Máster Interuniversitario de Seguridad de las Tecnologías de la Información y de las Comunicaciones (MISTIC)

Prácticas Profesionalizadoras / Trabajo Final de Máster

Metadirectorios bajo Node.js

Prueba de Evaluación Continua (PEC) – 4

Memoria Final

Alumno: Pere Llorens Vernet

Consultor: Cándido Rodríguez Montes

Empresa: PRiSE

Fecha: 10 de enero de 2014

Índice

1 Introducción

Hoy día las empresas tienen que almacenar una gran cantidad de información y datos en sus propias redes. Todos esos recursos de red deben ser debidamente almacenados, organizados y además gestionar el acceso de los usuarios a esos recursos. Por eso se utilizan servicios de directorios que permiten obtener todas estas características y crea una capa de abstracción entre los usuarios y todos esos recursos de la empresa que queremos compartir.

Utilizar un sistema de directorio es muy práctico y eficaz, pero existen de diferentes tipos y características. Esto en principio no lleva ningún problema ya que cada empresa es independiente de la otra y actúan separadas, pero que pasa cuando dos o más empresas quieren unir sus sistemas de directorios y compartir sus recursos.

Esto puede ser un problema si las diferentes partes que quieren compartir recursos utilizan sistemas diferentes o que son incompatibles. Por lo tanto se tiene que encontrar la forma más eficaz de compartir estos recursos sin suponer un peligro para los recursos y para su correcto acceso.

En este proyecto se pretende crear un servicio de directorio unificado a partir de dos sistemas que no son compatibles entre sí, LDAP y XML, de forma que los recursos de los dos sistemas se puedan compartir y acceder desde un solo sistema.

1.1 Objetivos

El objetivo del proyecto es desarrollar un directorio virtual básico bajo tecnología Node.js utilizando para ello la librería ldapjs. Bajo esta tecnología podremos desarrollar servidores y clientes LDAP utilizando para ello el lenguaje de programación JavaScript. Node.js ha experimentado un auge en los dos últimos años debido a los buenos datos de rendimiento que ofrece, por lo que podría ofrecer un marco de trabajo ideal para servicios de directorios.

El planteamiento inicial de este proyecto es que utilizando un sistema de directorio como un directorio virtual o un metadirectorio se debe unir un sistema LDAP y un sistema de ficheros (XML).

Por lo tanto los principales objetivos a obtener al finalizar este proyecto son:

- Utilizar un directorio virtual para unir dos sistemas diferentes.
- Este directorio virtual debe realizar operaciones de búsqueda en esos recursos.
- Se debe crear un conector de LDAP para integrar ese tipo de datos en el directorio virtual.
- Se debe crear un conector de fichero de texto (XML) para integrar ese tipo de datos en el directorio virtual.
- Crear una función de autorización para determinar que usuarios pueden acceder en determinados recursos.

2 Estudio teórico

En esta apartado se realizará un estudio teórico sobre dos sistemas de servicios de directorio, como son los metadirectorios y los directorios virtuales. Se explicaran sus características y sus principales ventajas e inconvenientes. Por lo tanto el objetivo es saber en qué consisten estas tecnologías y cuando se debe utilizar una u otra.

Antes de empezar a profundizar en los aspectos de los metadirectorios y los directorios virtuales, se explicara de una forma básica qué es un servicio de directorio y LDAP de esta manera se pretende tener una base para la cual iniciar posteriormente un estudio más profundo de los dos tipos de servicio de directorio que se pretende estudiar.

2.1 Servicio de Directorio

Un servicio de directorio es una aplicación que almacena, organiza y proporciona acceso a la información de un directorio, sobre los usuarios de una red de ordenadores, sobre recursos de red, y permite a los administradores gestionar el acceso de usuarios y tener una política unificada para los accesos a los recursos sobre dicha red.

Un directorio es una base de datos optimizada para lectura, navegación y búsqueda. Los directorios tienden a contener información descriptiva basada en atributos y tienen capacidades de filtrado muy sofisticada. Esto permite a los directorios dar una rápida respuesta a grandes volúmenes de búsquedas. Estos tienen la capacidad de replicar la información para incrementar la disponibilidad y la fiabilidad, al mismo tiempo que reducen los tiempos de respuesta.

El protocolo más extendido para hacer uso de un servicio de directorio es el protocolo LDAP (Lightweight Directory Access Protocol) que se encuentra definido en el RFC 2251. Este protocolo nos permite tener acceso a un servicio de directorio para buscar diversa información en un entorno de red. Además LDAP en si mismo también se la considera una base de datos a la que se le pueden hacer consultas.

Por lo tanto las principales características de los servicios de directorios son:

- Son una base de datos estructurada que almacena datos de forma estandarizada.
- La mayoría son accesibles mediante el protocolo LDAP.
- Implementa mecanismos de control de acceso y se utilizan también como servidores de autenticación o de gestión centralizada de políticas.
- Su principal función es el almacenamiento de datos y su uso para otros fines es muy limitado.

Para el desarrollo de un directorio virtual se puede enfocar de formas diferentes, las cuales nos permite trabajar con los directorios de distintas formas. Un método es utilizar un servicio de metadirectorios, el cual le permite sincronizar la información entre los directorios. Otro enfoque sería utilizar un directorio virtual, que brinda una visión única del directorio para la aplicación a utilizar, mientras se despliega la información de otros directorios sin necesidad de sincronización.

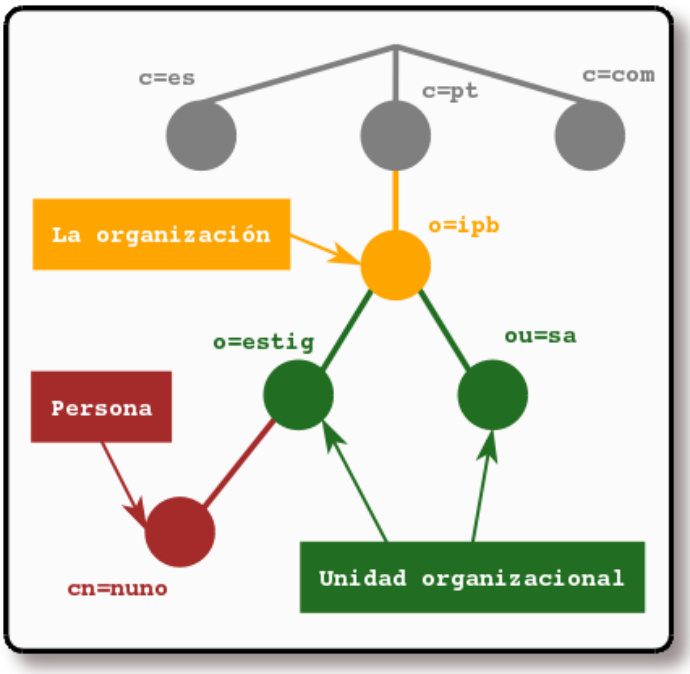
2.2 **LDAP**

LDAP (Lightweight Directory Access Protocol) es un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red. Se considera también un tipo de base de datos a la que pueden realizarse consultas. A manera de síntesis, LDAP es un protocolo de acceso unificado a un conjunto de información sobre una red.

Está basado en el estándar X.500, pero significativamente más simple y más adaptado para satisfacer las necesidades del usuario. A diferencia de X.500 LDAP soporta TCP/IP, que es necesario para el acceso a Internet. Se encuentra definido en el RFC 2251.

