciones y servidores soportan XML y por tanto pueden usar tecnologías como las hojas de estilo XSL para generar fácilmente contenidos dinámicos.

DSML también soporta servicios web basados en XML y que usan estándares como SOAP, que permiten compartir fácilmente información de directorio.

Además es posible crear un servicio DSML que abstraiga el uso de LDAP y el acceso a datos de las capas de presentación y de lógica de negocio. Estándares como XSLT proveen la capacidad de transferir documentos DSML y convertir en flujo de aplicaciones basadas en web con un mínimo esfuerzo. Podemos ver un esquema en la siguiente figura.

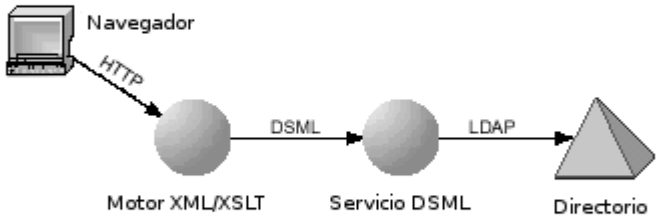

Figura 3.3 Acceso al directorio

Veamos un ejemplo de documento estructurado DSML que contiene una entrada de directorio para el usuario Juan Sánchez.

```
<dsml:dsml xmlns:dsml="http://www.dsml.org/DSML">
<dsml:directory-entries>
<dsml:entry dn="cn=Juan Sanchez,dc=xyz,dc=com">
<dsml:objectclass>
<dsml:oc-value>top</dsml:oc-value>
<dsml:oc-value>person</dsml:oc-value>
</dsml:objectclass>
<dsml:attr name="cn">
<dsml:value>Juan Sanchez</dsml:value>
</dsml:attr>
<dsml:attr name="sn">
<dsml:value>Smith</dsml:value>
</dsml:attr>
</dsml:entry>
</dsml:directory-entries>
</dsml:dsml>
```

En XML todo comienza con los signos < y > se conoce como elemento. Así que `<dsml:entry> y <dsml:directory-entries> son elementos.`

Algunos elementos tienen atributos. No debemos confundir los atributos XML con los atributos de LDAP. Los atributos de XML simplemente proveen de información sobre un elemento en particular. En el ejemplo anterior el elemento `<dsml:attr>` se usa para comenzar un atributo LDAP. Uno de los atributos de este elemento se llama name y contiene el nombre del atributo LDAP al que esta representando.

Los elementos se dan por acabados cuando aparece la etiqueta del nombre de elemento precedido de una barra, por ejemplo: `<\dsml:entry>`.

Los elementos se pueden anidar en otros. Aquellos que se encuentran entre las etiquetas de comienzo y fin de otro elemento se conocen como sus hijos.

```
<dsml:dsml ...>
      <dsml:directory-entries>
```

```
            <dsml:entry ...>
                  ... entry definition ...
            </dsml:entry>
      </dsml:directory-entries>
</dsml:dsml>
```

El elemento `</dsml:directory-entries>` indica que a continuacion se listaran las entradas. Cada entrada individual comienza con un `<dsml:entry ...>` y termina con un `</dsml:entry>`. Este elemento tiene un atributo XML que especifica el nombre distinguido de la entrada en cuestión.

Una vez dentro de las entradas su comportamiento difiere un poco respecto a LDIF. Los objectClass se tratan de forma diferente a los otros atributos. Se usa `<dsml:objectClass>` y su hijo `<dsml:oc-value>`.

Los atributos de cada entrada se listan tras la etiqueta `<dsml:attr>` y cada valor en su hijo `<dsml:value>`.

En cuanto a introducir cambios en las entradas DSMLv1 no permitía esto, estaba restringido a almacenar entradas completas. Esto se resuelve en la segunda versión en la que se permite representar hasta búsquedas y controles.

3.5 OpenLDAP

OpenLDAP es un proyecto colaborativo que trata de proveer una suite completa y robusta, de código abierto para LDAP.

El proyecto se gestiona por una comunidad mundial de voluntarios que usan Internet para comunicarse, planear y desarrollar el software LDAP y su documentación relacionada, al igual que la mayoría de proyectos libres.

El software para LDAP incluye servidores, clientes, SDKs y herramientas relacionadas. Cuenta con dos piezas fundamentales como son

- slapd
- slurpd

3.5.1 Slapd

Es el núcleo del proyecto. Es el servidor que recibe las peticiones LDAP y las gestiona. Vamos a enumerar de una forma sistemática sus características más importantes.

