DISEÑO E IMPLEMENTACIÓN DE UN PROXY CORBA-LDAP.

Autor: Manuel Roque Gómez Morgado.

Tutores: José Ángel Gómez Argudo.
María Teresa Ariza Gómez.

Fecha: Julio 2002.

El presente proyecto está dedicado a mis padres.

Sobre el proyecto:

El siguiente trabajo trata de relacionar dos de las tecnologías que hoy día experimentan un elevado crecimiento dentro del campo de la telemática, tales como el acceso a servicios de directorios basado en el protocolo LDAP y los sistemas distribuidos basados en objetos que cumplen con el estándar CORBA.

El fin del trabajo radica en conseguir que objetos que funcionando sobre una plataforma CORBA genérica puedan realizar peticiones a un servidor LDAP que contenga información de interés para estos objetos, y todo ello desde el desconocimiento por parte de los clientes CORBA del funcionamiento del protocolo LDAP y de la ubicación del servidor de directorio al que desean acceder.

El resultado de la implementación es un sistema en el que los usuarios debidamente autentificados realizan consultas al servidor LDAP desde un cliente CORBA que usa el proxy CORBA-LDAP como pasarela para poder interaccionar con el servidor de directorio

Abstract:

This project tries to join two of the most growing technologies in Telecomunications and Computing. These technologies are, the access to diretory services using the LDAP protocol and the distributed systems based in CORBA objects.

The project´s purpose is getting that objects that work over a system based in CORBA could make requests to an LDAP server that holds interesting information for these objects. In this situation, the CORBA clients don´t know neither where the server is nor how the LDAP protocol works.

The result of the work is a system where authenticated users send requests from a CORBA client to a LDAP server, using an element called “proxy CORBA-LDAP” as a bridge between the two different technologies.

MEMORIA

Índice de la memoria.

1. Capítulo 1: Introducción. (9)
    - 1.1 Objetivos y alcance. (13)
    - 1.2 Organización de la memoria. (15)

2. Capítulo 2: El protocolo LDAP. (17)
    - 2.1 ¿Qué es un directorio?. (17)
    - 2.2 Integración del directorio con otros servicios. (19)
    - 2.3 Clientes y servidores de directorio. (20)
    - 2.4 Directorios distribuidos.(21)
    - 2.5 Seguridad en servicios de directorio. (22)
    - 2.6 El servicio de directorio X.500. (22)
    - 2.7 Evolución del protocolo LDAP. (23)
    - 2.8 Arquitectura del protocolo LDAP. (26)
    - 2.9 Las recomendaciones del protocolo LDAP. (28)
    - 2.10 El modelo LDAP. (31)
        - 2.10.1 El modelo de información LDAP. (31)
        - 2.10.2 El modelo de nombrado LDAP. (31)
        - 2.10.3 El modelo funcional LDAP. (33)
        - 2.10.4 El modelo de seguridad LDAP. (36)
    - 2.11 Información sobre el servidor. (44)
    - 2.12 Seguridad en LDAP. (45)
    - 2.13 Mecanismos de seguridad en LDAP. (45)
        - 2.13.1 Autentificación anónima. (46)
        - 2.13.2 Autentificación simple. (46)
        - 2.13.3 Autentificación SASL. (46)
    - 2.14 URLs en LDAP. (48)
    - 2.15 Elementos del protocolo LDAP. (49)
    - 2.16 Codificación y transferencia de mensajes. (72)
    - 2.17 Notificaciones no solicitadas. (72)
    - 2.18 El formato LDIF. (73)
    - 2.19 Implementación del protocolo. Servidores comerciales. (75)

3. Capítulo 3: El servidor OpenLDAP. (79)
    - 3.1 Servicios de directorio. (79)
    - 3.2 El proyecto OpenLDAP. (81)
    - 3.3 Formatos de OpenLDAP. (81)
    - 3.4 Implementación del protocolo LDAP. (82)
    - 3.5 La aplicación Slapd. (83)
        - 3.5.1 El fichero de configuración de Slapd. (87)

3.6 Control de acceso al sistema. (93)
3.7 Herramientas de gestión y mantenimiento de bases de datos. (95)
3.7.1 La herramienta slapadd. (96)
3.7.2 La aplicación slapindex. (97)
3.7.3 La aplicación slapcat. (97)
3.7.4 La aplicación ldapdelete. (98)
3.7.5 La herramienta ldappasswd. (100)
3.7.6 La aplicación ldapsearch. (101)
3.7.7 La aplicación ldapmodify. (103)
3.7.8 La aplicación ldapadd. (104)
3.7.9 La aplicación ldapmodrdn. (104)
3.8 Esquemas de directorio en OpneLDAP. (105)
3.8.1 Definición de tipos de atributos. (107)
3.8.2 Definición de clases objeto. (111)
3.9 La aplicación Slurpd. (113)

4. Capítulo 4: El API JNDI. (115)
4.1 Sistemas de nombrado. (115)
4.2 El concepto de directorio. (117)
4.3 Uso tradicional de directorios. (119)
4.4 Introducción al API JNDI. (119)
4.5 Principios de diseño de JNDI. (119)
4.6 Arquitectura de JNDI. (120)
4.7 Empaquetado de JNDI. (121)
4.7.1 El paquete javax.naming. (122)
4.7.2 El paquete javax.naming.directory. (127)
4.7.3 El paquete javax.naming.event. (132)
4.7.4 El paquete javax.naming.ldap. (134)
4.7.5 El paquete javax.naming.spi. (137)
4.8 Ventajas del diseño de JNDI. (139)
4.9 Propiedades de entorno en JNDI. (142)
4.10 Remisiones en JNDI. (149)
4.11 URLs en JNDI. (149)
4.12 Aplicaciones de JNDI. (150)
4.12.1 Autentificación de usuarios. (150)
4.12.2 Correo electrónico. (151)
4.12.3 Consulta de servicios. (151)
4.12.4 Funcionamiento como DNS. (152)
4.12.5 Servidor de configuración. (152)

5. Capítulo 5: Implementación de LDAP en JNDI. (152)
    5.1 Elementos de un servicio de directorio en JNDI. (153)
    5.2 Conexiones LDAP en JNDI. (155)
    5.3 Implementación de atributos en JNDI. (158)
    5.4 URLs en JNDI. (159)
    5.5 Remisiones en JNDI. (160)
    5.6 Alias en JNDI. (161)
    5.7 Recuperar los atributos de una entrada. (162)
    5.8 Implementación de la operación MODIFY_RDN en JNDI. (162)
    5.9 Implementación de la operación SEARCH. (163)
        5.9.1 Controles de búsqueda en JNDI. (165)
        5.9.2 Resultados de la búsqueda. (167)
    5.10 La operación COMPARE. (168)
    5.11 La operación MODIFY. (168)
    5.12 La operación ADD. (170)
    5.13 La operación DELETE. (170)
    5.14 Esquemas LDAP en JNDI. (171)
    5.15 Controles en JNDI. (173)
    5.16 Operaciones extendidas en JNDI. (174)

6. Capítulo 6: El estándar CORBA. (177)
    6.1 Introducción a CORBA. (177)
    6.2 El grupo de gestión de objetos (OMG). (178)
    6.3 La norma CORBA (Common Object Request Broker). (178)
    6.4 La arquitectura CORBA. (181)
    6.5 Funcionamiento de CORBA. (184)
    6.6 El lenguaje de definición de interfaces IDL. (185)
        6.6.1 Tipos de datos simples en IDL. (186)
        6.6.2 Estructura de un fichero IDL. (187)
        6.6.3 Correspondencia de datos JAVA-IDL. (191)
    6.7 Excepciones en CORBA. (195)
    6.8 Adaptadores de objetos. (196)
    6.9 Desarrollo de aplicaciones CORBA. (197)
    6.10 Invocación dinámica. (198)
    6.11 Servicios CORBA. (199)
    6.12 Facilidades CORBA. (200)
    6.13 Dominios CORBA. (201)
    6.14 El protocolo GIOP. (202)
    6.15 El protocolo IIOP (.202)
    6.16 ORBs comerciales. (203)

7. Capítulo 7: El protocolo SSL (Socket Security Layer). (205)
    7.1 El protocolo TLS (Transport Security Layer). (206)
    7.2 Autentificación en SSL/TLS. (206)
    7.3 Certificados de clave pública. (207)
        7.3.1 Generación de un certificado de clave pública. (210)
    7.4 Negociación de la conexión SSL. (213)

8. Capítulo 8: Desarrollo del sistema. (215)
    8.1 Instalación del servidor de directorio. (215)
        8.1.1 Información contenida en el directorio. (223)
    8.2 La plataforma CORBA. (224)
        8.2.1 Elección del ORB. (225)
        8.2.2 La herramienta ANT. (226)
        8.2.3 JACORB. (227)
    8.3 El proxy CORBA-LDAP. (230)
        8.3.1 Diseño del proxy CORBA-LDAP. (231)
        8.3.2 Los métodos ofrecidos por el servidor CORBA. (232)
        8.3.3 Tipos de datos del proxy. (236)
        8.3.4 La interfaz del servidor CORBA. (238)
        8.3.5 Desarrollo del servidor CORBA. (240)
        8.3.6 Generación de clases. (249)
    8.4 Diseño del cliente CORBA. (250)
        8.4.1 Inicio del cliente . (251)
        8.4.2 La operación compare. (255)
        8.4.3 La operación delete. (256)
        8.4.4 La operación add. (258)
        8.4.5 La operación modify. (259)
        8.4.6 La operación modify_rdn. (260)
        8.4.7 La operación search. (261)
        8.4.8 La operación listar. (262)
        8.4.9 Consulta de características del servidor. (263)
        8.4.10 Operaciones de esquema. (264)
        8.4.11 Presentación de resultados. (265)
    8.5 Implementación del sistema. (265)
    8.6 Empaquetado de las aplicaciones. (267)
        8.6.1 Deninición de la interfaz CORBA. (267)
        8.6.2 Empaquetado del proxy CORBA-LDAP. (267)
        8.6.3 Empaquetado del cliente. (267)

9. Capítulo 9: Historia del proyecto. (269)

10. Capítulo 10: Conclusiones y líneas de avance. (275)
    10.1 Líneas de avance. (277)

11. Anexos. (279)
    11.1 Anexo 1. Esquema LDAP usado en las pruebas. (280)
    11.2 Anexo 2. Información de directorio usada en las pruebas. (283)
    11.3 Anexo 3. Configuración de Jacorb. (295)
    11.4 Anexo 4. El protocolo Kérberos. (305)
    11.5 Anexo 5. Guía del administrador. (310)
    11.6 Anexo 6. Guía del usuario. (313)
    11.7 Anexo 7.Guía de instalación. (317)
    11.8 Anexo 8. Pruebas. (319)
    11.9 Anexo 9. Soporte de SSL. (353)
    11.10 Anexo 10: Control de calidad. (355)

12. Bibliografía. (357)

Capítulo 1: Introducción.

Entre los servicios telemáticos que actualmente están experimentando un considerable desarrollo, tanto a nivel de aceptación por parte de los diseñadores de sistemas como de los usuarios de éstos, se encuentran los servicios de directorio.

Un servicio de directorio proporciona al usuario un volumen razonablemente elevado de información dentro de un ámbito concreto, organizando dicha información mediante un modelo de entradas que contienen atributos en los que se almacena la información propiamente dicha.

Una entrada representa una entidad concreta dentro de un servicio de directorio, y los atributos representan datos característicos que posee dicha entidad.

La información se almacena siguiendo un patrón estratégico que permita facilitar la consulta de información por parte del usuario, ofreciendo a éste último unas potentes y eficientes operaciones de consulta sobre la información almacenada. La principal estructura usada por los servicios de directorio para almacenar la información es un esquema en forma de árbol, en el cual las entradas quedan organizadas de manera que el acceso a ellas puede hacerse de manera rápida y sistemática, además de conceptualmente simple.

La evolución de este tipo de servicio en los últimos años ha sido de una notable importancia, sobre todo desde el momento en el que es capaz de ofrecer prestaciones al usuario que anteriormente sólo podía obtener mediante el uso de bases de datos complejas. La simplificación cada vez mayor de este tipo de bases de datos ha llevado a la aparición de unas nuevas bases de datos optimizadas para la búsqueda y consulta de información, quedando en un segundo plano las operaciones de actualización. Este importante paso en la evolución de las bases de datos supone el inicio de los servicios de directorio.

El segundo pilar básico en el éxito de este tipo de tecnología estriba en el uso tanto del modelo cliente-servidor como de un protocolo simple, y que no consume demasiados recursos, que marca las reglas de la comunicación entre cliente y servidor. Este protocolo es LDAP (Lightweight Directory Acces Protocol), que es la evolución del complejo protocolo DAP (Directory Access Protocol) usado en el hoy día desfasado y en vías de desaparición servicio de directorio X.500 de OSI.

Además de la simplicidad, el protocolo LDAP posee una arquitectura que funciona sobre TCP/IP, lo cual supone un factor determinante en la adopción definitiva de LDAP como protocolo principal de los servicios de directorio, ya que esta característica encaja no sólo con la arquitectura de red predominante en Internet sino también la preferida por la gran mayoría de diseñadores de redes locales.

La información contenida en este tipo de servidores es de muy diversa índole, pasando desde información general sobre los usuarios de una empresa o de cualquier otro aspecto de interés hasta información manejada por los elementos de la red. Es el caso del uso de un sistema de directorio capaz de simular el funcionamiento de un servidor de nombres manteniendo información de nombres de máquinas asociada con sus direcciones IP.

También es frecuente encontrar servidores de directorio funcionado como servidores de autentificación, ya que mantienen entradas con nombres de usuario asociadas a claves de acceso o cualquier otro tipo de información que pudiera identificar a los clientes que acceden al sistema.

Merece especial mención el uso de directorios como servidores de configuración, de manera que un nuevo servicio de red, aplicación o elemento en general podría configurarse de manera automática y sin intervención externa sin más que conectarse al servidor de directorio y leer la información de configuración que le corresponde.

Todo lo anterior conlleva a que la tecnología de los servicios de directorio sea un interesante campo que es cada vez más necesario conocer y dominar.

El desarrollo del proyecto planteado gira en torno a la experimentación con sistemas de directorio, pero añadiendo un mayor grado de complejidad que va más allá del simple modelo cliente-servidor, ya que se busca la implantación de este tipo de servicios en sistemas distribuidos orientados a objeto, de manera que cualquier cliente pudiera tener acceso al sistema desde cualquier ubicación y sin conocer el funcionamiento del protocolo LDAP ni poseer los medios necesarios para manejar directamente la información proporcionada por el servidor de directorio.

El problema se traslada ahora al sistema de objetos distribuidos, ya que debe existir en él un objeto capaz de interactuar con el servidor LDAP, problema que ya resuelve el propio protocolo LDAP, pero además este objeto debe proporcionar servicio a cualquier otro objeto existente en el sistema distribuido que sea capaz de invocarlo, imponiéndose así el paradigma de cooperación entre objetos imperante en el diseño de sistemas distribuidos.

Llegado este punto se plantea cómo construir el sistema de objetos distribuidos. Varias son las opciones que pueden barajarse, entre las que se encuentra el uso de aplicaciones que directamente se comuniquen mediante el uso de sockets, el entorno de computación distribuido DCE, el modelo de objetos para componentes distribuidas de Microsoft, el entorno RMI que ofrece Java, etc.

Las soluciones propuestas anteriormente son tecnologías aplicables al escenario planteado con un mayor o menor grado de dificultad. Es especialmente interesante la opción de RMI, ya que su evolución va ligada al lenguaje Java.

Sin embargo, el modelo adoptado para construir el sistema distribuido es el estándar CORBA de OMG, ya que es quizás la tecnología de sistemas distribuidos orientada a objetos con un mayor futuro y capacidad de desarrollo, ya que constantemente están apareciendo nuevos servicios y utilidades.

Otra de las grandes ventajas de CORBA es que ofrece una arquitectura independiente tanto del lenguaje de programación como del sistema operativo usado y la arquitectura de la máquina, aumentando así el grado

de generalidad y permitiendo una mayor heterogeneidad de los componentes del sistema distribuido.

La arquitectura CORBA permite además una elevada independencia entre objetos ya que a la independencia del lenguaje de programación se une la ventaja de liberar al cliente de tener que conocer de antemano la ubicación del servidor que desea invocar, incluso sería posible que desconociera la interfaz que ofrece dicho servidor para la realización de las funciones que implementa.

El objeto que sirve de nexo entre la arquitectura CORBA y el modelo LDAP es el llamado ***proxy CORBA-LDAP***. El término proxy aceptado como un vocablo de uso corriente dentro del argot del mundo informático puede traducirse por el término español ***delegado,*** ya que pertenece a la vez a dos sistemas distintos, representando al primer sistema en el segundo y viceversa.

Esta doble naturaleza de un proxy es aplicable al sistema que se pretende desarrollar, ya que será un elemento que represente el sistema LDAP dentro de CORBA (actuando para ello como servidor CORBA), y al sistema CORBA dentro de LDAP (actuando para ello como cliente LDAP).

Este elemento se encarga de recoger las peticiones de los clientes CORBA, convertir los datos que proporcionan los clientes a datos que pueda usar para comunicarse con el servidor, realizar la petición, recoger los resultados, convertirlos a un formato manejable por los objetos CORBA y enviarlos hacia el objeto que realizó la invocación.

La comunicación entre el proxy y el servidor LDAP se realiza a través de un conjunto de métodos que proporciona JNDI que es un API del lenguaje Java que ofrece un entorno en el que se puede llevar a cabo este fin.

Finalmente el servidor de directorio elegido es OpenLDAP, para lo cual ha prevalecido su sencillez a la hora de manejarlo y que es un software gratuito y de libre distribución.

1.1 Objetivos y alcance.

Tal como se ha indicado en la introducción la idea fundamental del proyecto es la interacción de la tecnología CORBA con la tecnología LDAP.

Para ello se pretende desarrollar principalmente un elemento que sirva de nexo entre dos sistemas basados cada uno de ellos en los dos estándares anteriores. Este elemento es el proxy CORBA-LDAP, que es una aplicación software que presenta como principal característica la dualidad servidor CORBA / cliente LDAP de manera que representa a los clientes CORBA en el modelo LDAP y al servidor LDAP en el modelo CORBA.

Este elemento va a ser desarrollado mediante el lenguaje de programación Java que por un lado usará las clases proporcionadas por el software que implemente el estándar CORBA usado (en este caso la distribución "Jacorb") y por otro lado las clases ofrecidas por el API JNDI para poder construir un cliente LDAP capaz de interactuar con el servidor de directorio mediante los objetos que este API proporciona.

El proxy debe implementar para los clientes todas las posibles operaciones que pueden realizarse sobre el directorio y que no sean competencia exclusiva del administrador del sistema.

El proxy CORBA-LDAP sólo tendrá sentido cuando se encuentre funcionando en el escenario descrito anteriormente, por lo que la instalación de un servidor de directorio basado en el protocolo LDAP (en este caso OpenLDAP) debe considerarse parte del sistema a implementar, así como la información en él contenida.

De la misma manera es conveniente implementar también un objeto CORBA que actúe como cliente y que sea capaz de realizar peticiones al proxy.

El cliente CORBA se considera un elemento de menor importancia en el proyecto, y no porque su relevancia sea menor, sino porque una de las grandes utilidades y ventajas del sistema propuesto es el desarrollo de clientes CORBA independientes del proxy, con lo cual un diseñador podría construir un cliente CORBA cada vez más sofisticado, complejo y con un mayor juego de servicios, hasta el punto de poder desarrollar una compleja plataforma de gestión de directorio, que eso sí, funcionaría siempre sobre CORBA que es lo que se pretende.

Por ello el cliente CORBA desarrollado pretende ofrecer al usuario humano una interfaz gráfica fácil de manejar que facilite la invocación de operaciones por parte de éste, pero sin pretender ser una aplicación compleja y potente.

Además se estudiará la seguridad del sistema, tanto a nivel de autentificación de usuarios como a nivel de transmisión segura de datos a través de las conexiones abiertas en el sistema. Sin embargo este aspecto supondrá una mejora del sistema planteado en caso de poder llevarse a cabo, pero no un objetivo imprescindible del sistema que se pretende construir.

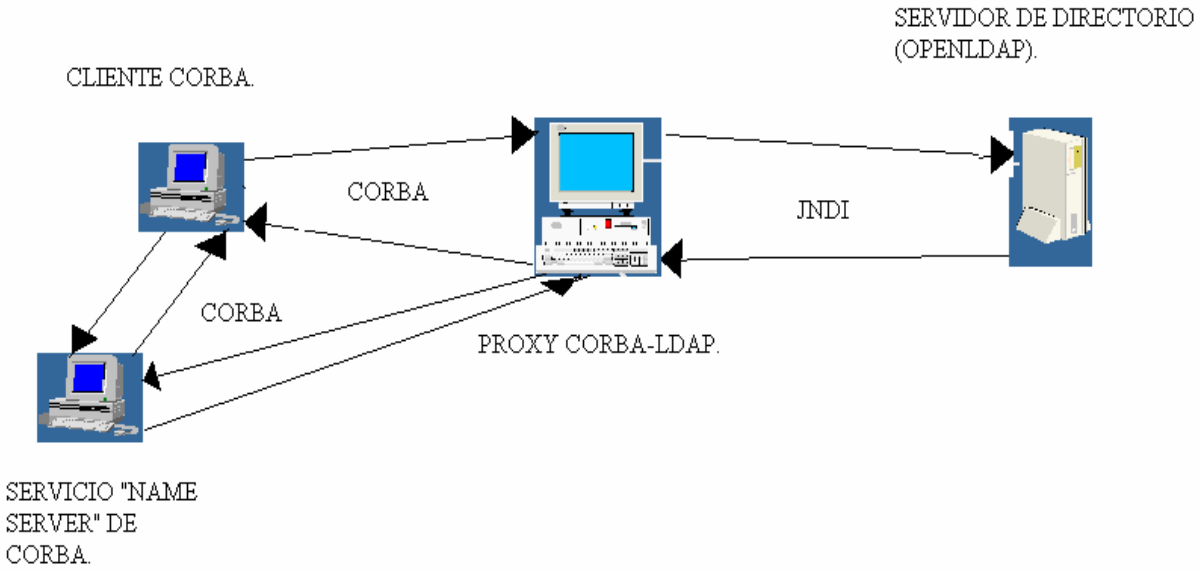
El conjunto de todos estos elementos forma el sistema perseguido por este proyecto en el que el proxy CORBA-LDAP es el eje fundamental del mismo.

Recapitulando, el conjunto de objetivos pueden resumirse en principales (aquellos que son imprescindibles cumplir para la realización del sistema) y secundarios (aquellos que suponen mejoras al sistema pero no son estrictamente necesarios para su funcionamiento):

- ***Objetivos principales.***
  - Construcción del proxy CORBA-LDAP usando para ello JNDI.
  - Que el proxy ofrezca a los clientes CORBA todas las posibles operaciones que puedan realizarse sobre el directorio.
  - Instalación de un servicio de directorio LDAP basado en OpenLDAP.
  - Desarrollo de una aplicación relativamente simple que funcione como cliente CORBA.

- ***Objetivos secundarios.***
  - Construir una interfaz gráfica que permita al usuario humano manejar con comodidad las operaciones que el cliente CORBA puede invocar.
  - Desarrollo de un directorio basado en un esquema propio y con información relativa a ese esquema.
  - Estudio de la seguridad del sistema. Se intentará mejorar el proceso de autentificación de usuarios y la transmisión segura de información a través de la red.

A continuación se representa esquemáticamente el sistema que se pretende construir.



1.2 Organización de la memoria.

La memoria que detalla todo el trabajo realizado para el diseño y la implementación de todo el sistema descrito comienza con índice al que sucede un capítulo de introducción en el que se plantea la situación inicial de la que se parte y se realiza un análisis previo en el que se analiza el escenario perseguido y los medios de los que se dispone para implementarlo. La introducción concluye analizando los objetivos que se desean alcanzar.

Los siguientes seis capítulos recogen la tecnología usada, pretendiendo exponer los temas de los que trata mediante un estilo intermedio entre un estilo explicativo y un estilo propio de una documentación. A lo largo de estos capítulos se presenta el funcionamiento del protocolo LDAP, de la tecnología CORBA, el API JNDI, su relación con el protocolo LDAP, y los servidores de directorio, incluyendo el elegido en particular (OpenLDAP). Se incluye también el protocolo SSL soportado por LDAP.

Una vez analizada la tecnología de la que se dispone se procede a la descripción del desarrollo del sistema, en la cual se exponen los objetivos perseguidos en el diseño, los criterios adoptados, los elementos implementados e información esquemática sobre cómo se han llevado a cabo estos procesos.

En historia del proyecto se describe el desarrollo de las etapas de las que ha constado la realización del trabajo. Se recomienda muy especialmente su lectura, ya que contiene información sobre la evolución del sistema.

A continuación se muestran las conclusiones extraídas y se exponen posibles líneas de avance mediante las cuales el sistema podría ser sensiblemente mejorado o podría servir de modelo para trabajos futuros, o de bloque constructor de sistemas más complejos.

Hay un conjunto de Anexos que recogen información adicional del sistema e información con la que se ha trabajado en algunas fases del proyecto y que ayuda a comprender mejor algunos aspectos de la memoria.

En estos anexos se recogen entre otros aspectos los resultados de una batería de pruebas que fue diseñada para comprobar el correcto funcionamiento del sistema.

Por último se adjunta la bibliografía usada en el proyecto.

Capítulo 2: El protocolo LDAP.

En este apartado se pretende describir la arquitectura y el funcionamiento del propio protocolo LDAP, así como su evolución y futuro, así como ofrecer una visión intuitiva del protocolo que sirva para la comprensión de su descripción exhaustiva.

2.1 ¿Qué es un directorio?

Un directorio es un conjunto de información sobre objetos almacenada siguiendo un cierto orden y que proporciona detalles sobre cada objeto.

En el argot informático es una base de datos especializada que guarda datos sobre objetos siguiendo una cierta estructura.

Esta especialización hace referencia a los siguientes aspectos que son característicos de un directorio.

- ***Relación entre lecturas y escrituras.*** En un directorio la operación de lectura es mucho más frecuente que la de escritura, particularidad que no es aplicable en absoluto a una base de datos.

- ***Extensibilidad.*** Mediante el uso del concepto de esquema es posible definir qué tipo de información se almacena en el directorio, qué reglas debe cumplir esa información y cómo se realizan las operaciones sobre los datos. En los directorios el esquema puede ser

modificado fácilmente para adaptarlo a las necesidades del usuario, lo cual es algo que no puede aplicarse a las bases de datos de propósito general.

- ***Distribución de los datos.*** Algunas bases de datos de propósito general permiten la distribución de los datos, si bien lo hacen de manera ineficiente y usando una serie de protocolos complejos para la actualización de la información. Por el contrario en los servicios de directorio la información referente a un servidor puede distribuirse de forma sencilla y controlada por dicho servidor.

- ***Replicación de la información.*** Las bases de datos admiten que su información pueda ser replicada en otros servidores, si bien la rigidez de las restricciones que se imponen en el sincronismo de las operaciones de actualización debidas a la consistencia de la información supone un problema en cuanto a la complejidad. En el caso de los directorios esta limitación es mucho más flexible debido a que las operaciones de actualización son mucho más esporádicas. Este aspecto comienza a ser crítico cuando las aplicaciones usan esa información para operaciones de autentificación o de configuración.

- ***Rendimiento.*** Las necesidades en cuanto al rendimiento de una base de datos y de un directorio son bastante distintas ya que de un servidor de base de datos se espera que sea capaz de manejar hasta cientos de transacciones por segundos, mientras que de un servidor de directorio se espera que maneje miles de consultas por segundo. Esto se debe a que el servicio de directorio puede formar parte normalmente del núcleo de un gran conjunto de aplicaciones, y debe estar preparado para responder a todas las consultas que se le realicen.

- ***Estándares.*** El hecho de que las bases de datos de propósito general suelan usar variantes del lenguaje SQL no supone un grave problema, dado que rara vez se necesita que distintas bases de datos interactúen con bases de datos de distintos fabricantes. En el caso de los directorios, este aspecto es bastante crítico ya que el servidor de directorio suele interactuar con gran variedad de aplicaciones que pueden estar desarrolladas por distintos fabricantes, lo cual hace que el estricto cumplimiento del estándar definido sea fundamental.

El directorio permite a los usuarios encontrar información que cumpla con las necesidades especificadas por dicho usuario en la consulta.

Los directorios no soportan transacciones. Las transacciones son operaciones de base de datos que permiten controlar la ejecución de una operación compleja, de modo que dicha operación se completa totalmente o no se ejecuta en absoluto. Las bases de datos convencionales ofrecen esta funcionalidad, a costa de hacer su implementación más compleja. Pero el tipo de información que se almacena generalmente en el directorio no requiere una consistencia estricta y se considera aceptable que una determinada información no esté accesible de forma temporal.

La mayoría de las bases de datos convencionales usan el lenguaje de consulta SQL que permite el desarrollo de complejas funciones de consulta y actualización pero con el inconveniente de un elevado coste computacional y de una alta complejidad de la operación. Por el contrario los directorios soportan ser accedidos por protocolos sencillos como LDAP, que es un protocolo sencillo y optimizado que permite desarrollar simples aplicaciones para realizar operaciones en el directorio.

2.2 Integración del servicio de directorio con otros servicios.

Una de las principales aplicaciones de un servicio de directorio es que puede actuar como una herramienta aglutinadora en el diseño de aplicaciones que permitan desarrollar nuevos servicios basados en la cooperación entre distintas aplicaciones y el servicio de directorio.

El servicio de directorio puede funcionar como un servidor de autentificación proporcionando el servicio de contraseña única, así como contener información necesaria para que otros servidores puedan determinar si cierto usuario tiene permiso de acceso a determinada información.

El servicio de directorio puede usarse como repositorio en el cual distintos servidores y aplicaciones pueden guardar información común de configuración, control de accesos, etc.

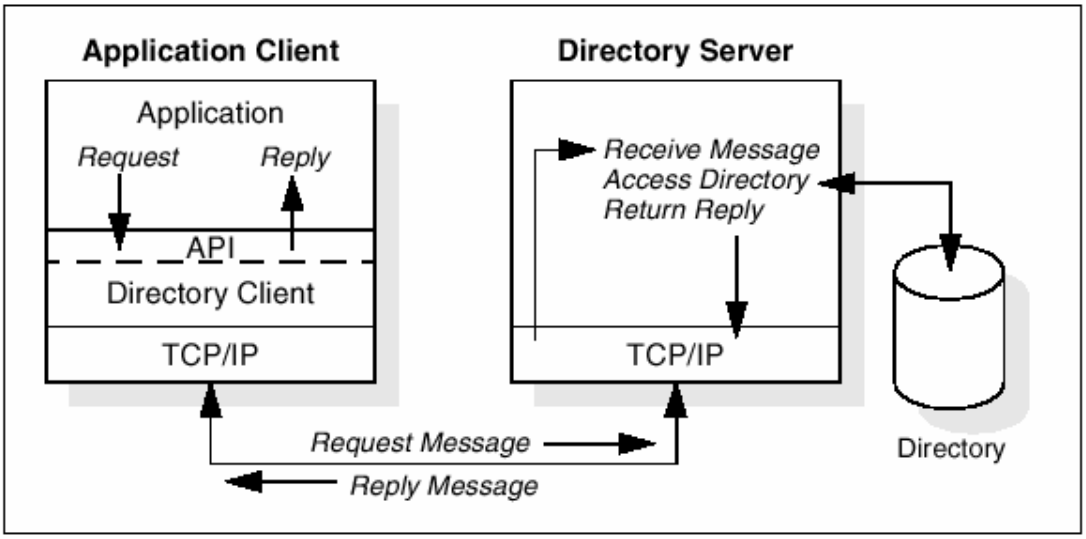
Una gran ventaja es que el directorio proporciona un protocolo estándar para gestionar la información en él presente, lo que evita tener que desarrollarlo.

2.3 Clientes y servidores de directorio

El servicio de directorio responde a un modelo cliente servidor, en el que las aplicaciones cliente contactan con el servidor y hacen una determinada petición.

Las aplicaciones que usan servicios de directorio normalmente no interactúan directamente con el servidor; para ello invocan una función construida mediante un cierto interfaz de programación, que es capaz de interactuar con el servidor y devolver el resultado.

Este proceso se muestra en la siguiente ilustración:



En ocasiones pueden intercambiarse los papeles y el servidor pasar a funcionar como cliente para solicitar a la aplicación que comenzó el proceso siendo cliente algún tipo de información necesaria para continuar con la operación, como puede ser información de autentificación.

Un “API” define un interface de programación escrito en un determinado lenguaje de programación particular, y cuyos métodos son usados para acceder a un servicio. El contenido de los mensajes manejados por los métodos que ofrece este API debe cumplir con el formato definido por el protocolo de acceso a directorio que se esté usando.

Existen APIs que permiten acceder a servidores LDAP a aplicaciones que usen C++ y Java.

2.4 Directorios distribuidos

A la hora de describir un servicio de directorio los términos local, global y distribuido son frecuentemente usados.

Un servicio de directorio ***local*** hace referencia a un servicio al que sólo tienen acceso un conjunto de usuarios localmente ubicados; un ejemplo de directorio local sería un servidor de directorio que da servicio a los usuarios de una red de área local

Un servicio de directorio ***global*** implica que el servicio da soporte a un ámbito muy amplio de usuarios, como puede ser una región, un país, o el conjunto de usuarios de Internet.

El término ***distribuido*** no tiene relación directa con los aspectos antes descritos, ya que es lo opuesto a ***centralizado.*** Un servicio de directorio distribuido es aquel en el que cooperan varios servidores para ofrecer el servicio, mientras que en el servicio centralizado es un solo servidor el que ofrece el servicio.

Puede observarse que un servicio tanto distribuido como centralizado puede ser de ámbito tanto local como global.

El servicio de directorio distribuido puede dividirse aún en ***replicado y particionado***. En un servicio replicado existe un servidor maestro y una serie de servidores esclavos que contienen la misma información que el maestro, y son periódicamente actualizados por éste; en un servicio particionado la información se encuentra dividida entre distintos servidores, es decir, que cada uno contiene un trozo del directorio, y juntos cooperan entre sí para ofrecer al usuario la información como un todo.

Estos términos también pueden aplicarse al tipo de información contenida en un directorio, según se haga referencia a objetos locales o de interés local, o bien de interés para un grupo global de usuarios.

Los directorios centralizados presentan la ventaja frente a los distribuidos de ser una solución de diseño más sencilla y de simplificar la gestión del sistema, si bien los directorios distribuidos tienen a su favor frente a los centralizados que pueden proporcionar un servicio más eficiente gracias al reparto de la carga, y una mayor robustez frente a errores, ya que si se cae un servidor el sistema puede seguir funcionando.

2.5 Seguridad en servicios de directorio

El concepto de seguridad en servicios de directorio puede aplicarse a varios niveles, ya que se hace imprescindible autentificar a los usuarios que acceden al servicio, concederles un determinado tipo de permiso y de privilegios a la hora de realizar operaciones y que la información que circula por la red durante la comunicación esté cifrada, protegiéndola así de eventuales intrusos.

En cuanto a la identificación de usuarios los protocolos de acceso a directorio normalmente tienen mensajes con campos en los que viaja información de autentificación a fin de que el servidor pueda verificar que un usuario es quien dice ser y concederle acceso al sistema, evitando así que usuarios no deseados pudieran tener acceso al directorio.

La manera de autentificar a los usuarios puede ser muy variada y depende del protocolo que soporte el servidor.

Los permisos de accesos son una serie de privilegios que el servidor concede a los usuarios autentificados y determinan el tipo de operación que el cliente puede realizar sobre la información contenida en el directorio. Algunos de estos permisos son los de búsqueda, lectura, escritura, etc.

En cuanto a la transmisión de información a través de la red durante la comunicación, se busca que ésta se haga de manera segura, para lo cual se usan protocolos que soportan algoritmos de cifrado y que a partir de una clave que comparten cliente y servidor manipulan la información para que pueda ser transmitida de forma segura.

2.6 El servicio de directorio X.500

La popularidad que alcanzan los servicios de directorio lleva al Comité Consultivo Internacional de Telecomunicaciones y Telégrafos (CCITT), que actualmente se denomina ITU-T (Unión Internacional de Telecomunicaciones, sector de estandarización de telecomunicaciones) a crear el estándar X.500 en 1988 que aportaba normalización a un tipo de servicio que estaba cobrando cada vez más importancia y que carecía de ella. Este estándar se convirtió en 1990 en el estándar ISO 9594, “Data Conmunications Network Directory” (Comunicación de datos de directorio a través de la red), recogido en las recomendaciones X.500 a X.521, aunque comúnmente sigue siendo conocido como X.500.

El servicio de directorio X.500 organiza la información de directorio en un espacio de nombres jerárquico capaz de soportar gran cantidad de información, y define potentes operaciones de búsqueda para poder recuperar con facilidad la información.

X.500 ofrece una gran funcionalidad, hasta el punto de que usado con determinadas pasarelas permiten la interacción de servicios de directorio de distintos fabricantes e incompatibles en principio.

X.500 especifica que la comunicación entre clientes y servidores debe seguir el ***protocolo de acceso a directorio*** (DAP). Dicho protocolo encaja dentro de un modelo estratificado de protocolos, y como tal requiere para su funcionamiento de toda la pila de protocolos OSI, lo cual supone un elevado consumo de recursos que no siempre están disponibles en una red.

De aquí surge la necesidad de desarrollar una interfaz de acceso al servicio X.500 que no consuma tantos recursos o un protocolo de acceso ligero.

2.7 Evolución del protocolo LDAP.

El protocolo ***LDAP*** "Lightweight Directory Access Protocol" surge como alternativa que simplifica el complejo protocolo DAP. Su principal aportación es que no requiere para su funcionamiento de toda la pila de protocolos OSI y puede funcionar sobre la popular y cada vez más usada arquitectura TCP/IP, simplificando en gran medida los recursos necesarios para el funcionamiento.

Históricamente hubo dos precursores de LDAP que aparecieron como recomendaciones publicadas por el IETF: el Servicio de Asistencia de Directorios (RFC 1202) y la especificación de protocolo DIXIE (RFC 1249); si bien fueron dos recomendaciones informativas que no fueron propuestas como estándar.

El Servicio de Asistencia de Directorios (Directory Assistante Service DAS) definía un método por el cual un cliente podía conectarse con un proxy que funcionaba bajo una arquitectura OSI y que era capaz de realizar operaciones en un directorio X.500 en representación del cliente.

DIXIE proponía un escenario parecido a DAP, si bien la interpretación de DAP era mucho más directa.

La primera versión de LDAP fue definida en la recomendación RFC 1487 (X.500 Lightweight Access Protocol) que fue reemplazada por la RFC 1777 (Lightweight Directory Access Protocol).

LDAP mejora las ideas propuestas tanto en DAS como en DIXIE y propone una implementación más neutral que simplifica en gran medida la complejidad de los clientes para potenciar el desarrollo de aplicaciones que usan servicios de directorio.

La recomendación RFC 1777 define el propio protocolo LDAP y junto con las siguientes recomendaciones define la versión 2 del protocolo LDAP (LDAP v2).

- ***RFC 1778.*** Representación mediante cadenas de sintaxis de atributos estándar.

- ***RFC 1779.*** Representación mediante cadenas de nombres distinguidos.

- ***RFC 1959.*** Formato de URLs tipo LDAP.

- ***RFC 1960.*** Representación mediante cadenas de filtros de búsqueda LDAP.

El creciente desarrollo de los sistemas de directorio lleva a la aparición de una nueva versión de LDAP. Es la versión 3 (LDAP v3) definida en la recomendación RFC 2251 que se basa en la versión 2 pero introduce una serie de novedades y actualizaciones que son recogidas por las siguientes recomendaciones.

- ***RFC 2252.*** Definición de sintaxis de atributos.

- ***RFC 2253.*** Representación de nombres distinguidos mediante cadenas codificadas en UTF-8.

- ***RFC 2254.*** Representación mediante cadenas de filtros de búsqueda LDAP.

- ***RFC 2255.*** Representación de URLs tipo LDAP.

- ***RFC 2256.*** Especificaciones sobre esquemas.

La versión tres de LDAP mejora la versión dos introduciendo los siguientes aspectos:

- ***Soporte de remisiones.*** Un servidor que no contenga la información solicitada por el cliente puede indicar al cliente que se dirija a otro servidor en el que probablemente se encuentre la información que busca.

- ***Seguridad.*** LDAP v3 permite la autentificación simple de usuarios mediante su identificador y su clave, pero además soporta mecanismos de autentificación SASL, que son un conjunto de protocolos basados en mensajes de desafío y respuesta en los cuales el servidor requiere una serie de datos al cliente para su autentificación (por ejemplo un certificado X.500). Incluso podría soportar protocolos que aún no han sido definidos siempre que cumpla con las características propias de la familia de protocolos SASL.

- ***Internacionalización.*** LDAP v3 usa UTF-8 para soportar el juego de caracteres internacionales.

- ***Extensibilidad.*** El protocolo puede evolucionar mediante la definición de nuevas operaciones (extensiones y controles) y la introducción de nuevos esquemas.

LDAP define el protocolo de comunicación entre cliente y servidor pero no define una interfaz de programación para el cliente. Actualmente existen paquetes que proporcionan esa funcionalidad (APIs) para diferentes lenguajes de programación tales como C, C++ o Java.

El API que ha alcanzado mayor popularidad es JNDI para Java. Es un API desarrollado por “Sun” y que se describe en apartados posteriores.

Actualmente “Novell” ha publicado su propio API para acceso a servicios de directorio con el nombre de JLDAP.

2.8 Arquitectura del protocolo LDAP.

El protocolo LDAP define el contenido de los mensajes intercambiados entre un cliente y un servidor LDAP. Estos mensajes contienen las solicitudes que el cliente realiza, las respuestas del servidor a estas solicitudes y el tipo de datos que se codifican en los mensajes.

El protocolo LDAP funciona sobre una arquitectura TCP/IP con lo cual además del intercambio de mensajes LDAP habrá mensajes destinados a establecer, gestionar y terminar conexiones TCP/IP entre cliente y servidor.

El diseño de aplicaciones que usen el protocolo LDAP no requiere un conocimiento profundo del contenido exacto de los mensajes y su estructura. Lo verdaderamente necesario es el modelo lógico definido por estos mensajes y sus tipos de datos, cómo se organiza el directorio, qué operaciones son posibles, cómo se protege la información, etc.

La interacción general entre un cliente y un servidor LDAP sigue la siguiente estructura:

- El cliente establece una sesión con el servidor. Esto significa que el cliente inicia el proceso de conexión para lo cual usa la dirección IP del servidor y el puerto TCP/IP en el que éste permanece a la escucha de peticiones. En esta fase de la comunicación el cliente proporciona información de autentificación como su identificador, su clave, el protocolo de autentificación que va a usarse, o el protocolo usado para encriptar la información.

- A continuación el cliente realiza operaciones sobre la información del directorio. Están permitidas operaciones de lectura y de actualización, si bien el servidor LDAP se encuentra optimizado para operaciones de búsqueda, permitiendo recuperar información de una zona determinada del directorio a partir de sus atributos.

- Cuando el cliente termina de realizar operaciones cierra la conexión con el servidor.

Existen interfaces de programación de aplicaciones (APIs) que permiten desarrollar aplicaciones capaces de interactuar con el servidor LDAP, con lo cual estos paquetes pueden considerarse como extensiones de la arquitectura LDAP. Sin embargo, el protocolo LDAP no especifica nada al respecto sobre como deberían ser estas APIs, pero sí recoge su existencia.

El protocolo LDAP incorpora de X.500 el tratamiento de la información de directorio usando el concepto de ***entrada*** . Una entrada define un objeto de directorio que tiene asociado un nombre que lo identifica dentro del directorio; es el llamado ***nombre distinguido (DN)*** que está compuesto por una secuencia de nombres llamados ***nombres distinguidos relativos (RDN)*** que nombran las partes de la ruta que permite localizar el objeto dentro del directorio.

Las entradas se organizan formando una estructura en forma de árbol que ordena las entradas jerárquicamente según sus nombres distinguidos.

Esta estructura en forma de árbol se conoce como ***árbol de información de directorio*** (***DIT***, Directory Information Tree).

Cada entrada contiene uno o más ***atributos*** que la describen. Cada atributo tiene un valor o una serie de valores que son cadenas de texto en las que se guarda la información, ya que éste es el formato usado por LDAP para contener la información de los atributos.

Una entrada de directorio describe un objeto de una determinada clase. Una ***clase objeto*** es una descripción general sobre un tipo de objeto, que contiene información sobre el tipo de atributos que el objeto puede soportar y cuándo son obligatorios u opcionales dichos atributos. Este tipo de información está contenida en el esquema de directorio.

El protocolo LDAP define una serie de operaciones que deben ser soportadas por un servicio de directorio. Estas operaciones son:

- Búsqueda de información que cumpla con un determinado criterio (***Search***).
- Añadir entradas al directorio que cumplan con las reglas del esquema (***Add***).
- Borrar una entrada existente en el directorio (***Delete***).
- Modificar el contenido de una entrada (***Modify***).
- Modificar el nombre distinguido de una entrada (***Modify_rdn***).
- Comprobar si una entrada contiene una pareja atributo/valor determinada (***Compare***).

- Establecer una conexión con el servidor (***bind***).
- Terminar una conexión de manera ordenada (***unbind***).
- Abortar una operación (***abandon***).

2.9 Las recomendaciones del protocolo LDAP.

A continuación se expone el contenido de las recomendaciones (RFCs) que describen el protocolo LDAP.

- ***RFC 2251: Lightweight Directory Access Prot ocol (v3).*** Esta recomendación es el núcleo del conjunto de recomendaciones que describen el protocolo LDAP. Define cómo se nombran las entradas, el formato de los mensajes intercambiados entre cliente y servidor, las operaciones que el cliente puede realizar y especifica que los datos deben ser codificados mediante caracteres UTF-8. Especifica que el esquema puede ser consultado para saber qué tipo de información debe contener una entrada; indica cómo usar las remisiones; también describe cómo las operaciones pueden ser modificadas mediante el uso de controles y cómo pueden crearse nuevas funciones mediante extensiones.

  La norma también describe cómo se lleva a cabo el proceso de autentificación de un usuario usando autentificación simple o basada en mecanismos SASL.

- ***RFC 2252: Lightweight Directory Access Protocol (v3). Attribute Syntax Definitions.***
  Especifica que LDAP debe usar cadenas de caracteres para la transmisión de información en mensajes LDAP. Define como se representan datos enteros, marcas de tiempo, etc.

  También define qué tipo de dato contiene un atributo y qué criterios deben seguirse a la hora de realizar comparaciones sobre dicho atributo. Toda esta información se guarda en el esquema de directorio.

- ***RFC 2253: Lightweight Directory Access Protocol (v3). UTF-8 representation of distinguished names.***
  Los nombres distinguidos son identificadores que designan un objeto del directorio. Esta recomendación especifica que los nombres distinguidos deben ser representados como cadenas. Especifica que un nombre distinguido (DN) se compone de un conjunto de nombres distinguidos relativos (RDN) separados por el carácter ",". Los nombres distinguidos relativos componen el nombre distinguido a través de los antecesores de la entrada desde la raíz del árbol, y están compuestos por una pareja atributo/valor de la entrada antecesora a la que nombran.

  Por ejemplo, el nombre distinguido "cn=John Smith,ou=Austin,o=IBM,c=US" se compone de cuatro nombres distinguidos relativos y nombra a una entrada llamada cn=John Smith que cuelga de la entrada ou=Austin, que cuelga de la entrada o=IBM, que a su vez cuelga de la raíz c=US.

- ***RFC 2254: The String representation of LDAP filtres.***
  Los filtros de búsqueda proporcionan una potente arma para realizar operaciones de búsqueda en el directorio. Esta recomendación describe cómo se construye un filtro de búsqueda. El formato del filtro especifica que los operandos deben ir entre paréntesis y precedidos del correspondiente operador.

- ***RFC 2255: The LDAP URL format.***
  Las URLs (Uniform Resource Locator) permiten localizar páginas web, ficheros y cualquier otro tipo de recurso dentro de una red. Una URL LDAP puede especificar dónde se encuentra un servidor LDAP, una entrada de un directorio, una remisión, una referencia a un objeto, etc.

  El formato de este tipo de datos es: ***ldap://dirección IP:número de puerto/nombre distinguido de la entrada.***

- ***RFC 2256: A summanry of the X.500(96) User Schema for use with LDAP v3.***
  Muchos de los objetos de esquema que un servidor LDAP debe soportar están definidos ya en X.500. Esta recomendación expone cuáles son esos objetos y cómo están estructurados. Son atributos tan usados como nombres comunes, direcciones, etc; y clases tan útiles como organizaciones, usuarios, etc.

Estas recomendaciones constituyen el núcleo del protocolo LDAP, pero existen ciertas aportaciones que incrementan la funcionalidad del protocolo, aunque muchas de ellas aún no han alcanzado el nivel de estándar y son sólo propuestas de extensiones que cada fabricante implementa de manera particular.

Entre estas propuestas se encuentran:

- ***Autentificación obligatoria.*** Es un intento de desarrollar un método de autentificación estándar que pueda estar disponible en todos los clientes y servidores y con preferencia sobre otros métodos de autentificación que operen sobre TCP/IP.

- ***Extensiones para servicios de directorio dinámicos.*** Es una extensión del protocolo que busca permitir a los clientes un acceso fiable mientras la información está siendo actualizada.

- ***Uso de códigos para distintas lenguas.*** Permite la codificación de atributos que sean naturales en la lengua del desarrollador.

- ***Extensión para el protocolo de capa de transporte seguro (TLS Transport Layer Security).*** Es una operación adicional que busca la integración del protocolo TLS en servidores LDAP.

- ***Controles para la representación de resultados.*** Son modificaciones a las operaciones ya existentes que permiten que los resultados sean devueltos por partes (páginas), ordenados alfabéticamente, etc.

- ***Referencias y remisiones.*** Una referencia permite que objetos que por alguna circunstancia como su tamaño o su formato no pueden ser almacenados en el directorio, sean apuntados desde éste con información que indique al usuario cómo acceder a ellos.

  Una remisión es información que el servidor envía al cliente indicándole dónde es probable que se encuentre la información que busca.

- ***APIs LDAP.*** Son interfaces de programación que permiten el desarrollo de aplicaciones cliente.

2.10 El modelo LDAP

Conceptualmente el protocolo LDAP se encuentra dividido en cuatro modelos funcionales que hacen referencia a los principios sobre los que está construido el protocolo.

- ***Modelo de información.***
- ***Modelo de nombrado.***
- ***Modelo funcional.***
- ***Modelo de seguridad.***

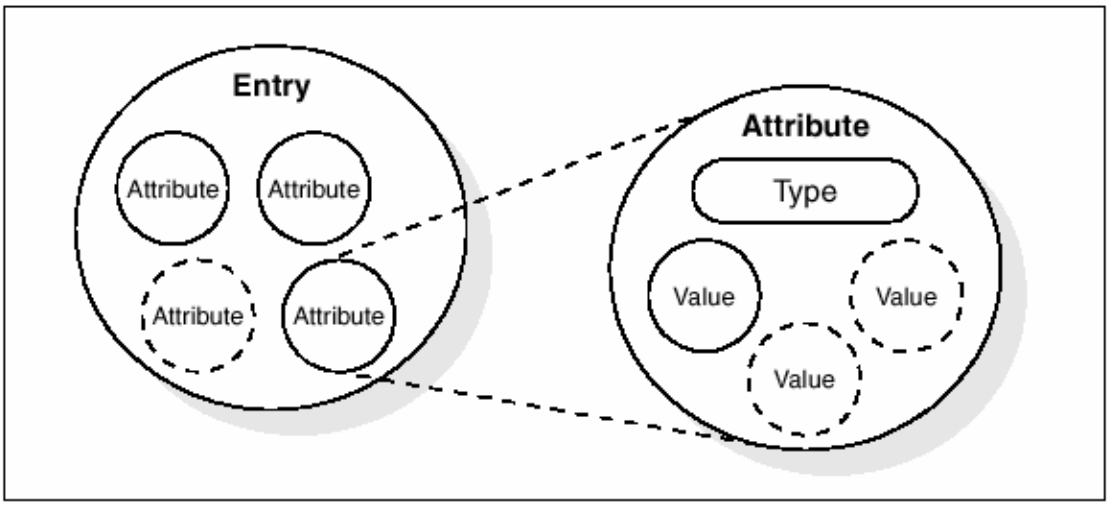
2.10.1 El modelo de información LDAP.

La unidad básica de información en un directorio LDAP es una entrada (dicho concepto representa un objeto del directorio). Las entradas se componen de un nombre que las identifica en el directorio y un conjunto de atributos.

Cada atributo consta de un identificador que contiene el tipo de atributo, es decir el tipo de dato que se guarda en él. Este identificador hace referencia a la sintaxis del atributo que especifica el tipo de valores que se almacenan en el atributo (cadenas, enteros,etc).

Un atributo puede contener múltiples valores particulares.

El modelo anteriormente descrito se ilustra a continuación:



Sobre los atributos pueden imponerse restricciones sobre el número de valores que puede contener y el tamaño de estos valores.

El esquema se encarga de organizar los atributos y sus sintaxis. El esquema define una serie de reglas que las entradas del directorio LDAP deben cumplir.

El esquema de directorio introduce el concepto de ***clase objeto*** que es el tipo concreto del objeto que se guarda en el directorio y contiene información acerca de los atributos que la entrada debe contener, así como los que podría contener.

El concepto de ***tipo de atributo*** en el contexto del esquema define la arquitectura de un atributo particular, es decir, proporciona información sobre qué tipo de dato guarda el atributo (***sintaxis del atributo***), las limitaciones que se imponen sobre él y los criterios que deben usarse a la hora de realizar comparaciones sobre dicho atributo.

Cada servidor puede publicar su propio esquema que podría en principio no ser compatible con el de otros servidores, de ahí que se haga necesaria la aparición de un estándar que recoja unas ciertas reglas a cumplir con el fin de garantizar la interoperabilidad entre servidores. Este estándar se recoge en la RFC 2252.

La información de esquema debería estar disponible para los clientes que deseen consultarla al igual que el resto de información del directorio. Esto no siempre es así, ya que existen servidores particulares que no hacen pública su información de esquema, o sólo hacen pública una parte de esa información.

Se permite que una clase objeto del esquema herede de otra, es decir, se permite que una cierta clase objeto sea construida a partir de otra, obteniendo como suya toda la información declarada en la clase objeto de la cual deriva (tipos de atributos obligatorios, opcionales, descripción, etc). Lo que hará única a la clase objeto serán las especificaciones que le sean aportadas aparte de las que ya ha heredado.

También se soporta el concepto de ***alias***, mediante el cual una clase objeto puede ser designada además de por su nombre por un identificador corto, simple y de fácil manejo para el usuario.

2.10.2 El modelo de nombrado LDAP.

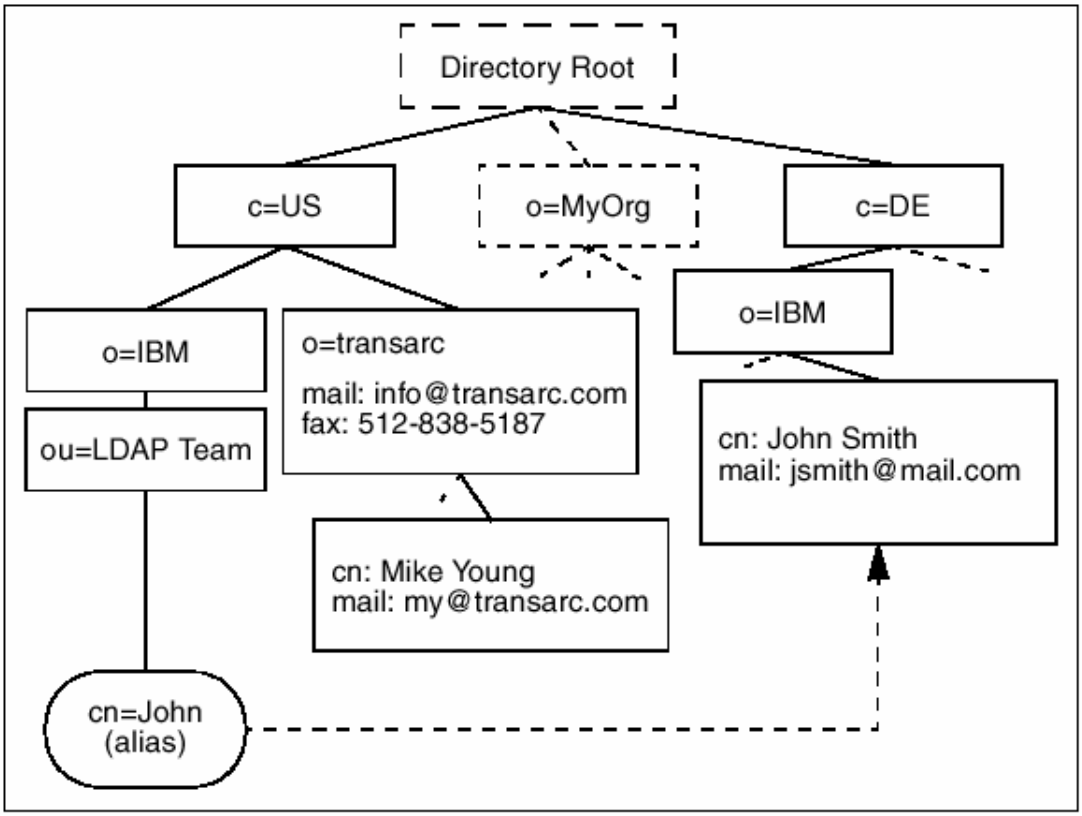
El modelo de nombrado LDAP define cómo las entradas son nombradas, identificadas y localizadas, y como se organiza la información dentro del directorio.

El protocolo LDAP especifica que la información se guarda siguiendo una estructura jerárquica arborescente llamada "***Árbol de información de directorio***" (DIT).

Las entradas se almacenan en el DIT a partir de su nombre distinguido (DN) que es una concatenación de nombres relativos distinguidos (RDN) separados por comas.

La organización y el contenido de las entradas dentro del DIT debe respetar las reglas definidas en el esquema.

La siguiente ilustración muestra un DIT de un directorio particular:



Puede observarse como las entradas se almacenan en forma de árbol. El nombre distinguido (DN) de una entrada estaría compuesto por un conjunto de nombres distinguidos relativos distinguidos (RDN) separados por comas que definen una ruta desde la raíz hasta la propia entrada. De esta forma la entrada que contiene el atributo “cn:Mike Young” sería nombrada como “cn=Mike Young, o=transarc, c=US” donde cada componente es un nombre distinguido relativo.

A nivel de codificación los atributos usados para definir un nombre distinguido son representados como una cadena decimal que contiene el identificador del objeto.

El concepto de ***alias*** dentro de la información de directorio hace referencia a una entrada que apunta a otra dentro del DIT, es decir, define un enlace simbólico entre dos entradas.

Este concepto se ilustra en el gráfico anterior en el que la entrada cn=John apunta a la entrada cn=John Smith.

Un alias está definido por la clase objeto ***alias*** que permite albergar un atributo de nombre ***aliasedObjetName*** en el cual se codifica el objeto al que se hace referencia.

El alias se codifica como el nombre distinguido de la entrada a la que apunta.

El concepto de ***referencia*** define objetos que se encuentran fuera del directorio, pero cuya presencia se encuentra registrada en alguna entrada que llegado el momento informará al cliente cómo localizar dicho objeto, es decir, una referencia a él.

Una ***remisión*** es una generalización del concepto de alias, pero en esta ocasión una entrada de un directorio apunta a otra entrada que no se encuentra en su mismo DIT, sino que se encuentra en otro DIT que puede ser mantenido por el mismo servidor o por un servidor distinto que puede encontrarse en cualquier otra ubicación física.

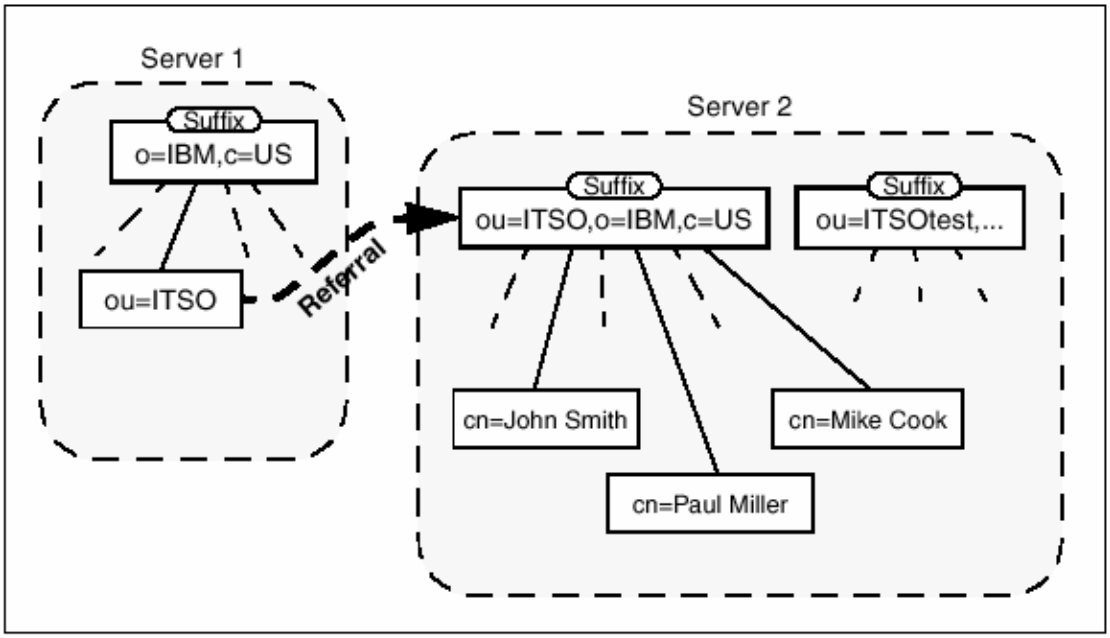
Las referencias están implementadas por la clase objeto ***referral*** que define un atributo llamado ***ref*** que contiene la referencia al objteto.

El atributo ***ref*** presenta la particularidad de que su contenido no es devuelto a no ser que se solicite explícitamente, ya que pertenece a un grupo de

atributos llamados ***operacionales*** que son mantenidos por el servidor para propósitos administrativos.

La remisión se codifica como una URL LDAP.

La siguiente ilustración muestra una referencia entre entradas de dos directorios.



Cuando un cliente realiza una consulta a un servidor LDAP la respuesta puede ser una referencia. Entonces el cliente debe decidir qué hacer con la referencia, pudiendo optar entre:

- ***Seguirla.*** En este caso el cliente inicia automáticamente una nueva conexión con el servidor al que apunta la referencia.

- ***Ignorarla.*** El cliente no hace nada y da por concluida la operación.

- ***Lanzarla.*** En este caso notifica al usuario sobre la existencia de la excepción y éste decide qué hacer.

Las distintas APIs permiten un control sobre la remisión que llega hasta el punto de permitir que las aplicaciones puedan capturar una excepción cada vez que se encuentra una remisión y decidir cómo gestionarla.

Las remisiones son una herramienta útil incluso a nivel de diseño del sistema, ya que permiten construir servidores distribuidos, en los que una entrada de cierto servidor puede apuntar a la información de otro, pero con

la particularidad de que en este caso es competencia del propio servidor la gestión de la remisión y de las comunicaciones que conlleve, quedando liberado el cliente en este sentido.

De lo anterior puede deducirse que el protocolo LDAP permite que un servidor LDAP pueda albergar más de un DIT, implementando así varios directorios con un solo servidor.

El concepto de remisión aparece propiamente definido en la versión tres de LDAP. En la versión dos las remisiones no eran explícitamente soportadas, aunque existían las llamadas respuestas de error de resultado parcial que devolvían al cliente una respuesta de error indicando que el servidor no podía llevar a cabo su consulta, pero a la vez le notificaba que la información que buscaba era probable que se encontrase en ciertas ubicaciones.

2.10.3 El modelo funcional LDAP.

Define las operaciones que soporta el protocolo LDAP independientemente del API usado por el cliente.

Las operaciones LDAP pueden ser clasificadas en tres categorías:

- ***Operaciones de búsqueda***. Incluye operaciones que realizan consultas al directorio tales como “compare” y “search”.

- ***Operaciones de actualización***. Permiten hacer modificaciones en el directorio. Incluye “add”, “modify” y “modify_rdn”.

- ***Operaciones de autentificación***. Se ejecutan para manejar aspectos de la conexión. Incluye las operaciones “bin”, “unbind” y “abandon”.

2.10.3.1 La operación LDAP_SEARCH

Es la operación más común de todas las que están definidas. Permite al usuario realizar una búsqueda sobre las entradas de una determinada zona del árbol y que sean devueltas aquellas que cumplen un cierto criterio que se especifica en un filtro de búsqueda.

Esta función utiliza para ello la capacidad de búsqueda y de lectura en la misma función.

La operación de búsqueda permite especificar qué entrada sirve de punto de partida de la búsqueda, la profundidad de la búsqueda (si se realiza en la propia entrada, en sus hijos, o en todo el subárbol que cuelga de la entrada) y la condición que deben cumplir las entradas para ser devueltas, etc.

Los parámetros que pueden ser especificados en una operación de búsqueda son:

- ***Base.*** Es el nombre distinguido de la entrada que sirve como punto de partida para la búsqueda.

- ***Ámbito.*** Indica la profundidad de la búsqueda.

- ***Filtro de búsqueda.*** Es una expresión lógica que contiene identificadores de atributos, valores, y operadores lógicos, y que expresa la condición que deben cumplir las entradas que van a ser devueltas.

  La expresión consta de un conjunto de operandos que van precedidos de un operador determinado.

  Los operadores soportados por LDAP se recogen en la siguiente tabla:

OPERADOR	***DESCRIPCIÓN***
=	Operador de igualdad
<=	Operador menor o igual
>=	Operador mayor o igual
=*	Operador de presencia
~=	Operador de aproximación.
&	Operador lógico *y*
|	Operador lógico *o*
!	Operador lógico *not*

Un ejemplo de filtro de búsqueda es:

(| (sn=Smith) (&(ou=Austin)(sn=Miller)))

Este filtro hace que se devuelvan las entradas que tengan un atributo sn con el valor Miller y un atributo ou con el valor Austin, o bien aquellas entradas que tienen un atributo sn con el valor Smith.

- ***Atributos a devolver.*** Se da una lista de los identificadores de atributos que deben ser devueltos. De esta manera se muestra el interés por un conjunto determinado de atributos de las entradas que cumplan con el filtro de búsqueda. Cabe la posibilidad de recibir sólo los identificadores de los atributos, sin sus valores particulares.

- ***Tratamiento de alias.*** Indica qué debe hacerse con los posibles alias que se encuentren en la búsqueda. Entre las posibilidades están:

  - ***Seguirlos.*** Automáticamente se desreferencia el enlace.
  - ***Ignorarlos.*** Nunca se desreferencia el enlace.
  - ***Seguirlos durante la resolución de nombres.*** Sólo se siguen los alias contenidos en la entrada base de la búsqueda.
  - ***Seguirlos tras la resolución de nombres.*** Se siguen los alias excepto los contenidos en la entrada base de la búsqueda.

- ***Limitaciones.*** Pueden imponerse restricciones sobre el número de entradas que pueden ser devueltas y el tiempo que se puede emplear en la ejecución de la operación.

Si no se encuentran entradas en la búsqueda que cumplan con el filtro se devuelve una indicación informando que no se han encontrado resultados.

2.10.3.2 La función LDAP_COMPARE

Permite comprobar si una entrada determinada tiene una pareja atributo/valor dada. En caso afirmativo devuelve un valor “verdadero”, y en caso negativo un valor “falso”.

La operación de comparación es muy parecida a la de búsqueda, aunque es mucho más simple. Se trata en esencia de una búsqueda partiendo de la entrada en cuestión en el ámbito de la propia entrada, ignorando tanto los alias como las remisiones y con un filtro de búsqueda del tipo “atributo=valor”.

En caso de que la entrada no exista la operación devuelve el valor “falso”, lo cual es una diferencia con la operación “search” que en ese caso devolvería un mensaje indicando que no ha encontrado entradas apropiadas.

2.10.3.3 La función LDAP_DELETE

Esta función permite borrar una entrada existente del directorio a partir de su nombre distinguido.

Hay que ser especialmente cuidadoso con esta función ya que permite borrar entradas que contienen hijos, dando lugar a un directorio fraccionado en dos partes, lo cual supone un grado de inconsistencia generalmente inaceptable.

En caso de que la entrada a borrar no exista se notifica al cliente ese aspecto.

2.10.3.4 La función LDAP_ADD

Esta función permite añadir una nueva entrada al directorio.

Para ello se especifica el nombre distinguido de la entrada que va a ser añadida y un conjunto de atributos con los que será inicializada.

Debe tenerse en consideración el esquema a la hora de añadir una entrada al directorio, ya que la entrada debe contener un atributo que indique la clase objeto a la que pertenece y un conjunto de atributos que cumplan con las exigencias recogidas en la definición de esa clase objeto.

2.10.3.5 La función LDAP_MODIFY_RDN

Esta función permite modificar el nombre distinguido de una entrada del directorio. Para ello se modifican uno o más nombres distinguidos relativos de los que componen el nombre distinguido de la entrada.

A esta función hay que pasarse el nombre de la entrada cuyo nombre va a ser modificado y el nuevo nombre de la entrada.

Esta función presenta una característica que es a la vez una ventaja y un inconveniente. Cambiar el último nombre relativo distinguido de una entrada cuando ésta no tiene hijos (es una hoja del árbol) es una simple modificación en el nombre de una entrada. En cambio si se modifica un nombre distinguido relativo intermedio del nombre distinguido además de un cambio en el nombre de la entrada se produce un cambio de contexto, es decir que la entrada es trasladada a otro lugar del árbol en el que cuelga de la nueva entrada que ha sido especificada como nombre relativo distinguido en su nombre relativo.

Hay que tener un especial cuidado a la hora de realizar cambios de contexto ya que si la entrada tiene hijos éstos no son trasladados para poder seguir derivando de la misma entrada, sino que quedan descolgados de la estructura, produciéndose un fraccionamiento en dos partes del árbol de directorio, lo cual provoca una inconsistencia de la información que puede ser intolerable para muchas aplicaciones, sobre todo aquellas que intenten acceder a esa zona del árbol.

El límite del cambio de contexto se encuentra en el propio servidor, ya que una entrada no puede ser trasladada fuera de los límites del servidor que la contiene.

2.10.3.6 La función LDAP_BIND

Establece una conexión entre el cliente y el servidor. Esta operación permite al cliente autentificarse mediante alguno de los métodos que soporte el servidor LDAP.

El protocolo LDAP v3 permite el acceso al directorio de usuarios anónimos, es decir, no identificados. LDAP v2 no lo permite.

Las operaciones que estos usuarios puedan realizar sobre la información del directorio dependerán de los privilegios de acceso que se le concedan a los clientes autentificados.

2.10.3.7 La función LDAP_UNBIND

Solicita el cierre de una conexión LDAP entre cliente y servidor de manera ordenada.

2.10.3.8 La función LDAP_ABANDON

Permite al cliente solicitar al servidor que aborte una operación que se encuentra en curso.

2.10.3.9 La función LDAP_MODIFY

Esta función permite al usuario modificar el contenido de una entrada, es decir, sus atributos.

Para ello el cliente se autentifica e indica la entrada sobre la que desea realizar las modificaciones.

Dichas modificaciones pueden consistir en añadir un nuevo atributo, añadir un valor nuevo a un atributo ya existente, eliminar un atributo, eliminar un valor particular de un atributo, y reemplazar todos los valores particulares de un atributo por un único valor particular.

En esta operación pueden especificarse varias modificaciones simultáneas. En ese caso, o se realizan todas las modificaciones, o no se realiza ninguna.

2.10.3.10 Controles y operaciones extendidas.

Los controles y las operaciones extendidas son elementos que permiten la evolución y el desarrollo del protocolo sin tener que cambiarlo.

Los ***controles*** modifican el comportamiento de una operación, y se añaden al final del mensaje de operación del protocolo.

Se pasan como parámetros a las funciones del API que está siendo usado por las aplicaciones clientes.

Un importante control es el manejo de referencias. Existen otros controles que aún no están aceptados dentro del estándar y tienen sólo la categoría de propuesta, tales como la búsqueda persistente que permite a un oyente registrar los cambios que se producen en el directorio, el control “sort” que devuelve las entradas ordenadas alfabéticamente, etc. Estos controles son implementados algunos fabricantes particulares de servidores LDAP.

Un control tiene asociado un identificador de objeto que designa al propio control, un campo de que mantiene los parámetros para que puedan ser usados por el control y un nivel crítico.

Este nivel crítico es un valor lógico, que si es verdadero hace que si el servidor soporta dicho control lo tenga en cuenta a la hora de realizar la operación, pero si no lo soporta se vea obligado a rechazar la petición.
Si el nivel crítico tiene el valor lógico falso el servidor puede ignorar el control si no lo soporta y seguir adelante con la operación.