En LDAP, las entradas están organizadas en una estructura jerárquica en árbol. Esta estructura puede reflejar los límites geográficos, organizacionales o también basándose en los nombres de dominio de internet. La estructura basándose en el dominio de internet se está convirtiendo en la más popular.

Un ejemplo de estructura de dominio. Las entradas que representan los dominios aparecen en la parte superior del árbol. Debajo de ellos, están las entradas que representan las organizaciones. Debajo de estás, pueden estar las entradas que representan las unidades organizacionales, empleados, impresoras, documentos o todo aquello que pueda imaginarse.



LDAP define operaciones para interrogar y actualizar el directorio. Provee operaciones para añadir y borrar entradas del directorio, modificar una entrada existente y cambiar el nombre de una entrada. La mayor parte del tiempo, sin embargo, LDAP se utiliza para buscar información almacenada en el directorio. Las operaciones de búsqueda de LDAP permiten buscar entradas que concuerdan con algún criterio especificado por un filtro de búsqueda. La información puede ser solicitada desde cada entrada que concuerda con dicho criterio.

2.3 Metadirectorio

Se explicaran las principales características de los metadirectorios, sus principales ventajas e inconvenientes, como funciona y en que situaciones es mejor usar este servicio de directorio.

2.3.1 Introducción

Los metadirectorios son servicios de directorio que tienen la capacidad de recolectar y almacenar información de varios y diversos servidores de directorios. En algunos casos los metadirectorios tienen la capacidad de integrar información disponible en bases de datos.

La información de las diversas fuentes es agregada para proveer una vista única de dichos datos. Cabe anotar que en la consolidación de los datos, la información puede ser transformada según las reglas que se tengan definidas en el metadirectorio para los procesos de recolección e importación de los datos.

Por lo tanto los metadirectorios permiten integrar en un único repositorio la información existente de diversas fuentes, de tal modo que se puedan realizar búsquedas de manera centralizada y no tener que hacer varias búsquedas en varios servidores de directorios.

2.3.2 Descripción del funcionamiento

Los mecanimos de metadirectorio se encargan de conectar diferentes fuentes de datos de manera que se mantengan sincronizadas. Los flujos de información que soporta un sistema de metadirectorio son multi-direccionales e implican la necesidad de procedimientos de encaminamiento de datos y de niveles de adaptación a las diferentes fuentes participantes que, en general, deben soportar tanto lectura como escritura.

Un servidor de metadirectorio, recopila datos de una variedad de fuentes de datos, a través de conectores que se pueden configurar para cada fuente, en un repositorio central. Una vez que tenga consolidadas los datos de identidad, con un índice de un atributo único, se pueden insertar todos o algunos de los datos asociados a muchos repositorios y diversas aplicaciones simultáneamente. Estas aplicaciones por lo tanto, tienen un conjunto de base coherente de datos de identidad, independientemente de si se habla directamente el uno al otro o no.

Por lo tanto, un aspecto importante de los metadirectorios, es que estos funcionan por medio de agentes que se encargan de recolectar los datos en los diferentes servidores de directorios y enviarlos al metadirectorio para su consolidación. De esta forma permite sincronizar la información entre los directorios.

- Es un servicio de sincronización de datos entre diferentes servicios de directorio.
- El motor de sincronización copia los datos de una instancia de servidor de directorio a otro, cuando se detecta su modificación.
- Algunos metadirectorios pueden sincronizar datos con tablas de bases de datos relacionales y archivos, pero sus características de transformación de datos suelen ser muy limitadas.
- También permiten una fusión de los datos de varias fuentes similares para presentar una visión consolidada.

2.3.3 **Ventajas e inconvenientes**

Ventajas:

- Existe un único punto de referencia que provee un mayor nivel de abstracción para las aplicaciones que requieren información dispersa por toda la organización en diferentes fuentes de información.
- Existe un único punto de administración, lo que reduce la carga administrativa evitando tener que realizar múltiples accesos a diferentes servidores de directorios.
- Se puede eliminar la información redundante al poseer un repositorio unificado de datos.

Inconvenientes:

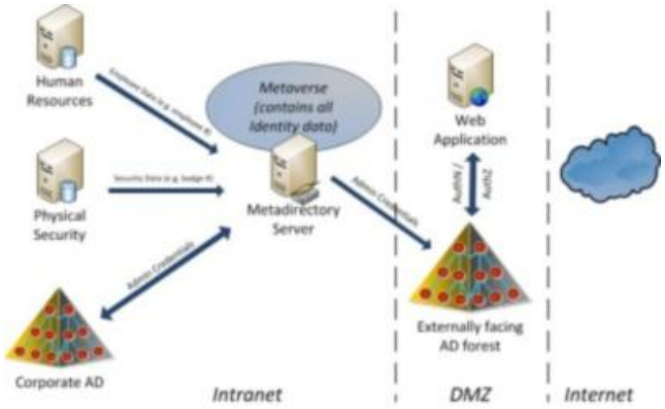
- Una estructura de servidor de metadirectorio, son poderosos, pero son caros para comprar, son caros de diseñar e implementar, y tienen un montón de piezas móviles para mantener una vez desplegado.
- Hay un tiempo de latencia entre el momento en una fuente de datos hace una actualización y cuando todos los sistemas intermedios tienen esa actualización.
- Puede haber un problema de escalabilidad, un servidor de metadirectorio mueve una gran cantidad de datos de identidad en torno a sus distintos puntos finales, incluso si los puntos finales no lo necesitan de inmediato

2.3.4 **Ejemplo de uso**

Los metadirectorios son muy utilizados en entornos donde se necesita sincronizar información entre diferentes fuentes de datos, esto es muy solicitado para centralizar la gestión, las políticas de acceso o para aplicar single sing-on (SSO).

Un uso muy utilizado es para la gestión de los usuarios en grandes redes de diferentes empresas y que usan diferentes tipos de bases de datos. Los metadirectorios simplifican el tratamiento de los datos, esto es así, ya que el cambio de cualquier dato de un usuario no es necesario que se haga en cada uno de los sistemas de las diferentes empresas, sino que este cambio de propaga y es replicado de manera transparente.

Por lo tanto un uso común es cuando se necesita replicar información común entre diferentes fuentes de información.



2.4 Directorio virtual

Se explicaran las principales características de los directorios virtuales, sus principales ventajas e inconvenientes, como funciona y en que situaciones es mejor usar este servicio de directorio.

2.4.1 Introducción

Los directorios virtuales son parecidos a los metadirectorios, estos se encargan de crear una vista unificada de la información que procede de diferentes fuentes dentro de la organización. A diferencia de ellos un directorio virtual se encarga de crear una vista única por medio de sentencias ejecutadas en tiempo real y que son construidas por medio del mapeo de campos en un esquema virtual.

En este sentido los directorios virtuales poseen un mayor nivel de flexibilidad dado que no tienen que realizar el almacenamiento de los datos en un repositorio central y con esto se elimina la necesidad de tener que implementar procesos de sincronización y replicación de datos entre servicios de directorios.

2.4.2 Descripción del funcionamiento

Un servicio de directorio virtual permite agregar on-line diferentes fuentes heterogéneas de datos, ofreciendo una visión coherente de estas fuentes a través de un servicio de directorio. El servidor de directorio virtual se encarga de recoger las peticiones de los clientes y de transformarlas y reenviarlas a las fuentes de datos, consolidando y traduciendo sus respuestas de acuerdo al esquema de acceso para el que se le haya configurado.