1) Implementa LDAPv3 completamente. Soporta LDAP sobre IPv4, IPv6 y IPC de UNIX.
2) Soporta autenticación fuerte gracias a SASL. La implementación de slapd de SASL usa el software CYRUS SASL que soporta bastantes mecanismos como DIGEST-MD5, EXTERNAL, y GSSAPI.
3) Provee privacidad e integridad mediante TLS (o SSL). La implementación de slapd de TLS usa OpenSSL .
4) Control por topología. Slapd puede ser configurado para restringir el acceso basándose en la información de la topología de la red. Esta característica usa los TCP *wrappers*.
5) Control de acceso. Slapd provee de unas potentes facilidades que permiten controlar el acceso a la información de las bases de datos. Se puede controlar el acceso a las entradas teniendo en cuenta la información LDAP de autorización, la dirección IP, el nombre de dominio u otros criterios. Soporta tanto el control de acceso estático como el dinámico.

6) Internacionalización. Slapd soporta Unicode y etiquetas de lenguaje.
7) Elección de las bases de datos de soporte (*backends*). Slapd viene con una gran variedad de *backends* que poder elegir. Se incluye BDB, LDBM, SHELL y PASSWD.
8) Múltiples instancias de bases de datos. Slapd puede ser configurado para servir a más de una base de datos al mismo tiempo. Esto significa que un único servidor slapd puede responder a muchas peticiones del árbol de directorios, usando la misma o distintas bases de datos de *backend*.
9) Módulos de API genéricos. Si se requiere aún más adaptación slapd permite escribir módulos externos muy fácilmente. Slapd consta de dos partes distintas: un *front end*, que maneja la comunicación con clientes LDAP; y módulos, que manejan tareas especificas como pueden ser las operaciones con bases de datos. Gracias a que estas dos partes se comunican mediante una interfaz en C muy bien definida, se pueden escribir módulos propios que extiendan las capacidades de slapd. También se proveen varios módulos programables de bases de datos que permiten presentar los datos a slapd usando lenguajes de programación muy populares. Estos son Perl, shell, SQL y TCL.
10) Hilos. Slapd usa los hilos con un rendimiento muy grande. Un único proceso multi-hilo de slapd puede hacer frente a numerosas peticiones con una pequeña reserva de hilos. Esto reduce la carga del sistema cuando se pretende un alto rendimiento.
11) Replicación. Slapd puede ser configurado para mantener copias de la información de directorio. Este concepto es vital para grandes entornos donde no se puede dejar toda la responsabilidad en un único servidor. La replicación en Slapd puede basarse en dos métodos: LDAP Sync y slurpd.
12) Proxy-caché. Slapd puede ser configurado para dar un servicio de proxy LDAP.
13) Slapd es altamente configurable gracias a un único fichero de configuración que permite cambiar prácticamente todos los parámetros, que además vienen por defecto con unos valores muy razonables.

3.5.2 Slurpd

Slurpd es un demonio, que ayuda a slapd a proveer el servicio de replicación. Es el responsable de distribuir los cambios de la base de datos principal de slapd (maestro) hasta las réplicas slapd. Libera a slapd de tener

que preocuparse de que algunas réplicas estén caídas o sean inalcanzables cuando aparece un cambio; slurpd gestiona automáticamente los reintentos. Además slapd y slurpd se comunican mediante un simple fichero de texto en el que aparecen los *logs* de los cambios.

3.6 Instalación y configuración

3.6.1 Obtener el software

El primer paso es obtener el software de la página oficial www.openldap.org. Existen dos versiones, una estable y otra más nueva pero que puede contener algún error. Actualmente tenemos las versiones 2.2.8 y 2.1.19, siendo esta última la estable. Se usará la estable.

3.6.2 Requisitos software

Estos requisitos dependerán de las funcionalidades que usemos, pero en general necesitaremos los siguientes paquetes:

1) Librerías TLS/SSL. Se usará OpenSSL que se puede descargar de www.openssl.org.
2) Un gestor de bases de datos que soporte facilidades de almacenamiento de tipo DBM. La elección es el paquete DB 4.1.24 (o posterior) de Berkeley y se puede conseguir en www.sleepycat.com
3) Librerías SASL. Usaremos las que provee la universidad de Carnegie Mellon. Se descarga de http://asg.web.cmu.edu/sasl/sasl-library.html.
4) Soporte para hilos POSIX, que los proveerá el sistema operativo en nuestro caso.

3.6.2.1 Instalación de Berkeley DB

Para los ejemplos que construyamos usaremos el sistema operativo Linux con el kernel 2.4.20 y lo primero que debemos hacer es instalar la base de datos de Berkeley que acabamos de señalar en los puntos anteriores, pues la que viene con el este kernel es anterior.

Debido a que vamos a instalar desde fuente, debemos desinstalar la versión anterior de Berkeley, ya que puede dar algunos problemas.