Una ***extensión u operación extendida*** permite desarrollar nuevas funciones que un servidor puede soportar sin tener que cambiar por ello el protocolo.
Una extensión consta de una pareja de mensajes que son una solicitud de operación extendida y una respuesta a una operación extendida.

El mensaje del protocolo para operaciones extendidas consta de un identificador de una cadena decimal que se usa para identificar la operación extendida y una cadena específica que contiene los datos de la operación.

2.10.4 El modelo de seguridad LDAP.

El modelo de seguridad hace referencia a la autentificación de los usuarios y a al protección de la información del directorio.

Está basado en la operación “bind” en la cual el cliente especifica el protocolo que va a usar para autentificarse. Por tanto existen tantas operaciones “bind” como protocolos de autentificación sean soportados ya que darán paso en algunos casos a un intercambio de información adicional requerida por el protocolo específico de autentificación que se esté usando.

Un posible mecanismo de autentificación soportado por LDAP es aquel en el que el cliente envía al servidor su identidad compuesta por una cadena que contiene su nombre distinguido y también su clave de acceso. Este método de acceso es fácil de implementar y usar pero presenta el problema de que la información de autentificación se transmite por la red de forma no segura, con lo cual esta información puede ser accedida en un momento dado por intrusos.

El protocolo LDAP v2 permite la autentificación mediante el protocolo Kérberos pero este tipo de autentificación ha sido sustituida en LDAP v3 por el soporte de mecanismos SASL (Simple Authentication and Security Layer) que son un conjunto de protocolos entre los que se encuentran Kérberos, GSSAPI (“Generic Secure Service API” pretendió ser un API genérico para el uso de este tipo de mecanismos, pero aún no está lo suficientemente desarrollada. Entre los mecanismos que sí incorpora está kerberos), CRAM MD-5, EXTERNAL, DIGEST, etc.

Esta familia de protocolos se basa en una serie de desafíos y respuestas que se intercambian cliente y servidor.

Esta característica permite incluso el uso de futuros protocolos que se desarrollen bajo las especificaciones de SASL.

Adicionalmente en LDAP v3 existen operaciones extendidas de protocolo que no son más que operaciones extendidas normales que hacen referencia al protocolo de seguridad usado. Se encuentra definida una operación extendida que se encarga de activar el protocolo TLS (Transport Layer Security) que permite que los mensajes intercambiados entre cliente y servidor sean cifrados con el fin de llevar a cabo una transmisión segura de información.

El protocolo TLS es una mejora del protocolo de seguridad SSL (Socket Security Layer) ya que introduce un mayor número de algoritmos de cifrado.

El protocolo TLS tiene un mecanismo que le permite comunicarse con servidores SSL compatibilizando así ambos protocolos.

Una vez que el usuario ha accedido al sistema entran en juego los mecanismos de control de acceso que conceden unos ciertos privilegios a los usuarios autentificados según esté especificado en el funcionamiento del servidor.

2.11 Información sobre el servidor.

El protocolo LDAP v3 especifica que el servidor debe contener información sobre él mismo. Esta información debe encontrarse en una entrada llamada "***root DSE***" cuyo nombre distinguido es una cadena vacía.

Esta entrada contiene atributos que definen al servidor y que pueden ser devueltos cuando operaciones de búsqueda lo requieran.
Esta información contiene datos acerca de:

- Los directorios que alberga el servidor. Los distintos DIT que almacena el servidor son designados mediante el nombre ***suffix*** o ***contexto de nombrado***.

- El nombre distinguido de una entrada especial que representa la raíz del árbol de esquema.

- La versión del protocolo LDAP soportado. El cliente LDAP debe determinar la versión del protocolo que soporta el servidor, para lo cual puede consultar esta propiedad. La versión del protocolo a usar la indica el cliente en la operación bind. Si no se indica ninguna versión se toma la tres por defecto.

- El conjunto de operaciones extendidas que soporta el servidor.

- El conjunto de controles LDAP que soporta el servidor.

- Una lista con los mecanismos SASL soportados.

- Una lista de servidores alternativos.

2.12 Seguridad en LDAP.

El concepto de seguridad es un aspecto importante en los sistemas de directorio LDAP, ya que se plantea el problema de la transmisión segura frente a intrusos de información especialmente sensible y confidencial como información de autentificación.

El problema de la seguridad también engloba la autentificación, es decir, verificar que un usuario es quien dice ser.

En los sistemas de directorio LDAP el término seguridad cubre los siguiente aspectos.

- ***Autentificación.***

- ***Integridad de la informción.*** Trata de asegurar que la información que llega a un punto de la comunicación sea la misma que el otro punto envió.

- ***Confidencialidad.*** Para ello se encripta la información.

- ***Autorización.*** Establece qué permisos tiene un usuario. Este aspecto no se encuentra recogido en el estándar LDAP y cada servidor particular implementa su propia política de autorización.

2.13 Mecanismos de seguridad en LDAP.

El protocolo LDAP define tres de mecanismos de autentificación:

- ***Autentificación anónima.***
- ***Autentificación simple.***
- ***Métodos SASL.***

El mecanismo de seguridad se negocia durante la fase de establecimiento de la comunicación.

2.13.1 Autentificación anónima.

Es la manera trivial de autentificarse, pero está soportada sólo por LDAP v3. Se permite que usuarios anónimos accedan al sistema cuando la información contenida en él es de uso general y está dirigida a todos los clientes potenciales posibles.

Una vez autentificado el cliente, entran en juego los controles de acceso específicos que el servidor haya definido para clientes anónimos.

2.13.2 Autentificación simple.

Es el mecanismo de autentificación más simple, aunque es uno de los más sencillos y fáciles de usar, siendo un mecanismo muy común en diversas familias de protocolos como HTTP.

Cuando se usa el protocolo de autentificación simple en LDAP el cliente le envía al servidor un nombre distinguido que lo identifica y una clave de acceso. El cliente se considera autentificado si el nombre distinguido y la clave se corresponden con la información de autentificación que contiene el servidor.

Presenta la desventaja de que la información no está protegida frente a intrusos.

Esto puede solventarse estableciendo comunicaciones sobre un protocolo de transmisión segura como TLS o SSL.

2.13.3 Autentificación SASL.

La autentificación SASL (Simple Authentication and Security Layer) engloba una serie de mecanismos de autentificación que pueden ser añadidos a protocolos orientados a conexión.

Los mecanismos SASL están recogidos en el estándar RFC 2222, y son incorporados a LDAP en la versión tres del protocolo ya que en la versión dos sólo se soportaba un tipo de mecanismo incluido en esta familia como lo es el mecanismo Kérberos.

Cada protocolo que intente usar un mecanismo SASL debe ser extendido con un comando que soporte el método y con mensajes capaces de transportar la información de autentificación.

La información que debe ser intercambiada depende del tipo de mecanismo utilizado, aunque en LDAP los siguientes campos siempre deben ser inicializados:

- ***Nombre distinguido.*** Es una cadena que contiene el nombre distinguido que identifica al cliente.

- ***Mecanismo.*** Identifica el método de autentificación que va a usarse. Los métodos SASL que son usados en LDAP son:

  - ***Kérberos v4.***
  - ***Kérberos v5.***
  - ***S/key.***
  - ***GSSAPI.***
  - ***CRAM-MD5.***
  - ***EXTERNAL.***
  - ***DIGEST.***

- ***Credenciales.*** Contiene información que identifica al usuario. El tipo de información contenida en este parámetro va desde cadenas de texto hasta tipos complejos de datos como certificados X.509.

La autentificación basada en SASL permite el desarrollo de nuevos mecanismos de autentificación que serán usados de la misma manera que los ya existentes.

Es evidente que para usar un mecanismo de autentificación SASL el servidor debe soportarlo y el cliente debe tener la información y los métodos necesarios para implementarlo.

2.14 URLs en LDAP

Las URLs (Uniform Resource Locator) proporcionan una forma útil y simple de localizar recursos en una red. Una URL es un identificador codificado en un cierto formato dependiente del protocolo en el que se use que identifica recursos en la red.

El uso de URLs está tan aceptado y común que se encuentra presente en una gran diversidad de protocolos, algunos tan usados como http. Por ejemplo http://www.ibm.com/Products/index.html en la cual se identifica una página web determinada.

También se encuentra presente en el protocolo FTP como por ejemplo ftp://ds.internic.net/rfc/rfc2255.txt en el que el recurso nombrado es un fichero de texto.

La importancia adquirida por el protocolo LDAP ha llevado a la definición de URLs que identifique recursos LDAP (RFC 2255). De esta manera pueden usarse URLs para identificar servidores, nombrar entradas, codificar remisiones o referencias, etc.

LDAP define también el uso de URLs para realizar operaciones de búsqueda en el directorio.

Una URL LDAP comienza con el identificador "***ldap://***" que indica una conexión LDAP normal sobre TCP/IP o bien con el identificador ***"ldaps://"*** que indica que la conexión LDAP usa el protocolo SSL.
El formato general de una URL LDAP es:

ldap[s]://[<host>[:<port>]] [/ [<dn> [? [<attributes>] [? [<scope>] \ [? [<filter>] [? <extensions>]]]]]]

El significado de los parámetros se expone a continuación:

- ***Host.*** El nombre o la dirección IP de la máquina en la que se encuentra el servidor LDAP.

- ***Port.*** El número de puerto TCP/IP en el que se encuentra funcionando el servidor LDAP. El puerto por defecto es el 389 y para LDAP sobre SSL el 636.

- ***Dn.*** El nombre distinguido de la entrada del directorio nombrada, que será interpretado como entrada base si se especifican en la URL parámetros de búsqueda.

El resto de parámetros tienen sentido si se usa la URL LDAP en una operación de búsqueda, para lo cual se incluye el carácter "?".

- ***Attributes.*** Una lista de elementos separados por comas. Cada elemento es el identificador de un atributo que debe ser devuelto si se realiza una operación de búsqueda.

- ***Scope.*** Es el ámbito en el que se realiza la búsqueda. Sus posibles valores son:

  - ***Base.*** Realiza la búsqueda en la propia entrada.
  - ***One.*** Realiza la búsqueda en los hijos de la entrada.
  - ***Sub.*** Realiza la búsqueda en el subárbol que cuelga de la entrada.

- ***Filter.*** Es el filtro de búsqueda que debe usarse.

- ***Extensions.*** Permite añadir extensiones a la operación de búsqueda.

2.15 Elementos del protocolo LDAP.

El protocolo LDAP se describe usando el lenguaje de especificación ASN.1 (Abstrac Syntax Notation 1) y la transferencia de la información se basa en un subconjunto de las reglas básicas de codificación ASN.1.

Para poder soportar futuras extensiones en el protocolo se establece que los servidores y clientes deben ignorar los elementos codificados cuyas etiquetas desconozcan.

A continuación se muestra el formato de una unidad de datos del protocolo, PDU (Protocol Data Unit) en ASN.1, así como el conjunto de tipos de datos definidos.

El formato de una PDU LDAP es:

```
Lightweight-Directory-Access-Protocol-V3 DEFINITIONS
        IMPLICIT TAGS ::=

        BEGIN

        LDAPMessage ::= SEQUENCE {
                messageID       MessageID,
                protocolOp      CHOICE {
                        bindRequest     BindRequest,
                        bindResponse    BindResponse,
                        unbindRequest   UnbindRequest,
                        searchRequest   SearchRequest,
                        searchResEntry  SearchResultEntry,
                        searchResDone   SearchResultDone,
                        searchResRef    SearchResultReference,
                        modifyRequest   ModifyRequest,
                        modifyResponse  ModifyResponse,
                        addRequest      AddRequest,
                        addResponse     AddResponse,
                        delRequest      DelRequest,
                        delResponse     DelResponse,
                        modDNRequest    ModifyDNRequest,
                        modDNResponse   ModifyDNResponse,
                        compareRequest  CompareRequest,
                        compareResponse CompareResponse,
                        abandonRequest  AbandonRequest,
                        extendedReq     ExtendedRequest,
                        extendedResp    ExtendedResponse },
                 controls       [0] Controls OPTIONAL }

        MessageID ::= INTEGER (0 .. maxInt)

        maxInt INTEGER ::= 2147483647 -- (2^^31 - 1) --

        LDAPString ::= OCTET STRING

        LDAPOID ::= OCTET STRING

        LDAPDN ::= LDAPString

        RelativeLDAPDN ::= LDAPString
```

```
AttributeType ::= LDAPString

AttributeDescription ::= LDAPString

AttributeDescriptionList ::= SEQUENCE OF
                              AttributeDescription

AttributeValue ::= OCTET STRING

AttributeValueAssertion ::= SEQUENCE {
      attributeDesc   AttributeDescription,
      assertionValue  AssertionValue }

AssertionValue ::= OCTET STRING

Attribute ::= SEQUENCE {
      type    AttributeDescription,
      vals    SET OF AttributeValue }

MatchingRuleId ::= LDAPString

LDAPResult ::= SEQUENCE {
      resultCode      ENUMERATED {
            success                   (0),
            operationsError           (1),
            protocolError             (2),
            timeLimitExceeded         (3),
            sizeLimitExceeded         (4),
            compareFalse              (5),
            compareTrue               (6),
            authMethodNotSupported    (7),
            strongAuthRequired        (8),
                  -- 9 reserved --
            referral                  (10),  -- new
            adminLimitExceeded        (11),  -- new
            unavailableCriticalExtension (12),  -- new
            confidentialityRequired   (13),  -- new
            saslBindInProgress        (14),  -- new
            noSuchAttribute           (16),
            undefinedAttributeType    (17),
            inappropriateMatching     (18),
            constraintViolation       (19),
            attributeOrValueExists    (20),
```

```
                invalidAttributeSyntax       (21),
                        -- 22-31 unused --
                noSuchObject                 (32),
                aliasProblem                 (33),
                invalidDNSyntax              (34),
                -- 35 reserved for undefined isLeaf --
                aliasDereferencingProblem    (36),
                        -- 37-47 unused --
                inappropriateAuthentication  (48),
                invalidCredentials           (49),
                insufficientAccessRights     (50),
                busy                         (51),
                unavailable                  (52),
                unwillingToPerform           (53),
                loopDetect                   (54),
                        -- 55-63 unused --
                namingViolation              (64),
                objectClassViolation         (65),
                notAllowedOnNonLeaf          (66),
                notAllowedOnRDN              (67),
                entryAlreadyExists           (68),
                objectClassModsProhibited    (69),
                        -- 70 reserved for CLDAP --
                affectsMultipleDSAs          (71), -- new
                        -- 72-79 unused --
                other                        (80) },
                -- 81-90 reserved for APIs --
        matchedDN       LDAPDN,
        errorMessage    LDAPString,
        referral        [3] Referral OPTIONAL }

Referral ::= SEQUENCE OF LDAPURL

LDAPURL ::= LDAPString -- limited to characters permitted in
                                URLs

Controls ::= SEQUENCE OF Control

Control ::= SEQUENCE {
        controlType             LDAPOID,
        criticality             BOOLEAN DEFAULT FALSE,
        controlValue            OCTET STRING OPTIONAL }
```

```
BindRequest ::= [APPLICATION 0] SEQUENCE {
        version                 INTEGER (1 .. 127),
        name                    LDAPDN,
        authentication          AuthenticationChoice }

AuthenticationChoice ::= CHOICE {
        simple                  [0] OCTET STRING,
                                -- 1 and 2 reserved
        sasl                    [3] SaslCredentials }

SaslCredentials ::= SEQUENCE {
        mechanism               LDAPString,
        credentials             OCTET STRING OPTIONAL }

BindResponse ::= [APPLICATION 1] SEQUENCE {

COMPONENTS OF LDAPResult,
serverSaslCreds    [7] OCTET STRING OPTIONAL }

UnbindRequest ::= [APPLICATION 2] NULL

SearchRequest ::= [APPLICATION 3] SEQUENCE {
        baseObject      LDAPDN,
        scope           ENUMERATED {
                baseObject              (0),
                singleLevel             (1),
                wholeSubtree            (2) },
        derefAliases    ENUMERATED {
                neverDerefAliases       (0),
                derefInSearching        (1),
                derefFindingBaseObj     (2),
                derefAlways             (3) },
        sizeLimit       INTEGER (0 .. maxInt),
        timeLimit       INTEGER (0 .. maxInt),
        typesOnly       BOOLEAN,
        filter          Filter,
        attributes      AttributeDescriptionList }

Filter ::= CHOICE {
        and             [0] SET OF Filter,
        or              [1] SET OF Filter,
        not             [2] Filter,
        equalityMatch   [3] AttributeValueAssertion,
```

```
        substrings      [4] SubstringFilter,
        greaterOrEqual  [5] AttributeValueAssertion,
        lessOrEqual     [6] AttributeValueAssertion,
        present         [7] AttributeDescription,
        approxMatch     [8] AttributeValueAssertion,
        extensibleMatch [9] MatchingRuleAssertion }

SubstringFilter ::= SEQUENCE {
        type            AttributeDescription,
        -- at least one must be present
        substrings      SEQUENCE OF CHOICE {
                initial [0] LDAPString,
                any     [1] LDAPString,
                final   [2] LDAPString } }

MatchingRuleAssertion ::= SEQUENCE {
        matchingRule    [1] MatchingRuleId OPTIONAL,
        type            [2] AttributeDescription OPTIONAL,
        matchValue      [3] AssertionValue,
        dnAttributes    [4] BOOLEAN DEFAULT FALSE }
SearchResultEntry ::= [APPLICATION 4] SEQUENCE {
        objectName      LDAPDN,
        attributes      PartialAttributeList }

PartialAttributeList ::= SEQUENCE OF SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }

 SearchResultReference ::= [APPLICATION 19] SEQUENCE
                                                OF   LDAPURL

SearchResultDone ::= [APPLICATION 5] LDAPResult

ModifyRequest ::= [APPLICATION 6] SEQUENCE {
        object          LDAPDN,
        modification    SEQUENCE OF SEQUENCE {
                operation       ENUMERATED {
                                  add     (0),
                                  delete  (1),
                                  replace (2) },
                modification    AttributeTypeAndValues } }
```

```
AttributeTypeAndValues ::= SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }

ModifyResponse ::= [APPLICATION 7] LDAPResult

AddRequest ::= [APPLICATION 8] SEQUENCE {
        entry           LDAPDN,
        attributes      AttributeList }

AttributeList ::= SEQUENCE OF SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }

AddResponse ::= [APPLICATION 9] LDAPResult

DelRequest ::= [APPLICATION 10] LDAPDN

DelResponse ::= [APPLICATION 11] LDAPResult

ModifyDNRequest ::= [APPLICATION 12] SEQUENCE {
        entry           LDAPDN,
        newrdn          RelativeLDAPDN,
        deleteoldrdn    BOOLEAN,
        newSuperior     [0] LDAPDN OPTIONAL }

ModifyDNResponse ::= [APPLICATION 13] LDAPResult

CompareRequest ::= [APPLICATION 14] SEQUENCE {
        entry           LDAPDN,
        ava             AttributeValueAssertion }

CompareResponse ::= [APPLICATION 15] LDAPResult

AbandonRequest ::= [APPLICATION 16] MessageID

ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
        requestName      [0] LDAPOID,
        requestValue     [1] OCTET STRING OPTIONAL }
```

```
ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
        COMPONENTS OF LDAPResult,
        responseName     [10] LDAPOID OPTIONAL,
        response         [11] OCTET STRING OPTIONAL }

END
```

La función de un mensaje LDAP es proporcionar una envoltura que contenga los campos necesarios que mantienen los parámetros usados en una conexión LDAP.

Si el servidor LDAP recibe una PDU en la cual no es capaz de identificar la etiqueta LDAPMessage, será incapaz de leer el identificador del mensaje, la operación a realizar y cualquier otro campo del mensaje; en este caso deberá devolver al cliente un mensaje de desconexión con el correspondiente código de error de protocolo y cerrar la conexión.

En otros casos en los que el servidor no pueda analizar correctamente el mensaje recibido, pero sí ha determinado la etiqueta LDAPMessage, deberá devolver un mensaje indicando el error al cliente, pero en este caso no cerrará la conexión.

Si es el cliente quien recibe una PDU del servidor que no puede analizar debe descartarla y si es necesario terminar abruptamente la conexión.

- El campo "***MessageID"*** es un identificador que sirve para marcar una pareja solicitud/respuesta. La PDU que devuelve el servidor como respuesta a una solicitud debe contener en este campo el mismo identificador que la petición que originó la respuesta para que el cliente pueda asociarlas.

  Este campo debe tener un valor distinto al de todas las solicitudes que han sido enviadas desde el comienzo de la sesión de la cual el mensaje que la contiene forma parte, salvo que se reciba la respuesta a la petición, en cuyo caso el identificador puede volver a ser usado.

- El tipo "***LDAPString"*** identifica cadenas representadas por el tipo ***OCTET STRING*** de ASN.1 que siguen la codificación UTF-8.

- El tipo ***"LDAPOID"*** se aplica a cadenas que siguen la codificación UTF-8 y cuyo contenido es la representación decimal de un identificador de objeto, como por ejemplo la cadena 1.3.6.1.4.1.1466.1.2.3.

- Los tipos "***LDAPDN***" y "***RelativeLDAPDN***" representan cadenas que contienen nombres distinguidos y nombres distinguidos relativos respectivamente.

- El tipo ***"Attribute type"*** es una cadena cuyo contenido es el identificador del atributo al que representa. Cada tipo de atributo tiene un identificador de objeto único asignado.

  Un tipo de atributo podría tener varios identificadores de atributo. El identificador de atributo debe comenzar por una letra y sólo puede contener letras ASCII, caracteres decimales y guiones. Los identificadores de atributo son además insensibles a mayúsculas.

- El tipo ***"AttributeDescription"*** es una generalización del tipo "AttributeType". Tiene la misma declaración en ASN.1 que él pero soporta opciones adicionales. Se compone de un nombre identificador de atributo y de una lista de opciones separadas de él por el carácter ";".

  Las opciones proporcionan información adicional sobre la naturaleza del atributo. La norma define la opción ***"binary"*** aunque existe la posibilidad de que nuevas opciones sean definidas por organismos de estandarización.

  Si un servidor recibe un tipo de dato de esta clase con una opción que no soporta debe tratarlo como un tipo de dato no conocido.

- El tipo de dato "AttributeDescriptionList" contiene una lista de cero o más tipos de atributo. El contener cero tipos tiene un significado especial dentro de la operación de búsqueda.

- La opción "***binary"*** redefine cualquier tipo de codificación definida para un atributo y en su lugar transfiere los atributos como un valor binario codificado a partir de las Reglas Básicas de Codificación. Cliente y servidor deben ser conscientes del formato de codificación del atributo a la hora de solicitarlo y devolverlo absolutamente, ya que en el caso de que el cliente pidiese al servidor un atributo en formato binario y éste tenga almacenado el atributo en un formato

sencillo y sea incapaz de convertirlo a formato binario tratará el tipo de atributo solicitado como un tipo desconocido.

La principal utilidad de la opción "binary" está en la transmisión de tipos de datos ASN.1 complejos que son requeridos por el cliente, tales como certificados, objetos Java serializables, etc.

- El tipo ***"AttributeValue"*** contiene una cadena que codifica un valor particular de un atributo, pudiendo estar este en formato binario si se usa la opción binary.

  En principio el tamaño de este tipo de dato no tiene impuesto ningún límite. Las sintaxis de los valores contenidos en este tipo de datos deben respetar las reglas impuestas en el esquema y las sintaxis definidas ya que en caso contrario serán ignorados.

- El tipo ***"AttributeValueAssertion***" contiene un tipo de atributo asociado a una regla de comparación. El valor usado para las comparaciones se encuentra representado por el tipo "A***ssertionValue".***

- El tipo ***"Attribute"*** representa los atributos de las entradas del directorio. Consta de un tipo de atributo y de uno o más valores particulares de dicho atributo. No se permiten valores duplicados, y éstos no siguen un orden fijo dentro del conjunto.

- El tipo ***"MatchingRuleId"*** contiene un identificador de objeto que designa un cierto criterio que debe ser usado para la realización de comparaciones en operaciones de búsqueda en el directorio. Esta comparación se realiza sobre el tipo AssertionValue. Las reglas de comparación que soporta un servidor deben ser publicadas junto con la información referente al servidor.

- El tipo de dato ***"LDAPResult"*** se usa para que el servidor envíe al cliente mensajes de error.

  Los campos del 16 al 21 indican problemas con los atributos, los campos 32, 33, 34, 36 indican problemas con los nombres, del 48 al 50 problemas de seguridad, del 51 al 54 problemas con el servicio y del 64 al 69 y el 71 problemas de actualización.

- El campo ***"errorMessage"*** de esta estructura se usa para devolver al cliente una cadena legible (en texto claro) que informa sobre el diagnóstico del error, si bien este tipo de mensajes no están estandarizados, por lo cual en ocasiones el servidor podría optar por devolver una cadena vacía.
  El campo ***"MatchedDN"*** contendrá el último nombre de la entrada que se ha desreferenciado en caso de encontrar un alias o una remisión y que se produzca un error durante el proceso. En caso de que no haya enlaces del tipo anterior se devuelve parte del nombre que estaba siendo desreferenciado.

- Un error de referencia indica que el servidor no contiene la entrada buscada para lo cual envía al cliente un mensaje con el formato anterior en el cual guarda en el campo ***"resultCode"*** el valor referral y en el campo ***"referral"*** la URL LDAP que identifica al destino. El cliente que recibe la URL iniciará si lo desea una nueva conexión con el servidor que le indique la URL, se posicionará en la entrada especificada por la URL (si la contiene) y usará el filtro de búsqueda impuesto por la URL (si lo contiene).

- El tipo de dato ***"Controls"*** es una secuencia del tipo ***"Control"*** que representa información adicional que se envía con una petición y permite mejorar el comportamiento de la operación que va a realizarse.

  El tipo "Control" contiene un identificador de objeto que identifica al control evitando así problemas de nombres con futuros controles. El campo ***"Criticality"*** contiene un valor lógico que en caso de ser verdadero obligará al servidor a realizar la operación sólo si puede llevarla a cabo junto con el control (control crítico); en caso de contener el valor falso el servidor podrá llevar a cabo la operación ignorando el control si no lo soporta. El campo ***"controlValue"*** indica el formato del valor contenido en el control.

2.15.1 El mensaje BIND_REQUEST.

Permite el establecimiento de una conexión con el servidor LDAP y también llevar a cabo el proceso de autentificación del usuario cliente.

El formato de este mensaje es:

```
BindRequest ::= [APPLICATION 0] SEQUENCE {
        version             INTEGER (1 .. 127),
        name                LDAPDN,
        authentication      AuthenticationChoice }

AuthenticationChoice ::= CHOICE {
        simple              [0] OCTET STRING,
                            -- 1 and 2 reserved
        sasl                [3] SaslCredentials }

SaslCredentials ::= SEQUENCE {
        mechanism           LDAPString,
        credentials         OCTET STRING OPTIONAL }
```

El significado de los campos se expone a continuación:

- ***Versión.*** El cliente especifica la versión del protocolo que va a usarse. No existe fase de negociación propiamente dicha para especificar este parámetro; el cliente debe conocer de antemano la versión soportada por el servidor.

- ***Name.*** El nombre de la entrada de directorio con la cual el cliente quiere ser identificado, o bien una cadena vacía en caso de usar autentificación anónima.

- ***Authentication.*** Información usada para identificar al usuario, por ejemplo una clave. El uso de esta información podría verse afectado por factores externos a la información contenida en este mensaje como datos referentes a otros servicios de autentificación.

En algunos mecanismos de autentificación SASL puede ser necesario enviar varios “mensajes bind” para intercambiar información con el servidor.

Si durante este proceso el cliente desea terminar el proceso de autentificación debe además cerrar la conexión subyacente.

Durante este proceso no está permitido que el cliente envíe solicitudes para realizar operaciones.

El proceso de autentificación puede ser abortado si durante el intercambio de mensajes el cliente cambia el tipo de mecanismo, la información de autentificación que estaba siendo usada, etc.

El mensaje contiene un campo en el que se especifica el mecanismo de autentificación que por defecto será el simple.

En caso usar autentificación SASL debe proporcionarse una cadena que identifique el tipo de mecanismo e información para la autentificación en el campo "credentials".

Cuando finaliza el proceso de autentificación el servidor envía al cliente un mensaje ***"BindResponse"*** con un campo que indica el fin del proceso.

2.15.2 El mensaje BIND_RESPONSE.

Este mensaje contiene información enviada por el servidor al cliente sobre el estado de la operación de conexión y autentificación.

El formato de este mensaje es:

```
BindResponse ::= [APPLICATION 1] SEQUENCE {
        COMPONENTS OF LDAPResult,
        serverSaslCreds    [7] OCTET STRING OPTIONAL }
```

Si la operación se ha realizado correctamente el campo ***"resultCode"*** contendrá un mensaje de éxito y en caso contrario información sobre el tipo de problema por el cual la operación no ha podido ser completada.

Si el cliente recibe un mensaje de este tipo notificándole que la versión del protocolo usada no es soportada por el cliente deberá cerrar la conexión y volver a iniciarla correctamente, ya que no se permite negociación en esta fase de la conexión.

El campo ***"serverSaslCreds"*** permite al cliente autentificar al servidor cuando el mecanismo lo requiera. Este campo podría no estar presente si el mecanismo SASL usado no requiere información de autentificación del servidor.

2.15.3 El mensaje UNBIND_REQUEST.

La finalidad de este mensaje es terminar una sesión.

El formato del mensaje es:

```
UnbindRequest ::= [APPLICATION 2] NULL
```

Este mensaje no tiene respuesta asociada. Tras su transmisión el cliente y el servidor deben asumir que la conexión ha terminado, descartar las operaciones en curso y cerrar la conexión.

2.15.4 El mensaje SEARCH_REQUEST.

La operación de búsqueda permite al cliente consultar información del directorio que cumpla con un cierto requisito.

El formato de este tipo de mensaje es:

```
SearchRequest ::= [APPLICATION 3] SEQUENCE {
        baseObject      LDAPDN,
        scope           ENUMERATED {
                baseObject              (0),
                singleLevel             (1),
                wholeSubtree            (2) },
        derefAliases    ENUMERATED {
                neverDerefAliases       (0),
                derefInSearching        (1),
                derefFindingBaseObj     (2),
                derefAlways             (3) },
        sizeLimit       INTEGER (0 .. maxInt),
        timeLimit       INTEGER (0 .. maxInt),
        typesOnly       BOOLEAN,
        filter          Filter,
        attributes      AttributeDescriptionList }

    Filter ::= CHOICE {
        and             [0] SET OF Filter,
        or              [1] SET OF Filter,
        not             [2] Filter,
        equalityMatch   [3] AttributeValueAssertion,
```

```
        substrings      [4] SubstringFilter,
        greaterOrEqual  [5] AttributeValueAssertion,
        lessOrEqual     [6] AttributeValueAssertion,
        present         [7] AttributeDescription,
        approxMatch     [8] AttributeValueAssertion,
        extensibleMatch [9] MatchingRuleAssertion }

    SubstringFilter ::= SEQUENCE {
        type            AttributeDescription,
        -- at least one must be present
        substrings      SEQUENCE OF CHOICE {
            initial [0] LDAPString,
            any     [1] LDAPString,
            final   [2] LDAPString } }

    MatchingRuleAssertion ::= SEQUENCE {
        matchingRule    [1] MatchingRuleId OPTIONAL,
        type            [2] AttributeDescription OPTIONAL,
        matchValue      [3] AssertionValue,
        dnAttributes    [4] BOOLEAN DEFAULT FALSE }
```

El significado de los parámetros usados se expone a continuación:

- ***Basic object.*** La entrada que sirve de base de la búsqueda.
- ***Scope.*** El ámbito en el que se realiza la búsqueda.
- ***DerefAliases.*** Qué hacer con los alias que sean encontrados.
- ***Ziselimit.*** El número máximo de entradas que pueden ser devueltas. El valor cero deshabilita esta restricción.
- ***Timelimit.*** El tiempo máximo que puede durar una operación de búsqueda. El valor cero deshabilita esta restricción.
- ***TypesOnly.*** Indica que sólo deben devolverse los identificadores de atributo, sin sus valores.
- ***Filter.*** Es el filtro de búsqueda que contiene las condiciones que deben cumplir las entradas a devolver.

- ***Attributes.*** Es una lista de los atributos que deben ser devueltos de cada entrada que cumpla con las condiciones de la búsqueda. Existen dos valores especiales que son la cadena vacía y el carácter "*". Ambos indican que todos los atributos deben ser devueltos. La diferencia es que el carácter "*" puede usarse en combinación con nombres concretos, lo cual resulta útil para consultar atributos operacionales que deben ser específicamente solicitados.

2.15.5 El mensaje SEARCH_RESPONSE.

Los resultados resultantes de una búsqueda son devueltos en este tipo de mensajes que pueden contener los tipos de datos:

- ***SearchResultEntry.*** Entrada del directorio. Contiene entradas individuales de la búsqueda.

- ***SearchResultReference.*** Referencias encontradas durante la búsqueda. Si encuentra la referencia mientras se posiciona en la entrada base de la búsqueda deberá devolver el tipo de dato "SerachResulDone" conteniendo un campo con la referencia.

- ***ExtendedResponse.*** Respuesta a operaciones extendidas.

- ***SearchResultDone.*** Fin de las entradas encontradas, o indicación de que ha ocurrido un error.

El formato de estos mensajes es:

```
SearchResultEntry ::= [APPLICATION 4] SEQUENCE {
        objectName      LDAPDN,
    attributes      PartialAttributeList }

PartialAttributeList ::= SEQUENCE OF SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }
SearchResultReference ::= [APPLICATION 19] SEQUENCE OF
                                                LDAPURL

SearchResultDone ::= [APPLICATION 5] LDAPResult
```

2.15.6 El mensaje MODIFY_REQUEST.

Esta operación permite al cliente modificar la información contenida en una entrada del servidor.

El formato de este tipo de mensajes es:

```
ModifyRequest ::= [APPLICATION 6] SEQUENCE {
        object          LDAPDN,
        modification    SEQUENCE OF SEQUENCE {
                operation       ENUMERATED {
                                add     (0),
                                delete  (1),
                                replace (2) },
                modification    AttributeTypeAndValues } }

AttributeTypeAndValues ::= SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }
```

El significado de los distintos campos se expone a continuación:

- ***Objet.*** Contiene el nombre distinguido que identifica al objeto cuyo contenido va a ser modificado.

- ***Modification.*** Es una lista que contiene las operaciones a realizar sobre la entrada. La operación de modificación es atómica de manera que o se realizan todas las modificaciones o ninguna.

  Aunque alguna modificación pudiera ir en contra de las reglas del esquema, el resultado debe cumplir con las reglas de éste.

- ***Operation.*** Contiene un identificador de la operación a realizar.

  - ***Add.*** Permite añadir un nuevo atributo a la entrada o un nuevo valor particular a un atributo ya existente.
  - ***Delete.*** Permite eliminar un atributo concreto de una entrada o bien un valor particular del atributo.
  - ***Replace.*** Sustituye todos los valores de un atributo por un nuevo valor particular. Si este nuevo valor es una cadena en blanco elimina el atributo.

2.15.7 El mensaje MODIFY_RESPONSE.

El resultado de una operación “modify” consiste en información acerca de cómo se ha desarrollado la operación.

Esta información se devuelve en un mensaje de este tipo, cuyo formato se muestra a continuación:

```
ModifyResponse ::= [APPLICATION 7] LDAPResult
```

Este mensaje informa al cliente sobre el éxito de la operación (en cuyo caso se han llevado a cabo todas las modificaciones), sobre si ha habido algún tipo de problema y la causa que lo ha provocado (en cuyo caso ninguna modificación se habrá realizado).

Si la conexión falla durante la operación el cliente no tendrá conocimiento sobre qué modificaciones se han llevado a cabo.

Si se intentan modificar atributos de los que componen el nombre distinguido de la entrada se devolverá un error indicando que esa modificación no está permitida.

Si no se han definido reglas de correspondencia para un atributo, el cliente debe eliminar valores particulares usando la operación “replace” y no “delete”.

2.15.8 El mensaje ADD_REQUEST.

Este mensaje permite iniciar la operación de añadir una entrada al directorio inicializada con un conjunto de atributos y valores particulares.

El formato de este tipo de mensajes es:

```
AddRequest ::= [APPLICATION 8] SEQUENCE {
        entry           LDAPDN,
        attributes      AttributeList }

AttributeList ::= SEQUENCE OF SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }
```

El significado de los parámetros se expone a continuación:

- ***Entry.*** El nombre distinguido de la entrada que va a ser añadida. El servidor no hará ningún tipo de desreferencia de alias durante la localización del lugar en el que añadir la entrada.

- ***AttributeList.*** Contiene el conjunto de atributos y valores con los que es inicializada la entrada. Esta información debe seguir las reglas del esquema. El servidor se encargará de generar las marcas de tiempo y el resto de estructuras necesarias para la generación de la entrada.

No debe existir ninguna entrada con el nombre distinguido de la nueva entrada a añadir, pero sí debe existir obligatoriamente la entrada padre de la que colgará la nueva entrada del directorio.

Si la entrada padre existe pero no es una entrada del el directorio (ya que puede pertener a otro directorio al que se acceda mediante una remisión), se devuelve al cliente una referencia al directorio que contiene la entrada.

2.15.9 El mensaje ADD_RESPONSE.

Este mensaje se envía como respuesta a la operación de añadir una entrada al directorio indicando si la operación se ha realizado correctamente, en cuyo caso habrá una nueva entrada en el directorio, o si la operación no ha podido llevarse a cabo y la causa.

El formato de este tipo de mensajes se muestra a continuación:

```
AddResponse ::= [APPLICATION 9] LDAPResult
```

2.15.10 El mensaje DELETE_REQUEST.

Este mensaje permite solicitar al servidor que una entrada del directorio sea eliminada.

El formato de este mensaje es:

```
DelRequest ::= [APPLICATION 10] LDAPDN
```

El parámetro indicado es el nombre distinguido de la entrada que se pretende eliminar.

El servidor no desreferenciará alias durante la operación.

Sólo deben eliminarse entradas hojas del directorio, es decir entradas que no tengan hijos.

2.15.11 El mensaje DELETE_RESPONSE.

El servidor responde a una operación de borrado con este mensaje, indicando el éxito o el fracaso de la operación, y en este último caso la causa.

El formato de este mensaje es:

```
DelResponse ::= [APPLICATION 11] LDAPResult
```

2.15.12 El mensaje MODIFY_RDN_REQUEST.

Esta operación permite modificar el nombre de una entrada del directorio o mover un conjunto de entradas (un subárbol) a otro lugar del árbol (sólo en el caso de la versión 3 de LDAP).

El formato de este mensaje es:

```
ModifyDNRequest ::= [APPLICATION 12] SEQUENCE {
        entry           LDAPDN,
        newrdn          RelativeLDAPDN,
        deleteoldrdn    BOOLEAN,
        newSuperior     [0] LDAPDN OPTIONAL }
```

El significado de los parámetros se expone a continuación:

- ***Entry.*** El nombre distinguido de la entrada que se desea modificar. Esta entrada no debería tener hijos.

- ***Newrdn.*** El nombre distinguido relativo que compone el nombre distinguido de la nueva entrada que se encuentra más a la izquierda.

Esta es la pareja atributo/valor que forma el nombre distinguido relativo de la entrada.

- ***Deleteoldrdn.*** Es un valor lógico que si tiene el valor verdadero indica que el antiguo nombre no debe mantenerse como un atributo de la nueva entrada. Si tiene el valor falso el antiguo nombre sí será mantenido como un atributo.

- ***NewSuperior.*** El nombre distinguido de la entrada de la que cuelga ahora o sigue colgando la entrada cuyo nombre ha sido modificado. Si no se especifica este parámetro, la entrada seguirá colgando del mismo nodo que antes de la operación.

2.15.13 El mensaje MODIFY_RDN_RESPONSE.

Lo envía el servidor al cliente indicando que la operación se ha realizado correctamente o que ha habido un problema y la causa.
El formato de este mensaje es:

```
ModifyDNResponse ::= [APPLICATION 13] LDAPResult
```

Si el nuevo nombre de la entrada coincide con uno ya existente la operación no se llevará a cabo y se devolverá el correspondiente error.

Si el nuevo valor del atributo distinguido que identifica a la entrada no está permitido por el esquema también se devolverá el correspondiente error.

2.15.14 El mensaje COMPARE_REQUEST.

Permite comprobar si cierta entrada contiene una pareja atributo/valor determinada.

El formato de este mensaje es:

```
CompareRequest ::= [APPLICATION 14] SEQUENCE {
        entry       LDAPDN,
        ava         AttributeValueAssertion }
```

El significado de los parámetros se expone a continuación:

- ***Entry.*** El nombre de la entrada sobre la que se desea realizar la operación.

- ***Ava.*** La representación de la pareja atributo/valor deseada.

Esta operación es especialmente útil en la consulta de atributos que no pueden ser leídos pero sí comparados, tales como claves de acceso, etc.

2.15.15 El mensaje COMPARE_RESPONSE.

Lo envía el servidor al cliente para indicarle el resultado de una operación de comparación, informándole tanto del éxito como del fracaso y sus causas.

El formato de este mensaje es:

CompareResponse ::= [APPLICATION 15] LDAPResult

2.15.16 El mensaje ABANDON_REQUEST.

Solicita al servidor que aborte una operación en curso.

El formato de este mensaje es:

AbandonRequest ::= [APPLICATION 16] MessageID

El campo “MessageID” identifica la operación que se desea finalizar.

No existe un mensaje de respuesta para esta solicitud, con lo cual el cliente interpretará que el servidor ha finalizado la operación y el servidor inmediatamente abortará la operación en curso y devolverá los resultados coherentes que hasta ese momento pudiera haber generado la operación, como por ejemplo entradas resultantes en una operación de búsqueda. En este caso devolverá todas las entradas pero no un mensaje que indique que ya no hay más entradas.

El cliente no debe enviar varias veces este mensaje aplicado a la misma operación, y debe permanecer a la escucha de posibles resultados que hasta ese momento pudieran haber sido generados.

2.15.17 El mensaje EXTENDED_REQUEST.

El cliente envía al servidor este mensaje cuando requiere que una operación extendida sea ejecutada.

El formato de este mensaje es:

```
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
        requestName      [0] LDAPOID,
        requestValue     [1] OCTET STRING OPTIONAL }
```

- El campo ***"requestName"*** contiene un identificador de objeto que designa a la operación extendiada

- El campo ***"requestValue"*** información en un determinado formato y representada en una cadena de octetos necesaria para llevar a cabo la operación.

2.15.18 El mensaje EXTENDED_RESPONSE.

Es la respuesta a una operación extendida.

El formato de este mensaje es:

```
ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
        COMPONENTS OF LDAPResult,
        responseName     [10] LDAPOID OPTIONAL,
        response         [11] OCTET STRING OPTIONAL }
```

Contiene un identificador de objeto que designa a qué petición obedece la respuesta e información referente al resultado de dicha petición.

2.16 Codificación y transferencia de mensajes.

Los elementos del protocolo son codificados para su transmisión mediante las reglas de codificación básica (BER) de ASN.1.

Sin embargo debido a particularidades de ciertos elementos algunas de estas reglas podrían sufrir ligeras modificaciones.

Las PDUs correctamente codificadas son directamente mapeadas sobre los campos de información de las PDUs del protocolo TCP que sirve como soporte de transporte para la transmisión de mensajes LDAP.

2.17 Notificaciones no solicitadas.

Una notificación no solicitada es un mensaje enviado por el servidor al cliente de manera asíncrona y sin responder a ninguna solicitud del cliente para indicarle algún cambio referente a la conexión.

Este tipo de mensaje tiene la naturaleza de aviso y el servidor no esperará ninguna respuesta a él por parte del cliente.

Una notificación no solicitada se codifica como un tipo ***"LDAPMessage"*** en el que el campo "messageID" tiene el valor 0 y el campo "protocolOP" es de la forma "extendedResp" con el campo "responseName" presente.

La recomendación RFC 2251 define un tipo de notificación no solicitada llamada ***notificación de desconexión.***

El mensaje de notificación de desconexión es usado por el servidor para indicar que va a cerrar la conexión a causa de un error.

En este caso el cliente desconoce el resultado que han podido tener las operaciones que se encontraban en curso en ese momento.

La causa de este evento se indica en el campo "resultCode", y pude ser por errores de protocolo, que un mensaje no haya podido ser decodificado, errores asociados a la conexión, o que el servidor pase a un estado de no disponibilidad.

Una vez enviado este mensaje el servidor y el cliente cerrarán la conexión y si el cliente desea volver a consultar la información del servidor deberá establecer una nueva conexión.

2.18 El formato LDIF.

El formato ***LDIF*** (LDAP Data Interchange Format) se usa para representar entradas LDAP en formato de texto. El formato LDIF se define en la RFC 2849.

Mediante un simple fichero de texto que cumpla con las reglas del fomato, el diseñador puede construir un directorio, especificando las entradas que desee, los atributos que dichas entradas contienen y la relación que deben guardar entre ellas para formar el DIT.

La información así especificada es perfectamente legible por los servidores particulares de directorio, que tras leer el fichero LDIF correspondiente, incluirán en su DIT todas las entradas que en dicho fichero hayan sido especificadas, o construirá un nuevo DIT partiendo de la informació contenida en el fichero (lo más normal).

El formato LDIF permite por tanto la creación de nuevas entradas en el directorio de forma cómoda y directa, siendo especialmente útil en situaciones en las que el número de entradas que se desea añadir sea elevado, ya que en este caso el proceso de incorporar entradas al directorio debería hacerse mediante la operación de añadir definida en LDAP, lo cual resultaría costoso, tedioso, y prácticamente inviable en muchos casos prácticos.

El conjunto de entradas deseado se especifica en LDIF mediante bloques de texto separados por líneas en blanco. Cada bloque de texto representa una entrada particular del directorio. Dicha entrada debe comenzar con la etiqueta “dn:” y a continuación el nombre distinguido de la entrada, con el fin de que el servidor pueda asociar este nombre al objeto de directorio que implementará esta entrada concreta.

Dentro del bloque de texto que representa una entrada debe ir además de su nombre distinguido, la declaración del conjunto de atributos que la entrada contiene. Los atributos se declaran en líneas distintas (sin usar líneas en blanco, ya que en este caso se consideraría que comienza una nueva entrada). Dichas líneas deben comenzar por el identificador del atributo que

se desea declarar, seguido del carácter ":", y a continuación el valor particular que se desea dar a este atributo concreto.

En caso de pretender que un atributo tenga múltiples valores, deberá repetirse la línea de declaración del atributo tantas veces como se desee, especificando en cada una un valor particular del atributo.

El diseñador no debe olvidar incluir entre los atributos de una entrada el atributo "objectclass", en el cual se especifica la clase objeto a la que pertenece la entrada que está siendo declarada. Dicho atributo debe ser inicializado al menos con un valor, y el resto de atributos deber respetar las reglas de esquema que conlleva la pertenencia a esa clase objeto, ya que en caso contrario el servidor rechazará la entrada.

Entre estos atributos debe encontrarse también un atributo especial que contenga el nombre distinguido relativo de la entrada con respecto a su predecesora (la entrada de la que cuelga), a fin de que este atributo pueda ser usado para construir tanto su nombre distinguido como los nombres distinguidos de las entradas que se encuentran el el subárbol que cuelga de ella.

Siguiendo todas las especificaciones del formato LDIF señaladas anteriormente, se llega a que el formato de un fichero LDIF tiene la siguiente estructura.

```
# comment
dn: <distinguished name>
<attrdesc>: <attrvalue>
<attrdesc>: <attrvalue>
<attrdesc>: <attrvalue>
<attrdesc>: <attrvalue>
...

# comment
dn: <distinguished name>
<attrdesc>: <attrvalue>
<attrdesc>: <attrvalue>
<attrdesc>: <attrvalue>
<attrdesc>: <attrvalue>
...

# comment
dn: <distinguished name>
<attrdesc>: <attrvalue>
```

<attrdesc>: <attrvalue>
<attrdesc>: <attrvalue>
<attrdesc>: <attrvalue>
...

etc

Las líneas de comentario comienzan por el carácter “#”.

En la estructura anterior pueden observarse una serie de bloques separados por líneas en blanco, que son las distintas entradas que se declaran.

El nombre distinguido de la entrada se representa mediante la etiqueta “dn”, y el resto de líneas designan un identificador de atributo (<attrdesc>) y un valor particular de ese atributo (<attrvalue>).

El serividor de directorio comienza a leer desde la primera entrada, comprobando su contenido y verificando que respeta el esquema, en cuyo caso la añade al directorio. Este proceso se repite con todas las entradas hasta que el fichero termine o hasta que se encuentre algún tipo de error, en cuyo caso no se añade la entrada que lo contenga ni ninguna otra declarada con posterioridad a ésta. Las entradas que ya habían sido añadidas al directorio no sufren ningún tipo de alteración ni se ven afectadas por errores en entradas posteriores.

2.19 Implementación del protocolo. Servidores comerciales.

Durante los últimos cinco años, LDAP (Lightweight Directory Access Protocol) se ha convertido en el estándar relativo al acceso de usuarios y aplicaciones a la información almacenada en los servidores de directorios más empleado. Existe una gran variedad de servidores LDAP, pero todos operan de diferentes maneras y con diversos resultados. De las pruebas realizadas por el organismo ***“Network World Test Alliance de IDG Communications”***, el servidor del fabricante “iPlanet (Netscape)” resultó ser el mejor por sus superiores prestaciones en cuanto a gestión y rendimiento.

Este organismo ha realizado pruebas sobre un amplio espectro de servidores de directorios LDAP, desde los muy simples y rápidos utilizados dentro de grandes sitios de Internet hasta las bases de datos relacionales con LDAP integrado, pasando por los servidores de directorios corporativos con soporte del protocolo y los servidores X.500 con soporte basado en LDAP.

En este estudio se invitó a participar a 30 fabricantes, y se realizaron pruebas en base a cuestiones como la simplicidad de la gestión, el cumplimiento de estándares y el rendimiento. Por encima de todo, se examinó cómo estos servidores de directorio tratan los clientes LDAP - usuarios y aplicaciones- que acceden a los datos.

También se analizó el conjunto de herramientas accesorias que cada producto ofrecía al usuario. Este conjunto incluye típicamente herramientas de formateado de datos y herramientas de consulta, que permiten ver la información contenida en el directorio. También puede incorporar otras piezas, como los editores de esquemas, que aunque ayudan a facilitar la tarea del administrador, son parte propiamente dicha del servidor LDAP.

En el estudio se evaluaron servidores como:

- "Directory Server" de iPlanet (Netscape).
- IDDS de Innosoft.
- NDS eDirectory de Novell.
- Global Directory Server de Critical Path.
- ETrust Directory de Computer Associates.
- DirX de Siemens.
- Oracle Internet Directory de Oracle.
- Active Directory de Microsoft.
- OpenLDAP (de libre distribución).

Las pruebas se realizaron sobre "Windows NT", y los dos últimos servidores nombrados no entraron en dichas pruebas.

El que mejor resultado ofrece en cuanto a velocidad es el servidor de iPlanet, seguido del servidor de Innosoft.

En cuanto a rendimiento el servidor que ofreció mejores resultados fue "Directory Server", aunque con poca diferencia sobre el resto.

Con los dos últimos directorios que no entraron en el proceso de pruebas se realizó una comparación entre ambos que arrojó el siguiente resultado: Se

encontró que Active Directory, aunque tiene algunos inconvenientes en cuanto a rendimiento de cargas y ofrece un conjunto solapado de herramientas de gestión, trata las peticiones LDAP bastante bien y ofrece un excelente conjunto de herramientas de monitorización. Con la alternativa Linux, el rendimiento obtenido es desalentador y carece de herramientas de administración, lo que dificulta su uso. OpenLDAP está mejor posicionado como un kit de inicio para ayudar a obtener experiencia con LDAP que como un servidor de producción.

Sobre OpenLDAP se concluyó que es otro servidor LDAP puro, no tiene interfaz gráfica de usuario, y obligó a editar archivos de configuración o editar comandos shell para efectuar cualquier cambio. Además obtuvo muy pobres valores de rendimiento en la mayoría de las baterías de pruebas, situándose casi siempre por debajo del resto de servidores evaluados.

Sin embargo, ofrece una gran simplicidad y una fácil instalación, lo que le convierte en una buena herramienta de prototipo para un directorio LDAP, especialmente con IDDS y Directory Server, ya que la configuración y operación de los tres productos es muy similar.

Capítulo 3: El servidor OpenLDAP.

OpenLDAP es el servidor de directorio basado en el protocolo LDAP que ha sido elegido para funcionar como servidor de directorio del sistema.

Este apartado expone las características y el uso de este software.

3.1 Servicios de directorio.

Un directorio es un tipo especial de base de datos optimizada para realizar operaciones de lectura y de búsqueda. El mantenimiento de la información en servicios de directorio se hace mediante entradas que son las entidades de información del directorio. Dichas entradas contienen atributos que guardan información particular referente a esa entrada.

Los servicios de directorio suelen además soportar filtros de búsqueda, que son expresiones que permiten al usuario realizar búsquedas complejas en el directorio basándose en el contenido de los atributos.

Normalmente los directorios no soportan operaciones complejas típicas de bases de datos, tales como transacciones.

Normalmente los servicios de directorio soportan operaciones de actualización de la información, si bien no se encuentran optimizados para tal fin, ya que su principal utilidad es dar respuestas rápidas a consultas de búsqueda sobre un elevado volumen de información.

Existen varios modelos de organización de directorios para proporcionar servicio en un sistema, ya que es posible el almacenamiento de diferentes tipos de información y el uso de distintas estrategias para gestionar dicha información. Pueden diseñarse sistemas en los que la información se encuentre distribuida en varios directorios residentes en distintas máquinas, o bien que la información de un directorio maestro se encuentre replicada en varios esclavos encargándose el maestro de actualizar periódicamente la información contenida en los directorios esclavos, etc.

En el caso de tener varios directorios conteniendo la misma información los directorios esclavos son usados para replicar la información del directorio maestro con el fin de aumentar la disponibilidad y la fiabilidad de la información en un sistema más o menos amplio, reduciéndose así el tiempo de respuesta. Para ello cada cierto tiempo el directorio maestro actualiza la información de los directorios esclavos que dependen de él y aspiran a ofrecer al usuario la misma información que se encuentra en el directorio maestro.

Este modo de funcionamiento presenta la desventaja de que tras una eventual actualización del directorio maestro la información del sistema es inconsistente durante el intervalo de tiempo en el cual el servidor maestro contacta con los esclavos y le transmite los cambios.

Otra posibilidad es la de ofrecer un servicio de directorio distribuido. En este caso existen en la red una serie de equipos que albergan un servidor de directorio. Todos estos servidores cooperan entre si con el fin de ofrecer un servicio global de directorio que permita al cliente ver el directorio como un todo.

Una tercera posibilidad a la hora de diseñar un sistema de directorio en una red, es la de tener un directorio local gestionado por un servidor local que contenga toda la información de directorio disponible.

La elección del sistema a usar dependerá en todo caso del tipo de servicio que se quiera ofrecer y de las necesidades reales del sistema en el que se desea implantar el servicio de directorio.

3.2 El proyecto OpenLDAP

OpenLDAP es un constate proyecto de colaboración que busca desarrollar un software libre que funcione como un servidor que implementa el protocolo LDAP así como un conjunto de herramientas destinadas a tanto a la gestión como al mantenimiento del servidor.

Las líneas de avance de OpenLDAP van destinadas a soportar cada vez más características de LDAP ya que esta tecnología se encuentra en constante evolución, así como conseguir un software más robusto y que se asemeje más a un software comercial.

Los colaboradores que trabajan en el proyecto OpenLDAP son voluntarios de todo el mundo que a través de Internet colaboran y desarrollan OpenLADP y su documentación asociada.

El proyecto está coordinado por la fundación OpenLDAP, una corporación no lucrativa cuyo objetivo es el desarrollo del software OpenLDAP y su documentación.

3.3 Formatos de OpenLDAP.

El software OpenLDAP se encuentra disponible gratuitamente en la red bajo la licencia de software libre.

A la hora de intentar descargar el software de la red se tienen una serie de versiones que son fruto de la evolución constante que sufre el software OpenLDAP. El software disponible en la red se encuentra clasificado en:

- ***Release.*** Es la última versión de OpenLDAP publicada para uso general.

- ***Stable release.*** Esta versión la última publicada que ha sido probada en diferentes sistemas y bajo diversas circunstancias y se asegura su correcto funcionamiento.

- ***Test release.*** Algunas versiones tienen disponible un software asociado que se usa para testar la aplicación.

3.4 Implementación del protocolo LDAP.

El fin de la distribución OpenLDAP es la implementación del protocolo LDAP (Lightweight Directory Access Protocol) que es un protocolo de acceso a directorio que corre sobre TCP/IP y se encuentra definido en la RFC 2251.

El modelo de almacenamiento de información en el directorio está basado en entradas. Una ***entrada*** es un conjunto de atributos englobados bajo un nombre único que los idéntica y que se conoce como ***nombre distinguido (DN)***. Cada atributo contenido en una entrada tiene un tipo y uno o más valores particulares. Normalmente los nombres de los atributos son representativos del tipo de información que contiene (por ejemplo el atributo mail suele contener direcciones de correo electrónico).

Estas entradas están organizadas en el directorio formando una estructura jerárquica en forma de árbol que refleja las dependencias entre entradas siguiendo un orden lógico e intuitivo para el usuario dentro del directorio.

OpenLDAP también ofrece soporte de esquemas, es decir que permite al administrador controlar qué atributos son necesarios o posibles dentro de una determinada entrada, ya que el esquema guarda información referente a la construcción y mantenimiento de las entradas del directorio.

También se incorpora el método de localización de entradas de LDAP, que está basado en el uso del nombre distinguido, el cual está construido concatenando el nombre propio de la entrada (***nombre distinguido relativo RDN***) y los nombres de sus entradas antecesoras.

Para acceder a la información se implementan las funciones definidas en LDAP acerca de la consulta y la actualización de directorios tales como añadir entradas, borrar entradas, cambiar el nombre de una entrada o realizar búsquedas o comparaciones, siendo esta última operación la principal dentro del directorio y para la cual se encuentra optimizado el protocolo.

OpenLDAP también atiende a cuestiones de seguridad, ya que el propio protocolo LDAP ofrece al cliente una serie de mecanismos para su autentificación con el fin de proteger la información de usuarios no autorizados.

OpenLDAP está construido en base al modelo cliente-servidor propuesto en LDAP, en el cual uno o más servidores contienen la información y entre

todos implementan el servicio de directorio. Un cliente LDAP se conecta al servidor OpenLDAP y realiza una consulta que es respondida por el servidor con la información requerida y en algunos casos con un puntero a un lugar en el cual el cliente puede encontrar información adicional, es decir una referencia o una remisión según sea el caso.

El software OpenLDAP implementa el servidor LDAP mediante una aplicación llamada ***Slapd*** que simula un directorio maestro, y otra aplicación llamada ***Sldrup*** que simula un servidor esclavo que replica la información del maestro.

Además incorpora una serie de clientes LDAP capaces de hacer consultas al servidor de directorio así como un conjunto de herramientas que permiten gestionar dicho directorio.

3.5 La aplicación SLAPD.

Slapd es un servidor de directorio LDAP capaz de funcionar en diferentes plataformas.

Normalmente esta aplicación funciona en modo demonio permaneciendo en un segundo plano a la escucha de peticiones realizadas por los clientes.

Slapd soporta las siguientes características:

- Soporte de las versiones dos y tres del protocolo LDAP.
- Es capaz de funcionar sobre IPv4 y sobre IPv6.
- Es capaz de soportar mecanismos de autentificación complejos tales como mecanismos SASL, si bien para ello es necesaria la instalación de software adicional. Entre los mecanismos SASL se encuentran protocolos como DIGEST-MD5, EXTERNAL o GSSAPI.
- Proporciona soporte para una capa de transporte seguro que permita proteger la información mediante el uso del protocolo SSL (o TLS en cuyo caso es necesario la distribución adicional ***OpenSSL***).
- Proporciona facilidades para el control de acceso permitiendo al administrador decidir qué usuarios tienen permiso para acceder a distintos puntos del árbol y cuáles son esos permisos.

- Soporta la internacionalización de caracteres, basándose en el uso de lenguaje Unicode.

- El soporte para el almacenamiento de información viene dado por una base de datos en la cual se guardan las entradas y que queda en un segundo plano. Slapd incorpora diferentes bases de datos que pueden escogerse. Estas bases de datos son:
  - ***LDBM.*** Es una base de datos de alta velocidad basada en disco.
  - ***SHELL.*** Es un interface de una base de datos que interpreta comandos.
  - ***PASSWD.*** Es una base de datos implementada en un fichero cuyo fin es el de guardar claves sencillas.
  - ***SQL.*** Base de datos en formato SQL.

- Puede ser configurado para dar servicio usando varios tipos de bases de datos al mismo tiempo.

- Slapd está organizado en dos partes. Por un lado se tiene un módulo que se encarga de mantener y gestionar las comunicaciones con clientes LDAP y por otra parte otro módulo realiza operaciones específicas de base de datos.

  Estos dos módulos cooperan entre si mediante un conjunto de operaciones recogidas en un API escrito en C. Gracias a esto un usuario puede desarrollar sus propias aplicaciones usando este API. También la distribución incorpora una serie de módulos que implementan bases de datos programables con lo cual se pueden incorporar al servidor datos externos usando lenguajes de programación de bases de datos, tales como Perl, SQL o TCL.

- Soporta hilos para aumentar la velocidad de respuesta.

- Puede ser configurado para actuar como maestro en un sistema en el que haya servidores esclavos que repliquen la información contenida en él.

- Es fácilmente configurable, ya que este proceso se realiza a través de un fichero de configuración que permite cambiar una elevada variedad de aspectos.

Slapd también tiene numerosas limitaciones entre las cuales se encuentra que el principal soporte de base de datos que es LDBM no es la tecnología ideal para manejar las operaciones de búsqueda que soporta LDAP.

Esta limitación junto con otras muchas se encuentran en fase de estudio e irán siendo solventadas o al menos mejoradas en futuras versiones del software.

Para arrancar la aplicación hay que invocar el comando ***slapd.***
Como generalidades de la aplicación en sí hay que señalar que funciona normalmente como una aplicación del tipo demonio, desvinculándose y permaneciendo a la escucha de conexiones LDAP por el puerto por que se encuentre configurado y que por defecto es el 389.

Al arrancar crea un fichero (indicado por en el fichero de configuración) con extensión ***pid*** en el cual guarda el identificador de proceso que el sistema le ha asignado, y en otro fichero de extensión ***args*** (también especificado en el fichero de configuración) guarda los argumentos con los que ha sido invocada la aplicación en la línea de comandos.

Además de cómo una aplicación de tipo demonio puede hacerse que funcione como una aplicación normal que se ejecuta en un primer plano mediante el uso de la opción –d en la línea de comandos.

El formato general de este comando se detalla a continuación si bien existen más parámetros que modifican el comportamiento de la aplicación.

slapd [-f slapd-config-file] [-h URLs][-d debug-level] [-n service-name] [-s syslog-level] [-l syslog-local-user] [-r directory] [-u user] [-g group]

- ***-d debug level.*** Este parámetro pasado sin argumentos hace que Slapd funcione como una aplicación en un primer plano. Si junto con esta opción se pasa un entero que se interpreta como el nivel de monitorización de la aplicación que se desea en cuyo caso se irán imprimiendo por la salida estándar una serie de mensajes informando sobre las operaciones que realiza la aplicación con el grado de detalle solicitado mediante el número proporcionado.

- ***-s syslog level.*** Junto con un entero indica a qué nivel deben ser registradas las expresiones provenientes de la monitorización de la aplicación.

- ***-n service-name.*** Indica el nombre del servicio para usarlo en registros y otras operaciones. Por defecto es “slapd”:

- ***-l syslog-local-user.*** Indica el usuario local de las aplicaciones de monitorización.

- ***-h URLlist.*** Sirve para especificar una lista de URLs separadas por el carácter espacio en blanco en las cuales la aplicación Slapd ofrece servicio a sus clientes.

  Entre las URLs posibles se encuentran:

  - ***ldap:///*** Indica que el servicio se ofrece en todos los interfaces de la máquina local en el puerto por defecto (389).
  - ***ldap://dirección ip:número de puerto/*** Indica que el servicio se ofrece en el interface especificado por la dirección IP y en el puerto dado por el número de puerto.
  - ***ldaps://dirección IP:número de puerto/*** La diferencia con la anterior es que ahora la comunicación se realiza de forma segura mediante el protocolo SSL (o TLS), es decir que la comunicación se realiza sobre un puerto seguro que por defecto es el 636.
  - ***ldapi://dirección IP:número de puerto/*** En esta ocasión la particularidad es que la comunicación se realiza sobre IPC (Unix Domain Sockets).

- ***-f file.*** Especifica la localización del fichero de configuración.

- ***-r directory.*** Indica un directorio en el cual se guarda información durante el proceso de inicialización de Slapd antes de haber leído el fichero de configuración.

- ***-u user.*** Inicializa Slapd con el nombre de usuario o el identificador dado por esta propiedad.

- ***-g group.*** Inicializa Slapd con el nombre de grupo o el identificador de grupo indicado por esta propiedad.

3.5.1 El fichero de configuración de SLAPD.

Cuando la aplicación Slapd arranca debe conocer la ubicación del fichero de configuración. Este es un fichero en modo texto que sigue unas reglas determinadas y que inicializa parámetros que determinan el funcionamiento del servidor LDAP.

El fichero de configuración es designado normalmente con el nombre ***slapd.conf*** y contiene tres tipos de información de configuración según a qué aspecto del funcionamiento hagan referencia.

- ***Global.*** Este tipo de información se detalla en primer lugar y hace referencia a aspectos generales del sistema. Las propiedades aquí configuradas pueden ser reconfiguradas posteriormente en otra parte del fichero de configuración, ya que si se vuelve a definir la propiedad prevalece el último valor configurado.

- ***Información referente al soporte de directorio.*** Se coloca en segundo lugar en el fichero de configuración y hace referencia al directorio particular que se esté usando.

- ***Información específica de la base de datos.*** Suele colocarse al final del fichero de configuración y hace referencia al tipo de base de datos que mantiene la información del directorio.

En el fichero de configuración las líneas en blanco son ignoradas, así como también lo son las líneas de comentario que comienzan con el carácter “#”. Si una línea comienza con un espacio en blanco es considerada como continuación de la línea anterior.

Por tanto el formato general de un fichero de configuración es el siguiente:

```
# global configuration directives
<global config directives>

# backend definition
backend <typeA>
<backend-specific directives>

# first database definition & config directives
database <typeA>
<database-specific directives>
```

```
# second database definition & config directives
database <typeB>
<database-specific directives>

# second database definition & config directives
database <typeA>
<database-specific directives>

# subsequent backend & database definitions & config
    directives
...
```

Las directivas son los parámetros que son configurados en el fichero de configuración, y por tanto deben ir acompañadas de argumentos que las inicialicen. Si los parámetros contienen espacios en blanco deben ir entre comillas, así como si contienen caracteres especiales tales como comillas o el carácter "\", éstos deben ir precedidos del carácter de escapado "\".

La distribución OpenLdap incorpora un fichero de configuración de muestra que es el que se usa por defecto y que configura los parámetros más básicos para el funcionamiento del servidor.

Los siguientes apartados detallan las directivas que pueden usarse, clasificándolas según el criterio señalado anteriormente, es decir, según a qué tipo de información hacen referencia.

3.5.1.1 Parámetros globales.

Estos parámetros se aplican a todos los directorios que maneje el servidor y a todas las bases de datos.

Los parámetros que aparecen entre los caracteres "< >" son valores particulares que debe configurar el administrador.

Estos parámetros son los siguientes:

- ***Access to <what> [by <who> <accesslevel> <control>]+***
  Mediante este parámetro se configura el acceso al directorio. Mediante <what> se indica a que parte del directorio se concede el acceso, <who> designa los usuarios a los que se concede dicho acceso; <accesslevel> el tipo de acceso que puede ser de menor a mayor jerarquía:

- Ninguno
- Autentificación
- Comparación
- Búsqueda
- Lectura
- Escritura

Cada uno de los permisos implica tener todos los permisos de menor jerarquía.

Con el campo <control> se designan aspectos especiales, como puede ser el acceso a un atributo concreto.

La sintaxis de estos parámetros se detalla en el apartado “Control de acceso al sistema”.

- ***Attributetype <RFC 2252 Attribute type description>.*** Este parámetro define un tipo de atributo que será incorporado al esquema y que se sigue la recomendación RFC 2252.

- ***Defaultaccess { none, compare, search, read, write }.*** Especifica los permisos por defecto que tiene un usuario cuando accede al sistema de forma anónima. En este caso los permisos se presentan ordenados por orden de jerarquía. Tener un cierto permiso implica tener el resto de permisos de inferior jerarquía.

- ***Idletimeout <integer>.*** Indica el tiempo en segundos que una conexión que no se esté usando puede permanecer activa. Pasado este tiempo se cierra la conexión por parte del servidor. Si se configura este parámetro con un valor “0” se deshabilita este parámetro de modo que una conexión no será cerrada por el hecho de permanecer inactiva un cierto tiempo.

- ***Include <filename>.*** Indica que el servidor Slapd debería completar la información dada en el fichero de configuración con información adicional que se encuentra en el fichero indicado por la directiva include. El fichero indicado debe respetar el formato de configuración del servidor Slapd.

  Esta directiva es especialmente útil a la hora de incorporar información de esquema al directorio que se encuentre en ficheros con la extensión ***schema.***

- ***Loglevel <integer>.*** Esta directiva indica el nivel al que se registra el comportamiento del sistema. El entero es un número que puede ser 0, -1, o una potencia de dos entre 0 y 2048 indicando así el nivel de monitorización que se desea.

- ***Objectclass <RFC 2252 Object Class Description>.*** Permite definir una clase objeto para ser incorporada al esquema según la RFC 2252.

- ***Referral <URI>.*** Especifica una remisión que debe ser devuelta cuando Slapd no puede encontrar una base de datos para atender una solicitud.

- ***Sizelimit <integer>.*** Especifica el máximo número de entradas que el servidor puede devolver como respuesta a una solicitud de búsqueda. Por defecto 500.

- ***Timelimit <integer>.*** Especifica el tiempo máximo en segundos que Slapd empleará en responder a una solicitud de búsqueda. Si durante este tiempo no ha podido completarse la operación se devolverá un resultado indicando que se ha excedido el límite de tiempo. Por defecto 3600.

3.5.1.2 Parámetros referentes al soporte de directorio.

Estos parámetros se aplican sólo al directorio para el cual están definidos.

Los parámetros que aparecen entre los caracteres "< >" son valores particulares que debe configurar el administrador.

Estos parámetros son los siguientes:

- ***Backend <type>.*** Indica el comienzo de la definición de un directorio. El campo <type> debe ser el nombre del tipo de soporte que se le da al directorio, es decir, el tipo de base de datos que va a usarse para contener la información del directorio. Los posibles valores para este parámetro son:
  - Ldbm
  - Shell
  - Passwd

Los parámetros que se mencionan a continuación son específicos del soporte LDBM.

- ***Cachesize <integer>.*** Indica el tamaño en entradas de la memoria caché que ocupa una instancia de una base de datos del tipo LDBM.

- ***Dbcachesize <integer>.*** Especifica el tamaño en bytes de la memoria caché asociada con cada fichero de ***índices***, siendo éstos un tipo de información que mantiene la base de datos para propósitos administrativos. Si el método subyacente no soporta esta directiva la ignora sin notificarlo.

  Aumentar este parámetro permite usar más memoria pero también puede provocar problemas, bajando el rendimiento a la hora de modificar entradas o construir índices para esas entradas.

- ***Dbnolocking.*** Esta opción si está presente deshabilita la seguridad de la base de datos, permitiendo una mayor velocidad en las operaciones pero a costa de perder la seguridad sobre la información.

- ***Dbnosync.*** Permite que el contenido de una base de datos no sea inmediatamente sincronizado con los cambios en memoria. Habilitar esta opción implica ganar en velocidad a costa de la seguridad y la consistencia de la información.

- ***Directory <directory>.*** Indica un directorio en el que están contenidos los ficheros LDBM que contienen la información sobre la base de datos y sus actuales índices asociados.

- ***Index {<attrlist>, default} [pres, eq, approx, sub, none].*** Indica los índices que deben mantenerse para un atributo concreto. Si sólo se rellena el campo <attrlist> se mantienen los índices definidos por defecto. Entre los posibles índices que pueden mantenerse se encuentran índices de igualdad, de presencia, de aproximación, de subcadenas etc.

- ***Mode <integer>.*** Indica el índice de protección que una base de datos recién creada debe tener.

3.5.1.3 Parámetros específicos de la base de datos.

Este tipo de parámetros aplicables a una base de datos concreta que contiene la información sobre un directorio. Son soportados por cualquier tipo de bases de datos.

Los parámetros que aparecen entre los caracteres "< >" son valores particulares que debe configurar el administrador.