Los directorios virtuales se encargan de crear una vista unificada de los diferentes directorios que contienen la información. La principal característica es que a diferencia de los metadirectorios, un directorio virtual no hace uso de agentes para la recolección de los datos, sino que hace las búsquedas de información solo cuando son solicitadas por algún usuario y en tiempo real.

Los directorios virtuales son más flexibles y escalables al momento de integrar la información almacenada en las diferentes fuentes de información. Por lo general, se puede incluir repositorios tales como servicios de directorios, bases de datos y otras fuentes integradas por medio de tecnologías de integración tales como Web Services.

Las principales características que tiene un directorio virtual son:

- Agrega datos de identidad a través de las diferentes fuentes para crear un único punto de acceso.
- Crea una alta disponibilidad de la información para los datos autorizados.
- Actúa como cortafuegos de identidad para la prevención de ataques de denegación de servicio en los almacenes de datos primarios a través de una capa virtual adicional.
- Da soporte a un espacio de nombre de búsqueda común para la autenticación centralizada.
- Presenta una vista virtual unificada de la información de usuario almacenada en múltiples sistemas.
- Delegar la autenticación a fuentes de back-end a través de medios de seguridad específicos de origen.

- Virtualiza las fuentes de datos para apoyar la migración de los almacenes de los datos legados, sin modificar las aplicaciones que dependen de ellos.
- Enriquecer identidades con atributos extraídos de múltiples almacenes de datos, basado en la relación entre las entradas de usuario.
- Es un traductor de protocolos que permite a las aplicaciones el acceso a los datos (habitualmente residentes en diferentes repositorios o BBDD relacionales) a través de un único protocolo y mediante una vista consolidada.
- Algunos sistemas de directorio virtual permiten la transformación en caché.

2.4.3 Ventajas e inconvenientes

Ventajas:

- Permite una implementación más rápida, porque los usuarios no tienen que añadir y sincronizar las fuentes de datos para aplicaciones específicas adicionales.
- Aprovecha la infraestructura existente y la identidad de las inversiones en seguridad para implementar nuevos servicios.
- Ofrece una alta disponibilidad de las fuentes de datos.
- Proporciona panoramas específicos de la aplicación de los datos de identidad que pueden ayudar a evitar la necesidad de desarrollar un esquema de la empresa principal.
- Permite una visión única de los datos de identidad sin violar los reglamentos internos o externos que rigen los datos de identidad.
- Actúa como cortafuegos de identidad mediante la prevención de ataques de denegación de servicio en los almacenes de datos primaria y proporciona mayor seguridad en el acceso a datos confidenciales.
- Puede reflejar los cambios realizados de fuentes autorizadas, en tiempo real.
- Presenta una vista virtual unificada de la información de usuario de múltiples sistemas, de manera que parece residir en un único sistema.
- Puede proteger todos los lugares de almacenamiento de back-end con una sola política de seguridad.

Inconvenientes:

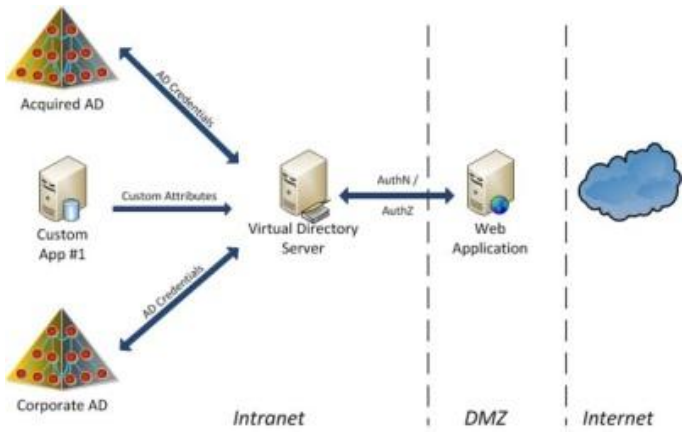
- El directorio virtual clásico basado en proxy no puede modificar las estructuras de datos subyacentes o crear nuevos puntos de vista basados en las relaciones de los datos a través de múltiples sistemas. Así que si una aplicación requiere una estructura diferente, tal como una lista lineal de las identidades, o una jerarquía más profunda para la administración delegada, un directorio virtual es limitada.
- Muchos directorios virtuales no se pueden correlacionar los mismos usuarios a través de múltiples fuentes diversas en el caso de usuarios duplicados.
- Los directorios virtuales no cuentan con tecnologías avanzadas de almacenamiento en caché, no se pueden escalar para entornos heterogéneos y de gran volumen.

2.4.4 Ejemplo de uso

En uno de los casos donde se debería utilizar un directorio virtual en contra de un metadirectorio es cuando debemos unir varios directorios de datos entre sí, que por ejemplo provienen de diferentes empresas.

Hacerlo con un directorio virtual nos proporciona una forma de poder compartir las diferentes infraestructuras de directorios de manera más ágil, sencilla y a menor coste que

un metadirectorio. En este caso, cada directorio de cada una de las empresas puede estar utilizando diferentes protocolos incompatibles entre sí o utilizar estructuras diferentes. Utilizar un directorio virtual nos proporcionaría un único punto de acceso de todos los directorios para todo el grupo de empresas. En contra del metadirectorio que deberíamos fusionar todos los directorios.



2.5 **Conclusiones**

En los metadirectorios, toda la información y datos de todos los directorios es agregada en un repositorio central, así se tiene toda la información centralizada y controlada y también se tiene una política centralizada, todas las consultas se hacen en ese repositorio central. Cuando nueva información se crea o es modificada en un directorio es actualizada en el repositorio central.

En directorio virtual, el usuario tiene la visión de un solo repositorio central (como en metadirectorio), pero realmente no es así. Se crea un repositorio que realmente está vacío, pero de forma "virtual" toda esta información está en ese repositorio, ya que cuando se hace una consulta en ese repositorio vacio, la información es buscada en tiempo real en el directorio donde se encuentra realmente y es ofrecida al usuario. Todos los directorios están enlazados ha este repositorio vacio, por este motivo se pueden acceder a través de él.

El motivo que nos puede decantar a utilizar un método u otro pueden ser varios.

Los directorios virtuales nos proporcionan una infraestructura a un bajo coste y también nos permite unir diferentes directorios con distintos protocolos con rapidez y sencillez, aunque realmente toda la información no está realmente unida.

En cambio los metadirectorios tienen un coste más elevado y más dificultad en juntar varios directorios distintos entre sí. Pero al estar realmente unidas, toda la información es actualizada en toda la estructura cuando hay un cambio, también se evita tener información duplicada y además es más fácil tener una sola política de acceso a la información solicitada.

En el caso de este proyecto se utilizara un directorio virtual. Los motivos son que no hace falta tener una gran infraestructura física para funcionar, es el método más rápido y simple y la curva de aprendizaje para utilizarla es más baja. Todo esto nos permite realizar este sistema de directorio en el periodo de tiempo disponible y en ese caso sin ningún coste adicional.

3 Especificaciones

En este apartado se explicaran las características del directorio virtual que se quiere realizar, los tipos de datos que se quieren integrar, la tecnología que se utilizara, además de todo lo necesario para que funcione.

3.1 Tipos de datos a integrar

En este directorio virtual integraremos dos tipos de datos diferentes, un sistema LDAP y un sistema de ficheros XML.

El sistema LDAP, como he explicado en el apartado Estudio teórico, es un sistema que se organiza en un árbol de directorios y sus características están especificadas en el RFC 2251.

