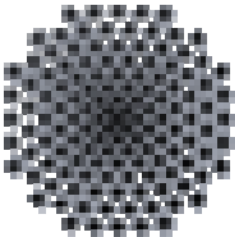
SEMINARAUSARBEITUNG

LDAP – APIs und Clients

im Rahmen des
Hauptseminars Verzeichnisdienste



Universität Stuttgart
WS 2000/2001

Michael Wenig
1842238

INHALTSVERZEICHNIS

Einleitung

Diese Ausarbeitung bezieht sich auf einen gleichnamigen Vortrag im Rahmen des Hauptseminars "Verzeichnisdienste" der Abteilung Betriebssysteme der Fakultät Informatik an der Uni-Stuttgart im Wintersemester 2000/2001.

Diese Ausarbeitung baut auf den Ausarbeitungen zu "LDAP – Einführung und Überblick", sowie "LDAP – detaillierter Überblick" auf. Aus diesen Ausarbeitungen werden die grundlegenden Konzepte, der Aufbau von LDAP, sowie die hier verwendeten Begriffe vorausgesetzt.

Diese Ausarbeitung stützt sich nun auf die technische Realisierung der Client-Seite. Hierbei wird zunächst kurz auf fertige Clients eingegangen und anschließend als Hauptteil die Benutzung einer spezielle API (Application Programming Interface), der Java-API von Netscape, vorgestellt.
Zum Schluss werden in einem kurzen Vergleich zu einer API für die Programmiersprache C die grundlegenden Unterschiede zwischen einer objektorientierten und einer Handle-basierten API herausgearbeitet.

1. Verfügbarkeit

Um LDAP auf der Client-Seite verwenden zu können, wird entweder eine fertige Client-Anwendung oder eine API (Application Programming Interface) benötigt. Da LDAP sich mittlerweile zu einem ernstzunehmenden Standard für den Zugriff auf Verzeichnisdienste etabliert hat, sind Clients und APIs für praktisch alle verfügbaren Plattformen (Microsoft, Solaris, Linux, Netware, BeOS, Apache, ...) und Sprachen (Java, C, Perl, TCL, ColdFusion, PHP, ...) vorhanden.

2. Clients

LDAP-Clients sind fertige Programme, die verwendet werden können, um auf Informationen in einem LDAP-Verzeichnisdienst zuzugreifen. Je nach Client beinhaltet es die Abfrage von Informationen, die Manipulation der Inhalte und/oder die Administration des Dienstes.

2.1 Benutzerverwaltung

Eine gängige Verwendung für einen Verzeichnisdienst ist die Verwaltung von Benutzern, Passwörtern und Berechtigungen für ein Netzwerk. Hierbei werden die Benutzerbezogenen Daten im Verzeichnis gespeichert und die Authentifizierung eines Benutzers durch einen Vergleich der vom Benutzer eingegebenen Daten mit den im Verzeichnis gespeicherten Daten durchgeführt.

Zu diesem Zweck existieren für praktisch jedes Betriebssystem entsprechende Konzepte um die Benutzeranmeldung über einen Verzeichnisdienst durchzuführen. Dabei ist die Integration natürlich je nach Betriebssystem mehr oder weniger aufwendig.

2.2 Firmen-Telefonverzeichnis

Eine weitere gängige Verwendung ist die Bereitstellung von Personen- und Kontakt-Informationen innerhalb einer Firma. An dieser Stelle wird als Beispiel ein elektronisches Telefonverzeichnis der Mitarbeiter einer Firma vorgestellt. Es ist aber ebenso vorstellbar, Kunden und sonstige Kontakte ebenfalls auf diesem Wege zu verwalten.

