Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Synchronisation heterogener LDAP Schemas

Diplomarbeit

Leipzig, 11. Dezember 2002

vorgelegt durch

Oliver Schulze

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Directory Services oder auch Verzeichnisdienste werden im ITU-T Standard X.500 nur als "The Directory" bezeichnet [1].

> "The *Directory* is a collection of open systems which cooperate to hold a logical database of information about a set of objects in the real world."

Ein Verzeichnis hat eine baumartige objektorientiere Struktur und kann im Netzwerk verteilt werden. Die Verteilung kann aus Gründen der Performance, Verfügbarkeit oder auch Datensicherheit erfolgen. Ob und welche Art von Zugriff Benutzer oder Appliktionen auf Daten erhalten, wird in detaillierten Zugriffsregeln festgelegt. Zur Addressierung von Daten im Verzeichnis verfügen die Dienste über einen einheitlichen Namenskontext und mächtige Suchfunktionen.

Im Unterschied zu Datenbanken, wird auf Verzeichnisse wesentlich häufiger lesend zugegriffen als schreibend [16]. Datenbanken sind dafür optimiert, eine grosse Menge von verschiedenen und komplizierten Transaktionen durchzuführen. Verzeichnisse unterstützen nur sehr einfache Transaktionen. Sie sind meist weniger komplex als Datenbanken und können leicht erweitert werden.

Als Resultat der großen Bemühungen zur Erstellung der X.500 Spezifikation wird heute die Unterstützung von Verzeichnissen in vielen Programmen als Norm angesehen. Verzeichnisdienste halten die Daten vieler verschiedener Anwendungen an einer einzigen Stelle und vereinfachen damit die Konfiguration und Administration in großen Netzwerken enorm.

Das Einsatzgebiet von Directory Services ist in den meisten Fällen die zentrale Verwaltung von Daten eines Netzwerkes. Sie halten zum Beispiel die Informationen über Benutzer, Computer oder Anwendungen. Sie erlauben es, die Konfigurationsdaten von jedem Rechner aus zu verändern und bieten eine einheitliche Schnittstelle für den Zugriff unabhängig vom Betriebssystem.

Unterschiede im Datensatzschema verschiedener Hersteller verhindern aber den reibungslosen Zugriff mit beliebigen Anwendungen. Zur Realisierung einer zentralen Authentifizierungsstelle bleibt in heterogenen Netzwerken oft keine andere Möglichkeit als der Einsatz mehrerer Verzeichnisse, deren Daten synchron gehalten werden.

1.2 Zielstellung

Das Ziel dieser Arbeit ist die Entwicklung und Bewertung von Ansätzen zur Integration heterogener LDAP Schemas. Diese Aufgabe gliedert sich in folgende Teile. Als erstes müssen der Begriff des Verzeichnisschemas und die Möglichkeiten des Verzeichniszugriffs geklärt werden. Danach wird als Vorbereitung auf die Synchronisation heterogener Schemas die Replikation homogener untersucht. Nun erst werden die Ansätze fremder Projekte zum Synchronisationsproblem betrachtet und darauf aufbauend eigene erarbeitet. Sie werden hinsichtlich ihrer Administrierbarkeit, Flexibilität zur Integration verschiedener Schemas und Einsatzmöglichkeit zur Synchronisation von Benutzerdaten bewertet. Ein ausgewählter eigener Ansatz wird implementiert und beurteilt. Im Ergebnis sollen eine Empfehlung zur Synchronisation und ein Ausblick auf den Einsatz heterogener Schemas entstehen.

1.3 Gliederung

Das Kapitel 2 ab Seite 4 geht zunächst auf die Geschichte und die Standards von Verzeichnisdiensten ein. Besonderes Augenmerk wird auf die Spezifikationen X.500 und LDAP gelegt. In diesem Kapitel wird das grundlegende Verständnis für Verzeichnisdienste aufgebaut.

Die Grundlage für die Synchronisation von heterogenen LDAP Schemas ist die Replikation zwischen Servern homogener Schemas. Sie wird in Kapitel 3 ab Seite 20 erläutert.

In Kapitel 4 ab Seite 32 werden fremde Ansätze zur Synchronisation untersucht. Darauf folgend zeigt Kapitel 5 ab Seite 43 ihre Umsetzung in allgemeine Modelle und Schlußfolgerungen für ein neues Modell.

Das 6te Kapitel ab Seite 47 beschreibt die Implementation des Ansatzes aus Kapitel 5. Sie wird im Kapitel 7 auf ihre Leistungs- und Einsatzfähigkeit getestet.

Abschliessend wird in Kapitel 8 ab Seite 65 festgestellt, welche Ansätze für die Integration heterogener LDAP Schemas in der Benutzerverwaltung geeignet sind und welche Möglichkeiten auf diesem Gebiet in Zukunft geboten werden.

Kapitel 2

Verzeichnisdienste

2.1 Geschichte

Die ersten elektronischen Verzeichnisse wurden ausschließlich verwendet um Benutzer zu authentifizieren und ihnen Zugang zu den Ressourcen eines Computers zu gewähren. Sie wurden in der Anfangszeit des Computers entwickelt. Beispiele sind UNIX /etc/passwd (Bell Labs), UserDirectory (University of Michigan) oder Michigan Terminal System(MTS)[16].

Die erste Entwicklung von verteilten Systemen um 1980 initiierte auch die Entstehung von verteilten Verzeichnissen. Eines der bedeutensten war Grapevine vom Xerox Palo Alto Research Center. Es kommunizierte bereits über Ethernet und verfügte über eine Datenbank, in der zum Beispiel Netzwerk- und Benutzerinformationen gespeichert wurden.

Durch den Einfluß des entstehenden Internets wurde im Jahr 1982 WHOIS und 1984 DNS entwickelt. DNS konnte bereits verteilt administriert und auf mehrere Server repliziert werden.

Nach 1990 begann wurden sehr viele Verzeichnisdienste entwickelt, die an spezielle Applikationen gebunden waren. Sie waren nur dafür gedacht, einer oder wenigen Anwendungen zu dienen. Die meisten waren Groupware-Produkte, so wie Lotus Notes, Microsoft Exchange, Microsoft Outlook oder Novell Groupwise Directory.

1993 kam Novell Directory Services (NDS) in Netware 4.0 auf den Markt. Es gehört zu der Gruppe der "Network Operation System" (NOS) Verzeichnisse. NOS sind zum Beispiel Novell Netware oder Microsoft Windows NT. NOS Verzeichnisse wurden eingeführt, um in einem LAN zentral Benutzer authentifizieren und Ressourcen administrieren zu können.

Alle bisher vorgestellten Verzeichnisdienste waren nur auf eine bestimmte Aufgabe spezialisiert. Aber schon im Jahr 1983 begannen sowohl ISO als auch ITU, welches damals allerdings noch CCITT hieß, mit der Entwicklung von offenen Standards für Verzeichnisse. Allerdings noch mit verschiedenen Zielen. Diese beiden Projekte wurden 1986 zusammen bearbeitet und es entstand 1988 das "CCITT X.500 Blue Book" [32]. Es wurde 1990 von CCITT veröffentlicht und 1991 unter ISO 9594. Weitere Versionen entstanden 1993 und 1997. In ihnen wurde X.500 zum Beispiel um genaue Replikations- und Sicherheitsdefinitionen erweitert.

2.2 X.500

X.500 ist ein von Softwareherstellern, Betriebssystemen oder Anwendungen unabhängiger Standard [16]. Die meisten heute anzutreffenden Directory Services sind aber trotzdem keine reinen X.500 Anwendungen. Statt dessen verfügen sie über mehr Funktionen und Möglichkeiten als in der Spezifikation vorgesehen. Da sie aber den Großteil der Konzepte implementieren, werden hier wichtige Teile des Standards erläutert. Dazu gehören das Datenmodell, das Namensmodell, das Funktionale Modell, das Verteilte Modell und das Sicherheitsmodell [2].

X.500 wird mit Hilfe der *"Abstract Syntax Notation One"* (ASN.1) beschrieben [12].

2.2.1 Datenmodell

In X.500 werden die Informationen, die ein Verzeichnisdienst hält, als "*Directory Information Base*" (DIB) bezeichnet [1]. Die DIB ist objektorientiert und hierarchisch organisiert.

Die DIB besteht aus einer Menge von Einträgen oder zusammengesetzten Einträgen, von denen jeder einen einzigartigen Namen hat. Ein Eintrag kann vom Typ "*Objekt*", "*Alias*", "*Untereintrag*" oder "*Family Member*" sein. Untereinträge werden nur für Operationen und administrative Zwecke des Verzeichnisses verwendet. Ein Family Member ist ein Teil eines zusammengesetzten Eintrags. Alle Teile eines zusammengesetzten Eintrags bilden eine eigene Hierarchie, deren Wurzel ein sogenannter "*Anchestor*" ist und den zusammengesetzten Eintrag repräsentiert. Objekte sind Abbildungen von "Objekten der realen Welt". In ihnen werden die Informationen des Verzeichnisses abgelegt. Ein Alias ist ein Verweis auf ein Objekt und stellt damit einen alternativen Namen dar [2].

Die Einträge der DIB werden in einer Baumstruktur gehalten, dem "*Directory Information Tree*" (DIT). Die Wurzel des Baumes wird in der DIB

durch ein Root-Objekt dargestellt, das per Definition keine Daten enthält. Alle Objekte des DIB sind Knoten in diesem Baum, wobei Alias-Einträge immer Blätter sind. Kanten bestimmen den hierarchischen Zusammenhang zwischen Objekten. Eine Kante zwischen zwei Objekten besteht dann, wenn eines dem anderen untergeordnet ist. Jedes Objekt kann beliebig viele andere Objekte enthalten, also als Container dienen. Die entstehende Hierarchie kann (und sollte) eine Bild der Organisationsstruktur der gespeicherten Daten sein. Die Abbildung 2.1 zeigt einen schematischen DIT und Abbildung 2.2 auf Seite 6 den selben DIT als Abbildung realer Objekte und ihrer Hierarchie.

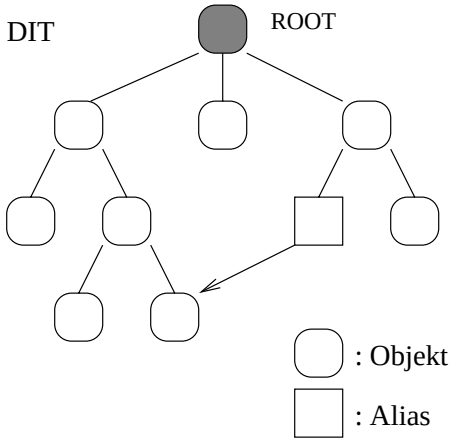

Abbildung 2.1: Struktur des DIT.

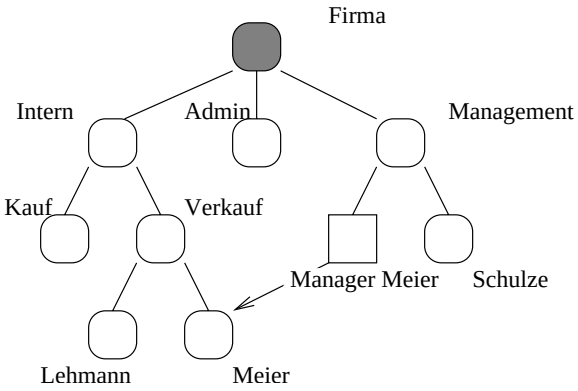

Abbildung 2.2: DIT mit Objekten der realen Welt.

Abbildung 2.3 zeigt, daß jedes Objekt neben seinen untergeordneten Objekten ein oder mehrere Attribute besitzt. Ein Attribut ist ein Paar aus Attributetyp und einer Menge von Attributwerten. Welche Attribute ein

Objekt enthält und ob das Vorkommen wahlweise oder zwingend ist, legt seine Objektklasse fest.

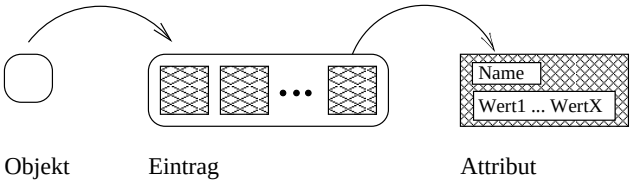

Abbildung 2.3: Aufbau eines Objektes des DIT.

Es gibt drei Typen von Objektklassen: *abstract*, *structural* and *auxilary*. Eine Klasse kann eine Unterklasse sein, wodurch sie alle Eigenschaften einer anderen Klasse, der Oberklasse, übernimmt. Sie kann allerdings auch noch zusätzliche Eigenschaften ergänzen. Unterklassen können bis zu einer beliebigen Tiefe abgeleitet werden. Als oberste Klasse ist *top* definiert, die das zwingende Attribut "objectClass" enthält . Jede Klasse vom Typ structural ist implizit eine direkte oder indirekte Unterklasse von top und damit besitzt jedes Objekt ein Attribut "ObjectClass". Die Werte dieses Attributes sind die Namen aller Objektklassen, denen das Objekt angehört. Ist ein Objekt Mitglied einer Unterklasse, ist es immer auch Mitglied aller ihrer Oberklassen.

Jedes Objekt gehört zu genau einer Klasse vom Typ structural. Der Typ abstract dient dazu künstlich Oberklassen zu definieren, von denen Eigenschaften abgeleitet werden können. Ein Objekt kann nicht nur zu Klassen vom Typ abstract gehören. Die Klasse top ist ein Beispiel für eine abstrakte Klasse. Der Typ auxilary wird dazu verwendet, Objekte einer Klasse zu gruppieren. Ein Objekt kann neben seiner structural Klasse auch ein oder mehreren auxilary Klassen angehören. Damit lassen sich für einen Teil der Objekte einer structural Klasse zusätzliche Eigenschaften definieren. Es ist aber auch möglich, unterschiedlichen structural Klassen dieselben Eigenschaften hinzuzufügen, ohne sie von einer gemeinsamen Oberklasse ableiten zu müssen.

Jedes Attribut in einer Objektlasse hat eine Attributsyntax. Sie definiert eine Menge von Regeln, die Anzahl der möglichen Werte und den Typ des Attributwertes. Die Regeln beinhalten eine Ordnungsrelation, eine Gleichheitsrelation, eine Match-Bedingung für Zeichenketten und ob das Attribut "*operational*" ist, das heißt ob es vom Benutzer verändert werden darf. Operationale Attribute werden vom Verzeichnis selbst erstellt und verwaltet. Die Relationen werden bei Such- und Vergleichsoperationen des Verzeichnisses verwendet, um Einträge mit bestimmten Attributwerten zu vergleichen.

Darf ein Attribut genau einen Wert annehmen, heißt es *einwertig* und sonst *mehrwertig*. Möglichen Werttypen sind einfache Typen, wie Zeichenketten, Zahlen und binäre Daten, oder auch komplexere Datentypen aus Zusammensetzungen von Typen [15]. Statt eines Typs kann eine Syntax auch den Namen einer anderen Attributsyntax enthalten. In dem Fall erbt sie deren Eigenschaften.

Einige Objektklassen, Attributtypen und -syntax sind international standardisiert, zum Beispiel in RFC 2252 [38]. Andere wiederum sind vom Hersteller des Verzeichnisdienstes definiert. Die jeweiligen Definitionen für ein Verzeichnis werden in seinem "*Schema*" festgehalten.

2.2.2 Namensmodell

Jedes Objekt kann durch einen Namen eindeutig identifiziert werden. Mit diesem Namen läßt sich in Verzeichnissen allerdings nicht nur das Objekt auffinden, sondern auch gleichzeitig seine Position in der Hierarchie bestimmen. Der eindeutige Name heißt "*Distinguished Name*" (DN). Die Namensgebung muß allerdings nicht eineindeutig sein, da ein Objekt mittels Alias durchaus mehrere Namen besitzen kann.

Ein DN ist eine Liste von "*Relative distinguished names*" (RDN). Sie setzt sich aus den RDN der Vorgängerobjekte im Baum und dem eigenen RDN zusammen. Das Root-Objekt besitzt definitionsgemäß keinen RDN. In Abbildung 2.4 wird die Namensbildung veranschaulicht [2, 15].

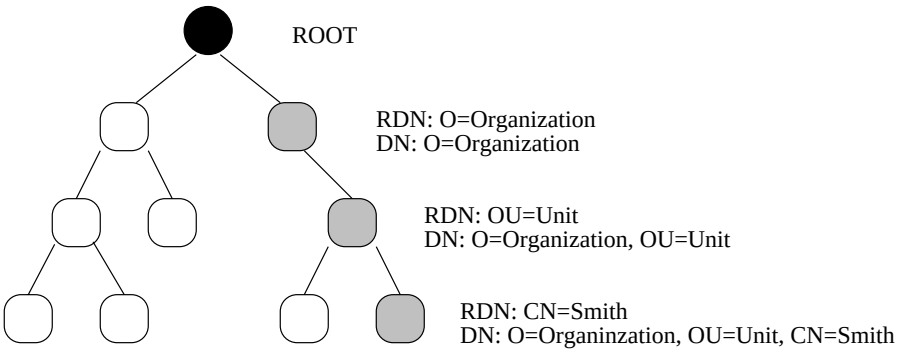

Abbildung 2.4: RDN Bildung
Der DN wird aus den RDN der Vorgänger und dem eigenen RDN gebildet.

Ein RDN wird im X.500 Standard als eine Menge von Paaren dargestellt. Jedes Paar besteht aus einem Namen und Wert eines Attributs des Objektes. Bei bestimmten Verzeichnisdiensten kann allerdings auch nur genau ein Paar zulässig sein, wie zum Beispiel bei Microsoft Active Directory. Zur der Darstellung des RDN wird Attributname und -wert mit einem Gleichheits-

zeichen verbunden. Besteht der RDN aus mehr als einem Attribut, dann werden die Paare durch Kommas getrennt und die ganze Menge geklammert (z.B. "CN=Smith" oder "(CN=Smith, uidNumber=501)"). Innerhalb eines Containers muß der RDN jedes enthaltenen Objekts einzigartig sein. Dadurch erhält jedes Objekt im DIT rekursiv einen eindeutigen Namen [15].

2.2.3 Funktionales Modell

Der Verzeichnisdienst bietet nur wenige standardisierte Operationen an, mit denen auf ihn zugegriffen werden kann. Sie sind im "*Directory Access Protocol*" definiert [15]. Benutzer kommunizieren mit dem Verzeichnis über einen "*Directory User Agent*" (DUA). Das ist eine Applikation, die für sie die entsprechenden DAP Anfragen stellt. Wie die Schnittstelle des DUA zum Benutzer aussieht, ist nicht definiert [15].

Es sind folgende Anfragen des DUA an das Verzeichnis möglich: *read*, *compare*, *list*, *search*, *add*, *modify*, *delete* und *moddn* [2]. Davon sind die ersten vier lesende Zugriffe und der Rest modifizierende. Sie werden in den Tabellen 2.1 und 2.2 erläutert.

Anfragen	Beschreibung
Read	Liest einen bestimmtes Objekt aus und gibt alle oder eine angeforderte Teilmenge der Attribute zurück.
List	Gibt alle direkten Nachfolger im DIT eines bestimmten Objektes zurück.
Compare	Vergleicht ein Attribut eines bestimmten Objektes mit einem angegebenen Wert.
Search	Gibt alle Objekte aus dem DIT zurück, die einen bestimmten Filter erfüllen. Wie bei Read werden alle oder nur eine angeforderte Teilmenge der Attribute jedes Objekts geliefert.

Tabelle 2.1: DAP Anfragen

Die Filter, die bei einer Suche verwendet werden können, geben eine Menge von Gleichheits- und Ordnungsvergleichen von Attributen mit Werten an. Die Relationen, die der Verzeichnisdienst hier und bei Compare verwendet, sind die beim Attributtyp im Schema angegebenen. Damit werden zum Beispiel auch case-insensitive Zeichenketten case-insensitive verglichen. Die Vergleichswerte können in einem Filter auch mit Wildcards verwendet wer-

den.

Modifikationen	Beschreibung
Add	Fügt ein neues Blatt-Objekt in den DIT ein. Es müssen alle zwingenden Attribute einen Wert erhalten (z.B. "objectClass").
Delete	Entfernt ein Blatt-Objekt aus dem DIT.
Modify	Verändert ein Objekt durch das Hinzufügen, Löschen oder Ersetzen von einem oder mehreren Attributen. Es werden keine RDN bildende Attribute verändert.
Modify DN	Ändert entweder den RDN des Objektes oder verschiebt das Objekt in einen neuen Container. Wenn es verschoben wird, werden sämtliche enthaltene Objekte mit ihm verschoben.