Seguimos los siguientes pasos:

1) Desinstalar la antigua versión
```
rpm -e db4 --nodeps
```
2) Descargar la versión adecuada. Se trata del archivo db-4.2.52.tar.gz.
3) Desempaquetar y descomprimir.
```
Gzip -d < db-4.2.52.tar.gz | tar xvf -
```
4) Compilar. Para ello debemos entrar en el directorio de construcción build_mix y allí ejecutamos la configuración:
```
cd db-4.2.52/build_unix
../dist/configure
make
```
5) Por último, instalar.
```
make install
```

3.6.2.2 Instalación de OpenSSL

OpenSSL es una herramienta que implementa los protocolos de seguridad SSL (v2/v3) y TLS (v1), además de proveer de una biblioteca criptográfica de uso general muy potente.

Actualmente se encuentra disponible la versión 0.9.7e, aunque cuando comenzamos el desarrollo del proyecto hicimos uso de la anterior, ya que todavía no existía ésta. Se trataba de la versión 0.9.7d y en un principio daba algunos fallos.

El principal error consistía en una bandera que se pasaba por defecto y que no era válida. Para solucionarlo hemos creado un parche, que aplicado al fichero makefile, corregirá este problema.

El parche consiste en las siguientes líneas:

```
64c64
> CFLAG= -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H
-DOPENSSL_NO_KRB5 -DL_ENDIAN -DTERMIO -O3 -fomit-frame-pointer -m486 -Wall
-DSHA1_ASM -DMD5_ASM -DRMD160_ASM
---
> CFLAG= -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H
-DOPENSSL_NO_KRB5 -DL_ENDIAN -DTERMIO -O3 -fomit-frame-pointer -march=i486
-Wall -DSHA1_ASM -DMD5_ASM -DRMD160_ASM
```

La instalación sigue los pasos típicos:

1) Obtención del software. En este caso debemos descargarnos el fichero openssl-0.9.7.d.tar.gz
2) Desempaquetar y descomprimir.
```
gunzip –c openssl–0.9.7d.tar.gz | tar xvf –
```
3) Compilación. Para ello debemos añadirle manualmente algunas rutas y variables de entorno y así asegurar un correcto funcionamiento.

```
cd openssl–0.9.7d
env cc=gcc LDFLAGS="–L/usr/local/lib –R/usr/local/lib"
PERL=/usr/bin/perl \ LD_RUN_PATH=/usr/local/lib ./Configure --
prefix=/usr/local --\ openssldir=/usr/local/openssl linux–elf
patch –p0 Makefile ../parche
make
```
4) Podemos comprobar que la compilación ha tenido éxito mediante el siguiente comando.
```
make test
```
5) Por último copiamos los ejecutables a su localización
```
make install
```

3.6.2.3 Instalación de Cyrus SASL

La instalación de Cyrus SASL vamos a realizarla debido a que es un requisito para OpenLDAP, aunque no vamos a aprovechar todas las características que nos ofrece. De hecho no usaremos autenticación vía SASL.

Cyrus SASL provee de una capa de seguridad, que da soporte a labores de autenticación para protocolos orientados a conexión.

Para usar esta especificación, el protocolo en cuestión, en este caso LDAP, debe incluir un comando para identificar y autenticar un usuario en un servidor. De forma opcional también permitiría negociar una capa de seguridad para las siguientes interacciones del protocolo.

En cuanto al comando que se utiliza, debe incluir un argumento obligatorio que indique el mecanismo de SASL que está usando. Los nombres de estos mecanismos deben estar registrados en IANA.

A continuación mostramos algunos de los mecanismos que aparecen en la librería estándar.

- ANONYMOUS
- CRAM - MD5
- PLAIN
- GSSAPI (junto con MIT Kerberos 5 o Heimdal Kerberos 5)
- DIGEST - MD5

El mecanismo que usaremos será PLAIN ya que es el único que permite usar una autenticación basada en PAM. En los otros casos, SASL utiliza un fichero, de tipo gdbm, para guardar los datos necesarios para la autenticación de cada usuario, en el lado del servidor.

Para su instalación seguimos los pasos, al igual que el resto de paquetes.