Eine Bereitstellung eines elektronischen Telefonbuches hat gegenüber eines gedruckten Papier-Heftes deutliche Vorteile:

- **Aktualität:**
  Das Telefonbuch ist immer auf dem neuesten Stand. Die Problematik von veralteten Telefonverzeichnissen wird so entschärft.
- **Umweltbelastung:**
  Ein elektronisches Telefonbuch benötigt keine großen Mengen an Papier und Druckfarbe, wodurch die Umwelt deutlich geschont werden.
- **Kosten:**
  Telefonverzeichnisse in Papierform müssen jedem Mitarbeiter zur Verfügung gestellt werden. Durch Mitarbeiterfluktuationen kann davon ausgegangen werden, dass ein solches Verzeichnis bereits beim Drucken wieder veraltet wird. Regelmäßige Aktualisierungen müssen jedem Mitarbeiter in relativ kurzen Intervallen zur Verfügung gestellt werden. Die hierfür benötigten Ressourcen (Papier, Druckfarbe, Bindung, ...) sind mit hohen Kosten verbunden.
- **Komfort:**
  Ein gedrucktes Telefonbuch unterstützt jeweils nur eine Suchmethode. Meist werden Informationen nach der alphabetischen Reihenfolge oder der Zuordnung zu Abteilungen geordnet. Eine Suche über den Namen bzw. die Abteilung kann somit schnell und einfach durchgeführt werden, eine Suche nach dem Namen zu einer Telefonnummer dagegen nur durch Lesen des gesamten Verzeichnisses.
  In einem elektronischen Verzeichnis können beliebige Suchmethoden zur Verfügung gestellt werden, ohne das dadurch die Kosten extrem steigen.

Im folgenden wird nun das Novell "eGuide" vorgestellt. Dabei handelt es sich um Java-Applet, welches den Benutzer bei der Suche nach Personen in einer beliebigen LDAP-Implementierung unterstützt. In einem Novell-Netware-Netzwerk sind diese Informationen sogar direkt aus der Benutzerverwaltung vorhanden, so dass sich hier eine entsprechende Weiternutzung als Telefonbuch besonders anbietet.

Das eGuide ist als Telefonbuch für den Einsatz in einem firmeninternen Intranet gedacht. Durch die Implementierung als Java-Applet ist eine weitestgehende plattformunabhängige Benutzung möglich. Das Applet bietet dem Benutzer eine komfortable Oberfläche, über die er Personen suchen kann, sowie je nach Konfiguration Daten ändern kann.

Die weitere Beschreibung bezieht sich auf die spezielle Konfiguration, wie sie bei der Bausparkasse Wüstenrot AG eingesetzt wird.

Das Applet soll anhand der Oberfläche vorgestellt werden:

Die Oberfläche gliedert sich in drei Bereiche:

- Der linke Bereich dient der Eingabe der Suchkriterien
- Die Ausgabe des Ergebnisses erfolgt auf der rechten Seite.
- Die obere Zeile dient der persönlichen Konfiguration der Anwendung, sowie der Anbindung von email, instant-messaging und video-conferencing. Zusätzlich können die eigenen Daten bearbeitet werden. Auf diese Funktionen soll hier nicht näher eingegangen werden, da sie nur sekundär mit dem Verzeichnisdienst in Verbindung stehen.

Die Suche nach einer oder mehreren Personen erfolgt durch die Eingabe der Suchkriterien im linken Bereich.

Hier kann zunächst eine Auswahl getroffen werden, nach welchem Kriterium gesucht werden soll. Zur Verfügung stehen hierbei:

- UserID
- Nachname
- Vorname
- Abteilung

Als nächstes kann der Suchmodus ausgewählt werden. Beispiele hierfür sind:

- beginnt mit
- ist genau gleich
- ist nicht gleich

Über den Plus-Button kann ein weiteres Kriterium angegeben werden, über den Lupen-Button wird die Suche gestartet. Aus Datenschutzgründen wurden in der Grafik die Informationen ausgegraut.

Das Suchergebnis wird im rechten Bereich zunächst als Liste angezeigt. Durch Auswahl eines Eintrages werden dann die vollständigen Daten des entsprechenden Benutzers, wie im Bild angegeben, angezeigt.

3. APIs allgemein

Es gibt verschiedene Ansätze für das Design von APIs. Diese sind meist abhängig von der Sprache und dem vorgesehenen Einsatzgebiet. Die beiden Hauptvertreter sind

- Handle-basiert (z.B. C-API)
- objektorientiert (z.B. Java-API)

Bei der Handle-basierten API wird bei jedem Funktionsaufruf ein Zeiger (Handle) auf die benötigten Daten, bzw. Strukturen für das Ergebnis mitgegeben.