Tabelle 2.2: DAP Modifikationen

Modifikationen werden im Verzeichnis immer ganz oder gar nicht ausgeführt. Wenn also eine Modify-Operation vorgenommen wird, so werden entweder alle zu ändernden Attribute korrekt verändert oder die ganze Operation wird verworfen. Damit wird erreicht, daß sich die Datenbank immer in einem konsistenten Zustand befindet, der durch das Schema bestimmt wird.

2.2.4 Verteiltes Modell

Auf Seiten des Verzeichnisdienstes antwortet ein "*Directory Service Agent*" (DSA) den Anfragen der DUA. Die DIB ist aus einem oder mehreren DSA zusammengesetzt, die auf unterschiedlichen Systemen verteilt sein können. Jeder DSA stellt für Anfragen von DUA ein oder mehrere "*Access Points*" zur Verfügung [2].

Bei einer Partitionierung speichert jeder DSA nur einen Teil des DIT in seiner lokalen Datenbank. Abbildung 2.5 zeigt, daß jeder DSA ein oder mehrere "*Namenskontexte*" enthalten kann [15]. Ein Namenskontext ist ein Teilbaum des DIT. Dieser Teilbaum umfaßt einen Eintrag als Wurzel und alle Nachfolger bis hinunter zu den Blättern des Baumes oder bis zur Wurzel eines anderen Namenskontextes. Das "*Kontextprefix*" bezeichnet den DN des Wurzelknotens. Das Root Objekt ist nicht Teil eines Teilbaums, sondern ein

DSA spezifisches Objekt.

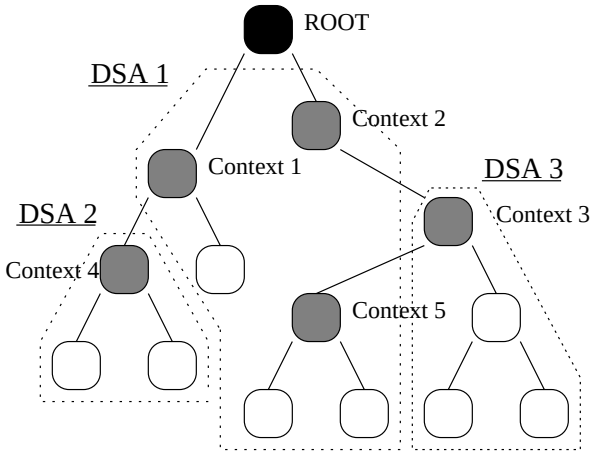

Abbildung 2.5: Der DIT wird partitioniert.

Trotz einer Partitionierung ist für Benutzer das Verzeichnis transparent und erscheint als eine einheitliche Datenbank. Er kann auf einen beliebigen DSA des Directories zugegreifen und trotzdem jede Information aus der gesamten DIB erhalten.

Befinden sich die vom DUA angeforderten Daten nicht in der lokalen Datenbank des angesprochenen DSA, so ist dieser in der Lage mit den anderen DSA über das "*Directory System Protocoll*" (DSP) zu kommunizieren [15]. In der "*Knowledge Information*" des DSA, die im Verzeichnis abgelegt wird, ist enthalten, welcher DSA die Daten beschaffen kann. Für den DSA bestehen nun zwei Möglichkeiten dem DUA zu antworten. Die erste ist, den Access Point des anderen DSA zu empfehlen und der DUA wendet sich dann selbst mit seiner Anfrage an diesen. Die Methode heißt "*Referal*" und wird in Abbildung 2.6 veranschaulicht. Bei der anderen Methode holt sich der DSA die Information selbst und reicht sie wie in Abbildung 2.7 an den DUA weiter. Wegen der verketteten Abfragen heißt diese Variante "*Chaining*". Der DUA kann bei einer Anfrage die bevorzugte Methode angeben. Die Entscheidung welche Variante verwendet wird, fällt aber nur der DSA.
Durch die Partitionierung wird die Anzahl der Objekte auf einem DSA verringert. Damit verbessern sich die Such- und Zugriffszeiten der Datenbank [14]. Um diese Eigenschaften noch weiter zu verbessern, bietet sich die "*Replikation*" an. Sie ist eine wichtige Eigenschaft von Verzeichnisdiensten und gibt die Möglichkeit Kopien eines Teilbaums des DIT auf verschiedenen DSA gleichzeitig abzulegen. Damit das Verzeichnis konsistent bleibt, werden die Kopien automatisch synchron gehalten. Im X.500 Standard ist nur eine Art von Synchronisation definiert, das sogenannte "*Shadowing*" [9]. Das Verfah-

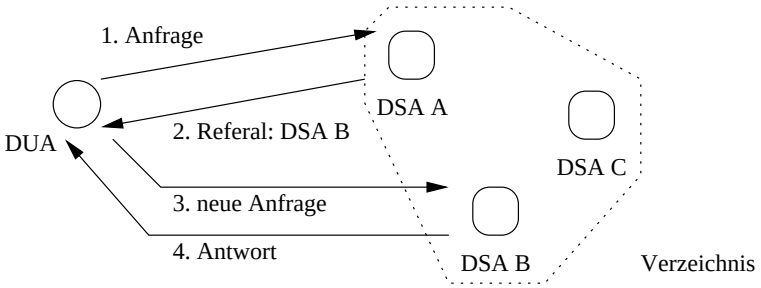

Abbildung 2.6: Referal.

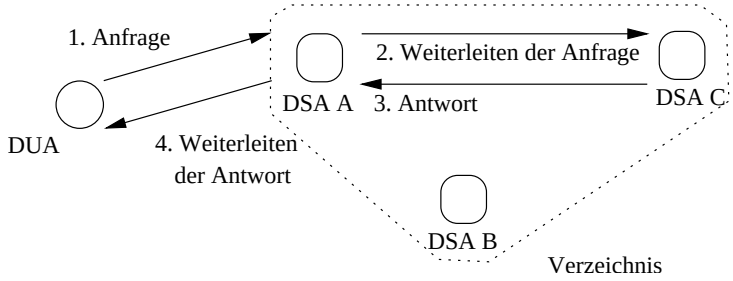

Abbildung 2.7: Chaining.

ren wird in Kapitel 3 näher erläutert. Eine Replikation zum Caching von Daten ist in X.500 zwar vorgeschlagen, aber nicht definiert.

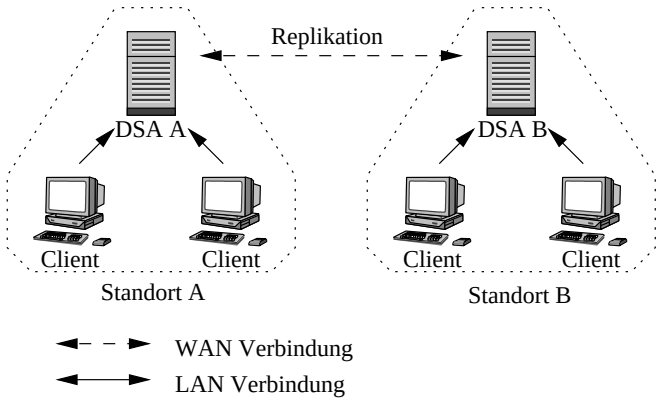

Abbildung 2.8: Replikation zwischen verteilten Standorten.

Die Replikation eignet sich zum Beispiel sehr gut zur Verbindung zweier Standorte zwischen denen nur eine langsame WAN Verbindung besteht. Wie Abbildung 2.8 auf Seite 12 zeigt, wird auf jedem Standort ein Replikat der Daten angelegt. Die Clients können so schnell auf die Daten jeder Kopie zugreifen und der Verkehr über WAN wird verringert. Durch die Replikation

wird auch die Ausfallsicherheit und Verfügbarkeit erhöht. Denn bei Ausfall eines DSA sind Duplikate vorhanden, die Clientanfragen beantworten können [14].

2.2.5 Sicherheitsmodell

Die Sicherheit spielt bei Verzeichnisdiensten eine sehr wichtige Rolle. In Verzeichnissen werden oft Benutzerinformationen oder andere sensitive Daten gehalten. Um unberechtigten Zugang zu den Daten zu verhindern, muß sich jeder DUA beim Zugriff auf einen DSA authentifizieren. Er übermittelt also seine Identität, zum Beispiel einen DN aus dem DIT, und beweist diese mit einem Paßwort oder einer anderen Legitimation. Es wird in drei Arten der Authentifizierung, den "*Authentication level*", unterschieden [54]: *none*, *simple* und *strong*.

Beim Authentication level *none* identifiziert sich der DUA nicht gegenüber dem DSA. Das ist bei Verzeichnisdiensten, die allgemein lesbar sein sollen sehr sinnvoll. Ein Beispiel für solche Dienste wäre ein öffentliches Telefon- und Adressverzeichnis. Das zweite Level ist die einfache Authentifizierung. Dazu gehören alle Methoden, bei denen sich der Benutzer über eine unsichere Methode identifiziert. Das ist zum Beispiel *plain-text*, die unverschlüsselte Übermittlung von Benutzername und Paßwort. Bei diesen Methoden können die Zugangsdaten relativ leicht im Netz abgehört und dann weiterverwendet werden. Bei der starken Authentifizierung wird mit digitalen Signaturen gearbeitet. Diese garantieren eine hohe Sicherheit gegen unberechtigte Zugriffe und das Ausspionieren von Paßwörtern im Netz.

Sobald ein DUA authentifiziert ist, wird vor jeder seiner Operationen überprüft, ob er dazu berechtigt ist. Dazu enthält das Verzeichnis die "*Access control information*" [2, 54], die für jeden Eintrag und für jeden Benutzer die erlaubten Zugriffe definiert. Eine "*Access control decision function*" entscheidet anhand von Identität des Benutzers, seinem Authentication Level, verlangter Operation und verlangtem Eintrag, ob die Erlaubnis erteilt wird. Diese Entscheidung wird für jeden Eintrag, jedes Attribut und jeden Attributwert neu getroffen.

Durch die Abhängigkeit des Zugriffs vom Authenticaton level kann zum Beispiel verhindert werden, daß ein Benutzer sein Paßwort über eine unverschlüsselte Verbindung ändert. Nur bei einer starken Authentifizierung ist das Risiko minimal, daß ein Angreifer in den Besitz des neuen Paßwortes gelangt.

Wenn ein Zugriff verweigert wird, kann der Verzeichnisdienst entweder eine Fehlermeldung an den Benutzer zurückliefern und die gesamte Operation

abbrechen. Oder aber er überspringt diesen Zugriff und fährt fort, eventuell mit einer Nachricht an den Benutzer. Das ist sinnvoll, wenn der Benutzer einen Eintrag auslesen möchte, aber nur für einen Teil der Attribute Leseberechtigung hat.

2.3 Lightweight Directory Access Protocoll

2.3.1 Geschichte

Directory Services nach X.500 setzten sich kaum durch. Ein Grund dafür war das auf dem OSI Protokoll Stack basierende "Directory Access Protocol" [15]. Es benötigte zu viele Ressourcen um effektiv genutzt werden zu können. Um dem Abhilfe zu schaffen wurden 1991 von der IETF zwei Protokolle veröffentlicht, welche den TCP/IP Stack benutzten: "*Directory Assistance Service*" (DAS) [41] und "*Directory Interface to X.500 Implemented Efficiently*" (DIXIE) [40]. Bei der Implementation kommuniziert der Client nicht mehr direkt mit dem X.500 Verzeichnis, sondern mit einen Gateway. Dieser übersetzt die Anfragen in DAP-Anfragen und sendet sie an den X.500 Server weiter. Der Nachteil beider Protokolle war die sehr enge Bindung an eine spezielle X.500 Implementation [16].

Deshalb wurde von OSI und IETF das "*X.500 Lightweight Directory Access Protocol*" (LDAP) entwickelt, das ebenfalls auf TCP aufbaut und das mit jedem X.500 Verzeichnis verwendet werden kann. Die Veröffentlichung erfolgte 1993 als LDAPv2 in RFC1487 [35] und 1995 in RFC1777 [36]. LDAPv1 wurde nicht in einem RFC beschrieben. LDAP bietet gegenüber DAP fast den gleichen Umfang an Funktionalität, bei weitaus geringeren Kosten. Durch die Verwendung von TCP entfällt der grosse Ressourcenverbrauch des kompletten OSI Netzwerk Stacks. Die Datenkodierung wird gegenüber X.500 vereinfacht, indem in LDAP die meisten Datentypen durch einfache Zeichenketten repräsentiert werden und nur ein Teil der Kodierungsregeln aus X.500 verwendet werden muß [16]. Das LDAP Protokoll wird wie X.500 mit Hilfe von ASN.1 definiert.

Wie DAS und DIXIE wurde LDAP zuerst nur über Gateways eingesetzt. Eine der ersten Implementationen war *ldapd* der Universität Michigan [1]. Diese stellte 1995 fest, daß die Anzahl der DAP Anfragen gegenüber den LDAP Anfragen auf ihre Verzeichnisse sehr gering war. Als Konsequenz wurde der erste Standalone LDAP Server *slapd* entwickelt, der durch den Wegfall des LDAP-Gateways wesentlich einfacher und leistungfähiger war als *ldapd*[16]. An dieser Stelle wurde aus dem Protokoll LDAP ein Verzeichnisdienst und LDAP selbst unabhängig von X.500.

[1] siehe: http://www.umich.edu/~dirsvcs/ldap/

Reine LDAP Server verfügen nicht über alle Eigenschaften von X.500 Verzeichnissen. Das Datenmodell und die Namenskonventionen wurden übernommen. Die Sicherheit wurde erst mit LDAPv3 an die starken Authentifizierungsmethoden von X.500 angepaßt. Die Operationen unter LDAP unterscheiden sich von denen unter DAP und werden in Sektion 2.3.3 erläutert.

Der Standard LDAPv3 wurde 1997 in RFC2251 [37] veröffentlicht und ist heute noch gültig.

2.3.2 LDAPv3

Nach RFC2251 [37] ist LDAPv3 abwärtskompatibel zu LDAPv2. Jeder Version 2 Client kann also auch mit einem Version 3 Server kommunizieren. Auf jedem LDAPv3 Server sind Informationen verfügbar, die seine Version, die unterstützen Schemas, unterstützte Authentifizierungs- und Verschlüsselungsverfahren und anderes beschreiben. Diese Informationen sind DSA spezifisch und daher als Attribute des Root Objektes definiert. Ein Beispiel ist das mehrwertige Attribut "supportedSASLMechanisms", das die unterstützten SASL Verschlüsselungsmechanismen aufzählt. LDAPv3 hat aber noch weitere Vorteile gegenüber der Version 2.

Durch die Verwendung des ISO-10646 Zeichensatzes [48] und dessen Codierung nach UTF-8 [42] kann jedes internationale Schriftzeichen im Verzeichnis verwendet werden. Attribute und DN der Einträge können also in jeder beliebigen Sprache gespeichert werden. In LDAPv2 wurde nur der IA5 Zeichensatz [11] unterstützt.

Die Authentifizierung an LDAPv3-Servern wird durch die Unterstützung "*Simple Authentication and Security Layer*" (SASL) [43] flexibler. Die Menge an Authentifikationsmethoden, die SASL bereithält, gibt sehr vielen Anwendungen die Möglichkeit sich gesichert am Verzeichnis anzumelden.

Version 3 unterstützt im Gegensatz zu Version 2 Referals bei verteilten Verzeichnissen.

Als letzter Unterschied sei noch die Erweiterungsfähigkeit von LDAPv3 erwähnt. Mit den so genannten *"Extended Operations"* können neue LDAP Operationen geschaffen oder bestehende verändert werden, ohne das Protokoll zu verletzen. LDAP ist also auch in Zukunft anpassungsfähig.

Zwischen LDAPv3 und v2 gibt es noch weitere Unterschiede, die hier aber auf Grund ihres geringen Gewichtes nicht alle aufgezählt werden. Alle weiteren Ausführungen werden sich auf LDAPv3 beziehen, da es der aktuelle

Standard ist.

2.3.3 LDAP Operationen

Die Verbindung zwischen LDAP Client und Server wird vom Client mit einem *"Bind Request"* aufgebaut. In diesem übermittelt er seine Identifikations- und Autorisationsdaten. Sind sie gültig, bestätigt der Server die Verbindung. Ansonsten wird sie abgelehnt und beendet. Nach einem erfolgreichen Bind werden vom Client alle gewünschten Transaktionen ausgeführt.
Die Verbindung wird durch einen *"Unbind Request"* beendet, den der Server nicht mehr beantwortet. Die meisten Serverimplementationen beenden eine Verbindung auch nachdem sie eine bestimmte Zeit lang inaktiv war.

Der Client kann dem Server mehrere Anfragen parallel stellen. Es können also weiter LDAP Anfragen gesendet werden, auch wenn die erste noch nicht beantwortet ist. Da der Server entscheidet, in welcher Reihenfolge diese Anfragen bearbeitet, ist LDAP ein asynchrones Protokoll [55]. Jede Anfrage erhält eine eindeutige ID, mit der sich die zugehörigen Antworten identifizieren lassen. Wenn die Antwort des Servers zu lange auf sich warten läßt, ist es dem Client möglich mit einem *"Abandon request"* eine Transaktion abzubrechen. Gibt es mehrere Antworten auf eine Anfrage, zum Beispiel bei einer Suchanfrage, werden die Teilergebnisse einzeln zum Client geschickt. Nachdem der letzte Teil versandt wurde, wird dem Client noch ein Ergebnis-Code geschickt, der die Antwort abschließt [14]. Die Teile verschiedener Antworten können in beliebiger Reihenfolge den Client erreichen. Clients müssen bei LDAP nicht auf den vollständigen Abschluß der Operation warten. Sie können schon mit den ersten Teilergebnissen arbeiten, während der Server die Anfrage noch bearbeitet.

In den LDAP API für viele Programmiersprachen, zum Beispiel C oder Java, sind trotzdem synchrone LDAP Anfragen vorhanden. Deren Verwendung bei der Implementations ist weniger aufwendig und für Einsteiger leichter zu verstehen. Bei der synchronen Variante wird vom Client eine asynchrone Anfrage gestartet und so lange blockiert, bis das vollständige Ergebnis eingetroffen ist.

Die meistgenutzte Operation an einem Verzeichnis ist die Suchfunktion *search*. Diese wird mit folgenden Parametern aufgerufen.

baseObject ist der DN eines Objektes im DIT der als Startpunkt der Suche dienen soll.

scope definiert Tiefe der Suche im DIT von Base aus. Ein Scope vom Typ *wholeSubtree* initiiert die Suche im gesamten Teilbaum unterhalb des

Startpunktes, *singleLevel* in allen direkten Nachfolgern oder *baseObject* nur im Startpunkt.

derefAliases gibt an, ob und wie Alias-Definitionen bei der Suche nach dem baseObject und nach dessen Nachfolgern aufgelöst werden.

sizelimit bestimmt die maximale Anzahl an Einträgen, die von der Suche zurückgegeben werden darf.

timelimit setzt die maximale Zeit für die Suche.

typesonly legt fest, ob nur die Namen der Attribute der gefundenen Objekte oder auch ihre Werte zurückgegeben werden sollen.

filter gibt die Bedingungen an, die die gesuchten Objekte erfüllen sollen. Dazu werden Attributewerte mit Konstanten verglichen. Mehrere dieser Vergleiche können logisch kombiniert werden.

attributes ist die Menge von Attributen, die von den gefundenen Objekten jeweils zurückgegeben werden sollen. Ist die Menge leer, werden alle Attribute zurückgegeben.

Eine spezielle list- oder read-Funktion wie in DAP ist nicht in LDAP vorgesehen. Sie können aber durch verschiedene Scope Angaben simuliert werden. Eine Suche mit Scope singleLevel emuliert die Funktion list. Mit Scope objectBase wird read erzeugt.

Neben der Suche ist die Vergleichfunktion *compare* der einzige lesende Zugriff. Sie empfängt als Parameter den DN eines Eintrags, einen Attributnamen und einen Wert. Stimmt der Wert des Attributes im Eintrag mit dem übergebenen Wert überein, wird die Operation erfolgreich beendet, ansonsten nicht. Da der Vergleich nur auf dem Server stattfindet, erhält der Client bei Mißerfolg nicht den tatsächlichen Wert des Attributes. Diese Vorgehensweise ist sehr nützlich bei Attributen, die nicht gelesen werden dürfen, wie zum Beispiel Paßwörtern für Benutzer.