La instalación sigue los pasos típicos:

1) Obtención del software. En este caso debemos descargarnos el fichero cyrus-sasl-2.1.18.tar.gz
2) Desempaquetar y descomprimir.

```
gunzip -c cyrus-sasl-2.1.18.tar.gz | tar xvf -
```

3) Compilación. Para ello debemos añadirle manualmente algunas rutas y variables de entorno y así asegurar un correcto funcionamiento.

```
cd cyrus-sasl-2.1.18
./configure --without-des --with-openssl=/usr/local --disable-
checkapop \
--disable-cram --disable-otp --enable-anon=yes --enable-plain
\
--with-plugindir=/usr/local/lib/sasl2
\
--with-bdb-libdir=/usr/local/BerkeleyDB.4.2/lib/ --with-dblib=berkeley
make
```

4) Por último copiamos los ejecutables a su localización

```
make install
```

5) Debemos realizar un último paso para completar la instalación. Las librerías se encuentran en /usr/local/lib/sasl2, sin embargo, la mayoría de programas que hacen uso de ellas, las buscan en /usr/lib/sasl2. Existen diferentes formas de solucionar este problema. Se subsanará mediante un enlace simbólico a dicha ruta.

```
ln –s /usr/local/lib/sasl2 /usr/lib/sasl2
```

3.6.2.4 Instalación de OpenLDAP

Se hará de igual forma que hemos instalado la base de datos de Berkeley. Partimos de la versión estable que en los momentos de comenzar el proyecto era la 2.1.29 (actualmente es la 2.2.24).

1) Descarga del archivo openldap-2.1.29.tgz de la dirección adecuada, www.openldap.org.

2) Desempaquetar.

```
gunzip –c openldap-2.1.29.tgz | tar xvfB –
```

3) Configurar.

```
cd openldap-2.1.29.tgz
./configure
```

Llegados a este punto puede ocurrir que no encuentre la base de datos, como nos ocurre en nuestro caso. Así que debemos mostrarle donde está. Para ello:

```
env   CPPFLAGS="-I/usr/local/Berkeley/include"
LDFLAGS="-L/usr/local/Berkeley/lib" ./configure
```

Suponiendo que estos directorios son los indicados, en nuestro caso es así.

4) Construir el software. Esto conlleva dos partes:

a)Construir las dependencias.

```
make depend
```

b) Compilar el software

```
make
```

Ambos se deberían completar sin errores.

6) Se puede ejecutar un test que permita asegurarnos de la correcta construcción.

```
make test
```

7) Finalmente instalamos el software. Esto requiere permisos de superusuario.

```
su root –c 'make install'
```

3.6.3 Configuración del servidor

La configuración del servidor se lleva a cabo mediante un único fichero, que se encuentra en /usr/local/etc/openldap.conf, según las rutas que hemos indicado en la instalación.

Para mostrar la configuración básica de este fichero vamos usar un ejemplo práctico, pero a la vez muy sencillo y constructivo. Vamos a construir un directorio de páginas blancas para una compañía.

```
# Fichero de configuracion de OpenLDAP
# /usr/local/etc/openldap/slapd.conf

# Seccion global

## Incluimos el esquema minimo requerido.
include       /usr/local/etc/openldap/schema/core.schema

## Parametros informativos
loglevel      296
pidfile       /usr/local/var/slapd.pid
argsfile      /usr/local/var/slapd.args


## Ajustes de seguridad
password-hash         {SSHA}
```

```
#######################################################

## Define the beginning of example database.
databasebdb

## Definimos el sufijo raiz de trabajara el servidor.
suffix              "dc=miorganizacion,dc=org"

## Definimos un nombre para el usuario que tendra los privilegios.
rootdn              "cn=Manager,dc=miorganizacion,dc=org"

## Definimos el password usado con  rootdn. Ademas esta codificado mediante
un
## hash. Se trata del hash de la palabra "secret."
rootpw              {SSHA}2aksIaicAvwc+DhCrXUFlhgWsbBJPLxy

## Directorio que contiene los archivos de la base de datos
directory           /var/ldap/miorganizacion.org

## Archivos creados tendran permisos de lectura/escritura para el
propietario
mode                0600

## Indices que se van a mantener
index               objectClass         eq
index               cn                  pres,eq

## Parametros del backend. Habra 2000 entradas en cache
cachesize           2000

# Acceso mas simple, sin ajustar; lectura para todo el mundo
access to *
   by * read
```

El primer paso para implementar un directorio es determinar que información vamos a almacenar en el directorio. El contexto de nombres del servidor ha sido definido como `dc=miorganizacion,dc=org`.

Podemos almacenar la información de los empleados en una `organizational unit person:ou=people,dc=miorganizacion,dc=org`.

Hay distintas formas de identificar los datos que vamos a colocar en la entrada de un empleado. La información almacenada en la base de datos de recursos humanos nos podría servir de comienzo. Por supuesto puede ocurrir que no se desee incluir toda esa información en el directorio, ya sea porque no se vaya a utilizar o por cualquier otro motivo. Sólo debemos incluir los datos necesarios. Si hiciesen falta más siempre se pueden añadir

más tarde. Eliminando los datos innecesarios al comienzo nos libera de pensar que hacer con ellos o como proteger su acceso.