Bei der objektorientierten API werden auf Objekten, welche die benötigten Daten halten, Methoden aufgerufen, die wiederum Objekte als Ergebnis zurückgeben.

Im Rahmen dieser Ausarbeitung soll nun die Java-API von Netscape als Vertreter der objektorientierten APIs vorgestellt werden.

4. Netscape-Java-API

4.1 grundlegender Aufbau

Die Nestcape-Java-API ist objektorientiert aufgebaut. Sie erlaubt sowohl eine synchrone als auch eine asynchrone Verbindung mit dem Server.

Bei der synchronen Verbindung wird bei einem Methodenaufruf der aufrufende Thread solange blockiert, bis das Ergebnis vom Server eingetroffen ist.

Bei der asynchronen Verbindung dagegen wird ein Aufruf an den Server übermittelt und direkt im Anschluss der laufende Thread weiter ausgeführt. Beim Eintreffen der Antwort vom Server werden dann eingetragene SearchListener benachrichtigt. Ein Listener ist ein Objekt mit einer definierten Schnittstelle, die eine Methode enthält, die in diesem Fall beim Eintreffen einer Nachricht aufgerufen wird.

Die Verwendung der asynchronen Verbindung hat insbesondere im Bereich Performance dramatische Auswirkungen, da die Anwendung bis zum Eintreffen der Antwort bereits weitere Funktionalitäten ausführen kann. Diese Verbindungsart eignet sich insbesondere beim Zugriff auf eine Vielzahl von Servern mit automatischem Referral-Handling und langsamen Verbindungen. Dagegen steht eine etwas aufwendigere Implementierung.

Aus Übersichtsgründen wird im folgenden jeweils eine synchrone Verbindung verwendet.

Die Schlüsselklassen für die hier vorgestellte Zugriffe sind:

- LDAPConnection
- LDAPEntry
- LDAPAttributeSet und LDAPAttribute
- LDAPModificationSet und LDAPModification
- LDAPSearchResults und LDAPSearchResult
- LDAPException und LDAPReferralException

Für jede Funktion gibt es in der Regel mehrere Möglichkeiten. Diese unterscheiden sich insbesondere in den angegebenen Informationen. Diese Ausarbeitung gibt hier jeweils nur eine Möglichkeit an. Weitere Möglichkeiten können in der Dokumentation zur API nachgelesen werden.

Zentrales Objekt ist die LDAPConnection. Diese repräsentiert eine Verbindung zu einem Server. Die meisten Funktionen sind direkt als Methoden in dieser Klasse implementiert. Die Konfiguration der Verbindung, sowie einzelner Funktionen werden ebenfalls durch diese Klasse vorgenommen.

Die restlichen Klassen dienen der Datenhaltung und der Fehlerbehandlung.

4.2 Verbindungsaufbau

Eine Verbindung zu einem LDAP-Server wird durch eine Instanz der Klasse LDAPConnection repräsentiert.

```
1:    LDAPConnection connection = new LDAPConnection();
2:    connection.connect(<ServerListe>, <DefaultPort>);

3:    doSomethingWithLDAP();

4:    connection.disconnect();
```

In Zeile 1 wird eine neue LDAPConnection-Instanz erzeugt. In Zeile 2 wird erstmals eine Verbindung mit dem Server aufgebaut. Die Serverliste gibt mindestens einen Server an, mit dem die Verbindung aufgebaut werden soll. Werden mehrere Server angegeben, so kann bei Ausfall des ersten Servers eine Verbindung mit dem zweiten Server versucht werden. Die Liste wird als String übergeben, wobei die einzelnen Server durch Leerzeichen getrennt werden. Eine Serverangabe setzt sich zusammen aus Servernamen oder IP-Adresse und optional einer Portangabe. Wird diese weggelassen, so wird der im zweiten Parameter angegebene Default-Port verwendet.

Als Beispiel sollen nun die Parameter mit folgenden Werten belegt werden:

ServerListe = `"ldap.test1.de ldap.test2.de:3342 ldap.test3.de"`
DefaultPort = `LDAPv2.DEFAULT_PORT` (=389)

Es wird nun in folgender Reihenfolge versucht eine Verbindung aufzubauen. Glückt dieser Versuch, so werden die folgenden Server ignoriert:

- ldap.test1.de:389
- ldap.test2.de:3342
- ldap.test3.de:389

Zeile 3 steht stellvertretend für irgendwelche Funktionalitäten auf der LDAP-Verbindung. Die Möglichkeiten für diese werden nun im weiteren vorgestellt.