Durch die Operation *add* wird ein neues Blatt in den DIT eingefügt. Sie benötigt dazu die DN des Eintrags und eine Liste aller zu erstellenden Attribute. Zu dieser Liste gehören die Objektklasse, alle Attribute die den RDN des Objektes bilden und alle anderen zwingenden Attribute. Die zwingenden Attribute werden durch die Objektklasse bestimmt.

Das Gegenstück zu *add* ist *delete*. Mit dieser Funktion wird ein einzelnes Blattobjekt aus dem DIT entfernt. Ein Objekt mit Nachfolger kann nicht entfernt werden. Beim Löschen eines Alias-Objektes wird es nicht dereferenziert.

Mit der *modify* Operation werden im DIT vorhandene Objekte verändert. Dazu wird als Parameter die DN des zu ändernden Eintrags und eine Liste von Modifikationen übergeben. Die gesamte Liste wird als atomare Operation behandelt. Einzelne Elemente können das Schema verletzen, die Gesamtoperation muß aber immer wieder in einen konsistenen Zustand führen. Dadurch lassen sich zum Beispiel zwingende Attribute erst entfernen und dann wieder hinzufügen. Die Attribute, die den RDN des Eintrags bilden, dürfen auch dann nicht entfernt werden, wenn sie keine zwingenden Attribute sind. Jede Modifikation der Liste löscht, ersetzt oder erstellt ein Attribut des Eintrags. Bei mehrwertigen Attributen können auch einzelne Werte des Attributes gelöscht oder ersetzt werden.

Einträge können durch die *modifyDN* Funktion umbenannt oder im DIT verschoben werden. Dazu wird entweder mit dem Parameter *newrdn* der neue RDN angegeben oder mit *newSuperior* der neue Vorgängerknoten.

2.3.4 Controls und Extended Operations

Extended Operations wurden in LDAPv3 eingeführt um die zusätzliche Definition von Operationen zu erlauben, ohne das Protokoll neu definieren zu müssen. Implementationen können so noch Dienste anbieten, die nicht vom LDAP Standard erfaßt werden. Die Anfrage einer Extended Operation durch den Client heißt *"Extended Request"*, die Antwort *"Extended Response"*.

Ein Extended Request besteht aus einem eindeutigen *"Object Identifier"* (OID) und einem optionalen Wert "RequestValue". Der OID ist eine Zeichenkette, die dem Extended Request eindeutig eine Operation zuordnet. RequestValue enthält die Parameter für diese Operation.

Die Extended Response enthält einen Ergebnis-Code, einen optionalen OID und einen optionalen Wert. Im Wert ist die Antwort auf die Anfrage gekapselt. Wenn der Server den OID des Extended Request nicht kennt, wird nur der Ergebis-Code zurückgeliefert. Er enthält in diesem Fall eine Fehlermeldung.

Der OID, die Syntax und die Semantik der Parameter und Antworten wird in offenen Standards, wie den RFCs, oder vom Hersteller der Implementation festgelegt. Ein Beispiel für eine Extended Operation ist der *"sendAllUpdatesRequest"* von Novell. Er hat den OID 2.16.840.1.113719.1.27.100.23 und löst die Replikation eines Novell eDirectory Servers zu allen seinen Replikationspartnern aus.

Um die Möglichkeiten der anderen LDAP Operationen zu erweitern, wurden sie in LDAPv3 um die sogenannten *"Controls"* ergänzt. Jede der in der

vorherigen Sektion genannten Operationen darf ein oder mehrere Controls als Parameter hinzugefügt werden. Diese bestehen aus OID, Kritizität und einem optionalen Wert. Die Funktion OID und optionaler Wert ist analog zu denen der Extended Operations. Die Kritizität gibt an, ob die Operation auch dann durchgeführt werden soll, wenn der Server die OID nicht kennt. Die Antworten des Servers können ebenfalls Controls enthalten. Ein Beispiel ist die *"LDAP Server Notification"* mit der OID 1.2.840.113556.1.4.528 von Microsoft [22]. Eine *search* Anfrage mit dieser Control veranlaßt, daß Active Directory den Client benachrichtigt, sobald Änderungen in der Datenbank vorgenommen werden. Der Client erhält dann als Antwort auf die Suchanfrage den neuen Eintrag. Die ganze Operation wird asynchron behandelt und erst dann abgebrochen, wenn der Client einen *abandon request* sendet oder die Verbindung getrennt wird.

Eine Liste der vom Server unterstützten Controls und Extended Operations befindet sich bei jedem LDAPv3 Server im Root Objekt. Die OIDs sind in den mehrwertigen Attributen "supportedControls" und "supportedExtensions" für jeden Client abrufbar.

Kapitel 3

Replikation

LDAP beschreibt nur den Zugang zum Server. Standards für Replikation, Partitionierung und die Kommunkiation zwischen DSA werden nicht in den RFC erwähnt. Als Folge verwirklichen die Hersteller die Verteilung der Verzeichnisse unterschiedlich.

Die IETF arbeitet aus diesem Grund seit November 1998 an einem Standard für "*LDAP Duplication/Replication/Update Protocols*" (LDUP) [1]. Ziel ist es, ein Replikationsmodell für LDAP Server zu entwickeln. Im November 2002 wurde ein Standard für die Bedingungen einer LDAP Replikation unter RFC 3384 [46] herausgegeben.
In diesem Kapitel wird zuerst die Replikation nach X.500 und LDUP näher erläutert. Danach wird auf die unterschiedlichen Ansätze der Replikation von OpenLDAP, Novell Directory Services und Microsoft Active Directory eingegangen.

3.1 Shadowing

X.500 definiert das Shadowing von Daten im Verzeichnisdienst. Für den zu replizierenden Teilbaum des DIT, die "*Replication area*", wird genau ein "*Master DSA*" bestimmt. Nur auf diesem sind Schreiboperationen erlaubt.

Um die Replikation zwischen zwei Servern zu ermöglichen, muß als erstes ein " *Shadowing agreement*" zwischen diesen beiden bestehen. Darin wird in "Shadow supplier", der die Daten liefert, und "Shadow Consumer", der sie empfängt, unterschieden. Der Master DSA ist für seine Replication area immer der Shadow supplier. In X.525 sind zwei Ansätze definiert, wie eine Replikationstopologie aussehen kann. Beim "Primary shadowing" gibt es nur einen Shadow supplier, den Master DSA, und alle Shadow consumer erhalten ihre Daten direkt von ihm. Das "Secondary shadowing" kann auch

[1] siehe: http://www.ietf.org/html.charters/ldup-charter.html

ein Shadow consumer die Rolle des Shadow suppliers für andere DSA übernehmen. Diese erhalten die Daten nur noch indirekt vom Master DSA.

Das Shadowing agreement enthält weiterhin einen "*updateMode*", der festlegt, wann die Synchronisation erfolgt. Das Update kann sowohl vom Supplier als auch vom Consumer periodisch nach Ablauf einer festgelegten Zeitspanne initiiert werden. Nur der Supplier hat die zusätzliche Möglichkeit Veränderungen in der Replication area an den Consumer im Modus "*onChange*" weiterzuleiten, also sofort nachdem er sie registriert.

Zur Initialisierung eines Shadowing agreements wird ein totales Update durchgeführt. Es werden alle Einträge der Replication area vom Supplier zum Consumer übertragen. Im Gegensatz dazu werden spätere Updates nur noch inkrementell durchgeführt. Der Supplier sendet nur die Veränderungen der Daten seit der letzten Synchronisation. Das spart Zeit und verringert die übertragene Datenmenge.

Modifikationsanfragen von Clients an Shadow consumer werden grundsätzlich nicht von ihm bearbeitet sondern wie in Abbildung 3.1 mittels Chaining oder Referal an den Master DSA weitergeleitet. Beim Primary shadowing passiert das direkt, beim Secondary shadowing dagegen kann es auch indirekt durch suksessive Weiterleitung an den jeweiligen Supplier geschehen.

Shadowing läßt genau zwei Fälle zu, bei denen ein DSA gleichzeitig Shadow Supplier und Shadow Consumer ist. Im ersten Fall ist er Shadow supplier und consumer von verschiedenen Replicaton areas. Im zweiten nimmt er beide Rollen für die gleiche Replication area an, aber in Bezug zu unterschiedlichen DSA.

3.2 LDUP

Shadowing wird unter LDUP als "Master-Slave" oder auch "Single-Master" Replikation bezeichnet. Der Nachteil des Shadowing ist der einzelne Master DSA. Er ist ein *single point of failure*[2], denn bei einer Störung können keine neuen Änderungen mehr am Datensatz vorgenommen werden. Erst wenn ein Ersatz die Masterfunktion übernehmen kann oder der Master wieder hergestellt ist, kann die Replikation fortgeführt werden. Single-Master ist auch schlecht geeignet zur Überbrückung von langsamen WAN Verbindungen mit einem Master auf der einen Seite und ein- oder mehreren Consumer auf der anderen. Jede Änderung auf der Consumer Seite wird zweimal übertragen, einmal zum Master und einmal zurück zum Consumer. Der zeitliche Auf-

[2]single point of failure: fällt dieser Punkt aus, so wird die ganze Topologie funktionsuntüchtig.

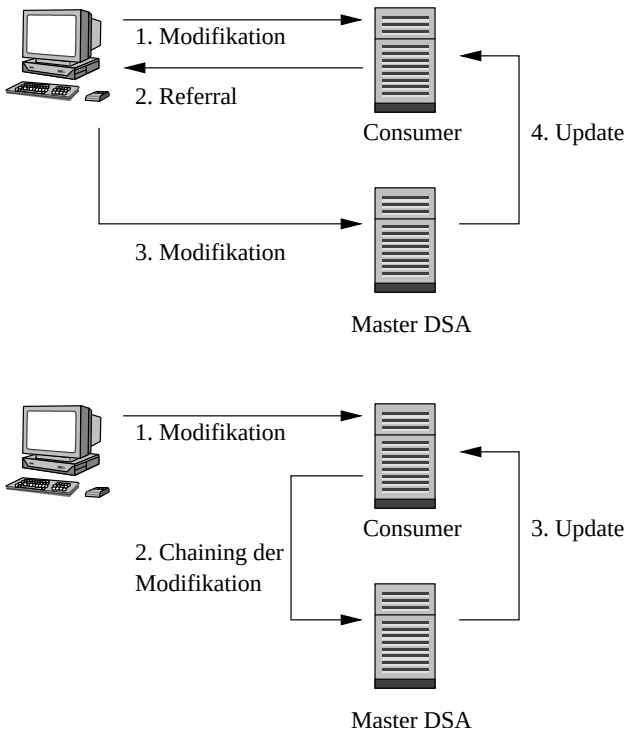

Abbildung 3.1: Single-Master Replication
Modifikationen werden an den Master weitergeleitet.

wand für eine Aktualisierung ist hier besonders hoch.

LDUP enthält den Entwurf einer "Multi-Master" Replikation, bei der mehr als ein Master DSA zulässig ist. Auf jedem Master können Schreibzugriffe durchgeführt werden, ohne daß vorher ein anderer Master kontaktiert werden muß. Eine gleichzeitige Änderung desselben Eintrags der Replication area auf verschiedenen Master DSA darf jedoch nicht zu einer Inkonsistenz des Verzeichnisses führen. Dafür sollen verschiedene Daten sorgen, die für jeden Eintrag verwaltet werden, und im Zweifelsfall zur Konfliktlösung beitragen. Die Single-Master Replikation wird in LDUP als eine spezielle Multi-Master Lösung mit nur einem Master betrachtet.

Teile des LDUP Projektes, wie "*LDAP Client Update Protocol* (LCUP) [59], befinden sich noch in der Planungsphase. Die Entwürfe entsprechen allerdings Replikationsmethoden schon verhandener Implementationen. Deshalb werden an dieser Stelle die Erweiterungen zum Datenmodell und die Prozeduren zur Konfliktlösung erläutert, die laut LDUP für eine Implementierung von Multi-Master Replikation notwendig sind.

3.2.1 Konsistenz

Sowohl X.500 als auch LDUP erlauben in einem verteilten Verzeichnis kurzlebige Inkonsistenzen. Sie treten auf, wenn auf einem DSA Veränderungen vorgenommen und noch nicht zu den anderen repliziert wurden. Die Inkonsistenz muß mit dem nächsten Replikationszyklus beseitigt werden. Diese Art der Konsistenz wird in X.500 "*Kurzlebige Konsistenz*" und in LDUP "*Lose Konsistenz*" bezeichnet. Die Daten innerhalb eines DSA bleiben konsistent.

3.2.2 Unique Identifier

Im Gegensatz zum Verzeichnisdienst auf einem Server oder in einer Single-Master Topologie reicht bei einer Multi-Master Replikation der DN eines Eintrags nicht mehr als eindeutige Identifizierung aus. Der DN eines Eintrags kann umbenannt werden, durch ModDN, oder aber auch gelöscht, und zwar dann, wenn auch der Eintrag gelöscht wird. Als Beispiel wird auf einem Master A ein Eintrag E umbenannt oder gelöscht. Gleichzeitig wird auf Master B eben dieser Eintrag verändert. Master B kann diese Änderung nicht anhand der DN an A übertragen, da diese DN auf A nicht mehr existiert. Oder es wird der falsche Eintrag verändert, falls in der Zwischenzeit ein neuer Eintrag dieses Namens auf A erstellt wurde.

Aus diesem Grund verlangt RFC3384 für jeden Eintrag einen verzeichnisweit einzigartigen Bezeichner, der während der gesamten Lebenszeit des Eintrags nicht geändert werden darf. Er wird "*Unique Identifier*" (UID) genannt. Alle Replikationsoperationen addressieren einen Eintrag an Hand seines UID. Die globale Eindeutigkeit des UID kann durch ein spezifisches Prefix des erstellenden DSA und ein lokal eindeutigs Suffix erreicht werden kann. Ein genauer Algorithmus zur Erzeugung wird in LDUP erst in Zukunft erscheinen[59]. Der UID wird als operationales einwertiges Attribut des Eintrags gespeichert.

3.2.3 Konfliktlösung

Wenn zwischen zwei Replikationszyklen auf zwei verschiedenen Master DSA dieselben Daten verändert werden, entstehen Konflikte. Um die verlangte Konsistenz zu erhalten, müssen beide Server nach der Replikation im selben Zustand sein. Die Lösung der Konflikte kann das Verwerfen von bereits vorgenommenen Änderungen auf einem oder beiden DSA bedeuten. Damit keine Daten verloren gehen, werden die verworfenen Daten und deren Ablehnung in einem Log vermerkt. Ein Administrator kann eventuelle Fehler wieder aufheben.

Die Entscheidung welche Daten bestehen bleiben, übernimmt ein bestimmter Algorithmus. Er muß so angelegt sein, daß jeder DSA zu der gleichen

Entscheidung gelangt und die Konsistenz des Verzeichnis gesichert ist. Wie die Konfliktlösung genau erfolgen soll, ist nicht im RFC definiert. Bestehende Implementationen bedienen sich einer "*Change Sequence Number*" (CSN)[3]. Bei jeder Änderung wird die Ausführungszeit in Form einer CSN gespeichert. Im Konfliktfall wird der Eintrag mit der spätesten CSN als der Aktuellere übernommen, alle anderen werden verworfen.

Wenn jeder Eintrag mit genau einer CSN versehen wird, ist eine Konfliktlösung bestenfalls auf Objektebene möglich. Es wird das ganze Objekt bei der Änderung übertragen, auch wenn nur ein Attribut geändert wurde. Es werden eventuell mehr Änderungen verworfen als nötig. Wenn gleichzeitig auf Server A das Attribut X und auf Server B das Attribut Y desselben Eintrags geändert werden, kann die Konfliktlösung auf Objektebene nur eine der beiden Änderungen übernehmen. Wenn zum Beispiel der Eintrag auf Server A eine frühere CSN erhält als der auf Server B, dann werden alle Attributwerte von B nach A repliziert. Die Änderung von X wird rückgängig gemacht.
Bei einer Lösung auf Attributebene wird jedem Attribut eine CSN zugeordnet. Das Verzeichnis muß mehr Daten verwalten, kann aber genauer Konflikte lösen. Die genaueste Konfliktlösung findet auf Attributwertebene statt. Es erhält jeder Attributwert seinen eigenen Zeitstempel.

Diese Methode die Updates zu ordnen hat einen wichtigen Nachteil: alle Server müssen synchrone Uhren haben. Geht die Uhr eines Servers zu sehr nach, werden alle von ihm stammenden Änderungen verworfen. Das NTP [51] oder das SNTP Protokoll [52] bietet hier eine Lösung zur Synchronisation der Serveruhren an.

3.2.4 Problemfälle

In einer Multi-Master Umgebung kann es passieren, daß ein Replication Update aus verschiedenen Gründen nicht mehr durchgeführt werden kann. Einige dieser Fälle wurden in den "*LDUP Update Reconciliation Procedures*" (URP) [57] vorgestellt. Die Problemfälle werden im Folgenden erläutert:

- Ein Update bearbeitet einen bereits gelöschten Eintrag, ein Attribut oder einen Attributwert.
- Ein Update erstellt einen Nachfolger eines gelöschten Eintrags.
- Ein Update erzeugt einen Eintrag mit schon vorhandenem DN.
- Ein Update weist einwertigen Attributen einen zusätzlichen Wert zu.

[3]Der Begriff CSN wurde in [57] verwendet.

- Ein Update verletzt das Schema.
- Ein Update führt zu Kreisen im DIT, indem ein Eintrag sein eigener direkter oder indirekter Vorgängerknoten wird.

Die in URP vorgestellten Lösungen wurden nicht als RFC übernommen und der Entwurf verworfen. Sie werden daher an dieser Stelle nicht angeführt.

3.3 OpenLDAP

OpenLDAP [4] ist eine Weiterentwicklung der LDAP Implementation der Universität Michigan Version 3.3. Es ist ein Open Source Projekt und unter der OpenLDAP Public License verfügbar. OpenLDAP umfaßt neben einem LDAP Server auch den Replikationsdienst *slurpd*, LDAP Clients für den Zugriff und Bibliotheken zum Entwickeln von LDAP Anwendungen. Ab der Version 2.0 unterstützt OpenLDAP LDAPv3. Die aktuelle stabile Version ist 2.0.23. Es existiert gleichzeitig eine experimentelle Version, in der neue Entwicklungen enthalten sind.

OpenLDAPs Verzeichnisdienstserver *slapd* besteht aus zwei Teilen. Das Front-end kommuniziert mit den Clients über das LDAP Protokoll. Das Back-end verwaltet den Datenzugriff. Dafür stehen in der Implementation verschiedene Varianten zur Verfügung. Die Gebräuchlichste ist *back-ldbm*, die Daten im Berkeley B-Baum DB Format speichert. Dazu wird die Berkeley DB Implementation von Sleepycat [5] genutzt. Weiterhin sind folgende Back-ends für OpenLDAP fertig gestellt:

back-shell erlaubt den Datenzugriff durch die Ausführung von externen Programmen (für gewöhnlich Shell Scripte).

back-passwd erlaubt den lesenden Zugriff auf die UNIX /etc/passwd Datei.

back-ldap leitet die Anfragen an einen anderen LDAP Server weiter. OpenLDAP agiert mit back-ldap als LDAP Proxy.

back-sql erlaubt den Zugriff auf eine relationale Datenbank.

back-dnssrv den Einsatz von OpenLDAP als LDAP Root Server nach RFC 3088 [44].

Die stabile OpenLDAP Version unterstützt nur Single-Master Replikation. slapd erzeugt eine Logdatei, in der jede ausgeführte Änderungsoperation im

[4] siehe: http://www.openldap.org/
[5] siehe: http://www.sleepycat.com

"LDAP Data Interchange Format" (LDIF) [47] gespeichert wird. Der Dienst slurpd überprüft das Logfile regelmäßig auf neue Einträge und sendet diese per LDAP an die Consumer, in [60] "*Slaves*" genannt. slurpd identifiziert sich beim Zugriff auf den Slave mit dem DN "*UpdateDN*", nur über ihn sind Schreibzugriffe auf Slave Servern möglich. Der Master Server erstellt automatisch für jeden Eintrag die in Tabelle 3.1 gezeigten operationalen Attribute. Sie werden genau in RFC 2252 definiert [38]. Slave Server erzeugen diese Attribute nicht selbst, sondern übernehmen sie vom Master.