Una manera alternativa de comenzar con una base de datos existente sería determinar que atributos del empleado vamos a hacer accesibles y definir un esquema conforme a esto. Hacerlo de forma inversa también sería posible: se selecciona un esquema estándar y se usa los atributos ya definidos. Es aconsejable este último planteamiento ya que facilita los cambios entre servidores de distintos fabricantes. En un esquema hecho por nosotros podrían necesitar de un rediseño para adaptarlo a un nuevo fabricante (incluso para una nueva versión del mismo fabricante).

Para el directorio se usará el esquema `inetOrgPerson` definido en la RFC 2798. Este esquema es muy adecuado para las necesidades que se plantean.

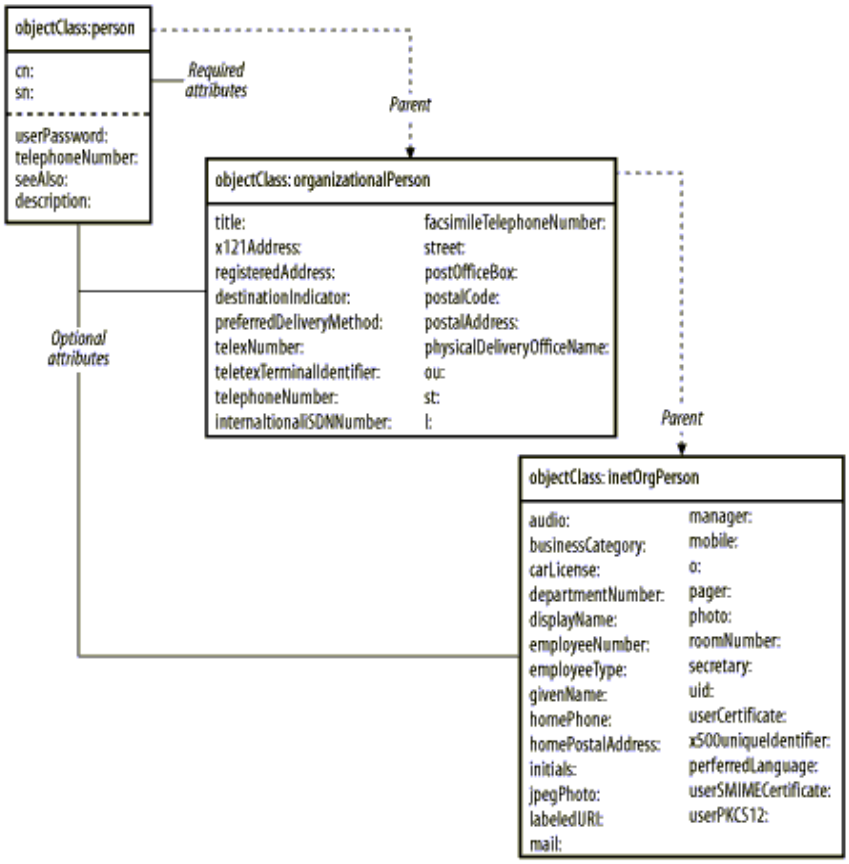

Figura 3.4 Jerarquía de la clase de objetos inetOrgPerson

En la figura 3.4 vemos como `inetOrgPerson` es un descendiente de `organizationaPerson`, que a su vez es hijo de la clase de objeto `person`.

Como podemos apreciar los únicos atributos requeridos del objeto inetOrgPerson son `cn` y `sn` que se heredan de la clase `person`.

El directorio en el que estamos trabajando usara el atributo `cn` como nombre distinguido relativo RDN para cada entrada. Recordemos que este RDN de cada entrada debe ser único entre las ramas hermanas (las que tienen la misma entrada padre). En grandes organizaciones puede ocurrir que dos personas tengan el mismo nombre y apellidos. En estos casos deberíamos usar un valor más específico que el `cn`.

Otra forma de reducir el número de colisiones sería rediseñar el árbol para reducir el número total de entradas que comparte el mismo padre. En otras palabras, agrupar a los empleados en algún tipo de unidad organizacional, tal como grupos o departamentos.

Para nuestro ejemplo tendremos suficiente con el espacio de nombres asignado sin miedo a ningún conflicto. En la figura 9 podemos ver el espacio de nombres de nuestro directorio, hasta ahora.