Über die Methode disconnect in Zeile 4 wird die Verbindung wieder abgebaut. Funktionen auf der Verbindung können erst nach einer weiteren Ausführung von connect ausgeführt werden.

4.3 Authentifizierung

Meist sind nicht alle LDAP-Funktionen für jeden verfügbar. Die Berechtigungen für die einzelnen Funktionen sind abhängig vom jeweiligen Benutzer.

Damit der Server weis, welcher Benutzer die Funktionalitäten ausführen will, ist es nötig, sich beim Server zu authentifizieren.

Die Java-API bietet hier mehrere Möglichkeiten:

- Anonymous
- Simple
- Zertifikatbasiert über SSL (Secure Socket Layer)
- SASL (Simple Authentication Security Layer)

Die letzten beiden Methoden sollen in dieser Ausarbeitung hier nicht näher betrachtet werden.

Die anonyme Authentifizierung genügt aus, wenn keine auf einzelne Benutzer beschränkten Funktionen ausgeführt werden sollen. Beispielsweise ist die Suche nach einer Telefonnummer in der Regel nicht beschränkt.

Die anonyme und die simple Authentifizierung erfolgen über dieselbe Methode, mit dem Unterschied, dass bei der anonymen Authentifzierung die Parameter "User" und "Password" mit null-Werten belegt werden.

```
connection.authenticate(<user>, <password>);
```

Für die anonyme Authentifizierung werden beide Parameter mit `null` belegt.
Bei der simplen Authentifzierung wird der erste Parameter mit dem DN des Benutzers belegt und der zweite Parameter mit dem Passwort im Klartext.

Beispiel:

User = `"uid=ubxwgm,ou=user,o=wsft.de"`
password = `"geheim"`

Vorteil dieser Methode ist die einfache Verwendung, Nachteil dagegen ist die Übermittlung des Passwortes im Klartext. Diese Methode eignet sich somit nur für unkritische Anwendungen in einem begrenztem Netzwerk, wie z.B. ein Intranet.

4.4 Fehlerbehandlung

Bei fast jedem Zugriff können Fehler auftreten. Diese können mehrere Ursachen haben:

- die Verbindung zum Server konnte nicht aufgebaut werden
- es wurden ungültige Informationen bei der Authentifizierung angegeben
- eine Anfrage ist syntaktisch nicht korrekt
- ein angefordertes Objekt wurde nicht gefunden
- ein Alias konnte nicht aufgelöst werden

Alle Fehler werden, wie in Java üblich, über Exceptions behandelt. Die API definiert zwei verschiedene Exception-Klassen:

- LDAPException:
  Fehler der oben genannten Art
- LDAPReferralException:
  Fehler in Verbindung mit Referrals (siehe unten)

Die Klasse LDAPException definiert zwei Methoden:

getLDAPResultCode() liefert die Fehlerursache, getLDAPErrorMessage liefert eine textuelle Beschreibung dazu. Die Beschreibungen werden über lokalisierte Property-Files definiert.

4.5 Umgang mit Referrals

Für den Umgang mit Referrals können mehrere Methoden verwendet werden.

- automatisch (Default)
- automatisch mit Authentifizierung
- manuell

Die Einstellung der zu verwendenden Methode erfolgt durch

```
connection.setOption(   LDAPv2.REFERRALS,
                              new Boolean(<auto>));
```

Wird der Parameter `auto` mit `true` belegt, so werden Referrals automatisch behandelt, bei `false` wird bei Auftreten eines Referrals eine LDAPReferralException geworfen, auf die dann entsprechend reagiert werden kann.

Soll eine Authentifizierung beim Umgang mit Referrals erfolgen, so ist eine Klasse zu implementieren, welche das Interface LDAPRebind implementiert. Eine Instanz dieser Klasse ist dann über setOption an die Connection anzuhängen:

```
connection.setOption(   LDAPv2.REFERRALS_REBIND_PROC,
                              <LDAPRebind-Instanz>);
```

Für alle Methoden kann die Anzahl der zu verfolgenden Servern ("Hops") angegeben werden. Standard ist hier eine Limitierung auf 10 Hops.