Attribut	Beschreibung
creatorsName	DN des Benutzers, der den Eintrag erstellt hat
createTimestamp	Zeitstempel des Erstellungsdatums
modifiersName	DN des Benutzers, der den Eintrag als letzter modifiziert hat
modifyTimestamp	Zeitstempel der letzten Modifikation des Eintrags

Tabelle 3.1: operationale OpenLDAP Attribute

In der experimentellen Version ist zusätzlich eine Multi-Master Replikation implementiert. Wie bei der Single-Master Replikation besteht jeder Master aus jeweils einem slapd Dienst, der ein Logfile erzeugt, und einem slurpd Dienst. Der Unterschied besteht darin, daß Änderungen, die von UpdateDN durchgeführt wurden, nicht im Logfile vermerkt werden. Sie sind auch die einzigen Schreiboperationen, die operationale Attribute ändern dürfen.

Diese Methode geht davon aus, daß alle Server ständig untereinander ohne Verzögerung erreichbar sind. Jede Änderung der Daten eines Masters soll sofort zu allen anderen Servern übermittelt werden. Eine Konfliktlösung bei gleichzeitiger Änderung des gleichen Eintrags auf verschiedenen Mastern ist (noch) nicht implementiert. Die Multi-Master Replikation von OpenLDAP muß noch starken Änderungen unterzogen werden, bevor sie einsatzfähig ist.

3.4 Novell eDirectory

Novell veröffentlichte 1993 mit Netware 4.0 seinen "*Novell Directory Service*" (NDS). NDS ersetzte das mit NetWare 2 und 3 verwendete "*Bindery*". Bindery verwaltete bis dahin in Novell Netzwerken administrative Informationen. Die Netzwerkobjekte, wie Benutzer, Computer oder Drucker, wurden in einer flachen Struktur gespeichert, die nicht die Abbildung der Organisationshierarchie erlaubte. Für grössere Unternehmen war Bindery nicht skalierbar genug. Im Unterschied dazu orientierte sich NDS stark an den Konzepten von X.500, obwohl es kein X.500 Verzeichnisdienst ist. Es

war lange Zeit das Vorbild für ein skalierbares und erweiterbares Verzeichnis zur zentralen Benutzer- und Ressourcenverwaltung.

Der Zugriff erfolgte über das "*Novell Directory Access Protocl*", welches auf das "*Netware Core Protocol*" (NCP) aufbaute. NCP selbst ist kein Transportprotokoll, sondern benötigt noch IPX, Netware/IP[6] oder ab NDS 8 TCP/IP. LDAPv2 wird von NDS ab der Version 5.73 unterstützt und ab Version 7 auch LDAPv3. Die ersten Implemntationen verwendeten dazu noch einen LDAP Gateway, der die Anfragen zu NDAP umformte.

Um die Flexibilität von NDS zu erhöhen, wurde es auf andere Plattformen als Netware portiert. Heute sind auch Versionen für Microsoft Windows NT/2000, Sun Solaris, True64 UNIX und Linux verfügbar. Novell symbolisierte diesen Schritt, indem NDS mit Version 8.5 zu NDS eDirectory und mit Version 8.6 zu "*Novell eDirectory*" umbenannt wurde.

3.4.1 Replikationsmechanismen

Novell eDirectory ist ein standalone LDAP Server und unterstützt Multi-Master Replikation. Die eDirectory Datenbank wird dazu in Partitionen unterteilt. Ein Server kann mehrere Partitionen enthalten. Eine Partition kann auf mehrere verschiedene Server repliziert werden. Alle Replikate derselben Partition formen einen "*Replica ring*" [26]. Innerhalb eines Replica ring nimmt ein Replikat genau eine der sechs folgenden Rollen an.

Master Replica Für jede Partition wird genau ein Master Replica bestimmt. Für gewöhnlich übernimmt der als erstes eingerichtete Server der Partition diese Aufgabe. Auf ihm sind Schreiboperationen für Clients zulässig. Er verwaltet alle Operationen in der Partition und löst gegebenenfalls Konflikte. Der Master Replica verwaltet auch das Neuerstellen, Zusammenfügen, Umbewegen oder Löschen von Partitionen.

Read/Write Replica In einer Partition kann es beliebig viele Read/Write Replicas geben. Sie enthalten die gleichen Informationen wie der Master. Auf ihnen sind Schreiboperationen für Clients zulässig.

Read-Only Replica In einer Partition kann es beliebig viele Read-Only Replicas geben. Sie enthalten die gleichen Informationen wie der Master. Schreiboperationen sind für Clients nicht zulässig.

Filtered Read/Write Replica In einer Partition kann es beliebig viele Filtered Read/Write Replicas geben. Sie enthalten nur einen Teil der Informationen des Master Replicas. Schreiboperationen sind für Clients zulässig. Welcher Teil der Informationen auf ihnen gehalten wird, legt ein Filter fest.

[6] in Netware/IP wird IPX Protokoll durch IP gekapselt

Filtered Read-Only Replica In einer Partition kann es beliebig viele Filtered Read-Only Replicas geben. Sie enthalten nur einen Teil der Informationen des Master Replicas. Schreiboperationen sind für Clients nicht zulässig. Welcher Teil der Informationen auf ihnen gehalten wird, legt ein Filter fest.

Subordinate Reference Replica Wenn auf einem Server eine Elternpartition gespeichert ist, deren Kindpartition[7] kein Replikat auf dem Server besitzt, wird automatisch ein Subordinate Reference Replica angelegt. Dieses Replikat wird wieder entfernt, wenn ein Read-Only oder Read/Write Replica der Kindpartition in die Elternpartition eingefügt wird.

Ein Multi-Master Szenario für eine Partition ergibt sich, sobald neben dem Master mindestens ein Read/Write Replica erstellt wird. Filtered Replicas erzeugen eine Teilansicht der Partition. Das reduziert die Größe der Datenbank des Replikats, verringert das Datenvolumen zur Synchronisation und kann sensitive Daten von öffentlichen Servern fernhalten. Mit Hilfe des Filters wird die Replikation auf eine bestimmte Menge von Objekten (z.B nur Benutzerobjekte) oder auf eine bestimmte Menge von Attributen (z.B. nur Telephonnummer, Addresse) beschränkt werden.

3.4.2 Konfliktlösung

Novell eDirectory löst Konflikte der Multi-Master Replikation auf Attributwertebene mit Zeitstempel und UID. Der UID eines Eintrags wird in dem operationalen Attribut "*GUID*" festgehalten. Der Zeitpunkt der letzten Änderungen wird für jeden Attributwert gespeichert. Er ist allerdings nicht per LDAP abrufbar. Vor dem Einspielen eines Updates wird jede Veränderung anhand des Stempels auf ihre Aktualität geprüft und gegebenenfalls verworfen, wenn sie veraltet ist.

Wie bei OpenLDAP ist die Zeitsynchronisation der Server erforderlich um eine reibungslose Replikation zu gewährleisten. Novell liefert dazu ein "*NetWare Loadable Module*" namens Timesync.nlm. Es synchronisiert die Uhren der eDirectory Server per NTP.

3.5 Microsoft Active Directory

Active Directory löst mit seiner X.500 nahen Architekter das veraltete Windows NT Domänenmodell ab. Dieses Modell hat wie in Bindery eine flache Struktur und ist nicht erweiterbar. Es kann theoretisch maximal 40,000

[7] Eine Partition ist Kind einer anderen, wenn die enthaltenen Einträge (indirekte) Nachfolger von einem Eintrag der Elternpartition sind.

Objekte enthalten, praktisch werden aber kaum mehr als 10,000 erreicht [61]. Windows NT 4.0 verwendet eine Single-Master Replikation mit einem "*Primary Domain Controller*" als Master und mehreren "*Backup Domain Controllern*" als Slave.

Active Directory wird auf dem Windows 2000 Advanced Server ausgeliefert und unterstützt LDAPv3. Außerdem wird in Windows 2000 Netzwerken Microsofts proprietäres WINS durch DNS ersetzt und das Autorisationsprotokoll NTLM durch Kerberos 5 abgelöst.

Trotz aller Neuerungen ist die Domäne noch immer die Basis für Active Directory. Sicherheitsrichtlinien und administrative Rechte werden nur innerhalb von Domänen vergeben. Diese Grenze kann aber mit "*Trust Relationships*" zwischen zwei Domänen aufgehoben werden. Benutzer einer Domäne können mit dieser Hilfe auf die Ressourcen der anderen zugreifen und Administratoren die Benutzer der jeweils anderen Domäne verwalten. Das Setzen von Trust Relationships erstellt einen "*Domain Tree*". Alle enthaltenen Domänen haben zusammenhängende DNS Namen. Sie werden wie in Abbildung 3.2 erzeugt. Domain Trees wiederum können mit Trust Relationships zu einem "*Domain Forest*" verbunden werden.

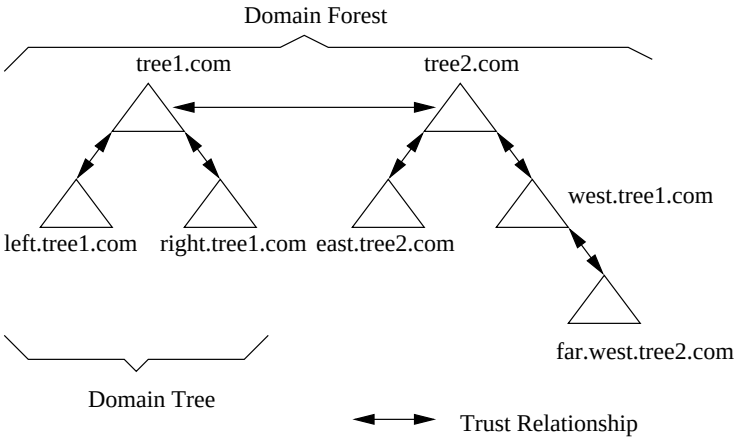

Abbildung 3.2: Domain Forest
Domain Trees und Forests werden mit Trust Relationships gebildet. Die Domainnamen müssen zusammenhängend sein.

3.5.1 Replikationsmechanismen

Ein Server, auf dem Active Directory läuft, heißt "*Domain Controller*" (DC). Er wird in mindestens drei Partitionen unterteilt. Die "*Schema Partition*" enthält alle Objektklassen und Attributtypen, die in Active Directory

gespeichert werden können. Sie wird zu allen DC des Forests repliziert. Jeder DC eines Forests benutzt also das gleiche Schema. In der "*Configuration Partition*" werden Informationen über die Replikationstopologie und dazugehörige Metadaten gespeichert. Auch sie ist auf allen DC des Forests gleich. Die "*Domain Partition*" enthält alle Objekte der Domäne, zum Beispiel Benutzer und deren Rechte. Sie wird nicht über die Domäne hinaus repliziert. Pro Domäne existiert genau ein Global Catalog. Das ist ein Domain Controller mit einer vierten Partition, die eine Teilmenge der Objekte der Domain Partition aller Domänen enthält. Die Objekte dieser Partition werden von Active Directory selbst erzeugt, der Benutzer hat also keinen Schreibzugriff. Der Global Catalog erlaubt das Auffinden von Daten über Domänen hinweg ohne einen DC der anderen Domäne kontaktieren zu müssen.

Active Directory implementiert eine Multi-Master Replikation. In einer Domäne hält jeder DC eine les- und schreibbare Kopie der Domain Partition. Für bestimmte Konfigurationsdaten wird aber mittels sogenannter "*Flexible Single Master Operation*" (FSMO) ein alleiniger Master bestimmt [19]. Die folgenden zwei FSMO Rollen sind Forest-weite Master:

Schema Master Nur auf diesem DC sind Schemaänderungen erlaubt.

DNS Master Nur auf diesem DC ist das Hinzufügen und Entfernen von Domänen im Forest erlaubt.

Die folgenden drei Rollen werden einmal pro Domäne vergeben:

RID Master Wenn ein DC einen Benutzer, Computer oder eine Gruppe erstellt, vergibt er eine Securtiy ID. Damit keine ID doppelt vergeben wird, verwaltet der Relative ID (RID) Master alle IDs. Er teilt jedem DC der Domäne einen Bereich zu, aus dem dieser Security IDs vergeben darf. Ist der Bereich ausgeschöpft, kann vom RID Master ein neuer angefordert werden.

PDC Emulator Aus Kompatibilitätsgründen zum Windows NT 4.0 Domänen Modell übernimmt ein DC die Rolle eines Primary Domain Controllers.

Infrastruktur Master Dieser DC verfolgt Objektreferenzen über Domänengrenzen hinweg. Er aktualisiert sie jedes Mal, wenn ein referenziertes Objekt verschoben wird.

3.5.2 Konfliktlösung

Die Konfliktlösung bei der Multi-Master Replikation erfolgt anders als bei OpenLDAP oder Novell eDirectory über eine nahezu zeitunabhängige Methode. Dazu wird eine "*Update Sequence Number*" (USN) verwendet.. Sie ist

ein lokaler Zähler auf jedem DC, der die Anzahl der Änderungen auf diesem DC mitzählt. Bei einer Modifikation wird die aktuelle USN als Attribut im veränderten Objekt vermerkt. Jeder DC verwaltet einen Vektor von USN, der zu jedem Replikationspartner die USN der letzten empfangenen Operation enthält. Bei der Replikation werden dann nur Änderungen übertragen, deren USN größer als die letzte empfangene ist. Der UID eines Eintrags ist im Attribut "*objectGUID*" im Binärformat enthalten.

Jedes Attribut erhält bei jeder Änderung durch einen Benutzer einen neuen Stempel. Dieser setzt sich wie folgt zusammen:

Die **Versionsnummer** wird bei jeder Änderung um eins erhöht.

Der **Zeitstempel** enthält die lokale Zeit zur Änderung.

Der **Ursprungs-DC** identifiziert den DC, auf dem diese Änderung durchgeführt wurde.

Erhält ein DC das Update eines Attributes, vergleicht er als erstes die Versionsnummern miteinander. Ist die des Updates größer, wird es übernommen. Ist sie kleiner, wird es verworfen. Bei gleicher Versionsnummer entscheidet der Zeitstempel. In den wenigen Situationen, in denen der Zeitstempel auch gleich ist, wird die Entscheidung anhand des Ursprungs-DC getroffen. In den meisten Fällen wird der Konflikt schon durch die Versionnummer behoben.

Erwähnenswert ist noch der Umgang mit mehrwertigen Attributen. Im Gegensatz zu eDirectory verwaltet Active Directory nur einen Zeitstempel pro Attribut und nicht einen pro Attributwert. Active Directory unterstützt also nur eine Konfliktlösung auf Attributebene. Es überträgt bei der Replikation das gesamte Attribut neu, auch wenn nur ein Wert verändert wurde. Beim gleichzeitigen Update eines mehrwertigen Attributes auf verschiedenen Servern geht eine der Veränderungenn bei der Replikation immer verloren. Von Microsoft wird deshalb empfohlen, in großen Domänen nur einen Master zur Änderung mehrwertiger Attribute zu benutzen.

Kapitel 4

Synchronisation

4.1 Schemadifferenzen

In einem heterogenen Netzwerk können gleichzeitig verschiedene Implementationen von LDAP Servern zur Benutzerverwaltung verwendet werden. Der Grund für den Einsatz einer solchen Topologie kann technischer oder politischer Natur sein. Wenn zum Beispiel Clients mit verschiedenen Betriebssystemen im Netzwerk existieren, kann es sein, daß jeder Typ ein anderes Schema bei der Authentifizierung der Benutzer und dem Laden ihrer Konfiguration verlangt. Jeder Client benötigt einen Punkt, an dem er Benutzer authentifizieren kann.

Die Schemadifferenz ist keines Falls ein triviales Thema. Um das zu verdeutlichen, zeigt Tabelle 4.1 auf Seite 32 einen Teil der Attribute der Benutzerobjekte von Microsoft Active Directory und Novell eDirectory. Im Beispiel ist zu erkennen, daß Attribute trotz gleicher Bedeutung nicht zwangsläufig den gleichen Namen haben müssen und umgekehrt.

Beschreibung	MS Active Directory	Novell eDirectory
Vorname	givenName	givenName
Nachname	sn	sn
vollständiger Name	displayName	fullName
Name des Standorts	physicalDeliveryOfficeName	l
Stadt des Standorts	l	physicalDeliveryOfficeName
Straße des Standorts	streetAddress	street

Tabelle 4.1: Schemadifferenzen
Schemadifferenzen zwischen Novell eDirectory und Microsoft Active Directory

Durch unterschiedliche Attributnamen für denselben Wert in verschiedenen Schemas wird es dem Benutzer unmöglich gemacht, sich mit allen Clients an einer Implementation zu authentifizieren. Das ist zum Beispiel der Fall, wenn der Benutzername in unterschiedlichen Attributen gespeichert wird. Jeder Client muß sich an einem Server anmelden, der das gleiche Schema wie er selbst verwendet. Diese Strategie wirft das Problem auf, daß die Benutzeraccounts auch auf verschiedenen Servern synchron existieren müssen.

Zur Synchronisation unterschiedlicher Verzeichnisdienste können die implementierten Replikationsmechanismen nicht verwendet werden, obwohl einige Server ihre Updates per LDAP versenden[1]. Es ist nicht sinnvoll ein Update zu versuchen, wenn es wegen ungültiger Objektklassen oder Attribute abgelehnt wird. Ein sicheres Update zu einem Server mit einem anderen Schema ist mit einfacher Replikation schlicht unmöglich.

Sollen zwei LDAP Implementationen synchron gehalten werden, müssen die Objektklassen und Attribute der Schemas einander zugeordnet werden. Wenn sich die Struktur der Verzeichnisbäume unterscheidet, muß eine solche Zuordnung auch für die DN erfolgen. Während der Synchronisation werden in jedem Updaterequest entsprechend dem Zielserver DN, Attributnamen und -werte geändert, so daß er das Update korrekt annehmen kann. Diese Transformation wird "*Mapping*" genannt und über eine Mappingtabelle gesteuert. Die Tabelle wird von einem Administrator erstellt, der die Bedeutungen der Schemas auf beiden Implementationen einander zuordnen kann.

Die Synchronisation verschiedener Servertypen ist ein allgemeines Problem und es wurden schon einige Lösungen entwickelt. An dieser Stelle werden drei bereits implementierte vorgestellt. Das sind "*LDAP Directory Synchronizer Utility*" (LDSU) von Compaq [31], "*Microsoft Directory Synchronization Service*" (MSDSS) [24] und "*DirXML*" von Novell [27]. LDSU und DirXML sind allgemeine Synchronisationsprogramme. Sie bieten die Möglichkeit LDAP Verzeichnisse mit anderen Datenbanken zu synchronisieren. MSDSS ist allein zur Synchronisation von Benutzeraccounts zwischen Active Directory und eDirectory gedacht.

In den folgenden drei Sektionen werden diese Lösungen diskutiert. Es wird auf die allgemeine Strategie, auf die Möglichkeiten zur Konfliktlösung für Multi-Master Synchronisation und auf die Vorgehensweise bei der Synchronisation von Paßwörtern. Paßwörter und mit ihnen alle anderen verschlüsselt gespeicherten Attribute stellen bei der Synchronisation ein besonderes Problem dar. Sie werden bei der Modifikation vom Benutzer unverschlüsselt übertragen, vom Server verschlüsselt und dann abgespeichert. Die Verschlüsse-

[1] zum Beispiel OpenLDAP

lung ist in den meisten Fällen nicht in sinnvoller Zeit umkehrbar. Active Directory benutzt zum Beispiel den RSA MD-4 Algorithmus [21]. Active Directory und eDirectory gehen noch einen Schritt weiter, denn sie verweigern außerdem das Auslesen der verschlüsselten Attributwerte. Das gilt auch für Benutzer mit Administrationsrechten.

4.2 Compaq LDSU

LDSU wurde 1998 in der Version 1.0-6 veröffentlicht. Es synchronisiert die Daten zwischen einem LDAP Verzeichnis und einer Textdatei. Diese Textdatei wird für gewöhnlich durch LDSU beim Exportieren der Daten aus einem Verzeichnis erzeugt. Eine zweite LDSU Instanz importiert danach die Daten in einen anderen Verzeichnisdienst. Die aktuelle LDSU Version V1.9 wurde im Januar 2002 veröffentlicht. Es ist für Windows 2000/NT, Compaq Tru64 Unix, Sun Solaris und Linux verfügbar.