El sistema XML, es un sistema para el intercambio de información estructurada entre distintas plataformas, por lo tanto puede contener información, que debidamente interpretada nos permitirá leer su estructura de datos y crear nuestro sistema de directorio.

En este caso, la información que contiene el fichero XML está estructurada de una forma concreta para que el directorio virtual que se quiere realizar pueda interpretar correctamente su estructura de datos. Por lo tanto la información debe estar estructurada como en el siguiente ejemplo.

```
<directory>
      <obj>
            <dn>dc=otroejemplo,dc=es</dn>
            <attributes>
                  <objectclass>dcObject</objectclass>
                  <objectclass>organization</objectclass>
                  <objectclass>top</objectclass>
                  <o>otroejemplo</o>
                  <dc>otroejemplo</dc>
            </attributes>
      </obj>
      <obj>
            <dn>dc=people,dc=otroejemplo,dc=es</dn>
            <attributes>
                  <objectclass>dcObject</objectclass>
                  <objectclass>organization</objectclass>
                  <objectclass>top</objectclass>
                  <o>people</o>
                  <dc>people</dc>
            </attributes>
      </obj>
</directory>
```

3.2 Estructura del directorio

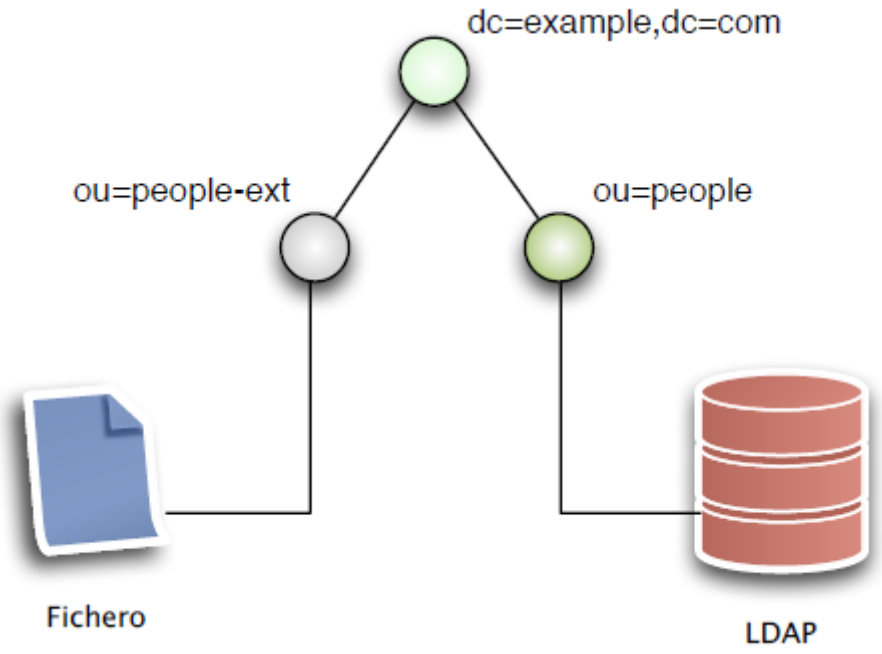
Para poder integrar estos dos tipos de datos mencionados anteriormente de forma organizada en nuestro directorio virtual, crearemos una estructura organizada en árbol en que cada rama hija representa el sistema que queremos integrar.

En este caso se creara el directorio principal que albergará los nuevos sistemas de datos a integrar, representado con el nombre *dc=example,dc=com*. Este directorio se

ramificara en dos ramas hijas, cada una de ellas contendrán los dos tipos de datos que queremos conectar. Sus nombres serán *ou=people-ext* y *ou=people*.

El primer directorio hijo contendrá el sistema de ficheros, el segundo directorio hijo contendrá el sistema de datos LDAP. Cuando nuestro directorio virtual tenga integrado los dos tipos de datos, quedará como resultado que cada rama hija anteriormente mencionadas integraran todo el árbol de directorios que contienen sus respectivos sistemas de datos.

Por lo tanto nuestro directorio virtual seguirá el siguiente esquema:



3.3 Credenciales de usuarios

Para la creación de este directorio virtual, también debemos tener en cuenta que usuarios tienen acceso a los directorios. Por ese motivo se debe crear una función de autorización que indique los usuarios que tienen acceso al directorio virtual y sobre que ramas puede actuar.

Por lo tanto se deben crear unos usuarios que puedan actuar sobre todo el directorio (administrador), o solo en parte de ella.

3.4 Requisitos de funcionamiento

Para la creación de este directorio virtual utilizaré el framework *Node.js*, que utiliza el lenguaje de programación *Javascript*. También para realizar nuestro directorio virtual se necesita instalar diferentes módulos en este framework. Los módulos de *Node.js* que yo utilizaré son el *xml2js*, *fs* y *ldapjs*.

4 Diseño e implementación

En este apartado se explicaran las decisiones de diseño del directorio virtual, como también una explicación del código que lo compone. Por lo tanto aquí se mostrarán las partes del código más importantes para el desarrollo del directorio virtual y una explicación de la misma.

4.1 Configuración inicial del servidor LDAP

4.1.1 Importación de recursos

El primer paso antes de comenzar a escribir el código del directorio virtual tenemos que importar los módulos de *Node.js* necesarios para obtener las funciones necsarias para construir todo el código. En este caso importamos los módulos *ldapjs*, *xml2js* y *fs*.

```
var ldap = require('ldapjs');
var xml2js = require('xml2js');
var fs = require('fs');
```

4.1.2 Configuración servidor LDAP

Nuestro directorio virtual funcionara en un servidor LDAP, por lo tanto se tiene que configurar los parámetros necesarios para que funcione y preparar su puesta en marcha.

Como se ha mostrado anteriormente un servidor en LDAP tiene forma de árbol y la parte superior corresponde al nombre de dominio que le queremos dar. Otra variable a configurar es el puerto donde queremos que se ejecute el directorio virtual, ahí es donde irán las peticiones del servidor LDAP. Estas son las dos variables a configurar.

Además hay dos variables más, pero son internas para el funcionamiento del directorio. La primera es la creación de una variable global para la creación del servidor LDAP y así utilizarla donde sea necesario. La otra variable es una variable contenedora de todo el árbol del directorio virtual, por lo tanto contendrá toda la información de los dos tipos de datos que queremos integrar.

```
var SUFFIX = 'dc=example, dc=com';
var db = {};
var server = ldap.createServer();
var portServer = 1389;
```

4.1.3 Configuración datos XML

Se debe configurar de donde se importaran los datos del fichero XML que se quieren integrar en el directorio virtual, por lo tanto la única configuración que se debe realizar es indicar la ubicación de dicho fichero.

```
var pathFile = '/home/user/desktop/ldap.xml';
```

4.1.4 Configuración datos LDAP

Para configurar la integración de los datos LDAP, se deben indicar diferentes parámetros del servidor LDAP cliente donde residen los datos. Los datos que se deben indicar son la url y el puerto donde reside dicho servidor, el nombre del directorio superior de su árbol (que corresponde normalmente en el nombre de dominio) y un usuario con la contraseña para autenticarnos en dicho servidor.