Die Einstellung der maximalen Hops erfolgt durch

```
connection.setOption(   LDAPv2.REFERRALS_HOP_LIMIT,
                              new Integer(<Hops>));
```

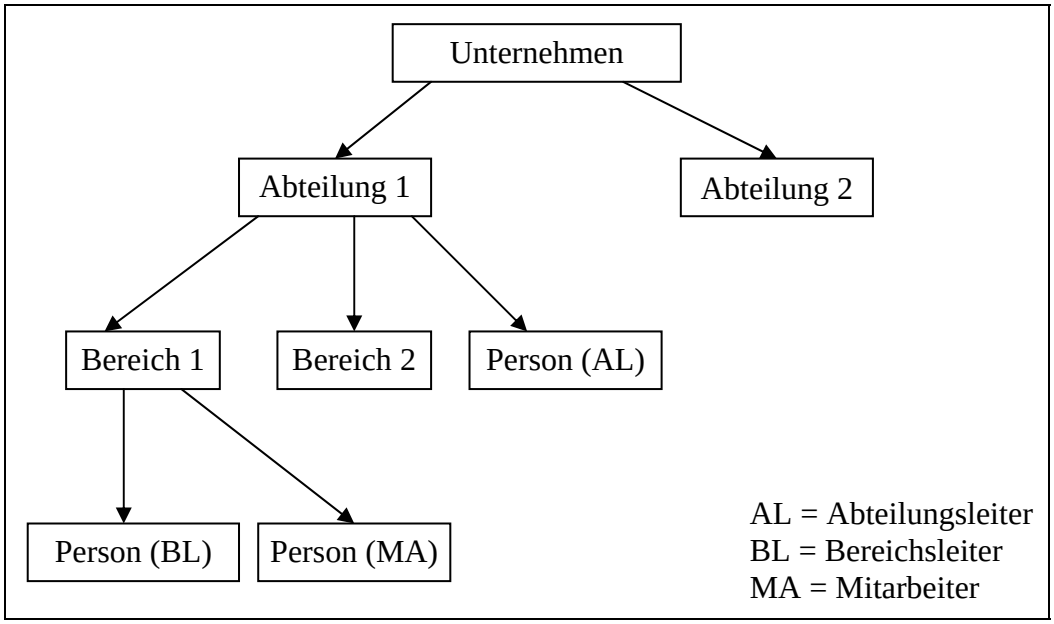
Standardmäßig werden Referrals automatisch von der API behandelt. Dies ist nicht zu verwechseln mit Chaining-Mechanismen, die Referrals automatisch auf der Serverseite behandeln. Beim automatischen Referral-Handling über die API wird lediglich dem Benutzer die Implementierung abgenommen. Dieser Mechanismus verwendet eine anonyme Authentifizierung bei den verwiesenen Servern.

4.6 Suchen

Die API bietet vielfältige Möglichkeiten für die Suche nach Einträgen im LDAP-Verzeichnis.

Für das bessere Verständnis werden die verschiedenen Suchmöglichkeiten anhand eines Beispiels erläutert.

Für das Beispiel wird folgende einfache LDAP-Struktur der Personen eines Unternehmens verwendet:



Hierzu gehören

- `searchbase`: Such-Basis
- `searchscope`: Such-Raum
- `filter`: Suchkriterien
- `attributes`: gewünschte Attribute
- `attributesonly`: Ausgabe "nur Attribute" oder "Attribute und Werte"
- `constraints`: Nebenbedingungen (z.B. Zeit- und Mengenbeschränkung)

Als weitere Optionen stehen zur Verfügung:

- Sortierung auf dem Server (bei entsprechender Konfiguration)
- Sortierung auf dem Client
- Abbruch der Suche
- Ermittlung von Attributnamen
- Auflisten von Subentries
- Filter vordefinieren

Diese erweiterten Optionen sollen im Rahmen dieser Ausarbeitung nicht näher behandelt werden.

Der String-Parameter **searchbase** gibt den Startpunkt der Suche an. Es handelt sich hierbei um den DN des Startknotens.