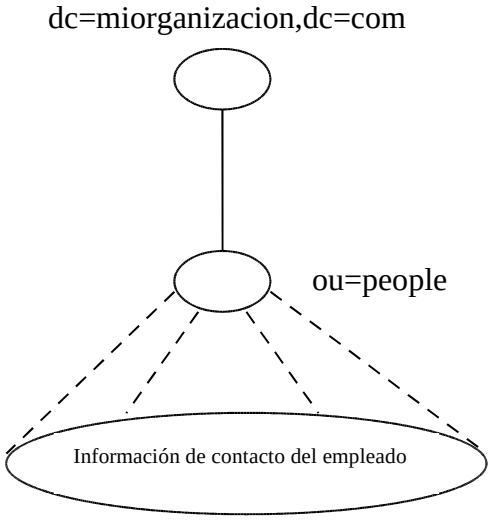

Figura 3.5. Espacio de nombres del directorio

Veamos ahora la entrada para un empleado que contenga los atributos necesarios para nuestro directorio. Debemos tener en cuenta los atributos requeridos.

```
## Entrada LDIF para la empleada "Rosa M. Sanchez"
dn: cn= Rosa M. Sanchez,ou=people,dc=miorganzacion,dc=org
objectClass: inetOrgPerson
cn: Rosa M. Sanchez
sn: Sanchez
mail: rsanchez@miorganizacion.org
mail: rosita@valinux.com
labeledURI: http://www.miorganizacion.org/
roomNumber: 1234 Dudley Hall
departmentNumber: Engineering
telephoneNumber: 222-555-2345
mobile: 222-555-1011
```

Se nos plantea una duda: ¿se debe mantener el árbol profundo y estrecho o poco profundo y ancho? Depende de dos factores.

El primero sería la necesidad de orden en el árbol de información. Un árbol más profundo significa que una entrada tiene que cumplir más requisitos para ser colocada en un lugar o en otro. Por ejemplo, en lugar de colocar a todos los empleados bajo el `ou=people`, podemos usar características como departamentos, descripción del trabajo o localización geográfica para un mejor agrupamiento. Sin embargo si estas características cambian muy frecuentemente lo único que se consigue es crear más trabajo (con RDNs más largos).

En el segundo factor se tendría en cuenta que la implementación del servidor del directorio favorezca un diseño sobre el otro. En el caso de OpenLDAP depende de las necesidades en cada momento. El factor que determina esto es el número de actualizaciones, o escrituras que se hagan al directorio. Para actualizar una entrada el servidor slapd echa un cerrojo al padre de la entrada pedida. Si ocurren muchas actualizaciones al mismo tiempo, el cerrojo estará echado mucho tiempo y esto hará más lentas las consultas ya que habrá procesos esperando a que finalice ese cerrojo.

Actualizar slapd.conf

Una vez que hemos seleccionado el esquema, el siguiente paso es modificar slapd.conf para que mantenga los tipos de atributos y clases de objetos seleccionados. Para soportar la clase de objeto `inetOrgPerson`, se debe incluir `inetorgperson.schema`, `core.shema` y `cosine.schema` en el fichero slapd.conf. Añadimos entonces estas líneas:

```
# /usr/local/etc/openldap/slapd.conf
## Incluimos el esquema minimo requerido.
include         /usr/local/etc/openldap/schema/core.schema

## Requeridos para poder usar el objeto inetOrgPerson.
include         /usr/local/etc/openldap/schema/cosine.schema
include         /usr/local/etc/openldap/schema/inetorgperson.schema
```

Para soportar búsquedas de empleados, deberíamos modificar un conjunto de índices para incluir una lista más completa de atributos. Además de crear un índice para el atributo cn, también lo haremos para sn y mail. También se añadirá un índice de igualdad (eq) y uno de subcadenas (sub) que soportará búsquedas del estilo “empleado cuyos apellidos comiencen con B”. Finalmente añadiremos un índice de igualdad para el atributo departmentNumber con la finalidad de permitir búsquedas de empleados a partir de un departamento conocido.

```
# Indices que se van a mantener
index         objectClass         eq
index         cn,sn,mail          eq,sub
index         departmentNumber    eq
```

Llegados a este punto debemos verificar que la localización del directorio existe y tiene los permisos adecuados.

```
mkdir -p /var/ldap/miorganizacion.org
chmod 700 /var/ldap/miorganizacion.org
```

Arrancando el servidor

Una vez que le hemos dado los últimos retoques al fichero de configuración, el siguiente paso es arrancar el demonio slapd ejecutando el siguiente comando:

```
/usr/local/libexec/slapd
```

Podemos usar el comando *ps* para verificar que slapd se esta ejecutando. En el sistema Linux que estamos usando tenemos la siguiente salida:

```
$ ps –ef | grep slapd
root   8235     1  0 12:37 ?   00:00:00 /usr/local/libexec/slapd
root   8241  8235  0 12:37 ?   00:00:00 /usr/local/libexec/slapd
root   8242  8241  0 12:37 ?   00:00:00 /usr/local/libexec/slapd
```

Para parar el servidor debemos usar una opción especial. Esto es debido a que debemos dar al demonio una oportunidad de salvar en disco los datos modificados. El mejor modo de pararlo es mandar una señal INT al proceso padre de slapd, como hacemos en el siguiente comando:

```
kill –INT 'cat /var/run/slapd.pid'
```

La localización del archivo .pid fue definida en el fichero de configuración.