LDSU erwartet beim Import eine Textdatei mit einem sehr einfachen Format. Wenn eine Datenbank im flachen Textformat darstellbar ist, läßt sie sich mit LDSU in ein LDAP Verzeichnis importieren.

4.2.1 Strategie

Eine LDSU Instanz unterstützt die Modi Export, Import, Change, Transaction, Update und Reformat. Wird sie aufgerufen, liest sie aus einer Konfigurationsdatei eine Reihe von Parametern, wie zum Beispiel den akutellen Modus und die Daten des LDAP Servers. Eine Layout Datei bestimmt die Attribute des Verzeichnisses, die LDAP benutzen soll, deren Syntax und Wertigkeit. Das Mapping der Daten erfolgt mit Regeln aus dem *"Record Description File"* (RDF). Das Format dieser drei Textdateien ist von Compaq selbst entwickelt und sie sind sehr weitreichend konfigurierbar.

Im Export Modus stellt LDSU dem LDAP Server eine Suchanfrage. Deren Base und Filter werden in der Konfigurationsdatei festgelegt. Es werden höchstens so viele Werte pro Attribut ausgelesen, wie die Layout Datei zuläßt. Jedes von der Suchanfrage zurückgelieferte Objekt wird gleich behandelt. Die Bestandteile der DN und jedes Attribut werden Variablennamen zugeordnet. Werte eines mehrwertigen Attributes haben denselben Namen mit zusätzlichem Index. Diese Variablen werden dann entsprechend dem RDF in eine Ausgabedatei geschrieben. Das RDF verfügt über eine Menge von Funktionen, mit denen die Variablen manipuliert und selektiert werden können. In der Ausgabedatei stehen die Variablen zeilen- und spaltenweise. In der einfachsten Form entsprechen alle Spalteneinträge einer Zeile den Attributen eines Objektes. Ein Objekt kann sich aber auch über mehrere Zeile erstrecken. Die Spalten werden durch ein vom Benutzer definiertes Zeichen

getrennt.

Eine LDSU Instanz im Import Modus öffnet eine Eingabedatei und wertet die Zeilen und Spalten entsprechend ihres eigenen RDF aus. Die Einträge können an Hand ihrer Position adressiert werden. Den Variablennamen aus dem Layout wird beim Abarbeiten des RDF ein Wert zugewiesen. Dadurch entsteht der DN und die Attribute des neuen Objektes. Die erzeugten Objekte werden mit dem realen Inhalt des Verzeichnisses verglichen. LDSU nimmt danach Änderungen so vor, daß das LDAP Verzeichnis der Eingabedatei entspricht. Einträge, die nicht mehr in der Datei stehen werden gelöscht, neue Einträge erstellt und veränderte Einträge modifiziert. Die Änderungen werden mit den in der Konfigurationsdatei angegebenen Authorisationsdaten vorgenommen.

Im Change Modus erstellt eine Instanz drei Ausgabedateien. Der Verzeichnisdienst wird ausgelesen und alle Veränderungen in Bezug auf eine Quelle ausgegeben. Für das Erzeugen, Verändern und Löschen von Objekten wird jeweils eine Datei nach ihrem eigenen RDF angelegt. Als Quellen für den Change Modus kommen drei Möglichkeiten in Frage. Als erstes eine Datei mit Zwischenergebnissen aus einem vorangegangenen LDSU Change, einem sogenannten Metafile. Die zweite Quelle ist die Ausgabedatei eines LDSU Exports mit zugehörigem RDF. Bei der dritten Version werden die veränderten Objekte anhand eines bestimmten Attributes selektiert. Dafür kommen Zeitstempel oder USN in Frage. Zum Beispiel können von einem OpenLDAP Server die Objekte mit dem Attributwert vom Attribut "modifyTimeStamp" größer als der Zeitpunkt des letzten Change ausgewählt werden, um alle geänderten Objekte zu identifizieren.

Mit einer Instanz im Transaction Modus werden die drei Ausgabedateien des Changemodus eingelesen. Die Werte werden mit anhand eines RDF manipuliert und dann die entsprechende LDAP Operationen Add, Modify und Delete ausgeführt. Erzeugen und Löschen wird vorgenommen, wie in den entsprechenden Dateien beschrieben. Vor dem Ausführen einer Änderungsoperation wird das betroffene Objekt des Verzeichnisses ausgelesen. Es überprüft, ob die Änderungen nötig sind, und erst dann werden sie ausgeführt.

Der Update Modus dient dazu, ein Verzeichnis nur mittels eines RDF zu verändern. Es werden keine Ausgabedateien erzeugt. Die LDSU Instanz liest das LDAP Verzeichnis aus, manipuliert die Werte der Objekte und überträgt die ermittelten Veränderungen an das gleiche Verzeichnis. In diesem Modus kann eine grosse Anzahl von Objekten im Verzeichnis der gleichen Veränderung unterworfen werden.

Im letzten, dem Reformat Modus, werden nur Dateien verändert und kein

Verzeichnis dazu kontaktiert. Mit ihm kann die RDF Funktionalität genutzt werden, die Daten einer Eingabedatei zu konvertieren und auszugeben.

Um zwei oder mehr Verzeichnisdienste mittels LDSU permanent zu synchronisieren, muß LDSU periodisch ausgeführt werden. Es muß sich nicht auf demselben Rechner befinden, wie einer der LDAP Server. Wenn die Instanzen auf verschiedenen Computern installiert werden, dann ist der nötige Aufwand zum Austausch der Ausgabedateien sehr groß. Die verteilte Lösung ist daher nicht praktikabel.

4.2.2 Konfliktlösung

Die Funktionalität der Record Description Files (RDF) erlaubt eine eingeschränkte Konfliktlösung auf Objektebene. Es bietet die Möglichkeiten einer vereinfachten Scriptsprache, wie zum Beispiel Vergleiche und Operationen auf Zahlen und Zeichenkette. Im besonderen kann auch eine LDAP Suchanfrage ausgelöst werden. Die Konfliktlösung sieht dann wie folgt aus. Im RDF wird der DN des eigentlichen Zieleintrags ermittelt. Der aktuelle Zeitstempels der letzten Änderung des Eintrags wird mittels LDAP Search ausgelesen und mit dem zu setzenden verglichen. Ist der aktuelle Wert kleiner, wird die Änderung angenommen und ansonsten verworfen. Diese Möglichkeit besteht nur dann, wenn beide LDAP Server die gleiche Syntax für Zeitstempel verwenden und ihre Systemuhren synchron laufen. Die Lösung ist eingeschränkt gültig, weil sie keine Änderungen in der kurzen Zeit zwischen dem Abfragen des aktuellen Zeitstempels und dem Senden des Updates berücksichtigen kann.

LDSU selbst bietet als Option für Updates an, den Elternknoten von einem zu erzeugenden Objekt zu kreieren, wenn er nicht vorhanden ist. Als zweite Möglichkeit kann man den Fehlschlag der Operation auch in einem Log festhalten. Die Konflikte müssen dann vom Administrator gelöst werden.

Alle Konflikte einer Multi-Master Replikation kann die einfache Syntax von RDF nicht lösen. LDSU unterstützt keine Objektzuordnung per UID. Als Folge erkennt LDSU eine LDAP ModDN Operation nicht als Umbenennen. Statt dessen als Löschen des Objekts unter dem alten Namen und Erzeugen eines neuen Objektes unter dem neuen Namen. Eine weitere Folge sind falsche Operationen, wenn Updates für Objekt gesendet werden, die auf dem Zielserver seit dem letzten Update verschoben oder gelöscht wurden. Es gehen durch dieses Verhalten eventuell Updates verloren oder es werden fälschlicherweise neue Objekte erzeugt.

4.2.3 Paßwortsynchronisation

Die Paßwortsynchronisation ist für LDSU nicht möglich. Entschlüsselung von lesbaren Paßwortcodierungen sind aus Sicherheits- und Zeitgründen nicht akzeptabel. Nicht lesbare Paßwörter befinden sich vollständig ausserhalb der Reichweite von LDSU.

4.3 MSDSS

Microsoft Directory Synchronization Services ist ein Bestandteil der *"Microsoft Services for Netware"* (SFN). Das ist eine Sammlung von Programmen, die die Integration von Windows 2000 in ein NetWare Netzwerk erleichtern sollen. Es synchronisiert die Datenbestände zwischen den verschieden Servern und emuliert NetWare Dienste, wie Datei- und Druckserver. SFN ist nur für Windows 2000 Server verfügbar. Es liegt zur Zeit in der Version 5.1 vor.

4.3.1 Strategie

MSDSS läuft als Dienst auf einem Windows 2000 Server. Es kann sowohl für die einmalige Migration zwischen den Verzeichnisdiensten, als auch für die permanente Synchronisation genutzt werden. Bei beiden Strategien werden die Objekte, die Zugriffsrechte und die Verzeichnisstruktur migriert. Die Objektmigration beinhaltet das Mappen von Attributen und der Position im Verzeichnis.

Die permanente Synchronisation kann auf zwei Weisen erfolgen. Die *Einwegsynchronisation* überträgt einmalig den Verzeichnisinhalt eines NDS Servers oder Containers in einen Container in MSAD. Danach werden in regelmäßigen Abständen Änderungen im Datensatz des MSAD zu NDS übertragen. In der Gegenrichtung erfolgt kein Abgleich. Es wird davon ausgegangen, daß keine Änderungen mehr in NDS vorgenommen werden. Im Gegensatz dazu werden bei der *Zweiwegesynchronisation* auch spätere Änderungen regelmäßig von NDS zu MSAD übernommen. Die Vorteile der Einwegsynchronisation sind die vereinfachte Administration und die verringerte Netzwerklast. Die Zweiwegesynchronisation verlangt außerdem eine Schemaerweiterung des NDS.

MSDSS stellt eine Standardtabelle für das Mapping zur Verfügung. Sie ordnet Novell und Windows Objekte einander zu. Obwohl das Attributmapping nicht veränderbar ist [25], kann das Objektmapping angepaßt werden. Dazu wird ein Paar von Objekten gebildet, von denen jeweils eines aus jedem Verzeichnisdienst stammt. Die Objekte dieses Paares werden untereinander

synchronisiert, unabhängig von ihrer Position in der Verzeichnisstruktur.

Mit dem Prinzip von Synchronisationsfiltern kann eine Menge von Objekten mit bestimmten Eigenschaften eingegrenzt werden. Bei Aktivierung der Filter werden nur diese Objekte synchronisiert. So können zum Beispiel explizit Administratoreinträge von der Synchronisation ausgeschlossen werden.

4.3.2 Konfliktlösung

Die Art der Konfliktlösung hängt von der Migrationsrichtung ab. Beim Übertragen eines Objektes von NDS zu MSAD werden Konflikte auf Objektebene gelöst. In der entgegengesetzten Richtung wird die Auflösung auf Attributebene vorgenommen und verursacht dadurch weniger Netzlast.

Falls die Synchronisation eines Objektes fehlschlägt, wird es in eine Liste eingefügt. In Abhängigkeit von der Konfiguration werden höchsten eine bestimmte Anzahl von Synchronisationsversuchen unternommen. Gelingen sie nicht, wird die Synchronisation für dieses Objekt aufgegeben. Ein Administrator kann die Liste einsehen und gegebenen Falls Korrekturen im Verzeichnis vornehmen.

4.3.3 Paßwortsynchronisation

Paßwörter können mittels MSDSS von MS Active Directory zu Novell Directory Services übertragen werden. Wird das Paßwort eines Benutzers in MSAD neu gesetzt, wird es von MSDSS auch in NDS geändert. Für die Gegenrichtung gibt es keine Synchronisation. Paßwortänderungen dürfen nur noch unter MSAD erfolgen, wenn Synchronität garantiert werden soll.

Sobald in NDS ein neuer Benutzer erstellt wird, erzeugt MSDSS auch in MSAD einen neuen Benutzereintrag. Er erhält alle nötigen Attribute außer dem initialen Paßwort. MSDSS stellt dem Administrator vier Möglichkeiten zur Auswahl, welche Paßwörter die neuen Benutzer erhalten sollen. Sie werden in Tabelle 4.2 auf Seite 39 erläutert.

4.4 Novell DirXML

DirXML gibt NDS eDirectory die Möglichkeit zum Datenaustausch mit anderen Anwendungen. Das XML Format ("*eXtensible Markup Language*") spielt in dem Dienst eine besondere Rolle, weil es sowohl zum Datenaustausch als auch zur Konfiguration benutzt wird. DirXML setzt auf eDirectory auf und ist für NetWare, Windows 2000/NT, Solaris und Linux verfügbar.

Paßwort	Beschreibung
kein Paßwort	Das initiale Benutzerpaßwort unter MSAD ist leer. Dieses Paßwort ist unsicher.
Benutzername	Das initiale Paßwort ist gleich dem Benutzernamen. Dieses Paßwort ist unsicher.
fester Wert	Das initiale Paßwort für neue Benutzer ist immer gleich. Der Wert wird vom Administrator bestimmt. Dieses Paßwort ist unsicher.
zufälliger Wert	Das initiale Paßwort wird zufällig erzeugt und in einem Log festgehalten. Es kann dem Benutzer einzeln mitgeteilt werden. Diese Methode ist sicherer als die anderen drei, erfordert aber auch mehr Aufwand.

Tabelle 4.2: Initiale Paßwörter bei der Migration mittels MSDSS

Hier wird nur der spezielle Fall zur Synchronisation der Novell eDirectory Datenbank mit einem anderen Verzeichnisdienst betrachtet.

4.4.1 Strategie

Das DirXML Programm unterteilt sich in eine "*DirXML Engine*" und einen "*DirXML Driver*". Die Engine muß auf einem Read/Write Replica des NDS installiert werden. Sie benötigt administrativen Zugriff auf den zu replizierenden Teilbaum des Verzeichnisses.

Nach der Aktivierung sendet eDirectory automatisch jede Änderung seines Datensatzes an die DirXML Engine. Sie wird ins XML Format gebracht und an die Engine übergeben. Die Engine filtert die Daten und führt mit den konfigurierten Regeln ein Mapping durch. Der Vorteil der XML Darstellung ist der offene verständliche Standard. Die Regeln der Engine können leicht vom Administrator angepaßt werden. Die transformierten Daten werden an den "*Subscriber Channel*" des DirXML Driver übergeben. Er ist für die Interaktion mit der Zielapplikation zuständig. Der Driver bringt die XML Daten in das Zielformat, in diesem Fall LDAP Requests, und übergibt die Daten an die Applikation. Auf diese Weise werden Änderungen von eDirectory an andere Verzeichnisse gesendet.

In der Gegenrichtung steht der Driver am Anfang. Er überwacht die Zielapplikation und sammelt die Änderungen. Das Verfahren ist abhängig von

der Applikation. So wird Microsoft Active Directory zum Beispiel in regelmäßigen Abständen abgefragt. Andere Verzeichnisse, wie iPlanet Directory Server oder Netscape Directory Server, verfügen über einen "*Changelog*" Mechanismus, der den Driver selbst benachrichtigen kann. Der Driver kann lokal auf dem gleichen Rechner wie das zu überwachende Verzeichnis installiert werden. Bei einigen Directories, wie zum Beispiel MSAD, muß er es zwingend. Er wird dann von DirXML remote konfiguriert. Wenn der Driver die Daten empfangen hat, so wandelt er sie ins XML Format um und sendet sie an die Engine. Diese wendet die Filter und Mappingregeln für die Gegenrichtung an. Das Resultat wird an eDirectory übergeben.

Die Änderungen, die DirXML selbst an eDirectory oder zu synchronisierenden Verzeichnis durchführt, werden als eigene Updates erkannt und nicht in die Gegenrichtung gesendet. Daten dieser Art werden sofort herausgefiltert und ignoriert. Die Erkennung erfolgt zum Beispiel durch das *modifiersName* Attribut. Ist der Attributwert derselbe, mit dem sich die Engine bzw. der Driver am Verzeichnis authentifiziert, dann wird die Änderung nicht weitergegeben.

Konnte DirXML einen Eintrag A in eDirectory erfolgreich einem Eintrag B in der Zielapplikation zuordnen, werden sie auch in Zukunft miteinander assoziiert. DirXML vermerkt zu diesem Zweck die UID von B in einem Attribut von A. Während der Installation von DirXML wird die Klasse top im Schema von eDirectory um ein Attribut "association" erweitert. Es ist mehrwertig und in ihm werden die UIDs der mit dem Objekt assoziierten Einträge gespeichert [27].

Eine Engine kann mehrere Driver verwalten. Jeder aktivierte Driver synchronisiert eDirectory mit einer Applikation.

4.4.2 Konfliktlösung

Die mit DirXML gelieferten Treiber geben bei Konflikten soweit wie möglich der aktuellsten Version den Vorrang. Die Konfliktlösung bei der Synchronisation von eDirecotry mit Microsoft Active Directory findet auf Attributebene statt[28].

4.4.3 Paßwortsynchronisation

Mit "*DirXML Password Synchronization for Windows*" [29] stellt Novell im DirXML Packet eine Möglichkeit vor, Paßwörter zwischen einer Windows Domäne und einem Novell Netzwerk zu synchronisieren. Sowohl im Novell Client als auch im Windows Client existieren Paßwortfilter. Ändert der Benutzer sein Paßwort, dann durchläuft es den jeweiligen Filter. Der Filter

wird von Novell dazu genutzt, das noch unverschlüsselte Paßwort zu entnehmen und über einen "*Password Synchronization Agent*" an den DirXML Driver zu übermitteln.

Paßwortänderungen können so erfolgreich zwischen beiden Verzeichnisdiensten synchronisiert werden. Die Paßwörter schon bestehender Benutzer werden nicht synchronisiert. Nach der Installation der Paßwortsynchronisation müssen existierende Benutzer ihre Paßwörter ändern.

4.5 Bewertung und Vergleich

Im Folgenden werden die drei vorgestellten Lösungen zur Synchronisation von heterogenen Verzeichnissen bewertet und verglichen. Die Bewertung erfolgt in Hinsicht auf Administrierbarkeit, Flexibilität und die Einsatzmöglichkeiten in der Benutzerverwaltung.

Compaq LDSU wird mit vorgefertigten Konfigurationen für den Datenaustausch mit vielen gängigen Verzeichnisdiensten geliefert. Keine davon enthält eine Konfliktlösung. Es setzt keinen bestimmten Verzeichnisdienst voraus und hat durch RDF einen hohes Potential an Transformationen. Eine detailierte Hilfe macht es möglich, sich relativ schnell in die RDF Syntax einzuarbeiten. Das proprietäre Format von RDF und die vielen kleinen Konfigurationsdateien erhöhen aber den administrativen Aufwand. Zudem können die Konfiguration nur mit einem Editor geändert werden, was häufig einen lokalen Zugriff notwendig macht. Die fehlende Möglichkeit zur Paßwortsynchronisation läßt LDSU für die Synchronisation von benutzerverwaltenden Verzeichnissen nicht zu.

Microsoft Directory Synchronization Service kann nur Microsoft Active Directory und Verzeichnisdienste von Novell miteinander synchronisieren. Die Regeln für Datenaustausch und -transformation sind sehr starr. Es kann nur an Hand von Filtern entschieden werden, welche Objekte synchronisiert werden sollen. Wie die Transformationen auszuführen sind, kann nicht angepaßt werden. MSDSS ist sehr einfach auch remote zu administrieren. Für die Synchronisation von Benutzerdaten zwischen Novell Verzeichnissen und Microsoft Active Directory ist MSDSS gut geeignet.

Novell DirXML enthält in der Grundkonfiguration eine gute Auswahl an Treibern für die Kommunikation von Novell eDirectory mit anderen Verzeichnisdiensten. Die Konfiguration setzt vollständig auf XML und die Regeln zur Transformation der Daten lassen sich leicht anpassen. Sämtliche Administration kann auch remote durchgeführt werden. Mit dem optionalen DirXML Password Synchronization for Windows lassen sich zudem noch

die Paßwörter zweier wichtiger Verzeichnisdienste synchronisieren. DirXML ist sehr gut für die Synchronisation von NDS und MSAD geeignet.