Otras variables que son internas para el funcionamiento del directorio virtual son parámetros de configuración del tiempo de conexión para indicar el tiempo que esperara nuestro servidor para la correcta conexión al servidor cliente. También se indica el tipo de búsqueda que se hará, en este caso se devolverá todo el contenido inferior del árbol de la rama indicada.

```
var client = ldap.createClient({
    url: 'ldap://127.0.0.1:1390',
    timeout: 5000,
    connectTimeout: 10000
});
var optsClient = {
    scope: 'sub',
}
var dnLDAPServer = 'dc=otroejemplo,dc=com';
var usernameClient = 'cn=root';
var passwordClient = 'secret';
```

4.1.5 **Variables globales**

Estas dos variables son creadas para uso interno del código, se utilizaran para indicar si las operaciones de importación y búsqueda en los correspondientes datos integrados ha finalizado.

```
var endLDAP = false;
var endFile = false;
```

4.2 **Operaciones del servidor LDAP**

Aquí se explicaran las principales operaciones que realiza el servidor LDAP para crear el directorio virtual que necesitamos y tenga las funcionalidades necesarias.

4.2.1 **Inicializar servidor**

En primer lugar antes de crear cualquier otra operación, se tiene que poner en marcha el servidor LDAP. Por lo tanto con los datos que se han configurado previamente se ejecuta la correspondiente función que pone nuestro servicio en escucha.

```
server.listen(portServer, function() {
    console.log('LDAP server listening at %s', server.url);
});
```

4.2.2 **Operación de autenticación**

Para tener acceso al servidor LDAP, se debe tener las credenciales correctas. Sin estas credenciales no se puede tener acceso a ninguna de las operaciones que puede realizar este servidor y por lo tanto no se tiene acceso a los datos que se han integrado y ha ningún directorio del árbol.

En este servidor se han creado tres usuarios diferentes con su correspondiente contraseña. Si no se dan las credenciales correctas se rechaza la conexión.

En este servidor se han creado tres usuarios *cn=root, dc=example, dc=com, cn=rootLDAP, dc=example, dc=com* y *cn=rootFile, dc=example, dc=com*. Cada usuario con su correspondiente contraseña, en la función de autorización se especificará sobre que ramas pueden realizar operaciones.

```
server.bind(SUFFIX, function(req, res, next) {
    console.log('== LDAP.bind ==');
    var dn = req.dn.toString();
    switch(dn){
    case 'cn=root, dc=example, dc=com':
        if (req.credentials !== 'secret') {
        return next(new ldap.InvalidCredentialsError());
        }
        break;
    case 'cn=rootLDAP, dc=example, dc=com':
        if (req.credentials !== 'secretldap') {
        return next(new ldap.InvalidCredentialsError());
        }
        break;
    case 'cn=rootFile, dc=example, dc=com':
        if (req.credentials !== 'secretfile') {
        return next(new ldap.InvalidCredentialsError());
        }
        break;
    default:
        return next(new ldap.InvalidCredentialsError());
    }

    res.end();
    return next();
});
```

4.2.3 **Operación de búsqueda**

Esta es la principal operación de nuestro servidor LDAP y por la cual ha sido concebido el directorio virtual, realizar búsquedas en los arboles de directorio de los datos integrados.

Aquí se determina en que parte del árbol de debe hacer la búsqueda, en la rama de los datos del fichero XML o en los datos del LDAP. Esto se determinara según la petición que haga el cliente en el directorio virtual.

En el caso que se haga una petición de *dc=people, dc=example, dc=com*, se iniciara la búsqueda en los datos LDAP, en el caso de una petición de *dc=people-ext, dc=example, dc=com*, se iniciara la búsqueda en los datos XML. También se puede dar el caso que queramos todo el sistema de directorio al completo, en este caso se hará una petición a *dc=example, dc=com*.

También en el caso que se quiera buscar en una rama inexistente se generará un error indicando este problema.

```
server.search('dc=example,dc=com', authorize, function(req, res, next) {
    console.log('== LDAP.search ==');
    console.log('* base object: ' + req.dn.toString());
    console.log('* scope: ' + req.scope);
    console.log('* filter: ' + req.filter.toString());
    db = {};
    var searchDn = req.dn.toString();
    var scope = req.scope;

    if (searchDn === 'dc=example, dc=com') {
        var obj = {
            dn: 'dc=example, dc=com',
            attributes: {
                objectclass: ['dcObject', 'organization', 'top'],
```

```
                o: 'example',
                dc: 'example'
            }
        };
        var rootDN = 'dc=example, dc=com';
        db[rootDN] = obj;

        if(scope === 'base' &&
req.filter.matches(db[rootDN].attributes)){
            res.send(db[rootDN]);
            console.log('== LDAP.endSearch ==');
            res.end();
            return next();
        } else if (scope === 'one ' || scope === 'sub') {
            endLDAP = false;
            endFile = false;
            searchLDAP(req, res, next);
            searchFile(req, res, next);
        }
        else{
            res.end();
            return next();
        }
    } else {
        endLDAP = false;
        endFile = false;
        if (searchDn.indexOf('dc=people, dc=example, dc=com') >= 0) {
            endFile = true;
            searchLDAP(req, res, next);
        } else if (searchDn.indexOf('dc=people-ext, dc=example, dc=com')
>= 0) {
            endLDAP = true;
            searchFile(req, res, next);
        } else {
            return next(new ldap.NoSuchObjectError(searchDn));
        }
    }

});
```

4.3 Funciones del servidor LDAP

Aquí se especifican las funciones creadas para que las operaciones del directorio virtual puedan ser realizadas correctamente.

4.3.1 Función de autorización

En esta función es donde se indica que privilegios tienen los usuarios y por lo tanto sobre que ramas pueden realizar operaciones.

Como se ha indicado anteriormente existen tres usuarios diferentes y cada uno tiene unos privilegios diferentes. Ahora detallaré que tipo de privilegios tiene cada uno.

El usuario *cn=root, dc=example, dc=com* es el administrador de todo el directorio virtual y por lo tanto puede realizar cualquier operación de búsqueda en todas las ramas que conforman el directorio virtual. Eso es que puede realizar operaciones en el directorio principal que es *dc=example, dc=com* y todas sus ramas hijas.

El usuario *cn=rootLDAP, dc=example, dc=com* puede realizar operaciones de búsqueda en todo la rama de datos LDAP. Por lo tanto puede realizar operaciones de búsqueda en el directorio *dc=people, dc=example, dc=com* y todas sus ramas hijas.

El usuario *cn=rootFile, dc=example, dc=com* puede realizar operaciones de búsqueda en todo la rama de datos XML. Por lo tanto puede realizar operaciones de búsqueda en el directorio *dc=people-ext, dc=example, dc=com* y todas sus ramas hijas.

En el caso que un usuario intente realizar una operación en un directorio donde no tiene acceso, dará como resultado un error de privilegios de usuario.

```
function authorize(req, res, next) {
    console.log('== LDAP.authorize ==');
    var dn = req.dn.toString();
    var bindDN = req.connection.ldap.bindDN.toString();
    switch(bindDN){
    case 'cn=root, dc=example, dc=com':
        var rootDN = 'dc=example, dc=com';
        if (!(dn.indexOf(rootDN) >= 0))
            return next(new ldap.InsufficientAccessRightsError());
        break;
    case 'cn=rootLDAP, dc=example, dc=com':
        var rootDN = 'dc=people, dc=example, dc=com';
        if (!(dn.indexOf(rootDN) >= 0))
            return next(new ldap.InsufficientAccessRightsError());
        break;
    case 'cn=rootFile, dc=example, dc=com':
        var rootDN = 'dc=people-ext, dc=example, dc=com';
        if (!(dn.indexOf(rootDN) >= 0))
            return next(new ldap.InsufficientAccessRightsError());
        break;
    default:
        return next(new ldap.InsufficientAccessRightsError());
    }
    return next();
}
```

4.3.2 **Función de conexión e importación de datos LDAP**

En esta función se establece una conexión con el servidor LDAP cliente que contiene todos los datos LDAP que se quiere integrar y se importan todos esos datos.