Añadiendo las primeras entradas del directorio

En este momento tenemos dos opciones para añadir entradas. La primera sería *slapdadd* y otros comandos *slap** que son herramientas para mantener la base de datos. Permiten al administrador importar directamente entradas desde la base de datos y exportar el directorio completo, como un archivo LDIF. Trabajan directamente sobre la base de datos sin usar para nada el servidor slapd.

La segunda sería usar las herramientas que acompañan a la distribución de OpenLDAP, tales como ldapmodify, que pueden actualizar un directorio usando las operaciones en red que aparecen en LDAPv3.

Vamos a usar la primera. Pero antes vamos a crear un fichero que contenga las entradas en LDIF de los nodos de nivel más alto.

```
## Construimos el nodo raiz.
dn: dc=miorganizacion,dc=org
dc: miorganizacion
objectClass: dcObject
objectClass: organizationalUnit
ou: miorganizacion Dot Org

## Construimos el ou people.
dn: ou=people,dc= miorganizacion,dc=org
ou: people
objectClass: organizationalUnit
```

Suponiendo que guardamos las entradas anteriores en un archivo llamado /tmp/top.ldif podemos añadirlas al directorio ejecutando:

```
 slapadd -v -l /tmp/top.ldif
```

y obtenemos la siguiente salida, que indican que fueron añadidas satisfactoriamente.

```
added: "dc=plainjoe,dc=org" (00000001)
added: "ou=people,dc=plainjoe,dc=org" (00000002)
```

Por último, y para verificar que el contenido del directorio es correcto vamos a realizar una búsqueda en él. Esta búsqueda será del tipo “muéstralo todo”,

```
$ ldapsearch -x -b "dc=plainjoe,dc=org" "(objectclass=*)"
```

y obtenemos:

```
#
# filter: (objectclass=*)
# requesting: ALL
#
```

```
# plainjoe,dc=org
dn: dc=plainjoe,dc=org
dc: plainjoe.org
objectClass: dcObject
objectClass: organizationalUnit
ou: PlainJoe Dot Org


# people,dc=plainjoe,dc=org
dn: ou=people,dc=plainjoe,dc=org
ou: people
objectClass: organizationalUnit

# Search result
search: 2
result: 0 Success

# numResponses: 3
# numEntries: 2
```

Como último apunte indicar que esta búsqueda no es posible si el servidor slapd no está arrancado. Cuando añadimos las entradas con el comando slapdadd se permite que el servidor no esté arrancado, pero en el caso de las herramientas como ldapsearch necesita comunicarse con el servidor.

3.7 Seguridad en el directorio

Llegados a este punto todavía no se ha tocado el tema de la seguridad. En el directorio existe mucha información sensible y hasta ahora no hemos visto cómo hacer que los usuarios no accedan a lugares donde no deben acceder.

Es importante saber qué nivel de seguridad deseamos y qué información vamos a proteger. ¿Vamos a proteger sólo los passwords? ¿O también los nombres de usuarios? ¿Qué partes de la información personal puede consultarse desde fuera de la organización? ¿Y desde dentro?

Vamos a distinguir dos casos en cuanto a la seguridad se refiere. En primer lugar, la información de usuario que se transmite por la red debería ser accesible sólo por las partes interesadas, cliente y servidor, y nadie más.

Por otro lado, una vez que el cliente se ha autenticado, el servidor le va a permitir realizar una serie de operaciones, pero no todas. No va a poder, por ejemplo, borrar la información personal de otro usuario. Esta va a ser la otra parte a la que nos referimos: el control de acceso.

3.7.1 Encriptación de la información

Aunque parece importante mantener toda la información a buen recaudo, lejos de manos no autorizadas, resulta crítico cuando hablamos de passwords.

Para proteger passwords debemos tener en cuenta que vamos a acceder al directorio mediante un módulo PAM (concretamente con pam_ldap, como ya hemos visto anteriormente). Este módulo usa un bind simple para autenticarse contra el servidor LDAP y según está configurado (según la

configuración básica que usamos anteriormente), sería enviado como texto plano. Así que se debería evitar mandar información sensible en este paso.