Der **searchscope-Parameter** gibt an, wieweit die Suche durchgeführt werden soll. Er kann mit einer der folgenden Konstanten belegt werden:

- SCOPE_BASE
- SCOPE_ONE
- SCOPE_SUB

Bei SCOPE_BASE wird nur die angegebene Such-Basis durchsucht. Dieser Modus eignet sich insbesondere für den Zugriff auf einzelne Entries, deren DN vorher bekannt ist. Anwendung hierfür ist z.B. die Ermittlung der Telefonnummer oder der Zugriff auf Authorisationsangaben eines Benutzers.

Bei SCOPE_ONE werden alle Entries, die exakt eine Ebene unter der angegebenen Such-Basis enthalten sind, durchsucht. Dieser Modus eignet sich in der Beispiel-Struktur z.B. für die Ermittlung aller Personen, die einer Abteilung aber keinem Bereich zugeordnet sind (wie der Abteilungsleiter).

Bei SCOPE_SUB werden alle Entries, die unter der angegebenen Such-Basis enthalten sind, durchsucht. Dieser Modus eignet sich z.B. für die Ermittlung aller Personen einer Abteilung, unabhängig davon, ob sie zusätzlich einem Bereich zugeordnet sind.

Über den String-Parameter **filter** werden die Suchbedingungen angegeben. Ein Kriterium folgt dem Schema

```
<Attribut> <Operator> <Wert>
```

Als Operatoren stehen zur Verfügung:

- = exakte Übereinstimmung
- >= größer gleich
- <= kleiner gleich
- =* Dem Attribut muss (mindestens) ein Wert zugewiesen sein
- ~= etwa gleich

Mehrere Kriterien können durch boolesche Operatoren und Klammerungen kombiniert werden.

Bei den booleschen Operatoren ist zu beachten, dass diese nicht in der "regulären" Syntax angegeben werden, sondern der Operator vorangestellt wird:

(<Operator> (<Kriterium 1>) (<Kriterium 2>) ...)

Als Operatoren stehen hier zur Verfügung:

- & UND-Verknüpfung
- | ODER-Verknüpfung
- ! NOT-Verknüpfung

Der String-Array-Parameter **attributes** bietet die Möglichkeit das Suchergebnis auf einzelne Attribute zu beschränken. Wird hier null angegeben, so werden alle Attribute, ausser operationalen Attributen (dies sind Attribute zur Server-Verwaltung, wie "creatorsName") zurückgegeben. Sollen einzelne Attribute zurückgegeben werden, so müssen die Attribut-Namen in einem Array übergeben werden. Nähere Informationen zur Eingrenzung der Attribute sind in der API-Dokumentation zu finden.

Der boolesche Parameter **attributesonly** legt fest, ob nur die Attributnamen (`true`) oder die Attributnamen und die zugehörigen Werte (`false`) zurückgegeben werden sollen.

Der **constraint**-Parameter erwartet eine Instanz der Klasse LDAPSearchConstraints. In diesem können z.B. folgende Einstellungen vorgenommen werden:

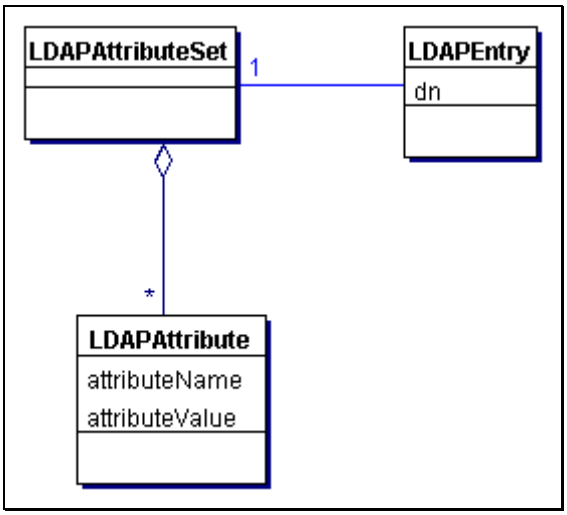
- maximale Ergebnismengengröße
- maximale Suchzeit
- Verfahren zur Alias-Dereferenzierung

4.7 Manipulation von Entries

Die Manipulation umfasst das Anlegen, Ändern, Umbenennen und Löschen von Entries. Da derartige Zugriffe praktisch immer auf bestimmte Benutzer eingeschränkt ist, ist hier eine authentifizierte Verbindung Voraussetzung.