Para realizar esta conexión se utilizaran los datos de configuración dados del servidor cliente, estos datos nos indican en qué dirección y puerto se ubica dicho servidor y nos proporciona las credenciales necesarias para acceder a los datos.

Por lo tanto en el proceso de petición de los datos tenemos que realizar dos operaciones. Primero tenemos que autenticarnos con un usuario con suficientes privilegios para acceder a todos los datos. I cuando nos hemos autenticado correctamente, se da una petición de operación de búsqueda de todo el árbol de datos

Si en algún paso del proceso se da un error se lanza un error indicando el error e interrumpiendo la conexión. Los posibles errores pueden ser de que no se ha autenticado de forma correcta o que la operación de búsqueda haya producido un error.

Al finalizar la importación de datos de forma correcta se indica mediante la variable global *endLDAP* y se hace una llamada a la función de búsqueda para que inicie la operación de los datos integrados.

```
function searchLDAP(req, res, next) {
    console.log('== LDAP.searchLDAP ==');
    var rootDNLDAP = 'dc=people, dc=example, dc=com';
    var obj = {
        dn: rootDNLDAP,
        attributes: {
            objectclass: ['dcObject', 'organization', 'top'],
            o: 'people',
            dc: 'people'
        }
    };
    db[rootDNLDAP] = obj;
    var scope = req.scope;
    if(scope === 'base' &&
req.filter.matches(db[rootDNLDAP].attributes)){
        res.send(db[rootDNLDAP]);
        console.log('== LDAP.endSearch ==');
        res.end();
        return next();
    }

   //Conectar servidor ldap
    client.bind(usernameClient, passwordClient, function(errBind) {
        if(errBind){
            console.log(errBind.message);
            client.unbind(function(errUnbind) {
                if(errUnbind){
                    console.log(errUnbind.message);
                } else{
                    console.log('client disconnected');
                }
            });
            endLDAP = true;
            endConnection(req, res, next);
        } else {
            console.log('connected server LDAP');
            client.search(dnLDAPServer, optsClient, function(errSearch,
search) {
                if(errSearch){
                    console.log(errSearch.message);
                    client.unbind(function(errUnbind) {
                        if(errUnbind){
                            console.log(errUnbind.message);
                        } else{
                            console.log('client disconnected');
                            endLDAP = true;
                            endConnection(req, res, next);
                        }
                    });
                } else {
                    console.log("Load server directory");
                    search.on('searchEntry', function(entry) {
                        var entryObject = entry.object;
                        if(entryObject){
                            var dnFile = entryObject.dn;
                            delete entryObject.dn;
                            delete entryObject.controls;
                            var obj = {
                                dn: dnFile+', '+rootDNLDAP,
                                attributes: entryObject
                            };
```

```
                            db[dnFile+', '+rootDNLDAP] = obj;
                        }
                    });
                    search.on('error', function(errSearchOn) {
                        console.error(errSearchOn);
                    });
                    search.on('end', function(result) {
                        endLDAP = true;
                        searchDB(req, res, next);
                    });

                }
            });
        }
    });

}
```

4.3.3 **Función de lectura e importación de datos XML**

En esta función es donde se realizara la lectura de los datos proveídos en un fichero XML. Para leer el fichero se necesita la dirección del fichero que se ha configurado anteriormente. Además con el módulo *fs* podemos hacer una lectura del fichero y con el módulo *xml2js* podemos interpretar de forma correcta la estructura del fichero XML.

Con todo esto podemos importar los datos XML y conformar la estructura en árbol que tiene descrita. Al finalizar la importación de datos de forma correcta se indica mediante la variable global *endFile* y se hace una llamada a la función de búsqueda.

```
function searchFile(req, res, next) {
    console.log('== LDAP.searchFile ==');

    var rootDNLDAP = 'dc=people-ext, dc=example, dc=com';
    var obj = {
        dn: rootDNLDAP,
        attributes: {
            objectclass: ['dcObject', 'organization', 'top'],
            o: 'people-ext',
            dc: 'people-ext'
        }
    };

    db[rootDNLDAP] = obj;
    var scope = req.scope;
    if(scope === 'base' &&
req.filter.matches(db[rootDNLDAP].attributes)){
        res.send(db[rootDNLDAP]);
        console.log('== LDAP.endSearch ==');
        res.end();
        return next();
    }

    fs.readFile(pathFile, function (errReadFile, data) {
        if (errReadFile) throw err;
        else{
            //importar xml
            console.log("Read file directory")
            var parser = new xml2js.Parser();
            parser.parseString(data, function (errParser, result) {
                if (errParser) throw err;
```

```
                else{
                    //xml a db, la estructura del xml debe ser la
indicada
                    console.log("Load file directory");
                    var objParams = readParams(result);
                    var nObj = objParams.length;
                    for(var x=0; x<nObj; x++){
                        var dnFile = objParams[x].dn[0];
                        var attrFile = objParams[x].attributes[0];
                        var obj = {
                            dn: dnFile+', '+rootDNLDAP,
                            attributes: attrFile
                        };
                        db[dnFile+', '+rootDNLDAP] = obj;
                    }
                    endFile = true;
                    searchDB(req, res, next);
                }
            });

        }
    });

}
```

4.3.4 **Función auxiliar de importación de datos XML**

Esta función apoya a la anterior para importar los datos XML de forma correcta. Su finalidad es buscar dentro de la estructura del fichero XML, los objetos que describen los directorios que conforman el árbol de directorios que contiene.

Para que esta función actúe de forma correcta la estructura del fichero XML debe ser la que se ha descrito anteriormente, sino los datos serán importados de forma incorrecta y se producirán errores.

```
function readParams(object){
    var param = Object.keys(object)[0];
    if(param === "0"){
        var keyExist = false;
        var attributesExist = false;
        Object.keys(object[0]).forEach(function(key) {
            if(key === "dn"){
                keyExist = true;
            } else if(key === "attributes") {
                attributesExist = true;
            }
        });
        if(keyExist && attributesExist)
            return (object);
        else
            return(false);
    }
    else
        var obj = readParams(object[param]);

    return (obj);
}
```

4.3.5 **Función de búsqueda de los datos integrados**

En esta función se realizara la búsqueda de los datos solicitados en los datos que se han integrado. Por lo tanto se hará una búsqueda de la rama del árbol solicitada.

Esta función solo se pondrá en marcha cuando se hayan importado de forma correcta los dos tipos de datos que conforman el directorio virtual, los datos XML y LDAP. Cuando se cumpla esta condición se iniciará la búsqueda del directorio solicitado y todas las ramas hijas si así se pide.