Estos parámetros son los siguientes:

- ***Database <type>.*** Este parámetro marca el comienzo de una nueva base de datos. El campo <type> debe contener el nombre de un tipo de base de datos soportado como podría ser "ldbm", "shell" o "passwd".

- ***Readonly { on, off }.*** Este parámtro permite poner si se selecciona el valor "on" la base de datos en modo sólo lectura.

- ***Replica.*** Especifica un servidor esclavo que replica el directorio contenido en el maestro.

- ***Replogfile <filename>.*** Indica el nombre de un fichero que usa Slapd para registrar los cambios que se producen. Este fichero lo lee Slurpd en el caso en el que esta aplicación se encuentre activa y haya servidores que repliquen la información. De otra manera la utilidad de este fichero es la monitorización del sistema.

- ***Rootdn <dn>.*** Indica el nombre distinguido (DN) de una entrada del directorio a la que no tiene restricciones de acceso o de operación sobre el directorio. Esta entrada representa al administrador del directorio.

- ***Suffix <dn suffix>.*** Este parámetro indica el sufijo que tendrán todas las entradas de un directorio concreto, por lo que debe ser distinta para cada directorio definido.

  Este parámetro representa la raíz del directorio. De esta manera Slapd reconoce el directorio al que pertenece el nombre de una entrada que pueda darse como parámetro en alguna operación.

- ***Updatedn <dn>.*** Se aplica cuando un servidor Slapd está funcionando como esclavo. Especifica el nombre distinguido de una entrada a la que se le permite realizar cambios en la información replicada. Este nombre debe ser el nombre que identifica a la aplicación Slurpd cuando se conecta al servidor esclavo para realizar los cambios pertinentes. También podría ser el nombre distinguido asociado con una entidad SASL.

- ***Updateref <URL>.*** Es sólo aplicable en el esclavo Slapd. Especifica la URL que debe devolver a los clientes que intentan ejecutar operaciones de actualización sobre la información replicada.

3.6 Control de acceso al sistema.

El control de acceso al sistema hace referencia a qué usuarios convenientemente identificados se le permite el acceso a ciertas zonas del árbol de directorio y con qué grado de privilegio.

Los controles de accesos se especifica mediante directivas en el fichero de configuración de la aplicación.

El formato general de este tipo de directiva es:
Access to <what> [by <who> <accesslevel> <control>]+

El campo <what> determina a qué entradas y atributos se les aplican los controles de acceso. Los nombres de las entradas pueden ser dados de dos maneras:

- Mediante el uso de una expresión que designe el nombre distinguido de una entrada tal como se recoge en la RFC 2253, como por ejemplo podía ser ***dn="cn=Babs Jensen, dc=example, dc=com".***

- Otra posibilidad usar un filtro tal como se define en la RFC 2254 como por ejemplo **filter="cn=*, dc=example, dc=com".**

Los atributos particulares de una entrada se seleccionan incluyendo un comando separado del campo <what>. Este comando es ***attrs.*** De esta manera se incluiría una expresión del tipo ***attrs=<attribute list>*** especificando una lista de atributos particulares de las entradas a los que se quiere controlar el acceso.

Existe un nombre de atributo especial que es ***"entry"*** y que identifica el acceso a la propia entrada.

Para designar todas las entradas de directorio podría usarse el selector **"*".** Por ejemplo, la expresión "dn=.*" designaría a todas las entradas del directorio.

El campo <who> identifica a la entidad a la cual se le concede el acceso. Estas entidades pueden ser o no entradas del directorio. Entre los valores posibles para este campo se encuentran:

- **"*".** Representa a todos los usuarios, tanto a los anónimos como a los autentificados.

- ***Anonymous.*** Representa a los usuarios anónimos, es decir a aquellos que acceden al sistema sin autentificarse.

- ***Users.*** Identifica todos los usuarios identificados.

- ***Self.*** Representa al usuario asociado con la entrada indicada en el campo <what>.

- ***Expresiones del tipo dn=< expresión>.*** Es el caso más general. El campo <expresión> es un nombre distinguido de la entrada a la que se desea identificar.

- ***Usuarios identificados por el nombre de su dominio.*** Son expresiones del tipo ***domain=<expresión>.***

- ***Entradas identificadas por un atributo con un valor particular.*** Son expresiones del tipo ***dnattr=<dn-valued attribute name>.*** Estas expresiones representan entradas cuyos nombres distinguidos se encuentran contenidos en un atributo de la entrada. La principal utilidad de esta expresión está en conceder acceso a grupos de usuarios.

El campo <access> indica el tipo de acceso que se concede al usuario. Los niveles de acceso se detallan a continuación de menos a más nivel. Cabe destacar que estar en posesión de un permiso implica estar en posesión de los jerárquicamente inferiores.

- ***None.*** No concede ningún tipo de acceso.
- ***Auth.*** Conde permiso al usuario para poder identificarse.
- ***Compare.*** Permite realizar operaciones de comparación.
- ***Search.*** Permite realizar operaciones de búsqueda basadas en filtros de búsqueda.
- ***Read.*** Este permiso se necesita para leer los resultados de cualquier operación que se realice en el directorio.
- ***Write.*** Concede permiso para modificar entradas.

A la hora de evaluar los controles de acceso al sistema el servidor Slapd atiende primero a las expresiones que se refieren a la base de datos actual, seguidas de las directivas globales.

Durante el proceso de evaluación primero lee el campo <what> y a continuación el campo <who> y finalmente lee el campo <access>.

El orden en el que las expresiones de control de acceso son declaradas en el fichero de configuración es importante. Se aconseja declarar primero aquellas expresiones que se refieran a aspectos más específicos tales como la selección de atributos o entradas, ya que si una entrada se encuentra englobada en más de una expresión prevalece el acceso concedido en la expresión que ha sido declarada en primer lugar.

3.7 Herramientas de gestión y mantenimiento de bases de datos.

La distribución Openldap incorpora además de los servidores Slapd y Slurpd una serie de clientes que permiten realizar consultas con el servidor así como un conjunto de aplicaciones que permiten mantener la base de datos.

3.7.1 La herramienta SLAPADD.

Esta aplicación permite la creación de una base de datos sin que el servidor Slapd esté activo, es decir, que esta aplicación se encarga de generar los ficheros necesarios de la base de datos para dar soporte a las entradas que se quieran añadir. Las entradas que debe incorporar la base de datos se especifican mediante un fichero en formato LDIF que se expone en el apartado "El formato LDIF".

La aplicación ***Slapadd*** utiliza la información de configuración del servidor Slapd contenida en el fichero de configuración.

Esta herramienta se ejecuta desde la línea de comandos con la sintaxis:

slapadd -l <inputfile> -f <slapdconfigfile> [-d <debuglevel>] [-n <integer>|-b <suffix>]

Los argumentos de la aplicación tienen el siguiente significado:

- ***-l <inputfile>.*** Indica el fichero en formato LDIF que contiene las entradas con las que va a crearse el directorio.

- ***-f <slapdconfigfile>.*** Indica la localización del fichero de configuración de Slapd con el fin de obtener información acerca de dónde guardar la información, qué índices deben crearse etc.

- ***-d <debuglevel>.*** Activa el proceso de monitorización del proceso al igual que en la aplicación Slapd.

- ***-n <databasenumber>.*** Es un argumento adicional que indica qupe base de datos debe ser modificada de entre las que se encuentren declaradas en el fichero de configuración. Un uno modificaría la primera, un dos la segunda y así sucesivamente. Esta opción no debe usarse con –b.

- ***-b <suffix>.*** Es un argumento adicional que indica qué base de datos debe ser modificada. En esta ocasión se modifica la base de datos cuyo parámetro suffix coincida con el campo <suffix>. Este parámetro no debe ser usado con –n.

3.7.2 La aplicación SLAPINDEX.

Esta herramienta permite actualizar los índices de una base de datos. Esta aplicación tiene su utilidad en situaciones en las que se tiene una base de datos definida con sus correspondientes índices y por alguna circunstancia en el fichero de configuración se modifica la información referente a los índices.

La aplicación ***Slapindex*** permite regenerar los índices de la base de datos sin tener que volver a generar de nuevo dicha base de datos.

El uso de este formato sigue la sintaxis:

slapindex -f <slapdconfigfile> [-d <debuglevel>] [-n <databasenumber> |-b <suffix>]

Los parámetros usados como argumentos tienen el mismo significado que los de la aplicación slapadd, con lo que pueden ser consultados en el apartado “La aplicación Slapadd”.

3.7.3 La aplicación SLAPCAT.

La aplicación ***Slapcat*** se usa para pasar una base de datos determinada a formato LDIF. De esta forma esta aplicación pasa información en formato LDBM a un formato más manejable como lo es el formato texto LDIF.

La sintaxis a seguir para usar esta aplicación es:

slapcat -l <filename> -f <slapdconfigfile> [-d <debuglevel>] [-n <databasenumber>|-b <suffix>]

Los parámetros usados como argumentos tienen el mismo significado que los de la aplicación slapadd, con lo que pueden ser consultados en el apartado “La aplicación Slapadd”.

3.7.4 La aplicación LDAPDELETE.

La aplicación ***Ldapdelete*** es un cliente LDAP que establece una conexión con el servidor LDAP se autentifica y borra una o más entradas del servicio de directorio.

Las entradas a borrar se indican dando sus nombres distinguidos en el formato definido por la RFC 2253 ya sea en la línea de comandos o a través de un fichero de texto.

La sintaxis de esta aplicación es:

ldapdelete [-n] [-v] [-k] [-K] [-c] [-C] [-M[M]] [-d debuglevel] [-f file] [-D binddn] [-W] [-w passwd] [-H ldapuri] [-h ldaphost] [-P 2|3] [-p ldapport] [-O security- properties] [-U username] [-x] [-I] [-Q] [-X authzid] [-Y mech] [-Z[Z]] [dn]...

Donde los parámetros indicados en la línea de comando tienen el siguiente significado:

- ***-n.*** Este parámetro indica que la operación debe ser simulada pero no ejecutada, con lo cual se indica el resultado pero no se borran las entradas. Es útil para ser usada junto con la opción –v.

- ***-v.*** Escribe diagnósticos de la operación por la salida estándar.

- ***-k.*** La aplicación usa autentificación ***Kérberos*** en lugar de autentificación simple. Esta opción es sólo válida si se instala el servidor con soporte para Kérberos.

- ***-K.*** Es un parámetro análogo a –k pero realiza la autentificación Kérberos en un solo paso.

- ***-C.*** Configura que las referencias deben ser seguidas.

- ***-c.*** Activa la operación en modo continuo. Esto significa que los posibles errores son comunicados pero la aplicación seguirá adelante con el siguiente nodo a borrar.

- ***-M[M].*** Activa el control de manejo de DSA IT. Si se pasa como MM hace que el control sea crítico.

- ***-d. <Debuglevel>.*** Establece el nivel de monitorización y depuración.

- ***-f <file>.*** Lee un fichero que permite reemplazar caracteres de la línea de comando del tipo ***"%s"*** por expresiones complejas tomadas del fichero.

- ***-x.*** Usa autentificación simple en lugar de SASL.

- ***-D <binddn>.*** Usa el nombre distinguido dado en el campo <binddn> para acceder al directorio.

- ***-W.*** Pide por teclado la clave del usuario.

- ***-w <passwd>.*** Usa el campo <passwd> con la clave de usuario en el proceso de autentificación.

- ***-H <ldapuri>.*** Especifica una URI que hace referencia al servidor LDAP.

- ***-h <ldaphost>.*** Especifica una dirección y número de puerto en el cual se encuentra funcionando el servidor. Este parámetro está en desuso a favor del parámetro –H.

- ***-p <ldapport>.*** Especifica un puerto TCP alternativo donde está funcionando el servidor. Se encuentra en desuso a favor del parámetro –H.

- ***-P { 2,3}.*** Especifica la versión del protocolo LDAP que va a usarse.

- ***-r.*** Esta opción borra además de la entrada especificada todas las entradas que cuelguen de ella.

- ***-O <security-properties>.*** Especifica propiedades específicas de SASL.

- ***-I.*** Activa el modo interactivo de autentificación SASL.

- ***-Q.*** Activa el modo de autentificación SASL no interactivo.

- ***-U <username>.*** Especifica el nombre del usuario en una autentificación SASL. La sintaxis del nombre depende del mecanismo SASL usado.

- ***-X <authzid>.*** Especifica el identificador de la autorización requerida en una autentificación SASL. El campo <authzid> debe tener el valor ***dn:_distinguished name*** o bien ***u:_username.***

- ***-Y <mech>.*** Indica el mecanismo SASL que va a usarse.

- ***-Z[Z].*** Emite la operación extendida "StartTLS". Si se usa como ZZ se requiere forzosamente la activación del protocolo TLS (Transport Layer Security) para llevar a cabo la operación.

3.7.5 La herramienta LDAPPASSWD

Esta aplicación establece una clave para un usuario LDAP. La clave que va a asociarse con el usuario debe ser dada en la línea de comandos, y en caso contrario será solicitada al usuario. Si se desactiva esta última opción de solicitar la clave al usuario, deberá ser el servidor el que genere una clave para el usuario.

Esta aplicación no está diseñada con el fin de actuar como una aplicación generadora de claves y no se encuentra optimizada para ello.

La sintaxis para el uso de esta aplicación es:

ldappasswd [-A] [-a oldPasswd] [-C] [-D binddn [-d debuglevel] [-H ldapuri] [-h ldaphost] [-n] [-p ldapport] [-S] [-s newPasswd] [-v] [-W] [-w passwd] [-O security-properties] [-I] [-Q] [-U username] [-x] [-X authzid][-Y mech] [-Z[Z]]

Los parámetros usados tienen el siguiente significado:

- ***-A.*** Pide al usuario la antigua clave en lugar de leerla de la línea de comando**.**

- ***-a <oldPasswd>.*** Establece la antigua palabra clave a través de la línea de comando.

- ***-n.*** No establece la nueva clave. Puede ser útil cuando se usa junto con las opciones –v o –d.

- ***-S.*** Pide la nueva clave al usuario, en vez de leerla de la línea de comando.

- ***-s.*** Establece la nueva clave a través de la línea de comando.

El resto de parámetros permitidos en esta aplicación tienen el significado que se indicó en el apartado “La aplicación ldapdelete”.

3.7.6 La aplicación LDAPSEARCH.

Esta aplicación establece una conexión con el servidor, se autentifica y solicita la realización de una operación de búsqueda usando un filtro que es una expresión lógica que recoge las condiciones que deben cumplir las entradas resultantes de la búsqueda. El formato de los filtros de búsqueda se recoge en la RFC 2254.

La sintaxis de la aplicación es:

ldapsearch [-n] [-u] [-v] [-k] [-K] [-t] [-A] [-C] [-L[L[L]]] [-M[M]] [-d debuglevel] [-f file] [-D binddn][-W] [-w bindpasswd] [-H ldapuri] [-h ldaphost] [-p ldap- port][-P 2|3] [-b searchbase] [-s base|one|sub] [-a never|always|search|find] [-l timelimit] [-z size- limit] [-O security-properties][-I] [-Q] [-U username] [-x] [-X authzid] [-Y mech] [-Z[Z]] filter [attrs...]

El significado de los parámetros se detalla a continuación:

- ***-u.*** Incluye el nombre del usuario tomado del nombre distinguido en la salida.

- ***-t.*** Escribe los valores de salida en un fichero temporal.

- ***-A.*** Devuelve sólo los identificadores de atributo pero no los valores particulares.

- ***-L.*** Devuelve los resultados en formato LDIF. Este parámetro puede ser pasado de la forma ***–LL*** desactivando la opción de devolver comentarios, y de forma ***–LLL*** que hace que los resultados no sean imprimidos por pantalla.

- ***-S <attribute>.*** Devuelve las entradas por orden alfabético en base al atributo dado en el campo <attribute>. Si este campo se configura como una cadena vacía las entradas se ordenan alfabéticamente atendiendo a su nombre distinguido.

- ***-b <searchbase>.*** El campo <searchbase> contiene el nombre de una entrada de la cual parte la búsqueda.

- ***-s { base, one, sub }.*** Especifica el ámbito de la búsqueda, pudiendo ser sólo en la entrada dada como base (base), sólo en los hijos de la entrada base (one), o en todo el subárbol que cuelga de la entrada base (sub).

- ***-a { never, always, search, find }.*** Especifica qué hacer con los alias que puedan encontrarse en la búsqueda, es decir, qué hacer con aquellas entradas que contienen un enlace a otra entrada dentro del mismo árbol. Las posibilidades son:

    - ***Never.*** Nunca sigue los alias.
    - ***Always.*** Sigue todos los alias que encuentre.
    - ***Search.*** Sólo sigue los alias durante la resolución de nombres, es decir que sólo seguiría un posible alias contenido en la entrada en la que se posiciona para realizar la búsqueda pero no el resto.
    - ***Find.*** Sólo sigue los alias tras la resolución de nombres, de manera que sigue los alias que puedan aparecer excepto el de la entrada en la que se posiciona.

- ***-l <timelimit>.*** Indica una cantidad en segundos que representa el tiempo máximo que puede durar una búsqueda. En caso de dar el valor cero, se interpreta como infinito.

- ***-z <sizelimit>.*** Indica el número máximo de entradas que pueden ser devueltas en una búsqueda. Si se configura con el valor cero, se interpreta como un número ilimitado.

El resto de opciones que aparecen en la sintaxis tienen el mismo significado que en la aplicación ldapdelte, con lo cual pueden consultarse en el apartado “La operación ldapdelete”.

3.7.7 La aplicación LDAPMODIFY.

La aplicación ***ldapmodify*** establece una conexión con el servidor se autentifica y realiza una operación de modificación de entradas del directorio. La información sobre las entradas a modificar y el tipo de modificación a realizarse sobre ellas puede especificarse bien desde la línea de comandos bien desde un fichero de texto.

La sintaxis para el uso de esta operación es:

ldapmodify [-a] [-b] [-c] [-C] [-r] [-n] [-v] [-k] [-K] [-M[M]] [-d ebuglevel] [-D binddn][-W] [-w passwd] [-H ldapuri] [-h ldaphost] [-p ldapport] [-P 2|3] [-O security-properties] [-I][-Q] [-U username] [-x] [-X authzid] [-Y mech] [-Z[Z]] [-f file]

Los parámetros que aparecen tienen el siguiente significado:

- ***-a.*** Permite añadir nuevas entradas al directorio invocando la aplicación ***ldapadd*** que puede consultarse en el apartado “La aplicación ldapadd”.

- ***-b.*** Interpreta los valores que comienzan con el carácter “/” como valores en formato binario y que el valor actual se encuentra en un fichero cuya ruta se especifica en el lugar donde los valores aparecen normalmente.

- ***-r.*** Reemplaza los valores existentes por defecto.

- ***-F.*** Sigue los cambios sin tener en cuenta el contenido de las líneas que comienzan con la directiva “replica”.

- ***-f <file>.*** Lee las modificaciones de un fichero de texto en lugar de leerlas de la línea de comandos.

El resto de los parámetros tienen el mismo significado que los de la aplicación ldapdelete con lo que pueden consultarse en el apartado “La aplicación ldapdelete”.

La manera más usada de indicar los cambios es mediante ficheros de texto.

El formato consiste en un conjunto de entradas separadas por una línea en blanco. El formato de una entrada es:

Distinguished Name (DN)
attr=value
[attr=value ...]

"DN" representa el nombre distinguido de la entrada, "attr" el identificador de un atributo, y "[attr]" un valor particular del atributo.

La operación realizada por defecto es la de añadir las entradas al directorio. Si se invoca la operación con el parámetro –r la operación que se realiza es reemplazar los valores de atributos existentes por los nuevos.

Si los atributos van precedidos por el carácter "-" y no llevan el carácter "=" ni un valor, se realiza operación de borrado.

En presencia del parámetro –r los atributos que se desean añadir deben ir precedidos por el carácter "+".

3.7.8 La aplicación LDAPADD.

Esta aplicación permite añadir nuevas entradas al directorio. La aplicación en sí es una variante de la operación ldapmodify, con lo cual se presenta su sintaxis, siendo el significado de sus parámetros el mismo que los de la operación ldapmodify sobre la que está construida. Dichos parámetros pueden consultarse en el apartado "La operación ldapmodify".

ldapadd [-b] [-c] [-C] [-r] [-n] [-v] [-k] [-K] [-M[M]] [-d debuglevel] [-D binddn] [-W] [-w passwd] [-h ldaphost] [-p ldapport] [-P 2|3][-E[E]] [-I[I]] [-U username] [-X authzid] [-Y mech] [-Z[Z]] [-f file]

3.7.9 La aplicación LDAPMODRDN.

Esta aplicación establece una conexión con el servidor, se autentifica y modifica el nombre distinguido relativo de una entrada determinada.

El nombre de la entrada a modificar y su nuevo nombre puede leerse de la línea de comandos o bien de un fichero de texto.

La sintaxis de la aplicación es:

Ldapmodrdn [-r] [-n] [-v] [-k] [-K] [-c] [-C] [-M[M]] [-d debuglevel] [-D binddn] [-W] [-w passwd] [-H ldapuri] [-h ldaphost] [-p ldapport] [-P2/3] [-O security-properties] [-I] [-Q] [-U username] [-x] [-X authzid][-Y mech] [-Z[Z]] [-f file] [dn rdn]

Donde los parámetros tienen el siguiente significado:

- ***-r.*** Borra todos los antiguos nombres relativos distinguidos que la entrada pudiera haber guardado como atributos (que es la opción por defecto).

- ***-f <file>.*** Lee la información acerca de las entradas de un fichero de texto en lugar de la línea de comandos.

El resto de opciones tienen el mismo significado que las opciones de la aplicación ldapdelete, con lo cual pueden consultarse en el apartado “La aplicación ldapdelete”.

3.8 Esquemas de directorio en OpenLDAP.

Un esquema de directorio es un conjunto de información con carácter administrativo almacenada en el directorio. Dicha información determina un conjunto de reglas que han de seguir tanto las entradas añadidas al árbol de directorio como sus atributos y valores particulares de éstos. El esquema también determina ciertos criterios que han de seguirse para realizar comparaciones.

Este tipo de reglas hacen referencia a los atributos obligatorios que un tipo de entrada debería contener y aquellos que podría tener pero que no son obligatorios. También se define el tipo de dato al que pertenece un atributo, los valores que podría tener, y los criterios que se siguen para realizar comparaciones sobre ellos.

El servidor Slapd guarda la información del esquema como si se trase de entradas del árbol de directorio, construyendo así el árbol de esquema, para lo cual mantiene enlaces con las siguientes entradas:

- ***Objectclass.*** Esta entrada sirve de raíz a un subárbol cuyas entradas representan definiciones de clases objeto, siendo una clase objeto el tipo de objeto que representa una entrada.

Cada clase objeto contiene información acerca de qué atributos debe contener una entrada que pertenezca a una clase objeto dada, qué atributos podría contener pero no son obligatorios, etc.

- ***AttributeDefinition.*** De esta entrada cuelga un subárbol cuyas entradas representan definiciones de atributo. Una definición de atributo hace referencia a cómo se construye un atributo concreto, cual es su identificador, qué tipo de dato contiene, etc.

- ***AttributeSyntaxDefintion.*** Sirve como raíz a un subárbol que contiene entradas que representan sintaxis de atributos, es decir que mantienen una cadena en formato OID (Object Identifier) que identifica un tipo de dato como puede ser un entero, una cadena de caracteres, etc.

- ***MatchingRules.*** Contiene un subárbol cuyas entradas representan criterios de comparación de atributos como pueden ser comparación carácter a carácter ignorando o no las mayúsculas, criterios numéricos etc.

En OpenLDAP el concepto de esquema se implementa a partir de una serie de ficheros de esquema que se especifican en el fichero de configuración y que contienen información referente a clases objeto y definiciones de atributos que son leídas e incorporadas al directorio en el momento en el que arranca el servidor, lo cual resta sentido a intentar una gestión dinámica del directorio, ya que cualquier tipo de cambio producido en el directorio durante su funcionamiento no es registrado.

El usuario no tiene control sobre las sintaxis de atributos ni sobre las reglas de comparación ya que estao están predefinidas en el servidor y no pueden ser modificados.

Cualquier objeto del esquema está identificado por un identificador global único (OID). Este tipo de identificadores son comunes en protocolos descritos por ASN 1., como por ejemplo SNMP (Simple Network Management Protocol).

Un desarrollador de esquemas que desee crear esquemas para uso particular que puedan se publicados debería obtener un OID registrado por ***"Internet Assigned Numbers Authority" (IANA).*** y a partir de él comenzar a desarrollar objetos para sus esquemas.

Se aconseja no usar OIDs fictíceos.

La documentación de OpenLDAP ofrece la siguiente tabla ilustrando el posible uso de OIDs para definir objetos en el esquema.

OID	Assignment
1.1	Organization's OID
1.1.1	SNMP Elements
1.1.2	LDAP Elements
1.1.2.1	AttributeTypes
1.1.2.1.1	myAttribute
1.1.2.2	ObjectClasses
1.1.2.2.1	myObjectClass

Un fichero de esquema en OpenLDAP contiene definiciones de tipos de atributos y clases objetos. El formato de este tipo de ficheros exige que las definiciones de atributos sean especificadas todas antes que cualquier definición de clase objeto, ya que las clases objetos usan las definiciones de atributos que han sido previamente definidas.

Una línea en el fichero de esquema que comience por el carácter “#” se considera una línea de comentario.

3.8.1 Definición de tipos de atributos.

Para definir tipos de atributos se usa la directiva ***attributetype***. Para ello se usa la descripción de tipos de atributos definida en la RFC 2252.

La sintaxis para la definición de un tipo de atributo es:

“whsp” representa el carácter espacio en blanco; “qdesc” y “qdstring” y “woid” representan cadenas de texto.

```
AttributeType = "(" whsp
     numericoid whsp            ; AttributeType identifier
    [ "NAME" qdescrs ]           ; name used in AttributeType
    [ "DESC" qdstring ]          ; description
    [ "OBSOLETE" whsp ]
    [ "SUP" woid ]              ; derived from this other
                                 ; AttributeType
    [ "EQUALITY" woid            ; Matching Rule name
```

```
[ "ORDERING" woid                    ; Matching Rule name
[ "SUBSTR" woid ]                    ; Matching Rule name
[ "SYNTAX" whsp noidlen whsp ]       ; see section 4.3
[ "SINGLE-VALUE" whsp ]              ; default multi-valued
[ "COLLECTIVE" whsp ]                ; default not collective
[ "NO-USER-MODIFICATION" whsp ]      ; default user modifiable
[ "USAGE" whsp AttributeUsage ]      ; default userApplications
whsp ")"
```

Cada definición de atributo comienza con la directiva “attributetype” seguida del carácter “=”. A continuación se indican entre paréntesis los parámetros que construyen el tipo de atributo.

El significado de los parámetros es:

- ***NUMERICOID.*** Este campo contiene el OID del objeto.
- ***NAME.*** Es el nombre del objeto, es decir, el identificador de atributo.
- ***DESC.*** Es una cadena que describe el uso del atributo.
- ***OBSOLETE.*** Indica si el atributo se encuentra en desuso.
- ***SUP.*** Hace referencia a otro tipo de atributo al cual éste extiende, de manera que hereda de él toda la información definida en ese tipo de atributo.
- ***EQUALITY.*** Indica el criterio que se usa para determinar la igualdad en una comparación.
- ***ORDERING.*** Indica un criterio de ordenación en operaciones de comparación.
- ***SUBSTR.*** Indica la regla que debe seguirse para realizar comparaciones en subcadenas.
- ***SYNTAX.*** Se indica el OID que identifica el tipo de dato al que representa el atributo. A veces se acompaña de un entero entre llaves que representa el tamaño del tipo de dato.
- ***SINGLE-VALUE.*** Indica que el atributo sólo puede tener un valor particular.

- ***COLLECTIVE.*** Indica que el atributo debe tener más de un valor.

- ***NO-USER-MODIFICATION.*** Hace que el atributo no pueda ser modificado por el usuario.

- ***USAGE.*** Es una cadena que indica para qué se usa el atributo.

En caso de querer usar más de una cadena como argumento de un parámetro ha de usarse para separarlas el carácter “$”.

Se muestra un ejemplo de la definición de un tipo de atributo con el fin de ilustrar lo anterior:

```
attributeType ( 2.5.4.41 NAME 'name'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} )
attributeType ( 2.5.4.3 NAME
    ( 'cn' $ 'commonName' ) SUP name )
```

OpenLDAP ofrece la siguiente tabla para ilustrar los distintos tipos de sintaxis de atributos soportadas así como sus identificadores OIDs.

Name	OID	Description
binary	1.3.6.1.4.1.1466.115.121.1.5	BER/DER data
boolean	1.3.6.1.4.1.1466.115.121.1.7	boolean value
distinguishedName	1.3.6.1.4.1.1466.115.121.1.12	DN
directoryString	1.3.6.1.4.1.1466.115.121.1.15	UTF-8 string
IA5String	1.3.6.1.4.1.1466.115.121.1.26	ASCII string
Integer	1.3.6.1.4.1.1466.115.121.1.27	integer
Name and Optional UID	1.3.6.1.4.1.1466.115.121.1.34	DN plus UID
Numeric String	1.3.6.1.4.1.1466.115.121.1.36	numeric string
OID	1.3.6.1.4.1.1466.115.121.1.38	object identifier
Octet String	1.3.6.1.4.1.1466.115.121.1.40	arbitary octets
Printable String	1.3.6.1.4.1.1466.115.121.1.44	printable string

OpenLDAP también proporciona la siguiente tabla recogiendo las reglas de comparación soportadas:

Name	Type	Description
booleanMatch	equality	boolean
objectIdentiferMatch	equality	OID
distinguishedNameMatch	equality	DN
uniqueMemberMatch	equality	DN with optional UID
numericStringMatch	equality	numerical
numericStringOrderingMatch	ordering	numerical
numericStringSubstringsMatch	substrings	numerical
caseIgnoreMatch	equality	case insensitive, space insensitive
caseIgnoreOrderingMatch	ordering	case insensitive, space insensitive
caseIgnoreSubstringsMatch	substrings	case insensitive, space insensitive
caseExactMatch	equality	case sensitive, space insensitive
caseExactOrderingMatch	ordering	case sensitive, space insensitive
caseExactSubstringsMatch	substrings	case sensitive, space insensitive
caseIgnoreIA5Match	equality	case insensitive, space insensitive
caseIgnoreIA5OrderingMatch	ordering	case insensitive, space insensitive
caseIgnoreIA5SubstringsMatch	substrings	case insensitive, space insensitive
caseExactIA5Match	equality	case sensitive, space insensitive
caseExactIA5OrderingMatch	ordering	case sensitive, space insensitive
caseExactIA5SubstringsMatch	substrings	case sensitive, space insensitive

3.8.2 Definición de clases objeto.

Para definir clases de objetos en un fichero de esquema se usa la directiva "***Objectclass"*** que sigue las reglas definidas en la RFC 2252.

La sintaxis que ha de usarse para añadir una clase objeto se muestra a continuación donde "whsp" representa un espacio en blanco, y "qdescrs"y "qdstring" representan cadenas de texto, y "oids" representa una lista entre paréntesis de nombres de identificadores de atributos previamente defindos separados por el carácter "$".

```
ObjectClass = "(" whsp
      numericoid whsp      ; ObjectClass identifier
      [ "NAME" qdescrs ]
      [ "DESC" qdstring ]
      [ "OBSOLETE" whsp ]
```

```
[ "SUP" oids ]      ; Superior ObjectClasses
[ ( "ABSTRACT" / "STRUCTURAL" / "AUXILIARY" ) whsp
                                        ; default structural
[ "MUST" oids ]     ; AttributeTypes
[ "MAY" oids ]      ; AttributeTypes
whsp ")"
```

Los parámetros usados en la definición de la clase objeto tienen el siguiente significado:

- ***NUMERICOID.*** Este campo contiene el OID del objeto.

- ***DESC.*** Es una cadena que describe el contenido de la clase objeto.

- ***OBSOLETE.*** Indica si la clase objeto se encuentra en desuso.

- ***SUP.*** Hace referencia a otra clase objeto a la cual ésta extiende, de manera que hereda de ella toda la información definida en esa clase objeto.

- ***ABSTRACT. STRUCTURAL. AUXILIARY.*** Estos identificadores hacen referencia a la naturaleza de la clase objeto:

  - ***ABSTRACT*** indica que la clase objeto es abstracta y que por tanto no puede instanciarse, sólo usarse para derivar nuevas clases de ella.
  - ***STRUCTURAL*** define las clases propiamente dichas que son usadas para guardar información de esquema.
  - ***AUXILIARY*** hace referencia a clases objetos que se guardan para fines administrativos.

- ***MUST.*** Indica los atributos que una entrada que pertenezca a esta clase objeto debe contener obligatoriamente.

- ***MAY.*** Indica los atributos que una entrada del directorio puede contener.

A continuación se adjunta un ejemplo de la definición de una clase objeto para ilustrar lo anterior.

```
objectclass ( 1.1.2.2.2 NAME 'myPerson'
    DESC 'my person'
    SUP inetOrgPerson
    MUST ( 'myUniqueName' $ 'givenName' )
    MAY 'myPhoto' )
```

3.9 La aplicación SLURPD.

La aplicación Slurpd funciona como un demonio que permite al servidor Slapd maestro pasar su información a otros servidores Slapd esclavos que la replican.

De esta manera cuando se produce una modificación en el directorio maestro, Slurpd despierta y contacta con los servidores esclavos para comunicarles los cambios que se han producido.

La ventaja es que se libera a Slapd de tener que comunicar los cambios y contactar con los esclavos.

El funcionamiento de este servidor sigue los siguientes pasos:

- Un cliente solicita una operación que implica una cierta actualización de la información en un servidor Slapd esclavo.

- El servidor esclavo devuelve al cliente una remisión para que el cliente se dirija al servidor maestro.

- El cliente LDAP intenta entonces realizar la operación de modificación sobre el servidor maestro.

- El servidor lleva a cabo la operación y notifica el proceso registrándolo en el correspondiente fichero de registro (log file) y devuelve al cliente una indicación para informarle de que la operación se ha realizado.

- A continuación entra en juego el servidor Slurpd que se percata de que se ha producido una modificación ya que permanece a la escucha del fichero de registro, lee la modificación y se la envía al servidor Slapd esclavo.

- Para finalizar el proceso el esclavo Slapd modifica su información y devuelve un mensaje indicando que ha llevado a cabo la petición.

Capítulo 4: El API JNDI.

En este apartado se describe el API JNDI que es el que utiliza la aplicación que implementa el proxy CORBA-LDAP para acceder al servidor de directorio.

4.1 Sistemas de nombrado.

El servicio de nombrado es una facilidad fundamental en cualquier sistema de ordenadores. El servicio de nombrado nos permite asociar nombres con objetos y localizar estos últimos basándose en sus nombres. Por ejemplo, cuando queremos acceder a un determinado fichero debemos suministrar su nombre.

La función principal por tanto de un servicio de nombres es mapear nombres amigables para el usuario con objetos tales como direcciones, identificadores o cualquier tipo de objeto típicamente usados por programas de ordenador.

Ejemplos de sistemas de nombrado son:

- Un servidor DNS (Internet Domain Name System) mapea nombres de máquinas fáciles de manejar por el usuario (por ejemplo www.sun.com) a direcciones IP.

- Un sistema de ficheros mapea un nombre de fichero (por ejemplo /home/usr/local/etc/openldap/slapd.conf) a un manejador de fichero

que el programa pueda usar para acceder a los contenidos del fichero.

Para localizar un objeto en un sistema de nombrado hemos de suministrar el ***nombre*** del objeto. Dentro de un sistema de nombrado dado se determina la sintaxis que deben seguir los nombres. Esta sintaxis se conoce como ***convención de nombrado***

Ejemplos de convenciones de nombrado son:

- En Unix los ficheros se nombran mediante un nombre PATH que recorre los directorios desde la raíz (/) hasta el propio fichero, separándolos por el carácter "/".
  /home/usr/local/etc/openldap/slapd.conf.

- La convención de nombrado de Lightweight Directory Access Protocol (LDAP), ordena los componentes de derecha a izquierda separándolos por comas (','), de manera que en el nombre cn=Rosanna Lee,o=Sun,c=US nombra una entrada LDAP cn=Rosanna Lee relativa a la entrada o=Sun que a su vez es relativa a c=us.

  Nótese que en LDAP cada componente del nombre debe ser una pareja nombre/valor con el nombre y el valor separados por el signo igual('=').

La asociación de un nombre con un objeto dentro de un sistema de nombrado se conoce como ***unión.***

Según la naturaleza del servicio de nombres y del tipo de objetos, algunos objetos no pueden ser almacenados directamente, es decir, no se puede guardar una copia del objeto dentro del servicio de nombrado. La solución en este caso es guardar los objetos por referencia, es decir, guardar un ***puntero*** o una ***referencia*** al objeto en el sistema de nombrado.

Una ***referencia*** es información sobre como acceder a un objeto, es decir, una representación compacta del objeto que sirve para acceder al objeto en si y obtener más información sobre él.

Por ejemplo, un objeto avión, podría contener una lista de pasajeros y de tripulación, su plan de vuelo, el estado de combustible y de los instrumentos, y su número de vuelo y hora de salida. Por contraste, una

referencia a un objeto avión, sólo podría contener su número de vuelo y su hora de salida.

Un ***contexto*** es un conjunto de uniones nombre a objeto. Cada contexto tiene una convención de nombrado asociada.

Un contexto proporciona una operación de localización (***resolución***) que devuelve el objeto y podría proporcionar operaciones como aquellas para unir nombres, desunir nombres y listar uniones de nombres.

Un nombre en un objeto contexto puede unirse a otro objeto contexto (llamado ***subcontexto***) que tenga la misma convención de nombrado.

Por ejemplo, un directorio de ficheros, como **/usr**, en el sistema de ficheros UNIX es un contexto. Un directorio de ficheros nombrado en relación a otro directorio de ficheros es un un subcontexto (algunos usuarios UNIX se refieren a esto como un **subdirectorio**). Es decir, en un directorio de fichero **/usr/bin**, el directorio **bin** es un subcontexto de **usr**.

Un sistema de nombrado está conectado a un conjunto de contextos del mismo tipo, es decir, que tienen la misma convención de nombrado y proporciona un conjunto común de operaciones.

Un ***espacio de nombres*** es el conjunto de nombres en un sistema de nombrado. Por ejemplo, el espacio de nombres LDAP contiene nombres de entradas LDAP.

4.2 El concepto de directorio.

El ***servicio de directorio*** supone una ampliación del servicio de nombrado. El servicio de directorio asocia nombres con objetos y también permite a dichos objetos tener ***atributos***. De esta manera no sólo podemos localizar un objeto por su nombre sino que también podemos obtener sus atributos o localizar el objeto basándonos en sus atributos.

Un ejemplo es el servicio de guías de la compañía telefónica. Mapea un nombre de suscriptor a su dirección y número de teléfono. A diferencia del ejemplo el servicio directorio del ordenador puede usarse y actualizarse en línea.

Los atributos son objetos contenidos en los objetos del directorio y que describen los objetos a los que representan (por ejemplo. Un usuario podría

estar representado por un objeto directorio que tenga como atributos la dirección e-mail del usuario, varios números de teléfono, dirección postal, e información de usuario).

Un atributo se compone de un ***identificador de atributo*** y un conjunto de ***valores de atributo.*** El identificador de atributo es un token que identifica un atributo independientemente de sus valores (por ejemplo Por ejemplo, dos cuentas de ordenador diferentes podrían tener un atributo **"mail"**; **"mail"** es el identificador del atributo. Un valor de atributo es el contenido del atributo. La dirección e-mail, por ejemplo, podría tener un identificador de atributo de **"mail"** y el valor de atributo de **john-smith@somewhere.com)**.

Un ***servicio de directorio*** proporciona operaciones para crear, añadir, eliminar y modificar los atributos asociados con los objetos de un directorio.

Podemos localizar un objeto directorio suministrando su nombre al servicio de directorio. De forma alternativa, muchos directorios, como aquellos basados en LDAP, soportan nociones de ***búsquedas***. Cuando buscamos, podemos sólo suministrar una consulta que consiste en una expresión lógica en la que especificamos atributos que el objeto u objetos deben tener. Este criterio de consulta se llama ***filtro de búsqueda***. Este estilo de búsqueda algunas veces es llamado ***localización invertida*** o ***búsqueda basada en contenido.***

El servicio de directorio busca y devuelve los objetos que cumplan el filtro de búsqueda.

Los directorios normalmente ordenan sus objetos en árboles. Por ejemplo, LDAP ordena todos sus objetos directorio en un árbol llamado ***Directory Information Tree (DIT).***

4.3 Uso tradicional de directorios.

Una aplicación que permita directorios es una aplicación que usa un servicio de nombres o de directorio. Estas aplicaciones pueden usar el directorio para almacenar y recuperar atributos de objetos de directorio.

Además las aplicaciones pueden compartir infraestructuras comunes proporcionadas por el directorio. Esta compartición hace que las aplicaciones que son desarrolladas a través del sistema, e incluso de la red, sean más coherentes y manejables ya que la información se encuentra disponible para que la usen todas las aplicaciones y servicios que lo requieran.

4.4 Introducción al API JNDI.

El ***Java Naming and Directory Interface (JNDI*)** es un interface de programación (API) que proporciona funcionalidades de nombrado y directorio a las aplicaciones escritas usando Java. Está definido para ser independiente de cualquier implementación de servicio de directorio. Así se puede acceder a una gran variedad de directorios (nuevos, emergentes, y ya desarrollados), de una forma común.

4.5 Principios de diseño de JNDI.

El diseño del API JNDI para Java se basa en los siguientes principios:

- El diseño de JNDI se ha hecho partiendo de los componentes Java existentes y haciéndolo consistente con las principales clases del núcleo de la plataforma Java.

- Se persigue que un desarrollador interesado en usar las prestaciones de un servicio de directorio no tenga por qué tener conocimientos avanzados de ese servicio de directorio para poder usar JNDI.

- Ofrece un interface único de programación para múltiples sistemas de directorio, de manera que se tiene un conjunto de clases que permite el acceso a dichos sistemas independientemente de características puntuales propias del sistema de directorio.

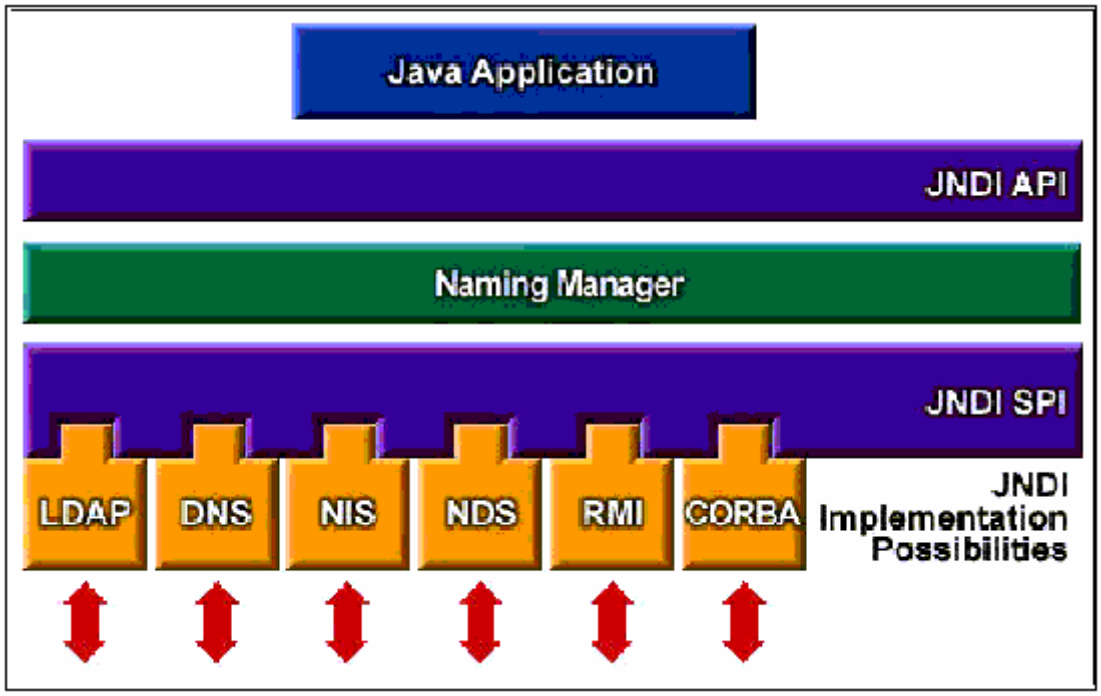
- Permitir que nuevas aplicaciones que implementen servicios de nombres o directorios puedan ser usadas por JNDI.

- Modularidad. Se busca hacer independiente el manejo de sistemas de nombrado y de directorio, así como la definición de un conjunto de funciones básicas que puedan usar otras funciones que implementen aspectos particulares del sistema que se desea usar.

4.6 Arquitectura de JNDI.

La arquitectura JNDI consiste en un API y un "service provider interface (SPI)" (proveedor de servicios). Las aplicaciones Java usan el API JNDI para acceder a una gran variedad de servicios de nombres y directorios. El SPI permite conectar de forma transparente una gran variedad de servicios de nombres y directorios, por lo tanto permite a las aplicaciones Java usar el API JNDI para acceder a sus servicios.

De esta manera se consigue escribir aplicaciones JAVA que usan JNDI para acceder a servicios de directorio sin tener que manejar la comunicación a nivel de conexiones, que es una tarea de la que se encarga el proveedor de servicios.



4.7 Empaquetado de JNDI.

JNDI está incluido en el JAVA 2 SDK, v1.3 y en la versión 1.4.

También puede disponerse de JNDI como una librería Java para ser usada con las distribuciones JDK 1.1 Y Java 2 SDK v1.2 de Java.

Para usar JNDI hemos de disponer de las clases JNDI y de al menos un proveedor de servicios. Tanto las versiones 1.3 como 1.4 de JAVA 2 SDK incluyen tres proveedores de servicios para usarse con los siguientes servicios de nombres/directorios.

- Lightweight Directory Access Protocol (LDAP).
- Common Object Request Broker Architecture (CORBA), servicio de nombres Common Object Services (COS).
- Registro de Java Remote Method Invocation (RMI).

Si estamos usando algunas de las distribuciones anteriormente mencionadas y que no incorporan JNDI, además de instalar las librerías con las clases JNDI correspondientes hemos de usar un proveedor de

servicios. Podemos encontrar alguno disponible en la web de JNDI, o bien obtenerlo de otros vendedores.

JNDI está dividido en cinco paquetes.

- Javax.naming
- Javax.naming.directory
- Javax.naming.event
- Javax.naming.ldap
- Javax.naming.spi

4.7.1 El paquete javax.naming.

El paquete ***javax.naming*** contiene las clases e interfaces necesarios para acceder a servicios de nombrado.

La base del paquete javax.naming es el interface ***Context*** que es el que permite implementar el concepto de contexto en un sistema de nombrado. Así mismo ofrece métodos para localizar, unir, desunir, renombrar objetos, crear subcontextos o bien destruirlos.

Cada método del interface Context tiene dos sobrecargas: una que acepta un argumento del tipo Name y otra que acepta un nombre del tipo java.lang.String.

Name es un interface que representa un nombre genérico, es decir una secuencia ordenada de cero o más componentes.

Con un objeto Name podemos representar un ***nombre pegado***, es decir un nombre que expanda varios espacios de nombres.

La principal aplicación de las sobrecargas que aceptan Name se encuentra en poder manipular nombres, es decir, componerlos, comparar componentes etc.

Una de las funciones principales que nos ofrece el interface javax.naming es el método lookup() que nos permite realizar búsquedas dentro del servicio de nombrado.

Los resultados de la búsqueda pueden ser devueltos de dos formas.

- Mediante ***uniones***. El cliente recibe una enumeración de uniones nombre a objeto. Cada unión está representada por un ejemplar de la clase ***Binding*** y contiene el nombre del objeto unido, el nombre de la clase objeto y el propio objeto. Un método característico de este interface y que devuelve uniones es ***listBindings().***

- Mediante enumeraciones de ***NameClassPair***. La clase NameClassPair contiene un nombre de objeto y el nombre de la clase del objeto. Este tipo de dato es más fácil de manejar que las uniones ya que no incorpora el objeto en si, lo cual permite una operación menos costosa. Un método característico de este interface y que devuelve datos de este tipo es ***list().***

Las principales clases que componen el interface son:

- ***Reference.*** Esta clase permite representar una referencia, la cual contiene información sobre como construir una copia del objeto.

- ***InitialContext.*** En JNDI todas las operaciones de nombrado y directorio se realizan en relación a un contexto. Esto implica que no hay raíces absolutas, y que para empezar a trabajar hay que definir un contexto inicial que proporcione un punto de arranque para la realización de las operaciones de nombrado y directorio. Mediante esta clase definimos un contexto que nos servirá como partida para dichas operaciones.

- ***NamingException.*** JNDI define un árbol de clases para excepciones que se pueden lanzar durante la realización de operaciones de nombrado y directorio. La raíz de este árbol de clases es NamingException.

4.7.1.1 Contenido del paquete javax.naming.

A continuación se detallan los interfaces, clases y excepciones que incorpora el paquete java.naming.

- ***INTERFACES:***
  - ***Context.*** Representa un contexto de nombrado que es un conjunto de uniones de nombres y objetos dentro de un sistema de nombrado.
  - ***Name.*** Representa un nombre genérico en JNDI, es decir una secuencia ordenada de componentes que podrían responder a distintas convenciones de nombrado.
  - ***NameParser.*** Contiene métodos para analizar nombres de un sistema de nombrado jerárquico.
  - ***NamingEnumeration.*** Ofrece métodos para manejar listas devueltas por métodos de búsqueda.
  - ***Referenciable.*** Este interface es implementado por objetos que pueden proporcionar una referencia a ellos mismos.

- ***CLASES:***
  - ***BinaryRefAddr.*** Representa la dirección en formato binario del punto al que se desea acceder en la comunicación.
  - ***Binding.*** Representa una unión, es decir la asociación de un nombre con un objeto.
  - ***CompositeName.*** Representa un nombre compuesto, es decir, una secuencia de nombres que expanden múltiples sistemas de nombrado, pudiendo tener cada cual una convención de nombrado particular.
  - ***CompoundName.*** Representa un nombre dentro de un espacio de nombrado jerárquico.
  - ***InitialContext.*** Define el contexto inicial que permite acceder al sistema de nombrado y realizar operaciones.
  - ***LinkRef.*** Representa una referencia cuyo contenido es un nombre que está unido a un nombre en un contexto.
  - ***NameClassPair.*** Define el par objeto y nombre de clase objeto, de cualquier unión encontrada en un contexto.
  - ***RefAddr.*** Representa la dirección a la cual se quiere acceder en una comunicación.
  - ***Reference.*** Representa una referencia a un objeto que se encuentra fuera del sistema de nombrado o de directorio.

- ***StringRefAddr.*** Es la versión en formato cadena de la dirección a la cual se desea acceder en una comunicación.

➢ ***EXCEPCIONES:***

- ***AuthenticationException.*** Esta excepción se lanza cuando ocurre un error relacionado con la autentificación durante el acceso a un sistema de nombrado o de directorio.
- ***AuthenticationNotSupportedException.*** Se lanza cuando no se soporta el método de autentificación que se intenta utilizar.
- ***CannotProceedException.*** Se lanza cuando la operación alcanza un punto a partir del cual no puede continuar.
- ***CommunicationException.*** Esta excepción se lanza cuando el cliente es incapaz de comunicarse con el servidor de nombrado o de directorio.
- ***ConfigurationException.*** Se lanza cuando existe un problema de configuración.
- ***ContextNotEmptyException.*** Se lanza cuando se intenta eliminar un contexto que no está vacío.
- ***InsufficientResourcesException.*** Se lanza cuando no hay fuentes disponibles para realizar la operación solicitada.
- ***InterruptedNamingException.*** Se lanza cuando se interrumpe una operación de nombrado que estaba comenzando a realizarse.
- ***InvalidNameException.*** Esta excepción indica que el nombre proporcionado con el cliente no responde a la sintaxis del sistema al que se desea acceder.
- ***LimitExceededException.*** Esta excepción se lanza cuando una operación no puede llevarse a cabo por alguna limitación impuesta por el sistema o por el propio usuario.
- ***LinkException.*** Esta excepción se usa para notificar problemas ocurridos durante la resolución de conexiones.
- ***LinkLoopException.*** Se lanza cuando se detecta un bucle dentro durante la resolución de nombres o de conexiones.
- ***MalformedLinkException.*** Se lanza cuando se encuentra un enlace erróneo durante el establecimiento de una conexión.
- ***NameAlreadyBoundException.*** Es lanzada para indicar que un una unión no puede ser añadida porque el nombre que se intenta dar a la unión ya se encuentra en uso.
- ***NameNotFoundException.*** Se lanza cuando un componente de un nombre no puede ser resuelto porque no existe una unión que lo contenga.

- ***NamingException.*** Es la superclase de todas las excepciones lanzadas por las operaciones de los interfaces Context y DirContext.
- ***NamingSecurityException.*** Es la superclase de las excepciones relacionadas con la seguridad que son lanzadas por las operaciones de los interfaces Context y DirContext.
- ***NoInitialContextException.*** Se lanza cuando no puede crearse un contexto inicial.
- ***NoPermissionException.*** Esta excepción es lanzada cuando el cliente intenta ejecutar una operación para la cual no tiene permiso.
- ***NotContextException.*** Se lanza cuando una operación intenta acceder a un objeto apuntado por una operación de nombrado pero este objeto no es del tipo contexto.
- ***OperationNotSupportedException.*** Se lanza cuando un contexto no soporta la operación que se intenta ejecutar sobre él.
- ***PartialResultException.*** Se lanza para indicar que sólo una parte del resultado de la operación ha podido ser devuelto y que por tanto la operación no ha podido completarse.
- ***ReferralException.*** Es una clase abstracta que representa una excepción generada por un servidor LDAP v3 cuando encuentra una remisión.
- ***ServiceUnavailableException.*** Se lanza cuando se intenta acceder un servicio de directorio o de nombrado que no está disponible.
- ***SizeLimitExceededException.*** Esta excepción es lanzada cuando un método produce una respuesta que supera el límite de tamaño permitido.
- ***TimeLimitExceededException.*** Se lanza cuando una operación no es llevada a cabo dentro de unos límites de tiempo impuestos.

4.7.2 El paquete javax.naming.directory.

Extiende el paquete javax.naming proporcionando funcionalidad para acceder a servicios de directorio además de a servicios de nombrado.

Este paquete permite a las aplicaciones que los usan recuperar objetos atributos asociados con objetos almacenados en el directorio.

La base de este paquete es el interface ***DirContext*** que implementa un contexto de directorio y proporciona métodos que permiten operar con atributos asociados con un objeto de directorio.

Este paquete está construido como una extensión del interface Context con lo cual cualquier objeto de directorio representado por una clase de este paquete puede ser considerado también como un contexto de nombrado.

Entre las principales operaciones que ofrece el interface cabe destacar:

- ***GetAttributes()*** que permite recuperar los atributos del objeto del directorio que está siendo representado por el contexto sobre el que aplicamos el método.

- ***ModifyAttributes()*** que permite realizar operaciones sobre los atributos del objeto que está siendo representado por el contexto sobre el que se aplica el método. Las operaciones pueden realizarse sobre los atributos son:
  - Añadir: Permite añadir un nuevo atributo al objeto o bien añadir un nuevo valor particular a un atributo ya existente.
  - Borrar: Permite eliminar un atributo con todos sus valores particulares, o bien eliminar un solo valor particular del atributo.
  - Reemplazar: Permite sustituir todos los valores particulares de un atributo dado por un nuevo valor particular.

- ***Search()*** que permite realizar búsquedas en el directorio a partir del contenido de un conjunto de atributos y valores dados. En sistemas de directorios avanzados es posible proporcionar junto con la operación de búsqueda una serie de controles adicionales que definan el número máximo de entradas que pueden ser recibidas, el ámbito de la búsqueda, etc; También suelen permitir el uso de filtros de búsqueda, que son expresiones lógicas que deben cumplir las entradas del directorio para entrar en la búsqueda. El formato de estas expresiones lógicas para LDAP se encuentra en la RFC 2254.

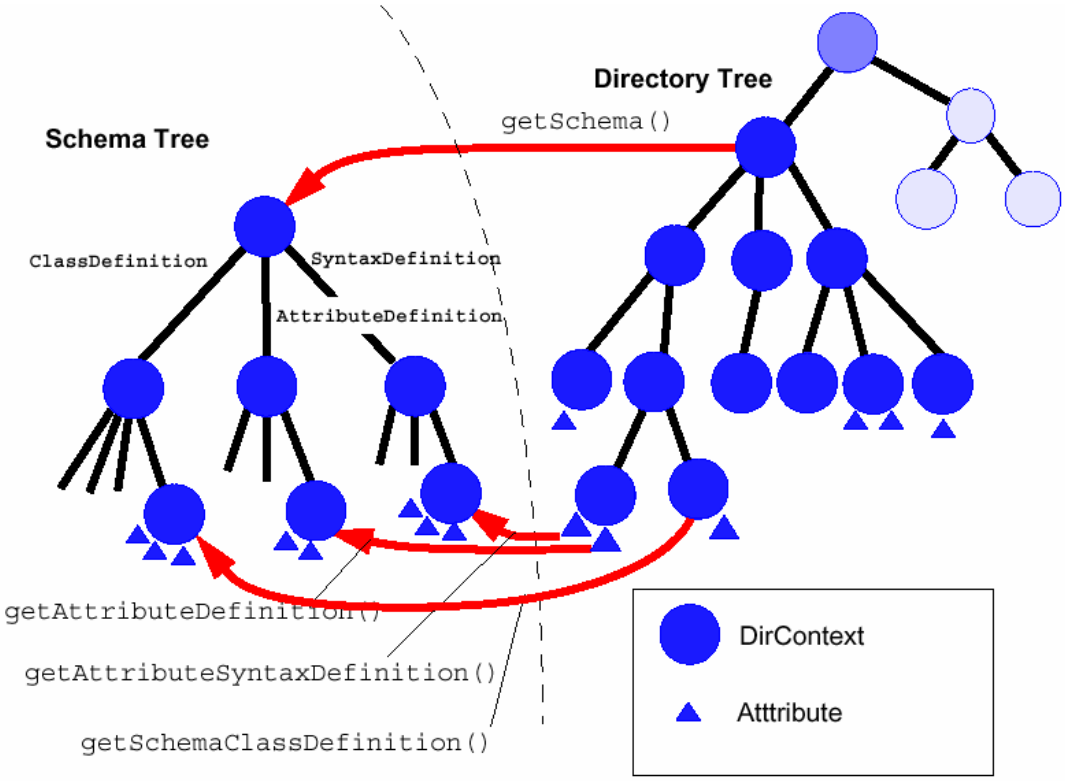
- ***Operaciones de esquema:*** Un esquema en un sistema de directorio define una serie de normas que dictan cómo debe organizarse la información del directorio y los atributos que pueden almacenarse en un objeto concreto, a nivel de tipo de datos, posibles valores, si se exige o no su presencia etc.

  El hecho de que los sistemas de directorio normalmente guarden la información de directorio siguiendo una estructura en forma de árbol, siguiendo el mismo modelo que se aplica a la información ordinaria, hace que JNDI sea igualmente válido para acceder al esquema y poder navegar en él de la misma manera que se hace con la información del directorio.

  El árbol que contiene la información de esquema suele tener una estructura basada en un nodo raíz del cual cuelgan cuatro nodos que son:

  - ***ClassDefinition***: De este nodo cuelgan nuevos nodos que representan las clases objetos que son admisibles dentro del esquema de directorio. El concepto ***de clase objeto*** viene a determinar el tipo al que pertenece un nodo del directorio; es decir que todo nodo del directorio debe ser de una clase objeto determinada, la cual aporta información sobre qué atributos debería tener obligatoriamente la entrada y qué atributos podría tener.
  - ***AttributeDefinition:*** De esta entrada cuelgan entradas que representan ***definiciones de atributos***, es decir, que contienen información acerca de cómo construir un tipo de atributo que se quiere usar en una entrada del árbol. De esta manera todos los atributos definidos en el esquema deben tener su correspondiente definición de atributo (AttributeDefinition) en la cual se indica el tipo de dato que contiene el atributo (cadena de caracteres, enteros etc), y los criterios a seguir a la hora de realizar comparaciones sobre dicho atributo, es decir las reglas que han de seguirse a la de comparar el atributo con un valor dado.
    - ***SyntaxDefinition***: Los hijos de esta entrada representan ***sintaxis de atributos***, es decir, contienen información sobre cómo construir un tipo de datos concreto como podría ser una cadena de caracteres, un entero etc.
    - ***MatchingRules:*** De este nodo cuelgan nodos que contienen información acerca de los criterios que se usan para realizar comparaciones sobre un tipo de datos concreto. Un ejemplo

de este tipo de criterios es ***CaseIgnoreMatch***, que indica que una cadena es igual a otra si coincide carácter a carácter sin importar que sean mayúsculas o minúsculas.



Para el manejo de este tipo de información se definen una serie de métodos muy útiles que permiten el acceso al árbol de esquema:

- ***GetSchema():*** Devuelve un objeto del tipo DirContext que apunta a la raíz del árbol de esquema.
- ***GetSchemaClassDefinition():*** Devuelve un objeto del tipo DirContext que apunta a un nodo que contiene información acerca de la clase objeto deseada.
- ***GetAttributeDefinition():*** Es un método de comportamiento análogo al anterior, pero en este caso el contexto devuelto contiene información sobre la definición de un tipo de atributo concreto.

- ***GetAttributeSyntaxDefiniton():*** El comportamiento es análogo al de las funciones anteriores, pero en este caso el nodo que devuelve contiene información acerca de una sintaxis de atributo.

4.7.2.1 Contenido del paquete javax.naming.directory.

A continuación se detallan los interfaces, clases y excepciones que componen el paquete javax.naming.directory junto con una explicación de su funcionalidad.

- ***INTERFACES:***
  - ***Attribute.*** Este interface implementa un atributo asociado con un objeto de un servicio de directorio.
  - ***Attributes.*** Este interface representa el conjunto de atributos de una entrada del servicio de directorio.
  - ***DirContex.*** Este interface proporciona métodos que permiten realizar operaciones sobre los objetos de un servicio de directorio y sus atributos.

- ***CLASES:***
  - ***BasicAttribute.*** Implementa el interface Attribute, y por tanto mapea el concepto de atributo en una clase JNDI.
  - ***BasicAttribute.*** Implementa el interface Attributes, es decir mapea en una clase Java el conjunto de atributos de una entrada de un servicio de directorio.
  - ***InitialDirContext.*** Esta clase permite crear un contexto de directorio para acceder a los objetos de un servicio de directorio.
  - ***ModificationItem.*** Representa una serie de modificaciones que se desean realizar sobre los atributos de una entrada concreta de un sistema de directorio.
  - ***SearchControls.*** Esta clase sirve como encapsulado a una serie de controles que se envían junto con la operación de búsqueda en un servicio de directorio y que definen aspectos concretos sobre cómo debe llevarse a cabo dicha búsqueda.
  - ***SearchResult.*** Cada elemento de una enumeración de un objeto de la clase NamingEnumeration es del tipo SearchResult. Por tanto esta clase sirve como contenedor de resultados particulares de una búsqueda.

- ***EXCEPCIONES:***
  - ***AttributeInUseException.*** Esta excepción se lanza cuando el cliente intenta añadir un atributo ya existente a una entrada de un sistema de directorio.
  - ***AttributeModificationException.*** Esta excepción es lanzada cuando se intenta realizar una operación no permitida por el esquema sobre un determinado atributo o grupo de atributos.
  - ***InvalidAttributeIdentifierException.*** Se lanza cuando una operación intenta crear un atributo con un identificador de atributo no válido.
  - ***InvalidAttributesException.*** Se lanza cuando un cliente intenta realizar una operación sobre un conjunto de atributos que no ha especificado correctamente.
  - ***InvalidAttributeValueException.*** Se lanza cuando se intenta añadir un valor no permitido por el esquema a un atributo.
  - ***InvalidSearchControlsException.*** Esta excepción es lanzada cuando a una operación de búsqueda se le pasan controles no válidos.
  - ***InvalidSearchFilterException.*** Se lanza cundo se le pasa a una operación de búsqueda una expresión de filtro de búsqueda no válida.
  - ***NoSuchAttributeException.*** Esta excepción es lanzada cuando una aplicación intenta acceder a un atributo inexistente.
  - ***SchemaViolationException.*** Se lanza cuando algún método atenta de alguna manera contra las reglas del esquema.

4.7.3 El paquete javax.naming.event.

El paquete javax.naming.event contiene clases e interfaces que soportan la ***notificación de eventos*** en servicios de nombres y directorios.

La notificación de eventos supone una herramienta para la monitorización y administración tanto de servicios de nombrado como de directorio ya que permite a las aplicaciones cliente registrar oyentes que permanecen a la escucha del sistema de nombrado o de directorio y reciben una notificación cuando se produce un evento en el sistema que responda al interés del oyente, pudiendo estar éste dirigido a eventos como la modificación de objetos del sistema, lectura de estos objetos, cambio de atributos en el caso de sistemas de directorio etc.

Por tanto el modelo de notificación de eventos se basa en un cliente que permanece a la escucha del sistema de nombrado o de directorio y captura eventos que son lanzados asíncronamente por el servidor cuando se produce una operación que responde al interés del oyente.

En JNDI un evento generado por un sistema de nombrado o de directorio está representado por la clase ***NamingEvent*** que se encuentra en este paquete.

El objeto evento en si está implementado en JNDI por la clase ***NamingEvent*** que contiene un tipo que identifica el tipo de eventos, clasificándolos JNDI en:

- Aquellos que afectan al espacio de nombres (por ejemplo añadir un nuevo objeto).

- Aquellos que no afectan al espacio de nombres (por ejemplo cambiar el contenido de un objeto).

Además el objeto evento también contiene información sobre el tipo de cambio que se ha producido, así como información referente al estado del objeto antes y después de haberse producido el evento. La clase NamingEvent define cuatro tipos de cambios en el sistema:

- ***OBJECT_ADD.*** Se añade un nuevo objeto al sistema.

- ***OBJECT_REMOVED.*** Se elimina un objeto del sistema.

- ***OBJECT_RENAMED.*** Un objeto del sistema es renombrado.

- ***OBJECT_CHANGED.*** Cambia el contenido de un objeto del directorio.

El oyente registrado para recibir notificaciones sobre los eventos suele denominarse ***agente*** y está implementado en JNDI por el interface ***NamingListener.***

Para comenzar recibir a notificaciones de eventos el agente debe estar registrado con un objeto de la clase ***EventContext*** o con un ***EventDirContext*** que representan objetos de sistemas de nombrado o de directorio respectivamente que generan eventos de los cuales permanece a la escucha el agente correspondiente.

El agente una vez registrado recibirá tal como se ha mencionado anteriormente notificaciones sobre eventos cuando éstos se produzcan y respondan al interés del oyente.

4.7.3.1 Contenido del paquete javax.naming.event

A continuación se detallan los interfaces, clases y excepciones que componen el paquete javax.naming.event y se especifica su utilidad.

- ***INTERFACES:***
  - ***EventContext.*** Este interface contiene métodos para registrar o desregistrar oyentes de eventos de un sistema de nombrado.
  - ***EventDirContext.*** Contiene métodos que permiten registrar o desregistrar oyentes de eventos de un sistema de directorio.
  - ***NameSpaceChangeListener.*** Especifica los métodos que un oyente interesado registrar los cambios en un sistema de nombrado debe implementar.
  - ***NamingListener.*** Este interface es la raíz de los interfaces que manejan eventos de nombrado.
  - ***ObjectChangeListener.*** Este interface especifica los métodos que un oyente de eventos de nombrado interesado en registrar los cambios en los objetos debe implementar.

- ***CLASES:***
  - ***NamingEvent.*** Esta clase representa un evento lanzado por un servidor de nombrado o de directorio.

- ***NamingExceptionEvent.*** Representa un evento lanzado cuando los procesos encargados de comunicar a los oyentes los eventos de nombrado lanzan una NamingException.

➢ ***EXCEPCIONES:***
- ***No hay excepciones definidas en el interface.***

4.7.4 El paquete javax.naming.ldap.

El paquete javax.naming.ldap contiene clases e interfaces que implementan características que son específicas del protocolo LDAP en su versión 3 (LDAP v3) y que no están disponibles en el paquete genérico javax.naming.directory ya que se trata de funciones particulares de un servicio de directorio concreto y no se encuentran entre las funciones comunes de los servicios de directorio.

Actualmente el uso de este paquete es poco frecuente por parte de aplicaciones que usan JNDI ya que está orientado principalmente a dar soporte a aplicaciones que necesitar hacer uso de ***"operaciones extendidas", "controles" o "notificaciones ldap no solicitadas".***

El concepto de ***extensión*** aparece con LDAP. Además de tener operaciones claramente definidas como lo pueden ser la búsqueda o la modificación, el protocolo LDAP v3 (RFC 2251) especifica una forma de transmitir entre cliente y servidor ***"operaciones listas para ser definidas".*** Estas operaciones se conocen como ***extensiones*** y ofrecen la posibilidad de mejorar el protocolo, ya que nos proporciona una vía para definir nuevas operaciones además de las que fueron definidas al ser diseñado el protocolo.

En JNDI las extensiones se construyen con clases que implementan los interfaces ***ExtendedRequest*** o ***ExtendedResponse,*** según sea una petición de operación extendida por parte del cliente o una respuesta a una petición de operación extendida por parte del servidor.

Dichas extensiones pueden estar definidas por organizaciones estándares como puede ser ***"Internet Engineering Task Force (IETF)"*** o bien por fabricantes particulares, conociéndose entonces como ***extensiones propietarias.***

El concepto de ***controles*** surge del protocolo LDAP v3 (RFC 2251) que permite que cualquier petición o respuesta sea mejorada por ***"modificaciones listas para ser definidas"*** llamadas controles, es decir, que las operaciones originales del protocolo permiten soportar modificaciones genéricas definidas por el diseñador y que vienen a enriquecer las prestaciones de estas operaciones. Un ejemplo de control, es el control ***sort*** que si es enviado junto con una operación de búsqueda, los resultados de la búsqueda se obtienen ordenados alfabéticamente.

Los controles se clasifican principalmente en:

- ***De petición***: Se envían con la solicitud de la operación que se desea realizar.

- ***De respuesta***: Son enviados por el servidor indicando un cierto tratamiento de la respuesta.

JNDI también establece otra clasificación según afecten o no a la conexión:

- ***Controles de conexión***: Se envían durante el establecimiento de la conexión.

- ***Controles de contexto***: Son los que afectan a los métodos aplicados sobre los distintos conceptos.

La necesidad de esta última clasificación está en que JNDI es un API de alto nivel que no trata directamente con las conexiones, quedando éstas como competencia del proveedor de servicios.

Los controles de petición y respuesta no tienen por qué ir necesariamente emparejados, es decir, que no siempre existe un control de respuesta por cada control de petición y viceversa.

Los controles en JNDI están implementados por clases que implementan el interface ***Control***.

Al igual que las extensiones, los controles pueden estar definidos por organizaciones estándares como puede ser ***"Internet Engineering Task Force (IETF)"*** o bien por fabricantes particulares (***controles propietarios***).

Las ***notificaciones no solicitadas*** son mensajes que se envían desde el servidor al cliente de forma asíncrona y sin responder a ninguna petición del cliente.

En JNDI están soportadas por la clase ***UnsolicitedNotificationEvent*** y por el interface ***UnsolicitedNotificationListener.***

La base de este paquete es el interface ***LdapContext*** que implementa un contexto que permite realizar operaciones extendidas, y trabajar con controles y notificaciones no solicitadas.

4.7.4.1 Contenido del paquete javax.naming.ldap

A continuación se detallan los interfaces, clases y excepciones que componen el paquete java.naming.ldap junto con una descripción de su funcionalidad.

- ***INTERFACES:***
  - ***Control.*** Este interface representa los controles del protocolo LDAP v3 definidos en la RFC 2251.
  - ***ExtendedRequest.*** Este interface representa las peticiones de operaciones extendidas del protocolo LDAP v3 definidas en la RFC 2251.
  - ***ExtendedResponse.*** Representa las respuestas de operaciones extendidas del protocolo LDAP v3 definidas en la RFC 2251.
  - ***HasControls.*** La función de este interface es la devolver controles con objetos devueltos en el formato NamingEnumeration.
  - ***LdapContext.*** Representa un contexto en el cual pueden realizarse operaciones del protocolo LDAP v3 que usen controles o extensiones.
  - ***UnsolicitedNotification.*** Este interface representa una notificación no solicitada tal y como está definida en la RFC 2251
  - ***UnsolicitedNotificationListener.*** Este interface se usa para dar soporte a los eventos de notificaciones no solicitadas.

- ***CLASES:***
  - ***ControlFactory.*** Esta clase abstracta representa una factoría para crear controles.

- ***InitialLdapContext.*** Define el contexto inicial para ejecutar operaciones LDAP v3 del estilo de los controles y las extensiones.
- ***UnsolicitedNotificacionEvent.*** Representa un evento lanzado en respuesta a una notificación no solicitada enviada por un servidor LDAP.