Von den drei Lösungen ist keine optimal. Leicht und remote zu administrieren sind MSDSS und DirXML. Allerdings setzen sie im Gegensatz zu LDSU jeweils die Installation eines bestimmten Verzeichnisses voraus. Sehr flexibel konfigurierbar sind LDSU und DirXML. Paßwörter können nur DirXML und MSDSS synchronisieren, aber ausschließlich zwischen eDirectory und MSAD.

Kapitel 5

Ansätze

Im vorigen Kapitel wurden vorhandene Implementationen zur Synchronisation heterogener Schemas erläutert. In diesem Kapitel werden ihre Methoden verallgemeinert. Die Nachteile der Vorgehensweisen werden betrachtet und es wird eine gute Lösung zur Implementation ausgewählt.

5.1 Zentrale Synchronisation

Für dieses Modell in der Topologie auf genau einem Rechner ein Synchronisationsdienst eingerichtet. Er hat die Aufgabe Zweige jeweils verschiedener Server synchron zu halten. Er überwacht die Daten in jedem Server allein und in den meisten Fällen nicht lokal. Er besitzt für jedes Paar von Servertypen eine Mappingtabelle.

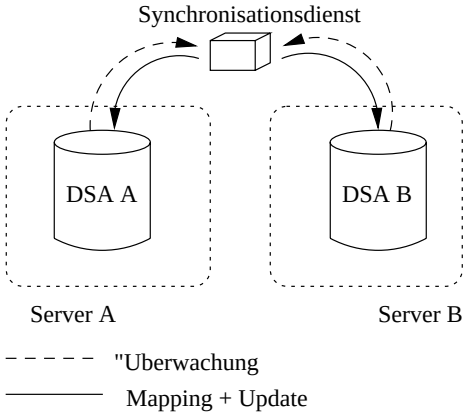

Abbildung 5.1: Zentrale Synchronisation

Ermittelt der Dienst eine Änderung im Datensatz eines Servers, so lädt er die neuen Daten und formt entsprechend seiner Mappingtabellen für jedes andere Verzeichnis ein Update. Abbildung 5.1 zeigt wie ein Synchronisati-

onsdienst zwei Server überwacht und die Änderungen zwischen ihnen austauscht.

Wenn der Dienst ein Update vornimmt, so darf er die daraus resultierende Änderung des Datensatzes nicht als zu synchronisierendes Update erkennen. Sie müssen ignoriert werden. Falls er Synchronisationsupdates nicht von anderen unterscheiden kann, wird der Dienst ein unnötiges Update an alle Verzeichnisse versenden. Eine Endlosschleife kann nicht enststehen. Der Versuch dasselbe Update ein zweites Mal auf einem Verzeichnis auszuführen ist erfolglos und hat keine Änderung des Datensatzes zur Folge.

Die Methoden von Compaq LDSU und MSDSS entsprechen diesem Modell.

5.2 Zentrales Mapping

Novell DirXML läßt sich nicht durch die Zentrale Synchronisation darstellen. Die lokale Einrichtung von Treibern für jeden Verzeichnisdienst legt ein Modell mit lokalen Agenten, aber zentralem Mapping nahe. Die Agenten überwachen lokal nur ein Verzeichnis und registrieren die Datensatzänderungen. Sie verfügen aber nicht über Mappingtabellen, sondern senden die Änderungen an einen zentralen Server. Dort werden die Updates gemappt und dann an alle anderen Agenten verteilt, die wiederum für das Einspielen zuständig sind. Die Prozedur wird in Abbildung 5.2 auf Seite 44 gezeigt.

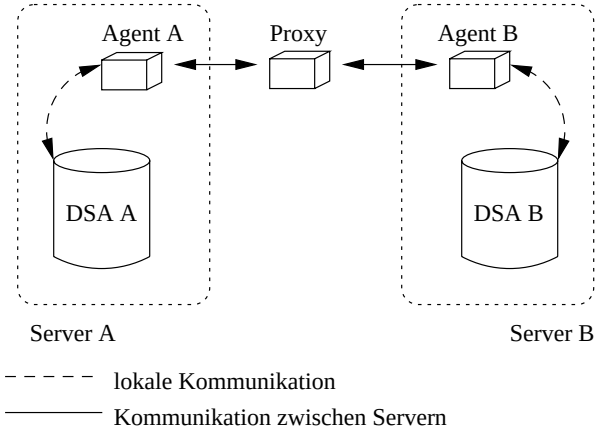

Abbildung 5.2: Zentrales Mapping

Im Spezialfall von DirXML ist der zentrale Dienst gleichzeitig der Agent, der eDirectory überwacht. Die Agenten müssen bei diesem Modell keine Kenntnis der Topologie besitzen. Sie benöten nur den Zugang zum zentralen Server.

5.3 Verteilte Synchronisation

Der nächste logische Entwicklungsschritt ist das Wegfallen des zentralen Servers. Die Mappingfunktion wird von den lokalen Agenten ausgeführt. Wie in Abbildung 5.3 auf Seite 45 wird auf jedem Server lokal ein Synchronisationsdienst eingerichtet. Er entdeckt Änderungen im Datensatz, formt sie um und sendet sie an alle anderen Agenten.

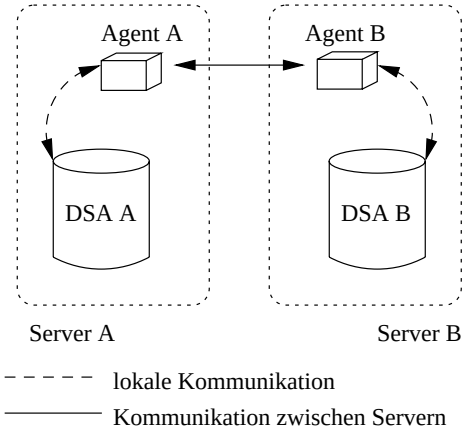

Abbildung 5.3: Verteilte Synchronisation

Die empfangenden Agenten tragen das Update selbst in ihrem Verzeichnis ein. So können sie die resultierende Änderung des Datensatzes als eigenes Synchronisationsupdate erkennen. In diesem Modell muß jeder Agent die Topologie des Netzes kennen.

5.4 Bewertung

Die Implementierung einer Lösung für Multi-Master-Konflikte ist bei allen drei Ansätzen realisierbar. Bevor ein Agent bzw. Synchronisationsdienst ein Update an einen Verzeichnisdienst weitergibt, liest er den Zieleintrag des Updates aus. Wenn er aktueller als das Update ist, verfällt es. Genauere Prozeduren zur Konfliktlösung werden im nächsten Kapitel erläutert.

Die Zentrale Synchronisation hat den Vorteil, daß sie zentral verwaltet werden kann. Es existiert nur ein Synchronisationsdienst, dessen Mappingtabellen ausgearbeitet und gewartet werden müssen. Neue Schemas in die bestehende Topologie zu integrieren sehr leicht, da nur ein Dienst aktualisiert werden muß, ist der Aufwand konstant. Der einzelne zentrale Synchronisationsdienst resultiert aber auch in einem *"single point of failure"* Je mehr verschiedene Server in der Synchronisationstopologie enthalten sind, desto mehr Mapping- und Updatefunktionen müssen je Änderung des Datensatzes

durchgeführt werden. Der Aufwand an Zeit und Ressourcen zum Versenden eines Updates ist linear abhängig von der Anzahl der Server. Wenn die Zentrale Synchronisation mit einer Konfliktlösung implementiert ist, steigt der Aufwand pro Update. Die zusätzliche Anfrage vor dem Übertragen des Updates verdoppelt etwa den Aufwand an Zeit und Netzwerkressourcen. Der Ansatz bleibt trotzdem linear von der Anzahl der Server abhängig.

Das Zentrale Mapping kann Konfliktlösung schon lokal vornehmen und damit Netzressourcen schonen. Updates werden lokal das erste Mal gefiltert. Durch den Proxy werden die Mappingdaten zentral gehalten. Das hält des Aufwand bei der Erweiterung um ein neues Schema auf dem Niveau der Zentralen Synchronisation. Als Nachteil existiert mit dem Proxy immer noch ein *"single point of failure"*. Der Aufwand an Zeit und Ressourcen zum Übermitteln eines Updates verhält sich linear in Bezug auf die Anzahl der Server. Das gilt auch für eine Implementierung mit Konfliktlösung.

Die Verteilte Synchronisation mit Hilfe von Agenten versendet keine unnötigen Updates. Sie werden schon lokal erkannt und verworfen. Die Verteilung entfernt den *"single point of failure"*, ein Agent kann ausfallen, ohne daß die Synchronisation der anderen gestört wird. Der Zeit- und Ressourcenaufwand zum Senden eines Updates ist ebenfalls linear von der Anzahl der Server anhängig. Die Erweiterung der Topologie um einen Server mit einem neuen Schema ist im Vergleich zur zentralen Synchronisation bzw. dem zentralen Mapping aufwendig. Jeder einzelne Agent muß eine zusätzliche Mappingtabelle erhalten. Der Aufwand zur Erweiterung steigt also mit jedem zusätzlichen Schema. Der Administrationsaufwand sinkt, wenn die Agenten ihre Synchronisationsfähigkeit für das Verteilen der Mappingtabellen nutzen.

In der Praxis sollte auf den Einsatz einer zentralen Synchronisation verzichtet werden, wenn die Möglichkeit des zentralen Mappings besteht. Es ist genauso einfach zu verwalten, verbraucht aber weniger Ressourcen. Die verteilte Synchronisation kann ihre vergleichsweise hohe Robustheit vor allem bei instabilen Netzwerken ausspielen. Bei einer grossen Anzahl von zu synchronisierenden Verzeichnisdiensten ist sie im Vorteil, da nicht jedes Update über einen zentralen Rechner laufen muß. Sind beide Bedingungen nicht gegeben, sollte dem zentralen Mapping der Vorrang gegeben werden, weil es einfacher zu administrieren ist.

Kapitel 6

Implementation

Für die Synchronisation verschiedener LDAP Implementationen wird von den Herstellern weit weniger Aufwand betrieben als für die Replikation zwischen gleichartigen LDAP Servern. Das Augenmerk der Entwickler liegt natürlich darauf, möglichst viele der eigenen Server zu verkaufen.

Im Rahmen dieser Arbeit wurde ein Programm auf Basis des Ansatz der verteilten Synchronisation aus Kapitel 5 geschrieben. Er wurde ausgewählt, weil er in Hinsicht auf die in Kapitel 4 vorgestellten Methoden eine neue Lösung darstellt. Ziel der Implementation ist, die Tauglichkeit des Konzepts zu ermitteln. Das Programm synchronisiert Benutzeraccounts zwischen Novell eDirectory und Microsoft Active Directory. Die beiden Verzeichnisse wurden wegen ihrer Marktdominanz als Repräsentanten ausgewählt.

6.1 Anforderungen

Die Synchronisationssoftware soll folgenden primären Anforderungen erfüllen.

1. Änderungen in der Datenbank des Verzeichnisses erkennen
2. Änderungen entsprechend dem Zielserver mappen
3. Konflikte lösen und Änderungen eintragen

Weitere wünschenswerte Eigenschaften des Programms sind:

1. Die Synchronisationstopologie ist robust gegenüber Teilausfällen.
2. Das Verfahren kann auf jeden Typ LDAP Server erweitert werden.
3. Paßwortänderungen und -synchronisation sind auf jedem Server möglich.
4. Es sind keine Änderungen am Verzeichnisdienst oder dessen Schema nötig.

6.2 Konzept

6.2.1 Änderungen erkennen

Die primäre Forderung nach dem Erkennen der Änderungen im Verzeichnis läßt sich prinzipiell über eines von zwei Verfahren erfüllen[20]:

Ein **Log-basierendes** Verfahren liefert die Änderungen, die seit der letzten Synchronisation am Verzeichnis vorgenommen wurden.

Ein **Status-basierendes** Verfahren liefert den aktuellen Zustand der Einträge, die seit der letzten Synchronisation verändert wurden.

Die Status-basierende Methode hat den Vorteil, daß sie nicht jede Operation zwischen den Replikationszyklen übermittelt. Wiederholte Updates eines Attributes werden zu einem Status zusammengefaßt. Der Agent muß sich an das Verfahren des überwachten Verzeichnisses anpassen und es so gut wie möglich ausnutzen. Er erhält soweit es nötig ist für jeden Verzeichnisdienst spezielle Zugriffsroutinen.

6.2.2 Mapping

Die Tabellen 6.1 auf Seite 48 und 6.2 auf Seite 49 zeigen das Mapping eines typischen Benutzerobjekts vom Active Directory Schema in das eDirectory Schema. Zur Vereinfachung werden nicht alle möglichen Attribute der Objekte aufgelistet.

MS Active Directory
dn: CN=Babs Jensen,CN=Users,DC=net1,DC=local
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: Babs Jensen
displayName: Barbara J Jensen
telephoneNumber: +1 313 747-4454
title: Mythical Manager, Research Systems
homeDirectory: \\SAMBA\jensen
userAccountControl: 660048

Tabelle 6.1: Mapping eines typischen Benutzerobjektes: Startwerte

Im Beispiel werden einige Möglichkeiten gezeigt, Attribute beim Mapping zu verändern.

1. Der Wert des Attributes *cn* wird dem Attribut *fullName* zugewiesen.

NDS eDirectory
dn: cn=Barbara J Jensen,ou=Users,dc=net2,dc=local
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: top
cn: Barabara J Jensen
fullName: Babs Jensen
title: Mythical Manager, Research Systems
homeDirectory: /usr/people/jensen
telephoneNumber: +1 313 747-4454

Tabelle 6.2: Mapping eines typischen Benutzerobjektes: Zielwerte

2. Der Wert des Attributes *displayName* wird dem Attribut *cn* zugewiesen.

3. Die Werte der Attributes *title* und *telephoneNumber* werden gleichnamigen Attributen zugewiesen.

4. Das Attribut *userAccountControl* wird weggelassen. Es gibt keine Entsprechung dafür.

5. Die Objektklasse *user* wird der Objektklasse *inetOrgPerson* zugewiesen. Die restlichen Objektklassen ergeben sich automatisch, weil sie jeweils Oberklassen sind.

6. Der DN des Eintrags entsteht aus dem Attribut *cn* und dem DN des Vorgängerknotens *ou=Users,dc=net2,dc=local*.

7. Der Wert des Attributes *homeDirectory* wird einer Transformation unterzogen.

Die Mappingfunktion weist einer Menge von Attributnamen aus dem Ausgangsschema jeweils einen Attributnamen aus dem Zielschema zu. Sie legt ausserdem für eine Menge von Containern aus dem Ausgangs DIT jeweils einen Container aus dem Ziel DIT fest. Ein Eintrag wird dann wie folgt der Mappingfunktion unterzogen:

- Attribute, die ohne Bedingung im Zieleintrag enthalten sein müssen, werden hinzugefügt. (Beispiel: userAccountControl bei Mapping von NDS zu MSAD)

- Attribute, die aus dem Originaleintrag hervorgehen, werden mit Hilfe des zugeordneten Attributes erzeugt. Unter Umständen erfolgt eine Transformation des Ursprungswertes. (Beispiel: homeDirectory)

- Ist einem Attribut kein Zielattribut zugeordnet, dann entfällt es. (Beispiel: userAccountControl bei Mapping von MSAD zu NDS)
- Der DN des Eintrags wird aus dem namensgebenden Attribut und dem eventuell umgeformten Namen des Vorgängerknotens gebildet.

Die Mappingfunktion muß während der Laufzeit in folgendem Sinne erweiterbar sein. Wenn beide zu synchronisierenden Verzeichnisse UIDs zur Identifizierung von Einträgen benutzen, soll auch die Mappingfunktion das tun. Hat sie einmal durch eine Operation zwei Einträge einander zugeordnet, dann werden deren UIDs miteinander assoziiert bis beide gelöscht werden. Beim Mappen von darauffolgenden Updates werden die bestehenden Assoziationen beachtet. Damit ist es möglich die Objekte einer Assoziation im DIT zu verschieben oder umzubennen und die Einträge trotzdem einander zuzuordnen.

6.2.3 Konfliktlösung

In Microsoft Active Direcotory und Novell eDirectory ist eine Konfliktlösung mindestens auf Attributebene implementiert. Eine solche Lösung kann nicht erstellt werden, da LDAP keine Abfrage der CSN für Attribute definiert. Mit LDAP lassen sich statt dessen die in RFC 2252 [38] beschriebenen Attribute *"modifiersTimeStamp"* und *"createTimeStamp"* bzw. die für MSAD äquivalenten Attribute *"whenChanged"* und *"whenCreated"* abrufen und als Objekt CSN verwenden. Sie enthalten den Zeitpunkt der letzten Änderung bzw. der Erstellung des Eintrages. Ein sinnvoller Einsatz setzt synchrone Uhren der Server voraus. Es ist zu beachten, daß diese Attribute operational sind und nur vom Server selbst gesetzt werden können.

Im Folgenden werden Konfliktlösungen auf Objektebene für die LDAP Operationen vorgestellt. Sie sind auf die Rolle des Synchronisationstools als LDAP Client ohne Schreibrechte auf operationale Attribute angepaßt. Wenn eine Operation nicht durchgeführt werden kann, dann wird sie immer in einem Log vermerkt. Der Administrator kann daraufhin noch Reparaturen vornehmen.

- MSAD erlaubt das Abfragen von Zeitstempeln auf Attributebene. Der Zugriff muß über ADSI erfolgen. Da das nicht die Eigenschaften aller Verzeichnisse repräsentiert, muß auf diese Möglichkeit verzichtet werden. Dasselbe gilt für NDS, das das Abfragen per NDAP erlaubt.
- Der mangelnde Zugriff auf operationale Attribute ist ein Nachteil der Implementation. Synchronisierte Einträge erhalten nicht die Zeitstempel der Originaloperation, sondern die der Synchronisationsoperation.

Für eine sukzessive Synchronisation wie in einer Ringtopologie notwendig, ist die Implementation daher nicht geeignet. Bei OpenLDAP können unter Umständen operationale Attribute per LDAP gesetzt werden, wie der experimentelle Entwicklungszweig des Projektes mit seiner Multi-Master Replikation zeigt. Es ist aber keine allgemein gültige Lösung und wird daher nicht beachtet.

Hinzufügen eines Eintrags

Es soll mit einem Update U ein Eintrag E zum Verzeichnis hinzugefügt werden. Existiert bisher kein Eintrag A mit dem selben UID, wird E erzeugt. Falls der Vorgängerknoten von E nicht existiert wird U im Log vermerkt. Exisitiert schon ein Eintrag A, wird U genau dann ignoriert, wenn der *createTimeStamp* von U kleiner oder gleich dem von A ist. Ist er grösser, dann werden die Attribute von A durch die von E erstetzt. Sollte vor dem Hinzufügen mindestens ein Eintrag mit dem DN von E existieren, werden die RDN durch Anhängen des UID einzigartig gemacht. Falls das Hinzufügen wegen anderer Gründe, zum Beispiel mangelnder obligatorischer Attribute fehlschlägt, wird U im Log vermerkt.

Löschen eines Eintrags

Es soll mit einem Update U ein Eintrag E aus dem Verzeichnis gelöscht werden. Ist kein Eintrag mit dem UID von E vorhanden, wird das Update ignoriert. Ist der *createTimeStamp* von U kleiner oder gleich dem von E, dann wird U ignoriert. Im anderen Fall wird die Anzahl der Nachfolger von E geprüft. Hat E keine Nachfolger, dann wird er gelöscht. Anderenfalls wird U ins Log eingetragen.

Verschieben eines Eintrags

Es soll mit einem Update U ein Eintrag E im Verzeichnis verschoben werden. Existiert kein Eintrag mit demselben UID wie E, dann wird U im Log vermerkt. Anderenfalls wird geprüft, ob der *modifyTimeStamp* von U grösser ist als der von E. Ist es nicht so, wird U ignoriert. Existiert der Vorgängerknoten des Ziels nicht oder ist er selbt ein Nachfolger von E, dann wird U ins Log eingetragen und ignoriert. Ansonsten wird U durchgeführt. Sollte vor dem Verschieben mindestens ein Eintrag mit dem selben DN existieren, werden die jeweiligen RDN durch Anhängen des UID einzigartig gemacht.