4.7.1 Hinzufügen von Entries

Um ein Entry hinzuzufügen muss dieser zunächst in Form von Objekten auf dem Client kreiert werden.



Das UML-Diagramm zeigt die Struktur der Objekte mit ihren wichtigsten Attributen. Es wird ein einzelnes LDAPEntry-Objekt erzeugt. Dieses wird definiert durch einen DN und ein LDAPAttributeSet-Objekt. Dieses wiederum enthält eine Menge von Attributen und zugehörigen Werten.

Zu beachten ist, dass das Attribut "objectclass" zwingend angegeben werden muss.

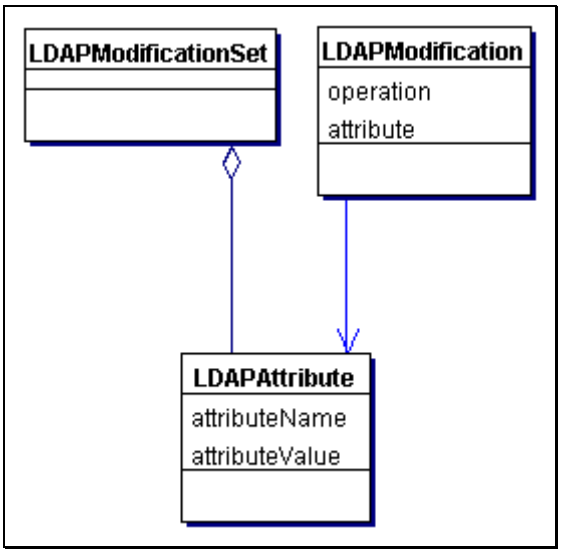
Nachdem das LDAPEntry-Objekt vollständig erstellt wurde, kann es über die Methode

```
connection.add(<LDAPEntry-Instanz>);
```

dem Verzeichnis hinzugefügt werden.

4.7.2 Bearbeiten von Entries

Die Bearbeitung von Entries erfolgt auf eine ähnliche Weise:



Soll nur ein einzelnes Attribut geändert werden, so kann dies über eine Instanz der Klasse LDAPModification erfolgen. Wie aus dem Diagramm ersichtlich ist, ist hierfür eine LDAPAttribute-Instanz nötig, welche den neuen Wert des Attributs enthält. Im LDAPModification-Objekt ist hinterlegt, was mit dem Attribut passieren soll. Mögliche Werte sind hier "ADD", "DELETE" und "REPLACE".

Sollen mehrere Attribute geändert werden, so ist dies über ein LDAPModificationSet durchzuführen. Diesem werden einzelne LDAPAttribute-Objekte zusammen mit dem jeweiligen Änderungsmodus hinzugefügt. Warum LDAPModificationSet nicht als Komposition aus LDAPModification-Instanzen realisiert wurde ist nicht nachvollziehbar, da so zwei verschiedene Konzepte verwendet werden. LDAPModificationSet muss intern den jeweiligen Modus irgenwie speichern, eine Speicherung in einer LDAPModification hätte sich hier angeboten.

Das erzeugte LDAPModification- oder LDAPModificationSet-Objekt kann nun durch die Methode

```
connection.modify(<Entry-DN>, <Modification-Object>);
```

dem Server zur Aktualisierung übergeben werden.

Wird auf ein Attribut mehrfach zugegriffen ist zu beachten, dass die Änderunge exakt in der Reihenfolge, in der sie dem LDAPModificationSet-Objekt angefügt wurden, durchgeführt werden.

4.7.3 Löschen und Umbenennen von Entries

Das Löschen von Entries erfolgt ebenfalls über eine Methode des Connection-Objekts:

```
connection.delete(<Entry-DN>);
```

Ein Entry kann umbenannt bzw. verschoben werden, indem die Methode

```
connection.rename(<Old-DN>, <New-DN>, <DeleteOldDN?>);
```

Über den dritten Parameter (boolean) kann festgelegt werden, ob der alte Entry gelöscht werden soll oder nicht. Hierdurch wird die Möglichkeit des Kopierens bzw. Verschiebens eröffnet.
Zu beachten ist, dass diese Funktionalität nicht bei allen Servern verfügbar ist.