➢ ***EXCEPCIONES:***
- ***LdapReferralException.*** Esta clase abstracta se usa para una excepción provocada por una remisión LDAP.

4.7.5 El paquete javax.naming.spi.

El paquete javax.naming.spi contiene clases que permiten adaptar diferentes proveedores de servicos que acceden a sistemas de nombrado o de directorio y que están diseñados por distintos fabricantes, a JNDI. De esta manera se permite que los distintos servicios sean accesibles desde cualquier aplicación que use JNDI.

Este paquete contiene por tanto unas determinadas clases que ofrecen a las aplicaciones JNDI un interfaz de acceso al proveedor de servicios.

También contiene clases encargadas de devolver objetos que sean naturales e intuitivos para el entorno Java en el cual se encuentra operando la aplicación que usa el paquete, es decir que consigue mapear objetos que se encuentren en un formato que maneje el sistema de nombrado o directorio, o bien objetos Java manejables por las aplicaciones Java y viceversa. Esto se consigue gracias a las ***factorías de estado*** y a las ***factorías de objetos***.

Las factorías de objetos permiten mapear objetos Java en objetos propios del servicio al que se va a acceder y las factorías de estado realizan la operación contraria, convirtiendo objetos propios del sistema servidor a objetos propios de un entorno Java.

Este paquete también proporciona soporte para que distintos proveedores de servicio puedan cooperan entre sí para realizar operaciones en JNDI en caso de tener que manejar nombres que expandan varios sistemas de nombres.

El uso de las clases e interfaces de este paquete es en principio transparente para el desarrollador de aplicaciones, ya que las clases que proporcionan son usadas por clases de otros paquete para llevar a cabo sus funciones, pero no son directamente usadas por el programador salvo que pretenda cambiar las características propias que ofrecen estos objetos.

4.7.5.1 Contenido del paquete javax.naming.spi.

A continuación se detallan los interfaces, clases y excepciones que forman el paquete java.naming.spi., así como la funcionalidad de éstos.

Cabe volver a mencionar que estas clases sirven como interface entre las aplicaciones JNDI y el proveedor de servicio utilizado, y que por tanto las aplicaciones JNDI no tratan directamente con estas clases.

Como cabe esperar la funcionalidad de estas clases se presenta como menos intuitiva desde el punto de vista del desarrollador de aplicaciones JNDI que opera directamente sobre los cuatro paquetes de usuario.

- ***INTERFACES:***
  - ***DirObjectFactory.*** Este interface representa una factoría para crear un objeto dados un objeto de directorio y sus atributos.
  - ***DirStateFactory.*** Representa una factoría para obtener el estado de un objeto y de sus atributos a la hora de construir una unión.
  - ***InitialContextFactory.*** Representa una factoría para la creación de contextos iniciales.
  - ***InitialContextFactoryBuilder.*** Representa un constructor para crear factorías de contextos iniciales.
  - ***ObjectFactory.*** Representa una factoría genérica para crear un objeto.
  - ***ObjectFactoryBuilder.*** Este interface representa un constructor que crea factorías de objetos.
  - ***Resolver.*** Este interface representa un contexto intermedio usado durante la resolución de nombres.
  - ***StateFactory.*** Representa una factoría que permite obtener el estado de un objeto a la hora de realizar una unión.

- ***CLASES:***
  - ***DirectoryManager.*** Esta clase contiene métodos que dan soporte a las implementaciones de contextos iniciales de directorio.

- ***DirStateFactory.Result.*** Implementa un par objeto/atributos que se usa para devolver el resultado de una operación ***DirStateFactory.getStateToBind().***
- ***NamingManager.*** Esta clase contiene métodos para crear contextos y objetos que se encuentran almacenados por referencia en un sistema de nombrado o de directorio.
- ***ResolveResult.*** Esta clase representa el resultado de una operación de resolución de nombres.

➢ ***EXCEPCIONES:***
- ***No hay excepciones definidas en el interface.***

4.8 Ventajas del diseño de JNDI.

A continuación se señalan las principales características relacionadas con el diseño del API JNDI que suponen claras ventajas para el usuario a la hora de comprender JNDI, manejarlo, construir aplicaciones y adaptarlo a sus necesidades.

➢ ***Separación de interfaces en Context y DirContext.*** Los dos interfaces principales en JNDI son Context y DirContext. Context es un interface que proporciona funciones para manejar sistemas de nombrado. DirContext es un interface que extiende el interface Context para proporcionar funciones para manejar sistemas de directorio tales como la manipulación de atributos, gestión de esquemas, etc.

La ventaja de esta separación está en hacer independiente el tratamiento de los servicios de directorios de los servicios de nombrado, consiguiendo así una separación tanto a nivel intuitivo como a nivel del uso de las clases estrictamente necesarias por parte de la aplicación. También es importante señalar que en este aspecto se aprecia el concepto de modularidad con el que ha sido diseñado JNDI.

➢ ***División de JNDI en cuatro paquetes de usuario.*** El término paquetes de usuario designa a los paquetes que contienen clases que son usadas directamente por el usuario para la construcción de

aplicaciones (java.naming, java.naming.directory, java.naming.event, java.naming.ldap). Por tanto queda excluido el paquete java.naming.spi que contiene clases que sirven como interface del proveedor de servicio ya que éstas no son directamente usadas por el usuario. Con esto se consigue aumentar la modularidad y localizar con más precisión los determinados métodos que implementan la operación deseada por el usuario.

- ***Separación de los paquetes de usuario y el interfaz del proveedor de servicios.*** La distinción entre paquetes usados por el usuario y el usado por el proveedor de servicio permite aislar completamente al usuario del funcionamiento del proveedor de servicios.

- ***Múltiples maneras de listar un contexto.*** Existen dos formas de listar un contexto.

  - ***List().*** Devuelve una lista de nombres asociados con los nombres de las clases a las que pertenecen los objetos que designan. Esta filosofía de listar contexto es usada por aplicaciones cuyo objetivo es navegar por el sistema de directorio o nombrado obtenido información contenida en él.
  - ***ListBinding().*** Devuelve una lista de nombres asociados con los nombres de las clases a las que pertenecen los objetos que designan y además los propios objetos. Este método es usado por aplicaciones que pretenden realizar operaciones posteriores sobre los objetos recibidos.

- ***Soporte de federación.*** El concepto de ***federación*** hace referencia a múltiples espacios de nombres o de directorios relacionados entre si, y donde cada espacio tiene una convención de nombrado determinada. El soporte de federación viene proporcionado por el interface ***Name.*** La posterior resolución de nombres que expanden varios espacios de nombres es competencia directa del proveedor de servicios.

- ***Soporte de esquemas.*** El paquete java.naming.directory ofrece métodos que permiten gestionar la información de esquema de un determinado servicio de directorio, permitiendo realizar consultas acerca de clases objeto, sintaxis de atributos, definiciones de atributos y reglas de correspondencia.

Cabe señalar que estos métodos son aplicables siempre y cuando el sistema de directorio que se esté usando publique su información de esquema y permita que se acceda a ella.

- ***Existencia de métodos sobrecargados en los interfaces Context y DirContext.*** Muchos de los métodos de estos interfaces que aceptan como parámetros nombres de entradas del sistema de directorio o de nombrado se encuentran sobrecargados, es decir, que el paquete ofrece dos versiones de estos métodos; Una primera versión nombra los objetos mediante cadenas, de manera que pueden nombrarse objetos del sistema con una notable simplicidad sin más que pasar al método en cuestión una cadena que contenga el nombre del objeto deseado.

  Las versiones sobrecargadas de los métodos aceptan en lugar de una cadena un objeto de una clase que implemente el interface Name. La utilidad de estos métodos radica en el tratamiento de nombres complejos que expandan varios sistemas de nombrado o que se compongan de partes que respondan a distintas convenciones de nombrado. El uso de estos métodos permite el tratamiento de estos nombres de forma transparente para el usuario que no tendrá que preocuparse de manejar la sintaxis de los mismos.

- ***Uso de objetos referenciables y serializables.*** El término referenciable suele designar objetos que por diversas razones tales como el tamaño que ocupan, la dificultad para manejarlos etc, es desaconsejable o no posible su incorporación a un sistema de nombrado o de directorio. Sin embargo el concepto de ***referencia*** nos ofrece una posibilidad para poder manejar dichos objetos consistente en almacenar en el sistema una determinada información a partir de la cual es posible localizar el objeto, en lugar del propio objeto en cuestión. Dicha información se conoce como referencia al objeto y es implementada en JNDI mediante la clase ***Referral.***

  El concepto de objeto ***serializable*** plantea una problemática bastante parecida a las referencias. Nuevamente se dispone de un objeto que por alguna razón no es conveniente o posible su almacenamiento en el sistema de directorio o nombrado pero se tiene necesidad de hacerlo. En caso de que el objeto esté construido como una extensión del interface ***Serializable*** es posible realizar una transformación sobre el objeto para almacenarlo en el sistema; dicha transformación consiste en extraer ciertas propiedades del estado del objeto a partir de las cuales el objeto pueda ser reconstruido. En este caso en lugar

del objeto en sí o información que apunte al objeto, se almacena en el sistema una versión simplificada del objeto basada en una serie de características que permiten reconstruir el objeto a partir de ellas.

- ***Conversión de referencias en objetos.*** En caso de que un objeto almacenado por referencia deba ser devuelto, el proveedor de servicio se encarga de buscarlo e instanciarlo para posteriormente devolverlo al cliente. De esta manera se libera al cliente de tener que realizar operaciones adicionales de localización y conversión de objetos almacenados por referencia.

4.9 Propiedades de entorno en JNDI.

El concepto de contexto en JNDI supone la base sobre la que se construyen todo tipo de operaciones relacionadas con sistemas de nombrado y de directorio. Un contexto es un conjunto de uniones nombre a objeto, es decir que puede contener varios objetos de distintos tipos con un nombre característico asociado que permitirá a la aplicación cliente localizar el objeto dentro del contexto.

Por ello un contexto supone un modelo ideal para implementar entradas de sistemas de nombrado o de directorio, que en esencia son objetos de una determinada clase asociados con un nombre que los identifica dentro del sistema.

En JNDI cada vez que un proveedor de servicios establece una conexión con un sistema de nombrado o de directorio lo hace definiendo un contexto inicial, el cual representa un objeto del sistema servidor que va a servir como base de las operaciones que se desean realizar sobre el sistema.

En el establecimiento de dicha conexión el proveedor de servicio comunica al servidor una serie de parámetros sobre los que se construye la conexión, tales como el tipo de autentificación que va a usarse, la identidad del usuario, su clave, el proveedor de servicios que se va usar, o incluso la URL del sistema al que se desea acceder.

Esta información necesaria para llevar a cabo la conexión son las llamadas ***propiedades de entorno*** ya que definen los parámetros del entorno en el cual se lleva a cabo la comunicación entre el proveedor de servios y el servidor.

JNDI establece la siguiente clasificación de las propiedades de entorno:

- ***Propiedades de entorno estándares de JNDI.*** Están definidas por JNDI y son comunes y aplicables para todos los proveedores de servicio que se usen. Este conjunto de propiedades contienen en su nombre el prefijo "java.naming".
  (Ejemplo: java.naming.security.principal).

- ***Propiedades de entorno específicas del servicio.*** Este conjunto de propiedades son comunes entre todos los proveedores de servicio que implementan un protocolo particular. Los nombres de este conjunto de propiedades contiene el prefijo "java.naming.service" donde "service" es el nombre del protocolo o del servicio implementado. (Ejemplo: java.naming.ldap).

- ***Propiedades de entorno de características específicas.*** Estas propiedades de entorno son comunes para todos los proveedores de servicio que soportan la característica concreta a la cual se refiere la propiedad. Los nombres de este conjunto de propiedades contienen el prefijo "java.naming.feature" donde "feature" es el nombre de la característica. (Ejemplo: java.naming.security.sasl).

- ***Propiedades de entorno específicas del proveedor de servicios.*** Son propiedades de entorno aplicables a un proveedor de servicios concreto como puede ser el proveedor de servicio de Sun contenido en JNDI. Los nombres de estas propiedades deben contener un prefijo que identifiquen a que proveedor de servicios se aplican (ejemplo: com.sun.jndi.ldap).

En JNDI las propiedades de entorno están implementadas por la clase ***java.util.Hashtable*** o alguna de sus subclases.

Una vez definidas por el desarrollador de aplicaciones las propiedades de entorno a usar (normalmente como un array de Hashtable) éstas son pasadas al constructor de la clase Contex o DirContext según a que sistema se desee acceder, de manera que el contexto que representa una entrada del sistema de nombrado o de directorio se define desde su creación con una serie de propiedades de entorno implícitas.

Durante las operaciones que se realicen en el sistema, cualquier contexto derivado del contexto inicial hereda de éste todas sus propiedades de entorno.

No todas las propiedades de entorno son aplicables a todos los contextos. Dado el caso de especificar en un contexto una propiedad de entorno no soportada, ésta será ignorada.

Existe una manera adicional de definir propiedades de entorno además de pasar un objeto Hashtable al constructor de un tipo de contexto, y es mediante los llamados ***"ficheros de recursos".*** Estos ficheros no son más que archivos de texto que contienen una lista de "nombre/valor", donde "nombre" es el nombre de la propiedad de entorno que se quiere inicializar y "valor" es el valor al cual se inicializa la propiedad de entorno.

Hay dos clases de ficheros de recursos:

- ***Ficheros de recursos de aplicación.*** Los ficheros de recursos deben nombrarse como ***jndi.properties***. JNDI pone disponible el fichero de recursos en el "classpath" del sistema, y cuando una aplicación JNDI arranque inspecciona si existe este fichero en el directorio "lib" contenido en el directorio en el que se encuentra instalada la plataforma Java que se esté usando, es decir en el directorio (JAVA_HOME/lib/jndi.properties). Si existe, abre el fichero e incorpora toda la información en él contenida a las propiedades de entorno que utiliza para crear objetos contexto.

- ***Ficheros de recursos del proveedor.*** Cada proveedor de servicio tiene opcionalmente asociado un fichero de recursos que contiene propiedades de entorno aplicables al proveedor de servicio. El nombre de este fichero de recursos es jndiprovider.properties y se encuentra localizado en el paquete que contiene las clases del proveedor de servicio.

A continuación se detallan las propiedades de entorno definidas en JNDI, aunque en la práctica sólo se usa un conjunto no muy amplio de ellas.

- ***Java.naming.applet***. Es un ejemplar de la clase java.applet.Applet. Los parámetros de este ejemplar se usan como propiedades de entorno en el caso de trabajar con aplicaciones que manejen applets.

- ***Java.naming.authoriative.*** Es una cadena que especifica la fuente autoritaria del servicio solicitado. Esta cadena puede tener los valores "true" o "false". Se selecciona la propiedad a "true" cuando se le solicita al proveedor de servicio la fuente de servicio más autoritaria, como podría ser en el caso de un servidor maestro. Si se

selecciona la propiedad a “false” el servidor no tendría por qué ser autoritario, aunque podría serlo.

- ***Java.naming.batchsize.*** Es una representación en formato cadena de un entero que especifica el tamaño preferido del caché a utilizar cuando se devuelven datos mediante el protocolo de servicio.

- ***Java.naming.dns.url.*** Es una cadena que contiene el URL de la máquina que actúa en la red como DNS y el nombre de dominio a usar para la implementación del contexto JNDI.

- ***Javax.naming.factory.control.*** Es una cadena que contiene una lista cuyos elementos están separados por el carácter “:”. Cada elemento de la lista es el nombre de una clase que puede usarse como una factoría de control. Cada clase nombrada en la lista debe implementar el interface ControlFactory. Una factoría de control, es una clase Java usada para implementar controles en LDAP.

- ***Javax.naming.factory.initial.*** Es una cadena que contiene el nombre de la clase de la factoría de contexto inicial. Dicha clase debe implementar el interface InitialContextFactory.

- ***Javax.naming.factory.object.*** Es una cadena que contiene una lista cuyos elementos está separados por el carácter “:”. Cada elemento de la lista es el nombre que una clase que puede usarse como factoría de objetos. Cada clase nombrada debe implementar el interface ObjectFactory o bien el interface DirObjectFactory. Una factoría de objeto es una clase Java que permite construir un tipo de objeto determinado, que puede incluso haber sido definido por el desarrollador de la aplicación.

- ***Java.naming.factory.state.*** Es una cadena que contiene una lista cuyos elementos están separados por el carácter “:”. Cada elemento de la lista es el nombre que una clase que puede usarse como factoría de estado. Cada clase nombrada debe implementar el interface StateFactory o DirStateFactory. Una factoría de estado es una clase Java que permite construir un objeto Java a partir de un objeto devuelto por el sistema de nombrado o de directorio.
- ***Java.naming.factory.url.pkgs.*** Es una cadena que contiene una lista cuyos elementos están separados por el carácter “:”. Cada elemento de la lista es el prefijo del nombre de un paquete que contienen clases que puede usarse como factoría de contexto. Cada clase

nombrada debe implementar el interface ObjectFactory o DirObjectFactory.

Una factoría de contexto es una clase Java que permite construir un objeto que represente un contexto en JNDI para acceder a un sistema de nombrado o de directorio.

- ***Java.naming.language.*** Es una cadena que especifica el idioma preferido para usarlo con este tipo de servicio. Los valores de esta propiedad están definidos por la RFC 1766.

- ***Java.naming.provider.url.*** Es una cadena que contiene una URL que configura el proveedor de servicio especificado por la propiedad java.naming.factory.initial.

- ***Java.naming.referral.*** Es una cadena que especifica cómo se deberían manejar las remisiones por parte del proveedor de servicio. Los posibles valores son “throw”, “ignore” y “follow”, y se detallan en el apartado “Remisiones en JNDI”.

- ***Java.naming.security.authentication.*** Es una cadena que especifica el tipo de autentificación a utilizar. Posibles valores de esta cadena son “none” en el caso de autentificación anónima, “simple” en el caso de usar autentificación simple, es decir, basada en la identidad del usuario y su clave. También son posibles valores los nombres de los métodos específicos usados por el servidor como podrían ser GSSAPI, Kerberos, etc.

- ***Java.naming.security.credentials.*** Es un objeto que especifica las credenciales de la entidad que realiza la autentificación. El tipo de dato del objeto está determinado por el proveedor de servicio y puede ser desde una cadena conteniendo una clave, hasta un certificado de autentificación en formato X.500, según lo exija el método de autentificación que se está usando.

- ***Java.naming.security.principal.*** Es una cadena que especifica la identidad que realiza la autentificación, es decir, el usuario que desea acceder al sistema.

- ***Java.naming.security.protocol.*** Es una cadena que especifica el protocolo de seguridad a utilizar.

Normalmente estas propiedades no suelen definirse usando el nombre de la propiedad sino a través de una serie de constantes que representan la propiedad y que son más cómodas de inicializar. Cada propiedad de entorno tiene asociada una de estas constantes.

La siguiente tabla presenta cada propiedad de entorno asociada con su constante respectiva, y el valor que tienen esas constantes por defecto.

PROPIEDAD DE ENTORNO	CONSTANTE ASOCIADA	VALOR POR DEFECTO
Java.naming.applet	Context.APPLET	Ninguno
Java.naming.authoritative	Context.AUTHORITATIVE	Valor por defecto del proveedor
Java.naming.dns	Context.DNS_URL	Ninguno
Java.naming.factory.control	LdapContext.CONTROL_FACTORIES	Lista vacía
Java.naming.factory.initial	Context.INITIAL_CONTEXT_FACTORY	Ninguno
Java.naming.factroy.object	Context.OBJECT_FACTORIES	Lista vacía
Java.naming.factory.state	Context.STATE_FACTORIES	Lista vacía
Java.naming.factory.url.pkgs	Context.URL_PKG_PREFIXES	Lista vacía
Java.naming.language	Context.LANGUAGE	Valor por defecto del proveedor
Java.naming.provider.url	Context.PROVIDER_URL	Valor por defecto del proveedor
Java.naming.security.authentication	Context.SECURITY_AUTHENTICATION	Valor por defecto del proveedor
Java.naming.security.credentials	Context.SECURITY_CREDENTIALS	Valor por defecto del proveedor
Java.naming.security.principal	Context.SECURITY_PRINCIPAL	Valor por defecto del proveedor
Java.naming.security.protocol	Context.SECURITY_PROTOCOL	Valor por defecto del proveedor
Java.naming.batchsize	Context.BATCHSIZE	Valor por defecto del proveedor
Java.naming.referral	Context.REFERRAL	Valor por defecto del

		proveedor

4.10 Remisiones en JNDI.

Algunos servicios de directorios concretos soportan el concepto de ***remisión*** que es una determinada información que redirige la petición del cliente a otro servidor en el cual es probable que se encuentre la información solicitada por éste.

El concepto de remisión es soportado en JNDI gracias a la clase ***ReferralException.*** Cuando una remisión es encontrada en el servidor, automáticamente se lanza una ReferralException. A partir de este momento el tratamiento de la remisión es competencia del proveedor de servicios, el cual ha sido configurado para tal fin mediante la propiedad de entorno java.naming.referral.

Los posibles valores de dicha propiedad de entorno y el tratamiento de la remisión que especifican son detallados a continuación:

- ***Ignore.*** En este caso el proveedor de servicio ignora la remisión y continua normalmente con la operación.

- ***Follow.*** En este caso el proveedor de servicio directamente sigue la remisión, es decir apunta al servidor especificado en la remisión y continua allí la operación.

- ***Throw.*** En esta situación el servidor captura la excepción y la devuelve a la aplicación JNDI que es la que se encarga de tratarla.

4.11 URLs en JNDI.

Las URLs (Universal Resource Locators) son tratadas en JNDI como nombres compuestos, de manera que las aplicaciones que usan JNDI pueden usar el formato URL para designar objetos en sistemas de nombrado o de directorios.

Un ejemplo de URL en un sistema de directorio LDAP podría ser ***ldap://localhost:389/cn=Jonathan,ou=marketing*** y esta dirección podría ser usada por una aplicación para apuntar a la entrada cn=Jonathan que

cuelga de la entrada ou=marketing en el servicio de directorio LDAP que se encuentra activo en la máquina local y que escucha por el puerto 389.

Además de usar las URL para designar nombres en JNDI pueden usarse para contener direcciones para crear referencias en sistemas de nombres o directorio federados, o bien como datos de configuración.

En el caso de los sistemas federados se tienen una serie de sistemas de nombrado o de directorio unidos entre sí. La utilidad de las URLs aparece cuando queremos añadir una referencia al sistema federado, es decir, cuando queremos que una entrada determinada de un sistema dado apunte a otra entrada que se encuentre en otro sistema del espacio federado. La información sobre cómo acceder a esta última entrada se codifica usando una URL en el formato adecuado que exija la convención de nombrado que usen los sistemas implicados.

En cuanto al uso de URLs como datos de configuración, éstas pueden usarse en directorios LDAP para contener referencias a entradas de directorios. Otro de los usos es el de configurar contextos iniciales a través de la propiedad de entorno Contex.PROVIDER_URL que contiene una URL que especifica el proveedor de servicios.

4.12 Aplicaciones de JNDI.

A continuación se detallan una serie de escenarios para los cuales puede resultar de gran utilidad desarrollar aplicaciones basadas en JNDI que acceden a sistemas de directorio. Con estos casos particulares se pretende mostrar la gran cantidad de campos en los que puede usarse JNDI así como su flexibilidad a la hora de implementar soluciones para los distintos escenarios.

4.12.1 Autentificación de usuarios.

En cualquier sistema seguro se plantea el problema de la autentificación de usuarios ya sea mediante su identificación y clave o mediante sistemas más complejos como podría ser el uso de certificados X.509 en el protocolo SSL.

Una posible aplicación de los sistemas de directorios es mantener la información de autentificación de usuarios en un sistema como bien podría ser una red de ordenadores.

Las entradas del directorio podrían representar los usuarios del sistema y la información referente a estos usuarios (claves, certificados, etc) estaría contenida en sus atributos.

Una aplicación escrita en JNDI podría autentificar a los usuarios realizando búsquedas en el sistema de directorio y comparando esta información con la aportada por dichos usuarios.

4.12.2 Correo electrónico.

Una aplicación muy común de los servicios de directorios es la de mapear nombres de usuarios con sus respectivas direcciones de correo electrónico, de manera que los clientes del sistema de directorio pueden consultar a partir de un nombre la dirección de correo electrónico de ese usuario.

De esta manera el directorio puede ser usado como un libro de direcciones en el cual el usuario registra información referente a personas y sus direcciones de correo electrónico para una posterior consulta.

En este caso el cliente dispondría de una aplicación que mediante JNDI realizara las consultas pertinentes.

4.12.3 Consulta de servicios.

En este caso se propone usar el servicio de directorio para almacenar nombres de servicios ofrecidos en la red con la dirección del equipo en que se encuentran funcionando los servidores y el número de puerto. Incluso podrían añadirse también a los atributos algunas características y prestaciones del servicio concreto.
De esta manera las aplicaciones JNDI clientes podrían consultar al servidor dónde se encuentra la aplicación que ofrece el servicio que el cliente está demandando, incluso podría buscar qué servidores ofrecen una determinada prestación con un determinado grado de servicio y dónde se encuentra dicho servidor.

4.12.4 Funcionamiento como DNS.

El servicio de directorio podría usarse como un DNS (“Domain Name Service”), es decir como un servidor que se encarga de mapear nombres con un cierto grado de familiaridad para el cliente con direcciones IP de equipos que son designados por el nombre anteriormente mencionado.

De esta manera las aplicaciones JNDI pueden acceder al servidor para consultar direcciones IP de equipos de los cuales sólo conocen un nombre representativo y a los cuales se desea acceder.

4.12.5 Servidor de configuración.

Otra aplicación de los servicios de directorio y de las aplicaciones JNDI consiste en la implantación de un servidor que guarde información de configuración de diversos elementos presenten en la red o que eventualmente puedan ser añadidos.

De esta manera una aplicación JNDI podría consultar en el servicio de directorio la configuración de un encaminador (como por ejemplo su tabla de rutas) e inicializarlo; o bien leer información de configuración de una nueva impresora que se añada a la red y configurarla, incluso asignar direcciones IP predefinidas a todos los equipos de una red evitando así tener que configurarlos manualmente.

Capítulo 5: Implementación de LDAP en JNDI.

En este apartado se describen los tipos de datos y las operaciones que ofrece el API JNDI para poder realizar consultas a sistemas de directorio, enfocando la exposición hacia los servicios de directorio basados en el protocolo LDAP.

5.1 Elementos de un servicio de directorio en JNDI.

Los elementos básicos sobre los que se construye todo servicio de directorio son las entradas del árbol de información y los atributos que contienen estas entradas, quedando en un segundo plano frente a éstos la convención de nombrado usada por el directorio, el esquema o las operaciones concretas del servicio de directorio particular tales como extensiones, controles, etc; aspectos que serán detallados en apartados posteriores.

El concepto de ***entrada*** de directorio se implementa en JNDI mediante el interface ***DirContext*** de JNDI, ya que toda entrada LDAP contiene un nombre, un conjunto de atributos y opcionalmente un conjunto de entradas hijos; características que este interface permite implementar con facilidad.

Además este interface proporciona métodos para manejar las entradas, por tanto cada vez que se pretenda manejar una entrada del directorio esta será representada por un objeto que implemente este interface como puede ser la clase ***InitialDircontext.***

El segundo elemento fundamental son los ***atributos.*** En este caso la implementación en JNDI de este concepto es un poco más compleja, ya que hay que hacer uso de dos estructuras.

El conjunto de todos los atributos que contiene una entrada se representa por un objeto que implementa el interface ***Attributes.*** Este tipo de objeto engloba en una sola estructura todo el contenido de la entrada en cuanto a atributos se refiere.

Una vez definido el conjunto de atributos el siguiente paso es definir los atributos particulares dentro de ese conjunto. Para este fin se usan objetos que implementan el interface ***Attribute,*** que ofrece la posibilidad de crear objetos que contienen un nombre que los identifica (identificador de atributo) y un conjunto de valores particulares (valores del atributo).

Los ***nombres*** que designan entradas del directorio son representados en JNDI por elementos del tipo cadena de caracteres (java.lang.String), que contienen un identificador de la entrada de directorio codificado en un formato que sigue la convención de nombrado usada por el servicio de directorio.

En caso de que ocurran problemas durante la realización de operaciones LDAP define un conjunto de códigos de estado que se devuelven con las respuestas enviadas por el servidor (RFC 2251). Estos mensajes se implementan en JNDI mediante excepciones. Este tipo de excepciones son subclases de la clase ***NamingException.*** El proveedor de servicios subyacente se encarga de traducir los códigos de estado que recibe a la subclase adecuada.

5.2 Conexiones LDAP en JNDI.

Cada vez que una aplicación instancia un objeto del tipo "InitialDirContext" el proveedor de servicio subyacente lleva a cabo el proceso de conexión con el servidor LDAP.

La consecuencia directa de este tipo de funcionamiento es la implementación de la operación ***bind*** de LDAP en la que el cliente lleva a cabo el proceso de establecimiento de una conexión con el servidor.

Durante la operación bind el cliente envía al servidor información que permite que este último lleve a cabo su autentificación y un conjunto adicional de opciones que caracterizan la conexión.

Toda esta información se implementa mediante las propiedades de entorno de JNDI. Para ello previamente a la creación del objeto de tipo InitialDirContext se define una estructura del tipo ***Hashtable,*** muy común en Java, en la que se inicializan las propiedades de entorno con los parámetros deseados. A la hora de crear el objeto InitialDirContext se pasa a su constructor la estructura Hashtable que contiene la información.

Estas propiedades de entorno pueden modificarse durante la conexión mediante los métodos ***addToEnvironment()*** y ***removeFromEnvironment()*** que permiten añadir nuevas propiedades y eliminar las existentes respectivamente.

La realización práctica de este proceso puede verse tanto en los ejemplos que se adjuntan en apartados posteriores como en el código desarrollado para el proxy CORBA-LDAP.

La operación ***unbind*** de LDAP se implementa en JNDI mediante el método ***"close()"*** que ofrecen los objetos del tipo InitialDirContext y que permite liberar los recursos usados por un contexto (siempre que estos recursos no estén siendo usados por otros contextos, ya que en este caso estos últimos contextos seguirán ocupando los recursos).

De esta manera puede implementarse también la operación ***abandon*** de LDAP ya que cuando si intenta destruir el contexto que mantenía la conexión, todas las operaciones que se encontraban en estado de ejecución son abortadas.

En el establecimiento de la conexión el cliente elige el método de autentificación que va usar (anónima, simple, SASL) y proporciona información acerca de su identidad y su clave.

El siguiente ejemplo ilustra el procedimiento que debe seguir una aplicación para establecer una conexión con un servidor LDAP y autentificarse correctamente:

```
// En primer lugar se crea la estructura Hashtable que va a contener
// las propiedades de entorno.
Hashtable env = new Hashtable();

// Se establece la factoría de contexto que permite construir objetos
// del directorio.
env.put(Context.INITIAL_CONTEXT_FACTORY,
   "com.sun.jndi.ldap.LdapCtxFactory");

// Se indica la entrada a la que va a representar el contexto
// (normalmente // la raiz de árbol) y la ubicación del servidor.
// Cabe la posibilidad de no indicar ninguna entrada, en cuyo
// caso habrá que usar nombres distinguidos para nombrar
// entradas del directorio en lugar de nombres relativos distinguidos
// referentes a la entrada aquí dada.
env.put(Context.PROVIDER_URL,
        "ldap://localhost:389/o=JNDITutorial");

// Se indica la versión del protocolo LDAP que el cliente desea usar.
// En este caso la versión tres.
env.put("java.naming.ldap.version", "3");


// Se configura el mecanismo de autentificación que va ser utilizado.
// En este caso el mecanismo de autentificación simple.
env.put(Context.SECURITY_AUTHENTICATION, "simple");

// Se configura la identidad del cliente mediante el nombre de la
// entrada que lo representa en el directorio.
env.put(Context.SECURITY_PRINCIPAL, "cn=S. User,
        ou=NewHires, o=JNDITutorial");

// Se establece la clave del cliente, en este caso “mysecret”.
env.put(Context.SECURITY_CREDENTIALS, "mysecret");
```

```
// Se crea el contexto inicial instanciando la clase InitialDirContext
// mediante un constructor que acepta la estructura definida
// anteriormente.
DirContext ctx = new InitialDirContext(env);

// Se realizan las operaciones pertienentes con el contexto

// Se procede a cerrar la conexión
ctx.close();
```

Si el cliente pretende usar ***autentificación anónima*** debe configurar la propiedad de entorno Context.SECURITY_AUTHENTICATION con el valor “none”, en cuyo caso son ignoradas el resto de propiedades referentes a la autentificación.

Esta misma propiedad debe ser configurada con el nombre de un mecanismo SASL (kerberos, GSSAPI, etc) si se pretender usar ***autentificación SASL.***

Por defecto las conexiones LDAP se realizan sobre conexiones TCP/IP. En principio estas conexiones no son seguras ya que los datos se transportan sin ningún tipo de protección frente a intrusos.

Una posibilidad de resolver este problema la proporcionan los protocolos SSL y TLS los cuales permiten realizar un cifrado de la información antes de enviar ésta por la red.

Si se dispone en el entorno de ejecución de clases capaces de implementar sockets SSL y el servidor LDAP se encuentra funcionando en un puerto seguro que use SSL, el proceso de especificación del uso de sockets SSL en JNDI se reduce a modificar dos propiedades de entorno durante la creación del contexto inicial. Estas propiedades son:

```
// Se indica la nueva URL que permite localizar al servidor y a la
// entrada a la que hace referencia el contexto inicial.
env.put(Context.PROVIDER_URL,
        "ldap://localhost:636/o=JNDITutorial");

// Se indica el protocolo de seguridad que va a usarse.
env.put(Context.SECURITY_PROTOCOL, "ssl");
```

Es importante que cliente y servidor compartan la conexión sobre SSL, ya que si se usa SSL para conectarse a un servidor que no usa SSL o al contrario, por propia naturaleza del protocolo SSL la aplicación se colgará.

5.3 Implementación de atributos en JNDI.

Las clases a usar en JNDI para la implementación de atributos son “Attributes” y “Attribute”. Estas clases representan un conjunto de atributos y un objeto atributo respectivamente.

JNDI proporciona dos clases que implementan ese interface que son ***BasicAttributes*** y ***BasicAttribute*** respectivamente.

El uso de este tipo de clases se ilustra mediante el siguiente ejemplo en el que se crean dos atributos del tipo cadena de caracteres. Uno de nombre “moneda” con los valores “euro” y “dólar” y otro de nombre “clima” con el valor “tropical”. A continuación se crea un objeto del tipo Attribute que sirve para contener los dos atributos anteriores.

```
// Se crea un atributo de nombre moneda con un valor euro inicial
BasicAttribute atributo_uno = new BasicAttribute("moneda",
        "euro");

// Se le añade el valor dólar al atributo moneda.
atributo_uno.add("dolar");

// Se crea un atributo de nombre clima con el valor tropical.
BasicAttribute atributo_dos = new BasicAttribute("clima",
        “tropical”);

// Se crea el contenedor de atributos.
// “true” indica que las mayúsculas son ignoradas
BasicAttributes attrs = new BasicAttributes(true);

// Se añaden los dos atributos a la estructura.
attrs.put(atributo_uno);
attrs.put(atributo_dos);
```

Es importante pasar el valor "true" al constructor de la clase BasicAttributes para poder ignorar las mayúsculas en los identificadores de atributos, ya que el protocolo LDAP no hace distinciones entre mayúsculas y minúsculas en los identificadores de atributos.

JNDI permite definir atributos que deben ser transmitidos en formato binario. Para ello se crean usando la opción ***";binary"*** tras el identificador del atributo.

5.4 URLs en JNDI.

El uso de URLs con formato LDAP está presente en múltiples ámbitos de JNDI. Las URLs se codifican en cadenas de caracteres que son usadas por los distintos métodos que manejan URLs para llevar a cabo operaciones LDAP.

Los escenarios en los que se usan URLs son:

- ***Uso de URL como un nombre para el contexto inicial.*** En este caso se pasa una URL que codifique la entrada deseada en lugar del nombre distinguido relativo de la entrada al contexto con el que se inició la comunicación.

- ***Uso de URLs para especificar componentes de consulta.*** Este caso sólo se aplica a operaciones de búsqueda, ya que en ningún otro caso la URL deberá contener componentes de consulta. Cuando se pasa una URL de este tipo a un método de búsqueda el resto de los argumentos son ignorados.

- ***Configurar proveedores de servicio.*** Para configurar un proveedor de servicio se suministra una URL que éste usará para contactar con el servidor LDAP.

- ***Especificar remisiones.*** Una remisión se codifica como una cadena que contiene una URL LDAP que indica la ubicación del objeto al cual remite al cliente.

- ***Uso de URLs en otras operaciones.*** El resto de operaciones en las que usan URLs LDAP están relacionadas con la federación de LDAP bajo otro espacio de nombres, contextos no relativos al actual, etc.

5.5 Remisiones en JNDI.

El proveedor de servicios es el encargado de manejar las remisiones que pudieran aparecer en las operaciones de directorio.

El tratamiento de las remisiones se configura mediante la propiedad de entorno ***"Context.REFERRAL"*** (java.naming.referral) que recoge las siguientes posibilidades:

- Ignorar las remisiones (valor ignore).
- Seguir las remisiones (valor follow).
- Lanzar una excepción cada vez que se encuentre una remisión (valor throw).

El manejo de remisiones por parte del servidor LDAP se realiza gracias al control ***"Manage Referral"*** que devuelve las entradas resultantes de seguir una remisión como entradas ordinarias.

Si el servidor no soporta este control devolverá todas las remisiones que encuentre, siendo el proveedor de servicios el encargado de informar al usuario mediante una excepción de que si se siguiera la remisión podrían encontrarse más resultado.

```
// La siguiente orden hace que las remisiones sean siempre seguidas.
// env es el nombre del objeto Hashtable usado.
env.put("java.naming.referrla", "follow");
```

Para crear una remisión hay que añadir el valor ***"referral"*** al atributo "objectclass" de la entrada a la cual se desea añadir la remisión. Una vez que la entrada está preparada para contener una remisión simplemente hay que añadir un atributo de nombre ***"ref"*** cuyo valor sea una URL LDAP que identifique a la entrada hacia la que se desea apuntar.

La gestión de la remisión se realiza igual que cualquier otro atributo, es decir, puede borrarse, actualizarse, añadirse nuevos valores, etc.

5.6 Alias en JNDI.

Un alias en LDAP consiste en una entrada que apunta a otra entrada del mismo directorio.

El protocolo LDAP permite las siguientes políticas de tratamiento de alias:

- Ignorarlos.
- Seguir todos los alias.
- Seguir sólo los alias de la entrada base de la búsqueda.
- Seguir todos los alias excepto el de la entrada base de la búsqueda.

El tratamiento de los alias en JNDI se indica mediante la propiedad de entorno "***java.naming.ldap.derefAliases".***

Esta propiedad puede configurarse con los valores ***"never"*** (por defecto), "***always", "finding***" o ***"searching",*** correspondiendo por este orden con los casos anteriormente enumerados.

```
// La siguiente línea hace que cualquier alias que se encuentre sea
// desreferenciado
env.put("java.naming.ldap.derefAliases", "always");
```

Los alias, al contrario que las remisiones son ignorados por aquellas operaciones que actualizan el directorio.

La creación de un alias en una entrada del directorio se lleva a cabo añadiendo a su atributo "objetclass" el valor ***"alias".*** Desde ese momento la entrada se encuentra preparada para contener los alias oportunos. Esto se realiza mediante el atributo de nombre ***"aliasedObjectName"*** cuyo valor será el nombre distinguido de la entrada a la que se apunta.

5.7 Recuperar todos los atributos de una entrada.

El método más usado en JNDI para obtener el conjunto de atributos que contiene una entrada es ***getAttributes().***

La forma más general de este método es:
getAttributes(Name name, String[] retAttrs)

El método recibe un objeto que identidica la entrada a devolver y una tabla que contiene los identificadores de atributos que el servidor ha de devolver (si en lugar de este objeto se le pasa el carácter especial "*" todos los atributos son devueltos).

Este método realiza la misma operación que "search" usando un filtro el filtro de búsqueda "objetclass=*", en el ámbito de búsqueda de la propia entrada y solicitando todos los atributos.

Existen otros métodos tales como ***lookup(), lookupLink()*** que devuelven directamente el objeto asociado con un nombre.

También son usados ***list() y listBindings()*** que intentan crear un objeto para cada elemento devuelto en el resultado.

5.8 Implementación de la operación MODIFY_RDN en JNDI.

El método JNDI que implementa esta operación es ***rename()*** y se encuentra disponible en el interface Context.

Para ello este método recibe una cadena que contiene el nombre distinguido de la entrada que se desea renombrar y otra cadena que contiene el nuevo nombre distinguido de la entrada.

Debe señalarse que dependiendo de este nuevo nombre distinguido la entrada puede sufrir cambios de contexto, es decir, puede ser trasladada a otra zona del árbol de directorio.

```
// La siguiente orden cambia el nombre de la entrada “OldHires” por
// “NewHires” de manera que se mantiene el contexto.
ctx.rename("ou=NewHires,o=JNDITutorial","ou=OldHires,o=JNDI
        Tutorial");

// A continuación se muestra un cambio de nombre que implica un
// cambio de contexto.
ctx.rename("cn=C. User, ou=NewHires", "cn=C. User, ou=People");
```

El protocolo LDAP ofrece la posibilidad de conservar el antiguo nombre de la entrada como un atributo. Para indicar si se desea hacer uso de esta posibilidad se configura la siguiente propiedad de entorno:

```
// La siguiente orden configura que el nombre antiguo sea
// almacenado.
env.put("java.naming.ldap.deleteRDN", "false");
```

5.9 Implementación de la operación SEARCH.

El interface DirContext proporciona los siguientes métodos de búsqueda que implementan la operación “search” de JNDI.

- **search(Name name, Attributes matchingAttrs).**
- **search(Name name, Attributes matchingAttrs, String[] retAttrs)**
- **search(Name name, String filter, SearchControls ctls)**
- **search(Name name, String filterExpr, Object[] filterArgs, SearchControls ctls)**

Cada uno de estos métodos tiene una versión sobrecargada que acepta un nombre del tipo java.lang.String en lugar de un objeto de clase Name.

Los dos primeros métodos son equivalentes si en lugar del un objeto de la clase String[] se pasa el argumento null.

El significado de los parámetros se expone a continuación:

- ***Name name.*** Contiene el identificador de la entrada que servirá de base para la búsqueda.

- ***Attributes matchingAttrs.*** Contiene el conjunto de condiciones del tipo “atributo-igual-valor” que deben cumplir las entradas que se buscan.

- ***String[] retAttrs.*** Es una lista con los identificadores de los atributos que el cliente solicita que se devuelvan. Si contiene el carácter especial “*” todos los atributos son devueltos.

- ***String filter.*** Es una cadena de caracteres que contiene un filtro de búsqueda.

- ***SearchControls ctls.*** Este objeto contiene los controles que se usan en la búsqueda, tales como el número máximo de entradas a devolver, el tiempo máximo de espera, etc.

- ***Object[] filterArgs.*** Contiene parámetros que reemplazarán a determinados alias que los representan en la invocación de la operación. De esta manera se consigue ahorrar en la escritura de aplicaciones.

- ***StringFilterExpr.*** Contiene un alias que será reemplazado por un parámetro del campo “filterArgs”. La enésima ocurrencia de este alias será reemplazada por el enésimo elemento de “filterArgs”.

El siguiente ejemplo ilustra una operación de búsqueda en el que usa un filtro.

```
// Se definen los controles que serán usados en la búsqueda.
SearchControls ctls = new SearchControls();

// Se construye el filtro de búsqueda en el que se exige que las
// entradas a devolver contengan un atributo “sn” con un valor
// “Geisel” y que contengan un atributo de nombre “mail” con
// cualquier valor.
String filter = "(&(sn=Geisel)(mail=*))";

// Se realiza la operación de búsqueda
NamingEnumeration answer = ctx.search("ou=People", filter, ctls);
```

Hay que señalar que si el cliente desea consultar un atributo operacional (guardado por el servidor para propósitos administrativos) debe solicitarlo explícitamente en los atributos que deben ser devueltos.

Otra posibilidad que tiene el cliente, es la de recibir los atributos con sólo sus identificadores, es decir, sin valores. Para ello debe configurarse la siguiente propiedad de entorno:

```
// env es el nombre del objeto Hashtable. Esta orden indica que sólo
// debe devolverse el identificador de atributo y no los valores.
env.put("java.naming.ldap.typesOnly", "true");
```

5.9.1 Controles de búsqueda en JNDI.

Los controles de búsqueda son parámetros que definen determinados comportamientos de la operación de búsqueda tales como

- ***Los atributos a devolver.*** Se define una tabla cuyos elementos son los identificadores de los atributos que se desean recibir y se incorporan a un objeto del tipo “SearchControl”.

  El siguiente ejemplo define un control que solicita los atributos “sn” y “telephonenumber” de las entradas que se encuentren en una búsqueda que use este control.

```
// Se define una tabla con los identificadores de atributos a
// devolver.
String[] attrIDs = {"sn", "telephonenumber"};

// Se instancia un objeto para contener los controles de
// búsqueda.
SearchControls ctls = new SearchControls();

// Se define el control de devolución de atributos (con la tabla
// de identificadores) dentro del objeto creado.
ctls.setReturningAttributes(attrIDs);
```

- ***El ámbito de la búsqueda.*** Se define la profundidad de la búsqueda a partir del objeto base.
  El ámbito puede ser:

- ***OBJECT_SCOPE.*** Busca en el propio objeto nombrado.

- ***SUBTREE_SCOPE.*** Realiza la búsqueda en todo el subárbol que cuelga de la entrada base.

- ***ONE_LEVEL_SCOPE.*** La búsqueda se realiza sólo en los hijos de la entrada base.

El siguiente ejemplo muestra la definición de un control que permita realizar la búsqueda en todo el subárbol que cuelga de la entrada.

```
// El objeto ctls es del tipo SearchControl y se supone ya
// definido.
ctls.setSearchScope(SearchControls.SUBTREE_SCOPE);
```

- ***El número máximo de entradas a devolver.*** Permite limitar el número de entradas que el servidor puede enviar al cliente. Se realiza mediante el control ***"setCountLimit".*** El valor cero desactiva este control. El siguiente ejemplo establece este límite en cinco entradas.

```
// ctls es un objeto del tipo SearchControl previamente
// definido.
ctls.setCountLimit(5);
```

- ***El número máximo de milisegundos a esperar.*** Permite definir el tiempo máximo que puede emplearse en una operación de búsqueda. Se realiza mediante el control ***"setTimeLimit()".*** El valor cero desactiva este control. El siguiente ejemplo establece el límite en un segundo.

```
// ctls es un objeto del tipo SearchControl previamente
// definido.
ctls.setTimeLimit(1000);
```

- ***Si se devuelve o no el objeto java asociado con la entrada.***

- ***Si los enlaces JNDI se desreferencian durante la búsqueda.***

5.9.2 Resultados de la búsqueda.

Cuando se usan métodos de búsqueda del interface DirContext teles como "search", se obtiene como resultado un objeto del tipo "***NamingEnumeration".***

Un objeto del tipo NamingEnumeration está compuesto por la unión de un conjunto de objetos del tipo ***"SearchResult".***

Un objeto del tipo SearchResult contiene la siguiente información:

- El nombre, en relación al contexto fuente con respecto al cual se resuelven los nombres, de la entrada LDAP a la que representa. En caso de que durante la búsqueda se siga alguna referencia o algún alias es posible que este campo contenga una URL LDAP que identifique a una entrada no relativa al contexto del que partió la búsqueda. Para leer este campo se usa el método ***getName()*** sobre el objeto SearchResult.

- Un objeto que representa a la propia entrada, construido a partir de los atributos de la entrada. Este objeto se encuentra presente si se solicitó mediante el control de búsqueda adecuado. El objeto puede recuperarse mediante el método ***getObject().***

- El nombre de la clase del objeto devuelto, o el nombre de la clase objeto a la que pertenece la entrada. Este nombre puede recuperarse con ***getClassName().***

- El conjunto de atributos de la entrada. Pueden recuperarse mediante ***getAttributes()*** sobre el objeto SearchResult.

- Controles de respuesta.

5.10 La operación COMPARE.

Esta operación se implementa en JNDI mediante el mismo método que se usa para realizar búsquedas, es decir, “search”.

La utilidad de esta operación está sobre todo en consultar atributos, tales como claves de acceso, no expuestos para la operación de búsqueda general, pero sí para usuarios particulares.

Para implementar correctamente esta operación debe restringirse adecuadamente los parámetros de “search”.

El filtro debe ser de la forma “nombre=valor”, no se permite usar comodines, el ámbito de la búsqueda debe ser la propia entrada y el servidor no debe devolver atributos.

Si la operación tiene éxito el resultado es un objeto cuyo nombre es una cadena vacía y que no contiene atributos.

5.11 La operación MODIFY.

Esta operación permite modificar los atributos de una entrada y los valores de estos atributos.

El interface Context implementa esta operación mediante el método ***modifyAttributes().*** Este método recibe el nombre de la entrada cuyo contenido se desea modificar y una tabla de objetos de la clase ***ModificationItem*** que contiene una lista de solicitudes de modificación.

Cada “ModificationItem” consiste en una constante numérica que identifica el tipo de modificación que se desea realizar, y un objeto del tipo “Attribute” que define la modificación.

Las modificaciones permitidas son:

- ***ADD_ATRIBUTE***. Permite añadir un nuevo atributo a la entrada o un nuevo valor particular a un atributo ya existente.

- ***REPLACE_ATTRIBUTE.*** Permite sustituir todos los valores particulares de un atributo por un único valor, o eliminar dicho

atributo si no se pasa un nuevo valor que sustituya a todos los que contiene en el momento de la modificación.

- ***REMOVE_ATTRIBUTE***. Permite eliminar un atributo o un valor particular de dicho atributo.

Las modificaciones se ejecutan en el mismo orden en el que aparecen en la lista, y se ejecutan todas o ninguna.

El siguiente ejemplo crea una lista de modificaciones en la cual se reemplaza el valor del atributo “mail”, se añade un valor adicional al atributo “telephonenumber” y se elimina el atributo “jpegphoto”.

```
// Se define un objeto del tipo array de ModificationItem que
// contendrá las modificaciones que se desean realizar.
ModificationItem[] mods = new ModificationItem[3];


// En el primer elemento de la tabla se guarda un objeto del tipo
// ModificationItem construido con la operación de reemplazar y con
// un objeto del tipo BasicAttribute que representa al atributo “mail”
// y al valor geisel@wizards.com.
mods[0]=new
   ModificationItem(DirContext.REPLACE_ATTRIBUTE,
   new BasicAttribute("mail", "geisel@wizards.com"));


// El segundo elemento se define de la misma manera que el primero.
// Ahora la modificación es añadir un atributo, y dicho atributo
// se llama “telephonenumber” y tiene el valor “+1 555 555 5555”
mods[1] = new ModificationItem(DirContext.ADD_ATTRIBUTE,
   new BasicAttribute("telephonenumber", "+1 555 555 5555"));


// El último elemento pretende borrar el atributo “jpegphoto”
// con todos sus valores.
mods[2]=new
ModificationItem(DirContext.REMOVE_ATTRIBUTE,
   new BasicAttribute("jpegphoto"));
```

```
// Por último se realizan las modificaciones sobre el contexto
// deseado.
ctx.modifyAttributes(name, mods);
```

5.12 La operación ADD.

Esta operación permite añadir una nueva entrada al directorio inicializada con un conjunto de atributos y valores.

En JNDI la creación de una nueva entrada se interpreta como la creación de un nuevo contexto, para lo cual se usa el método ***"createSubcontext()"*** que aplicado sobre un contexto crea un nuevo contexto que cuelga de éste.

El siguiente ejemplo muestra el uso de este método añadiendo una entrada de nombre contenido en el parámetro "nuevo_nodo" y con un conjunto de atributos que vienen dados por attrs.

```
// Se supone que los objetos "nuevo_nodo"y "attrs" han sido
// previamente definidos.
contexto.createSubcontext(nuevo_nodo,attrs);
```

5.13 La operación DELETE.

Esta operación permite eliminar una entrada del árbol de directorio.

En JNDI esta operación se interpreta como la destrucción de un contexto, y ofrece el método ***"destroySubcontext()"*** para llevarla a cabo.

El siguiente ejemplo muestra el uso de este método eliminando una entrada que viene dada por la cadena "borrar_nodo".

```
// La cadena "borrar_nodo" contiene el nombre distinguido de la
// entrada a eliminar.
contexto.destroySubcontext(borrar_nodo);
```

5.14 Esquemas LDAP en JNDI.

El API JNDI contiene métodos que acceden a la información del esquema que mantiene el servidor LDAP acerca de las clases objeto, los tipos de atributos, las sintaxis de éstos y las reglas de correspondencia utilizadas.

Los métodos de JNDI que implementan operaciones de esquema son:

- ***DirContext.getSchema().*** Devuelve el árbol de esquema del objeto nombrado, es decir la raíz del árbol de esquema.

- ***DirContext.getSchemaClassDefinition().*** Devuelve información de esquema acerca de la clase objeto nombrada.

- ***Attribute.getAttributeDefinition()***. Devuelve la definición de atributo para ese atributo.

- ***Attribute.getAttributeSyntaxDefinition().*** Devuelve la definición de la sintaxis del atributo sobre el que se aplique.

Las operaciones anteriormente señaladas se usan para obtener información de esquema sobre elementos concretos del árbol de directorio.

El método ***"lookup"*** es el más usado en la práctica, y permite obtener la información deseada directamente del árbol de esquema y de manera general, es decir, sin estar referida a ninguna entrada del árbol de directorio.

Como se ha explicado en apartados anteriores, el árbol de directorio se halla compuesto por cuatro subárboles, los cuales contienen entradas que guardan información acerca de clases objeto, tipos de atributo, sintaxis de atributos y reglas de correspondencia respectivamente. Una vez que se consiga apuntar a la raíz del árbol de esquema, el método "lookup" permite consultar entradas pertenecientes a dicho árbol si son correctamente nombradas.

En el siguiente código se ilustra un acceso al árbol de directorio y la posterior consulta de entradas de dicho árbol.

```
// El primer paso consiste en la obtención de un contexto que
// represente la raíz del árbol de esquema.
DirContext schema = ctx.getSchema("");

// Una vez obtenida la raíz se solicita la entrada que contiene la
// definición de la clase objeto deseada (en este caso persona) y se
// obliga a que el resultado sea del tipo DirContext.
DirContext personSchema =
        (DirContext)schema.lookup("ClassDefinition/persona");

// Podría solicitarse una entrada que contenga información sobre
// un tipo de atributo (en este caso cn)
DirContext cnSchema =
        (DirContext)schema.lookup("AttributeDefinition/cn");

// Si se quisiera solicitar una sintaxis de atributo (en este caso una
// cadena de caracteres) ha de pasarse el identificador de objeto de
// dicha sintaxis de atributo al método.
DirContext dsSchema =
        (DirContext)schema.lookup("SyntaxDefinition/1.3.6.1.4.1.146
        6.115.121.1.15");

// También es posible obtener información acerca de reglas de
// correspondencia como puede ser la regla de igualdad que
// ignora las mayúsculas.
DirContext mrSchema =
        (DirContext)schema.lookup("MatchingRule/caseExactOrderin
        gMatch-en");
```

JNDI proporciona descripciones generales sobre cómo se deberían comportar estos métodos, (pero no da muchos detalles acerca de ellos) tales como la estructura de los objetos devueltos, el efecto de las modificaciones en el árbol de esquema, etc. Estas características realmente vienen determinadas por el proveedor de servicio que se esté usando y el servidor LDAP.

El tratamiento que puede realizarse del árbol de esquema es análogo al que se realiza con el árbol de información del directorio, de manera que pueden realizarse sobre él operaciones de búsqueda, y operaciones de actualización de la entradas (añadirlas, borrarlas, etc). El comportamiento de estas operaciones vendrá dado por el proveedor de servicio y por el propio servidor que se esté usando, de manera que las entradas añadidas de esta

manera pueden ser ignoradas, no mantenidas cuando se apaga el servidor, rechazadas, etc.

5.15 Controles en JNDI.

Los controles y extensiones están soportados por las clases e interfaces del paquete javax.naming.ldap.

Los controles ya sean de petición o de respuesta están implementados por el interface ***"Control".*** Normalmente el desarrollador de aplicaciones trata con clases que implementan el interface y que contienen constructores amigables que facilitan su utilización.

El uso depende del tipo de control que normalmente viene implementado por un fabricante particular.

Los controles pueden clasificarse en controles de petición y controles de respuesta. Un control de petición se envía desde el cliente al servidor junto con una operación LDAP, y un control de respuesta se envía desde el servidor al cliente junto con una respuesta LDAP. A su vez los controles de petición se dividen en aquellos que afectan a cómo se crea la conexión y aquellos que afectan a los métodos de contexto. El primer grupo de controles se usa cuando se necesita establecer o restablecer una conexión con un servidor LDAP.

El segundo grupo se usa cuando se envía cualquier otro tipo de petición de información al servidor.

En el caso de controles que afecten a métodos de contexto, una vez creados los objetos que implementan las clases que definen los controles, éstos pueden ser asociados con las peticiones LDAP enviadas por los métodos de los interfaces Context y DirContext usando el método ***LdapContext.setRequestControls().***

Una vez configurados, los controles permanecen activos para el ejemplar de Context hasta que son reemplazados por los argumentos de otra llamada a "setRequestControls".

El método ***getRequestControls()*** permite encontrar los controles de petición que están activos.

Para el caso de controles de peticiones de contexto, éstos son usados siempre que se necesita establecer o restablecer una conexión, pero no afectan a otras operaciones tales como “search” o “modify”. Estos controles se especifican durante la creación de un objeto del tipo ***InitialLdapContext*** que es una clase que permite definir objetos que implementan entradas de directorio capaces de soportar extensiones y controles.

Si se desea cambiar algún control de petición de conexión se usa el método ***LdapContext.reconnect()*** que establece una nueva conexión con el servidor usando los nuevos controles. Esta conexión sólo afecta a la conexión que está siendo usada por el objeto sobre el que se llamó al método. Cualquier objeto que descienda de éste heredará los controles, pero los contextos que anteriormente compartían la conexión no se verán afectados.

El método ***getConnectControls()*** permite consultar los controles de petición que se encuentran activos para un contexto.

En cuanto a los controles de respuesta, el servidor puede enviarlos con cualquier respuesta sin necesidad de responder a ningún control de petición. De esta manera el cliente debería chequear todas respuestas que recibe en busca de controles de respuesta, aunque en la práctica esta operación que se realiza mediante el método ***getResponseControls()*** sólo se realiza cuando se espera algún control de este tipo por parte del cliente.

5.16 Operaciones extendidas en JNDI.

El protocolo LDAP define operaciones extendidas, las cuales toman una petición como argumento y devuelven una respuesta. La petición contiene un identificador que identifica la propia petición y sus argumentos. La respuesta contiene los resultados de realizar la petición.

La pareja de operaciones petición/respuesta extendidas se conoce como extensión.

El paquete javax.naming.ldap define el interface ***“ExtendedRequest”*** para representar el argumento de una operación extendida y el interface ***“ExtendedResponse”*** para representar el resultado de una operación extendida.

Normalmente una respuesta extendida está emparejada con una petición extendida pero no necesariamente al revés, es decir que cabe no esperar respuesta de toda petición extendida que se realice.

Una respuesta extendida sin pareja se llama ***notificación no solicitada.*** Para recibir este tipo de notificación es necesario tener registrado un oyente que capture los eventos que ocurren en el servidor. Para poder llevar a cabo esta operación es necesario que el servidor soporte un control llamado "control de búsqueda persistente"

Las aplicaciones usan normalmente clases que implementan estos interfaces, obtenidas a través de entidades de estandarización o de vendedores particulares.

Capítulo 6: El estándar CORBA.

En este apartado se describe el funcionamiento de un sistema CORBA a través de su arquitectura, funcionamiento, proceso de diseño y uso del lenguaje IDL.

6.1 Introducción a CORBA.

Hace no demasiados años los sistemas informáticos estaban restringidos a las grandes corporaciones que eran las únicas capaces de afrontar el elevado costes que éstos suponían. Estos sistemas informáticos eran centralizados, consistiendo en una máquina principal (mainframe) en la que se ejecutaban programas a los que los usuarios podían acceder desde un terminal conectado a dicha máquina.

La evolución de los sistemas informáticos lleva a la aparición de equipos cada vez más pequeños y baratos lo cual hace que empresas de menor envergadura comiencen a informatizarse aumentando así la popularidad de estos sistemas.

De esta manera surge la necesidad de interconectar equipos buscando el máximo aprovechamiento de la tecnología, lo cual conlleva la aparición de aplicaciones capaces de ejecutar parte de su código en distintas máquinas simulando para el usuario que la ejecución se realiza localmente.

La aparición posterior del paradigma de la programación orientada a objetos hace que aparezca el grupo OMG (Object Management Group), formado por un conjunto de empresas, y dedicado a crear una norma para el desarrollo de aplicaciones distribuidas orientadas a objetos.

Esta norma llamada ***CORBA*** ha ido evolucionando desde entonces con nuevos elementos y versiones mejoradas y se ha ido extendiendo por ámbitos académicos y empresariales hasta el punto de que hoy día es una de las tecnologías con más proyección de futuro dentro de la informática.

6.2 El Grupo de Gestión de Objetos (OMG).

El grupo de gestión de objetos se forma con la misión de crear un mercado de programación basada en componentes impulsando así la introducción de objetos de programación estandarizados.

Su propósito principal es desarrollar una arquitectura única, usando la tecnología de objetos, para la integración de aplicaciones distribuidas, garantizando la reusabilidad de los componentes, la interoperabilidad y la portabilidad.

Este fin responde tanto al aprovechamiento de los beneficios que ofrece la tecnología de objetos, y también en gran parte a evitar las dificultades que han tenido los productos de otros organismos de normalización como ISO para encontrar una adecuada respuesta en el mercado.

Mas allá de generar normas busca que las normas propuestas sean aceptadas y utilizadas ampliamente.

6.3 La norma CORBA (Common Object Request Broker).

CORBA es una especificación normativa que resulta de un consenso entre los miembros del OMG, un consorcio que agrupa hoy día a más de setecientas empresas tanto de la industria informática como de consumidores de la misma. Esta norma cubre cinco grandes ámbitos que constituyen los sistemas de objetos distribuidos.

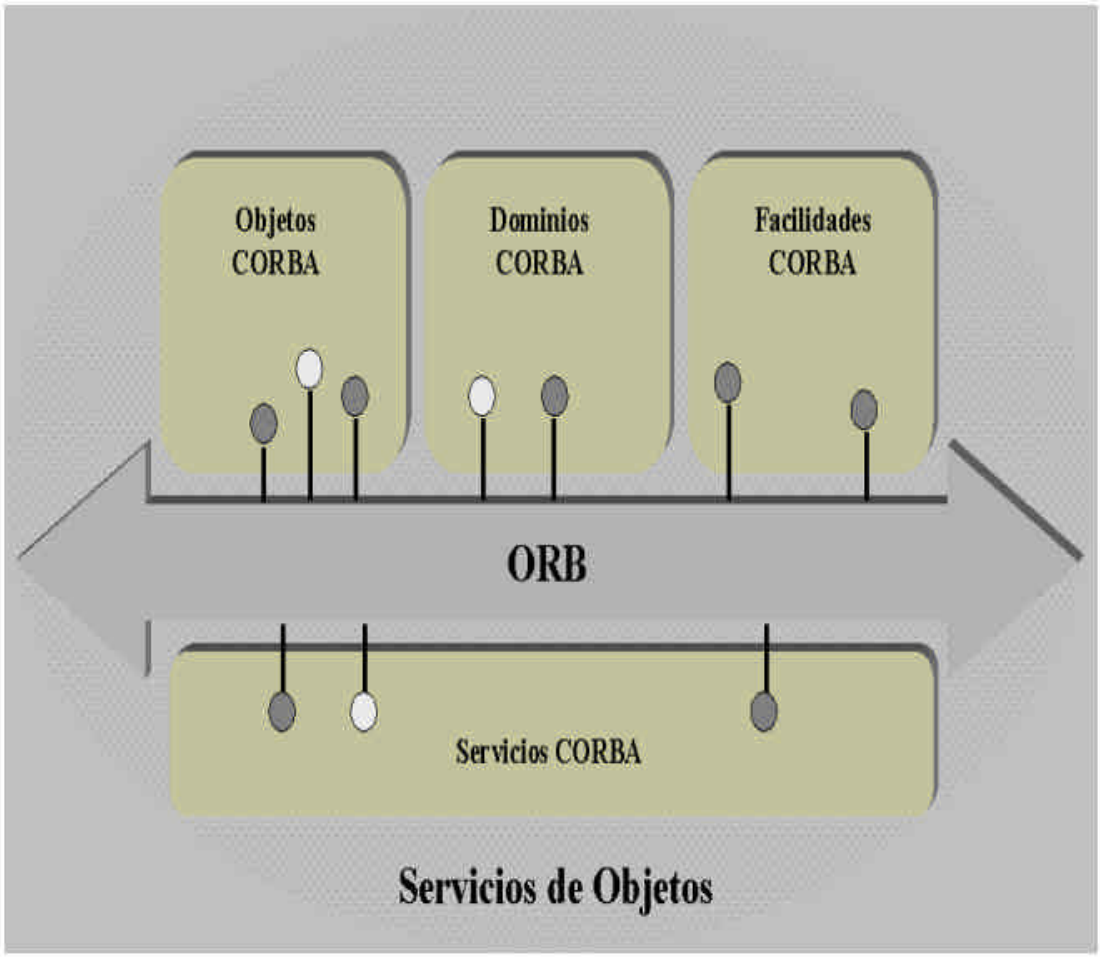
- Un lenguaje de descripción de interfaces llamado “Lenguaje de Definición de Interfaces” (IDL), traducciones de este lenguaje a

lenguajes de implementación como pueden ser C++, Ada, Java, etc; y una infraestructura de distribución de objetos llamada “bus de objetos distribuidos” (ORB. Object Request Broker).

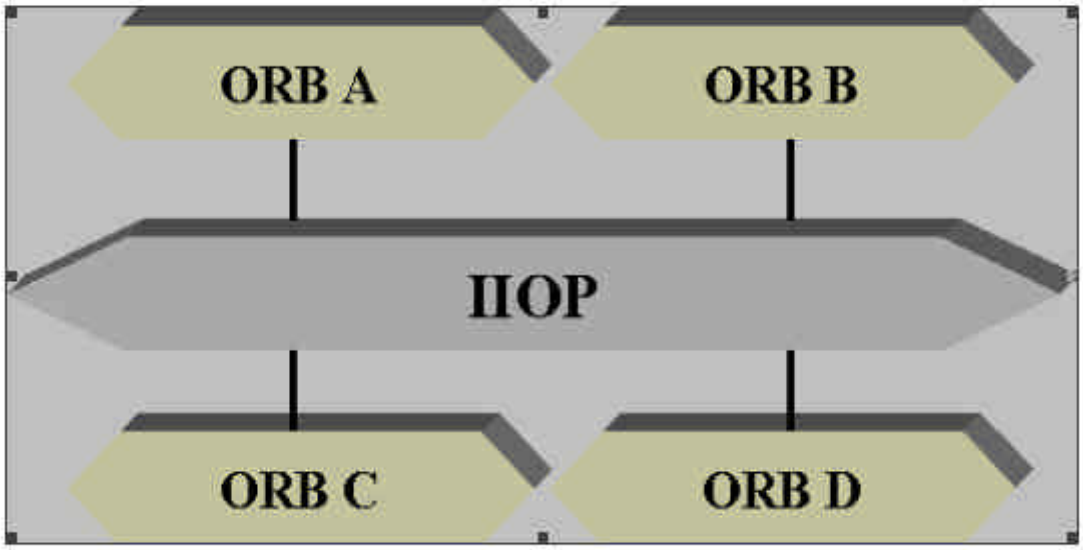
- Una descripción de servicios conocidos con el nombre de Servicios CORBA que complementan el funcionamiento básico de los objetos que dan lugar a una aplicación. Estas especificaciones cubren los servicios de nombrado, de persistencia, de eventos de transacciones, etc. El número de servicios se amplía continuamente para añadir nuevas capacidades a los sistemas desarrollados con CORBA.

- Una descripción de servicios orientados al desarrollo de aplicaciones finales estructurados sobre los objetos y servicios CORBA. Estas aplicaciones se designan con el nombre de Facilidades Comunes y cubren servicios de alto nivel tales como los interfaces de usuarios, los documentos compuestos, la administración de sistemas y redes, etc.

- Una descripción de servicios verticales denominados Interfaces de Dominio que proveen funcionalidad de interés para usuarios finales en campos de aplicación particulares tales como telecomunicaciones, finanzas, medicina, etc.



- Un protocolo genérico de intercomunicación, el Protocolo General Inter-ORB GIOP (General Inter-ORB Protocol), que define los mensajes y el empaquetado de los datos que se transmiten entre los objetos. Además define implementaciones, de ese protocolo genérico, sobre diferentes protocolos de transporte, lo que permite la comunicación entre los diferentes ORBs consiguiendo la interoperabilidad de elementos de diferentes vendedores. Por ejemplo el IIOP para redes con la capa de transporte TCP.

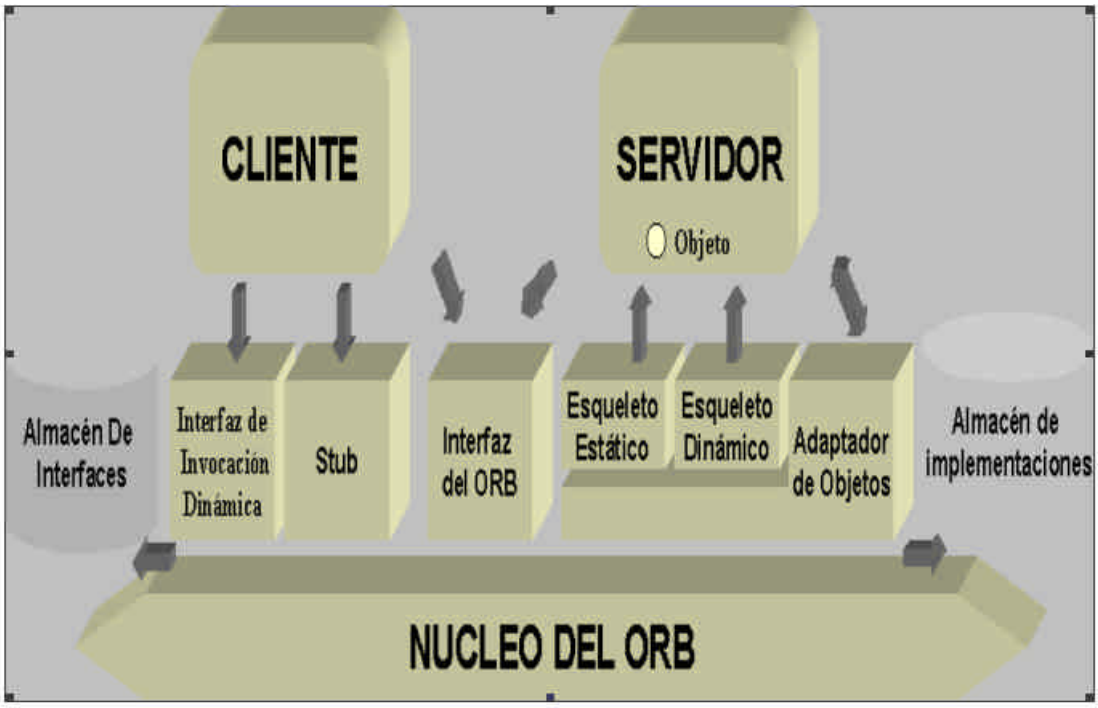


6.4 La arquitectura CORBA.

A continuación se muestra la estructura de un sistema distribuido basado en CORBA. En él puede observarse una parte cliente y una servidora.

En la parte cliente existirá una aplicación cliente propiamente dicha a la que CORBA le añade cierta infraestructura para permitir la comunicación con el servidor a través de la red.

La parte servidora está formada por el objeto que exporta su funcionalidad, integrado en el servidor, es decir, el proceso que se ejecuta y diversos elementos que permiten que las invocaciones realizadas por el cliente a los métodos del servidor lleguen a este, sean procesadas y sus resultados devueltos.



- El ***ORB*** es el sistema intermedio que establece relaciones cliente/servidor entre objetos. Mediante el uso del ORB un objeto cliente puede invocar de manera transparente un método de un objeto servidor que se encuentre en la misma máquina o en otra distinta.

  El ORB intercepta la llamada del objeto que realiza la petición, invoca el método en el objeto servidor, pasa los parámetros y devuelve los resultados.

  También se encarga de convertir los datos a un formato que puede ser transmitido por la red (marshal), y de una vez en el servidor convertir los datos de este formato a uno que el servidor pueda manejar (unmarshal).