Umbenennen eines Eintrags

Es soll mit einem Update U der RDN eines Eintrags verändert werden. Falls E nicht vorhanden ist, wird U ins Log geschrieben. Ansonsten wird geprüft, ob der *modifyTimeStamp* von U grösser ist, als der von E. Falls dem so

ist, wird der RDN von E verändert. Anderenfalls wird U ignoriert. Sollte vor dem Umbenennen ein Eintrag mit demselben DN existieren, werden die jeweiligen RDN durch Anhängen des UID einzigartig gemacht.

Modifizieren eines Eintrags

Es sollen mit einem Update U Attribute eines Eintrags E verändert werden. Falls E nicht existiert, wird U im Log vermerkt. Anderenfalls wird geprüft, ob der *modifyTimeStamp* von U grösser als der von E. Ist er grösser, dann werden die Attribute von E durch die von U ersetzt. Ansonsten wird U ignoriert.

6.2.4 Paßwortsynchronisation

Die Paßwortsynchronisation ist mit den Ansätzen aus Kapitel 5 nicht ohne weiteres möglich. Beim Auslesen bzw. Entschlüsseln sind ihnen die gleichen Schranken gesetzt wie jedem anderen LDAP Client. Aber man kann wie Novell DirXML die Synchronisationsvereinbarungen des Agenten zur Verbreitung von Paßwortänderungen benutzen. Ein neues Paßwort wird nicht mehr direkt im Verzeichnisdienst eingetragen, sondern mit einer LDAP Operation an den Agenten bzw. den Synchronisationsdienst gesandt. Er muß sie als Paßwortoperation erkennen und zum eigenen Verzeichnisdienst senden. Der Benutzer verwendet als Authorisationdaten seine üblichen Daten. Mit ihnen authorisiert sich der Agent selbst am Server. Ist die Authorisation und das Ändern des Paßwortes erfolgreich, kann das Update an alle anderen Agenten geschickt werden. Anderenfalls wird die Änderung zurückgewiesen und das Update verworfen.

Die meisten Verzeichnisdienste verlangen zum Ändern des Benutzerpaßwortes eine spezielle LDAP Modify Operation. Als Teil dieser Operation wird der alte Wert gelöscht und der neue Wert eingefügt. Der alte Wert muß dazu exakt angegeben werden. Das bloße Ersetzen des Paßworts ist nur Administratoren erlaubt.

Microsoft Active Directory verlangt zum Ändern des Paßwortes eine noch größere Sicherheit. Die LDAP Verbindung muß 128 bit verschlüsselt sein, damit eine Änderung möglich ist. Das neue Paßwort muß von Anführungszeichen umschlossen und in Unicode übergeben werden. Agenten für einen Active Directory Server müssen diese Eigenschaft beachten.

6.3 Umsetzung des Konzeptes

In diesem Abschnitt wird beschrieben, wie das Konzept der verteilten Synchronisation implementiert wurde. Ein Agent muß in der Lage sein LDAP

Anfragen zu stellen und zu empfangen. Er stellt sie, um die Veränderungen auf dem zugehörigen Verzeichnis zu erkennen oder zu erstellen. Er empfängt Anfragen von anderen Agenten, die ihm ein Update mitteilen wollen. Die Implementation OpenLDAP liegt als Quellcode vor. Sie enthält unter anderem das schon in Abschnitt 3.3 auf Seite 25 erwähnte Back-end back-ldap.

slapd ist mit Verwendung des back-ldap Backends ein LDAP Proxy. In seiner Konfiguration wird der Server angegeben, an den slapd die Anfragen seiner Clients weiterleiten soll. Die Proxyimplementation durch back-ldap ist unter bestimmten Umständen fehlerhaft.

slapd führt bei jeder modifizierenden Anfrage eine einfach Überprüfung der übergebenen Parameter durch. Zum Beispiel müssen Attribute und Klassen der Anfrage im Schema vorhanden sein. slapd antwortet bei nicht Bestehen der Überprüfung mit einem Fehler. Wenn der LDAP Server, den der Proxy maskiert, ein anderes Schema als slapd benutzt, werden unter Umständen Anfragen zurückgewiesen, die für den LDAP Server korrekt sind.

slapd führt bei lesenden Anfragen diese Überprüfung ebenfalls durch. Sie wird auf das Ergebnis der Anfrage angewendet, die der Proxy selbst stellte, um die Clientanfrage weiterzuleiten. Als Folge können Attribute in der Antwort des Proxies fehlen, obwohl sie in der Antwort des LDAP Servers noch vorhanden waren.

Die Fehler resultieren aus einer Überprüfung der LDAP Anfragen durch slapd außerhalb des back-ldap Moduls. Es wird versucht intern jedem Attribut eine Beschreibung hinzuzufügen, die unter anderem die Attributsyntax enthält. slapd sucht ausschließlich im eigenen Schema nach dem Attribut und entfernt es aus der Anfrage, falls es nicht gefunden wird. Diese Überprüfung ist für einen LDAP Proxy falsch und muß übersprungen werden. Das gilt nicht unbedingt für andere Backends. Bei back-ldbm ist zum Beispiel die Attributbeschreibung zum Einfügen des Wertes in die Datenbank erforderlich.

In dieser Arbeit basiert die Implementation des Agenten auf dem LDAP Back-end des slapd. Es wird genutzt um die Updates der anderen Agenten zu empfangen und an das Verzeichnis zu übergeben. Die Überwachung des Servers, das Mappen und Versenden von Updates wird in einen zusätzlichen erstellten Thread des Programms durchgeführt. Updates können gleichzeitig empfangen und gesendet werden. Die oben beschriebene Überprüfung der LDAP Anfragen wird in der Implementation übersprungen.

6.3.1 Erkennen der Veränderungen

Veränderungen im Datensatz werden je nach Verzeichnis auf das Log basierend oder auf den Status basierend ermittelt[20]. Die auf das Log basierende Methode wurde für OpenLDAP implementiert indem der Agent die Logdatei des Servers verfolgt. Status basierend wird eDirectory und Active Directory mit den Operationen *"persistent search"* [34] und ihrem Analogon *"server notification"* [22] überwacht. Beides sind asynchrone LDAP Search Operationen mit Control. Bei einer Modifikation im Verzeichnis gibt die Operation als Teilergebnis den Inhalt des neuen oder veränderten Eintrags zurück. Löschungen werden als Veränderungen angegeben, bei denen der Eintrag als gelöscht markiert wird. Das geschieht unter MSAD als Verschiebung in einen speziellen Ordner und unter eDirectory durch das Setzen eines operationalen Attributes.

- Active Directory überprüft LDAP Controls ungenügend auf Werte der Länge null. Das kann zum Absturz des Verzeichnisdienstes führen. [1]

Im Allgemeinen besitzen die Einträge jedes LDAP Verzeichnisses ein Attribut, was den Zeitpunkt der letzten Änderung enthält. Es heißt bei eDirectory und OpenLDAP *"modifyTimeStamp"* und bei Active Directory *"whenChanged"*. Mit diesem können immer die Modifikationen im Verzeichnis Status basierend gefunden werden. Wenn T der Zeitstempel ist, bei dem die letzte Statusabfrage erfolgte, ermittelt eine LDAP Search Anfrage mit dem Filter "(modifyTimeStamp>T)" die seitdem veränderten Einträge. Sie gibt den aktuellen Status der Einträge zurück.

Der implementierte Agent fügt die empfangenen Änderungen in eine Queue ein und bearbeitet deren Inhalt beim Zutreffen bestimmter Bedingungen. Sinnvolle Bedingungen sind der Ablauf einer bestimmten Zeitspanne oder eine bestimmte Anzahl von Anfragen in der Queue. Die Verwendung einer Queue stellt sicher, daß Veränderungen in der Reihenfolge ihres Auftretens an andere Agenten weitergeleitet werden. Das ist sinnvoll, da es Änderungen gibt, die von ihrer Abfolge abhängig sind. Zum Beispiel dürfen Kindknoten erst nach ihren Eltern erzeugt werden.

6.3.2 Mapping

Die gefundenen Einträge müssen entsprechend dem Zielagenten des Updates umgeformt werden. Der Agent liest beim Start aus einer Konfigurationsdatei seine Synchronisationsvereinbarungen. Darin sind alle Zielagenten, die dazugehörigen Zugangsdaten und die jeweiligen Mappingtabellen enthalten. Für

[1] siehe http://www.securitytracker.com/alerts/2002/May/1004369.html

jeden Eintrag aus der Queue wird für jeden Zielagenten ein Update erzeugt. Die verwendete Mappingfunktion ist komplexer als die in Abschnitt 6.2.2 beschriebene direkte Zuordnung von Attributnamen. Im Folgenden wird das Mapping umrissen.

Die Tabelle besteht aus einer Liste von Paaren. Jedes Paar besteht aus einem Mustereintrag und einem Roheintrag. Das Muster gibt DN, Attribute und wahlweise auch Attributwerte vor, die der Eintrag besitzen muß oder darf. Es sind auch Wildcards zulässig. Der Roheintrag enthält DN, Attribute und wahlweise auch Attributwerte des zu erstellenden Updates. Über eine Vergleichsoperation wird der Originaleintrag mit dem Muster verglichen. Wenn er paßt, dann erzeugt der Agent aus dem Roheintrag und den Attributwerten des Originals ein Update. Paßt er nicht, wird der Vorgang mit dem nächsten Paar in der Tabelle wiederholt. Kann kein Tabelleneintrag ein Update erzeugen, dann ist das Mapping fehlgeschlagen. Es wird davon ausgegangen, daß die Änderung nicht übertragen werden soll. Mit anderen Worten: wenn das Mapping fehlschlägt, befindet sich der Eintrag nicht im Synchronisationsgebiet[2].

Die Assoziationen zwischen den Einträgen der verschiedenen Server werden vom Agenten in einer Tabelle gehalten. Deren Entstehung und der Umgang den Zuordnungen wird in den nächsten beiden Abschnitten erklärt.

Das implementierte Mapping erfüllt die Forderungen aus Abschnitt 6.2.2. Es bietet zusätzlich noch die Möglichkeit des Filterns. So kann zum Beispiel durch die Angabe eines DN mit Wildcard die Synchronisation auf einen Teilbaum des Verzeichnisses beschränkt werden. Die Mappingfunktion wurde beim Entwurf weiter ausgelegt als nötig, damit sie noch entstehenden Bedürfnissen gerecht werden kann.

6.3.3 Übertragen der Updates

Die Übertragung der Daten kann relativ einfach erfolgen, zum Beispiel indem das Update mit den LDAP Operationen Add, Delete, Modify oder ModifyDN gesendet wird. Mit dieser Methode können aber keine CSN auf anderer als Objektebene übertragen werden, da Attributzeitstempel im LDAP Standard nicht definiert sind. Um das Konzept erweiterbar zu halten, kommunizieren Agenten mit LDAP Extended Operationen. In den LDUP Entwürfen werden noch keine konkreten Vorschläge zum Übertragen von Updates und der Kommunikation gemacht.

Updates zwischen Agenten werden Status basierend versandt. Der Zielagent

[2]Synchronisationsgebiet hat hier die zu Replication area anaologe Bedeutung

erhält den Eintrag in seinem aktuellen Zustand vom Quellagenten. Der Inhalt einer LDAP Extended Anfrage besteht aus OID und Wert. Als OID wird "_REPLICA_UPDATE_" verwendet. slapd mußte leicht verändert werden um diese selbst definierte OID zu unterstützen. Das Update wird im Wert wie in Abbildung 6.3 BER [13] codiert.

```
UpdateValue ::= SEQUENCE {
                MessageID      INTEGER,
                csnCreated     OCTET STRING,
                csnModified    OCTET STRING,
                UID            OCTET STRING,
                UpdateValue    AddRequest
}
```

Tabelle 6.3: Format eines Replication Update

Die Variable MessageID ist eine Ganzzahl, die die Nachricht identifiziert. Dieselbe Zahl ist in Antworten auf diese Anfrage enthalten. csnCreated und csnModified enthalten die CSN der Erstellung und der letzten Änderung des Eintrages. Weil das Datumsformat zum Beispiel bei MSAD und NDS unterschiedlich ist, werden die Zeitangaben in csnCreated und csnModified für die Agenten einheitlich formatiert. UID ist der Unique Identifier des Eintrags auf dem Quellserver. Der Typ AddRequest ist der eines LDAP Add Request und wird wie in RFC 2251 [37] definiert. UpdateValue ist der neue Inhalt des Eintrages im Format AddRequest. Es enthält den DN, alle Attribute und Werte des Objektes im aktuellen Zustand.

Die Antwort des Zielservers erfolgt in Form einer LDAP Extended Operation Response. Als OID wurde "_REPLICA_UPDATE_RESPONSE_" gewählt. Im Wert der Antwort ist die MessageID und bei erfolgreichem Update der UID des assoziierten Eintrags des Zielservers enthalten. Die UID wird mit der Original UID in die Assoziationstabelle des Agenten übernommen. Der Erfolg des Updates auf dem Zielserver wird durch das Element errorCode mitgeteilt. Die Codierung wird in Abbildung 6.4 gezeigt.

```
response ::= SEQUENCE {
                MessageID      INTEGER,
                errorCode      INTEGER,
                UID            OCTET STRING,
}
```

Tabelle 6.4: Format eines Replication Update Response

6.3.4 Konfliktlösung

Nachdem ein Update den Zielagenten erreicht hat, kann er es noch nicht sofort seinem Verzeichnisdienst übermitteln. Er sucht statt dessen in seiner Assoziationstabelle nach dem UID des Quelleintrages. Falls er ihn findet, wird das Update mit der in Abschnitt 6.2.3 beschriebenen Konfliktlösungsstrategie auf den Eintrag mit dem assoziierten UID angewandt.

Findet er ihn nicht, dann besteht noch keine Assoziation für diesen UID. Mit einer LDAP Search Operation wird im Zielverzeichnis der Eintrag gesucht, der den gleichen DN wie das Update besitzt. Existiert der Eintrag, dann wird eine neue Assoziation gesetzt und das Update unter Beachtung der Konflikte eingetragen. Anderenfalls wird ein neuer Eintrag angelegt. Der entstandene neue UID wird dem UID des Updates zugeordnet.

Bei erfolgreichem Update wird der assoziierte UID in der Antwort an den Quellagenten zurückgesandt. Das ist notwendig, damit alle Agenten die gleichen Assoziationstabellen verwenden.

Kapitel 7

Leistungsfähigkeit

Vor dem Einsatz eines Synchronisationsprogrammes muß sicher sein, daß es den gestellten Anforderungen gerecht wird. Seine Leistungsfähigkeit läßt sich an zwei Stellen messen. Die erste ist das Programm und sein Ressourcenverbrauch selbst. Die zweite ist der Einfluß auf den Verzeichnisdienst. Ein wichtiges Kriterium für ein Verzeichnis ist seine Verfügbarkeit und sie muß so wenig wie möglich von den Synchronisationsprozessen beeinflußt werden.

Das Programm DirectoryMark der Firma MindCraft ermittelt die Leistung eines Verzeichnisses. Es ist in der Version 1.2.1 frei erhältlich[1].

7.1 DirectoryMark

DirectoryMark erstellt einen Satz Daten im Verzeichnis und führt LDAP Operationen darauf aus. Die Antwortzeiten für die Operationen dienen als Maß für die Leistung. Es besteht aus drei Teilen.

Der Datensatz wird mit Hilfe des Perl Scripts *"dbgen.pl"* erzeugt. Er wird in LDIF in eine Datei ausgegeben. Die Einträge sind Benutzerobjekte mit zufällig gewählten Eigenschaften. Die Attribute, wie Name, Telefonnummer oder Paßwort, werden aus einer Datenbank mit realistischen Vorgaben zusammengestellt. Standardmässig sind die Beispieleinträge von der Objektklasse *"organizationalPerson"* wie in RFC 2256 [39] definiert. Die Daten werden in das Verzeichnis über Tools wie *"ldifde"* oder *"ldapadd"* importiert [2].

Aus der LDIF Datei erstellt *"scriptgen.pl"* das Script für das Testszenario. Es besteht aus einer Menge von LDAP Operationen, die alle über dem er-

[1] http://www.mindcraft.com/directorymark/

[2] Paßwörter können unter Active Directory nicht mit *ldifde* importiert werden. Es wird stattdessen *ldapmodify* mit SSL von OpenLDAP verwendet.

stellten Datensatz stattfinden. Über die Parameter von scriptgen.pl läßt sich der Anteil jeder LDAP Operation an der Gesamtzahl der Operationen und die Anzahl der LDAP Clients beim Test steuern. Die Verwendung mehrerer Clients ermöglicht die Simulation von konkurrierenden Zugriffen.

Das Script wird durch das Programm *"DirectoryMark.exe"* durchgeführt und ausgewertet. Das Ergebnis ist eine Sammlung von statistischen Daten. Für jede der LDAP Operation Search, Add, Delete, Modify, Compare und Modrdn werden acht Werte zurückgegeben. Vier davon sind Zeitwerte: minimaler, maximaler und durchschnittlicher Zeitaufwand sowie die Standardabweichung. Die anderen vier Werte geben die absolute Anzahl von Transaktionen, fehlgeschlagenen Transaktionen, Transaktionen mit Zeitüberschreitung sowie die Anzahl der beeinflußten Einträge an.

DirectoryMark stellt zwei Standardszenarien zur Ausführung bereit. Das Messaging-Szenario führt genau einmal am Anfang des Tests eine Bind Operation durch. Danach werden nur noch Search Operationen ausgeführt, indem das Ergebnis genau einem Eintrag entspricht. Die Auswertung ergibt die Zeit pro exakter Suchanfrage. Das Address-Lookup Szenario simuliert eine Menge von Clients, die in einem Addressbuch nach Einträgen suchen. Eine Bind Operation erfolgt alle fünf Operationen und dazwischen verschiedene andere.

Für die Simulation eines Servers zur Benutzerauthentifizierung sind beide Szenarien nicht geeignet. Die typische Vorgehensweise beim Anmelden eines Benutzers ist eine Bind Operation zur Authentifizierung und eine Search Operation zum Laden der Konfigurationsdaten. Das Authorisationsszenario besteht also aus abwechselnden Bind- und Search Operationen.

7.2 Ergänzungen

DirectoryMark hat den Nachteil, daß nur eine Auswertung über die gesamte Testzeit existiert. Die Veränderungen der Antwortzeiten in Bezug auf die Zeit seit Testbeginn werden nicht protokolliert. Es wird auch keine Statistik für die Bind Operation erhoben.

Um diese Nachteile auszugleichen wurde im Rahmen dieser Arbeit ein Benchmark geschrieben, daß die Perfomance in Abhängigkeit zur Zeit speichert. Testdatenbank und -szenario werden weiterhin durch dbgen.pl und scriptgen.pl erzeugt. Der Test läuft über eine bestimmte Zeit. In vordefinierten Abständen werden LDAP Anfragen gestellt, die Antwortzeit gemessen und zusammen mit der bisherigen Laufzeit des Programms protokolliert.

7.3 Synchronisationsszenarien

Für Replikationen und Synchronisationen gibt es zwei Varianten, das Full Update und das Inkremtentelle Update.

Ein Full Updates wird simuliert, indem die durch dbgen.pl erstellte Datenbank in Verzeichnis A geladen wird. Verzeichnis A wird danach mit Verzeichnis B synchronisiert. Als erstes beginnt die Aufzeichnung der Leistungsdaten der Server und des Synchronisationstools. Für die Server werden die Antwortzeiten protokolliert, für das Synchronisationsprogramm der Zeitaufwand für die Teilaufgaben. Es wird zur Messung das Authorisationsszenario verwendet.

Für das Inkrementelle Update werden schon synchronisierte Server verwendet. Auf jeweils einem werden regelmäßig Veränderungen eingespielt, die dann synchronisiert werden sollen. Nach einer ausreichenden repräsentativen Zeitspanne wird der Test beendet und ausgewertet.

Zum Vergleich werden die Antwortzeiten der Server ohne aktive Synchronisation gemessen.

7.4 Messungen

Die Messungen wurden unter Windows 2000 Advanced Server (Service Pack 3, Build 5.00.2195) vorgenommen. Die verwendete Hardware des ersten Computers ist ein AMD Athlon 800 MHz mit 1.0 GByte RAM. Der zweite ist ein Pentium III 750 Mhz mit 512 MByte RAM. Die Systemuhren beider Rechner wurden vor Beginn der Tests synchronisiert.