LDAPv3 provee de dos mecanismos para proteger passwords. Uno es usar SASL que provee de métodos de autenticación como Kerberos 5 o DIGEST-MD5. Y aunque este mecanismo protege passwords, no protege otro tipo de información y por el momento no esta soportada por pam_ldap. Así que queda descartado para nuestro propósito.

La otra solución es negociar una capa de transporte seguro que protegerá tanto la información que aparezca en el bind como la que se intercambie con el directorio después.

Debido a estos motivos esta es la opción elegida; la otra ya estaba descartada.

Para la capa de transporte seguro tenemos dos posibilidades, usar SSL o TLS. Existen pocas diferencias entre ambos, de hecho, alguien que estuviese “escuchando” en la red, no podría distinguir entre el uso de uno o de otro. Sólo vería tráfico encriptado.

El hecho que hace que elijamos TLS en lugar de SSL es el ser una operación extendida dentro de la especificación de LDAP. Es estándar, mientras que SSL no.

El uso de SSL se realiza mediante LDAPS, que es otro protocolo y por tanto utiliza otro puerto (636). Hay que indicar al servidor explícitamente que escuche por este puerto para que acepte este tipo de conexiones.

OpenLDAP soporta ambos usos, aunque como ya hemos comentado, TLS está estandarizado además de ser muy sencillo de utilizar.

Para activar la operación extendida TLS debemos añadir las siguientes líneas en el fichero ldap.conf, de los clientes. Por supuesto el servidor debe estar configurado para aceptar tal opción, además de disponer de los certificados digitales que se precisan.

```
# /etc/ldap.conf
...
SSL start_tls
TLS_CACERT /usr/local/etc/openldap/cacert.pem
TLS_REQCERT     allow
TLS_CHECKPEER no
```

Antes de intercambiar información con el servidor, el cliente comprueba que su certificado se corresponde con el del servidor, y por tanto este es quien dice ser.

Para ello el cliente dispone de un certificado. Este, al igual que los existentes en el lado del servidor han sido creados usando utilidades provistas por el paquete openssl.

Este tema es bastante importante como para no mostrar como los hemos creado y qué medidas tomamos para protegerlas. Por ello hemos dedicado un apartado del apéndice para tal fin.

A continuación mostramos las líneas que deben añadirse al archivo de configuración del servidor para que soporte TLS.

```
# /usr/local/etc/openldap/slapd.conf
...
TLSCypherSuite HIGH:MEDIUM:+SSLv2
TLSCACertificateFile    /usr/local/etc/openldap/cacert.pem
TLSCertificateFile      /usr/local/etc/openldap/servercrt.pem
TLSCACertificateKeyFile /usr/local/etc/openldap/serverkey.pem
```

3.7.2 Listas de control de acceso (ACLs)

Por supuesto la encriptación del tráfico entre clientes y servidores no protege de accesos que sí estén autorizados. Usuarios que tienen permiso para cambiar su información personal, no deben poder hacerlo con la información de otros. Hay que establecer un sistema de permisos

Para conseguir esta protección debemos especificar controles de acceso. Las siguientes dos entradas de control de acceso (ACEs) del fichero slapd.conf, prohíben a los usuarios ver los passwords que pertenezcan a cuentas distintas a las que les pertenece.

```
## Los usuarios pueden cambiar sus passwords. Otros usuarios pueden
## intentar autenticarse pero no pueden leer el valor del password.
access to dn=".*,dc=miorganizacion,dc=org" attr=userPassword
     by self write
     by * auth

## Por defecto tenemos acceso de lectura.
access to dn=".*,dc=miorganizacion,dc=org"
      by * read
```

Merece la pena fijarnos más en detalle en lo que hacen estas ACEs. La primera permite a un usuario autenticado por el directorio, escribir su propio password (self write). El acceso en escritura implica acceso en lectura. A otros usuarios se les ha concedido sólo permiso para autenticarse (* auth). Esto no es lo mismo que acceso en lectura ya que los usuarios nunca pueden obtener el valor del password. El servidor compara el password enviado por el cliente en la petición de bind con el que tiene almacenado en el directorio. El password almacenado nunca abandona el directorio.

La segunda ACE concede a todos los usuarios (*) acceso en lectura a toda la información del directorio.

Debemos recordar que las ACEs siguen la regla de que la primera que se cumpla es la que se ejecuta. Por tanto debemos poner las más restrictivas al

principio. Si no hubiésemos tenido en cuenta esto cuando escribimos las dos entradas anteriores y hubiésemos cambiado el orden no obtendríamos el resultado requerido. Al evaluar la primera, que implicaba acceso en lectura para todos, se quedaría ahí, no continuaría hasta la siguiente y nunca se llevaría el proceso de autenticación a cabo.