  Para este proceso no es necesario que el cliente sepa dónde se encuentra el objeto servidor que ejecuta el método, ni el lenguaje en el que este objeto se encuentra programado, ni el sistema operativo de la máquina que lo contiene, ni ningún otro aspecto que no sea su interfaz.

- Los ***"stubs del cliente"*** proporcionan una capa intermedia entre el cliente y el núcleo del ORB. Definen cómo los clientes invocan los servicios que proporcionan los objetos servidores. El stub se encarga de codificar la operación y sus parámetros y de enviar la petición de

forma remota, haciendo que desde el punto de vista del cliente se tenga la impresión de que la aplicación se ejecuta localmente.

- ***La interfaz de invocación dinámica*** permite descubrir en tiempo de ejecución métodos para ser invocados. CORBA define una biblioteca que permite localizar el método, generar los parámetros necesarios, realizar la llamada remota y recoger los resultados.

  La capacidad de soportar tanto invocaciones dinámicas como estáticas, además del almacén de interfaces, da a CORBA una gran ventaja sobre las plataformas intermedias competidoras. Las invocaciones estáticas son rápidas y fáciles de programar. Las invocaciones dinámicas proporcionan la máxima flexibilidad, pero son difíciles de programar. Esta últimas son muy usadas por herramientas que descubren servicios en tiempo de ejecución.

  Por lo que respecta al lado del servidor no hay diferencia entre la invocación dinámica y estática, ya que ambas tienen el mismo mensaje semánticamente hablando. En los dos casos el ORB localiza al Adaptador de Objetos del objeto y le transmite un mensaje que contiene el nombre del servicio a invocar y sus parámetros. La implementación recibe a través del esqueleto del objeto los datos necesarios, ejecuta el servicio y retorna el resultado para que sea enviado al cliente en forma de un nuevo mensaje.

- ***El almacén de interfaces*** permite obtener y modificar la descripción de todos los componentes que en él están registrados, los métodos que soporta y los parámetros que requiere. CORBA llama a esas descripciones firmas. El almacén de interfaces (Interface Repository) es una base de datos distribuida modificable en tiempo de ejecución.

- ***El interfaz del ORB*** consiste en unas pocas librerías de servicios locales para realizar labores auxiliares en la aplicación. Por ejemplo, CORBA proporciona APIs (Aplication Programming Interface, interfaces de programación de aplicaciones) para convertir referencias a objetos a cadenas. Estas llamadas pueden ser útiles si se necesita comunicar o almacenar referencias a objetos.

Hasta aquí los elementos de la parte del cliente. A continuación se exponen los elementos de la parte servidora.

- ***El esqueleto del servidor*** proporciona los elementos necesarios para que los clientes invoquen los servicios exportados por el objeto

servidor. Estos esqueletos realizan una función similar a los stubs del cliente tratando de hacer transparente todo el proceso de comunicación. El ORB usa esos esqueletos para enviar las peticiones a las instancias de la implementación del servidor, llamadas ***sirvientes.***

- ***El esqueleto de interfaces dinámicos (DSI)*** proporciona un mecanismo de enlazado en tiempo de ejecución para servidores que necesitan manejar llamadas a métodos que no tienen esqueletos estáticos definidos.

- ***El adaptador de objetos*** proporciona en tiempo de ejecución un entorno para la instanciación de objetos en el servidor, la asignación de referencias y la gestión de las peticiones que les lleguen.

- ***El almacén de implementaciones*** proporciona en tiempo de ejecución un almacén de información acerca de las clases que el servidor soporta, los objetos instanciados y sus identificadores.

- ***El interfaz del ORB*** tiene desde el punto de vista del servidor una función análoga a la expuesta desde el punto de vista del cliente.

6.5 Funcionamiento de CORBA.

El funcionamiento de un sistema CORBA puede exponerse a través de la secuencia básica de operaciones que se produce cuando un cliente CORBA intenta ejecutar un método contenido en un servidor CORBA.

- ***1.*** El ORB recibe una solicitud para un objeto que se encuentra en un servidor que está conectado al sistema. El ORB consulta su repositorio y determina si el servidor y el objeto se encuentran activos en ese momento.

- ***2.*** En caso de que el servidor se encuentre inactivo es activado por el ORB, pasándole la información necesaria para interactuar con el ORB por medio de su adaptador de objetos (POA).

- ***3***. El servidor una vez activado hace una llamada al adaptador de objetos para informarle al ORB que ya se encuentra listo para activar objetos.

- ***4.*** En caso de que el objeto no se encuentre activo, el POA llama a la rutina del servidor para activar el objeto necesario, y el servidor activa el objeto.

- ***5***. El POA pasa la invocación al objeto por medio del esqueleto y recibe la respuesta, que es enviada al cliente.

- ***6***. Las solicitudes adicionales recibidas por el POA son manejadas de igual forma que en los pasos 4 y 5.

- ***7***. El servidor puede decidir desactivar un objeto cuando no hay más solicitudes para ese objeto. Si se produce entonces una llamada a este objeto el POA deberá volver a activarlo. Antes de desactivarse el objeto guarda su estado en un almacenamiento persistente.

- ***8***. El servidor puede llegar a desactivarse por completo en caso de que todos sus objetos se encuentren desactivados.

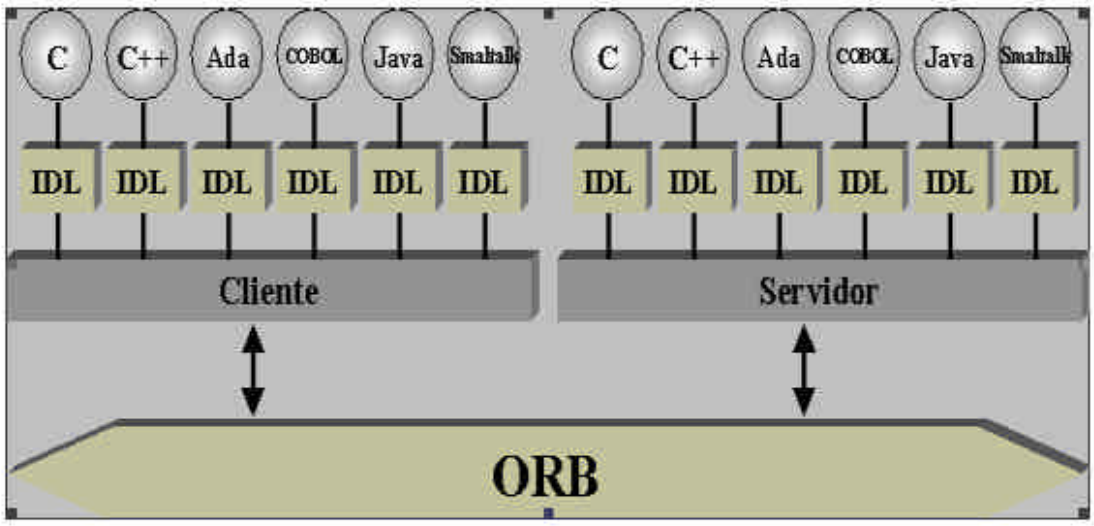
6.6 El lenguaje de definición de interfaces (IDL).

El IDL es el lenguaje en el que se especifican los interfaces de los objetos CORBA. Es un lenguaje de especificación y no puede sustituir el papel de lenguajes de programación como pueden ser C++ y Java, ya que no es capaz de implementar los interfaces que especifica.

CORBA utiliza un lenguaje de especificación, como el CORBA-IDL, para ocultar la implementación del objeto a los posibles clientes.

La especificación IDL se sitúa entre el cliente y la implementación de manera que éste nunca llegue a interactuar directamente sobre la misma. Con ello se consigue la independencia del lenguaje de implementación, lo que permitirá al programador elegir el que más se adapte a sus conocimientos y necesidades.

El programador deberá declarar todos los aspectos del objeto que serán accesibles para los clientes.



El lenguaje IDL es sensible a las mayúsculas, pero impone que la diferencia entre identificadores que se encuentren en el mismo ámbito no puede ser sólo que uno de ellos tenga sus caracteres en mayúsculas y el otro en minúsculas.

IDL también permite el uso de comentarios. Una línea que comienza con los caracteres "//" es considerada como una línea de comentario (estilo Java).

También se considera comentario un bloque de texto que comience con la secuencia de caracteres "/*" y termine con "*/" (estilo C++).

El lenguaje IDL posee un conjunto de palabras reservadas que no pueden ser usadas como identificadores en ningún ámbito y que designan tipos de datos, encapsulamiento de los datos, excepciones, etc

6.6.1 Tipos de datos simples en IDL.

El lenguaje IDL define una serie de tipos de datos simples que pueden ser usados tal cual o combinados entre si para mapear tipos de datos de los lenguajes de programación en los que van a desarrollarse las aplicaciones.

- ***void.*** Es análogo al tipo void de C++ o Java. No puede ser usado para declarar variables ya que no tiene sentido declarar elementos de tipo vacío. Su función es la de ser usado como tipo de retorno en funciones que no devuelven nada.

- ***bolean.*** Almacena valores de tipo lógico. IDL define dos tipos de constantes booleanas: *TRUE* (verdadero) y *FALSE* (falso). Este tipo de datos no existen en C++ aunque sí en Java.

- ***char.*** Es un tipo de al menos ocho bits que admite caracteres del tipo ***"ISO-Latin-1"*** que es un superconjunto de caracteres de los ASCII. IDL también soporta caracteres del tipo "wide char" con lo que permite la utilización de conjuntos de caracteres de tamaño arbitrario como ***UNICODE***.

- ***string.*** Se usa para representar cadenas de caracteres. Es muy similar a las clases Cstring de C++ y String de Java.

  Soportan caracteres del tipo "ISO-Latin-1" excepto el carácter NULL (nulo, '\0') ya que en C y C++ este carácter tiene un significado especial ya que son usados como terminadores de cadena.

  Los strings IDL pueden ser de longitud fija o de longitud variable. Los strings de longitud fija especifican el tamaño máximo de la cadena que van a contener, mientras que los de longitud variable no especifican nada, dando a la cadena la posibilidad de crecer dinámicamente sin ningún límite.

  En IDL las cadenas de tamaño fijo no incorporan en su información el terminador de cadena.

  Al igual que con el tipo char se han definido cadenas de caracteres anchos llamadas "***wstring***".

- ***long.*** En IDL no existe un tipo que represente un entero propiamente dicho como puede serlo "int" en C++ o en Java.

  El tipo long contiene un entero que permite representar valores entre (-2^{31}) y $(2^{31}-1)$ con un tamaño mínimo de 32 bits.

- ***unsigned long.*** Representa un entero sin signo entre 0 y $(2^{31}-1)$.

- ***short.*** Es un entero corto que representa valores entre (-2^{15}) y $(2^{15}-1)$.

- ***unsigned short.*** Es un entero corto sin signo que representa valores entre 0 y $(2^{15}-1)$.

- ***octet.*** Representa magnitudes de ocho bits que no van a ser traducidas durante la transmisión.

- ***float.*** Representa valores en coma flotante de precisión simple.

- ***double.*** Representa valores en coma flotante de precisión doble. Se ha añadido el tipo ***long double*** aunque no tiene un mapeado claro a ningún lenguaje.

- ***any.*** Es el contenedor universal de valores. Una variable del tipo any puede almacenar valores de cualquiera de los tipos CORBA-IDL, incluso tipos definidos por el usuario.

- ***Referencias a objetos.*** Al igual que en Java y a diferencia de C++ no existen punteros, por lo que el manejo de objetos se realiza mediante referencias.

- ***Estructuras.*** Es una construcción capaz de contener elementos de distintos tipos.

- ***Uniones.*** Al igual que las estructuras son capaces de contener elementos de distintos tipos, con la particularidad de que sólo puede albergar un tipo de elemento en cada instante. A diferencia de C y C++ las uniones en IDL son discriminadas, es decir, que contienen un campo que indica en cada instante qué tipo de elemento está en uso.

- ***Arrays.*** A igual que en Java o C++ son agregaciones de elementos del mismo tipo, cuyo número de elementos es limitado y conocido. Se soportan array de datos simples, estructurados, definidos por el usuario, etc, así como arrays de varias dimensiones que deberán ser declarados como tipos definidos por el usuario.

- ***Secuencias.*** Son arrays de longitud variable. Pueden contener datos predefinidos o definidos por el usuario. Pueden ser de longitud ilimitada o bien limitada, en cuyo caso la longitud de la secuencia no podrá superar un cierto tamaño. Las secuencias pueden incluso almacenar otras secuencias.

- ***Tipos definidos por el usuario.*** Mediante el uso de la palabra reservada "typedef" el usuario es capaz de definir nuevos tipos de datos basándose en los ya existes.

- ***Enumeraciones.*** IDL permite la creación de tipos que admiten valores de un conjunto definido por el usuario a través de una enumeración. Son muy parecidos a los que se definen en C++. No se permiten enumeraciones vacías.

- ***Constantes.*** Son idénticas a las constantes de C++. Pueden definirse constantes. Pueden definirse constantes de tipos flotantes, boléanos, cadenas, etc, pero no se permite hacerlo de tipos any ni de tipos complejos definidos por el usuario.

6.6.2 Estructura de un fichero IDL.

El esquema genérico de un fichero IDL se muestra a continuación:

```
Module <identificador>
{
        <declaración de tipos>;
        <declaración de constantes>;
        <declaración de excepciones>;

        interface <identificador> [:<herencia>]
        {
                <declaraciones de tipos>;
                <declaraciones de constantes>;
                <declaraciones de atributos>;
                <declaraciones de excepciones>;

                [<tipo_operación>] <identificador>(<parámetros>)
                [raises exception][context];
                        .
                        .
                        .
                [<tipo_operación>] <identificador>(<parámetros>)
                [raises exception][context];
                        .
                        .
                        .
```

```
        }
        interface <identificador> [:<herencia>]
                        .
                        .
                        .
}
```

Un ***módulo*** proporciona un nombre bajo el cual se engloban un conjunto de definiciones de clases denominadas ***interfaces.*** El nombre del módulo viene dado por uno o más identificadores separados por "::".

El objetivo de los módulos es el de introducir niveles adicionales en la jerarquía de nombres de IDL.

Un ***interface*** define un conjunto de métodos que un cliente puede utilizar en un objeto servidor. Dentro del interface se pueden declarar tipos de datos y constantes que van a ser usadas posteriormente, así como atributos que son valores para los que la implementación crea operaciones para leer el valor del atributo y para escribir un valor en él (operaciones get y set respectivamente), o bien sólo para leer si el atributo es de solo lectura.

Dentro del interface es posible declarar excepciones que indican que una operación no se ha realizado con éxito.

IDL soporta herencia múltiple, de manera que un interface puede derivarse de otros existentes.

Las ***operaciones (o métodos)*** son equivalentes a métodos CORBA. El nombre de la operación va precedido del tipo de dato que devuelve la operación y representa un método disponible en el objeto servidor que define el interface.

Sólo se declaran aquellos métodos que son públicos y van a poder ser accedidos por el cliente.

IDL no permite la sobrecarga de funciones, y los identificadores de los métodos deben ser únicos en el interface en el que son definidos.

Los ***parámetros*** pueden ser pasados del cliente al servidor (in), del servidor al cliente (out), o bien ser usados tanto para enviar como para recibir datos (inout). Esto de indica mediante un cualificador en la definición del parámetro.

También se indican las ***excepciones*** que la operación puede lanzar, y finalmente un conjunto de valores de atributos que describen el ***contexto*** en el que se encuentra el cliente (es equivalente a las propiedades de entorno de UNIX o DOS).

Los ***tipos de datos*** describen los valores aceptados por los parámetros, valores devueltos, excepciones, etc. Se definen independientemente del sistema operativo empleado.

Todos los tipos de datos son convertidos a su tipo correspondiente en el lenguaje adecuado para su posterior implementación.

6.6.3 Correspondencia de datos JAVA-IDL.

La correspondencia de datos entre IDL y Java es más reciente que entre IDL y C++ o C.

Es una correspondencia más simple y directa dado que Java soporta los conceptos de la programación orientada a objetos, el uso de interfaces, etc.

Java no soporta herencia múltiple, con lo cual si una clase implementada en el interfaz ya hereda de alguna otra no podrá heredar de nuevo de otra clase, lo cual supone un problema a la hora de diseñar objetos que necesariamente deben heredar de las clases generadas por el traductor Java-IDL cuando realiza la conversión de la información dada en el fichero IDL.

Otra de las particularidades de Java es que permite al usuario desentenderse de la gestión de memoria, que es un aspecto que suele causar bastantes problemas en otro tipo de lenguajes orientados a objeto como C++.

Cuando se usa el traductor Java-IDL éste genera la implementación Java de los interfaces, módulos y tipos de datos especificados en el fichero idl. Esta implementación consiste en una serie de clases Java que contienen métodos que permiten el manejo de los datos y la definición de los objetos que proporcionan los servicios ofrecidos en el interfaz.

Estas clases especiales son nombradas mediante la concatenación del nombre del interfaz concreto que implementan, y una palabra reservada que identifica la función de la clase.

- ***<tipo>Helper.*** La etiqueta <tipo> designa el nombre del tipo CORBA-IDL definido por el usuario. Esta clase incluye funciones necesarias para manipular el tipo en cuestión. Son operaciones de inserción y extracción de datos, de escritura y lectura del tipo en un stream y la obtención del identificador del almacén y del código de tipo.

  La principal función se llama ***"narrow"*** que se encarga de especificar explícitamente el tipo de dato de un objeto para que Java permita su uso.

- ***<tipo>Holder.*** La etiqueta <tipo> es el nombre del tipo de dato CORBA-IDL definido en la interfaz. Esta clase se encarga de que un parámetro pueda ser definido como de salida, o de entrada/salida, ya que Java sólo admite parámetros de entrada, y en muchas aplicaciones son necesarios parámetros de salida o bien de entrada/salida.

- ***<tipoBásicoJava>Holder.*** Crea clases "Holder" para tipos de datos básicos de Java.

- ***<interfaz>Package.*** Define un paquete java que contiene todas las clases que contiene un interfaz de nombre <interfaz> definido en el fichero idl.

- ***Otras clases.*** También se generan clases que implementan los esqueletos del cliente y de servidor, así como la versión Java del interfaz idl. Los nombres de estas clases dependen del intérprete IDL a Java particular contenido en el ORB concreto que se esté usando.

La representación en IDL de los principales tipos de datos Java se recoge en la siguiente tabla:

TIPO DE DATO JAVA	***TIPO DE DATO IDL***
Bolean	Bolean
Char	char
Char	wchar
Byte	octect
Java.lang.String	string
Java.lang.String	wstring
Short	short
Short	unsingned short
Int	long
Int	unsigned long
Long	long long
Long	unsigned long long
Float	float
Double	double
Public.static.final	const
Array	sequence

Existen otros tipos de datos en IDL cuyo correspondiente dato en Java no es uno de los tipos básicos, así como tipos definidos en IDL por el usuario y excepciones.

- El tipo ***enum*** de IDL genera en Java una clase final del mismo nombre con los siguientes tipos de datos:

```
public final class <enum_name> {
// one pair for each label in the enum
public static final int _<label> = <value>;
public static final <enum_name> <label> =
new <enum_name>(_<label>);
public int value() {...}
// get enum with specified value
public static <enum_name> from_int(int value);
// constructor
private <enum_name>(int) { ... }
    }
```

- El tipo ***struct*** de IDL define una clase final Java que contiene variables que implementan los campos de la estructura y constructores para todos los datos. La clase Holder generada por este tipo de dato tiene la siguiente estructura:

```
final public class <class>Holder implements
org.omg.CORBA.portable.Streamable {
public <class> value;
public <class>Holder() {}
public <class>Holder(<class> initial) {...}
public void _read(org.omg.CORBA.portable.InputStream i)
{...}
public void _write(org.omg.CORBA.portable.OutputStream o)
{...}
public org.omg.CORBA.TypeCode _type() {...}
    }
```

- El tipo ***union*** de IDL genera en Java una clase final del mismo nombre que contiene métodos para cada rama de la unión. La clase Java generada tiene la siguiente estructura:

```
final public class <union_class>Holder
implements org.omg.CORBA.portable.Streamable {
public <union_class> value;
public <union_class>Holder() {}
public <union_class>Holder(<union_class> initial) {...}
public void _read(org.omg.CORBA.portable.InputStream i)
{...}
public void _write(org.omg.CORBA.portable.OutputStream o)
{...}
public org.omg.CORBA.TypeCode _type() {...}
}
```

- El usuario puede definir datos propios basados en los datos ya existentes. Un caso particular bastante importante y de uso muy frecuente es la definición de arrays, que se definen en IDL de la forma

```
typedef <tipo de dato> <nomebre del array>[tamaño];
```

Son mapeados en objetos del tipo array de un determinado tipo de dato en Java.

- Cuando se declara un ***interface*** en IDL se genera un interface en Java con el mismo nombre y su correspondiente clase Helper.

- Las excepciones en IDL son mapeadas a excepciones Java del mismo estilo que cualquiera que un usuario pudiera definir en Java. El desarrollador de aplicaciones Java deberá escribir el código correspondiente para implementar en Java la excepción.

- El tipo de dato ***any*** de IDL se corresponde en Java con la clase org.omg.CORBA.Any que contiene los métodos necesarios para insertar o extraer instancias de tipos definidos.

6.7 Excepciones en CORBA.

El concepto de excepción ha sido adoptado por los desarrolladores para el manejo de las situaciones anómalas que se producen a lo largo de la ejecución, como pueden ser los errores de división por cero, salidas de rango, etc. Ésta es una característica típica de lenguajes como C++ y Java. Cuando una operación genera una excepción durante su ejecución, el ORB será el responsable de notificar al cliente de la situación excepcional que acaba de producirse. Una vez la excepción llega al cliente, ésta busca en orden ascendente un módulo donde ser recogida y manejada. Si ningún módulo es capaz de recogerla, el programa realizará una terminación anormal.

En CORBA-IDL existen dos tipos de excepciones:

- ***Excepciones de usuario.*** Son aquellas que el programador declara al definir la interfaz de un objeto. La declaración de una excepción de usuario se realiza mediante la palabra reservada ***exception*** indicando a continuación el nombre de la misma y el tipo del valor de retorno (entre llaves). Para indicar las diferentes excepciones de usuario que puede elevar un método, se utiliza la palabra reservada ***raises*** y entre paréntesis el nombre de las excepciones separadas por comas. Es obligatorio que todas la excepciones de usuario que puedan elevar un método estén indicadas en su declaración IDL.

- ***Excepciones del sistema.*** Son excepciones estándar definidas por CORBA. Cuando se desea que un método maneje alguna de estas excepciones no es necesario que sean declaradas, ya que lo están implícitamente. Este tipo de excepciones hacen referencias a

operaciones no válidas, datos no válidos, errores en la comunicación, en la conversión de datos, errores internos en el ORB, etc.

6.8 Adaptadores de objetos.

Como se expuso en el apartado “La arquitectura CORBA”, el adaptador de objetos se sitúa como nexo de unión entre las implementaciones de objetos CORBA y el propio ORB, consiguiendo que las peticiones que llegan a través del ORB sean procesadas por la implementación del objeto.

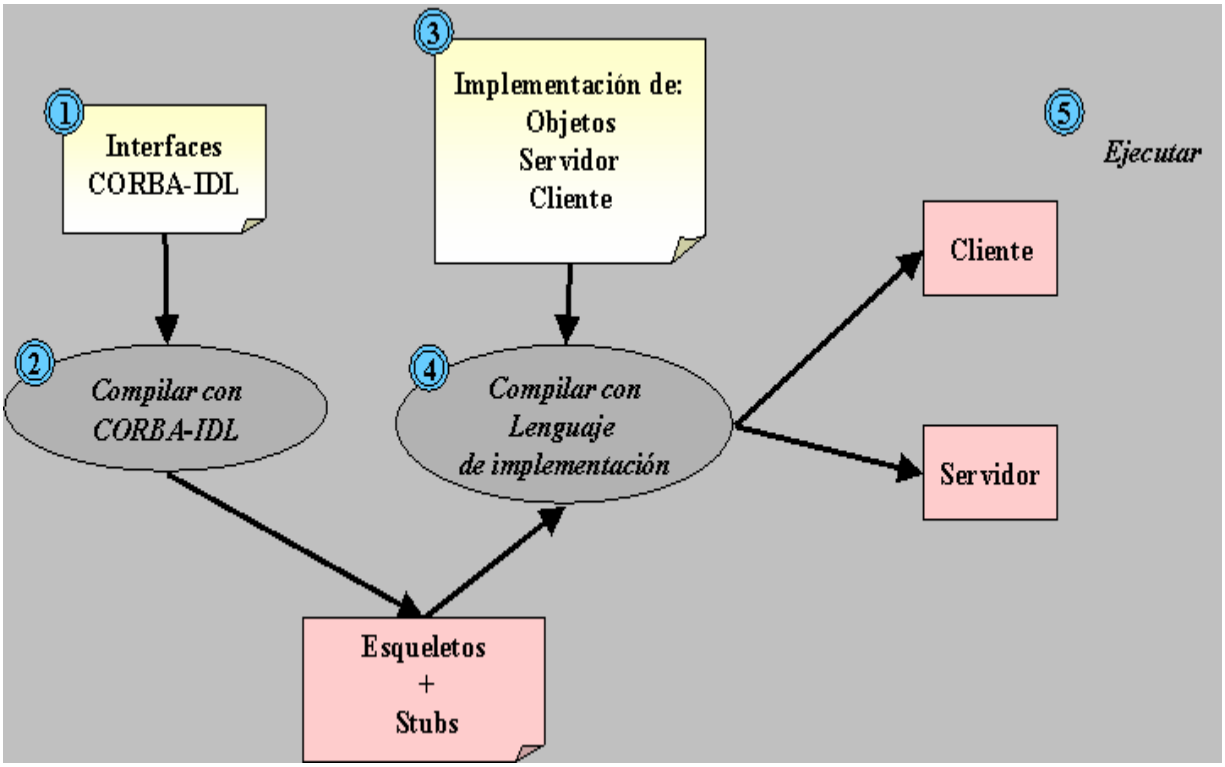
Las funciones de un adaptador de objetos son principalmente:

- ***Registrado de los objetos.*** Deben proporcionar funciones para registrar implementaciones de objetos.
- ***Generación de referencias a objetos.***
- ***Activación de procesos servidores.***
- ***Activación de objetos.***
- ***Multiplexación de peticiones a los objetos registrados.*** Debe asegurar que todas las peticiones sean recibidas por los objetos, aunque tengan múltiples conexiones sin que ninguna se bloquee indefinidamente.
- ***Gestión de las invocaciones.***

En la norma CORBA han aparecido dos adaptadores de objetos, el actual adaptador ***POA*** (Portable Object Adaptor) y el original y ya desaparecido ***BOA*** (Basic Object Adaptor).

6.9 Desarrollo de aplicaciones CORBA.

La creación de una aplicación CORBA es un proceso claramente definido tanto en las etapas que lo forman como en el orden de las mismas, lo cual permite generar aplicaciones de forma rápida y sistemática.



El gráfico anterior detalla por pasos el proceso de creación de una aplicación CORBA.

- ***1.*** En primer lugar se definen los interfaces de los objetos que van a participar en la aplicación mediante el lenguaje IDL. Este interfaz permitirá que los demás objetos conozcan las operaciones que tienen disponibles y cómo pueden ser invocadas.

  IDL define los métodos que el objeto exporta, los parámetros de los mismos, los atributos, excepciones, etc.

- ***2.*** Se compilan los interfaces con el compilador CORBA-IDL. De esta manera se obtienen los esqueletos del cliente y el servidor en el lenguaje de implementación que se esté usando.

- ***3.*** Crear la implementación de los objetos, el servidor y el cliente, es decir, se procede al desarrollo de las clases que dan funcionalidad al objeto y el código tanto del servidor como del cliente.

- ***4.*** Se procede a la compilación del código, es decir, a la creación de los ejecutables de la futura aplicación. Por un lado se compila la implementación del servidor y la de los objetos esqueletos dando lugar así a la parte servidora de la aplicación.

  Por otro lado se compila el cliente junto con los stubs para dar lugar a la parte cliente.

- ***5.*** Se procede a la ejecución de la aplicación. Este paso dependerá del ORB que se esté usando. En algunos casos previamente a la ejecución habrá que registrar el servidor en el almacén de implementaciones; en otros casos el registro será automático y en otros casos al registrar el servidor será el propio almacén de implementaciones el que arranque el servidor.

  Como norma general deberá arrancarse primero el servidor y a continuación el cliente.

6.10 Invocación dinámica.

Existen ocasiones en las que el cliente de un sistema distribuido basado en CORBA no conoce el interfaz de los objetos con los que desea trabajar en tiempo de compilación. En estos casos la invocación dinámica permite en tiempo de ejecución, la localización del interfaz y la generación dinámica de una invocación.

Una invocación dinámica se realiza en las siguientes etapas:

- Obtención del nombre del interfaz.
- Obtención de la descripción del método en el almacén de interfaces.
- Creación de la lista de argumentos.
- Creación de la petición.
- Realización de la petición.

6.11 Servicios CORBA.

Los servicios CORBA son conjuntos de objetos pre-implementados que complementan la funcionalidad de otros objetos para dar lugar a una aplicación final. Su propósito es general, es decir, son independientes del dominio de aplicación.

La norma CORBA especifica los siguientes tipos de servicios:

- ***Arranque.***
- ***Ciclo de vida.*** Este servicio ha sido pensado para crear, mover, copiar y eliminar objetos dentro de un entorno distribuido.
- ***Colección.*** Proporciona un mecanismo uniforme para la creación y manipulación uniforme de las colecciones de objetos más comunes, tales como listas, conjuntos, árboles.
- ***Concurrencia.*** El servicio de control de la concurrencia es mediar en el acceso concurrente a un objeto. Con ello se pretende que su consistencia no se vea comprometida cuando se genera concurrencia en la realización de acciones computacionales. De esta manera se permite a varios clientes coordinar sus accesos a recursos compartidos.
- ***Consulta.***
- ***Externalización.***
- ***Eventos.***
- ***Licencia.***
- ***Negociación.***
- ***Nombrado.*** El servicio de nombrado es el principal mecanismo para que los objetos de un ORB puedan localizar a otros objetos. Los nombres que identifican un objeto son familiares para el usuario. El servicio de nombrado mapea esos nombres a referencias a objetos. La asociación nombre-objeto se denomina vínculo de nombre. Un vínculo de nombre siempre está referido a un nombre de contexto que es un espacio en el cual el nombre del objeto es único.

Opcionalmente se pueden asociar uno o más nombres con la referencia de un objeto.

- ***Persistencia.***
- ***Propiedades.***
- ***Relación.***
- ***Seguridad.***
- ***Tiempo.***
- ***Transacciones.***

6.12 Facilidades CORBA.

Las Facilidades CORBA son servicios de alto nivel, al estilo de funciones de aplicación genéricas, que pueden ser configuradas para una necesidad específica en cualquier campo de aplicación. Se conocieron inicialmente como Facilidades Comunes Horizontales, para diferenciarlas de las Facilidades Comunes Verticales (llamadas actualmente Interfaces de Dominio) que son específicas para determinados ámbitos de aplicación; hoy en día se denominan "***CORBAfacilities***".

Mientras que los Servicios de Objetos están orientados a la manipulación de los objetos distribuidos, las Facilidades Comunes están orientadas al usuario final, brindándole servicios tales como impresión de documentos, correo electrónico, etc. El OMG definió originalmente cuatro categorías principales de Facilidades Comunes:

- ***Interfaz de usuario.*** Facilidades para el acceso del usuario a la aplicaciones y presentación de la información. Una de ellas es la Facilidad de Presentación Compuesta (Compound Presentation Facility), que provee un entorno para la compartición y subdivisión de las ventanas de despliegue de una arquitectura de documentos compuestos.

- ***Gestión de Información***. Facilidades para el modelado, almacenamiento, recuperación e intercambio de información. Un ejemplo es la Facilidad para el Almacenamiento y Recuperación de Información (Information Storage and Retrieval Facility), que es

aplicable a diferentes servicios de información como acceso a bases de datos, directorios, búsquedas en Internet, etc.

- ***Gestión de Sistemas***. Facilidades para la gestión de sistemas de información tales como control, monitorización (monitoreo) seguridad, configuración y políticas de operación (adición de usuarios, permisos, instalación de programas, etc).

- ***Gestión de Tareas***. Facilidades para la automatización del trabajo, brindando soporte tanto a los procesos de los usuarios como a los procesos del sistema. Por ejemplo, la Facilidad de Flujo de Trabajo (Workflow Facility), ofrece servicios para la gestión y coordinación de objetos que son parte de procesos de trabajo.

6.13 Dominios CORBA.

Así como las Facilidades Comunes, las Interfaces de Dominio definen funciones de aplicación genéricas, pero en este caso corresponden a dominios de aplicación específicos como comercio electrónico, finanzas, medicina, manufactura, telecomunicaciones, transporte, etc.

Estos servicios se conocen como "***CORBAdomains***", y en su implementación se pueden combinar algunas Facilidades Comunes y Servicios de Objetos. En la actualidad existe un gran número de Comisiones, Grupos Especiales de Interés y Grupos de Trabajo dedicados a su estudio, los cuales están desarrollando una actividad muy intensa que se refleja en las especificaciones en proceso y aprobadas.

Es especialmente interesante el ***Dominio de Telecomunicaciones. "Telecomunication Task Force"*** promueve el uso de CORBA como solución a las necesidades de la industria de telecomunicaciones.
Se han adoptado especificaciones como:

- Notification Service, Doc. OMG # telcom/98-06-15, Junio de 1998.
- Control and Management of A/V Streams, Doc. OMG # formal/98-07-12, Julio de 1998.

Existen otras especificaciones que se encuentran en proceso de adopción:

- CORBA/TMN and CORBA/SNMP Interworking (RFP)
- CORBA/Intelligent Networks Interworking (RFP)

- Telecom Log Service (RFP) (Similar a X.735 LCF)
- Wireless Access & Control (RFI)

6.14 El protocolo GIOP.

El Protocolo General Inter ORB (General Inter.-ORB Protocol) es el artífice de la interoperabilidad de ORBs de diferentes vendedores. El GIOP se implementa sobre protocolos de transporte orientados a conexión.

La especificación del protocolo GIOP consiste en los siguientes elementos:

- ***El CDR (representación común de datos).*** Define cómo deben codificarse los datos para su transferencia entre clientes y servidores.

- ***El formato de los mensajes GIOP***. Define los mensajes que se intercambian el ORB del cliente y el del servidor cuando se realizan invocaciones sobre objetos, se localizan implementaciones y se controlan los canales de comunicación.

- ***Condiciones de la capa de transporte***. La especificación GIOP describe condiciones que la capa de transporte debe cumplir para poder transmitir mensajes GIOP. La especificación también describe cómo se han de gestionar las conexiones y las restricciones en el orden de los mensajes GIOP.

6.15 El protocolo IIOP.

El protocolo ***IIOP*** es una implementación particular del protocolo GIOP.

El IIOP no es una especificación separada, sino que es una especialización o mapeado del GIOP al protocolo especifico TCP/IP. La especificación GIOP se considera una base para futuras implementaciones a otros protocolos de transporte.

La especificación IIOP (Internet IOP) describe cómo los agentes abren conexiones TCP/IP y las usan para transferir mensajes GIOP.

Un ORB puede soportar cualquier protocolo pero siempre debe incluir el protocolo IIOP.

El protocolo a usar se negocia al establecerse la comunicación entre los ORBs.

6.16 ORBs Comerciales.

Este apartado se mencionan algunas implementaciones de ORBs de fabricantes concretos que han alcanzado un cierto grado de éxito entre los desarrolladores de aplicaciones CORBA.

- ***ORB MICO.*** Mico es un ORB educacional que no necesita equipos de grandes prestaciones. Ha sido desarrollado por la Universidad de Frankfurt y aunque su rendimiento no es muy bueno es muy interesante para empezar.
  Está disponible en www.mico.org.

- ***ORB BACUS.*** Es un ORB de libre distribución para fines no comerciales. Su rendimiento es bastante bueno. Ya está actualizado a la norma CORBA 2.3. Ha superado sus pequeñas incompatibilidades en el mapeado a C++. También permite el mapeado a Java. Está disponible en www.occ.com/ob.

- ***ORB TAO.*** TAO es un ORB desarrollado en la Universidad de Washington por Douglas Schmidt y su equipo. Está considerado hoy por hoy como el ORB con mejores prestaciones. Permite la creación de servidores multihilo y tiene servicios para tiempo real.
  Está disponible en www.cs.wustl.edu/~schmidt/TAO.html.

- ***ORBexpress.*** El ORBexpress es un ORB desarrollado por Object Interface Systems. Permite la creación de servidores multihilo, incluido el thread pool.
  Se encuentra disponible en www.ois.com.

- ***JavaORB.*** Es el ORB desarrollado por Sun. Se incluye en las distribuciones de Java 2 SDK.
  Se encuentra disponible en java.sun.com.

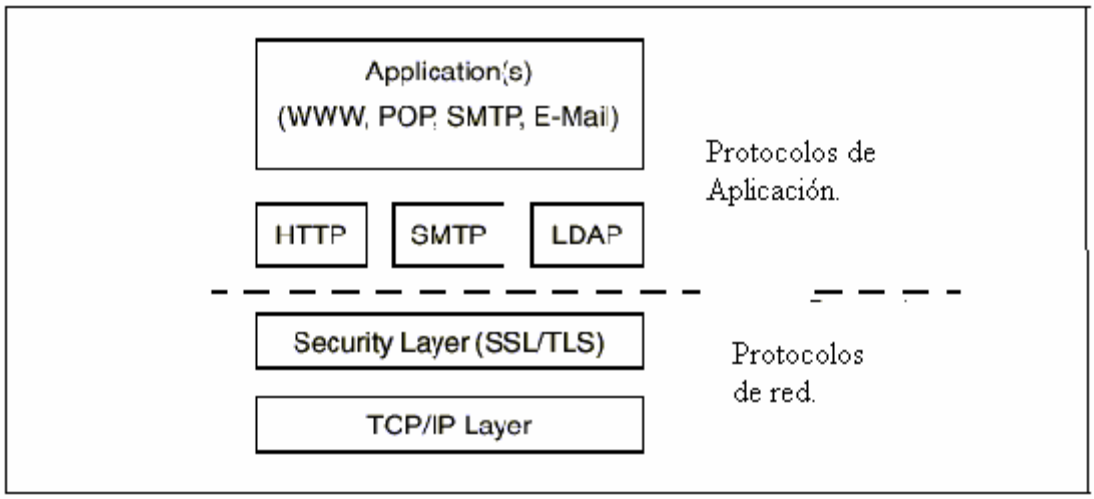
- ***JACORB.*** Está desarrollado por la universidad de Berlín. Es una distribución escrita en Java al cien por cien, sin código nativo. Está disponible en www.Jacorb.org.

Capítulo 7: El protocolo SSL (Socket Security Layer).

El protocolo SSL ha sido diseñado para proporcionar en un mismo protocolo mecanismos que permitan la autentificación de entidades y la transmisión segura de datos.

Los sockets SSL engloban a los sockets TCP/IP con lo cual cualquier aplicación que funcione sobre TCP/IP puede usar sockets SSL para llevar a cabo la comunicación.

La estructura de una pila de protocolos en la que se encuentre SSL tiene la siguiente estructura:



7.1 El protocolo TLS (TRANSPORT SECURITY LAYER).

SSL fue desarrollado por Netscape. Su gran aceptación ha llevado a la aparición del protocolo ***TLS (Transport Security Layer)*** que es la evolución de la versión 3.0 del protocolo SSL, difiriendo de éste en algunos aspectos de relevancia menor (TLS soporta un mayor número de algoritmos de cifrado).

Las previsiones de mercado indican que en un futuro no muy lejano TLS reemplazará a SSL.

El resto de características, generalidades y métodos que se exponen son aplicables a ambos protocolos.

7.2 Autentificación en SSL/TLS.

Esta familia de protocolos soporta ***autentificación del servidor***. Este proceso consiste en el intercambio de un conjunto de información mediante la cual el cliente autentifica al servidor.

De la misma manera también soporta ***autentificación del cliente,*** proceso en el cual se intercambia información con el fin de que el servidor pueda autentificar al cliente.

Cuando se lleva a cabo la autentificación tanto en el cliente como en el servidor el proceso se conoce como ***autentificación mutua.***

Además de la autentificación esta familia de protocolos ofrece privacidad de la información mediante la encriptación de los datos que circulan por la red.

Estos protocolos usan un método de clave pública para asegurar la comunicación y para autentificar las partes de la conexión. Esto se lleva a cabo mediante el uso de un par clave pública/clave privada.

La información intercambiada entre el cliente y el servidor durante la comunicación se encripta usando la clave privada y puede ser desencriptada usando la clave pública y viceversa.

7.3 Certificados de clave pública.

Los certificados de clave pública proporcionan un método seguro a las entidades para comunicar su clave pública a otra entidad en escenarios en los que se usan claves asimétricas.

Este tipo de método supone la evolución de los métodos de clave asimétrica, solucionando los problemas que se derivan del uso de claves.

- ***Métodos de clave secreta.*** En este método tanto el cliente como el servidor encriptan y descifran la información usando la misma clave (clave secreta). Antes de comenzar a intercambiar información cifrada, ambos extremos de la comunicación han debido negociar qué clave y qué algoritmo van a usar.

  El gran problema de este tipo de métodos está en la vulnerabilidad que presentan durante la negociación de la clave secreta, ya que este proceso no se realiza de forma segura y terceras entidades podrían aprovecharlo para conseguir la clave secreta y hacer inútil el cifrado de la información.

  Si este problema consigue solventarse el método es bastante bueno y eficiente, ya que la gran cantidad de claves posibles y la complejidad y fiabilidad de los algoritmos de cifrado usados proporcionan un robusto método de protección de la información.

Este método también se conoce como de clave simétrica ya que la misma clave se usa para cifrar y descifrar la información.

- ***Métodos de clave pública.*** Este tipo de métodos resuelve el problema de distribución de claves de los anteriores. Para ello los dos extremos de la comunicación guardan una clave propia llamada clave privada que no necesita ser comunicada.

  Además de estas claves ambos tienen una segunda clave propia llamada clave pública y que es comunicada de manera no segura y abierta, sin importar qué otras entidades pudieran llegar a conocerla. Las claves pública y privada de una entidad son criptográficamente inversas, es decir, que la información cifrada con la clave pública puede descifrarse con la privada y viceversa.

  De esta manera un extremo de la comunicación cifra su información usando la clave pública del otro extremo, de manera que sólo el destinatario que posee la clave privada correcta será capaz de descifrarla.

  Este proceso proporciona una gran robustez en cuanto al cifrado de la información, pero presenta el problema de no poder realizar una autentificación totalmente fiable ya que si un extremo cifra un mensaje usando su clave privada, el otro extremo podrá autentificarlo si es capaz de descifrar esta información con la clave pública del extremo emisor, pero este proceso podría llevarlo a cabo cualquier entidad ajena a la comunicación ya que también podría conocer la clave pública.

  Este tipo de métodos también se conocen como de clave asimétrica ya que no se usa la misma clave para cifrar y descifrar la información.

Los certificados de clave pública buscan solucionar situaciones en las que una entidad envía a otra una clave pública correcta, pero diciendo ser una entidad que no es, es decir, el receptor puede certificar que la información que recibe realmente proviene del emisor, pero no podrá solventar la duda de si el emisor es realmente quien dice ser.

Un certificado puede ser interpretado como un pasaporte digital expedido por una organización acreditada (conocida como entidad de certificación) y que permite identificar al propietario.

Para obtener un certificado expedido por una entidad de certificación el usuario debe probar su identidad, y en ese caso la entidad firmará el certificado.

Cabe la posibilidad de crear certificados firmados por el propio usuario, si bien, es evidente que su fiabilidad se reduce a entornos en los que la firma del usuario que lo crea sea fiable (por ejemplo, será fiable entre aplicaciones que maneje el propio usuario, o en entornos que este controle, pero no para comunicarse con otro usuario que no conozca al primero).

Los campos más significativos que contiene un certificado son:

- ***Entidad emisora.*** Es la entidad que expide el certificado. Puede ser el propio usuario si el avala la validez del certificado.

- ***Periodo de validez.*** Es la fecha en la que el certificado deja de ser válido.

- ***Subject.*** Representa la entidad a la que el certificado identifica.

- ***Subject´s public key.*** Contiene la clave pública. Es la información principal que el certificado proporciona. El resto de la información del mensaje sólo trata de asegurar la validez de esta clave.

- ***Signature.*** El certificado está digitalmente firmado por la entidad que lo expide y dicha firma generada a partir de la clave privada de la entidad que garantiza la validez del certificado.

De esta manera si una entidad identifica a otra sólo cuando recibe un certificado con una clave pública que identifica a esta segunda entidad, no podrá ser engañada por otras entidades que intenten hacerse pasar por otras que no son.

Varios certificados pueden ser concatenados en una cadena de certificados con lo cual se consigue autentificar por separado a la entidad que lo expide (primer certificado de la cadena), la entidad a la que identifica (el segundo certificado de la cadena), entidades que emitieron el certificado anterior (siguientes certificados) y la autoridad de certificación principal (el último certificado).

7.3.1 Generación de un certificado de clave pública.

El proceso para la obtención de un certificado de clave pública consiste en solicitar a una entidad de certificación que expida un certificado de este tipo en el cual se identifique al solicitante que previamente habrá demostrado su identidad ante esta entidad.

En otras ocasiones, ya sea porque el proceso anterior puede resultar lento y tedioso, o porque el certificado va a ser usado en un entorno en el cual un cierto usuario puede avalar la autenticidad de los certificados, es posible generar localmente certificados de clave pública que serán firmados por el propio usuario que los genera con el fin de que otras entidades puedan usarlos.

Para generar estos certificados localmente se cuenta con la herramienta ***Keytool*** que incorporan tanto las versiones 1.3 como 1.4 de la distribución Java 2 SDK y con la herramienta ***openssl*** que incorpora el software de libre distribución ***OpenSSL*** en su versión 0.9.6c.

7.3.1.1 Generación de un certificado mediante OpenSSL.

El proyecto OpenSSL busca el desarrollo de un conjunto de herramientas robustas, con buena presentación, gratuitas y de libre distribución que implementen los protocolos SSL y TLS así como un conjunto de librerías de cifrado de propósito general.

OpenSSL proporciona un conjunto de herramientas que proporcionan implementaciones de TLS y SSL, algoritmos de cifrado, generación de claves públicas, generación de certificados X509v3, estructuras de datos, métodos para manejar este tipo de información, etc.

Este software se encuentra disponible en www.openssl.org y es de fácil instalación (básicamente ejecutar un guión de configuración e instalarlo con la herramienta “make” de linux)

La herramienta que permite generar certificados del tipo X.509v3 es ***“openssl”.***

La siguiente línea de comando permite la generación de un certificado de este tipo.

Openssl req –new –X509 –nodes –out [nombre de fichero1.pem] –keyout [nombre de fichero2.pem] –days [número de días]

Esta línea crea un certificado de nombre "nombre de fichero1.pem" con una clave que se encuentra en el fichero de nombre "nombre de fichero2.pem" y con una validez de un periodo de tiempo dado por el campo [número de días].

Durante la generación se solicita al usuario:

- ***Country Name.*** Debe escribirse dos letras que identifiquen el país (por ejemplo SP).

- ***State or province Name.*** Se da el nombre entero del estado o de la provincia en la que se encuentra la entidad (por ejemplo Sevilla).

- ***Locality Name.*** Nombre de la ciudad en la que se encuentra la entidad (por ejemplo Sevilla).

- ***Organization Name.*** Nombre de la organización a la que representa la entidad (por ejemplo mi_organización).

- ***Organizational Unit Name.*** Nombre de una unidad dentro de la organización a la que representa la entidad (por ejemplo mi_departamento).

- ***Common Name.*** El nombre que identifica a la entidad. Suele ser el nombre del equipo. Se aconseja no usar la dirección IP del equipo en este nombre (por ejemplo ait21.us.es).

- ***Email Address.*** Dirección de correo en la que contactar con el usuario (por ejemplo mi_dirección@my_dominio.es).

7.3.1.2 Generación de un certificado mediante Keytool.

La herramienta keytool es una utilidad que permite la gestión de claves y certificados para lo cual maneja un almacén de claves privadas ("keytool") y los certificados X.509 asociados a esas claves privadas que autentifican su clave pública correspondiente.

El proceso de generación de certificados X.509 es más complejo que con la herramienta openssl, ya que parte de una pareja identificador/clave privada guardada en el almacén de claves y genera para dicha pareja el correspondiente certificado.

La orden a ejecutar es:

```
keytool -certreq –alias [nombre] -file [nombre de
fichero.csr]
```

Genera un certificado para la entidad representada en el almacén de claves por el identificador [nombre] y lo guarda en el fichero [nombre de fichero.csr].

Esta herramienta también permite gestionar el almacén de claves, para lo cual incorpora opciones para importar certificados y asociarlos con entradas del almacén, exportar certificados de entradas del almacén, listar el almacén, eliminar entradas, cambiar nombres, claves, etc.

La opción que permite importar certificados es especialmente útil a la hora de incorporar al almacén de claves certificados generados por otras herramientas, como puede ser openssl.

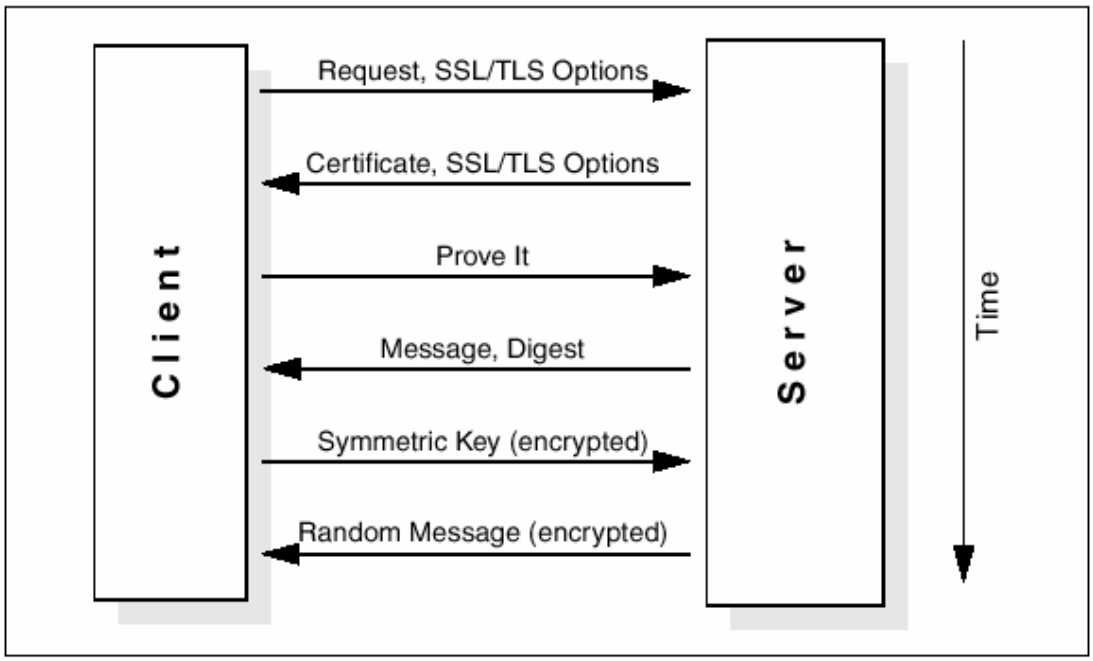
El siguiente comando permite añadir al almacén de claves el certificado contenido en el fichero [fichero.cer] y asociarlo con la entrada del almacén identificada por el campo [alias].

```
keytool -import -alias [alias] -file [fichero.cer]
```

7.4 NEGOCIACIÓN DE LA CONEXIÓN SSL

El proceso que a continuación se expone parte de la suposición de que el servidor ya tiene generado de alguna manera su par de claves.

El proceso se presenta gráficamente a continuación:



En el primer paso el cliente solicita al servidor el establecimiento de una conexión sobre SSL (o TLS) para lo cual envía una solicitud de establecimiento de conexión y envía en dicha solicitud las opciones que soporta.

El servidor contesta al cliente con un mensaje en el que le comunica las opciones que soporta y un ***certificado*** (es información de autentificación que puede ser interpretada como un pasaporte electrónico) que incluye entre otra información la clave pública del servidor, el nombre de la identidad para la cual el certificado fue generado, el nombre del certificado y una marca de tiempo que indica la validez del certificado.

El certificado debe ser expedido por una ***autoridad de certificación*** que garantice que la clave pública realmente pertenece a la entidad mencionada en el certificado.

El certificado se encuentra firmado por el certificador que puede ser autentificado con la clave pública del certificador.

A continuación el cliente solicita al servidor que se autentifique para verificar que realmente fue el servidor quien envió el certificado y no una tercera entidad.

El servidor contesta a esta solicitud con un mensaje que contiene un mensaje del tipo extracto (parecido a una suma de seguridad) encriptado con la clave privada.

El mensaje de tipo extracto se recupera del mensaje completo mediante unas funciones que trocean dicho mensaje.

Este tipo de mensaje presenta la particularidad de que es extremadamente difícil de trastocar y casi imposible de encontrar un mensaje que pudiera generar el mismo mensaje extraído.

El cliente puede descifrar el mensaje con la clave pública del servidor y a continuación comparar el resultado con el esperado. Si ambos coinciden el servidor queda autentificado y termina el proceso de autentificación.

A continuación cliente y servidor deben ponerse de acuerdo sobre la clave secreta (y simétrica) que va a usarse para encriptar los datos.

La encriptación se realiza mediante un algoritmo de clave simétrica ya que es más eficiente que el algoritmo de clave pública.

Para ello el cliente genera una clave simétrica, la encripta con la clave pública del servidor y la envía al servidor. Sólo el servidor con su clave privada podrá descifrar la clave secreta que ha sido enviada por el cliente.

Por último el cliente recibe la clave secreta, la descifra, y envía al cliente un mensaje encriptado con la clave secreta con el fin de verificar que la clave secreta no ha sufrido alteraciones.

A partir de ese momento ambos pueden intercambiar información cifrada mediante el uso de la clave secreta.

Capítulo 8: Desarrollo del sistema.

A continuación se pasa a describir los elementos del sistema, los criterios seguidos para su diseño, la implementación de dichos elementos y el funcionamiento del sistema.

8.1 Instalación del servidor de directorio.

El primer paso para el desarrollo del sistema es la instalación del servidor de directorio LDAP.

El servidor de directorio LDAP elegido es OpenLDAP, que presenta las siguientes ventajas:

- Es gratuito.
- Fácil de manejar.
- Incorpora diferentes bases de datos.
- Se encuentra en constante evolución.
- Funciona sobre sistemas Linux.

Las desventajas principales que presenta OpenLDAP son:

- Interfaz de usuario poco desarrollada.
- No soporta directamente autentificación SASL.
- Soporta un conjunto muy limitado de controles y extensiones en comparación con otros servidores comerciales.

Tras sopesar las ventajas y desventajas desde el punto de vista del escenario que se desear implementar se concluye que las prestaciones que ofrece OpenLDAP encajan con la idea del proyecto mientras que las desventajas son razonablemente tolerables en estas circunstancias.

El servidor OpenLDAP se encuentra disponible en ***www.openldap.org***, aunque la manera más fácil de conseguirlo es desde un servidor de la Universidad de Oviedo, al cual la dirección anteriormente mencionada remite al usuario que intente descargar el software desde España.

La versión de OpenLDAP elegida para la implementación del sistema es la ***versión 2.0.21*** que implementa un servidor de directorio basado en el protocolo LDAP v3.

Se ha mencionado que OpenLDAP se encuentra en constante evolución. Esto ha podido comprobarse durante la realización del proyecto ya que la primera versión del servidor que se eligió fue la versión 2.0.17, pero durante esta fase fueron saliendo nuevas versiones que mejoraban ligeramente a la anterior, y eso unido a la facilidad para migrar la información de un servidor OpenLDAP a otro aconsejaba la actualización más o menos periódica del servidor.

A día de hoy la última versión de OpenLDAP es la versión 2.1.1 aunque aún no es un software totalmente depurado y fiable.

El software se descarga en un fichero comprimido de un tamaño aproximado de 2 MB.

La instalación en un sistema Linux es relativamente simple, básicamente hay que seguir los siguientes pasos si se tienen privilegios de administrador de sistema:

- Desempaquetar el software. Puede hacerse mediante herramientas linux tales como ***"tar"***, ***"gunzip"*** o la utilidad archivador.

- Ejecutar el guión de instalación ***"./configure"*** que se encuentra en el directorio en el que se descomprimió OpenLDAP. Normalmente este guión detecta las opciones de instalación adecuadas entre las que se encuentra la gran mayoría de las características soportadas, pero el instalador puede variar alguna de estas opciones si lo desea (directorio en el que se instala el software, activar o desactivar bases de datos, mecanismos de autentificación, etc).

Para ver qué opciones de instalación puede configurar se ejecuta la orden ***./configure –help*** y estas opciones se muestran por pantalla. Simplemente habrá que añadir las opciones deseadas como argumentos de ./configure.

En el caso que nos ocupa el servidor ha sido instalado ejecutando el guión de configuración sin argumentos, con lo cual el directorio usado para la instalación es /usr/local/etc/openldap (directorio por defecto)

- Se procede a construir las dependencias mediante la orden ***make depend*** .

- Se compila el servidor ejecutando ***make .***

- Por último se procede a la instalación mediante ***make install .***

Se recomienda monitorizar la instalación con el fin de detectar posibles errores que dado el caso abortarían la instalación.

Una vez instalado el servidor se procede a su configuración. Este proceso se realiza en OpenLDAP a través del fichero de configuración que por defecto es /usr/local/etc/openldap/slapd.conf. El administrador del sistema de directorio debe actualizar este fichero con las características que desee implantar en el servidor, el tipo de base de datos de fondo que va a usarse, las raíces de los DIT que van a usarse, el control de acceso al directorio, los esquemas que se usan, etc.

El fichero de configuración debe ser inaccesible para cualquier otro usuario que no sea el administrador.

Una vez realizada esta tarea por parte del administrador de directorio se obtiene el siguiente fichero de configuración para el sistema.

```
# $OpenLDAP: pkg/ldap/servers/slapd/slapd.conf,v
# 1.8.8.7 2001/09/27 20:00:31 kurt Exp $
#
# See slapd.conf(5) for details on configuration options.
# This file should NOT be world readable.
#
include         /usr/local/etc/openldap/schema/core.schema
include         /usr/local/etc/openldap/schema/hdb.schema
include         /usr/local/etc/openldap/schema/ciudades.schema

# Define global ACLs to disable default read access.

# Do not enable referrals until AFTER you have a working directory
# service AND an understanding of referrals.
#referral     ldap://root.openldap.org

pidfile         /usr/local/var/slapd.pid
argsfile      /usr/local/var/slapd.args

#TLSCertificateFile      /root/certificados/serv3.pem
#TLSCertificateKeyFile   /root/certificados/servk3.pem
#TLSCACertificateFile    /root/certificados/serv3.pem



# Load dynamic backend modules:
# modulepath     /usr/local/libexec/openldap
# moduleload     back_ldap.la
# moduleload     back_ldbm.la
# moduleload     back_passwd.la
# moduleload     back_shell.la

#
# Sample Access Control
#       Allow read access of root DSE
#       Allow self write access
#       Allow authenticated users read access
#       Allow anonymous users to authenticate
#

#access to *  by sockurl="^ldapi:///$" write
#access to * by dn="cn=Manager,o=miempresa.es" write
#access to dn=* by * read
```

```
#access to *
        by self write
        by users read
        by anonymous auth

# if no access controls are present, the default is:
#       Allow read by all
#
# rootdn can always write!

access to * attr=userPassword
        by anonymous auth
        by * auth

access to *
        by anonymous read
        by dn="cn=usu2,us=usuarios,di=ciudades" search
        by * read



#######################################################
 ldbm database definitions
#######################################################

database        ldbm
suffix      "di=ciudades"
#suffix             "o=JNDITutorial"
rootdn              "cn=Manuel,di=usuarios,di=ciudades"
#rootdn             "cn=Manager,o=My Organization Name,c=US"
# Cleartext passwords, especially for the rootdn, should
# be avoid.  See slappasswd(8) and slapd.conf(5) for details.
# Use of strong authentication encouraged.
rootpw              secret
# The database directory MUST exist prior to running slapd AND
# should only be accessible by the slapd/tools. Mode 700
recommended.
#directory  /usr/local/var/openldap-ldbm
directory /usr/local/var/ciu
# Indices to maintain
#index      cn,sn,uid,Habitantes,Superficie
index objectClass       eq
```

Las primeras líneas son comentarios que trae por defecto el fichero de configuración.

A continuación se incluyen los ficheros con información de esquema deseados. En este caso el fichero “core.schema” que contiene las definiciones de esquema básicas de OpenLDAP. El fichero “hdb.schema” también viene incluido en OpenLDAP y contiene clases comúnmente usadas.

El tercer fichero de esquema de nombre “ciudades.schema” que se incluye ha sido diseñado por el autor y contiene las definiciones de clases objeto y atributos necesarias para dar soporte a un directorio de ciudades cuyo diseño e implementación se expondrá posteriormente, y que será la información almacenada en el directorio para todo el proceso de pruebas que se lleve a cabo.

Tras incluir los ficheros de esquema se definen los ficheros “pidfile” y “argsfile”. Estos son ficheros de texto cuya ubicación se indica aquí y que tienen la siguiente misión:

- El servidor escribe en el fichero “pidfile” el identificador de proceso que le ha sido asignado dentro del sistema linux.

- En el fichero “argsfile” se registran los argumentos con los que se ejecutó el servidor “Slapd” en la línea de comandos.

A continuación aparecen tres líneas entre comentarios que fueron usadas para designar un certificado X.509 y un almacén de claves a fin de llevar a cabo el proceso de autentificación del servidor cuando arranca sobre un puerto SSL. Para ello se nombra (por este orden) el fichero que contiene el certificado de la entidad, el fichero que implementa el almacén de claves y el fichero que contiene el certificado que identifica a la autoridad que firma el certificado.

Las siguientes líneas significativas que se encuentran comienzan con la directiva “acces to” y es la información de control de acceso, que concede permisos a los usuarios que acceden al sistema.

En primer lugar encontramos la siguiente estructura:

```
access to * attr=userPassword
        by anonymous auth
        by * auth
```

Se especifica que el acceso al atributo “userPassword” de todas las entradas que lo contengan para los usuarios anónimos está restringido al permiso de autentificación de manera que no pueden realizar lecturas ni comparaciones sobre él. Para aquellos usuarios que accedan al sistema convenientemente autentificados el permiso será el mismo.

Este control no se aplica al administrador del sistema, que tendrá todos los permisos sobre este atributo.

La finalidad de este control es que información tan sensible como las claves de los usuarios contenida en los atributos “userPassword” no pueda ser consultada por ningún cliente.

El control de acceso al sistema se completa con la siguiente especificación:

```
access to *
        by anonymous read
        by dn="cn=usu2,us=usuarios,di=ciudades" search
        by * read
```

En ella se indica que los permisos de acceso a cualquier entrada son de lectura para usuarios anónimos y todos los usuarios autentificados excepto el usuario de nombre distinguido “cn=usu2,us=usuarios,di=ciudades” al cual sólo se le concede permiso de búsqueda.

A continuación aparecen en el fichero de configuración las opciones propias de la base de datos que va a usarse.

En primer lugar se indica que el tipo de base de datos que da soporte al directorio es del tipo “ldbm”.

La directiva “suffix” contiene el nombre relativo de la entrada raíz del directorio, que en este caso es “di=ciudades”.

También se indica mediante la directiva “rootdn” el nombre distinguido de una entrada que representa al administrador del sistema, el cual tiene todos los permisos sobre cualquier elemento del directorio. En este caso el administrador es “cn=Manuel,di=usuarios,di=ciudades”.

La clave de este superusuario se especifica mediante la directiva “rootpw”.

Por último se indica un directorio en el cual el servidor debe guardar los ficheros necesarios para mantener la información y los índices que debe mantener para la gestión de la base de datos “ldbm”.

Una vez que el servidor está instalado y configurado el siguiente paso es hacerlo funcionar, para lo cual se ejecuta el demonio Slapd.

Cuando se describió el funcionamiento de esta aplicación se indicaron las posibles opciones que se pueden usar. En la práctica se aconseja indicar el fichero de configuración para que no haya problemas con otros ficheros del sistema.

Normalmente el servidor funciona sobre una conexión TCP/IP normal, por defecto en el puerto 389. Para arrancar la aplicación se usa:
Salpd –f /usr/local/etc/openldap/slapd.conf.

En caso de que se quiera arrancar el servidor sobre un puerto SSL (por defecto el 636) se usa:

Slapd -H ldaps:/// –f /usr/local/etc/openldap/slapd.conf

En este caso se aconseja quitar las líneas de comentario acerca de los ficheros que contienen certificados X.509 y el almacén de claves para que el proceso de autentificación del protocolo SSL pueda llevarse a cabo.

Una vez arrancado el servidor, se encuentra preparado para comenzar a recibir peticiones.

El uso de los clientes que ofrece el software OpenLDAP para la gestión del directorio queda bajo la gestión del administrador, que los usará para monitorizar y gestionar el directorio.

La primera tarea del administrador en la gestión del sistema es la de añadir la información de directorio una vez que el servidor sea arrancado por primera vez. Para ello el método adoptado consiste en tener la información que se desea añadir correctamente estructurada en un fichero con formato LDIF y usar el comando ***“slpadadd”*** para añadir las entradas al directorio.

El administrador del sistema es también el encargado de realizar posibles modificaciones sobre el fichero de configuración tales como añadir nuevos esquemas al directorio, añadir bases de datos distintas de las existentes, cambiar los controles de acceso, etc.

8.1.1 Información contenida en el directorio.

Dentro del directorio deben distinguirse principalmente tres tipos de información dentro del sistema:

- ***Información de esquema.***
- ***Información del árbol de directorio.***
- ***Información sobre los usuarios.***

El diseño comienza con la decisión por parte del diseñador de qué tipo de información va a contener, para lo cual se decide la naturaleza de las clases objeto de las entradas del árbol que van a contener la información, los atributos que van a ser usados, cuándo van a ser usados, etc.

Una vez definidos estos elementos se analiza la información de esquema disponible y se plantea si ofrece todos los requisitos necesarios para el diseño del directorio deseado. En caso negativo se procede al diseño de un esquema propio que cubra todas las necesidades en cuanto a clases objeto y atributos planteados en el diseño.

Una vez solventado este aspecto se procede diseñar un árbol de directorio que organice todas las entradas que el diseñador desea añadir. Para ello se confecciona un fichero con formato LDIF en el que estas entradas son declaradas siguiendo las normas de codificación de este formato.

Es importante definir los usuarios que van a ser reconocidos por el sistema y destinar una serie de entradas del árbol a contener información acerca de estos usuarios tales como sus nombres, identificadores, claves, etc.

Este tipo de entradas son también codificadas en el fichero LDIF como si de entradas de información normal se tratasen.

En el caso de nuestro sistema, se han realizado pruebas con un árbol que implementa un directorio con los datos de una empresa concreta, sus sucursales, sus departamentos y sus empleados. Para este fin el esquema principal que incorpora OpenLDAP fue suficiente ya que la información de esquema usada para este directorio al tratarse de información muy básica y comúnmente usada se encontraba disponible en este esquema. Simplemente hubo que añadir las entradas al directorio de la forma anteriormente mencionada.

Durante la realización del trabajo surgió la necesidad de usar otro tipo de información, así como un directorio más amplio, para lo cual se diseñó un esquema propio que trata de implementar la información de esquema necesaria para construir un directorio cuya información sean características de ciudades del mundo clasificadas por países y estos últimos por continentes.

Para ello se definieron clases objeto tales como “continente”, “país” o “ciudad”, y atributos como “moneda”, “capital”, “habitantes”, “superficie”, etc. Dicho esquema puede consultarse en el apartado Anexos.

Una vez definido el esquema se configuró un fichero LDIF con cincuenta y dos entradas clasificadas por raíz del árbol, continentes, países y ciudades; con atributos que cumplen las reglas del esquema y que contienen valores particulares que son datos reales de la información que representan.

El proceso de diseño finaliza añadiendo en el fichero LDIF una entrada de la cual cuelgan un conjunto de entradas que representan a los usuarios del directorio. Además del administrador especificado en el fichero de configuración se añadieron tres nuevos usuarios al sistema.

El fichero LDIF que contiene toda esta información puede también consultarse en el apartado Anexos.

Una vez diseñado el directorio se procedió a incluir el esquema en el fichero de configuración (puede verse el fichero de configuración) y a añadir las entradas al directorio.

Las pruebas realizadas sobre el funcionamiento del sistema usan este directorio de diseño propio.

8.2 La plataforma CORBA.

Para el desarrollo de la parte del sistema que funciona sobre CORBA se ha seguido el método de diseño de aplicaciones CORBA que fue indicado cuando se expuso el funcionamiento de CORBA.

Para ello se ha elegido un software que implementa la norma CORBA, se ha desarrollado un servidor CORBA que permanece a la escucha de peticiones que posteriormente se encargará de transmitir al servidor LDAP, recoger sus respuestas y enviarlas al cliente. El servidor CORBA es por

tanto el proxy CORBA_LDAP que a la vez juega el papel de cliente es la arquitectura LDAP.

El cliente CORBA se encarga de realizar peticiones al servidor. El diseño del cliente se ha realizado de manera que ofrezca al usuario humano un entorno gráfico basado en botones, menús y ventanas que permitan a éste realizar con facilidad, comodidad y de manera intuitiva las consultas al servidor que desee. Hay que decir que este objetivo planteado en la fase de realización del proyecto ha sido cumplido de una manera razonablemente básica, persiguiendo en todo momento el cumplimiento de las características anteriormente mencionadas pero sin la pretensión de conseguir un cliente complejo y capaz de ofrecer todas las ventajas que quizás serian deseables en una potente plataforma de gestión. Este aspecto será desarrollado posteriormente como una línea de avance del proyecto.

El método de invocación elegido para el sistema CORBA es el de invocación estática que además de ser el más ampliamente usado el claramente el más aconsejable dentro del escenario en el que estamos trabajando, ya que se tiene un servidor perfectamente definido que debe atender en principio a múltiples clientes, los cuales están diseñados también exclusivamente para realizar peticiones a ese servidor.

Por tanto el servidor deberá arrancar, conectarse al ORB, iniciar el POA y definir un sirviente que implemente los métodos que ofrece la interfaz del proxy CORBA-LDAP, registrarse en el servicio de nombres de CORBA y permanecer a la espera de peticiones.

Por su parte el cliente una vez conectado al ORB deberá obtener del servicio de nombres la referencia al servidor y comenzar a realizar peticiones.

8.2.1 Elección del ORB.

Una vez definido el escenario deseado se procede a la elección del software concreto con el que implementar el sistema. Para el desarrollo de aplicaciones CORBA debe elegirse un software que cumpla la norma CORBA, ofrezca un ORB, un compilador de ficheros “idl”, una serie de servicios básicos como el servicio de nombres y a ser posible que sea fácil de configurar y manejar.

En apartados anteriores se han expuesto algunas de las distribuciones que implementan la norma CORBA más usadas. Prácticamente todas cumplen

los requisitos generales anteriores, pero analizando el contexto en el que se trabaja salen a relucir dos características fundamentales que determinan la decisión del software:

- Dado el carácter del trabajo sería deseable usar un software de libre distribución y gratuito.

- Es conveniente usar una distribución optimizada para aplicaciones desarrolladas en Java.

El segundo aspecto es de obligado cumplimiento, ya que viene impuesto por la decisión de usar JNDI que es un API para Java, lo cual hace que al menos la aplicación que implementa al servidor esté desarrollada en Java.

En principio esto no tiene consecuencias sobre el lenguaje de programación a usar en el cliente, sin embargo el objetivo anteriormente mencionado de conseguir una interfaz gráfica hace aconsejable su desarrollo en Java basándonos en el uso del paquete “java.awt”.

Por ello se eligió ***JACORB***. Esta distribución está desarrollada por la Universidad de Berlín, es gratuita, de libre distribución y se encuentra escrita al cien por cien en Java.

Puede conseguirse fácilmente en www.Jacorb.org.

Para la instalación de Jacorb es necesario el uso de una herramienta llamada ***“ant”.***

8.2.2 La herramienta ANT.

La herramienta ***“ant”*** es una utilidad construida en Java que extiende la funcionalidad de la herramienta “make” de Linux, corrigiendo las limitaciones que tienen este tipo de herramientas basadas en la interpretación de ordenes.

El funcionamiento de esta herramienta se basa en la interpretación de ficheros de configuración con formato ***“xml”,*** que es un formato basado en clases java en lugar de comandos basados en la interpretación de órdenes.

Esta herramienta se encuentra disponible en de manera libre y gratuita en la dirección ***http://jakarta.apache.org/builds/ant/release/v1.4.1/bin/.***

Se consigue como un fichero comprimido de unos 3 MB de tamaño.

Su instalación es muy simple, basta con descomprimir el fichero en el directorio deseado.