El usuario recibirá toda la información que ha solicitado independientemente del origen de los datos.

```
function searchDB(req, res, next){
    if(endLDAP && endFile){
        var dn = req.dn.toString();
        var scopeCheck;
        console.log('Searching');
        switch (req.scope) {
        case 'base':
            if (req.filter.matches(db[dn].attributes)) {
                res.send({
                    dn: dn,
                    attributes: db[dn].attributes
                });
            }
        res.end();
        return next();
        break;
        case 'one':
        scopeCheck = function(k) {
            if (req.dn.equals(k))
                return true;

            var parent = ldap.parseDN(k).parent();
            return (parent ? parent.equals(req.dn) : false);
        };
        break;
        case 'sub':
        scopeCheck = function(k) {
            return (req.dn.equals(k) || req.dn.parentOf(k));
        };
        break;
        }
        Object.keys(db).forEach(function(key) {
        if (!scopeCheck(key))
            return;
        if (req.filter.matches(db[key].attributes)) {
            res.send({
                dn: key,
                attributes: db[key].attributes
            });
        }
        });

        console.log('== LDAP.endSearch ==');
        res.end();
        return next();
    }
}
```

5 Recursos utilizados

5.1 Node.js

Node.js es un entorno de programación en la capa del servidor basado en el lenguaje de programación JavaScript, con I/O de datos en una arquitectura orientada a eventos y basado en el motor JavaScript V8. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web.

Al contrario que la mayoría del código JavaScript, no se ejecuta en un navegador, sino en el lado del servidor. Node.js implementa algunas especificaciones de CommonJS.

5.2 JavaScript

JavaScript es un lenguaje de programación orientado a objetos. Se utiliza principalmente es su lado de cliente, implementado como parte de un navegador web y permitiendo una web dinámica. También existe la del lado del servidor, utilizada en este proyecto, se puede utilizar para aplicaciones externas a la web, creación de aplicaciones en servidores, etc.

5.3 Módulos Node.js

Node.js incorpora varios "módulos básicos" compilados en el propio binario, como por ejemplo FileSystem (fs), también pude incorporar módulos de terceros para extender sus funciones.

Para su instalación descargar en http://nodejs.org/download/.

5.3.1 Módulo FileSystem (fs)

Módulo básico de node.js que nos permite interactuar con los ficheros del sistema, podemos leer, escribir, borrar ficheros, entre otros tipos de acciones orientadas a los ficheros.

No necesita instalación, está incluido en la instalación de Node.js.

5.3.2 Módulo Ldapjs

Módulo creado por terceros, nos permite crear clientes y servidores LDAP. Implementa las operaciones más comunes de LDAPv3, es 100% compatible con el protocolo LDAP y con cualquier otro sistema que use LDAPv3.

Para su instalación ejecutar comando *npm install ldapjs*.

5.3.3 Módulo xml2js

Módulo creado por terceros, nos permite leer la estructura de un fichero XML. Convierte un XML en un objeto JavaScript, también hace la operación inversa.

Para su instalación ejecutar comando *npm install xml2js*.

6 Evaluación

Aquí se presentaran algunos ejemplos de funcionamiento y su correcto funcionamiento. Se harán diferentes peticiones a nuestro servidor mediante comandos OpenLDAP.

La estructura principal del directorio virtual es como se ha detallado anteriormente. Por lo tanto las únicas partes que pueden variar son el contenido de datos del fichero XML i del servidor cliente LDAP. En este ejemplo se han utilizado los siguientes datos.

Fichero XML:

```
<directory>
      <obj>
            <dn>dc=otroejemplo, dc=es</dn>
            <attributes>
                  <objectclass>dcObject</objectclass>
                  <objectclass>organization</objectclass>
                  <objectclass>top</objectclass>
                  <o>otroejemplo</o>
                  <dc>otroejemplo</dc>
            </attributes>
      </obj>
      <obj>
            <dn>dc=people, dc=otroejemplo, dc=es</dn>
            <attributes>
                  <objectclass>dcObject</objectclass>
                  <objectclass>organization</objectclass>
                  <objectclass>top</objectclass>
                  <o>people</o>
                  <dc>people</dc>
            </attributes>
      </obj>
</directory>
```

Servidor cliente LDAP:

```
dn: dc=example,dc=com
description: example
dc: example
o: example.com
objectClass: top
objectclass: dcObject
objectClass: organization
```

```
dn: ou=people,dc=example,dc=com
description: people example
ou: people
objectclass: dcObject
objectClass: organizationalUnit
objectclass: top

dn: ou=people-ext,dc=example,dc=com
description: people-ext example
ou: people-ext
objectclass: dcObject
objectClass: organizationalUnit
objectclass: top
```

6.1 Petición del árbol completo

En este ejemplo utilizaremos el usuario administrador para hacer una petición a todo el árbol completo del directorio virtual. Si utilizásemos otro usuario, obtendríamos un error por falta de privilegios.

Comando de petición:

```
ldapsearch -H ldap://localhost:1389 -x -D cn=root,dc=example,dc=com -w secret -b dc=example,dc=com
```

Resultado petición:

```
# extended LDIF
#
# LDAPv3
# base <dc=example,dc=com> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#

# example.com
dn: dc=example, dc=com
objectclass: dcObject
objectclass: organization
objectclass: top
o: example
dc: example
```

```
# people.example.com
dn: dc=people, dc=example, dc=com
objectclass: dcObject
objectclass: organization
objectclass: top
o: people
dc: people

# people-ext.example.com
dn: dc=people-ext, dc=example, dc=com
objectclass: dcObject
objectclass: organization
objectclass: top
o: people-ext
dc: people-ext

# otroejemplo.es.people-ext.example.com
dn: dc=otroejemplo, dc=es, dc=people-ext, dc=example, dc=com
objectclass: dcObject
objectclass: organization
objectclass: top
o: otroejemplo
dc: otroejemplo

# people.otroejemplo.es.people-ext.example.com
dn: dc=people, dc=otroejemplo, dc=es, dc=people-ext, dc=example, dc=com
objectclass: dcObject
objectclass: organization
objectclass: top
o: people
dc: people

# example.com.people.example.com
dn: dc=example, dc=com, dc=people, dc=example, dc=com
dc: example
description: example
o: example.com
objectclass: top
objectclass: dcObject
objectclass: organization
```

```
# people, example.com.people.example.com
dn: ou=people, dc=example, dc=com, dc=people, dc=example, dc=com
description: people example
objectclass: dcObject
objectclass: organizationalUnit
objectclass: top
ou: people

# people-ext, example.com.people.example.com
dn: ou=people-ext, dc=example, dc=com, dc=people, dc=example, dc=com
description: people-ext example
objectclass: dcObject
objectclass: organizationalUnit
objectclass: top
ou: people-ext

# search result
search: 2
result: 0 Success

# numResponses: 9
# numEntries: 8
```

En el caso que hagamos la petición con un usuario que no tenga suficientes privilegios obtendremos la siguiente respuesta.

Comando de petición:

```
ldapsearch -H ldap://localhost:1389 -x -D cn=rootLDAP,dc=example,dc=com -
w secretldap -b dc=example,dc=com
```

Resultado petición:

```
# extended LDIF
#
# LDAPv3
# base <dc=example,dc=com> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#


# search result
```

```
search: 2
result: 50 Insufficient access
matchedDN: dc=example, dc=com
text: Insufficient Access Rights


# numResponses: 1
```

También se puede dar el caso que no introduzcamos las credenciales correctamente, caso que nos dará el siguiente error.

Comando de petición:

```
ldapsearch -H ldap://localhost:1389 -x -D cn=rootLDAP,dc=example,dc=com -
w secret -b dc=example,dc=com
```

Resultado petición:

```
ldap_bind: Invalid credentials (49)
      matched DN: dc=example, dc=com
      additional info: Invalid Credentials
```

6.2 **Petición de datos XML**

En este caso haremos una petición a la parte de los datos XML, en este caso no nos hará falta el usuario administrador para acceder a estos datos.