4.8 Zugriff auf Server-Informationen

Server-Informationen werden in einem speziellen Entry, dem sogenannten Root-DSE (Directory System Entry) gespeichert. Dieses ist jeglichem Namenskontext übergeordnet, es besitzt also keinen bzw. einen leeren DN.

Zu diesen Informationen können beispielsweise gehören:

- Namenskontext des Servers
- URLs von alternativen Servern
- unterstützte Erweiterungen
- unterstützte Server-Controls (z.B. serverseitige Sortierung)
- unterstützte Protokoll-Versionen

Bei der Liste alternativer Server ist zu beachten, dass diese auf den Client repliziert werden muss, damit im Falle eines Serverausfalls ein entsprechender Alternativ-Server angesprochen werden kann.

Um das Root-DSE zu erhalten muss eine Suche durchgeführt werden, wobei folgende Parameter verwendet werden müssen:

- kein automatisches Referral-Handling
- Suchraum: SCOPE_BASE
- Suchbasis: ""
- Filter "objectclass=*"

Hier ist zu beachten, dass dieser Root-DSE erst ab der LDAP-Version 3 zur Verfügung steht. Wird eine derartige Suche auf einem Servern älterer Version durchgeführt, so wird ein Fehler ausgegeben.

4.9 weitere Möglichkeiten

Da die Möglichkeiten der Java-API sehr umfangreich sind, kann im Rahmen dieser Ausarbeitung nicht auf alle Funktionalitäten eingegangen werden.

Die API bietet beispielsweise weiterhin:

- Caching:
  Oft benötigte Elemente können clientseitig in einem Zwischenspeicher (Cache) gehalten werden, um den Nachrichtenverkehr mit dem Server zu reduzieren
- Connection-cloning:
  Bestehende Connection-Objekte können jederzeit geklont werden, wodurch ein zweites Connection-Objekt mit den gleichen Eigenschaften, aber ohne weitere Verbindung zum ursprünglichen Objekt erzeugt wird.
- DN-Komponenten-Extraktion:
  Die einzelnen Teile eines DNs können durch die API aufgespaltet werden, so dass auf die einzelnen Teil-DNs zugegriffen werden kann.
- Schema-Definition:
  Über die API können ebenfalls Schemata definiert werden.

5. Netscape C-API

Die C-API von Netscape hat aufgrund der rein strukturierten Zielsprache diverse Unterschiede zur Java-API.

Der grundlegende Aufbau, insbesondere was die Verteilung der Funktionalitäten angeht, ist mit dem der Java-API identisch.

Hauptunterschied ist, dass die Funktionalitäten nicht durch Methoden von Objekten realisiert werden, sondern durch Funktionen denen ein Zeiger auf die entsprechenden Daten (Connection, Entry, ...) mitgegeben wird.

Die Fehlerbehandlung erfolgt durch die Rückgabe von Fehlercodes. Nach jedem Funktionsaufruf muss also explizit die korrekte Durchführung überprüft werden.

Die C-API teilt den Nachteil, dass der Entwickler für die korrekte Speicherverwaltung verantwortlich ist. Benötigter Speicher muss explizit allociert und später wieder freigegeben werden.

6. Zusammenfasung

Für alle relevanten Sprachen und Plattformen gibt es APIs und Clients.
Clients sind für viele Standardaufgaben in einer direkt nutzbaren Form vorhanden, so dass z.B. ein Intranet-Telefonbuch bei vorhandenen Daten relativ schnell erstellt werden kann.

Sowohl die Java- als auch die C-API bietet alle denkbaren Möglichkeiten, die in der Benutzung oder auch Administration von einem Client benötigt werden könnten. Sie sind beide sehr mächtig und relativ einfach in der Bedienung. Die Netscape-APIs werden mit einer gut lesbaren und ausführlichen Dokumentation ausgeliefert, wodurch sie insbesondere für Neulinge sehr gut geeignet sind. Verstärkt wird dies durch den (fast) vollständig logischen Aufbau der APIs.

Weitere Informationen sind z.B. auf folgender Internet-Seite verfügbar:

http://developer.netscape.com/docs/manuals/dirsdk