La distribución “ant” está compuesta por manuales explicativos tanto a nivel de configuración como de desarrollo de ficheros “xml”, ejemplos, librerías y un conjunto de utilidades entre las que se encuentra la propia aplicación “ant”.

8.2.3 JACORB.

Como se ha indicado anteriormente puede conseguirse de forma simple y gratuita desde la dirección www.Jacorb.org. Aquí se encuentran disponibles varias versiones del software.

La versión usada para el sistema es la versión más reciente de Jacorb. Es la ***versión 1.4 beta 4*** , que puede descargarse en un fichero comprimido en formato “zip” de un tamaño aproximado de unos 6 MB.

8.2.3.1 Instalación de JACORB.

La instalación de este software comienza por descomprimir el fichero en formato “zip” en el directorio elegido.

Una vez descomprimido se procede a su instalación, para la cual se necesita de la aplicación ***“ant”.***

La versión de “ant” usada es la versión 1.4.1., la cual ofrece unas buenas prestaciones. Inicialmente se usó la versión 1.2.1 pero debido a cuestiones de incompatibilidad los resultados no fueron satisfactorios**.**

“Jacorb” se compone de unas librerías que contienen las clases java que implementan los elementos de CORBA, un conjunto de aplicaciones CORBA que sirven como demostraciones, ficheros con información general, ficheros de configuración y un directorio con las aplicaciones que ofrece “Jacorb”.

Para instalar “Jacorb” es preciso definir una serie de propiedades de entorno del sistema que será las que “ant” use para construir los ficheros

por lotes que permiten la correcta ejecución de los servicios que ofrece "Jacorb". Estas propiedades de entorno del sistema son:

- **<*Jacorb_home*>.** Define el directorio en el que se ha instalado "Jacorb"

- **<*ant_home*>.** Define el directorio en el que se ha instalado "ant"

- **<*JAVA_HOME*>.** Es el ruta del directorio "jre" del sistema Java que se está usando.

- ***CLASSPATH.*** Deben añadirse al classpath del sistema además de las librerías java del sistema, las librerías de "ant" y las de "Jacorb" y otras librerías con las que se deseen reconstruir la aplicación tales como librerías de clases que implementen el protocolo SSL.

- ***PATH.*** A esta variable debe añadirse la ruta del directorio "bin" de "Jacorb" en el cual se encuentran los ficheros por lotes que permiten ejecutar las aplicaciones, la ruta del directorio "bin" de "ant" en el cual se encuentran las aplicaciones y la ruta del directorio "bin" de sistema Java empleado.

Una vez definidos estos parámetros se ejecuta el comando "ant" en el directorio de "Jacorb", el fichero con formato "xml" será leído y las aplicaciones y las librerías serán correctamente configuradas.

Existe la posibilidad de hacer esta labor sin usar la herramienta "ant". Para ello deben editarse los ficheros por lotes de las aplicaciones que van a ser usadas y sustituir las rutas ahí indicadas para las clases, utilidades java y para las librerías de "Jacorb" por las rutas correctas del sistema en el que "Jacorb" ha sido instalado.

Las herramientas que se usan principalmente y que deben ser configuradas son:

- ***Idl.*** Es el intérprete para ficheros "idl"

- ***Ns.*** Permite arrancar el servicio de nombres.

- ***Jaco.*** Sirve para ejecutar aplicaciones CORBA.

8.2.3.2 Configuración de JACORB.

Una vez instalado "Jacorb" se procede a su configuración.

La configuración de "Jacorb" se lleva a cabo mediante dos ficheros de texto:

- ***Orb.properties.*** Es sólo importante si se desean ejecutar las aplicaciones de las librerías de "Jacorb" sin hacer uso de los ficheros por lotes.

  La información en él contenida configura el ORB de "Jacorb" como el ORB que deben usar las aplicaciones Java. El fichero debe ser copiado en el directorio en el que se encuentran las librerías java.

- ***Jacorb.properties.*** Es el verdadero fichero de configuración. Contiene información sobre la cómo encontrar el servidor de nombres, la configuración del ORB, del protocolo GIOP, la dirección del adaptador de objetos, la configuración del repositorio de interfaces, de las conexiones SSL en caso de que sean usadas, la configuración del servicio de comercio, etc.

La configuración realizada en el sistema ha consistido en copiar el primer fichero en la dirección indicada para realizar pruebas sobre la ejecución de las utilidades sin usar los ficheros por lotes.

También se ha copiado el segundo fichero en el directorio del usuario con el fin de que pueda ser fácilmente encontrado por las aplicaciones.

En ese fichero se ha configurado cómo una aplicación CORBA puede encontrar el servicio de nombres mediante la línea:

ORBInitRef.NameService=corbaloc::172.16.17.98:2002/StandardNS/NameServer%2DPOA/_root

De esta manera la aplicación buscará el servicio de nombres en la máquina cuya dirección IP es 172.16.17.98 y en el puerto 2002 de dicha máquina.

También es importante configurar la propiedad que indica la dirección del adaptador de objetos para que indique la dirección y el puerto de la máquina en la que se encuentra funcionando, con el fin de que aplicaciones que no son locales puedan encontrarlo, ya que esta dirección es la que se usa para construir referencias a objetos.

Esto se lleva a cabo con la línea:

OAIAddr=172.16.17.98

De esta manera se indica que la dirección que debe usarse para construir referencias a objetos es la 172.16.17.98. El número de puerto será el usado por defecto.

El resto de parámetros de la configuración son los usados por defecto que proporciona el fichero plantilla de configuración.

En el apartado de Anexos se muestra íntegramente el fichero de configuración usado en el sistema.

8.3 El proxy CORBA-LDAP.

Este elemento del sistema es a la vez el servidor CORBA y el cliente LDAP, sirviendo de nexo de esta forma entre el cliente CORBA que desea realizar consultas al servidor de directorio y el propio servidor de directorio.

Su funcionamiento es el de un objeto CORBA que actúa como servidor, conectándose al ORB y permaneciendo a la espera de peticiones del cliente, que le pedirá que realice operaciones relacionadas con los métodos que dicho servidor ofrece en la interfaz pública que conoce el cliente.

El objeto que ofrece estos métodos, llamado conceptualmente “sirviente” en el argot de CORBA es en realidad el cliente desde el punto de vista de del protocolo LDAP, ya que contiene métodos que implementan un cliente capaz de realizar todo tipo de operaciones recogidas en el protocolo LDAP sobre la información contenida en el directorio.

Esta dualidad servidor CORBA/cliente LDAP es la que da a este elemento el carácter de proxy, es decir de elemento que sirve como nexo de unión entre dos sistemas distintos, representando en cada uno de ellos al sistema opuesto.
El funcionamiento del proxy puede esquematizarse en las siguientes etapas:

- Permanece a la espera de solicitudes por parte del cliente CORBA.

- Recibe una solicitud del cliente CORBA en la que se pide que se realice una determinada consulta al servidor de directorio.

- Con los datos suministrados por el cliente construye la correspondiente petición al servidor LDAP.

- Establece una comunicación con el servidor LDAP y le envía la petición.

- Recoge los resultados enviados por el servidor de directorio en respuesta a la operación que el proxy solicitó.

- Convierte estos resultados a datos manejables dentro del mundo CORBA.

- Envía los resultados al cliente como respuesta de la petición de éste.

8.3.1 Diseño del proxy CORBA-LDAP.

El diseño del proxy CORBA-LDAP responde en una primera aproximación a la naturaleza de servidor CORBA del proxy, ya que este proceso comienza como el desarrollo de cualquier otra aplicación servidora CORBA, es decir, definiendo la interfaz del servidor, construyendo el correspondiente fichero idl, etc.

Llegado el momento de desarrollar objetos que implementen los métodos ofrecidos en la interfaz del servidor, aparece la naturaleza LDAP del proxy, ya que para el diseño de estos objetos se usa el API JNDI, y se manejan conceptos, métodos y tipos de datos propios del protocolo LDAP, con total independencia de la plataforma CORBA en la que se encuentra funcionando.

La fase de diseño del servidor va a basarse en los siguiente puntos:

- Decidir los métodos que se van a ofrecer al cliente.
- Decidir los tipos de datos que van a manejarse.
- Implementar el fichero “idl” del interfaz del servidor.
- Compilar el fichero mediante la herramienta “idl” de “Jacorb”.
- Desarrollar el objeto CORBA servidor.
- Desarrollar el sirviente.
- Compilar todas los ficheros java generados con la herramienta “javac”.

8.3.2 Los métodos ofrecidos por el servidor CORBA.

Esta fase resulta casi inmediata, ya que si el servidor funciona como un cliente LDAP deberá ofrecer todos aquellos métodos que puedan realizarse sobre el servidor LDAP.

Además de las operaciones básicas realizadas sobre los servicios de directorio están las operaciones sobre la información de esquema, y cabe plantearse alguna operación basada en las ya existentes y que permita simplificar el funcionamiento de éstas en distintas circunstancias.

También habrá que tener en cuenta operaciones que realicen consultas al servidor sobre características soportadas por éste.

Teniendo en cuenta todo lo anterior se ha decidido ofrecer los siguientes métodos.

- ***Compare.*** Este método implementa la operación "compare" de LDAP. Para ello debe recibir:

  - El método de autentificación que va a usarse.
  - El nombre distinguido del usuario.
  - La clave del usuario.
  - El nombre distinguido de la entrada sobre la que se desea realizar la comparación.
  - El atributo y el valor que se usan en la comparación.

  El método devolverá información sobre cómo se ha desarrollado la operación y el resultado de comparación.

- ***Add.*** Implementará el método "add" de LDAP mediante el cual se añaden entradas al directorio. Debe recibir:

  - El método de autentificación que va a usarse.
  - El nombre distinguido del usuario.
  - La clave del usuario.
  - El nombre distinguido de la entrada a añadir.
  - Los atributos con los que se inicializa la entrada y sus valores.

  El método debe devolver información sobre cómo se ha desarrollado la operación y el resultado de ésta.

- ***Delete.*** Este método permite realizar la operación "delete" de LDAP para eliminar entradas. Para ello debe recibir:

  - El método de autentificación que va a usarse.
  - El nombre distinguido del usuario.
  - La clave del usuario.
  - El nombre distinguido de la entrada que se pretende eliminar.

  Debe devolver información sobre cómo se ha desarrollado la operación y el resultado de ésta.

- ***Modify_rdn.*** Equivale a la operación "modify_rdn" de LDAP que permite cambiar el nombre de las entradas. Debe recibir:

  - El método de autentificación que va a usarse.
  - El nombre distinguido del usuario.
  - La clave del usuario.
  - El nombre distinguido de la entrada a renombrar.
  - El nuevo nombre de la entrada.
  - Una indicación sobre si debe conservar el antiguo nombre como un atributo.

  Debe devolver información sobre cómo se ha desarrollado la operación y el resultado de ésta.

- ***Modify.*** Ofrece la operación "modify" de LDAP, la cual permite manipular los atributos de una entrada. Debe recibir:

  - El método de autentificación que va a usarse.
  - El nombre distinguido del usuario.
  - La clave del usuario.
  - El nombre distinguido de la entrada sobre la que se desean realizar las modificaciones.
  - Una lista de las modificaciones a realizar, y sobre qué valores y atributos deben realizarse.
  - El número de modificaciones que contiene la lista con el fin de simplificar la operación.

  Debe devolver información sobre cómo se ha desarrollado la operación y el resultado de ésta.

- ***Search.*** Equivale a la operación "search" de LDAP mediante la cual pueden realizarse búsquedas en el directorio. Debe recibir:

  - El método de autentificación que va a usarse.
  - El nombre distinguido del usuario.
  - La clave del usuario.
  - El nombre distinguido de la entrada que servirá como punto de partida de la búsqueda.
  - Un filtro de búsqueda.
  - Los identificadores de los atributos a devolver.
  - El ámbito de la búsqueda.
  - El número máximo de entradas que pueden ser devueltas.
  - El tiempo máximo que puede durar la operación de búsqueda.
  - Una indicación sobre si deben devolverse o no valores de atributos.
  - Una indicación sobre qué hacer con los alias.
  - Una indicación sobre qué hacer con las remisiones.

  Debe devolver información sobre cómo se ha desarrollado la operación, y toda la información referente a las entradas encontradas.

- ***Listarcontroles.*** Permite consultar al servidor los controles LDAP que soporta. Debe recibir:

  - El método de autentificación que va a usarse.
  - El nombre distinguido del usuario.
  - La clave del usuario.

  Debe devolver información sobre cómo se ha desarrollado la operación y los controles soportados.

- ***Listarextensiones.*** Permite consultar al servidor las operaciones extendidas LDAP que soporta. Debe recibir:

  - El método de autentificación que va a usarse.
  - El nombre distinguido del usuario.
  - La clave del usuario.

  Debe devolver información sobre cómo se ha desarrollado la operación y las extensiones soportadas.

- ***Listarsasl.*** Permite consultar los mecanismos de autentificación SASL que soporta el servidor. Para ello debe recibir:

  - El método de autentificación que va a usarse.
  - El nombre distinguido del usuario.
  - La clave del usuario.

  Debe devolver información sobre cómo se ha desarrollado la operación y los mecanismos soportados.

- ***Inf_objeto_esquema.*** Este método permite consultar la información contenida en las entradas del árbol de esquema, ya sea relativa a clases objeto, tipos de atributos, sintaxis de atributos o reglas de correspondencia. Para ello debe recibir:

  - El método de autentificación que va a usarse.
  - El nombre distinguido del usuario.
  - La clave del usuario.
  - El nombre distinguido de la entrada del esquema que desea consultarse.

  Debe devolver información sobre cómo se ha llevado a cabo la operación y la información obtenida de la entrada.

- ***Inf_clases_entrada.*** Permite consultar la información de esquema referente a una entrada del árbol de directorio. Para ello debe recibir:

  - El método de autentificación que va a usarse.
  - El nombre distinguido del usuario.
  - La clave del usuario.
  - El nombre distinguido de la entrada del árbol de directorio cuya información de esquema desea obtenerse.

  Debe devolver información sobre cómo se ha llevado a cabo la operación y la información obtenida.

- ***Listarat.*** Este método es un caso particular de la operación “search”, y permite listar el contenido de una entrada del directorio. Debe recibir:

  - El método de autentificación que va a usarse.

- El nombre distinguido del usuario.
- La clave del usuario.
- El nombre distinguido de la entrada cuyo contenido se desea obtener.
- Los atributos que deben devolverse.
- Una indicación sobre si deben devolverse o no valores de atributos.
- Una indicación sobre qué hacer con los alias.
- Una indicación sobre qué hacer con las remisiones.

Debe devolver información acerca de cómo se ha desarrollado la operación y el contenido de la entrada solicitada.

8.3.3 Tipos de datos del proxy.

Los tipos de datos manejados por el proxy pueden clasificarse en aquellos que usa la parte cliente LDAP, y que vienen definidos por JNDI, y en aquellos datos usados por la parte servidor CORBA para comunicarse con el cliente.

Debido a que los tipos de datos usados en JNDI no tienen un equivalente definido en el lenguaje IDL, surge la necesidad de realizar un mapeado entre los datos JNDI y los tipos que el lenguaje IDL soporta normalmente.

- Las cadenas de caracteres en Java son mapeadas en el tipo “string” de IDL.

- El tipo de dato entero de Java va a ser mapeado en el tipo “short” de IDL.

- El tipo tabla de cadenas de caracteres de Java va a ser mapeado en un tipo IDL definido por el diseñador de nombre “tabla”, que es una secuencia de tamaño indefinido de cadenas de caracteres.

- Surge la necesidad para determinadas funciones de utilizar una tabla de cadenas de caracteres en Java con dos elementos. Este tipo resulta muy útil durante el proceso de desarrollo del sistema, ya que permite un ahorro de recursos con respecto al tipo de dato anterior. Su equivalente en IDL es un tipo de dato creado por el diseñador de nombre “nota”, y no es más que una secuencia de dos cadenas de caracteres.

- En determinadas funciones es conveniente el uso de tablas bidimensionales de cadenas de caracteres en Java. Son las llamadas matrices, que son implementadas en IDL mediante un tipo definido por el diseñador que se basa en los tipos creados anteriormente. Dicho tipo de nombre “matriz” se construye como una secuencia indefinida del tipo de dato “tabla”.

- También es necesario definir un tipo en IDL que implemente una excepción de carácter general en Java, de manera que posibles excepciones lanzadas en el servidor puedan ser capturadas para la construcción de una excepción de usuario que contenga la información proporcionada por la original pero que pueda ser transmitida como un dato CORBA hacia el cliente, ya que en caso contrario, el cliente recibirá una indicación de que se ha producido una excepción en el servidor pero desconocerá cualquier tipo de característica referente a ella. Para ello se crea la excepción de usuario “ExGeneral” en IDL, que contiene un campo del tipo “nota” en cuyas cadenas se guarda la información básica de la excepción producida.

- El problema de las excepciones no sólo se reduce a la excepción más general de Java, sino que también se hace patente la necesidad de capturar las excepciones propias de JNDI, es decir, las “NamingException”. La problemática es exactamente la misma que en el caso anterior. En esta ocasión se define una excepción de usuario en IDL de nombre “ExNombrado” que contiene un campo del tipo “nota” en cuyas cadenas se almacena la información que proporciona la excepción.

- Cuando el proxy recibe del servidor LDAP la información de una búsqueda, la información resultante viene contenida en un tipo de dato llamado “NamingEnumeration”, que es una secuencia de objetos del tipo “SearchResult” donde cada uno de ellos representa una entrada del directorio.

  Como se vio este tipo es bastante complejo, con lo cual su implementación en IDL no es nada trivial, y queda a criterio del diseñador la búsqueda de un proceso artificial que consiga representar la información de este tipo en IDL.

  El criterio adoptado es quizás el más natural posible, ya que los tipos “Attribute”, “Attributes”,etc, contenidos en el objeto “SearchResult” son en el fondo un envoltorio para la información que

verdaderamente mantienen, y ésta no es más que información del tipo cadena de caracteres que es el tipo usado para almacenar la información en el directorio.

Por ello cuando el proxy recibe este tipo de dato, extrae toda la información de los atributos, y construye una tabla de cadenas de caracteres en la cual almacena toda esta información siguiendo un cierto orden establecido y unas marcas que permitan distinguir una entrada de otra en la tabla. Esta tabla se envía de vuelta al cliente como respuesta a su petición. Por tanto el tipo de dato "SearchResult" de JNDI se mapea en IDL en una tabla de cadenas de caracteres, es decir en un objeto del tipo "tabla".

8.3.4 La interfaz del servidor CORBA.

Una vez aclarados y fijados los objetivos que se persiguen en el servidor CORBA, se procede a construir su interfaz en lenguaje IDL.

Esta interfaz debe contener todos los métodos y tipos de datos anteriormente expuestos, y además se ha añadido una envoltura del tipo "interface" en IDL para encapsular todos los métodos y tipos de datos.

Con todo ello resulta el fichero de nombre ***"corbaldap.idl"*** que puede consultarse en el apartado de Planos.

8.3.4.1 Interpretación del fichero "CORBALDAP.IDL".

El siguiente paso en el diseño de aplicaciones CORBA es generar todas las clases java necesarias que implementen la interfaz del servidor. Para ello se usa la herramienta de nombre ***"idl"*** que ofrece "Jacorb".

Esta herramienta recibe como parámetro el fichero con en formato "idl" que se desea interpretar. En este caso debería ejecutarse:

Idl corbaldap.idl

Este comando genera los siguientes objetos java que deberán ser compilados posteriormente:

- ***_corbaldapStub.java.*** Crea los cabos del servidor.

- ***Corbaldap.java.*** Implementa el interface “corbaldap”.

- ***CorbaldapHelper.java.*** Contiene métodos de apoyo para la clase “corbaldap”.

- ***CorbaldapHolder.java.*** Contiene métodos que ayudan a manejar los tipos de datos definidos.

- ***CorbaldapOperations.java.*** Contiene las definiciones de los métodos ofrecidos por el servidor.

- ***CorbaldapPoa.java.*** Contiene los métodos necesarios para implementar el adaptador de objetos.

- ***CorbaldapPoaTie.java.*** Junto con la clase anterior implementa el POA.

- ***El directorio corbaldapPackage.*** En esta carpeta se generan las clases que implementan los tipos de datos definidos en la interfaz.

    - ***ExGeneral.java.*** Implementa la excepción de usuario “ExGeneral”.
    - ***ExGeneralHolder.java.*** Contiene métodos que permiten el manejo de los datos que usa la excepción.
    - ***ExGeneralHelper.java.*** Contiene métodos que de apoyo para la excepción.
    - ***ExNombrado.java.*** Implementa la excepción de usuario “ExNombrado”.
    - ***ExNombradoHolder.java.*** Contiene métodos que permiten el manejo de los datos que usa la excepción.
    - ***ExNombradoHelper.java.*** Contiene métodos de apoyo para la excepción.
    - ***MatrizHelper.java.*** Contiene funciones de apoyo para el tipo “Matriz”.
    - ***MatrizHolder.java.*** Contiene métodos que permiten manejar el tipo “Matriz”.
    - ***NotaHelper.java.*** Contiene funciones de apoyo para el tipo “nota”.
    - ***NotaHolder.java.*** Contiene funciones de apoyo para el tipo “nota”.
    - ***TablaHelper.java.*** Contiene funciones de apoyo para el tipo “tabla”.

- ***TablaHolder.java.*** Contiene funciones de apoyo para el tipo "nota".

8.3.5 Desarrollo del servidor CORBA.

El código que implementa el servidor CORBA se encuentra en un fichero de nombre ***"clservidor.java"*** que debe encontrarse junto con el resto de fichero java que componen el sistema.

Este objeto debe recibir como parámetro de la línea de comando la URL LDAP en la que podrá localizar el servidor LDAP. Si se intenta arrancar sin dar ningún parámetro en la línea de comando el proceso será abortado y mostrará el correspondiente mensaje de error.

En el desarrollo del servidor deben cubrirse por este orden las siguientes tareas:

- Definir un objeto de la clase ORB.
- Localizar el POA.
- Activar el POA.
- Instanciar un objeto sirviente que implemente los métodos ofrecidos por el servidor.
- Pasar la URL LDAP al sirviente.
- Unir el sirviente al ORB.
- Localizar el servicio de nombres.
- Registrar el servidor en el servicio de nombre. Se usará como identificador el alias "pcorbaldap".
- Permanecer a la espera de peticiones.

Con todo ello se desarrolla el código del servidor que puede consultarse en el libro "Código de las aplicaciones" que se adjunta con el proyecto.

8.3.5.1 Desarrollo del sirviente.

El sirviente es un objeto que contiene el código que implementa los métodos que el servidor CORBA define en su interfaz.

Este objeto es instanciado durante el proceso de arranque del servidor, quedando preparado para recibir y atender cuantas peticiones le sean enviadas.

Este elemento es el núcleo del proxy CORBA-LDAP, ya que es aquí donde físicamente se realiza el proceso de conversión de datos CORBA a datos LDAP, se establece la conexión con el servidor, se realiza la operación pertinente, se reciben los resultados, se convierten a formato CORBA y se envían de vuelta al cliente.

Para ello se han desarrollado los métodos de la interfaz usando Java y su API JNDI, resultando un objeto que por naturaleza de sirviente CORBA debe extender la clase "corbaldapPOA". Esta clase consta de un conjunto de variables propias de la clase y un conjunto de métodos que implementan la interfaz

El código del sirviente CORBA se encuentra en un fichero de nombre ***"corbaldapImpl.java"*** que se encuentra junto con el resto de clases que componen el sistema y que puede ser consultado en el libro "Código de las aplicaciones" que se adjunta con el proyecto.

Dicho código ha sido desarrollado con comentarios que en todo momento explican al lector la operación que se está realizando paso por paso, siempre y cuando no se trate de pasos triviales (la minoría de los casos). Con esto se consigue cumplir con el estilo recomendado de construir un código que sea autoexplicativo.

A continuación se pasa a describir el diseño conceptual de este objeto:

Se comienza importando las clases necesarias que van a ser usadas en la aplicación.

Se define una variable del tipo cadena de caracteres que contendrá la URL LDAP del servidor, la cual será inicializada por el propio servidor con la información leída de la línea de comandos.

A continuación se exponen los métodos de la clase:

- ***Compare.*** Implementa el método "compare" de LDAP. Comienza definiendo los datos necesarios para llevar a cabo la operación y definiendo las propiedades de entorno. Se crea el contexto inicial, y se llama al método "search" sobre él para realizar la comparación. Se leen los resultados.

  Se rellena la tabla de resultados que contendrá una primera cadena indicando que la operación ha sido correctamente realizada, y una segunda que indicará si la comparación ha resultado positiva o negativa.

  A continuación se cierra la conexión y se devuelven los resultados. Durante todo el proceso se permanece a la escucha de posibles excepciones, en cuyo caso se construirá la excepción CORBA correspondiente y será lanzada.

  Finalmente se devuelven los resultados.

- ***Add.*** Implementa el método "add" de LDAP. Se inicia definiendo las variables necesarias para la operación e inicializando las propiedades de entorno. Se lee la matriz de entrada en cuyas columnas se encuentran los atributos con los que se inicializa la entrada, y en cuyas filas aparecen los valores particulares de dichos atributos. El método deja de leer de una fila o de una columna cuando encuentra una cadena vacía.

  Con esa información construye los correspondientes objetos del tipo "Attribute" y el correspondiente objeto del tipo "Attributes".

  Crea el contexto inicial, realiza la operación llamando sobre dicho contexto al método "createSubcontext" y rellena una tabla de dos elementos con los resultados. El primero indica que la operación ha tenido éxito y el segundo que la entrada ha sido añadida. Finalmente se cierra la conexión y se devuelven los resultados.

  Durante todo el proceso se permanece a la escucha de posibles excepciones, en cuyo caso se construirá la excepción CORBA correspondiente y será lanzada.

- ***Delete.*** Implementa la operación "delete" de LDAP. Para ello define las variables necesarias, inicializa las propiedades de entorno, crea el contexto inicial y realiza la solicitud de borrado pasando el nombre de la entrada que se desea eliminar al método "destroySubcontext" llamado sobre dicho contexto.

  Se rellena una tabla con dos elementos en el que el primero indica que la operación ha tenido éxito y el segundo que la entrada ha sido eliminada. A continuación se cierra la conexión y se devuelven los resultados.

  Durante todo el proceso se permanece a la escucha de posibles excepciones, en cuyo caso se construirá la excepción CORBA correspondiente y será lanzada.

- ***Modify_rdn.*** Implementa la operación "modify_rdn" de LDAP. Para ello define las variables necesarias para la operación e inicializa las propiedades de entorno. A continuación llama al método "rename" sobre dicho contexto, pasándole el nombre de la entrada y el nuevo nombre.

  Se rellena una tabla de dos elementos con los resultados en el que el primero indica que la operación se ha realizado correctamente y el segundo que el nombre ha sido modificado.

  Para terminar se cierra la conexión y se devuelven los resultados. Durante todo el proceso se permanece a la escucha de posibles excepciones, en cuyo caso se construirá la excepción CORBA correspondiente y será lanzada.

- ***Mofify.*** Implementa la operación "modify" de LDAP. El proceso comienza definiendo las variables necesarias para la operación. Se lee el número de modificaciones a realizar y a continuación se lee la tabla que el cliente ha enviado codificando las modificaciones. El formato de esta tabla se ha diseñado para que cada modificación se codifique mediante tres cadenas de caracteres consecutivas. La primera contiene el identificador del atributo que se desea modificar; la segunda un código que identifica el tipo de operación (el carácter "a" significa añadir, "b" borrar y "c" reemplazar); por último la tercera contiene el valor particular del atributo que va a usarse en la modificación de este último. Con toda esta información se crea una tabla cuyos elementos son del tipo "ModificationItem", y codifica la modificación.

Seguidamente se definen las propiedades de entorno, se crea el contexto inicial y se llama sobre él al método “modifyAttributes”, pasándole el nombre de la entrada que se desea modificar y la lista de modificaciones anteriormente definida.

Por último se cierra la conexión y se devuelve como resultado una tabla de dos cadenas de caracteres en la que la primera indica que la operación se ha llevado a cabo y la segunda que las modificaciones han sido realizadas.

Durante todo el proceso se permanece a la escucha de posibles excepciones, en cuyo caso se construirá la excepción CORBA correspondiente y será lanzada.

- ***Search.*** Este método implementa la operación “search” de LDAP. Para ello se recibe del cliente una serie de datos que determinarán las condiciones de la búsqueda y cómo ésta debe realizarse. Para ello se definen variables necesarias para llevar a cabo la operación, entre las cuales se encuentra una tabla de cadenas de caracteres razonablemente grande en la que será registrada toda la información de las entradas encontradas durante la búsqueda. (Se ha elegido un tamaño de 2000, que en principio es suficiente para soportar una media de unas cien entradas).

  Se define una variable del tipo “SearchControl” en la que se recogen todos los controles de búsqueda que han sido especificados en la llamada al método y se crea el contexto inicial.

  Llegado este punto se distingue qué hacer con las remisiones que se encuentren ya que según el tratamiento que se les dé el método deberá comportarse de una manera u otra:

  - Si se desea seguir o ignorar las remisiones, se llama al método “search” sobre el contexto inicial. Se inspecciona el resultado devuelto y si no contiene entradas se guarda en la propia tabla de entradas un mensaje indicando que la operación ha sido realizada, otro indicando el comienzo de una nueva entrada (se ha adoptado como identificador de comienzo de nueva entrada la cadena “<>”), y finalmente un mensaje indicando que no se encontraron entradas.

    En caso de que se reciban entradas, éstas se codifican en la tabla mediante una cadena que contiene el identificador de

nueva entrada, seguido de un mensaje que indica el número de la entrada dentro de las que se han encontrado. A continuación el nombre relativo distinguido de la entrada referente al contexto que sirvió de base para la búsqueda, y por último se leen los identificadores de los atributos y sus correspondientes valores y se almacenan secuencialmente en la tabla.

- Si se decidió capturar las remisiones el proceso es parecido estando la principal diferencia en que se permanece a la escucha de excepciones que informan que se ha encontrado una remisión; en este caso la remisión es apuntada en la tabla, se comprueba si existen más remisiones y se apunta al siguiente contexto.

Finalmente en ambos casos se cierra la conexión y se devuelve la tabla de resultado con todos los elementos sobrantes inicializados mediante cadenas vacías, para que puedan ser transmitidos a través del ORB.

Durante todo el proceso se permanece a la escucha de posibles excepciones, en cuyo caso se construirá la excepción CORBA correspondiente y será lanzada.

- ***Listarcontroles.*** Permite consultar al servidor los controles LDAP que soporta, para lo cual se definen las variables necesarias, entre las que se encuentra una tabla de treinta elementos en la que se almacenarán los resultados. Se inicializan las propiedades de entorno, se crea el contexto inicial y se llama sobre él el método "getAttributes" con la URL del servidor y solicitando los controles soportados.

  El resultado es leído y codificado en la tabla de resultados, en la cual se incluyen además mensajes explicativos sobre la operación y de presentación de los resultados.

  Finalmente se cierra la conexión, se inicializan los elementos sobrantes de la tabla con cadenas vacías y se devuelve dicha tabla como resultado.

  Durante todo el proceso se permanece a la escucha de posibles excepciones, en cuyo caso se construirá la excepción CORBA correspondiente y será lanzada.

- ***Listarextensiones.*** Permite consultar al servidor las extensiones LDAP que soporta para lo cual se definen las variables necesarias, entre las que se encuentra una tabla de treinta elementos en la que se almacenarán los resultados. Se inicializan las propiedades de entorno, se crea el contexto inicial y se llama sobre él el método "getAttributes" sobre la URL del servidor y solicitando las extensiones soportadas.

  El resultado es leído y codificado en la tabla de resultados, en la cual se incluyen además mensajes explicativos sobre la operación y de presentación de los resultados.

  Finalmente se cierra la conexión, se inicializan los elementos sobrantes de la tabla con cadenas vacías y se devuelve dicha tabla como resultado.

  Durante todo el proceso se permanece a la escucha de posibles excepciones, en cuyo caso se construirá la excepción CORBA correspondiente y será lanzada.

- ***Listarsasl.*** Permite consultar al servidor los mecanismos de autentificación SASL que soporta para lo cual se definen las variables necesarias, entre las que se encuentra una tabla de treinta elementos en la que se almacenarán los resultados. Se inicializan las propiedades de entorno, se crea el contexto inicial y se llama sobre él el método "getAttributes" sobre la URL del servidor y solicitando los métodos SASL soportados.

  El resultado es leído y codificado en la tabla de resultados, en la cual se incluyen además mensajes explicativos sobre la operación y de presentación de los resultados.

  Finalmente se cierra la conexión, se inicializan los elementos sobrantes de la tabla con cadenas vacías y se devuelve dicha tabla como resultado.

  Durante todo el proceso se permanece a la escucha de posibles excepciones, en cuyo caso se construirá la excepción CORBA correspondiente y será lanzada.

- ***Inf_objeto_esquema.*** Permite consultar información contenida en las entradas del esquema. Para ello se recibe el nombre de la entrada deseada, se crean las variables necesarias entre las que se encuentra

una tabla de cadena de caracteres de treinta elementos en la que se almacena la información resultante.

Se inicializan las propiedades de entorno, se crea el contexto inicial y se llama sobre él al método “getSchema” para conseguir un contexto que represente a la raíz del árbol de esquema. Finalmente se llama sobre este último contexto “lookup” con el nombre de la entrada deseada, y se fuerza a que el resultado sea del tipo “Dircontext”. El resultado es leído y codificado en la tabla de resultados, en la cual se incluyen además mensajes explicativos sobre la operación y de presentación de los resultados.

Finalmente se cierra la conexión, se inicializan los elementos sobrantes de la tabla con cadenas vacías y se devuelve dicha tabla como resultado.

Durante todo el proceso se permanece a la escucha de posibles excepciones, en cuyo caso se construirá la excepción CORBA correspondiente y será lanzada.

- ***Inf_clases_entrada.*** Permite consultar la información de esquema referente a las clases objeto a las que pertenece una entrada del árbol de directorio. Para ello se recibe el nombre de la entrada deseada, se crean las variables necesarias entre las que se encuentra una tabla de cadena de caracteres de treinta elementos en la que se almacena la información resultante.

  Se inicializan las propiedades de entorno, se crea el contexto inicial y se llama sobre él al método “getSchemaClassDefiniton” que consulta en el esquema toda la información referente a las clases objeto a las que pertenece la entrada.

  El resultado es leído y codificado en la tabla de resultados, en la cual se incluyen además mensajes explicativos sobre la operación y de presentación de los resultados.

  Finalmente se cierra la conexión, se inicializan los elementos sobrantes de la tabla con cadenas vacías y se devuelve dicha tabla como resultado.

  Durante todo el proceso se permanece a la escucha de posibles excepciones, en cuyo caso se construirá la excepción CORBA correspondiente y será lanzada.

- ***Listarat.*** Implementa un método que permite listar el contenido de una entrada del directorio. Para ello define una serie de variables entre las que se encuentra una tabla de cadenas de caracteres con cincuenta elementos. Este método es una variante del método "search", en el cual el ámbito de búsqueda es el propio objeto, con lo cual la construcción y funcionamiento son muy similares, sólo que ahora tras inicializar las propiedades de entorno y definir el contexto inicial se llama sobre él al método "getAttributes" en lugar de "search".

  Igualmente se distinguen los casos en los que las remisiones deben ser enviadas al cliente (en cuyo caso deben ser capturadas), y aquellos en los que no.

  La tabla de resultados se rellena de manera análoga, pero en esta ocasión no es necesario el identificador de nueva entrada, ya que sólo hay una.

  Finalmente se cierra la conexión, se inicializan los elementos sobrantes de la tabla con cadenas vacías y se devuelve dicha tabla como resultado.

  Durante todo el proceso se permanece a la escucha de posibles excepciones, en cuyo caso se construirá la excepción CORBA correspondiente y será lanzada.

- ***Controles y Extensiones.*** Se pensó en desarrollar aplicaciones que usaran controles y extensiones, pero dado el escaso número de estas opciones que maneja OpenLDAP y que las clases para implementar este tipo de operaciones están disponibles en la versión 1.4 y no 1.3 de Java 2 SDK. Las aplicaciones desarrolladas no dieron el funcionamiento deseado, ya que el servidor enviaba mensajes de operación no soportada a la mayoría de las peticiones. (En particular se desarrolló una aplicación que implementaba un oyente que permanecía a la escucha de los cambios en el directorio, sin embargo al no soportar OpenLDAP el control de escucha persistente, este servicio no tuvo éxito). En cualquier caso el manejo de referencias es un control es usado con éxito por el resto de operaciones.

8.3.6 Generación de clases.

Una vez construidos todos los ficheros java necesarios, (en este punto del diseño aún falta por desarrollar el cliente) se procede a la generación de los correspondientes ficheros en formato “class”.

Para ello se usa la herramienta “javac” contenida en las distribuciones ***Java 2 SDK versión 1.3 o Java 2 SDK versión 1.4*** que son software que implementa el lenguaje de programación Java.

La versión elegida finalmente es ha sido la versión ***1.3*** , ya que aunque la versión ***1.4*** es mucho más moderna y potente, ya que incorpora el paquete “jsse” que contiene clases relacionadas con la seguridad y la implementación del protocolo SSL, ofrece clases que implementan extensiones LDAP tales como las que permiten la activación del protocolo TLS,etc, presenta problemas de incompatibilidad con “Jacorb”.

Estos problemas están relacionados con los hilos que usan las aplicaciones java.

Cuando se ha intentado arrancar aplicaciones construidas con Java 2 SDK 1.4 se ha obtenido directamente un error indicando que la máquina virtual no soportaba esos métodos.

Se optó por cambiar la herramienta “jaco” de “Jacorb” para lo cual se desactivó el uso de hilos nativos, obteniéndose buenos resultados en al parte del servidor, pero no así en la parte del cliente, ya que éste fue posteriormente desarrollado mediante el uso de métodos gráficos que trabajan con hilos.

Debido a estos problemas se descartó finalmente el uso de la versión 1.4 a favor de la versión 1.3.

El proceso de obtención de ficheros “class” debe comenzar por el paquete “corbaldapPackage”, y seguir por el resto de clases definidas, teniendo en cuenta que para este último paso es necesario haber añadido al “classpath” las clases anteriormente generadas.

8.4 Diseño del cliente CORBA.

El cliente CORBA supone el último elemento del sistema. Su función es la de ofrecer al usuario un interfaz gráfico mediante el cual éste podrá realizar peticiones al servidor LDAP a través del proxy.

El cliente ha sido desarrollado en Java, por seguir un criterio de uniformidad de lenguaje de programación en todos los objetos desarrollados, y porque ofrece el paquete “java.awt” mediante el que se puede construir con relativa facilidad interfaces gráficas básicas.

Como se mencionó al principio del proceso el cliente no pretende ser una completa plataforma de gestión, sino ofrecer de forma básica aunque con eficiencia un entorno en el que el usuario humano pueda usar el cliente CORBA con facilidad y comodidad.

El cliente CORBA debe permitir al usuario realizar todas las operaciones que el servidor CORBA ofrece en su interfaz, para lo cual debe implementar métodos para cada operación que permitan al usuario introducir los parámetros necesarios para llevar a cabo la operación, interpreten estos parámetros, realicen la correspondiente consulta y sean capaces de mostrar los resultados.

La estructuración de la aplicación cliente se ha realizado partiendo de una ventana principal compuesta por una serie de botones, menús y campos de texto que prestan ayuda al usuario. Cada vez que el usuario elige una operación aparece una nueva ventana que solicita los parámetros necesarios para llevar a cabo la operación solicitada. En dicha ventana aparecen botones que permiten cancelar la operación, realizarla, y borrar los parámetros escritos para comenzar de nuevo a construir la petición.

Cuando el usuario termina de introducir los parámetros y pulsa el botón “enviar” el cliente realiza la operación.

Si la operación se realiza de forma satisfactoria se cierra la ventana usada y se abre una nueva en la que se muestran los parámetros.

Si ocurre algún problema durante la operación la ventana permanece activa para que el usuario pueda cambiar cómodamente el parámetro que desee y realizar nuevamente la operación.

Además de las operaciones el cliente incorpora un menú de ayuda que permite al usuario consultar información que le indicará cómo debe rellenar los campos y usar el cliente para realizar peticiones.

El desarrollo del código de todos los objetos que componen la aplicación cliente y que irán siendo paulatinamente nombrado durante la próxima exposición puede consultarse en libro “Código de las aplicaciones”. Dicho código contiene comentarios que guían al lector por las operaciones realizadas para conseguir implementar el cliente.

Una vez implementadas todas las clases que componen el cliente son generados sus correspondientes ficheros “class” mediante la herramienta “javac” de Java 2 SDK 1.3.

8.4.1 Inicio del cliente.

La aplicación cliente comienza con la ejecución del fichero de nombre “cliente.class”. Esta aplicación consta de un método “main” en el que se lleva a cabo el proceso de registro del cliente en el sistema CORBA y una posterior llamada al código que construye la ventana principal de la aplicación.

El cliente realiza las siguientes operaciones para comenzar a funcionar sobre CORBA:

- Inicializa el ORB.
- Busca el servidor de nombres.
- Obtiene del servidor de nombres la referencia al objeto servidor, la cual encuentra fácilmente ya que conoce el alias (pcorbaldap) que identifica a dicha referencia.

Una vez realizadas estas operaciones, se genera la ventana principal de la aplicación y entra en un bucle infinito que permite mantener la aplicación activa hasta que el usuario decida salir de ella mediante el menú “salir”.

En el proceso de creación de objetos que implementan peticiones de operaciones particulares se manejan varios tipos de datos, entre los que se encuentran:

- La referencia al servidor CORBA obtenida, que es pasada a todos los objetos generados para que estos puedan acceder al servidor.

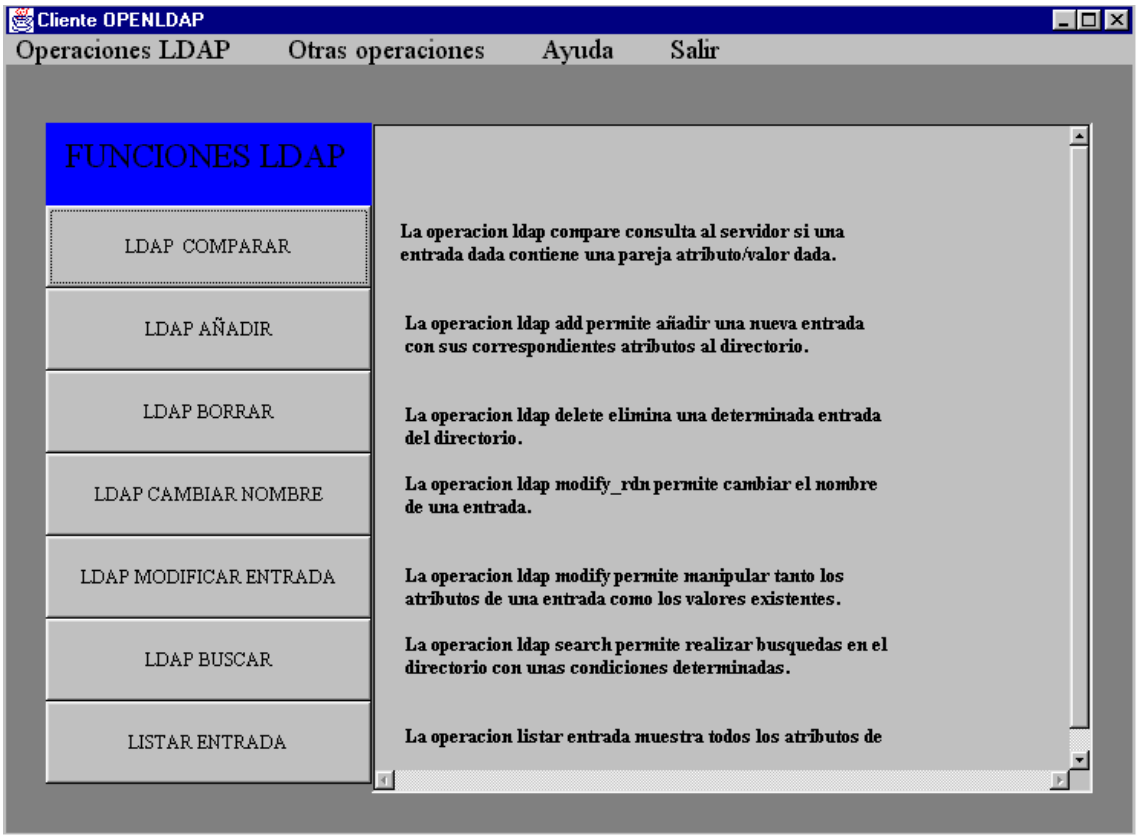
- Un objeto de la clase “autentificación”, que es una clase que contiene dos cadenas de caracteres. Las cadenas en principio están vacías, pero cada vez que el usuario realiza una operación esas cadenas son actualizadas con el identificador de usuario y la clave usada por la aplicación durante la operación.

  Hay que señalar que cuando se teclea la clave de usuario en cualquiera de los métodos usados, ésta se escribe de manera no visible para el usuario.

  Cada vez que se abre una nueva ventana la aplicación escribe por defecto la información de autentificación guardada en este objeto, ahorrando así este paso al usuario.

  Si se desea usar una nueva información de autentificación habrá que usar el botón “borrar” que eliminará la información de autentificación guardando cadenas vacías en el objeto, para que en la nueva ventana el usuario pueda establecer de nuevo el valor de estas propiedades.

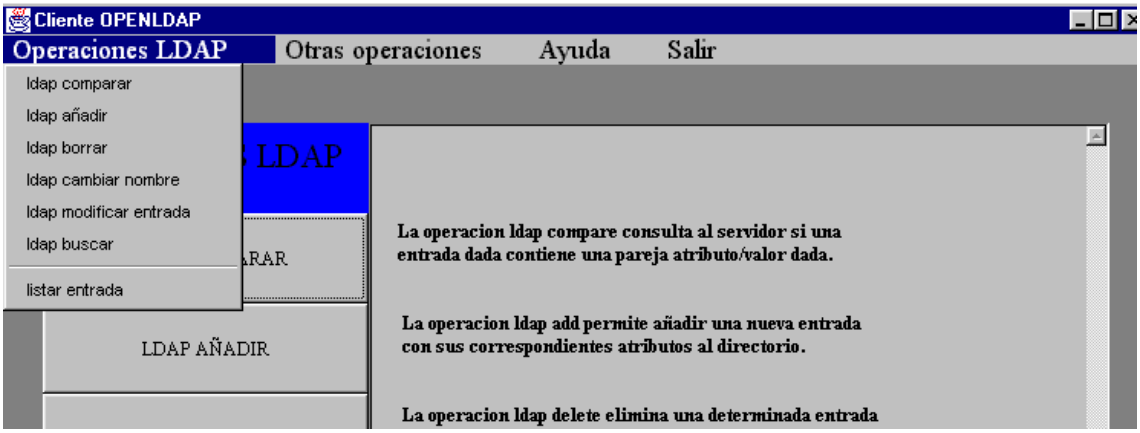
La ventana principal de la aplicación tiene el siguiente aspecto



En la ventana pueden distinguirse los botones que permiten realizar operaciones, un cuadro de texto en el que en todo momento se muestra ayuda sobre el aspecto seleccionado en el menú de ayuda.

También se observa una barra que contiene cuatro menús distintos.

- El menú “Operaciones LDAP” contiene las mismas operaciones que se representan en los botones.

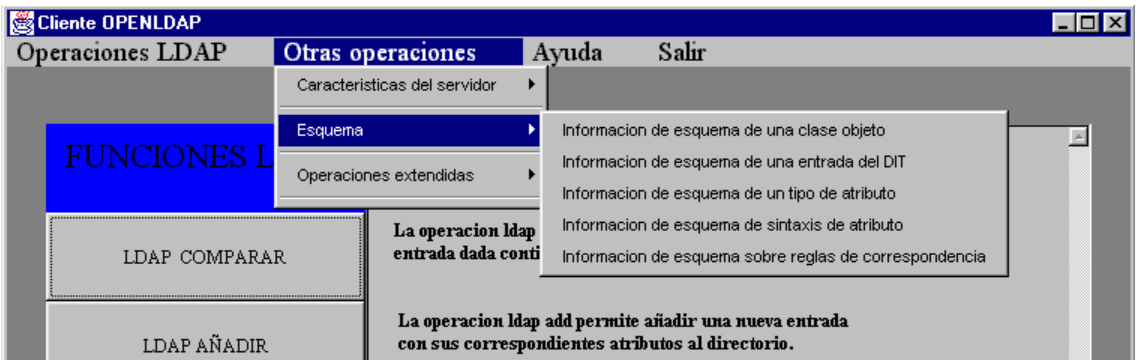


- El menú “Otras operaciones” contiene tres submenús en los cuales se ofrecen operaciones de esquema, de consulta de características del servidor, y extensiones:

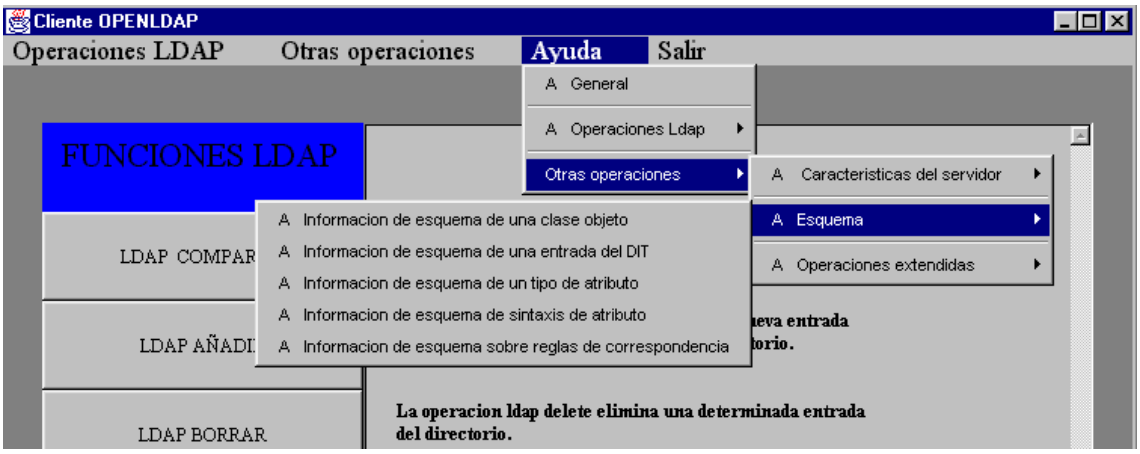
  - El submenú “Esquema” ofrece las operaciones:
    - ❖ Información de esquema de una clase objeto.
    - ❖ Información de esquema de una entrada del DIT.
    - ❖ Información de esquema de una sintaxis de atributo.
    - ❖ Información de esquema de un tipo de atributo.
    - ❖ Información de esquema sobre reglas de correspondencia.

  - El submenú “Características del servidor” ofrece:
    - ❖ Listar los controles soportados por el servidor.
    - ❖ Listar las extensiones LDAP soportadas por el servidor.
    - ❖ Listar los mecanismos de seguridad SASL soportados por el servidor.

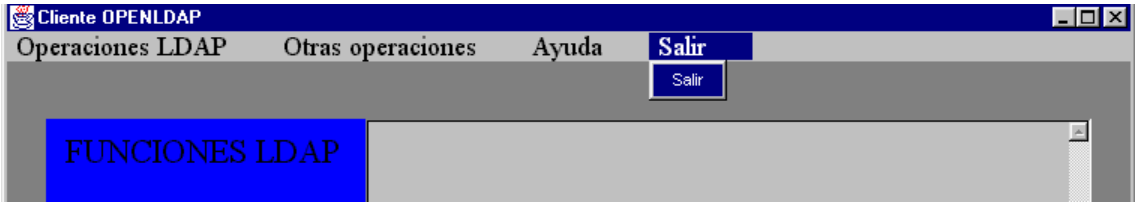
  - El submenú “Operaciones Extendidas” está vacío ya que fue diseñado para invocar operaciones extendidas que finalmente no han sido implementadas, pero deja constancia en cliente de que se ha tenido en cuenta este aspecto en el diseño.



- El menú “Ayuda” contiene submenús que permiten mostrar ayuda en el cuadro de texto acerca de las operaciones ofrecidas por el cliente.



- El menú “Salir” permite cerrar el cliente.



8.4.2 La operación COMPARE.

La operación de comparación puede invocarse desde uno de los botones de la ventana o desde el menú.

Para ello el cliente define un nuevo objeto de la clase ***“fcompare”*** construido mediante la referencia al servidor CORBA y el objeto que contiene la información de autentificación por defecto.

Este objeto muestra una ventana por pantalla en la que se especifican los parámetros de la comparación, y también contiene métodos que llevan a cabo la gestión de dicha ventana.

Este mismo objeto realiza la petición de comparación al servidor recoge los resultados y crea con ellos un objeto de la clase ***"compresult"*** que se encarga de crear una ventana en la que se muestran los resultados de la comparación.

Cuando se realiza la llamada al servidor, el objeto permanece a la escucha de posibles excepciones de nombrado, las cuales captura y construye a partir de ellas la correspondiente excepción CORBA que posteriormente se encarga de lanzar.

La interfaz gráfica que realiza esta operación se muestra a continuación:

8.4.3 La operación DELETE.

La operación de borrar entradas del directorio puede invocarse desde los botones de la ventana o bien desde el menú.

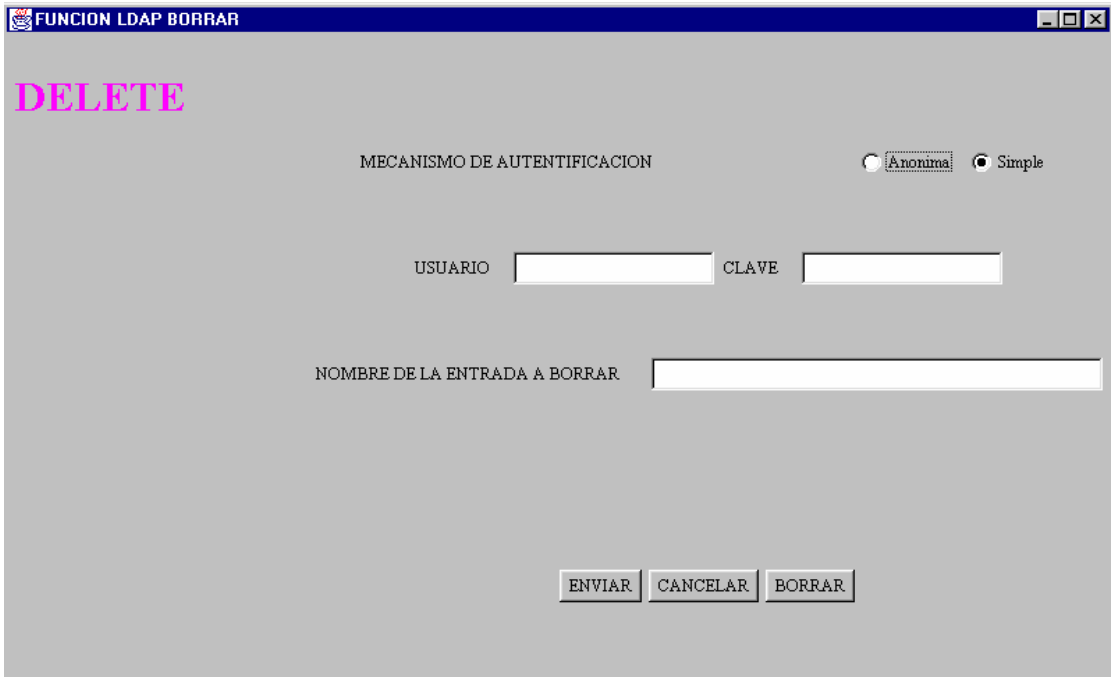
Cuando esto ocurre el cliente crea un objeto de la clase ***"fdelete"*** inicializando mediante la referencia al servidor CORBA y una referencia al objeto que contiene la información de autentificación por defecto. El objeto dibuja una ventana en la cual el usuario escribe los parámetros que se van a

usar para llevar a cabo la operación, y proporciona métodos que gestionan la ventana y realizan la petición al servidor.

Este objeto también se encarga de generar un objeto del tipo ***"deleresult"*** que muestra los resultados de la operación.

Cuando se realiza la llamada al servidor, el objeto permanece a la escucha de posibles excepciones de nombrado, las cuales captura y construye a partir de ellas la correspondiente excepción CORBA que posteriormente se encarga de lanzar.

La interfaz gráfica que realiza esta operación se muestra a continuación:



8.4.4 La operación ADD.

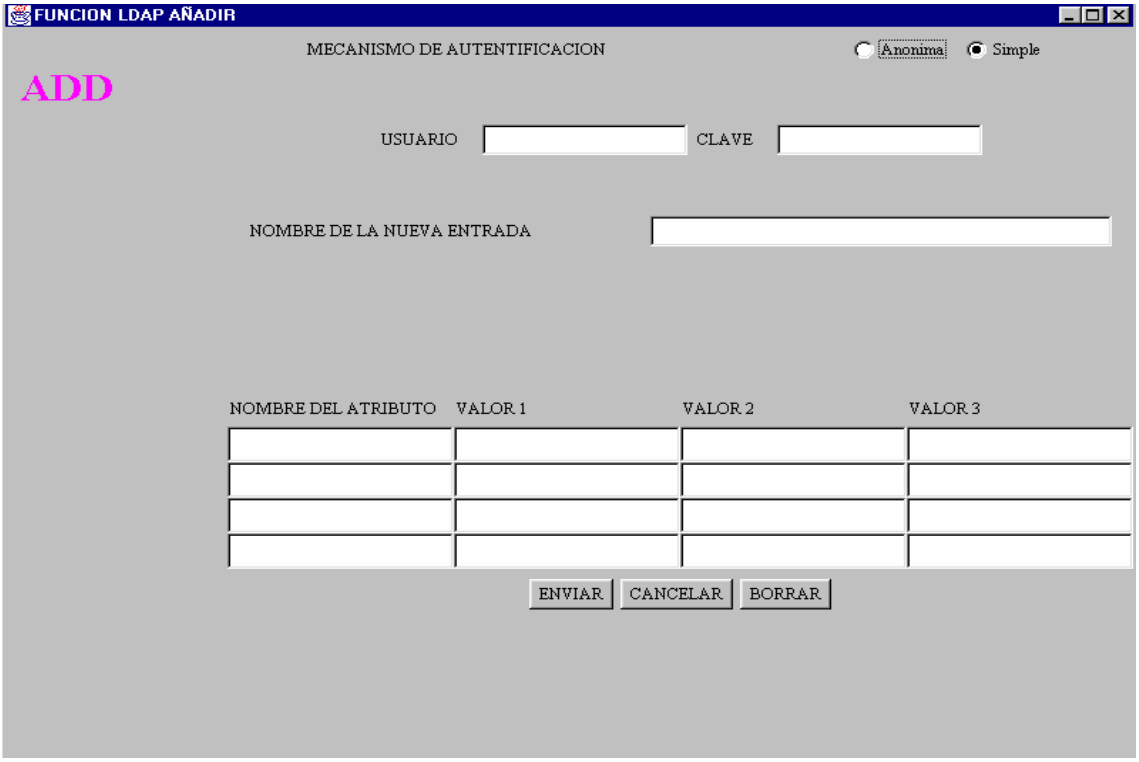
Cuando se invoca desde los botones de la ventana o desde el menú la operación de añadir entradas al directorio, el cliente genera un objeto de la clase ***"fadd"*** al cual pasa la referencia al servidor CORBA y la referencia al objeto que contiene la información de autentificación por defecto.

Este método muestra una ventana a través de la cual el usuario puede introducir la información necesaria para llevar a cabo la operación.

También contiene métodos que permiten gestionar la ventana, realizar la operación y crear un objeto de la clase ***"addresult"*** en el que se muestran los resultados de la operación.

Cuando se realiza la llamada al servidor, el objeto permanece a la escucha de posibles excepciones de nombrado, las cuales captura y construye a partir de ellas la correspondiente excepción CORBA que posteriormente se encarga de lanzar.

La interfaz gráfica que realiza esta operación se muestra a continuación:



8.4.5 La operación MODIFY.

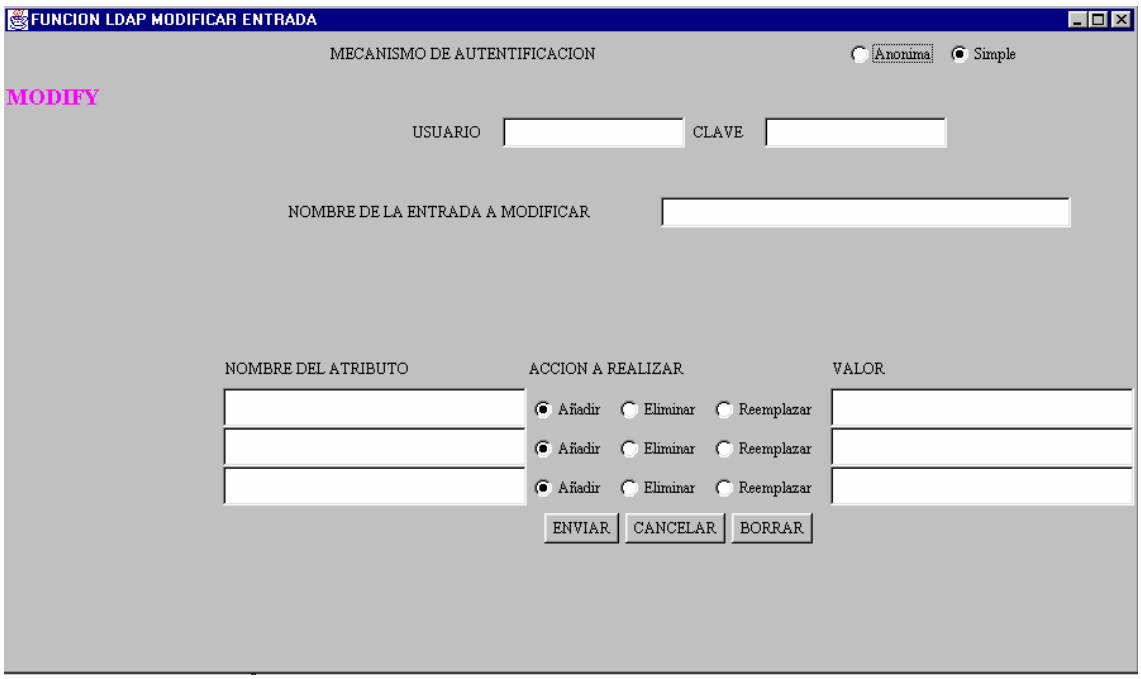
El cliente puede invocar la operación de modificación de los atributos de una entrada desde los botones de la ventana o desde el menú.

Para gestionar la petición el cliente crea un objeto de la clase ***"fmodify"***, el cual inicializa con la referencia al servidor CORBA y con una referencia al objeto que contienen la información de autentificación por defecto.

Este objeto muestra una ventana a través de la cual se reciben los parámetros necesarios para llevar a cabo la comunicación, y contiene métodos que gestionan el funcionamiento de la ventana, realizan la petición al servidor y crean un objeto de clase ***"modiresult"*** que se encarga de presentar los resultados.

Cuando se realiza la llamada al servidor, el objeto permanece a la escucha de posibles excepciones de nombrado, las cuales captura y construye a partir de ellas la correspondiente excepción CORBA que posteriormente se encarga de lanzar.

La interfaz gráfica que realiza esta operación se muestra a continuación:



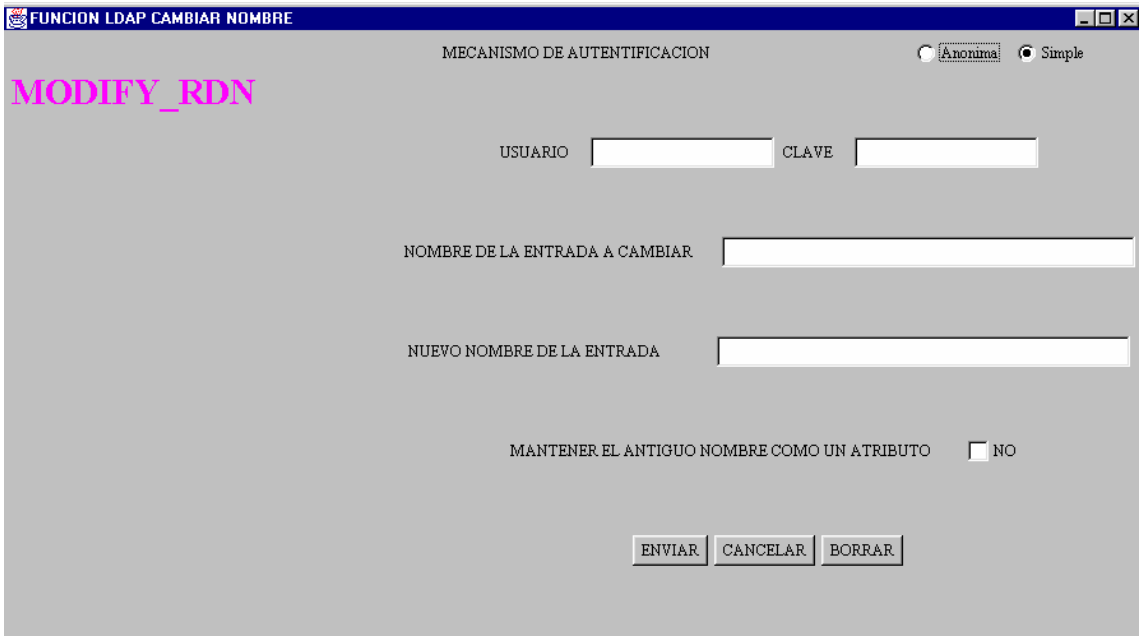
8.4.6 La operación MODIFY_RDN.

Esta operación puede invocarse tanto desde los botones de la ventana como desde el menú. Cuando esto sucede el cliente partiendo de la referencia al servidor CORBA y de la referencia al objeto que contiene la información de autentificación por defecto crea un objeto de la clase ***"fmodifyrdn"***.

Dicho objeto crea una ventana a través de la cual recibe los parámetros para llevar a cabo la operación. El objeto contiene métodos que gestionan la ventana, realizan la petición al servidor y crean un objeto de la clase ***"mrdnresult"*** que se encarga de presentar los resultados al usuario.

Cuando se realiza la llamada al servidor, el objeto permanece a la escucha de posibles excepciones de nombrado, las cuales captura y construye a partir de ellas la correspondiente excepción CORBA que posteriormente se encarga de lanzar.

La interfaz gráfica que realiza esta operación se muestra a continuación:



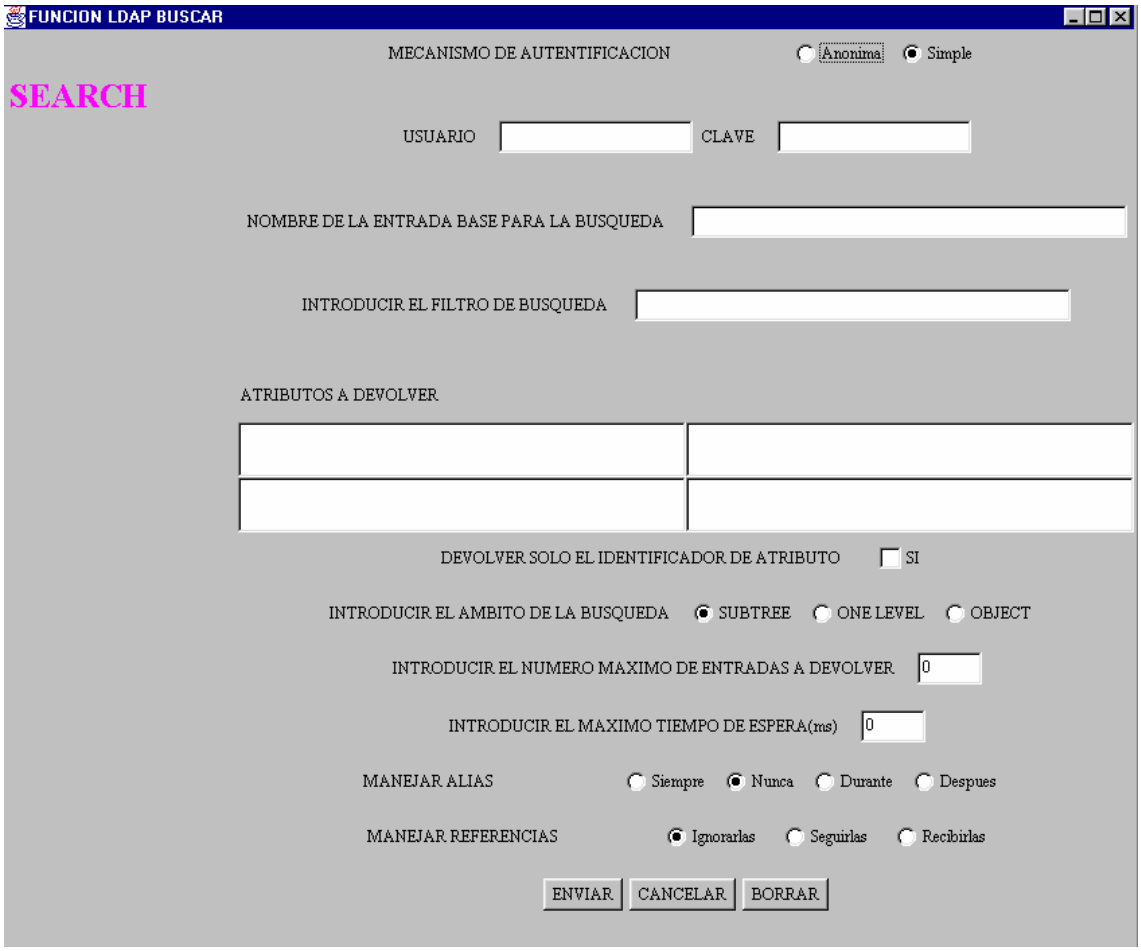
8.4.7 La operación SEARCH.

Esta operación puede invocarse desde los botones de la ventana del cliente o desde el menú. Cuando esto sucede el cliente genera un objeto de la clase ***"fsearch"*** inicializado con la referencia al servidor CORBA y al objeto que contiene la información de autentificación por defecto.

Este objeto ofrece al usuario una ventana a través de la cual éste especifica los parámetros que van a ser usados en la búsqueda. El objeto también contiene métodos que gestionan la ventana, realizan la operación y crean un objeto del tipo ***"seachresult"*** que presenta al usuario los resultados en una ventana.

Cuando se realiza la llamada al servidor, el objeto permanece a la escucha de posibles excepciones de nombrado, las cuales captura y construye a partir de ellas la correspondiente excepción CORBA que posteriormente se encarga de lanzar.

La interfaz gráfica que realiza esta operación se muestra a continuación:



8.4.8 La operación LISTAR.

Esta operación también puede invocarse desde su botón correspondiente o desde el menú. Cuando esto ocurre el cliente crea un objeto de clase ***"flistar"*** al que pasa la referencia al servidor CORBA y la referencia al objeto que contiene la información de autentificación por defecto.

Este método recibe los parámetros necesarios para realizar la operación a través de una ventana gráfica y mediante métodos de la propia clase gestiona la ventana, realiza la petición y crea un objeto de la clase ***"listresult"*** que muestra los resultados por pantalla.

Cuando se realiza la llamada al servidor, el objeto permanece a la escucha de posibles excepciones de nombrado, las cuales captura, y construye a partir de ellas la correspondiente excepción CORBA que posteriormente se encarga de lanzar.

La interfaz gráfica que realiza esta operación se muestra a continuación:

8.4.9 Consulta de características del servidor.

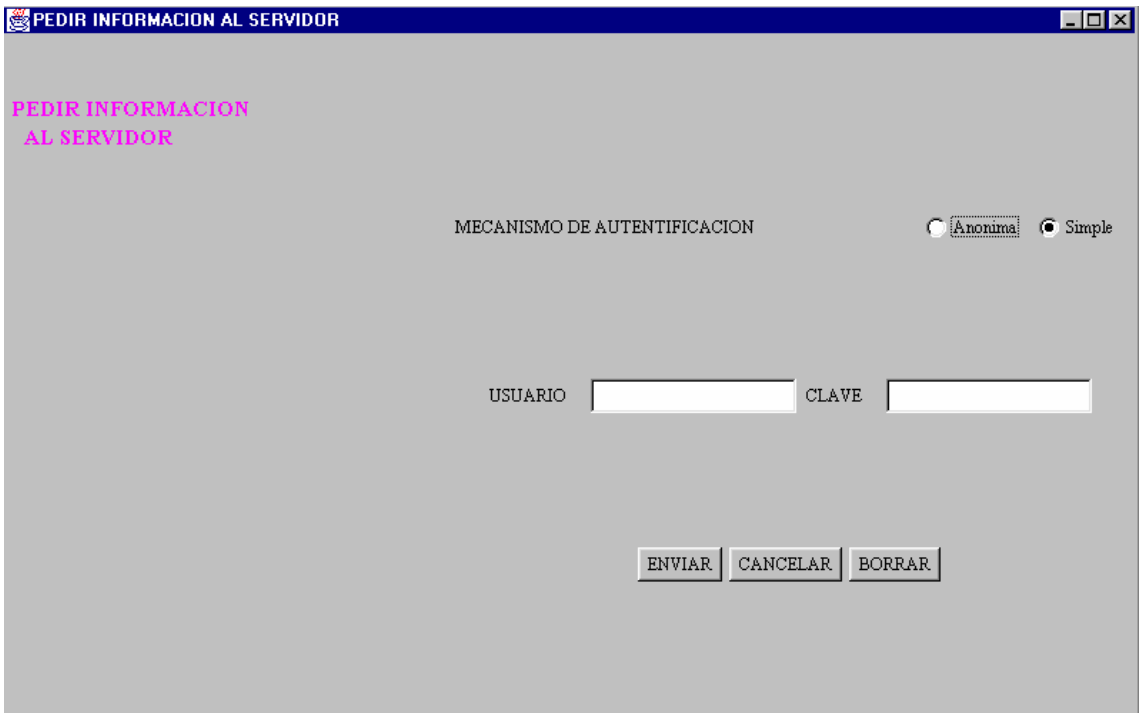
Desde el menú pueden invocarse tres operaciones distintas que realizan consultas al servidor acerca de los controles, las extensiones y los mecanismos SASL que soporta.