Comando de petición:

```
ldapsearch -H ldap://localhost:1389 -x -D cn=rootFile,dc=example,dc=com -
w secretfile -b dc=people-ext,dc=example,dc=com
```

Resultado petición:

```
# extended LDIF
#
# LDAPv3
# base <dc=people-ext,dc=example,dc=com> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#


# people-ext.example.com
dn: dc=people-ext, dc=example, dc=com
objectclass: dcObject
objectclass: organization
objectclass: top
```

```
o: people-ext
dc: people-ext

# otroejemplo.es.people-ext.example.com
dn: dc=otroejemplo, dc=es, dc=people-ext, dc=example, dc=com
objectclass: dcObject
objectclass: organization
objectclass: top
o: otroejemplo
dc: otroejemplo

# people.otroejemplo.es.people-ext.example.com
dn: dc=people, dc=otroejemplo, dc=es, dc=people-ext, dc=example, dc=com
objectclass: dcObject
objectclass: organization
objectclass: top
o: people
dc: people

# search result
search: 2
result: 0 Success

# numResponses: 4
# numEntries: 3
```

6.3 **Petición de datos LDAP**

En este caso haremos una petición a la parte de los datos LDAP, en este caso no nos hará falta el usuario administrador para acceder a estos datos.

Comando de petición:

```
ldapsearch -H ldap://localhost:1389 -x -D cn=rootLDAP,dc=example,dc=com -
w secretldap -b dc=people,dc=example,dc=com
```

Resultado petición:

```
# extended LDIF
#
# LDAPv3
# base <dc=people,dc=example,dc=com> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
```

```
#

# people.example.com
dn: dc=people, dc=example, dc=com
objectclass: dcObject
objectclass: organization
objectclass: top
o: people
dc: people

# example.com.people.example.com
dn: dc=example, dc=com, dc=people, dc=example, dc=com
dc: example
description: example
o: example.com
objectclass: top
objectclass: dcObject
objectclass: organization

# people, example.com.people.example.com
dn: ou=people, dc=example, dc=com, dc=people, dc=example, dc=com
description: people example
objectclass: dcObject
objectclass: organizationalUnit
objectclass: top
ou: people

# people-ext, example.com.people.example.com
dn: ou=people-ext, dc=example, dc=com, dc=people, dc=example, dc=com
description: people-ext example
objectclass: dcObject
objectclass: organizationalUnit
objectclass: top
ou: people-ext

# search result
search: 2
result: 0 Success

# numResponses: 5
```

```
# numEntries: 4
```

7 Manual

Se detallará un pequeño manual para la instalación del Node.js y sus módulos y también como se debe usar el directorio virtual.

7.1 Manual de instalación

El único elemento que nos hace falta para utilizar este directorio virtual, es el Node.js y algunos módulos de terceros.

En primer lugar descargaremos el Node.js desde aquí http://nodejs.org/download/ y lo instalaremos. Se puede instalar en un Windows, Mac o Linux. En mi caso he utilizado Linux.

El fichero creado que contiene la aplicación del directorio virtual se llama *ldapvirtual.js*. Ubicamos el fichero donde queramos tener la aplicación.

En el mismo directorio donde tenemos el fichero *ldapvirtual.js*, es donde tendremos que ubicarnos para ejecutar los comandos que instalaran los módulos necesarios que necesitamos.

Ejecutaremos los siguientes comandos para la instalación de los módulos. Para la instalación del módulo ldapjs, debemos ejecutar este comando *npm install ldapjs*. Para la instalación del módulo xml2js, debemos ejecutar este comando *npm install xml2js*.

Tras finalizar estos pasos nos debe haber creado un nuevo directorio junto el fichero *ldapvirtual.js*, que contiene los dos módulos mencionados.

Para iniciar el directorio virtual ejecutaremos el comando *node ldapvirtual.js*.

7.2 Manual de uso

Aquí se indicara como se debe configurar el directorio virtual y también como se deben realizar las búsquedas de los datos.

7.2.1 Configuración

Antes de poner en marcha el directorio virtual, se tiene que indicar algunos parámetros básicos. Estos parámetros son la ubicación de los datos LDAP y los datos XML.

Para configurar los datos LDAP se tiene que modificar el parámetro *url* de la variable *client*. Aquí se tiene que indicar la dirección ip y el puerto donde se encuentra el servidor cliente LDAP que contiene los datos.

```
var client = ldap.createClient({
    url: 'ldap://127.0.0.1:1390',
    timeout: 5000,
    connectTimeout: 10000
});
```

Para configurar los datos XML se tiene que indicar la ubicación del fichero XML que contiene dichos datos. Para indicar esto se tiene que modificar la variable *pathFile*.

```
var pathFile = '/home/user/desktop/ldap.xml';
```

Al finalizar la configuración ya se puede poner en marcha el directorio virtual mediante el comando *node ldapvirtual.js*.

7.2.2 **Ejemplos de uso**

Para este caso, para las búsquedas de los datos en el directorio virtual se utilizan comandos de OpenLDAP.

Un ejemplo de comando es el siguiente:

```
ldapsearch -H ldap://localhost:1389 -x -D cn=root,dc=example,dc=com -w
secret -b dc=example,dc=com
```

Se utiliza la operación *ldapsearch* para realizar la búsqueda, esta operación tiene diferentes parámetros a configurar.

El parámetro *-H* se indica la url donde está ubicado nuestro directorio virtual, el parámetro *-x* indica que utilizaremos autenticación simple, *-D* indica el usuario que utilizaremos para autenticarnos, *-w* indica la contraseña de este usuario y finalmente *-b* indicamos la búsqueda que queremos realizar.

8 Conclusiones

Existen multitud de tipos de estructuras de datos diferentes, cada una organizada de distinta forma y con diferentes funciones. En este proyecto se ha querido encontrar una forma de unir dos sistemas diferentes, para poder hacer consultas a estos datos de manera unificada.

La solución a este problema ha sido la creación de un directorio virtual. Con ese sistema de directorios se ha querido hacer un sistema unificado para realizar consultas a los datos y además crear un sistema único de credenciales para unificar la política de acceso.

Hemos podido ver que con un directorio virtual se han realizado los objetivos inicialmente planteados, también por las características de ser de un directorio virtual no ha supuesto un coste de ningún tipo. Además de ofrecer un alto rendimiento ya que el directorio solo existe en el momento de realizar la petición de los datos. Por lo tanto, no supone un gran coste para la máquina utilizada.

En conclusión de puede decir que el directorio virtual ofrece las funciones esperadas dentro de las especificaciones esperadas y por lo tanto se han cumplido los objetivos de este proyecto.

9 Bibliografía

[1] Servicio de directorio: http://es.wikipedia.org/wiki/Servicio_de_directorio

[2] Metadirectorios: http://en.wikipedia.org/wiki/Metadirectory

[3] Directorio virtual: http://en.wikipedia.org/wiki/Virtual_directory

[4] LDAP: http://es.wikipedia.org/wiki/LDAP

[5] XML: http://www.w3schools.com/xml/

[6] Node.js: http://nodejs.org/

[7] Ldapjs: http://ldapjs.org/

[8] Xml2js: https://npmjs.org/package/xml2js

[9] Metadirectorios i directorios virtuales: http://windowsitpro.com/identity-management/rise-virtual-directory-servers

[10] Manual OpenLDAP: http://www.openldap.org/