Cuando el usuario invoca cualquiera de estas tres operaciones el cliente genera un objeto de la clase ***"finformador"*** al que pasa la referencia al servidor CORBA y la referencia al objeto que contiene la información de autentificación por defecto.

Este objeto muestra una ventana que recoge los parámetros necesarios para llevar a cabo la operación, la realiza y crea un objeto de la clase ***"inforesult"*** que muestra los resultados al usuario.

Cuando se realiza la llamada al servidor, el objeto permanece a la escucha de posibles excepciones de nombrado, las cuales captura y construye a partir de ellas la correspondiente excepción CORBA que posteriormente se encarga de lanzar.

La interfaz gráfica que realiza esta operación se muestra a continuación:



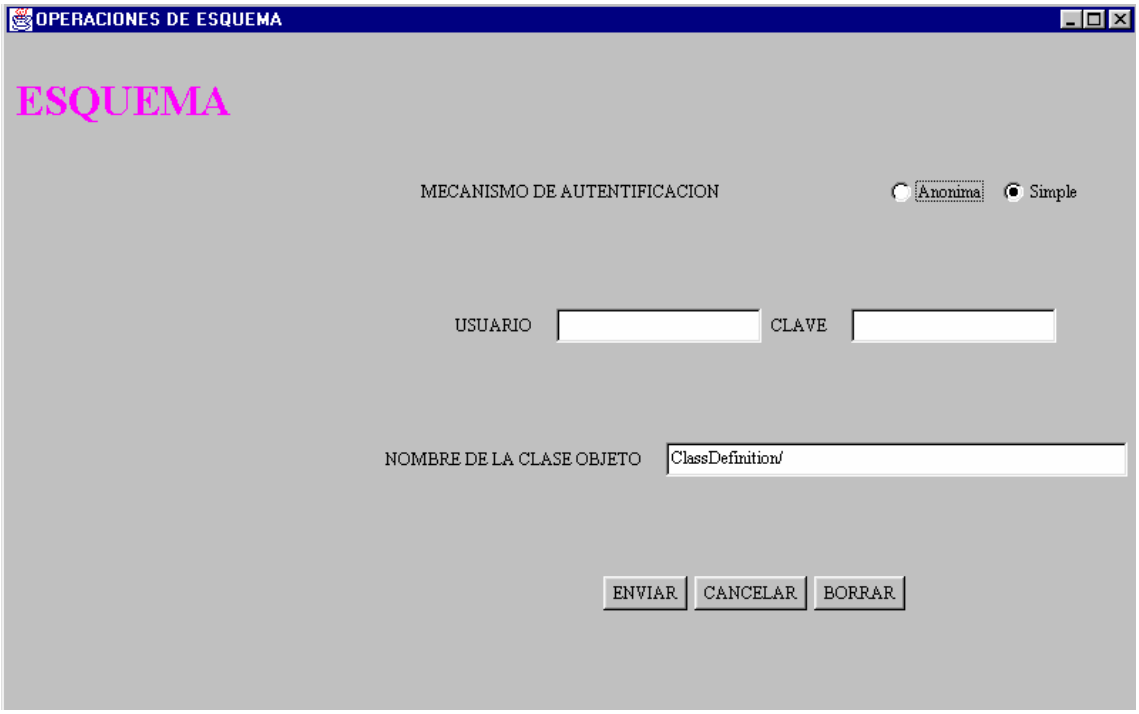
8.4.10 Operaciones de esquema.

Todas las operaciones referentes al esquema que ofrece el cliente pueden invocarse desde el menú. Para todas ellas el cliente crea un objeto de clase ***"fesquema"*** al que pasa la referencia al servidor CORBA y la referencia al objeto que contiene la información de autentificación por defecto.

Este objeto recoge los parámetros necesarios para llevar a cabo la operación mediante una ventana gráfica, realiza la operación y crea un objeto de clase ***"esquresult"*** que se encarga de mostrar los resultados.

Cuando se realiza la llamada al servidor, el objeto permanece a la escucha de posibles excepciones de nombrado, las cuales captura y construye a partir de ellas la correspondiente excepción CORBA que posteriormente se encarga de lanzar.

La interfaz gráfica que realiza esta operación se muestra a continuación (en el caso particular de solicitar una clase objeto):



8.4.11 Presentación de resultados.

Los objetos que presentan los resultados de las operaciones tienen todos un funcionamiento similar.

Reciben una tabla con los resultados que deben mostrar, y tras leerla generan una ventana en la cual presentan dichos resultados.

Este tipo de objetos son también invocados cuando se captura una excepción, y la información que muestran en este caso es la causa que ha producido la excepción.

Esta ventana permanece activa hasta que el usuario pulsa el botón aceptar, momento en el cual se oculta la ventana.

8.5 Implementación del sistema.

Para implementar el sistema completo y que comience a funcionar deben seguirse los siguientes pasos:

- Instalación y configuración del servidor OpenLDAP en un equipo que funcione bajo un entorno Linux.

- Arrancar el servidor OpenLDAP con la información de directorio deseada.

- Instalación en un segundo equipo de Java 2 SDK 1.3 y Jacorb. Este equipo será usado para albergar el servicio de nombres de CORBA.

  Configurar el fichero "Jacorb.properties" para especificar la ubicación del propio servicio de nombres y la dirección del adaptador de objetos, que debe ser la dirección IP de este equipo. Se debe añadir al "path" del sistema la ruta en la que se encuentra el directorio "bin" de "Jacorb".

- Se arranca el servicio de nombres mediante la utilidad ***"ns"*** de "Jacorb" a la cual se da como argumento en la línea de comando el nombre de un fichero de texto, que la propia aplicación crea si no existe, en el cual se guarda toda la información del servicio de nombrado. Se recomienda ejecutar: ***ns fichero_servicio_nombres***

- Instalación en un tercer equipo de Java 2 SDK 1.3 y Jacorb. Este equipo será el encargado de albergar físicamente el proxy CORBA-LDAP.

  Para ello debe configurarse el fichero "Jacorb.properties" para especificar la ubicación del servicio de nombres y la dirección del adaptador de objetos, siendo ésta la dirección del equipo que contiene el servicio de nombres.

  Se añade al "path" del sistema la ruta en la que se encuentra el directorio "bin" de "Jacorb". A continuación se instala en este equipo todas las clases que implementan el servidor CORBA, tanto las desarrolladas como las generadas por "idl".

- Se procede a arrancar el proxy mediante la herramienta "jaco" de "Jacorb", para lo que se ejecuta: ***jaco clservidor <URL LDAP>***, donde el campo <URL LDAP> contiene la ubicación del servidor de directorio.

- En un cuarto equipo se vuelve a instalar Java 2 SDK 1.3 y Jacorb. Este equipo será el encargado de albergar físicamente el cliente CORBA, para lo cual se configura "Jacorb" de la misma manera que se hizo en el servidor y se añade al "path" del sistema la ruta en la que se encuentra el directorio "bin" de "Jacorb". Se procede a la instalación de las clases que implementan el cliente CORBA, entre las que se encuentran tanto las desarrolladas por el diseñador como aquellas generadas por "idl".

- Se procede a arrancar el cliente para lo cual se ejecuta ***jaco cliente.***

A partir de este momento el sistema completo se encuentra en funcionamiento, y el usuario puede realizar mediante el cliente cualquier solicitud al servidor a través del proxy.

8.6 Empaquetado de las aplicaciones.

A continuación se indican las aplicaciones desarrolladas y cómo se organizan.

El conjunto de todos los paquetes de los que consta el proyecto se encuentran en un directorio de nombre
"diseño_e_implementación_de_un_proxy_corba_ldap".

8.6.1 Definición de la interfaz CORBA.

La definición de la interfaz CORBA se realiza mediante el fichero ***"corbaldap.idl"*** contenido en el directorio ***"corbaldap_definición"***

8.6.2 Empaquetado del proxy CORBA-LDAP.

El proxy CORBA-LDAP se encuentran implementado por en el paquete ***"corbaldap_servidor"*** que contiene las clases:

- ***Clservidor.java***. Es el corazón del servidor CORBA.
- ***CorbaldapImpl.java***. Es el núcleo del proxy CORBA_LDAP. Contiene los métodos del proxy y sirve para construir sirvientes.

8.6.3 Empaquetado del cliente.

El cliente CORBA se encuentra implementado en el paquete ***"corbaldap_cliente"*** que contiene las clases:

- addresult.java. Presenta los resultados de la operación add.
- autentificación.java. Contiene la información de autentificación por defecto.
- cliente.java. Contiene métodos que permiten conectar el cliente al ORB, desarrolla la ventana principal de la aplicación, la gestiona, y maneja el sistema de ayuda.

- compresult.java. Presenta los resultados de la operación compare.
- deleresult.java. Presenta los resultados de la operación delete.
- esqueresult.java. Presenta los resultados de las operaciones de esquema.
- fadd.java. Gestiona y realiza la operación add.
- fcompare.java. Gestiona y realiza la operación compare.
- fdelete.java. Gestiona y realiza la operación delete.
- fesquema.java. Gestiona y realiza las operaciones de consulta al esquema.
- finformador.java. Gestiona y realiza la operaciones de consulta sobre las características del servidor.
- flistar.java. Gestiona y realiza la operación listar.
- fmodify.java. Gestiona y realiza la operación modify.
- fmodifyrdn.java. Gestiona y realiza la operación modify_rdn.
- fsearch.java. Gestiona y realiza la operación search.
- inforesult.java. Presenta los resultados de las consultas sobre características del servidor.
- listresult.java. Presenta los resultados de la operació listar.
- modiresult.java. Presenta los resultados de la operació modify.
- mrdnresult.java. Presenta los resultados de la operación modify_rdn.
- searresult.java. Presenta los resultados de la operación search.

Capítulo 9: Historia del proyecto.

Este apartado prentende dar al lector una visión global sobre las etapas en las que el autor ha desarrollado el trabajo aquí expuesto, así como los problemas surgidos durante las mismas, las soluciones adoptadas, los criterios de diseño seguidos, las opciones descartadas, etc.

El proyecto parte de la idea de interconectar las arquitecturas CORBA y LDAP, para lo cual se plantea la realización de un proxy capaz de dar servicio a clientes CORBA y acceder a un servidor LDAP implementado mediante LDAP usando para ello el API JNDI de Java.

Una vez planteado el objetivo, el autor comienza la tarea de realización del proyecto desde el desconocimiento de la arquitectura CORBA, de la que conocía su existencia, su popularidad y aproximadamente para qué servía.

En este momento también era desconocida para el autor la tecnología LDAP, y por supuesto su implementación OpenLDAP, de la cual sólo sabía que estaba relacionada con el concepto de directorio, el cual no estaba perfectamente asimilado en aquel entonces.

A esto se le añadía el desconocimiento del lenguaje Java y por supuesto el uso del API JNDI, aunque sí se partía con unos conceptos básicos sobre programación orienta a objetos obtenidos de una idea global sobre el lenguaje de programación C++, pero careciendo en absoluto de experiencia en cuanto a programación con este tipo de lenguaje.

Se realiza una estimación de la realización del trabajo en unos seis meses. Durante este tiempo se debería adquirir todos los conocimientos necesarios para la realización del trabajo, tarea cuya duración se estimó en unos dos meses y medio aproximadamente.

El siguiente paso sería comenzar a realizar el proyecto, desarrollando e implementando todos los elementos del sistema, y analizando todas las posibilidades que se fuesen presentando de poder mejorar dicho sistema, y realizar pruebas sobre el mismo. Para esta tarea se estimó un tiempo de aproximadamente dos meses y medio o tres meses.

Por último se pensó en dedicar aproximadamente un mes a la redacción del proyecto.

La tarea de documentación comenzó con la búsqueda de información necesaria para comenzar a aprender acerca de los temas requeridos. Para ello se comenzó visitando una serie de páginas web recomendadas por el tutor en las cuales se podía encontrar manuales, tutoriales, normas, etc referentes a los temas buscados. La búsqueda a través de la red continuó a través de buscadores, mediante los cuales se accedía a páginas web, listas de correo, etc de personas que habían tenido alguna vez relación con alguno de los campos del proyecto y que ofrecían algo que ayudaba a enriquecer la documentación poseída. También fueron consultados libros en la biblioteca acerca de LDAP, Java y CORBA.

Este proceso de búsqueda de documentación duró aproximadamente unas dos semanas, pero se ha vuelto a utilizar en determinadas fases de la realización en las cuales han surgido problemas o posibles aspectos no previstos de antemano, para los cuales la documentación recopilada en esta fase no ofrecía información.

El proceso de estudio de la documentación comenzó con el lenguaje Java, para lo cual se usó un tutorial sacado de la web de Java. El proceso de aprendizaje no fue especialmente complejo el cuanto al funcionamiento, pero los conceptos de interfaces, paquetes y excepciones resultaron algo confusos en esta etapa, si bien fueron asimilados rápidamente cuando se comenzó a realizar pequeñas aplicaciones con el lenguaje Java.

Este proceso llevó aproximadamente unas dos semanas, dedicando una media de unas cuatro horas diarias, tiempo que fue empleado mayoritariamente en el estudio teórico del lenguaje, ya que el proceso de

experimentación con dicho lenguaje se ha llevado en paralelo con el resto de estudios teóricos realizados.

Una vez asimilado el lenguaje Java, se optó por continuar la fase de documentación con el estudio del protocolo LDAP. Esta fase resultó bastante árida y tediosa en principio, dado que la mayor parte de la documentación no se encontraba en español, y el funcionamiento del protocolo resultaba a veces poco intuitivo, debido a que se hacían referencias a elementos del protocolo cuya comprensión quizás no había sido del todo asimilada por parte del autor. Durante este proceso se usó el libro de LDAP de la empresa IBM y en ocasiones el estándar LDAP, si bien éste se planteó como elemento de consulta para seguir las explicaciones del libro anteriormente mencionado, y que a pesar de lo dicho, se recomienda su lectura ya que ofrece una visión detallada del protocolo.

Paralelamente a estas lecturas se revisaban documentos escritos por personas que alguna vez habían usado LDAP, lo cual contribuyó en gran medida más que a comprender el funcionamiento del protocolo a hacer más intuitivo su uso, y a adquirir un conocimiento cada vez más práctico de los sistemas de directorio.

Entre estos documentos se encuentran algunos del tipo “como hacer (how to)” que explicaban de manera concisa y acertada cómo instalar y usar el servidor OpenLDAP.

El tiempo que duró esta fase fue de aproximadamente unas dos semanas, con una dedicación media de unas cinco o seis horas diarias.

Terminada esa fase se creía tener un conocimiento bastante aceptable del protocolo, y se procedió a la lectura del tutorial de JNDI que ofrece Sun en su web, o mejor dicho de una versión en español de dicho tutorial que fue encontrada durante el periodo de búsqueda.

Inicialmente se comparó la traducción con el original observando que seguía de manera muy fiel la información de dicho original, con lo que fue adoptado como guía principal de JNDI, quedando la guía de JNDI de Sun como elemento consultivo, sobre todo de información gráfica.

Esta fase fue bastante provechosa, ya que la introducción al protocolo LDAP que realiza dicho manual, la descripción de los objetos ofrecidos por JNDI, su relación con LDAP y los ejemplos que se adjuntaban contribuyó

decisivamente a terminar de comprender el protocolo LDAP y adquirir más experiencia en el desarrollo de aplicaciones Java.

Este proceso duró aproximadamente un mes, con una dedicación de unas seis o siete horas diarias.

Llegado este momento prácticamente se había agotado el tiempo destinado a la documentación, y aún faltaba conocer CORBA.

El conocimiento del estándar CORBA se aplazó hasta que fuese estrictamente necesario, y se comenzó a desarrollar el sistema.

El trabajo práctico de desarrollo del código, instalación del servidor y pruebas se realizó en el laboratorio. Esta fase comenzó con la preparación del equipo que en principio se iba a usar para realizar el trabajo, ya que en la fase de diseño bastaba con ir desarrollando el sistema de manera local. Esta fase tuvo su grado de dificultad ya que el autor no se encontraba familiarizado con el uso del sistema operativo Linux, pero el proceso de aprendizaje no fue especialmente complejo, gracias a sus conocimientos de UNIX, y de Windows.

Se procedió a la instalación del servidor OpenLDAP y a comenzar a realizar pruebas con él usando un fichero LDIF suministrado con el tutorial de JNDI.

Este proceso duró aproximadamente una semana, con una dedicación media de unas seis o siete horas.

Una vez dominado OpenLDAP se instaló Java 2 SDK 1.3 y se comenzó con el proceso de diseño tanto del proxy como de un cliente capaz de usarlo (sin usar para nada CORBA). En esta fase se pusieron de manifiesto los buenos ejemplos que ofrecía el tutorial de JNDI y el sencillo funcionamiento tanto de Java como de JNDI, de manera que los avances en esta fase fueron muy rápidos y sólo se tuvieron los problemas normales que todo desarrollador de aplicaciones tiene mientras depura y enrobustece el código que ha diseñado.

En esta fase surgieron dudas sobre la utilidad de ciertas opciones que ofrecía LDAP tales como almacenamiento de objetos, etc, descartándose muchas de ellas, ya que se consideró que aunque podían ser válidas e interesantes en otro contexto se alejaban de la idea principal del proyecto; y en muchos casos se trataba de operaciones que OpenLDAP no soportaba.

También se profundizó en el funcionamiento de OpenLDAP llegando a conocer sus graves limitaciones en cuanto a implementación de protocolos de seguridad, así como su soporte de un grupo muy reducido de extensiones y controles.

En esta fase se descartó la instalación de un servidor Kérberos que implementase un método SASL de autentificación que contribuyese a fortalecer a seguridad del sistema, ya que aunque se consiguieron dos paquetes software que implementaban dicho protocolo (Heimdal y MIT Kerberos), el uso del software "Cyrus SASL" que OpenLDAP recomendaba usar para en conjunto implementar el soporte Kerberos no dio los resultados esperados quizás porque la versión ofrecida quedaba obsoleta para ser usada con la de OpenLDAP. Sin embargo se obtuvieron resultados favorables manejando las aplicaciones que implementaban Kerberos.

Por todo ello esta mejora planteada durante el proyecto fue seriamente considerada, ya que aunque se desviaba en principio de la idea original, mejoraba considerablemente el sistema, aunque a costa de aumentar también considerablemente su complejidad.

Sin embargo la idea de obtener un sistema seguro cobró cada vez más fuerza, por lo que se decidió seguir con autentificación simple, que hasta entonces se estaba usando, pero empleando el protocolo SSL, de manera que los datos circulasen encriptados por la red.

Para ello se investigó sobre el protocolo SSL y sobre los certificados que se usan en este proceso. Para poder implementarlo se hizo imprescindible cambiar la versión de Java 2 SDK que se estaba usando e instalar la versión Java 2 SDK 1.4, lo que permitió con ligeras modificaciones en el código conseguir que la transmisión se realizase sobre sockets seguros tras arrancar el servidor OpenLDAP sobre un puerto también seguro.

Durante este proceso se experimentó con un gran número de clases que ofrecía la nueva versión de Java (no así la anterior) y que implementaban controles, y extensiones; pero la simplicidad del servidor y en algunos casos la ineficiencia de estas aplicaciones, que no eran soportadas, y en otros la incompatibilidad de las mismas con el sistema, hicieron que no se llegase a implementar nada tangible en estos campos.

Paralelamente a este trabajo se llevó a cabo el estudio del estándar CORBA así como de IDL, reseñando que su funcionamiento fue más simple de entender de lo esperado, y el desarrollo de aplicaciones resultó bastante mecánico e intuitivo.

Toda esta fase duró aproximadamente unas ocho o nueve semanas, con una dedicación media de unas nueve horas diarias.

Llegado este punto se procedió con la “fase CORBA”. Para ello se instaló Jacorb no sin dificultades, ya que se tuvo problemas con la versión de Ant usada, recurriéndose en muchos casos a una instalación manual.

Fueron de gran ayuda las aplicaciones que Jacorb ofrece al cliente como demostraciones, comprobándose así lo simple y mecánico que resultaba su uso, lo cual permitió un rápido desarrollo del código restante de las aplicaciones para permitir su funcionamiento en CORBA.

Sin embargo aquí apareció un grave problema, ya que la distribución Jacorb no funciona correctamente con Java 2 SDK 1.4 ya que aparecen problemas de incompatibilidad en los hilos usados por las aplicaciones.

De esta manera no se pudo implementar un sistema CORBA sobre SSL, y más aún hubo que renunciar a la conexión sobre SSL en el tramo LDAP, ya que éste debía usar la misma versión que las aplicaciones CORBA, con lo que hubo que volver a la versión 1.3. de Java 2 SDK.

Antes de proceder a realizar pruebas se profundizó en el diseño de esquemas de LDAP y se desarrolló un esquema y un directorio que fue el usado durante las pruebas.

Se procedió a realizar pruebas y a depurar el código, apareciendo problemas sobre todo con variables de tablas, que se intentaban enviar sin ser inicializadas. Ello contribuyó a depurar el código.

Para la realización de esta fase, se emplearon aproximadamente dos semanas en terminar de montar el sistema, pero se dedicó más tiempo a intentar solventar el conflicto de versiones surgido. La dedicación durante ese tiempo fue de unas siete u ocho horas de media diaria.

Finalmente se procedió al desarrollo de la memoria en la cual se emplearon unas tres semanas con una dedicación media de siete u ocho horas diarias.

Capítulo 10: Conclusiones y líneas de avance.

Si se vuelve a leer el apartado de objetivos y alcance del proyecto se podrá comprobar que el trabajo desarrollado responde fielmente a la idea que se tenía al comenzar la realización del proyecto.

Se ha conseguido un sistema que conecta la tecnología LDAP con CORBA y que puede ser usado en múltiples escenarios según se desee sin más que actualizar la información del directorio y desarrollar una nueva aplicación CORBA que responda a las necesidades del diseñador y capaz de manejar el proxy CORBA-LDAP, elemento que permanecerá inalterado dado su elevado grado de independencia del cliente.

Las pruebas en el sistema han llevado a que el producto sea robusto y fiable, si bien como único contra hay que destacar que la seguridad del sistema es un tanto frágil ya que la información no viaja de forma segura por la red y la autentificación se realiza mediante identidad del usuario y clave en texto claro.

Sin embargo hay que estar relativamente satisfechos en este sentido, ya que el trabajo teórico realizado, y que no ha podido plasmarse en la implementación por problemas de compatibilidad, abre una solución rápida y sencilla de este problema con el uso de futuras versiones del software usado (sobre todo Jacorb), ya que dado su carácter de software de libre distribución se encuentra en constante desarrollo y cabe esperar que pronto aparezca una versión que corrija los problemas derivados de la actual, en

cuyo caso plasmar el esfuezo teórico y de diseño realizado en el campo de la seguridad será simple e inmedianto.

Las conclusiones extraidas de la realización del proyecto pueden resumirse en:

- Se ha desarrollado un sistema que interconecta CORBA con LDAP (el proxy).

- Se ha profundizado en la instalación de un servicio de directorio (OpenLDAP), así como en el diseño de un árbol de directorio con la información deseada por el diseñador, y en un conjunto de especificaciones de esquema que den soporte a dicho árbol.

- Se ha desarrollado un cliente CORBA que permite de forma sencilla realizar peticiones al proxy.

- Se han realizado pruebas sobre el sistema, aumentando la robusted de las aplicaciones y corrigiendo posibles defectos.

- El diseño proporciona al cliente un elevado grado de indendencia del proxy, con lo cual es posible rediseñar el cliente CORBA, sin tener que tocar el proxy, ni ningún otro elemento del sistema para ello.

- Se ha profundizado en la búsqueda a través de Internet de soluciones para problemas reales surgidos durante la realización del proyecto.

- El sistema presenta debilidades en materia de seguridad.

- Se ha establecido una posible solución al problema de seguridad que podría ser implantada tan pronto como el software usado haya evolucionado, y sin suponer un excesivo esfuerzo para actualizar el sistema.

- El autor ha adquirido conocimientos y experiencia en los campos estudiados, destacando especialmente el uso del lenguaje Java, que resulta crucial, dada la importancia que este lenguaje tiene hoy día, y que el propio autor ha podido comprabar durante la realización del proyecto.

- Mencionar también que se ha adquirido experiencia sobre el trabajo de laboratorio, y en el manejo de los recursos que se encuentran disponibles en él, tales como la red local, equipos, etc.

10.1 Líneas de avance.

El trabajo futuro que podría desarrollarse para mejorar las prestaciones del proyecto va íntimamente ligado en gran parte a la evolución del software de libre distribución que ha sido usado.

El sistema podría sufrir mejoras en el servidor de directorio, ya que en un futuro bastante próximo es bastante probable que la versión más actual de OpenLDAP incorpore muchos más elementos y funciones de las que ofrece el aquí usado.

Se espera que futuras versiones amplien el conjunto de controles y extensiones soportadas, y sean capaces de implementar mecanismos SASL como Kerberos por sí mismos.

Este mismo razonamiento puede hacerse con la distribución de Jacorb, siendo fundamental este aspecto, ya que permitirá ampliar fácilmente el sistema para que la transmisión de datos en la red se realice de manera segura.

Esta línea de avance podría plantearse con el uso de algún otro mecanismo o sistema de cifrado de la información distinto de los aquí descritos, pero probablemente sea mejor solución esperar a que el software de libre distribución preste un mejor servicio antes que usar algún mecanismo desfasado o de escasa aceptación.

Otra posibilidad sería realizar un estudio del sistema sobre distintos ORBs con la finalidad de probar su funcionamiento y prestaciones en otros escenarios, pero esto exigiría una modificación tanto del cliente como del proxy en cuanto al código que se encarga de manejar la parte CORBA.

Podría resultar interesante construir un cliente CORBA en C++, para lo cual habría que usar un ORB distinto de Jacorb, ya que éste se encuentra desarrollado completamente en Java.

Una continuación bastante interesante sería la construcción de una plataforma de gestión en CORBA que implemente un cliente potente, que ofreciera al usuario no sólo las funciones propias de LDAP, sino también operaciones que resulten de la combinación de las funciones ya existentes, mecanismos para navegar por el directorio en vez de realizar consultas particulares, presentación gráfica de la información, etc.

Por último podría ser interesante el estudio del API JLDAP de Novell, comparar su comportamiento y prestaciones frente a JNDI, y si resulta aconsejable rescribir el código del proxy CORBA-LDAP usando las funciones de este API pero en el marco de todo el contexto desarrollado durante el trabajo.

Por tanto las principales líneas de avance pueden resumirse en:

- Usar un servidor de directorio más potente como un servidor comercial o bien una versión más avanzada de OpenLDAP.

- Ampliar el conjunto de características soportadas, sobre todo en materia de controles y extensiones. Este aspecto va íntimamente ligado al servidor que se esté usando.

- Esperar una nueva versión de Jacorb que resuelva los problemas de compatibilidad con Java 2 SDK 1.4 y poder usar e protocolo SSL para transmisión segura en todos los tramos del sistema.

- Usar otros ORBs para probar el proxy en otros escenarios. Esta posibilidad exigiría cambiar en parte el código del proxy (y del cliente) para adaptarlo a las nuevas clases que ofrezca la nueva versión del ORB. Un ORB que ofrezca métodos en C permitiría la realización de aplicaciones en este lenguaje.

- Construcción de una plataforma de gestión que actúe como cliente CORBA y que ofrezca nuevas funciones derivadas de la combinación de las ya existentes, así como representaciones gráficas de los resultados, y un funcionamiento más cercano a una aplicación comercial.

- Estudiar el API JLDAP de Novell, compararlo con JNDI y plantearse reescribir el proxy partiendo del diseño hecho si este nuevo API resulta sensiblemente mejor, o si sencillamente el objetivo es usarlo.

11. Anexos.

ANEXOS

11.1 ANEXO 1: Esquema LDAP usado en las pruebas.

Se muestra el listado del esquema LDAP desarrollado por el autor para instalar un directorio que pretende ser original y mostar las grandes posibilidades que existen en este aspecto.
Este fichero se llama “ciudades.schema”.

```
# Este es el esquema correspondiente al directorio de ciudades
# En el se especifican las clases objetos y los tipos de atributos
# necesarios.


attributetype ( 1.1.2.1.1 NAME ('Descripcion')
        DESC 'Que contiene el directorio'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} )

attributetype ( 1.1.2.1.2 NAME 'Superficie'
        DESC 'Superficie en kilometros cuadrados'
        EQUALITY numericStringMatch
        ORDERING caseIgnoreOrderingMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )

attributetype ( 1.1.2.1.3 NAME 'Habitantes'
        DESC 'Numero de habitantes'
        EQUALITY numericStringMatch
        ORDERING caseIgnoreOrderingMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )

attributetype ( 1.1.2.1.4 NAME 'Lengua'
        DESC 'Cadena que indica el idioma'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} )

attributetype ( 1.1.2.1.5 NAME 'Capital'
        DESC 'Cadena que indica la capital del pais'
        SINGLE-VALUE
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768})
```

```
attributetype ( 1.1.2.1.6 NAME 'Moneda'
        DESC 'Cadena que indica la moneda del pais'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} )

attributetype ( 1.1.2.1.7 NAME 'NombrePropio'
        DESC 'Nombre unico de la entrada'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} )

attributetype ( 1.1.2.1.8 NAME 'Lugares'
        DESC 'Descripcion de lugares de interes. Es del tipo cadena'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} )

# El resto de atributos se usan para construir nombres distinguidos relativos

attributetype (1.1.2.1.9 NAME ( 'di' 'raizName' ) SUP name)

attributetype (1.1.2.1.10 NAME  ('co' 'continenteName')  SUP name)

attributetype (1.1.2.1.11 NAME ( 'pa' 'paisName' ) SUP name)

attributetype (1.1.2.1.12 NAME ( 'ci' 'ciudadName' ) SUP name)



# Comenzamos a describir las clases objeto

objectclass ( 1.1.2.2.1 NAME 'continente' SUP top STRUCTURAL
        DESC 'Describe la clase continente'
        MUST (co)
        MAY  (Superficie $ Habitantes $ NombrePropio))

objectclass ( 1.1.2.2.2 NAME 'pais' SUP top STRUCTURAL
        DESC 'Describe la clase pais del mundo'
        MUST ( Capital $ Moneda $ pa)
        MAY ( Superficie $ Habitantes $ Lengua $ NombrePropio))
```

```
objectclass ( 1.1.2.2.3 NAME 'ciudad' SUP top STRUCTURAL
        DESC 'Describe la clase ciudad del mundo'
        MUST ( Habitantes $ ci )
        MAY ( Lugares $ NombrePropio))

objectclass ( 1.1.2.2.4 NAME 'raiz' SUP top STRUCTURAL
        DESC 'Describe la clase ciudad del mundo'
        MUST (di)
        MAY ( Descripcion ))
```

11.2 ANEXO 2: Información de directorio usada en las pruebas.

Se muestra la información de directorio en formato LDIF que ha sido usada para llevar a cabo las pruebas.

Han sido necesarios dos directorios en algunas ocasiones, de ahí que se muestren dos ficheros LDIF distintos.

Este es el primer fichero. Su nombre es "ciudades.ldif", e implementa la información del directorio de ciudades. Este directorio ha sido realizado por el autor con datos reales, y usa la información del esquema desarrollado por el propio autor para este fin, y algunas clases del esquema básico de LDAP incluido en OpenLDAP.

```
# FICHERO LDIF DE DIRECTORIO DE CIUDADES

dn:di=ciudades
Descripcion:Directorio de ciudades
objectclass:raiz
di:ciudades

dn:co=Oceania,di=ciudades
superficie:8945724
Habitantes:28690800
objectclass:continente
co:Oceania

dn:co=America,di=ciudades
Superficie:42090655
Habitantes:711671000
objectclass:continente
co:America

dn:co=Asia,di=ciudades
Superficie:43869576
Habitantes:2981132000
objectclass:continente
co:Asia

dn:co=Europa,di=ciudades
```

Superficie:10359935
Habitantes:696063000
objectclass:continente
co:Europa

dn:co=Africa,di=ciudades
Superficie:30309677
Habitantes:630033000
objectclass:continente
co:Africa

dn:pa=Estados Unidos,co=America,di=ciudades
Superficie:9529063
Habitantes:258233
Lengua:ingles
Capital:Washington
Moneda:dolar
objectclass:pais
pa:Estados Unidos

dn:pa=Mejico,co=America,di=ciudades
Superficie:1972547
Habitantes:89955000
Lengua:espanol
Capital:Ciudad de Mejico
Moneda:peso
objectclass:pais
pa:Mejico

dn:pa=Brasil,co=America,di=ciudades
Superficie:851996
Habitantes:89955000
Lengua:portugues
Capital:Brasilia
Moneda:nuevo cruzado
objectclass:pais
pa:Brasil

dn:pa=Argentina,co=America,di=ciudades
Superficie:3761274
Habitantes:34000000
Lengua:espanol
Capital:Buenos Aires

Moneda:peso
objectclass:pais
pa:Argentina

dn:pa=China,co=Asia,di=ciudades
Superficie:9572900
Habitantes:1179467000
Lengua:chino
Capital:Pekin
Moneda:Yuan
objectclass:pais
pa:China

dn:pa=Japon,co=Asia,di=ciudades
Superficie:377750
Habitantes:124670000
Lengua:japones
Capital:Tokyo
Moneda:Yen
objectclass:pais
pa:Japon

dn:pa=Australia,co=Oceania,di=ciudades
Superficie:7682300
Habitantes:17729000
Lengua:ingles
Capital:Canberra
Moneda:dolar
objectclass:pais
pa:Australia

dn:pa=Rusia,co=Europa,di=ciudades
Superficie:17075400
Habitantes:148000000
Lengua:ruso
Capital:Moscu
Moneda:Rublo
objectclass:pais
pa:Rusia

dn:pa=Italia,co=Europa,di=ciudades
Superficie:301302
Habitantes:57235000

Lengua:italiano
Capital:Roma
Moneda:euro
objectclass:pais
pa:Italia

dn:pa=Alemania,co=Europa,di=ciudades
Superficie:356733
Habitantes:81187000
Lengua:aleman
Capital:Berlin
Moneda:euro
objectclass:pais
pa:Alemania

dn:pa=Espana,co=Europa,di=ciudades
Superficie:504782
Habitantes:39433942
Lengua:espanol
Capital:Madrid
Moneda:euro
objectclass:pais
pa:Espana

dn:pa=Francia,co=Europa,di=ciudades
Superficie:543965
Habitantes:57690000
Lengua:frances
Capital:Paris
Moneda:euro
objectclass:pais
pa:Francia

dn:pa=Nigeria,co=Africa,di=ciudades
Superficie:923768
Habitantes:91549000
Lengua:ingles
Capital:Abuja
Moneda:naira
objectclass:pais
pa:Nigeria

dn:pa=Marruecos,co=Africa,di=ciudades

Superficie:458730
Habitantes:26494000
Lengua:arabe
Capital:Rabat
Moneda:dirham
objectclass:pais
pa:Marruecos

dn:pa=Egipto,co=Africa,di=ciudades
Superficie:997739
Habitantes:57109000
Lengua:arabe
Capital:El Cairo
Moneda:libra
objectclass:pais
pa:Egipto

dn:ci=Nueva York,pa=Estados Unidos,co=America,di=ciudades
Habitantes:7346352
Lugares:Catedral de Saint Patrick
Lugares:Puente de Brooklyn
Lugares:Estatua de la Libertad
objectclass:ciudad
ci:Nueva York

dn:ci=Washington,pa=Estados Unidos,co=America,di=ciudades
Habitantes:620000
Lugares:Capitolio
Lugares:Casa Blanca
Lugares:National Gallery
objectclass:ciudad
ci:Washington

dn:ci=Ciudad de Mejico,pa=Mejico,co=America,di=ciudades
Habitantes:8831000
Lugares:Templo mayor azteca
Lugares:Jardines flotantes de Xochimilco
objectclass:ciudad
ci:Ciudad de Mejico

dn:ci=Brasilia,pa=Brasil,co=America,di=ciudades
Habitantes:1803500
Lugares:Carretera a Belo Horizonte

objectclass:ciudad
ci:Brasilia

dn:ci=Rio de Janeiro,pa=Brasil,co=America,di=ciudades
Habitantes:5603388
Lugares:Convento de San Benito
objectclass:ciudad
ci:Rios de Janeiro

dn:ci=Buenos Aires,pa=Argentina,co=America,di=ciudades
Habitantes:2960796
Lugares:Catedral de Buenos Aires
objectclass:ciudad
ci:Buenos Aires

dn:ci=Pekin,pa=China,co=Asia,di=ciudades
Habitantes:9926000
Lugares:Ciudad Tartara
Lugares:Palacio Imperial
Lugares:Ciudad China
objectclass:ciudad
ci:Pekin

dn:ci=Tokyo,pa=Japon,co=Asia,di=ciudades
Habitantes:8323699
Lugares:Palacio Imperial
objectclass:ciudad
ci:Tokyo

dn:ci=Canberra,pa=Australia,co=Oceania,di=ciudades
Habitantes:289000
Lugares:Edificios de Walter Bulegcareffon
objectclass:ciudad
ci:Canberra

dn:ci=Sydney,pa=Australia,co=Oceania,di=ciudades
Habitantes:3531000
Lugares:Instalaciones olimpicas
Lugares:Universidad
objectclass:ciudad
ci:Sydney

dn:ci=Abuja,pa=Nigeria,co=Africa,di=ciudades

Habitantes:298300
Lugares:Capital del pais
objectclass:ciudad
ci:Abuja

dn:ci=Rabat,pa=Marruecos,co=Africa,di=ciudades
Habitantes:518616
Lugares:Mercados
Lugares:Industrias textiles
objectclass:ciudad
ci:Rabat

dn:ci=El Cairo,pa=Egipto,co=Africa,di=ciudades
Habitantes:13540000
Lugares:Museo Nacional de Egipto
Lugares:Mezquitas
objectclass:ciudad
ci:El Cairo

dn:ci=Moscu,pa=Rusia,co=Europa,di=ciudades
Habitantes:8769900
Lugares:Recinto del Kremlin
Lugares:Catedral de Moscu
Lugares:Plaza Roja
objectclass:ciudad
ci:Moscu

dn:ci=San Petesburgo,pa=Rusia,co=Europa,di=ciudades
Habitantes:4456000
Lugares:Palacio de Invierno
objectclass:ciudad
ci:San Petesburgo

dn:ci=Milan,pa=Italia,co=Europa,di=ciudades
Habitantes:1464136
Lugares:Catedral gotica de Milan
Lugares:Santa Maria delle Graze
Lugares:Biblioteca Ambrosiana
objectclass:ciudad
ci:Milan

dn:ci=Roma,pa=Italia,co=Europa,di=ciudades
Habitantes:2825884

Lugares:Foro romano
Lugares:Coliseo
Lugares:Templos romanos
objectclass:ciudad
ci:Roma

dn:ci=Florencia,pa=Italia,co=Europa,di=ciudades
Habitantes:421299
Lugares:Catedral de santa Maria de las Flores
Lugares:Biblioteca
Lugares:Museo
objectclass:ciudad
ci:Florencia

dn:ci=Paris,pa=Francia,co=Europa,di=ciudades
Habitantes:2176243
Lugares:Saint Denis
Lugares:Notre Dame
Lugares:Torre Eiffel
objectclass:ciudad
ci:Paris

dn:ci=Marsella,pa=Francia,co=Europa,di=ciudades
Habitantes:874436
Lugares:Restos de edificios griegos
objectclass:ciudad
ci:Marsella

dn:ci=Berlin,pa=Alemania,co=Europa,di=ciudades
Habitantes:3400000
Lugares:Castillo de Charlottemberg
Lugares:Puerta de Brandemburgo
objectclass:ciudad
ci:Berlin

dn:ci=Munich,pa=Alemania,co=Europa,di=ciudades
Habitantes:1274716
Lugares:Ciudad olimpica
Lugares:Palacio real
objectclass:ciudad
ci:Munich

dn:ci=Madrid,pa=Espana,co=Europa,di=ciudades

Habitantes:3084673
Lugares:San Jeronimo
Lugares:Monasterio del Escorial
Lugares:Museo del Prado
objectclass:ciudad
ci:Madrid

dn:ci=Barcelona,pa=Espana,co=Europa,di=ciudades
Habitantes:1681132
Lugares:Parque Guell
Lugares:Sagrada Familia
objectclass:ciudad
ci:Barcelona

dn:ci=Sevilla,pa=Espana,co=Europa,di=ciudades
Habitantes:704857
Lugares:Catedral de Sevilla
Lugares:Giralda
Lugares:Alcazar
Lugares:Archivo de Indias
objectclass:ciudad
ci:Sevilla

dn:di=usuarios,di=ciudades
objectclass:raiz
di:usuarios

dn:cn=Manuel,di=usuarios,di=ciudades
sn:Manuel
cn:Gomez
userPassword:secret
objectclass:person

dn:cn=usu1,di=usuarios,di=ciudades
sn:Usuario
cn:Uno
userPassword:secret1
objectclass:person

dn:cn=usu2,di=usuarios,di=ciudades
sn:Usuario
cn:Dos
userPassword:secret2

objectclass:person

dn:cn=usu3,di=usuarios,di=ciudades
sn:Usuario
cn:Tres
userPassword:secret3
objectclass:person

Seguidamente se muestra un fichero de nombre “empresa.ldif” que contiene la información que se ha usado en un segundo directorio, empleado sobre todo para experimentar con remisiones. El siguiente fichero no ha sido desarrollado por el autor, y hace uso de clases definidas en el esquema básico de LDAP que se adjunta con OpenLDAP.

dn:o=miempresa.es
o:miempresa.es
objectclass:top
objectclass:organization

dn:ou=almcordoba,o=miempresa.es
ou:almcordoba
description:Almacen
postalcode:14023
telephoneNumber:957223355
street: C/Marruecos,3
l:Cordoba
objectclass:organizationalUnit

dn:ou=cordoba,o=miempresa.es
ou:cordoba
description:Sede Central Cordoba
streetAddress: C/Marruecos,1
l:Cordoba
postalcode:14023
telephoneNumber:957223344
objectclass:organizationalUnit

dn:ou=sevilla,o=miempresa.es
ou:sevilla
description:Sede Central Sevilla

streetAddress: C/Arco,25
l:Sevilla
postalcode:41023
telephoneNumber:957112233
objectclass:organizationalUnit

dn:uid=jgarcia,ou=cordoba,o=miempresa.es
description:Jefe Informatica
uid:jgarcia
cn:Jose Garcia Rua
sn:Garcia Rua
telephoneNumber:957423234
l:Cordoba
ou:cordoba
objectclass:organizationalUnit
objectclass:person
objectclass:uidObject

dn:uid=Ihuertas,ou=almcordoba,o=miempresa.es
description:Responsable Almacen
uid:Ihuertas
cn:Lolo Huertas Perez
sn:Huertas Perez
telephoneNumber:957423233
l:Cordoba
ou:cordoba
objectclass:person
objectclass:organizationalUnit
objectclass:uidObject

dn:uid=smontero,ou=sevilla,o=miempresa.es
description:Jefe Informatica
uid:smontero
cn:Santiago Montero Garcia
sn:Montero Garcia
telephoneNumber:951236235
l:Sevilla
ou:Sevilla
objectclass:person
objectclass:organizationalUnit
objectclass:uidObject

11.3 ANEXO 3: Configuración de JACORB.

A continuación se muestra el fichero de configuración de Jacor, de nombre "Jacorb.properties"

```
##
##  JacORB configuration options
##

############################################
#                                          #
#   Initial references configuration       #
#                                          #
############################################

#
# URLs where IORs are stored (used in orb.resolve_initial_service())
# DO EDIT these! (Only those that you are planning to use,
# of course ;-).
#
# The ORBInitRef references are created on ORB startup time. In the
# cases of the services themselves, this may lead to exceptions being
# displayed (because the services aren't up yet). These exceptions
# are handled properly and cause no harm!

#ORBInitRef.NameService=corbaloc::160.45.110.41:38693/NameService
#ORBInitRef.NameService=corbaloc::193.147.162.182:38693/StandardNS
/NameServer%2DPOA/_root
#ORBInitRef.NameService=file:/root/za
ORBInitRef.NameService=corbaloc::172.16.17.98:2002/StandardNS/Nam
eServer%2DPOA/_root

#ORBInitRef.NameService=file:/c:/NS_Ref
#ORBInitRef.NameService=http://www.x.y.z/~user/NS_Ref
#ORBInitRef.TradingService=http://www.x.y.z/~user/TraderRef

# JacORB-specific URLs
Jacorb.ProxyServerURL=http://www.x.y.z/~user/Appligator_Ref
```

```
####################################
#   ORB version number output      #
#                                  #
####################################

# if on, the ORB's version number is printed
# any time the ORB is initialized
Jacorb.orb.print_version=on

####################################
#                    #
#   Debug output configuration   #
#                    #
####################################

# use (java) Jacorb.util.CAD to generate an apropriate
# verbosity level
# 0 = off
# 1 = important messages and exceptions
# 2 = informational messages and exceptions
# >= 3 = debug-level output (may confuse the unaware user :-)
Jacorb.verbosity=1

# where does output go? Terminal is default
#Jacorb.logfile=LOGFILEPATH

# hexdump outgoing messages
Jacorb.debug.dump_outgoing_messages=off

# hexdump incoming messages
Jacorb.debug.dump_incoming_messages=off


#####################################################
#                                                   #
#   WARNING: The following properties should        #
#   only be edited by the expert user. They         #
#   can be left untouched in most cases!            #
#                                                   #
#####################################################
```

```
##################################
#                                #
#   Basic ORB Configuration      #
#                                #
##################################

# the GIOP minor version number to use for newly created IORs
Jacorb.giop_minor_version=2

# number of retries if connection cannot directly be established
Jacorb.retries=5

# how many msecs. do we wait between retries
Jacorb.retry_interval=500

# size of network buffers for outgoing messages
Jacorb.outbuf_size=2048

# log2 of maximum buffer size managed by the internal
# buffer manager.
#
# This is NOT the maximum buffer size that
# can be used, but just the largest size of buffers that
# will be kept and managed. This value will be added to
# an internal constant of 5, so the real value in bytes
# is 2**(5+maxManagedBufSize-1). You only need to increase this
# value if you are dealing with LOTS of LARGE data structures.
# You may decrease it to make the buffer manager release large
# buffers immediately rather than keeping them for later
# reuse.
Jacorb.maxManagedBufSize=18

# client-side timeout, set no non-zero to stop blocking
# after so many msecs.
#Jacorb.connection.client_timeout=0

# max time a server keeps a connection open if nothing happens
#Jacorb.connection.server_timeout=10000

#Jacorb.reference_caching=off

#
# The following property specifies the class which is used for
```

```
# reference caching. WeakHashtable uses WeakReferences, so entries
# get gc'ed if only the Hashtable has a reference to them. This
# is useful if you have many references to short-living non-persistent
# CORBA objects. It is only available for java 1.2 and above.
#
# On the other hand the standard Hashtable keeps the references until
# they are explicitely deleted by calling _release(). This is useful
# for persistent and long-living CORBA objects.
#
#Jacorb.hashtable_class=org.Jacorb.util.WeakHashtable
#
Jacorb.hashtable_class=java.util.Hashtable

# use GIOP 1.2 byte order markers (since CORBA 2.4-5)
Jacorb.use_bom=off

# add additional IIOP 1.0 profiles even if we are using IIOP 1.2
Jacorb.giop.add_1_0_profiles=off

##############################################
#                                            #
#       BiDirectional GIOP                   #
#                                            #
##############################################

# uncomment this initializer if you want to use BiDirectional GIOP

#org.omg.PortableInterceptor.ORBInitializerClass.bidir_init=org.Jacorb.or
b.connection.BiDirConnectionInitializer


##############################################
#                                            #
#      Proxy address in IOR                  #
#                                            #
##############################################

#
# with these two properties it is possible to
# tell the ORB what IP/port IORs should contain,
# if the ServerSockets IP/port can't be used
# (e.g. for traffic through a firewall).
#
```

```
# WARNING: this is just "dumb" replacing, so you
# have to take care of your configuration!
#

#Jacorb.ior_proxy_host=1.2.3.4
#Jacorb.ior_proxy_port=4711


############################################
#                                          #
#   The Object Adapter Internet Address    #
#                                          #
############################################

# IP address on multi-homed host (this gets encoded in
# object references). NOTE: Adresses like 127.0.0.X
# will only be accessible from the same machine!
OAIAddr=172.16.17.98
#OAPort=4711

##################################
#                                #
#   Appligator Configuration     #
#                                #
##################################
# if your applets don't need appligator, switch this off
Jacorb.use_appligator=off



##############################
#                            #
#   Default Interceptors     #
#   Please leave them in!    #
#                            #
##############################
org.omg.PortableInterceptor.ORBInitializerClass.standard_init=org.Jacorb.
orb.standardInterceptors.IORInterceptorInitializer
```

```
################################################
#                                              #
#   Implementation Repository Configuration    #
#                                              #
################################################
# Switch off to avoid contacting the ImR on every server start-up
Jacorb.use_imr=off

# if set to "on", servers that don't already have an entry on their
# first call to the imr, will get automatically registered. Otherwise,
# an UnknownServer exception is thrown.
Jacorb.imr.allow_auto_register=off

# if set to "on", the imr will try to "ping" every object reference,
# that it is going to return. If the reference is not alive, TRANSIENT
# is thrown.
Jacorb.imr.check_object_liveness=off

ORBInitRef.ImplementationRepository=http://www.x.y.z/~user/ImR_Ref

Jacorb.imr.table_file=Z:\table.dat
Jacorb.imr.backup_file=z:\backup.dat
Jacorb.imr.ior_file=/home/bwana/brose/public_html/ImR_Ref
Jacorb.imr.timeout=
Jacorb.imr.no_of_poas=
Jacorb.imr.no_of_servers=

# how many millis should the imr wait, until a connection from an
# application client is terminated. Default is 2000.
Jacorb.imr.connection_timeout=2000

# the implementation name, should be set to a different
# name in the code of persistent servers
Jacorb.implname=StandardImplName

#
# This is supposed to be a generic startup string for everything
# that calls Runtime.exec(). Might be replaced by jaco[.bat].
#
Jacorb.java_exec=java                                          -
Dorg.omg.CORBA.ORBClass=org.Jacorb.orb.ORB                     -
Dorg.omg.CORBA.ORBSingletonClass=org.Jacorb.orb.ORBSingleton
```

```
#########################
#                       #
#   SSL Configuration   #
#                       #
#########################

#
# The port number used by SSL, will be dynmically assigned
# by default
#

#OASSLPort=4711

# This interceptor must be set if programs need access to
# certificates using the CORBA Security API, SSL works also
# without this interceptor

#org.omg.PortableInterceptor.ORBInitializerClass.ForwardInit=org.Jacorb.
security.ssl.SecurityServiceInitializer


# qualified classname of access decision object
Jacorb.security.access_decision=org.Jacorb.security.level2.AccessDecision
Impl

# list of qualified classnames of principal authenticator objects,
# separated by commas (no whitespaces!). The first entry (that can
# be successfully created) will be available through the
# principal_authenticator property.
Jacorb.security.principal_authenticator=org.Jacorb.security.level2.Principal
AuthenticatorImpl

# the qualified classname of the ssl socket factory class
#Jacorb.ssl.socket_factory=org.Jacorb.security.ssl.sun_jsse.SSLSocketFact
ory
#Jacorb.ssl.socket_factory=org.Jacorb.security.ssl.iaik.SSLSocketFactory
#Jacorb.ssl.socket_factory=/root/SSLSocketFactoryImpl
Jacorb.ssl.socket_factory=SSLSocketFactoryImpl

# the qualified classname of the ssl server socket factory class
#Jacorb.ssl.server_socket_factory=org.Jacorb.security.ssl.sun_jsse.SSLSer
verSocketFactory
```

```
#Jacorb.ssl.server_socket_factory=com.sun.net.ssl.internal.ssl.SSLServerS
ocketFactoryImpl


# exchange ssl client server roles to enforce client authentication, but
# attention: this causes problems with peers that not prepared to handle
# this role change
Jacorb.security.change_ssl_roles=off

# IIOP/SSL parameters (numbers are hex values, without the leading "0x"):
# NoProtection = 1
# EstablishTrustInClient = 40
# EstablishTrustInTarget = 20
# mutual authentication = 60
# please see the programming guide for more explanation

Jacorb.security.support_ssl=off

Jacorb.security.ssl.client.supported_options=61
Jacorb.security.ssl.client.required_options=0

Jacorb.security.ssl.server.supported_options=61
Jacorb.security.ssl.server.required_options=0

# The name and location of the keystore. This may be absolute or
# relative to the home directory.
#
# NOTE (for Sun JSSE users): The "javax.net.ssl.trustStore[Password]"
# properties don't seem to take effect, so you may want to add trusted
# certificates to "normal" keystores. In this case, please set the
# property "Jacorb.security.jsse.trustees_from_ks"is to "on", so trusted
# certificates are taken from the keystore instead of a dedicated
# truststore.
Jacorb.security.keystore=.keystore
Jacorb.security.keystore_password=Jacorb


#
# IAIK specific settings
#

# files with public key certs of trusted CAs
#
```

```
# WARNING: If no CA certs are present, the IAIK chain verifier will
# accept ALL otherwise valid chains!
#
Jacorb.security.trustees=trustee1.cert,trustee2.cert

# the name of the default key alias to look up in the keystore
Jacorb.security.default_user=brose
Jacorb.security.default_password=Jacorb

# have iaiks ssl classes print debug output to stdout
Jacorb.security.iaik_debug=off

#
# Sun JSSE specific settings
#
# Use the keystore to take trusted certs from.
Jacorb.security.jsse.trustees_from_ks=off


############################
#                          #
#   POA Configuration      #
#                          #
############################

# displays a GUI monitoring tool for servers
Jacorb.poa.monitoring=off

# thread pool configuration for request processing
Jacorb.poa.thread_pool_max=20
Jacorb.poa.thread_pool_min=5

# if set, request processing threads in thePOA
# will run at this priority. If not set or invalid,
# MAX_PRIORITY will be used.
#Jacorb.poa.thread_priority=

# size of the request queue, clients will receive Corba.TRANSIENT
# exceptions if load exceeds this limit
Jacorb.poa.queue_max=100
```

```
##########################################
#                                        #
#   Trader configuration, please see     #
#   src/trading/README.PROPERTIES for    #
#   explanation                          #
#                                        #
##########################################

jtrader.util.max_threads=10
jtrader.util.min_threads=1
jtrader.util.query_timeout=5000
jtrader.impl.cache_max=100

# boolean values, e.g. true / false
#jtrader.modifiable_properties=
#jtrader.dynamic_properties=
#jtrader.proxy_offers=

jtrader.debug=false
jtrader.debug_verbosity=3

#integer values
jtrader.def_search_card=
jtrader.max_search_card=
jtrader.def_match_card=
jtrader.max_match_card=
jtrader.def_return_card=
jtrader.max_return_card=
jtrader.max_list=
jtrader.def_hop_count=
jtrader.max_hop_count=

#FollowOptions
#always=2
#if_no_local=1
#local_only=0
jtrader.def_follow_policy=
jtrader.max_follow_policy=
jtrader.max_link_follow_policy=

# any other custom properties can be added here.
# These are available through the API (call
# Jacorb.orb.Environment.getProperty())
```

11.4 ANEXO 4: El protocolo KERBEROS.

El protocolo de seguridad ***Kérberos*** fue creado por el instituto de tecnología de Massachussets (MIT) como una solución al problema de seguridad en la red.

Este protocolo usa una fuerte criptografía, lo cual permite que un cliente pueda demostrar su identidad a un servidor y viceversa a través de una conexión de red insegura.

Después de que cliente y servidor han comprobado su identidad mediante Kerberos, pueden encriptar todas sus comunicaciones para asegurar la integridad y veracidad de los datos.

El protocolo Kerberos se basa en tres pilares fundamentales:

- Un Servidor de Autentificación (AS).
- Un Servidor de Concesión de Tickets (TGS).
- Un servidor de auditoria (KADM).

En lugar de construir protocolos de autentificación en cada servidor de una red, Kerberos ofrece un servicio de autentificación centralizado cuya función es autentificar usuarios frente a servidores y viceversa.

Actualmente se encuentran en uso dos versiones de Kerberos: la cuatro que es la más ampliamente usada y la cinco que corrige algunas deficiencias de la versión cuatro tanto a nivel del entorno como de deficiencias técnicas.

Kerberos utiliza criptografía para verificar la identidad del usuario, para lo cual maneja dos tipos de claves:

- ***Clave secreta del usuario.*** Clave conocida sólo por el usuario y el servidor Kerberos con la finalidad de autentificar al usuario. El Servidor de Autentificación conoce las claves de todos los usuarios y las almacena en una base de datos centralizada. El Servidor de Autentificación comparte una única clave secreta con cada servidor, las cuales fueron distribuidas físicamente o de alguna otra manera segura.

- ***Clave de sesión.*** Es una clave generada por el servidor Kerberos tras autentificar al usuario con la finalidad de autentificar el intercambio realizado por un determinado par de usuarios que definen la sesión. Esta clave es generada atendiendo una solicitud realizada por los usuarios, siendo válida por un tiempo determinado de antemano (tiempo de vida), y conocida únicamente por aquellas entidades para las que fue generada.

En este protocolo existen dos tipos de credenciales:

- ***Un ticket*** es un certificado distribuido por el servidor Kerberos que contiene un identificador del usuario en el cliente y la dirección de red del cliente, junto con el identificador del servidor. Ese ticket es encriptado para que no pueda ser alterado por el cliente o por un intruso usando la clave secreta compartida por el Servidor de Autentificación y ese servidor.

- ***Un autenticador*** es una credencial generada por el cliente, conteniendo información adicional. Sólo cuando el servidor desencripta el tiket y verifica que el identificador de usuario en el ticket es el mismo que el del mensaje recibido por el cliente, el servidor considera que el usuario es auténtico y otorga el servicio requerido.

El protocolo básico de Kerberos permite a un usuario con conocimiento de la clave secreta del cliente, obtener un ticket y una clave de sesión. Los tickets son reusables, es decir, que el mismo ticket puede usarse en múltiples ocasiones para acceder al servidor. Esto lleva a la necesidad de tener un tipo de ticket distinto para cada servicio diferente.

El Servidor de Concesión de Tickets ofrece facilidades para conseguir tickets a los usuarios que fueron autentificados por el Servidor de Autentificación.

El funcionamiento de Kerberos puede exponerse en cuatro etapas:

- ***Conexión al sistema.*** Un cliente se autentifica frente a Kerberos recibiendo un ticket y una clave de sesión correspondiente al intercambio entre el cliente y Kerberos, encriptados con una clave secreta derivada de la password del usuario.

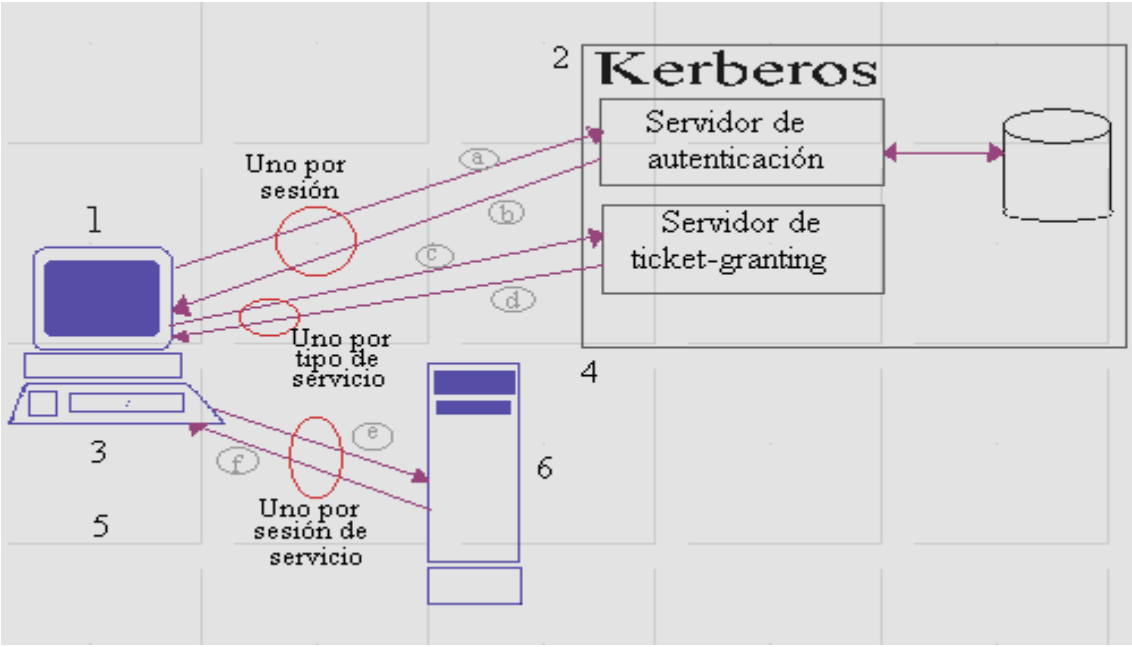
- ***Solicitud de acceso al servidor***: Para cada nuevo servicio que va a ser utilizado por el cliente, debe ser solicitado un ticket y una clave

de sesión para ese nuevo intercambio entre el cliente y el servidor. Esa información es enviada al cliente, encriptada con una clave de sesión que comparten el cliente y Kerberos.

- ***Acceso al servidor***: El cliente inicialmente envía el ticket recibido para el intercambio con el servidor. Debido a que ese ticket está cifrado con una clave secreta del servidor, él mismo deberá desencriptarla y así tendrá acceso a la clave de sesión entre el cliente y el servidor, que será válida por un período limitado (tiempo de vida), que generalmente es de 8 horas. Después de este procedimiento, la comunicación entre ellos podrá ser llevada a cabo de una manera confidencial.

- ***Mantenimiento de la Base de Datos de Kerberos***. Comprende el almacenamiento de las claves secretas del cliente y del servidor

Para ilustrar el procedimiento detallado anteriormente se expone a continuación un caso práctico que detalla en más profundidad el procedimiento.



- ***1.*** Un usuario desde su equipo requiere un servicio, para lo cual envía al servidor Kérberos una solicitud de ticket de acceso ***(a).***

- ***2.*** El servidor de autentificación verifica el correcto acceso del usuario a la base de datos, crea un ticket y una clave de sesión y

encripta ambos usando la clave derivada de la información de autentificación del usuario. El ticket y la clave de sesión son enviadas al usuario ***(b).***

- ***3.*** El equipo solicita al usuario su clave de acceso y la utiliza para desencriptar el mensaje. A continuación envía al Servidor de Concesión de Tickets el ticket y el autentificador que contienen el nombre del usuario, la dirección de red y el tiempo de vida del ticket para solicitar un ticket de acceso al servicio ***(c).***

- ***4.*** El Servidor de Concesión de Tickets desencripta el ticket y el autentificador, verifica la solicitud y crea un ticket para ser enviado al servidor junto con la clave de sesión ***(d).***

- ***5.*** El equipo del usuario envía al servidor el ticket y el autentificador, con lo cual solicita el servicio ***(e).***

- ***6.*** El servidor verifica que el ticket y el autentificador coincidan y entonces permite el acceso al servicio, enviando al cliente un autentificador del propio servidor ***(f).***

11.5 ANEXO 5. Guía del administrador.

El administrador es la persona encargada de:

1. Arrancar el servidor OpenLDAP.

El primer paso para iniciar el sistema el arrancar el servidor LDAP. Para ello usará el comando “Slapd” de la manera que se expone en la memoria, siendo la forma más común y mayormente empleada:

slpad –f <ruta del fichero de configuración>

2. Arrancar el proxy CORBA-LDAP.

Para arrancar el proxy CORBA-LDAP deberá ejecutar el comando de “Jacorb” que inicializa el proxy. Deberá dar en la línea de comando la dirección URL en la que se encuentra el servidor de directorio.

jaco clservidor ldap://<dirección IP>:<número de puerto>

3. Configuración del servidor.

El administrador debe gestionar la concesión de controles de acceso a los usuarios, es decir, los permisos de los que gozarán éstos una vez que hayan accedido al sistema convenientemente autentificados, para lo que manipulará el fichero de configuración del servidor, añadiendo las directivas necesarias.

El propio administrador se encargará de configurar los directorios que se consideren necesarios, así como incorporar las entradas que se encuentren en formato LDIF al directorio, para lo cual usará la herramienta “ldapadd” de OpenLDAP.

El administrador configurará la información de esquema que maneja el servidor, especificando en el fichero de configuración los esquemas a usar mediante la directiva:

Include <nombre del fichero de esquema>

El administrador debe declararse a sí mismo en el fichero de configuración del servidor mediante una entrada que lo represente como administrador del sistema, concediéndose así todos los permisos de usuario.

4 Configuración de CORBA.

La configuración del fichero de configuración del "Jacorb" "Jacorb.properties" corresponde al administrador, que deberá indicar ahí posibles cambios en la localización del servidor de nombres o del adaptador de objetos. Para ello usará las directivas expuestas en la configuración de Jacorb.

5. Gestión de usuarios.

El administrador debe conocer qué usuarios usan el sistema, para lo cual asignará una entrada en el directorio (perteneciente a la clase objeto "person") que represente al usuario que desea ser registrado en el sistema. Desde ese momento el usuario podrá hacer uso del sitema autentificándose correctamente.

Esto podrá realizarlo de manera local mediante la herramienta "ldapadd" o de manera remota mediante el cliente CORBA. El administrador comunicará al usuario su identificador y su clave.

También es tarea del administrador la gestión de las claves de los usuarios, cambios de claves, pérdida de claves, etc.

6. Gestión local del directorio.

El administrador dispone de todas las herramientas que ofrece OpenLDAP para gestionar el directorio localmente. También tendrá acceso remoto al directorio a través de CORBA, ya que él mismo se habrá registrado como superusuario del sistema.

7. Detener el servidor y el proxy.

Cuando se desee terminar con la ejecución del proxy se procederá a abortar la aplicación proxy mediante (CTRL+c).
Para detener el servidor OpenLDAP se usa el comando “kill” de Linux seguido seguido del número de identificador de proceso que fue asignado a la aplicación en el momento de iniciarse. Dicho identificador puede consultarse en el fichero “pidfile” o bien mediante el comando “ps –ef” de Linux.

11.6 ANEXO 6. Guía del usuario.

Esta guía describe los conocimientos que el usuario debe tener acerca del cliente CORBA para poder manejar el sistema, ya que ni el funcionamiento del servidor de directorio ni el funcionamiento del proxy son de su competencia.

1. Inicio de la aplicación.

El usuario iniciará la aplicación desde el directorio de la aplicación cliente ejecutando:

jaco cliente

2. Manejo de la aplicación.

Se presenta al usuario una ventana con una serie de botones que pueden usarse para invocar las operaciones LDAP que representan y un cuadro de texto principal.

La ventana principal ofrece también una serie de menús que permiten realizar las operaciones de los botones, realizar operaciones adicionales de consulta sobre el esquema y características del servidor.

Existe un menú que permite salir de la aplicación, y un último menú que proporciona un sistema de ayuda en línea al cliente sin más que elegir la función sobre la que se desea obtener ayuda. Una vez solicitada la ayuda, la información disponible se presenta en el cuadro de texto de la ventana principal de la aplicación.

La información se solicita de manera clara y concreta, por lo que el usuario sólo tendrá que rellenar el campo que aparece junto con cada solicitud.

En caso de duda puede consultarse ya ayuda que ofrece la aplicación.

2.1 **Invocación de operaciones.**

Para realizar una operación el cliente elegirá el botón o el menú que la gestiona, y rellenará los campos con los datos que en ellos se piden, seleccionará las opciones pertinentes y el tipo de autentificación que desea usar, rellenando obligatoriamente los

campos de autentificación en caso de elegir el método simple, ya que éstos son ignorados en la autentificación anónima.

En el proceso de autentificación el usuario escribe sus datos sólo en la primera operación que realiza, ya que esa información será mantenida en consultas posteriores como información de autentificación por defecto.

A la hora de introducir la clave de usuario, ésta no queda visible para el propio usuario.

Hay que hacer especial mención a las operaciones de esquema, en las que además de la información de autentificación se inicializa el prefijo que debe llevar la información que se solicita.

Una vez rellenados los campos se dispone de tres opciones que vienen dadas por los botones de cada ventana.

- ***Cancelar***. Elimina la ventana, no se realiza ninguna operación y se regresa a la ventana principal.

- ***Borrar.*** Se elimina la ventana y se genera una nueva del mismo tipo, pero en esta ocasión todos los campos están vacíos, y la información de autentificación por defecto es eliminada. Esta opción es especialmente útil para redefinir la autentificación del usuario cuando se accede a una aplicación.

- ***Enviar***. El cliente interpreta toda la información suministrada en la ventana y realiza la consulta. Si el usuario no ha suministrado algún campo imprescindible o no lo ha hecho en el orden adecuado, no se hará nada, permitiendo que el usuario que complete la información.

  Se aconseja rellenar los campos por orden, siendo este orden necesario en aquellos campos en los que se indican los atributos a devolver, los atributos a añadir o las modificaciones a realizar.

2.2 Presentación de resultados.

Una vez efectuada la operación, los resultados de la misma, ya sean datos válidos o información referente a una excepción capturada durante el proceso (ya sea por error en la operación o en el sistema), son presentados en una nueva ventana específica de presentación de resultados de la operación que se ha realizado.

En esta ventana el usuario puede consultar la información generada. Si la información contiene datos válidos la ventana que se usó para invocar la operación se cierra; permaneciendo activa en caso de que los resultados generados provengan de la captura de una excepción.

Cuando el usuario termine de consultar la información con solo pulsar el botón “aceptar” la ventana se ocultará y se volverá bien a la ventana de ejecución de la operación, bien a al ventana principal, según el caso.

3. Finalizar la aplicación.

Se aconseja realizar esta aplicación desde el menú “Salir”.

11.7 ANEXO 7: Guía de instalación.

La siguiente guía pretende exponer de manera mecánica, simple y concisa los pasos que el instalador deber seguir para instalar el sistema, dejándolo preparado para que el administrador pueda desarrollar su labor.

Su uso está orientado a servir cómo guía rápida de todo el proceso expuesto en la memoria.

1. Instalación de Java SDK 1.3

- Simplemente se debe descomprimir el software mediante la herramienta "archivador" de Linux.

- Incorporar las clases java al "classpath" del sistema y las herramientas java (directorio "bin") al "path".

2. Instalación de OpenLDAP

- Para ello descomprimir el software mediante la herramienta "archivador" de Linux.

- Ejecutar el guión de configuración desde el directorio en el que se ha descomprimido: ./configure <opciones>

- Construir las dependencias mediante make depend

- Compilar el servidor mediante la orden make

- Instalar el servidor mediante la orden make install

3. Instalación de Jacorb

- Descomprimir el software mediante la herramienta "archivador" de Linux.

- Actualizar las variables de entorno del sistema "classpath" (para que contenga las clases de "Jacorb", "ant" y "Java"), "path" (para que contenga la ruta de las herramientas de "Jacorb", "ant" y "Java"), "JAVA_HOME" (indicando donde está el directorio JRE de Java),

"Jacorb_home" (indicando el directorio que contiene "Jacorb") y "Ant_home" (indicando el directorio que contiene "Ant").

- Ejecutar la herramienta "ant" (previamente instalada sin más que descomprimir el software que la contiene) en el directorio en el que se ha descomprimido Jacorb.

- Añadir las clases de "Jacorb" al "classpath" del sistema y las utilidades al "path".

4. Instalación del proxy CORBA-LDAP.

- Ejecutar el intérprete de IDL sobre el fichero "corbaldap.idl" contenido en el directorio "corbaldap_definición". Para ello usar la herramienta "idl": idl corbaldap.idl

- Compilar con la herramienta "javac" de Java todas las clases generadas y añadirlas al "classpath" del sistema.

- Compilar con la herramienta "javac" de Java todas las clases del servidor contenidas en "corbaldap_servidor". Añadir estas clases al "classpath" del sistema.

5. Instalación del cliente CORBA.

- Ejecutar el intérprete de IDL sobre el fichero "corbaldap.idl" contenido en el directorio "corbaldap_definición". Para ello usar la herramienta "idl": idl corbaldap.idl

- Compilar con la herramienta "javac" de Java todas las clases generadas y añadirlas al "classpath" del sistema.

- Compilar con la herramienta "javac" de Java todas las clases del cliente contenidas en "corbaldap_cliente". Añadir estas clases al "classpath" del sistema.

11.8 Anexo 8: Pruebas.

En este apartado se detallan las pruebas realizadas para comprobar la robustez del código, y que el funcionamiento del sistema responda al modelo esperado.

Para ello se ha montado el sistema en la red del laboratorio, pero se ha usado solamente dos equipos para la instalación. Sin embargo los componentes del sistema han sido dispuestos de tal manera que en todos los pasos que se producen desde que el usuario solicita un servicio hasta que recibe los resultados de éste, se hace uso de la red, ya que cualquier elemento que realice el paso correspondiente deberá acceder al otro equipo para llevarlo a cabo.

La disposición de los elementos es la siguiente:

- En el primer equipo se ha instalado el cliente CORBA y servidor OpenLDAP.

- En el segundo equipo se ha instalado el proxy CORBA-LDAP y servicio de nombres de CORBA.

De esta manera excepto en el proceso que sigue el servidor CORBA para registrarse, todos los pasos se llevan a cabo a través de la red, pero ya que el proceso de registro del servidor CORBA se realiza mientras el sistema está arrancando, podemos afirmar que durante el funcionamiento normal del sistema se simula la existencia de cuatro equipos distintos (uno para el cliente CORBA, otro para el proxy, otro para el servidor OpenLDAP y otro para el servicio de nombres de CORBA), todos ellos conectados a través de la red local.

Para la realización de las pruebas, la información de directorio usada es la del fichero LDIF de ciudades del mundo que puede consultarse en el apartado Anexos.

Una vez inicializado el servicio de nombrado, el servidor OpenLDAP, el proxy y por último el cliente, se procede a la realización de las pruebas.

En las pruebas se indica el resultado que devuelve el cliente, resultado que ha sido enviado por el proxy. En aquellos casos en los que la información mostrada en el propio proxy sea más completa que la devuelta (casos en los que se producen errores) se indica también la información que se obtiene monitorizando el propio proxy.

11.8.1 Pruebas comunes a todas las operaciones.

Las siguientes pruebas han sido realizadas en todas las operaciones, obteniéndose en todos los casos los mismos resultados.

PRUEBA 1: Se inicializa el proxy indicando una URL LDAP en la que se codifica una dirección no existente.

El cliente responde con un mensaje en el que indica que ha ocurrido un problema durante la operación y devuelve la dirección IP y número de puerto a la que ha intentado acceder (en este caso 172.16.18.98:389).

Si se monitoriza el funcionamiento del proxy, aparece por pantalla una excepción java del tipo “javax.naming.communicationException” con el mensaje “172.16.18.98:389 root exception is java.net.NoRouteToHostException”.

PRUEBA 2: Se intenta acceder al servidor cuando éste no está funcionando.

El cliente responde con un mensaje en el que indica que ha ocurrido un problema durante la operación y devuelve la dirección IP y el número de puerto al que se ha intentado acceder (en este caso 172.16.17.98:389)

En el proxy aparce por pantalla una excepción del tipo “javax.naming. communicationException” con el mensaje “172.16.17.98:389 root exception java.net.ConnectionException conexión rehusada”.

PRUEBA 3: Se intenta acceder al directorio autentificándose con un nombre de usuario no válido. Para ello usa el nombre de usuario “cn=Juan,di=usuarios,di=ciudades”.

Se muestra por pantalla un mensaje que indica que se ha producido un problema durante la operación, y el código de error [LDAP: error code 49-Invalid Credentials].

En la pantalla del proxy se imprime una excepción del tipo “javax.naming.AuthenticationException”.

PRUEBA 4: Se intenta acceder al servidor usando una clave no válida. Para ello se usa el nombre del administrador (cn=Manuel,di=usuarios,di=ciudades), y se usa la clave "entrar", la cual no se corresponde con la clave del administrador.

El resultado es exactamente el mismo que el de la prueba número 3.

11.8.2 Pruebas con la operación COMPARE.

PRUEBA 5: Se consulta al servidor si la entrada de nombre "ci=Tokyo,pa=Japon,co=Asia,di=ciudades" contiene la pareja "ci=Tokio". Para ello se usa autentificación simple con el nombre del administrador y su clave.

El cliente responde con un mensaje de operación realizada con éxito. Sí contiene la pareja.

PRUEBA 6: Se consulta al servidor si la entrada de nombre "pa=Japon,co=Asia,di=ciudades" contiene la pareja "capital=Berlín". Para ello se usa autentificación simple con el nombre del administrador y su clave.

El resultado mostrado es un mensaje de operación realizada con éxito. No contiene la pareja.

PRUEBA 7: Comprobar si la entrada "pa=Japon,co=Asia,di=ciudades" contiene el atributo "capital", para lo cual se usa la pareja "capital=*". Para ello se usa autentificación simple con el nombre del administrador y su clave.

El resultado mostrado es un mensaje de operación realizada con éxito. Sí contiene la pareja.

PRUEBA 8: Se repite la prueba cinco pero en esta ocasión se usa el método de autentificación anónima.

El resultado es el mismo que el de la prueba 5, ya que el control de acceso configurado en el servidor concede permiso de lectura a los usuarios anónimos.

PRUEBA 9: Se intenta hacer una comparación en una entrada que no existe. Para ello se va a comprobar si la entrada "pa=Portugal,co=Europa,di=ciudades" (que no existe) contiene la pareja "capital=Lisboa".

Se muestra por pantalla que "ha ocurrido un error durante la operación", y el código de error [LDAP: error code 32-No such Object], indicando que la entrada no existe.

En el proxy se muestra una excepción del tipo "javax.naming.NameNotFound"

PRUEBA 10: Se intenta comprobar si un atributo no existente contiene cierto valor. Para ello se solicita comprobar si la entrada "pa=Italia,co=Europa,di=ciudades" contiene la pareja "Rios=Po".

El resultado mostrado es un mensaje de operación realizada con éxito. No contiene la pareja.

PRUEBA 11: Se realiza una comparación adjuntando como nombre de la entrada una cadena vacía.

El propio cliente no permite realizar la operación sin este dato.

PRUEBA 12: Se realiza una comparación sin indicar el identificador del atributo sobre el que se quiere hacer la comparación.

El propio cliente no permite realizar la operación sin este dato.

PRUEBA 13: Se realiza la comparación sin el valor del atributo. Se pide comprobar si la entrada "pa=Italia,co=Europa,di=ciudades" contiene la pareja (Capital=).

El resultado mostrado es un mensaje de operación realizada con éxito. No contiene la pareja.

11.8.3 Pruebas con la operación ADD.

PRUEBA 14: Añadir una nueva entrada al directorio. Se añade la entrada "pa=Portugal,co=Europa,di=ciudades" con el conjunto de atributos "objectclass=pais", "pa=Portugal", "Moneda=Euro", "Capital=Lisboa".
Para ello se ha usado autentificación simple, usando la identidad y la clave del administrador.

El resultado es el mensaje de operación realizada con éxito. La entrada ha sido añadida.

PRUEBA 15: Tras eliminar la entrada anteriormente añadida en la prueba 13, se repite dicha prueba 13 usando esta vez autentificación anónima.

El resultado es un error durante la operación y se muestra el código de error [LDAP:error code 50- no write access to parent] que indica que se intenta realizar una operación para la cual no se tiene permiso.

El proxy muestra por pantalla una excepción del tipo "javax.naming.NoPermisionException".

PRUEBA 16: Se repite la prueba 15, pero en esta ocasión se accede como el usuario "cn=usu1,di=usuarios,di=ciudades" y con su clave correspondiente".

El resultado el mismo que el de la prueba 15, ya que este usuario no goza de permiso para actualizar la información del directorio.

PRUEBA 17: Se intenta añadir una entrada que cuelgue de un nodo no existente. Para ello se intenta añadir la entrada "ci=Atenas,pa=Grecia,co=Europa,di=ciudades" con atributos "ci=Atenas", "Habitantes=200000". La entrada "pa=Grecia,co=Europa,di=ciudades" no existe. Se accede al sistema autentificado como el administrador.

Se muestra por pantalla que "ha ocurrido un error durante la operación", y el código de error [LDAP: error code 32-No such Object], indicando que la entrada no existe.

En el proxy se muestra una excepción del tipo "javax.naming.NameNotFound"

PRUEBA 18: Se intenta añadir una entrada ya existente. Para ello autentificado como el administrador del sistema se intenta añadir la entrada "pa=Francia,co=Europa,di=ciudades" (que ya existe).

El resultado es un mensaje del tipo "ha ocurrido un error durante la operación" y se muestra el código de error [LDAP:error code 68- Entry Already Exist], que indica que la entrada ya existe.

En el proxy se muestra una excepción del tipo "javax.naming.NameAlreadyBoundException"

PRUEBA 19: Se intenta añadir una entrada que no contenga el atributo "objectclass". Para ello autentificado como el administrador del sistema se intenta añadir la entrada "ci=Bolonia,pa=Italia,co=Europa,di=ciudades" con los atributos iniciales "ci=Bolonia" y "ci=90000" (los exigidos por el esquema), pero sin el atributo "objectclass=ciudad".

El resultado es un mensaje del tipo "ha ocurrido un error durante la operación" y se muestra el código de error [LDAP:error code 69- no objectclass attribute], que indica que el atributo "objectclass" está ausente.

El proxy muestra una excepción del tipo "javax.naming.directory.SchemaViolationException"

PRUEBA 20: Se intenta añadir una entrada que no cumple con las reglas del esquema. Para ello se repite la prueba 19, pero en esta ocasión sí se añade el atributo "objectclass=ciudad" pero no el atributo "ci=Bolonia", que según el esquema es obligatorio.

El resultado es un mensaje del tipo "ha ocurrido un error durante la operación" y se muestra el código de error
[LDAP:error code 65- missing required attribute]

En el proxy se obtiene la misma excepción que en la prueba 19.

PRUEBA 21: Se intenta añadir una entrada que cumple con la información de esquema, pero además tiene un atributo no definido por dicho esquema.
Para ello se repite la prueba 19, pero añadiendo el atributo "objectclass=ciudad" además del atributo "clima=lluvioso" (que no existe).

El resultado es un mensaje del tipo "ha ocurrido un error durante la operación" y se muestra el código de error
[LDAP:error code 17- attribute type undefined], que indica que el atributo que se intenta usar no está definido.

El proxy muestra una excepción del tipo
"javax.naming.InvalidAttributeIdentifierException".

PRUEBA 22: Se repite la prueba 21, pero en esta ocasión el atributo que no cumple con las reglas del esquema sí está definido. Se intenta añadir el atributo "moneda=euro" que aunque existe no está permitido en la clase objeto "ciudad".

El resultado es un mensaje del tipo "ha ocurrido un error durante la operación" y se muestra el código de error
[LDAP:error code 65- attribute not allowed], que indica que el atributo no está permitido en la clase objeto.

El proxy muestra una excepción del tipo
"javax.naming.directory.SchemaViolationException"

PRUEBA 23: Se intenta añadir un atributo que contiene un valor no válido, ya que no cumple con la sintaxis de atributo definida para ese tipo de atributo. Para ello se repite la prueba 19, pero se añade el atributo "objectclass=ciudad" y el atributo "habitantes=muchos".

El resultado es un mensaje del tipo "ha ocurrido un error durante la operación" y se muestra el código de error
[LDAP:error code 21- value contains invalid data], que indica que un atributo contiene un valor que no se corresponde con su sintaxis.

El proxy muestra una excepción del tipo
"javax.naming.directory.InvalidAttributeValueException"

PRUEBA 24: Se intenta añadir una entrada que tenga por nombre una cadena vacía habiéndose autentificado como el administrador del sistema.

El resultado es un mensaje del tipo "ha ocurrido un error durante la operación" y se muestra el código de error
[LDAP:error code 68- root DSE existe], que indica que la entrada que se mantiene para contener información del servidor, y cuyo nombre es una cadena vacía, ya existe.

El proxy muestra una excepción del tipo
"javax.naming.NameAlreadyBoundException"

PRUEBA 25: Se intenta añadir una entrada sin atributos, para lo cual se repite la prueba 19, pero en esta ocasión no se añade ningún atribu**to.**

El resultado es un mensaje del tipo "ha ocurrido un error durante la operación" y se muestra el código de error
[LDAP:error code 2- No attributes provided], que indica que deben suministrarse los atributos necesarios.

Si se monitoriza el funcionamiento del proxy, aparece por pantalla una excepción java del tipo "javax.naming.communicationException".

11.8.4 Pruebas con la operación DELETE.

PRUEBA 26: Se intenta borrar una entrada del árbol de tipo hoja, accediendo al sistema como un usuario anónimo. La entrada que se intenta eliminar es "ci=Roma,pa=Italia,co=Europa,di=ciudades".

El resultado es un mensaje del tipo "ha ocurrido un error durante la operación" y se muestra el código de error
[LDAP: error code 50-Insufficient Access Rights] que indica que el usuario no dispone de privilegios para realizar esa operación.

El proxy muestra una excepción del tipo
"javax.Naming.NoPermissionException".

PRUEBA 27: Se intenta eliminar una entrada del árbol que tenga hijos accediendo de forma anónima. La entrada que se intenta eliminar es "pa=Italia,co=Europa,di=ciudades".

El resultado es un mensaje del tipo "ha ocurrido un error durante la operación" y se muestra el código de error
[LDAP: error code 66-subtree delete not supported], que indica al usuario que no puede eliminar una entrada que tenga hijos.

El proxy muestra una excepción del tipo
"javax.naming.ContextNotEmptyException".

PRUEBA 28: Se repite la prueba 26 pero accediendo como un usuario sin privilegio de escritura. Se accede como "cn=usu1,di=usuarios,di=ciudades".

El resultado es análogo al de la prueba 26.

PRUEBA 29: Se repite la prueba 27 pero accediendo como un usuario sin privilegio de escritura. Se accede como anónimo.

El resultado es análogo al de la prueba 27.

PRUEBA 30: Se intenta eliminar una entrada existente. Para ello accediendo como el administrador se intenta eliminar la entrada "ci=Atenas,pa=Grecia,co=Europa,di=ciudades" (la cual no existe).

Se muestra por pantalla que "ha ocurrido un error durante la operación", y el código de error [LDAP: error code 32-No such Object], indicando que la entrada no existe.

En el proxy se muestra una excepción del tipo "javax.naming.NameNotFound"

PRUEBA 31: Se intenta borrar una entrada cuyo nombre sea una cadena vacía. Se realiza esta prueba accediendo al sistema como administrador.

El resultado es análogo al de la prueba 30.

PRUEBA 32: Se intenta eliminar una entrada existente accediendo como administrador. La entrada elegida para ser eliminada es "pa=Francia,co=Europa,di=ciudades".

El resultado es "operación realizada con éxito". La entrada ha sido eliminada.

Si se intenta buscar esta entrada mediante el método de búsqueda no se devuelven resultados.

11.8.5 Pruebas con operaciones de esquema.

PRUEBA 33: Se intenta obtener información de una clase objeto que no existe. Para ello se intenta mostrar la entrada "ClassDefinition/clima" (que no existe), accediendo al sistema como administrador.

Se muestra por pantalla que "ha ocurrido un error durante la operación", pero en esta ocasión no se muestra ningún código de error.

El proxy en cambio muestra una excepción del tipo "javax.naming.nameNotFoundException".

PRUEBA 34: Se lista el contenido de una clase objeto existente. Para ello se accede como administrador y se solicita la clase "ClassDefinition/ciudad".

El resultado es "operación realizada con éxito". Se muestra por pantalla la información de esquema de la clase objeto ciudad.

PRUEBA 35: Se repite la prueba 34 accediendo como el usuario "cn=usu1,di=usuarios,di=ciudades".

El resultado es el mismo que el de la prueba 34

PRUEBA 36: Se repite la prueba 34 accediendo esta vez como un usuario anónimo.

El resultado es el mismo que el de la prueba 34

PRUEBA 37: Se intenta acceder a un tipo de atributo que no existe. Para ello se solicita como administrador del sistema el atributo "clima".

Se muestra por pantalla que "ha ocurrido un error durante la operación", pero en esta ocasión no se muestra ningún código de error.

El proxy en cambio muestra una excepción del tipo "javax.naming.nameNotFoundException".

PRUEBA 38: Se accede a un tipo de atributo existente. En este caso el atributo "AttributeDefinition/moneda".

El resultado es "operación realizada con éxito". Se muestra por pantalla la información de esquema del tipo de atributo moneda.

PRUEBA 39: Se repite la prueba 38 accediendo como el usuario "cn=usu1,di=usuarios,di=ciudades".

El resultado es el mismo que en la prueba 38.

PRUEBA 40: Se repite la prueba 38, pero en esta ocasión se accede como un usuario anónimo.

El resultado es el mismo que en la prueba 38.

PRUEBA 41: Se pide información sobre una sintaxis de atributo. En esta ocasión se accede como administrador a la entrada "SyntaxDefinition/ 1.3.6.1.4.1.1466.115.121.1.15", que es identificador de objeto de una cadena de caracteres.

El resultado es "operación realizada con éxito". Se muestra por pantalla la información de esquema de la sintaxis de atributo cadena de caracteres.

PRUEBA 42: Se repite la prueba 41 accediendo como el usuario "cn=usu1,di=usuarios,di=ciudades".

El resultado es el mismo que el de la prueba 41.

PRUEBA 43: Se repite la prueba 41 accediendo como un usuario anónimo.

El resultado es el mismo que el de la prueba 41.

PRUEBA 44: Se accede como administrador a una sintaxis de atributo que no existe. Para ello se intenta acceder a "SyntaxDefinition/1.3.6.1.4.1.1466.115.121.1.15.1" (que no existe).

Se muestra por pantalla que "ha ocurrido un error durante la operación", pero en esta ocasión no se muestra ningún código de error.

El proxy en cambio muestra una excepción del tipo "javax.naming.nameNotFoundException".

PRUEBA 45: Se intenta acceder como administrador a la información de una regla de correspondencia. Para ello se intenta acceder a la entrada "MatchingRule/CaseIgnoreMatch".

El resultado es "operación realizada con éxito". Se muestra por pantalla la información de esquema de la regla de correspondencia solicitada.

PRUEBA 46: Se repite la prueba 45 accediendo como el usuario "cn=usu1,di=usuarios,di=ciudades".

Se obtiene el mismo resultado que en la prueba 45.

PRUEBA 47: Se repite la prueba 45 pero accediendo de forma anónima.

Se obtiene el mismo resultado que en la prueba 45.

PRUEBA 48: El administrador intenta acceder a una regla de correspondencia que no existe. Se intenta acceder a "MatchingRule/cadenas_iguales".

Se muestra por pantalla que "ha ocurrido un error durante la operación", pero en esta ocasión no se muestra ningún código de error.

El proxy en cambio muestra una excepción del tipo "javax.naming.nameNotFoundException".

PRUEBA 49: Se pide información de esquema de una entrada del DIT que no existe. Se pide información sobre "ci=Atenas,pa=Grecia,co=Europa,di=ciudades".

Se muestra por pantalla que "ha ocurrido un error durante la operación", y el código de error [LDAP: error code 32-No such Object], indicando que la entrada no existe.

En el proxy se muestra una excepción del tipo "javax.naming.NameNotFound"

PRUEBA 50: Se pide información de esquema de una entrada del DIT. En este caso se solicita "co=Oceanía,di=ciudades".

El resultado es "operación realizada con éxito". Se muestra por pantalla la información de esquema de la entrada solicitada.

PRUEBA 51: Se pide información de esquema de una entrada cuyo nombre es una cadena vacía.

El propio cliente no permite realizar la operación.

11.8.6 Pruebas con la operación LISTAR.

PRUEBA 52: Un usuario anónimo intenta listar el contenido de la entrada "ci=Washington,pa=Estados Unidos,co=America,di=ciudades".

El resultado es "operación realizada con éxito". Se muestra por pantalla el identificador de la entrada y todos sus atributos y valores.

PRUEBA 53: Se repite la prueba 52 accediendo como el usuario "cn=usu3,di=usuarios,di=ciudades".

El resultado es el mismo que el de la prueba 52.

PRUEBA 54: Se repite la prueba 52 accediendo como el administrador de directorio.

El resultado es el mismo que el de la prueba 52.

PRUEBA 55: Se intenta acceder a una entrada cuyo nombre sea una cadena vacía.

Se muestra una entrada que no contiene ningún nombre ni atributo.

PRUEBA 56: Se intenta acceder a una entrada no existente. En este caso se intenta acceder a la entrada "ci=Atenas,pa=Grecia,co=Europa,di=ciudades".

Se muestra por pantalla que "ha ocurrido un error durante la operación", y el código de error [LDAP: error code 32-No such Object], indicando que la entrada no existe.

En el proxy se muestra una excepción del tipo
“javax.naming.NameNotFound”

PRUEBA 57: Listar el contenido de una entrada y pedir que no se devuelvan valores de atributos. Se consulta la entrada “ci=Washington,pa=Estados Unidos,co=America,di=ciudades”.

El resultado es “operación realizada con éxito”. Se muestra por pantalla el identificador de la entrada y todos sus atributos pero en esta ocasión no se muestran valores.

11.8.7 Pruebas con la operación MODIFY.

PRUEBA 58: Se intenta añadir un atributo a una entrada sin tener privilegios. Para ello se intenta añadir el atributo “habitantes=100000000” a la entrada “pa=Brasil,co=America,di=ciudades”. Se accede al sistema como un usuario anónimo

El resultado es un mensaje del tipo ”ha ocurrido un error durante la operación” y se muestra el código de error
[LDAP: error code 50-Insufficient Access Rights] que indica que el usuario no dispone de privilegios para realizar esa operación.

El proxy muestra una excepción del tipo
“javax.Naming.NoPermissionException”.

PRUEBA 59: Añadir un atributo a una entrada que no existe. Para ello accediendo como el administrador se intenta añadir el atributo “habitantes=300000” a la entrada “ci=Atenas,pa=Grecia,co=Europa,di=ciudades”(que no existe).

Se muestra por pantalla que “ha ocurrido un error durante la operación”, y el código de error [LDAP: error code 32-No such Object], indicando que la entrada no existe.

En el proxy se muestra una excepción del tipo
“javax.naming.NameNotFound”

PRUEBA 60: Se repite la prueba 59, pero en esta ocasión no se introduce ninguna entradda.

El resultado es el mismo que en la prueba 59.

PRUEBA 61: Se intenta reemplazar un atributo no existente. Accediendo como administrador se intenta reemplazar los valores del atributo "capital" por el valor "Bruselas" en la entrada "co=Europa,di=ciudades" (que no contiene ese atributo).

El resultado es un mensaje del tipo "ha ocurrido un error durante la operación" y se muestra el código de error
[LDAP:error code 17- attribute type undefined], que indica que el atributo que se intenta usar no está definido.

El proxy muestra una excepción del tipo
"javax.naming.InvalidAttributeIdentifierException".

PRUEBA 62: Borrar un atributo que no existe. Se repite la prueba 61, pero en esta ocasión en vez de reemplazar el atributo se intenta borrar.

El resultado es un mensaje del tipo "ha ocurrido un error durante la operación" y se muestra el código de error
[LDAP:error code 16 - modify: delete values failed].

En el proxy se muestra una excepción del tipo
"javax.naming.directory.NoSuchAttributeException".

PRUEBA 63: Borrar un valor de atributo que no existe dentro de un atributo existente. Accediendo como administrador se intenta eliminar el valor del atributo "co=Asia" de la entrada "co=Asia,di=ciudades".

El resultado es el mismo que en la prueba 62.

PRUEBA 64: Se intenta añadir un valor no válido a un atributo. Para ello accediendo como administrador se añade el valor "habitantes=muchos" a la entrada "co=Europa,di=ciudades".

El resultado es un mensaje del tipo "ha ocurrido un error durante la operación" y se muestra el código de error
[LDAP:error code 21- value contains invalid data], que indica que un atributo contiene un valor que no se corresponde con su sintaxis.

El proxy muestra una excepción del tipo
"javax.naming.directory.InvalidAttributeValueException"

PRUEBA 65: Añadir un valor a un atributo. Accediendo como administrador se añade el valor "habitantes=1" a la entrada "co=Europa,di=ciudades".

Se obtiene un mensaje de "operación realizada con éxito". Las modificaciones han sido realizadas.

Si se lista el contenido de la entrada se observa que el valor ha sido añadido.

PRUEBA 66: Borrar un valor de atributo. Se elimina el valor de atributo añadido en la prueba 65.

Se obtiene un mensaje de "operación realizada con éxito". Las modificaciones han sido realizadas.

Si se lista el contenido de la entrada se observa que el valor ha sido eliminado.

PRUEBA 67: Añadir un nuevo atributo a una entrada. Accediendo como administrador se añade a la entrada "co=Europa,di=ciudades" el atributo "Nombrepropio" con el valor "Europa".

Se obtiene un mensaje de "operación realizada con éxito". Las modificaciones han sido realizadas.

Si se lista el contenido de la entrada se observa que el atributo y el valor han sido añadidos.

PRUEBA 68: Eliminar un atributo y todos sus valores. Para ello se elimina el atributo añadido en la prueba 67.

Se obtiene un mensaje de “operación realizada con éxito”. Las modificaciones han sido realizadas.

Si se lista el contenido de la entrada se observa que el atributo ha sido eliminado.

PRUEBA 69: Añadir un atributo sin valores iniciales. Se repite la prueba 67, pero en esta ocasión no se añade ningún valor al atributo “Nombrepropio”.

El resultado es un mensaje del tipo ”ha ocurrido un error durante la operación” y se muestra el código de error
[LDAP:error code 21- value contains invalid data], que indica que un atributo contiene un valor que no se corresponde con su sintaxis.

El proxy muestra una excepción del tipo
“javax.naming.directory.InvalidAttributeValueException”

PRUEBA 70: Añadir un atributo inicializado con más de un valor. Se repite la prueba 67, pero además del valor “Europa” se añade el valor “Europe”.

Se obtiene un mensaje de “operación realizada con éxito”. Las modificaciones han sido realizadas.

Si se lista el contenido de la entrada se observa que el atributo y los valores han sido añadidos.

PRUEBA 71: Reemplazar los valores de un atributo. Accediendo como administrador se reemplazan los valores del atributo “Nombrepropio” de la entrada Europa añadidos en la prueba 70 por un único valor “Comunidad Europea”.

Se obtiene un mensaje de “operación realizada con éxito”. Las modificaciones han sido realizadas.

Si se lista el contenido de la entrada se observa que el atributo sólo contiene el valor “Comunidad Europea”.

PRUEBA 72: Reemplazar los valores de un atributo por una cadena vacía. Se repite la prueba 71 (ahora el atributo “Nombrepropio” sólo tiene un valor), usando una cadena vacía para reemplazar el valor del atributo.

El resultado es un mensaje del tipo ”ha ocurrido un error durante la operación” y se muestra el código de error
[LDAP:error code 21- invalid data], que indica que un atributo contiene un valor que no se corresponde con su sintaxis.

El proxy muestra una excepción del tipo
“javax.naming.directory.InvalidAttributeValueException”

PRUEBA 73: Se realizan tres operaciones a la vez. Para ello accediendo como administrador se añade a la entrada “co=Europa,di=ciudades” el valor “habitantes=2”, se reemplaza el contenido del atributo “superficie” por el valor “2” y se elimina el atributo “co=Europa” (esta última operación no está permitida por el esquema).

El resultado es un mensaje del tipo “ha ocurrido un error durante la operación” y se muestra el código de error
[LDAP:error code 65- missing required attribute] que indica que ese atributo es indispensable.

El proxy muestra una excepción del tipo
“javax.naming.directory.SchemaViolationException”

Si se lista el contenido de la entrada se observa que ninguna de las dos primeras modificaciones se han llevado a cabo.

PRUEBA 74: Se repite la prueba 73 pero en esta ocasión el valor que se intenta eliminar es “NombrePropio”.

Se obtiene un mensaje de “operación realizada con éxito”. Las modificaciones han sido realizadas.
Si se lista el contenido de la entrada se observa que las tres modificaciones han sido correctamente realizadas.

11.8.8 Pruebas con la operación MODIFY_RDN.

PRUEBA 75: Ejecución de la operación por parte de un usuario sin permiso. Se accede de manera anónima y se intenta cambiar el nombre de la entrada "pa=Alemania,co=Europa,di=ciudades" por el nombre "pa=Germany,co=Europa,di=ciudades".

El resultado es un mensaje del tipo "ha ocurrido un error durante la operación" y se muestra el código de error
[LDAP: error code 50-Insufficient Access Rights] que indica que el usuario no dispone de privilegios para realizar esa operación.

El proxy muestra una excepción del tipo
"javax.Naming.NoPermissionException".

PRUEBA 76: Cambiar el nombre de una entrada que tenga hijos. Para ello se repite la prueba 75, pero esta vez accediendo como administrador del sistema.

Se obtiene un mensaje de "operación realizada con éxito". La entrada ha sido renombrada.

PRUEBA 77: Se buscan todas las entradas que cuelgan de "co=Europa,di=ciudades".

Aparece la entrada "pa=Germany", y las entradas que colgaban de "pa=Alemania".

PRUEBA 78: Se buscan todas las entradas que cuelgan de "pa=Germany,co=Europa,di=ciudades".

Ahora no se recibe ninguna entrada, ya que tras el cambio de nombre de la prueba 76 se produjo una situación de inconsistencia en el árbol por haber renombrado una entrada con hijos, ya que ahora estos hijos no cuelgan de la entrada.

PRUEBA 79: Se repite la prueba 76 pero esta vez intercambiando los nombres para devolver el directorio a su estado original.

Se obtiene un mensaje de “operación realizada con éxito”. La entrada ha sido renombrada.

PRUEBA 80: Se repiten las pruebas 78 y 79

Ahora todas las entradas son devueltas y el árbol vuelve a ser consistente.

PRUEBA 81: Se repite la prueba 76, pero en esta ocasión se intenta dar como nombre una cadena vacía.

El propio cliente no permite realizar la operación.

PRUEBA 82: Se repite la prueba 76, pero en esta ocasión se da como nombre de la entrada a cambiar una cadena vacía.

El propio cliente no permite realizar la operación.

PRUEBA 83: Se intenta renombrar una entrada que no existe. Para ello accediendo como administrador se intenta renombrar la entrada “ci=Atenas,pa=Grecia,co=Europa,di=ciudades”(que no existe) por el nombre “ci=Atenass,Grecia,co=Europa,di=ciudades”.

Se muestra por pantalla que “ha ocurrido un error durante la operación”, y el código de error [LDAP: error code 32-No such Object], indicando que la entrada no existe.

En el proxy se muestra una excepción del tipo “javax.naming.NameNotFound

PRUEBA 84: Renombrar una entrada para que cuelgue de otra entrada que no existe. Para ello accediendo como administrador se cambia el nombre de la entrada "pa=Alemania,co=Europa,di=ciudades" por "pa=Alemania,di=directorio" (la entrada di=directorio no existe).

Se muestra por pantalla que "ha ocurrido un error durante la operación", y el código de error [LDAP: error code 71 - cannot rename between DSAa], indicando que no puede trasladar la entrada fuera del árbol de directorio.

El proxy muestra una excepción del tipo "javax.naming.namingException".

PRUEBA 85: Renombrar una entrada hoja y que siga manteniendo su ubicación en el árbol. Para ello accediendo como administrador se cambia el nombre de la entrada "ci=Berlin,pa=Alemania,co=Europa,di=ciudades" por "ci=Berlin2, ,pa=Alemania,co=Europa,di=ciudades".

Se obtiene un mensaje de "operación realizada con éxito". La entrada ha sido renombrada.
Si se lista las entradas que cuelgan de "pa=Alemania,co=Europa,di=ciudades" se comprueba que el cambio efectivamente tuvo éxito.

PRUEBA 86: Cambiar una entrada fuera de su contexto. Para ello accediendo como administrador se cambia la entrada "ci=Berlin2, ,pa=Alemania,co=Europa,di=ciudades" por "ci=Berlin2,pa=Egipto,co=Africa,di=ciudades", de manera que ahora la entrada cuelga de nodo "pa=Egipto,co=Africa,di=ciudades".

Se obtiene un mensaje de "operación realizada con éxito". La entrada ha sido renombrada. Si se listan las entradas que cuelgan de "pa=Alemania,co=Europa,di=ciudades" se no aparece "Berlin2", en cambio si se lista "pa=Egipto,co=Africa,di=ciudades", sí aparece, confirmando así que la operación tuvo éxito.

PRUEBA 87: Cambiar de contexto una entrada que tenga hijos. Para ello accediendo como administrador se cambia el nombre de la entrada "pa=Alemania,co=Europa,di=ciudades" por "pa=Alemania,co=Africa,di=ciudades", pasando la entrada a colgar de "co=Africa,di=ciudades".

Se obtiene un mensaje de "operación realizada con éxito". La entrada ha sido renombrada.

Si se listan las entradas que cuelgan de "co=Africa,di=ciudades" aparece "pa=Alemania", pero de esta última entrada no cuelgan los hijos que colgaban anteriormente, con lo cual la operación ha provocado un estado de inconsistencia en el árbol.

PRUEBA 88: El administrador restaura por completo el directorio para que esté como cuando se comenzó a realizar pruebas. Para ello borra los ficheros de la base de datos LDBM y vuelve a usar la herramienta "ldapadd" para volver a añadir las entradas del fichero LDIF.

La operación tiene éxito.

11.8.9 Pruebas con la operación SEARCH.

PRUEBA 89: Un usuario anónimo realiza una búsqueda. Para ello solicita todas las entradas que cuelgan de "pa=Italia,co=Europa,di=ciudades".

Se obtiene un mensaje de "operación realizada con éxito". Se muestra el contenido de la propia entrada "pa=Italia", y las entradas "ci=Roma", "ci=Florencia", "ci=Milan".

PRUEBA 90: Se realiza una búsqueda sin dar un objeto base desde el que comenzar la búsqueda. Para ello el administrador usa un filtro del tipo "objectclass=*", pero en el nombre de la entrada base se pasa una cadena vacía.

Se muestra por pantalla que “ha ocurrido un error durante la operación”, y el código de error [LDAP: error code 32-No such Object], indicando que la entrada no existe.

En el proxy se muestra una excepción del tipo “javax.naming.NameNotFound”

PRUEBA 91: Se da una entrada base no válida. Se repite la operación 90, pero dando como nombre de la entrada base “pa=Grecia,co=Europa,di=ciudades” (que no existe).

El resultado es el mismo que en la prueba 90.

PRUEBA 92: Se realiza una búsqueda sin dar un filtro de búsqueda. El administrador intenta buscar todas las entradas que cuelgan de la base “co=America,di=ciudades”, pero da como filtro de búsqueda una cadena vacía.

Se muestra por pantalla que “ha ocurrido un error durante la operación”, y el mensaje “empty filter”, que indica que se ha dado un filtro vacío.

El proxy muestra una excepción del tipo “javax.naming.directory.InvalidSearchFilterException”.

PRUEBA 93: Uso de un filtro de búsqueda no válido. Se repite la prueba 92 pero usando un filtro del tipo “objectclass--*” (sintaxis no válida).

El resultado es igual al de la prueba 92, pero en esta ocasión el mensaje es “missing equals”, ya que el error viene por falta del operador “=”.

PRUEBA 94: Se realiza una búsqueda en un subárbol. El administrador lista todas las entradas (objectclass=*) que cuelgan de la entrada “co=America,di=ciudades” en un ámbito de búsqueda del tipo subárbol (subtree).

Se obtiene un mensaje de “operación realizada con éxito”. Se reciben once entradas que son todas las que componen el subárbol que cuelga de esa entrada y se muestran todos sus atributos.

PRUEBA 95: Se realiza una búsqueda en la que se solicitan atributos concretos pero sin valores. Para ello se repite la prueba 94, pero se pasan como atributos a devolver "moneda" y "capital". También se pide que no se devuelvan valores.

Se obtiene un mensaje de "operación realizada con éxito". Se reciben once entradas que son todas las que componen el subárbol que cuelga de esa entrada, pero en esta ocasión sólo se muestran los atributos solicitados y sin valores.

PRUEBA 96: Se buscan todos los objetos del árbol. Para ello el administrador da como base la raíz del árbol (di=ciudades), el filtro "objectclass=*", y especifica que el ámbito de búsqueda es el subárbol.

Se obtiene un mensaje de "operación realizada con éxito". Se reciben cincuenta y dos entradas que son todas las que componen el árbol y se muestran todos sus atributos.

PRUEBA 97: Se realiza una búsqueda bajo ciertas condiciones. El administrador solicita todas las entradas del árbol que tengan un atributo "habitantes" con un valor comprendido entre cincuenta y cien millones y que además tenga el euro como moneda. Para ello da como base la raíz del árbol (di=ciudades), un ámbito de búsqueda de subárbol y el siguiente filtro de búsqueda "(&(&(habitantes>=50000000)(habitantes<=100000000))(moneda=euro)).

Se obtiene un mensaje de "operación realizada con éxito". Se reciben tres entradas: "pa=Italia,co=Europa,di=ciudades",
"pa=Alemania,co=Europa,di=ciudades" y
"pa=Francia,co=Europa,di=ciudades".
La búsqueda ha devuelto las entradas correctas.

PRUEBA 98: Uso del control de tiempo máximo de realización de la búsqueda. Se repite la prueba 96 pero ajustando este control a 50 milisegundos.

El resultado es el mismo que en la prueba 96.

PRUEBA 99: Uso del control de límite de entradas. Se repite la prueba 96 pero limitando el número de entradas a devolver a cinco.

Se muestra por pantalla que "ha ocurrido un error durante la operación", y el código de error [LDAP:error code 4 - Sizelimit Exceeded], que indica que el límite de entradas ha sido rebasado.

El proxy muestra una excepción del tipo "javax.naming.SizeLimitExceededException".

PRUEBA 100: Realizar una búsqueda en un ámbito de un nivel inferior a la base. Se repite la prueba 96 pero usando un ámbito del tipo un nivel.

Se obtiene un mensaje de "operación realizada con éxito". Se reciben tres entradas: "co=America,di=ciudades", "co=Europa,di=ciudades", "co=Africa,di=ciudades", "co=Asia,di=ciudades" y "co=Oceania,di=ciudades".

PRUEBA 101: Realizar una búsqueda en el ámbito de la propia entrada base. Se repite la prueba 100 pero usando un ámbito de búsqueda basado en la propia entrada.

Se obtiene un mensaje de "operación realizada con éxito". Se recibe una única entrada, que es la propia raíz.

11.8.10 Pruebas con solicitudes de información al servidor.

PRUEBA 102: Consulta de los controles del servidor. Un usuario anónimo consulta los controles del servidor.

Se obtiene un mensaje de "operación realizada con éxito", y se muestra:

- suportedControl=2.16.840.1.113730.3.4.2 que es el control que permite manejar remisiones.

PRUEBA 103: Un usuario anónimo consulta las extensiones soportadas por el servidor.

Se obtiene un mensaje de "operación realizada con éxito", y se muestra**:**

- supportedExtension= 1.3.6.1.4.1.4203.1.11.1, que permite cambiar la identidad del usuario durante la conexión.

- supportedExtension= 1.3.6.1.4.1.4266.20037, que permite activar el protocolo TLS.

PRUEBA 104: Un usuario anónimo consulta los mecanismos de seguridad soportados por el servidor.

Se obtiene un mensaje de "operación realizada con éxito", y se muestra**:**

- supportedSASLMechanisms=GSSAPI.

11.8.11 Pruebas con alias.

PRUEBA 105: Añadir un alias a una entrada. El administrador para ello añade a la entrada "pa=Argentina,co=America,di=ciudades" un alias hacia la entrada "pa=Brasil,co=America,di=ciudades". Se realiza mediante la operación "modify".

Se obtiene un mensaje de "operación realizada con éxito". Las modificaciones han sido realizadas.
Si se lista la entrada se comprueba que el alias ha sido añadido.

PRUEBA 106: Añadir un alias hacia una entrada que no existe. El administrador añade a la entrada "pa=Brasil,co=America,di=ciudades" un alias hacia la entrada "pa=Grecia,co=Europa,di=ciudades" (que no existe).

El resultado es el mismo que el de la prueba 105.

PRUEBA 107: Seguir un alias. Se realiza una búsqueda partiendo de "co=America,di=ciudades" y se piden todos los objetos que cuelgan de ella. Los alias deben ser seguidos siempre.

Se obtiene un mensaje de "operación realizada con éxito". Se devuelven todas las entradas excepto "ci=Buenos, Aires,pa=Argentina,co=America,di=ciudades" y la propia "pa=Argentina".

PRUEBA 108: Seguir un alias que apunta a una entrada no existente. Para ello se lista el contenido de la entrada "pa=Brasil,co=America,di=ciudades" siguiendo el alias que contiene.

Se muestra por pantalla que "ha ocurrido un error durante la operación", y el código de error [LDAP: error code 32-No such Object], indicando que la entrada no existe.

En el proxy se muestra una excepción del tipo "javax.naming.NameNotFound"

PRUEBA 109: Añadir más de un alias a una entrada. Se añade un nuevo alias a la entrada "pa=Argentina,co=America,di=ciudades" apuntando en esta ocasión a "co=Asia,di=ciudades".

Se obtiene un mensaje de "operación realizada con éxito". Las modificaciones han sido realizadas.
Si se lista la entrada se comprueba que contiene dos alias.

PRUEBA 110: Listar una entrada que contiene más de un alias siguiendo los alias. Se lista el contenido de la entrada "pa=Argentina,co=America,di=ciudades" siguiendo los alias.

Se muestra por pantalla que "ha ocurrido un error durante la operación", y el código de error
[LDAP: error code 33- alias has multivaluedaliasedObjectName**],** indicando que el objeto contiene más de un alias.

El proxy muestra una excepción del tipo "javax.naming.namingException".

PRUEBA 111: Eliminar el segundo alias de la entrada "pa=Argentina,co=America,di=ciudades".

Se obtiene un mensaje de "operación realizada con éxito". Las modificaciones han sido realizadas.

PRUEBA 112: Se repite la prueba 107, pero en esta ocasión sólo se sigue el alias mientras se posiciona en la entrada base.

Se obtiene un mensaje de "operación realizada con éxito". Se devuelven todas las entradas incluidas "ci=Buenos, Aires,pa=Argentina,co=America,di=ciudades" y la propia "pa=Argentina", ya que en esta ocasión el alias no se desreferencia.

PRUEBA 113: Se repite la prueba 107, pero en esta ocasión sólo se siguen los alias que no pertenezcan a la entrada base.

Se obtiene un mensaje de "operación realizada con éxito". Se devuelven todas las entradas excepto "ci=Buenos, Aires,pa=Argentina,co=America,di=ciudades" y la propia "pa=Argentina", ya que en esta ocasión el alias se desreferencia.

PRUEBA 114: Formar un bucle de alias. Reemplazar el atributo "alias" de "pa=Argentina,co=America,di=ciudades" por "pa=Mejico,co=America,di=ciudades", y añadir a la entrada "pa=Mejico,co=America,di=ciudades" un alias hacia "pa=Argentina,co=America,di=ciudades", de manera que se forma un bucle.

Se obtiene un mensaje de "operación realizada con éxito". Las modificaciones han sido realizadas. Se ha permitido añadir un bucle.

PRUEBA 115: Entrar en un bucle de alias. Para ello se lista el contenido de la entrada "pa=Mejico,co=America,di=ciudades" siguiendo los alias.

Se muestra por pantalla que "ha ocurrido un error durante la operación", y el código de error [LDAP: error code 33- circular alias], indicando que se ha entrado en un bucle de alias.

11.8.12 Pruebas con remisiones.

PRUEBA 116: Añadir una remisión dentro del mismo árbol. Se añade mediante la operación "modify" una remisión a la entrada "pa=Italia,co=Europa,di=ciudades" que apunte hacia "pa=Rusia,co=Europa,di=ciudades".

Se obtiene un mensaje de "operación realizada con éxito". Las modificaciones han sido realizadas.

PRUEBA 117: Recuperar un atributo remisión. Este atributo es operacional y sólo es devuelto si se solicita explícitamente, para lo cual se lista la entrada "pa=Italia,co=Europa,di=ciudades" pidiendo que sea devuelto el atributo "ref".

Se obtiene un mensaje de "operación realizada con éxito". Se muestra la entrada solicitada y sólo el atributo "ref" y su valor "ldap://localhost:389/pa=Rusia,co=Europa,di=ciudades".

PRUEBA 118: Seguir una referencia. Se lista el contenido de la entrada "pa=Italia,co=Europa,di=ciudades" indicando que se sigan las referencias.

Se obtiene un mensaje de "operación realizada con éxito". Se muestra el contenido de la entrada "pa=Rusia,co=Europa,di=ciudades".

PRUEBA 119: Devolver las referencias. Se lista el contenido de la entrada "pa=Italia,co=Europa,di=ciudades" indicando que las referencias deben devolverse.

Se obtiene un mensaje de "operación realizada con éxito". Se muestra la referencia "ldap://localhost:389/pa=Rusia,co=Europa,di=ciudades".

PRUEBA 120: Se añade una segunda referencia a la entrada "pa=Italia,co=Europa,di=ciudades". En esta ocasión la referencia apunta a "pa=China,co=Asia,di=ciudades".

Se obtiene un mensaje de "operación realizada con éxito". Las modificaciones han sido realizadas.

PRUEBA 121: Se repite la prueba 118.

El resultado es el mismo que el de la prueba 118, ya que la primera referencia que encuentra es la que sigue. (La que apunta a “pa=Rusia”).

PRUEBA 122: Se repite la prueba 119.

Se obtiene un mensaje de “operación realizada con éxito”. Se muestra la referencia “ldap://localhost:389/pa=Rusia,co=Europa,di=ciudades” y la referencia “ldap://localhost:389/pa=China,co=Asia,di=ciudades”.

PRUEBA 123: Se añade una remisión a un objeto que no existe. Se añade una remisión a la entrada “co=Africa,di=ciudades” a la entrada “pa=Grecia,co=Europa,di=ciudades” (que no existe).

Se obtiene un mensaje de “operación realizada con éxito”. Las modificaciones han sido realizadas.

PRUEBA 124: Seguir una remisión que apunta a un destino inexistente. Se lista el contenido de la entrada “co=Africa,di=ciudades” siguiendo las remisiones.

Se muestra por pantalla que “ha ocurrido un error durante la operación”, y el código de error [LDAP: error code 32-No such Object], indicando que la entrada no existe.

En el proxy se muestra una excepción del tipo “javax.naming.NameNotFound”.

PRUEBA 125: Formar un bucle de remisiones. Se añade una remisión a la entrada “pa=Rusia,co=Europa,di=ciudades” que apunte a “pa=Italia,co=Europa,di=ciudades”.

Se obtiene un mensaje de “operación realizada con éxito”. Las modificaciones han sido realizadas.

PRUEBA 126: Entrar en un bucle de remisiones. Se lista el contenido de la entrada "pa=Rusia,co=Europa,di=ciudades" siguiendo las remisiones que se encuentren.

Se muestra por pantalla que "ha ocurrido un error durante la operación", y aparece el mensaje "referral limit exceeded" que indica que se ha superado el límite de remisiones que se pueden desreferenciar en una búsqueda.

PRUEBA 127: Formar un bucle con un alias y una remisión. Se reemplaza el alias de la entrada "pa=Mejico,co=America,di=ciudades" para que apunte a "pa=Rusia,co=Europa,di=ciudades", y la remisión de esta segunda entrada para que apunte a la primera.

Se obtiene un mensaje de "operación realizada con éxito". Las modificaciones han sido realizadas.

PRUEBA 128: Entrar en un bucle formado por una remisión y un alias. Se lista el contenido de la entrada "pa=Rusia,co=Europa,di=ciudades" siguiendo los alias y las remisiones.

El resultado es el mismo que el de la prueba 126.

PRUEBA 129: Se arranca un segundo servidor que funciona en la misma máquina sobre el puerto 489. En ese servidor se añade un segundo árbol de directorio cuyo fichero LDIF puede consultarse en el apartado Anexos.

La prueba se realiza con éxito.

PRUEBA 130: Añadir una referencia a otro directorio. Para ello se añade una remisión a la entrada "co=Europa,di=ciudades" hacia la entrada "ou=Sevilla,o=miempresa.es" que es una entrada del directorio almacenado en el servidor que se inició en la prueba 129.
La remisión se codifica como "ldap://localhost:489/ou=Sevilla,o=miempresa.es".

Se obtiene un mensaje de "operación realizada con éxito". Las modificaciones han sido realizadas.

PRUEBA 131: Seguir una referencia que apunte hacia una entrada de otro directorio. Se lista el contenido de la entrada "co=Europa,di=ciudades" siguiendo las remisiones.

Se obtiene un mensaje de "operación realizada con éxito". Se muestra el contenido de la entrada "ou=Sevilla,o=miempresa.es" contenida en el servidor que se inició en la prueba 129.

11.8.13 Pruebas de acceso a la información de usuarios.

PRUEBA 132: Un usuario que no sea el administrador consulta la información de la entrada que representa a otro usuario. Para ello el usuario "cn=usu1,di=usuarios,di=ciudades" lista la entrada "cn=usu3,di=usuarios,di=ciudades".

Se obtiene un mensaje de "operación realizada con éxito". Se muestra la información personal de usuario requerida excepto el atributo que contiene la clave de autentificación del usuario.

11.9 ANEXO 9: Soporte de SSL.

Como ya se ha señalado, unos de los objetivos que se han intentado alcanzar ha sido la implantación del protocolo SSL en el sistema.

Cuando se planteó esta posibilidad, el proyecto aún se encontraba en la fase de desarrollo de la parte del cliente LDAP del proxy. Se comprobó que el servidor OpenLDAP es capaz de implementar por sí mismo dicho protocolo, ya que puede arrancar sobre un puerto seguro SSL (por defecto el 636). Para verificar dicho comportamiento se utilizaron las herramientas que proporciona OpenLDAP para acceder al servidor con los parámetros necesarios para que el cliente realizase la conexión sobre SSL. Los resultados de este proceso fueron positivos, y se pensó que la conexión basada en LDAP podría implementarse sobre SSL de manera sencilla.

El estudio de JNDI mostró que el uso del protocolo SSL en comunicaciones LDAP es bastante común, hasta el punto la activación de este protocolo desde el cliente se realiza en JNDI con la configuración de una simple propiedad de entorno, siempre y cuando se dispongan de las clases capaces de implementar sockets SSL.

En esta fase se estaba usando el software Java 2 SDK 1.3 que no proporcionaba las librerías necesarias para implementar sockets SSL, por lo que se instaló la versión 1.4 que sí ofrecía al usuario dichas librerías. Esto llevó al desarrollo de un cliente LDAP que accedía al servidor usando LDAP sobre SSL, con lo que este tramo quedó convenientemente implementado.

El problema surgió con el uso de Jacorb, ya que la versión 1.4 beta 4 usada, así como las anteriores, presentan problemas de compatibilidad con la versión 1.4 de Java, con lo cual, tras múltiples intentos frustrados por intentar compatibilizar ambas distribuciones, hubo que descartar la versión 1.4 y volver a la 1.3. Esto llevó a la pérdida de la conexión sobre SSL en el tramo LDAP, que anteriormente había sido implementada.

Entre los intentos por compatibilizar Jacorb con Java 2 SDK 1.4 estuvo el intentar que Jacorb no usase hilos nativos, que al parecer eran la causa del problema. En este caso se consiguió arrancar el servidor CORBA sobre SSL, pero no así el cliente, ya que estaba desarrollado con métodos que hacían uso de hilos.

Otro de los intentos consistió en conseguir las fuentes de Jacorb y compilarlas con la versión 1.4 de Java para conseguir unos ficheros “class” construidos con la versión 1.4. Este proceso se llevó a cabo mediante la herramienta “ant”, y tras observarse las librerías generadas se vieron una serie de clases que implementaban sockets SSL y que la distribución descargada originalmente (ya compilada) no traía. Sin embargo esto no fue suficiente, y no resolvió los problemas de compatibilidad.

En el fichero de instalación de Jacorb se recomienda usar además de las librerías Java sobre SSL, una distribución llamada “IAIK” que tiene la misma finalidad. Esta librería fue descargada en su versión más reciente e instalada, pero los resultados obtenidos no fueron satisfactorios, ya que el uso de esta nueva distribución no eximía del uso de la versión 1.4 de Java. (Sí en el cliente CORBA pero no en el cliente LDAP).

Finalmente tuvo que descartarse el uso de SSL debido a estos problemas.

Destacar que la configuración de Jacorb para usar SSL es relativamente sencilla, ya que sólo hay que configurar en el fichero “jacorb.properties” una serie de propiedades en las que se especifican las librerías que se usan, las opciones deseadas (autentificación, uso de certificados, etc) y que se active el protocolo.

11.10 ANEXO 10: Control de calidad.

Durante el desarrollo del trabajo se han seguido una serie de criterios a fin de garantizar la calidad del producto final.

Estos criterios varían tanto en su naturaleza como en su ámbito de aplicación dependiendo de la fase del proyecto.

Durante la fase de desarrollo del código de las aplicaciones creadas se ha seguido un proceso de documentación de los algoritmos y operaciones efectuadas, persiguiendo así que el código sea autoexplicativo, y el lector pueda seguir en todo momento el desarrollo de la aplicación.

Una vez construido el sistema se han realizado pruebas en condiciones desfavorables (lo más posibles) buscando así obtener un código robusto capaz de seguir funcionando ante sistuaciones anómalas.

La batería de pruebas detallada en apartados anteriores no es más que una recopilación de un conjunto mucho más amplio de pruebas que se han realizado durante el proceso de desarrollo de los módulos del sistema y sobre el producto final, con el objeto de depurar el código y aumentar su fiablilidad y robustez, obteniéndose así un producto final que cumple con la finalidad de conseguir una aplicación fiable y eficiente.

Se ha tenido en cuenta la opinión de potenciales usuarios del producto a la hora de diseñar la interfaz del cliente, y durante su desarrollo, con el fin de asegurar un mayor grado de aceptación de la aplicación.

El cliente ofrece un sistema de ayuda básico pero suficiente para que el usuario pueda obtener información en línea sobre cómo usar la aplicación.

El conjunto de capítulos que recogen la tecnología usada de la presente memoria, han sido desarrollados además de con un espíritu descriptivo de la tecnología empleada, buscando dar una visión propia de un manual del uso de dicha tecnología. Con ello el lector puede usar la información aquí contenida como documentación, y como guía rápida de las tecnologías expuestas.

Destacar que el proxy responde al criterio de diseño seguido de búsqueda de independencia del resto del sistema y portabilidad. El proxy también es reusable con cualquier otro cliente CORBA que pudiera ser desarrollado.

12 BIBLIOGRAFÍA

- ***Common Objet Request Broker Architecture***. Es el estándar que recoge la nombra CORBA.

- ***The Java Tutorial. A practical guide for programmers***. Es el tutorial de Java de Sun. Disponible en "java.sun.com".

- ***Java 2 SDK standard Edition documentation.*** Documentación sobre el software Java 2 SDK, donde se explica el contendio de sus paquetes, clases, interfaces, etc.

- ***The JNDI tutorial.*** Es un tutorial completo, bien desarrollado y con abundantes ejemplos, en el que se puede aprender acerca de JNDI.

- ***The Java Naming and Directory Interface.*** Manual disponible en www.sun.com en el que describe el funcionamiento y la composición de JNDI.

- ***Understanding LDAP de IBM.*** Es un libro bastante completo que describe el protocolo LDAP. Esta disponible en http://www.redbooks.ibm.com/redbooks/SG244986.html

- ***OpenLDAP 2.0 Administrator´s guide.*** Es la guía del administrador de OpenLDAP, en la cual se recoge todo el funcionamiento de este servidor de directorio.

- ***RFCs de LDAP.*** Los estándares en los que se especifica el protocolo LDAP. Disponibles en ftp://ftp.rediris.es/docs/rfc/22xx/.

- ***Jacorb 1.4 Programming Guide.*** Es el manual de Jacorb. Disponible en www.Jacorb.org.

PLANOS

1. El fichero CORBA-LDAP.IDL

Este fichero se encuentra contenido en el directorio ***"corbaldap_definición".***

```
interface corbaldap
  {


    // nota mapea el tipo array de Strings de tamaño dos de java. Esta
    //estructura se usa para recibir información del
    // servidor referente al resultado de ciertas operaciones.

    typedef string nota[2];


    // tabla mapea el tipo array de Strings de java. En esta ocasión el /
    //tamaño del array en java puede ser cualquiera.
    // Este tipo de datos recoge información sobre operaciones de
    //busqueda principalmente.

    typedef sequence <string> tabla;


    // matriz mapea el tipo array de arrays de Strings de java, es decir es
    //una tabla bidimensional. Se usa para recoger
    // información que será enviada al servidor en la operación add.

    typedef sequence <tabla> matriz;



    // La excepción de usuario ExGeneral representa el tipo Exception en
    //java, de manera que cuando se captura una
    // excepción de este tipo en el servidor corba se obtiene su
    //información, se construye esta excepción con una tabla
    // de dos Strings donde va la información y se lanza hacia el cliente.


    exception ExGeneral{
          nota general;
```

```
};


// La excepción de usuario ExNombrado representa el tipo
//NamingException de JNDI, de manera que cuando se captura una
// excepción de este tipo en el servidor corba (lanzada por el servidor
//OpenLDAP) se obtiene su información, se construye esta excepción
//y se lanza
// hacia el cliente.


exception ExNombrado{
        nota nombrado;
};



// La función compare de LDAP permite ver si una entrada tiene una
//pareja atributo valor determinado. Recibe parametros
// sobre la autentificación, el nombre de la entrada, la pareja
//atributo=valor.
// Devuelve una tabla en la que los dos primeros elementos informan
//sobre el éxito y el resultado de
// la operación.


tabla compare(in string autentif_metodo, in string autentif_login, in string
        autentif_clave, in string nodo, in string pareja_valor)
        raises(ExGeneral,ExNombrado);



// La función add de LDAP permite añadir una nueva entrada al DIT.
//Recibe información acerca de la autentificación,
// el RDN de la nueva entrada y una serie de atributos y valores, que
//al menos deben cumplir las exigencias del
// esquema. Devuelve un mensaje informando sobre el éxito y el
//resultado de la operación.
```

nota add(in string autentif_metodo, in string autentif_login, in string autentif_clave, in string nuevo_nodo, in matriz atributos_iniciales) raises(ExGeneral,ExNombrado);
// La función delete de LDAP permite borrar una entrada del DIT. //Recibe información sobre la autentificación y el
// RDN de la entrada a eliminar. Devuelve un mensaje informando //sobre el éxito y el resultado de la operación.

nota delete (in string autentif_metodo, in string autentif_login, in string autentif_clave, in string borrar_nodo)
raises(ExGeneral,ExNombrado);

// La función modify_RDN de LDAP permite cambiar el RDN de //una entrada. Recibe información sobre la autentificación,
// el RDN de la entrada, el nuevo RDN, y una indicació sobre si //conservar el antiguo RDN como un atributo. Devuelve
// un mensaje informando sobre el éxito y el resultado de la //operación.

nota modify_rdn (in string autentif_metodo, in string autentif_login, in string autentif_clave, in string nombre_antiguo, in string nombre_nuevo, in string conservo_antiguo)
raises(ExGeneral,ExNombrado);

// La función modify de LDAP permite realizar operaciones sobre los //atributos de una entrada (añadir, borrar, reemplazar).
// Recibe información de autentificación, la entrada a modificar, y //una tabla en la que se pasan los atributos, los
// valores y la operación a realizar. También se pasa el número de //operaciones. Devuelve un mensaje informando sobre el éxito y el //resultado de la operación.

nota modify (in string autentif_metodo, in string autentif_login, in string autentif_clave, in string nodo, in tabla modificaciones, in short numero_operaciones) raises(ExGeneral,ExNombrado);

// La función search de LDAP permite realizar búsquedas en el DIT.
//Recibe información de autentificación, el nodo
// desde el que iniciar la búsqueda, una expresión lógica (String) con
//los parámetros de la búsqueda. Los atributos a
// devolver, el nivel de la búsqueda, el número máximo de entradas a
//devolver, el tiempo máximo de espera. Si
// devuelve sólo los identificadores de atributos (sin valores), y que
//hacer con alias y remisiones.
// El resultado es una tabla con toda la información de la búsqueda.

tabla search (in string autentif_metodo, in string autentif_login, in string autentif_clave, in string nodo_raiz, in string filtro, in tabla atributos_devolver, in string nivel, in short limite_nodos, in short limite_tiempo, in string solo_identf, in string aliass, in string remision) raises(ExGeneral,ExNombrado);

// Listarcontroles hace una consulta al servidor sobre los controles
//LDAP que soporta.

tabla listarcontroles (in string autentif_metodo, in string autentif_login, in string autentif_clave) raises(ExGeneral,ExNombrado);

// Listarextensiones hace una consulta al servidor sobre las
//extensiones LDAP que soporta.

tabla listarextensiones (in string autentif_metodo, in string autentif_login, in string autentif_clave) raises(ExGeneral,ExNombrado);

// Listarsasl hace una consulta al servidor sobre los macanismos de //autentificación SASL que soporta.

tabla listarsasl (in string autentif_metodo, in string autentif_login, in string autentif_clave) raises(ExGeneral,ExNombrado);

// Inf_clases_entrada recibe una entrada y permite hacer consultas de // esquema de diversa índole (sobre clases objeto, // sobre sintaxis de atributos, tipos de atributos).

tabla inf_objeto_esquema (in string autentif_metodo, in string autentif_login, in string autentif_clave,in string entrada_esquema) raises(ExGeneral,ExNombrado);

// Inf_objeto_esquema recibe una entrada y devuelve información de // esquema sobre las clases objeto de la entrada.

tabla inf_clases_entrada (in string autentif_metodo, in string autentif_login, in string autentif_clave, in string entrada_esquema) raises(ExGeneral,ExNombrado);

// listarat es una variante de la función search. Permite listar los // atributos de una entrada dada.

tabla listarat (in string autentif_metodo, in string autentif_login, in string autentif_clave, in string nodo_raiz, in tabla atributos_devolver, in string aliass, in string remision, in string solo_identf) raises(ExGeneral,ExNombrado);

};

2. Empaquetado del proxy.

El proxy diseñado se compone de las siguientes clases, englobadas en el paquete ***"corbaldap_servidor".***

../ corbaldap_servidor

- clservidor.java
- corbaldapImpl.java

3. Empaquetado del cliente.

El cliente diseñado se compone de las siguientes clases, englobadas en el paquete ***"corbaldap_cliente".***

../ corbaldap_cliente

- addresult.java
- autentificación.java
- cliente.java
- compresult.java
- deleresult.java
- esqueresult.java
- fadd.java
- fcompare.java
- fdelete.java
- fesquema.java
- finformador.java
- flistar.java
- fmodify.java
- fmodifyrdn.java
- fsearch.java
- inforesult.java
- listresult.java
- modiresult.java
- mrdnresult.java
- searresult.java

PLIEGO DE CONDICIONES

1. Requisitos de usuario.

La simplicidad del sistema hace que el usuario no necesite tener ningún tipo de conocimiento de CORBA, aunque deberá estar familiarizado con el uso de LDAP, ya que para realizar operaciones necesitará introducir los nombres de las entradas siguiendo la convención de nombrado de LDAP.
El usuario debe conocer también el tipo de información al que accede.

Es deseable que el usuario tenga conocimientos básicos de informática a nivel de usuario y esté familiarizado con el manejo de ventanas, botones y menús, resultando así más sencillo el manejo de la aplicación.

El menú de ayuda en línea que ofrece el cliente CORBA proporciona al usuario un soporte adicional a la hora de usar la aplicación, reduciendo aún más los conocimientos previos que debe tener el usuario.

En cuanto a acceso al sistema es conveniente que dicho usuario esté registrado de alguna manera por parte del administrador y tenga en el directorio una entrada que lo represente, con el fin de poder acceder al sistema usando autentificación simple.

2. Requisitos del sistema.

La instalación del sistema puede realizarse de manera local, usando para ello un solo equipo.

El escenario ideal es el uso de cuatros equipos conectados mediante una red local tal como se expuso en la memoria, aunque este comportamiento puede simularse con dos equipos tal como se expuso en el apartado de pruebas.

El uso de tres equipos es una solución intermedia igualmente válida.

La instalación se ha realizado en la red local del laboratorio, usando para ello ciertos equipos que se recomiendan para la instalación del sistema en otras redes. Estos equipos son: PC con procesador Pentium IV a 1.5 Ghz, con 128 MB de memoria RAM y 20 GB de disco duro.

Este tipo de equipos garantiza un comportamiento ideal del sistema, aunque los requerimientos pueden ser relajados, siendo el es escenario más

desfavorable recomendado un PC con procesador con una velocidad de 600Mhz, al menos 64 MB de memoria RAM, y una capacidad de disco duro mínima recomendada de 2 GB en el que puedan ser instalados tanto el sistema operativo como los programas del sistema y el software necesario para la instalación y funcionamiento de dicho sistema.

El sistema operativo utilizado ha sido Linux, condición indispensable al menos para el equipo que contenga el servidor de directorio OpenLDAP.

Se recomienda usar Linux en el resto de equipos, aunque bien podría usarse Windows, siempre y cuando el software usado (principalmente Java 2 SDK) sea el apropiado para esa plataforma. El software del proyecto está desarrollado en Java, con lo cual es independiente del tipo de plataforma usada.

La distribución de Linux usada para la realización del proyecto, y por tanto recomendada para posteriores instalaciones, es “ Linux Red Hat 7.1 ”.
Se recomienda trabajar en el entorno gráfico KDE que ha ofrecido durante la realización mejores prestaciones que GNOME.

PRESUPUESTO

Presupuesto.

Para la realización del presupuesto se indican las tareas realizas a lo largo del mismo junto con su coste económico y en tiempo.

El coste económico se estima a partir de la cualificación profesional requerida para la realización de la tarea.

En este caso se distinguen cuatro tipos de cualificaciones y se establece un salario aproximado para cada una de ellas.

- ***Ingeniero sénior***. Ingeniero superior con experiencia, responsable de las tareas de planificación y diseño, así como de la toma de decisiones estratégicas.

- ***Ingeniero júnior***. Ingeniero superior sin experiencia, al que se le delimita el trabajo a realizar, pero aún toma decisiones de implementación.

- ***Ingeniero técnico***. Encargado de las tareas de codificación, depuración y ejecución de pruebas.

- ***Administrativo.*** Responsable de las tareas de documentación.

CUALIFICACIÓN	*SALARIO MENSUAL*	*SALARIO DIARIO*
Ingeniero Superior de Telecomunicaciones sénior.	2.160 €	108 €
Ingeniero Superior de Telecomunicaciones júnior.	1.560 €	78 €
Ingeniero Técnico de Telecomunicaciones.	1.320 €	66 €
Administrativo.	960 €	48 €

Las cantidades de la tabla anterior han sido estimadas bajo una dedicación al cien por cien, durante ocho horas de trabajo diarias, y veinte días de trabajo mensuales.

Las tareas y sus costes se desglosan a continuación. Destacar que el coste empleado en los equipos y software se considera nulo, ya que los equipos usados son del laboratorio y el software puede ser gratuitamente descargado por cualquier usuario.

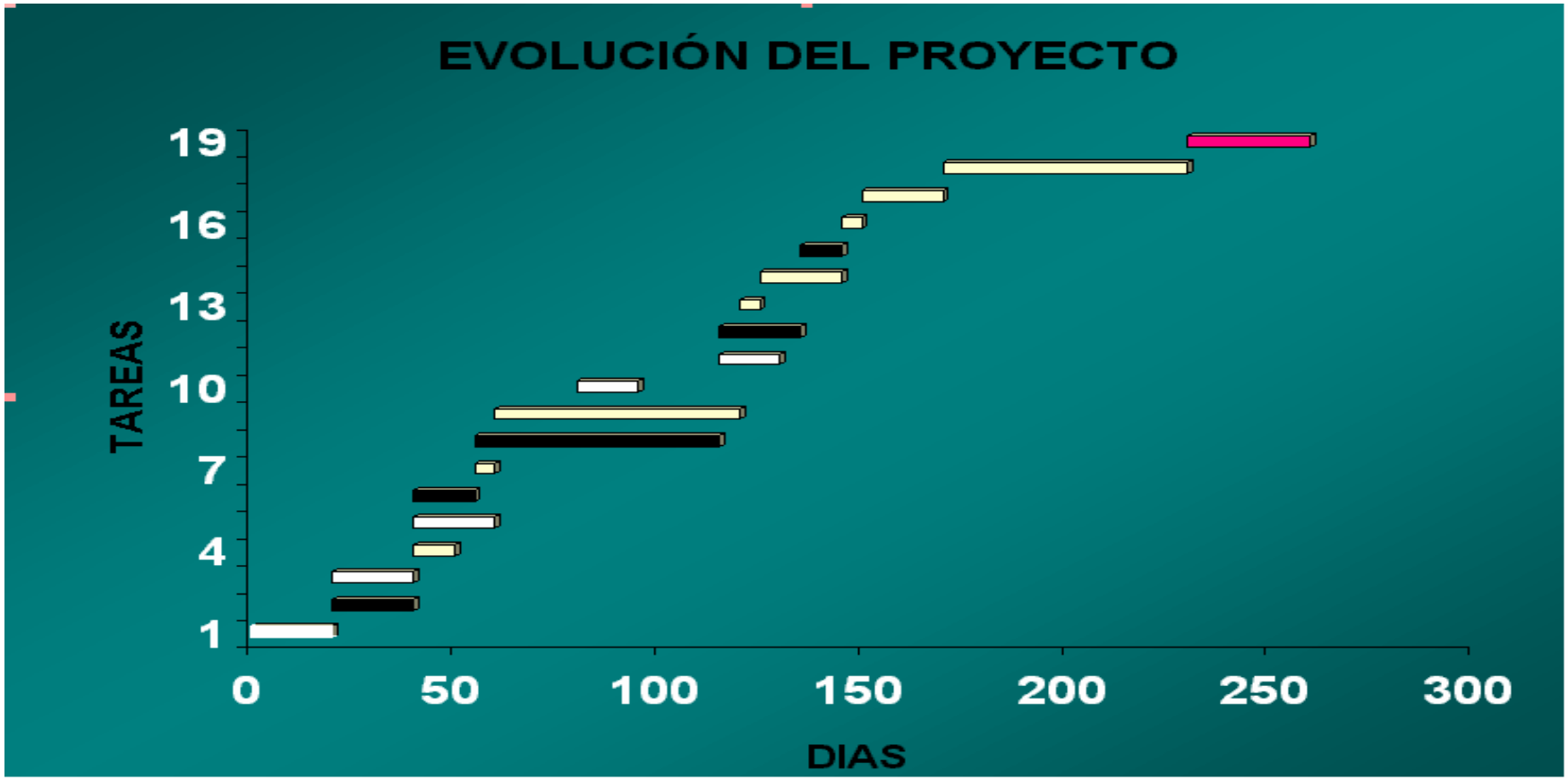
Por último se adjunta un diagrama en el que se presenta gráficamente la duración de las tareas a lo largo del tiempo, como éstas se solapan, y cómo se relacionan.

TAREA	**CUALIFICACIÓN**	**DÍAS**	**COSTE**
1. Planificación global.	Sénior	20	2.160 €
2. Decisión del software y hardware a usar.	Júnior	20	1.560 €
3. Diseño del proxy CORBA-LDAP.	Sénior	20	2.160 €
4. Instalación del software.	Técnico	10	660 €
5. Diseño del cliente CORBA.	Sénior	20	2.160 €
6. Diseño del servicio de directorio.	Júnior	15	1.170 €
7. Instalación del servicio de directorio.	Técnico	5	330 €

8. Desarrollo del proxy CORBA-LDAP	Júnior	60	4.680 €
9. Codificación del proxy CORBA-LDAP.	Técnico	60	3.960 €
10. Apoyo y toma de decisiones durante la realización.	Sénior	15	1.620 €
11. Supervisión del trabajo y control de calidad.	Sénior	15	1.620 €
12. Desarrollo del cliente CORBA.	Júnior	20	1.560 €
13. Instalación del proxy CORBA-LDAP.	Técnico	5	330 €
14. Codificación del cliente CORBA.	Técnico	20	1.320 €
15. Diseño de una batería de pruebas.	Júnior	10	780 €
16. Instalación del cliente CORBA.	Técnico	5	330 €
17. Realización de pruebas y depuración de errores.	Técnico	20	1320 €

18. Elaboración de la documentación del proyecto.	Técnico	60	3.960 €
19. Elaboración de la documentación definitiva del proyecto.	Administrativo	30	1.440 €

COSTE TOTAL DEL PROYECTO (sin IVA)..... 33.120 €
COSTE TOTAL (IVA)..... 38.419,20 €



	Sénior
	Júnior
	Técnico
	Aministrativo