In jedem Test wurde ein Active Directory Verzeichnis mit einem eDirectory Verzeichnis synchronisiert. Die Beispieldatenbank wird von dbgen.pl erzeugt. Während der gesamten Synchronisationsdauer erzeugt der Benchmark die Umgebung des Authorisationsszenarios. Es führt ohne Pause abwechselnd LDAP Bind und ein LDAP Search durch. Diese Konfiguration erzeugt eine hohe, aber keine maximale Last.

Während des Full Updates wird das Szenario nur auf den Supplier angewendet. Der Consumer verfügt noch über keine Daten, die abgefragt werden können. Beide Server werden während des Inkrementellen Updates getestet.

Nach Abschluß des Szenarios kann mit Hilfe der Ausgabelogs der Agenten der Zeitaufwand für die Teilaufgaben Mapping, Konfliktlösung und Eintragen der Updates entnommen werden. Er wird jeweils absolut in Millisekun-

den und relativ zur Gesamtsynchronisationsdauer angegeben. Die Differenz zu 100% Ausführungsdauer entsteht durch andere Routinen des Agenten, die für das Konzept des Synchronisation keine Bedeutung haben.

7.4.1 Full Update

Auf dem Supplier wird eine Datenbank mit 5000 Beispieleinträgen angelegt. Die gesamte Synchronisationsdauer dieses Beispiels bleibt unter einer Stunde. Der Test wurde dreimal nacheinander durchgeführt. Die Durchschnittswerte der Ergebnisse werden in den folgenden Tabellen 7.1 und 7.2 aufgeführt. Absolutwerte sind auf volle Millisekunden genau gerundet, Relativwerte auf eine Stelle nach dem Komma genau.

Synchronisationsdaten		
	absolut (ms)	relativ
Gesamtdauer:	3243	100%
Mappingdauer:	3	0.1%
Konfliktlösungsdauer:	101	3.4%
Eintragungsdauer:	2965	91.4%
Leistungsdaten von MSAD		
	bind	search
Durchschnitt (ms)	6	2
Minimum (ms)	2	1
Maximum (ms)	8271	347
Standardabweichung (ms)	25	8

Tabelle 7.1: Synchronisationsdaten - Full Update A
Full Update von MSAD zu eDirectory.

7.4.2 Inkrementelles Update

Beide Server haben synchronisierte Datenbanken von 5000 Beispieleinträgen. Die Einträge je eines Verzeichnisses werden über den Zeitraum von 15 Minuten ständig verändert. Es werden LDAP Modify Operationen über die Einträge ausgeführt. Die Anzahl der Modifikationen pro Sekunde ist sehr hoch gewählt um einen sichtbaren Effekt am Verzeichnisdienst auszulösen. Dieser Umstand wird in der Auswertung Beachtung finden. Die Tests werden wie beim Full Update durchgeführt und der Durchschnitt der Ergebnisse wird in den Tabellen 7.3, 7.4 und 7.5 wiedergegeben.

Synchronisationsdaten		
	absolut (ms)	relativ
Gesamtdauer:	695	100%
Mappingdauer:	2	0.3%
Konfliktlösungsdauer:	6	0.9%
Eintragungsdauer:	633	91.1%
Leistungsdaten von eDirectory		
	bind	search
Durchschnitt (ms)	11	13
Minimum (ms)	2	2
Maximum (ms)	17515	2322
Standardabweichung (ms)	121	70

Tabelle 7.2: Synchronisationsdaten - Full Update B
Full Update von eDirectory zu MSAD.

	Dauer	
	absolut (ms)	relativ
Mapping:	2	0.2%
Konfliktlösungen:	92	10.2%
Eintragungen:	714	79%

Tabelle 7.3: Synchronisationsdaten - Inkrementelles Update A
Inkrementelles Update von eDirectory zu MSAD. eDirectory führte 2996 Modifikationen aus.

	Dauer	
	absolut (ms)	relativ
Mapping:	4	0.4%
Konfliktlösungen:	123	13.7%
Eintragungen:	681	76%

Tabelle 7.4: Synchronisationsdaten - Inkrementelles Update B
Inkrementelles Update von MSAD zu eDirectory. MSAD führte 8469 Modifikationen aus.

7.4.3 Vergleichstest

Jeder Server erhält eine Datenbank mit 5000 Beispieleinträgen. Es erfolgt für jeden Verzeichnisdienst ein Benchmark mit dem Authorisationsszenario über einen Zeitraum von 15 Minuten. Die Resultate werden in der Tabelle

	MS Active Directory		Novell eDirectory	
	bind	search	bind	search
Durchschnitt (ms)	4	1	10	7
Maximum (ms)	886	249	1186	714
Minimum (ms)	1	1	2	2
Standard Abweichung (ms)	5	1	24	17

Tabelle 7.5: Leistungsdaten - Inkrementelles Update
Authorisationsszenario während inkrementeller Synchronisation

7.6 auf Seite 63 gezeigt und bieten Vergleichswerte zu den vorherigen Tests.

	MS Active Directory		Novell eDirectory	
	bind	search	bind	search
Durchschnitt (ms)	3	2	7	6
Maximum (ms)	379	102	444	697
Minimum (ms)	3	1	2	2
Standard Abweichung (ms)	4	3	18	17

Tabelle 7.6: Leistungsdaten - Vergleichswerte
Authorisationsszenario ohne gleichzeitige Synchronisation

7.5 Auswertung

Die Meßergebnisse zeigen, daß der größte Teil der Synchronisationsdauer nicht von den Agenten beansprucht wird. Den größten Zeitaufwand verursachen die Modifikationsoperationen am Verzeichnis. Besonders hoch ist der absolute Wert für das Full Update zu Novell eDirectory. Beim Erstellen eines Benutzers erzeugt eDirectory für ihn ein Schlüsselpaar mit dem er in Zukunft seine Verbindugen sichert. Der Algorithmus ist sehr aufwendig und verlängert die Zeit zum Erzeugen des Accounts erheblich.

Die LDAP Modify Operationen der inkrementellen Updates benötigen weniger Zeit. Als Folge sinkt ihr Anteil an der Synchronisationsdauer im Vergleich zum Full Update. Er bleibt aber weiterhin der aufwendigste Teil.

Die Verfügbarkeit der Verzeichnisdienste wird bei einem Full Update relevant beeinflußt. Vor allem die Antwortzeiten von eDirectory werden länger und schwanken zudem stärker. Die gleichen Effekte treten in geringerem Maße auch bei MSAD auf. Die Frequenz der Modifikationsoperationen ist im Test des inkrementellen Updates sehr hoch gewählt. Im wirklichen Leben ist die Anzahl von schreibenden Operationen auf ein Verzeichnis im Vergleich zu den lesenden verschwindend gering. Unter realen Bedingungen wird demnach die Verfügbarkeit während inkrementeller Updates noch näher an der des Vergleichstests liegen.

Die Synchronisationsdauer kann nur wenig durch Optimierung von Mapping oder Konfliktlösung verbessert werden. Die Lösung für dieses Problem ist bei der Verringerung und Optimierung der Schreibzugriffe zu suchen.

Kapitel 8

Zusammenfassung

8.1 Bewertung

Verzeichnisdienste sind eine starke Stütze für die Administration großer Netzwerke. Mit ihnen können sehr große Zahlen an Benutzern und Konfigurationsdaten leicht verwaltet und verfügbar gemacht werden. Wie in Kapitel 2 gezeigt, gibt es für homogene Netzwerke sehr gute Lösungen. Sie wurden in Kapitel 3 mit den Forderungen des LDUP Projektes verglichen. Für Microsoft Netzwerke ist Microsoft Active Directory gut einsetzbar. Novell hat mit eDirecotory ein gutes Produkt, daß auf vielen Plattformen unterstützt wird. Eine freie Lösung existiert mit OpenLDAP, das aber qualitativ nicht so weit fortgeschritten ist wie die anderen beiden Programme. Die Entscheidung für einen dieser Verzeichnisdienste sollte an Hand der vorhandenen Ressourcen gefällt werden.

Vor allem in heterogenen Netzwerken ist die Entscheidung für einen bestimmten Verzeichnisdienst nicht leicht. In einem Universitätsnetzwerk stehen technischen Ansprüchen oft auch politische und finanzielle Fragen gegenüber. Bestehende Strukturen erschweren den Einsatz eines beliebigen Verzeichnisses und binden Standorte an einen Hersteller. Der Einsatz mehrerer Verzeichnisdienste mit verschiedenen Datensätzen widerspräche dem Sinn der Verzeichnisse. Deshalb wurden in dieser Arbeit bestehende Möglichkeiten zur Synchronisation der Daten diskutiert. In Kapitel 4 wurden Compaq LDSU, Microsoft Directory Synchronization Service und Novell DirXML vorgestellt. Die beiden letzteren sind sehr gut in einer heterogenen Novell/Microsoft Umgebung einsetzbar. Sie erfüllen die Anforderungen an sichere Konfliktlösung und an die Paßwortsynchronisation, die ein besonderes Problem darstellt.

In Netzwerken ohne ihren Herstellerverzeichnisdienst sind sowohl MSDSS als auch DirXML nutzlos. Aus diesem Grund wurde in Kapitel 5 ihre Ar-

beitsweise analysiert und zur Verallgemeinerung ihrer Konzepte genutzt. Im Zuge dessen wurde der *single-point-of-failure* auf Kosten der leichten Administrierbarkeit entfernt. Sie läßt sich mit etwas Aufwand wieder herstellen ohne das Konzept zu verletzen. Dieser Schritt wurde in dieser Arbeit nicht unternommen. Die Implementation erfolgte unter Verwendung von OpenLDAP und dem Backend back-ldap. Soweit es möglich war, wurden Empfehlungen von LDUP beachtet. Es ergab sich dabei, daß durch den eingeschränkten Zugriff der Agenten auf das Verzeichnis die Konfliktlösung nur auf Objektebene realisiert werden kann und es nicht möglich ist ringförmige Synchronisationstopologien aufzubauen.

Das Programm wurde mit der Synchronisation von MSAD und eDirectory getestet. Besonderes Interesse lag auf der Synchronisationsdauer und dem Einfluß der Agenten auf die Verzeichnisse. Als Ergebnis ließ sich feststellen, daß die Agenten nur bei Full Updates die Verzeichnisse relevant beeinflussen. Full Updates treten aber nur zur Initialisierung einer Synchronisation auf und können im Normalbetrieb vernachlässigt werden. Die Synchronisationsdauer hängt im grossen Maße nicht von den Agenten ab, sondern von der Schreibgeschwindigkeit des Verzeichnisses. Eine Verbesserung des Mappingalgorithmus oder der Konfliktlösung stellen nur sehr kleine Geschwindikeitszunahmen in Aussicht.

8.2 Ausblick

In großen Netzwerken ist der Einsatz von Verzeichnisdiensten in Zukunft zu empfehlen. Sie ersetzen auf einfache Weise die älteren Strukturen der einheitlichen Benutzerauthentifizierung. Sie verringern den administrativen Aufwand und erhöhen gleichzeitig die Aktualität der Daten.

Wenn sich in heterogenen Netzwerken der Einsatz unterschiedlicher Verzeichnisse nicht vermeiden läßt, sollte auf eines der Synchronisationsprodukte der Hersteller zurückgegriffen werden. In vielen Fällen sind sie die beste und vor allem einfachste Wahl.

Das Konzept der verteilten Synchronisation kann noch weiter verfeinert werden. Zum Beispiel kann man eine Schemaänderung der Verzeichnisse für die Agenten einführen. Sie können dann Statusinformationen zur Synchronistion im Verzeichnis ablegen und als Folge dessen ringförmige Topologien bilden. Sie müssen dazu die Zeitstempel der Ursprungsoperation in jedem Verzeichnis vermerken. Falls LDUP den Zugriff auf Zeitstempel der Attribut- und Attributwertebene möglich macht, kann sogar eine der Replikation gleichwertige Synchronisation aufgebaut werden.

In dieser Arbeit wurde an vielen Stellen auf das LDUP Projekt Bezug genommen. Es muß aber noch viel geleistet werden, bis LDUP die Akzeptanz findet, die LDAP schon hat. LDUP benötigte die Zeit von 1998 bis 2002, um sein erstes RFC zu veröffentlichen. In der Zwischenzeit hat jeder große Hersteller von Verzeichnisdiensten gute Replikations- und auch Synchronisationslösungen erarbeitet. In Anbetracht dessen ist es fraglich, ob die Weiterentwicklung von LDUP noch Sinn macht. Falls irgendwann detaillierte Spezifikationen erstellt werden, wird kaum ein Hersteller mehr Bereitschaft zeigen sie umzusetzen, da sich ihre proprietären Algorithmen etabliert haben.

Anhang A

Abkürzungen

ACDF	Access Control Decision Function
ACI	Access Control Information
ACL	Access Control List
ADSI	Active Directory Service Interface
API	Application Programmer Interface
ASN.1	Abstract Syntax Notation One
CCITT	Committee for International Telegraph and Telephone
CSN	Change Sequence Number
DAP	Directory Access Protocol
DAS	Directory Assistance Service
DC	Domain Controller
DIB	Directory Information Base
DIT	Directory Information Tree
DIXIE	Directory Interface to X.500 Implemented Efficiently
DN	Distinguished Name
DNS	Domain Name Service
DSA	Directory Service Agent
DUA	Directory User Agent
ISO	International Standards Organization
IETF	Internet Engineering Task Force
IPX	Internetwork Packet Exchange
ITU-T	International Telecomunication Union - Telecommunication Standardization Sector
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LDIF	LDAP Data Interchange Format
LDSU	LDAP Directory Synchronisation Utility
LDAP	Lightweight Directory Access Protocol
LDUP	LDAP Duplication/Replication/Update Protocols

MSAD	Microsoft Active Directory
MSDSS	Microsoft Directory Synchronization Service
NDS	Novell Directory Service
NOS	Network Operating System
NTLM	Windows NT LanManager
OID	Object Identifier
OSI	Open Systems Interconnection
RDF	Record Description File
RDN	Relative Distinguished Name
RFC	Request for Comments
SFN	Services for Netware
SASL	Simple Security and Authentication Layer
TCP/IP	Transmission Control Protocol / Internet Protocol
TLS	Transport Layer Security
UID	Unique Identifier
WINS	Windows Internet Naming Service
XML	Extensible Markup Language

Anhang B

Literaturverzeichnis

[1] ITU-T,
Information technology - Open Systems Interconnection - The Directory: Overview of concepts, models and services.,
Recommendation X.500, International Telecommunications Union, Genf, 2001
URL: http://www.itu.int/rec/recommendation.asp?type=folders&parent=T-REC-X.500

[2] ITU-T,
Information technology - Open Systems Interconnection - The Directory: Models.,
Recommendation X.501, International Telecommunications Union, Genf, 2001.
URL: http://www.itu.int/rec/recommendation.asp?type=folders&parent=T-REC-X.501

[3] ITU-T,
Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks.,
Recommendation X.509, International Telecommunications Union, Genf, 2001.
URL: http://www.itu.int/rec/recommendation.asp?type=folders&parent=T-REC-X.509

[4] ITU-T,
Information technology - Open Systems Interconnection - The Directory: Abstract service definition.,
Recommendation X.511, International Telecommunications Union, Genf, 2001.
URL: http://www.itu.int/rec/recommendation.asp?type=folders&parent=T-REC-X.511

[5] ITU-T,
Information technology - Open Systems Interconnection - The Directory: Procedures for distributed operation.,
Recommendation X.518, International Telecommunications Union, Genf, 2001.
URL: http://www.itu.int/rec/recommendation.asp?type=folders&parent=T-REC-X.518

[6] ITU-T,
Information technology - Open Systems Interconnection - The Directory: Protocol specitifcations.,
Recommendation X.519, International Telecommunications Union, Genf, 2001.
URL: http://www.itu.int/rec/recommendation.asp?type=folders&parent=T-REC-X.519

[7] ITU-T,
Information technology - Open Systems Interconnection - The Directory: Selected attribute types.,
Recommendation X.520, International Telecommunications Union, Genf, 2001.
URL: http://www.itu.int/rec/recommendation.asp?type=folders&parent=T-REC-X.520

[8] ITU-T.
Information technology - Open Systems Interconnection - The Directory: Selected object classes.,
Recommendation X.521, International Telecommunications Union,Genf, 2001.
URL: http://www.itu.int/rec/recommendation.asp?type=folders&parent=T-REC-X.521

[9] ITU-T,
Information technology - Open Systems Interconnection - The Directory: Replication.,
Recommendation X.525, International Telecommunications Union, Genf, 2001.
URL: http://www.itu.int/rec/recommendation.asp?type=folders&parent=T-REC-X.525

[10] ITU-T,
Information technology - Open Systems Interconnection - The Directory: Use of systems management for administration of the Directory.,
Recommendation X.530, International Telecommunications Union, Genf, 2001.

URL: http://www.itu.int/rec/recommendation.asp?type=folders&
parent=T-REC-X.530

[11] ITU-T,
International Reference Alphabet (IRA) (Formerly International Alphabet No. 5 or IA5) - Information technology - 7-bit coded character set for information interchange,
Recommendation T.50, International Telecommunications Union, Genf, 1992.
URL: http://www.itu.int/rec/recommendation.asp?type=items&
parent=T-REC-T.50-199209-I

[12] ITU-T,
Information technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation,
Recommendation X.680, International Telecommunications Union, Genf, 1997.
URL: http://www.itu.int/rec/recommendation.asp?type=folders&
parent=T-REC-X.680

[13] ITU-T,
Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
Recommendation X.690, International Telecommunications Union, Genf, 2002
http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&
parent=T-REC-X.690

[14] Jens Banning,
LDAP unter Linux, Netzwerkinformationen in Verzeichnissen verwalten, Addison-Wesley, 2001

[15] Franco Michela, Markus Palme,
Microsoft Active Directory,
Microsoft Press, 1999

[16] IST LDAP Project, UC Berkeley CalNet Directory Service,
Overview and History of Directory Services - longer version,
University of California, 1999
URL: http://ldap-project.berkeley.edu/reports/overview/overview-longer.html

[17] Norbert Klasen,
Directory Services for Linux, RWTH Aachen, 2001
URL: http://www.daasi.de/staff/norbert/thesis/html/thesis.html

[18] Microsoft Corporation,
Understanding the Role of Directory Services Versus Relational Databases, Windows 2000 Technical Resources, 2001
URL: http://www.microsoft.com/windows2000/techinfo/howitworks/activedirectory/dsvsrd.asp

[19] Microsoft Corporation,
Active Directory Architecture, Microsoft TechNet, 2002,
URL: http://www.microsoft.com/technet/prodtechnol/ad/windows2000/deploy/projplan/adarch.asp

[20] Microsoft Corporation,
Active Directory Replication Modell, 2002,
http://www.microsoft.com/windows2000/techinfo/reskit/samplechapters/dsbh/dsbh_rep_mqeg.asp

[21] Microsoft Corporation,
User Authentication with Windows NT, Q 102716,
Microsoft Knowledge Base, 1993
URL: http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q102716&

[22] Microsoft Corporation,
Platform SDK: Directory Services - LDAP_SERVER_NOTIFICATION_OID, Microsoft Developer Network, 2002,
URL: http://msdn.microsoft.com/library/en-us/netdir/ldap/ldap_server_notification_oid.asp

[23] Microsoft Corporation,
Platform SDK: Directory Services - LDAP_SERVER_DIRSYNC_OID, Microsoft Developer Network, 2002,
URL: http://msdn.microsoft.com/library/en-us/netdir/ldap/ldap_server_dirsync_oid.asp

[24] Microsoft Corporation,
MSDSS Technical Overview,2001
URL: http://www.microsoft.com/windows2000/sfn/msdss.asp

[25] Microsoft Corporation,
MSDSS Deployment: Understanding Synchronization and Migration, White Paper, Redmond, 2000

[26] Novell Inc.,
Novell eDirectory Administration Guide, 2001

URL: http://www.novell.com/documentation/lg/
ndsedir86/pdfdoc/taoenu.pdf

[27] Novell Inc.,
Novell DirXML Administration Guide, 2002
URL: http://www.novell.com/documentation/lg/
dirxml11/pdfdoc/dirxml/dirxml.pdf

[28] Novell Inc.,
Novell DirXML Driver for Active Directory Implementation Guide, 2002

[29] Novell Inc.,
Novell: DirXML Password Synchronization for Windows Administration Guide, 2002
URL: http://www.novell.com/documentation/lg/
pwdsync10/pdfdoc/passsync.pdf

[30] Novell,
NDS and LDAP
URL: http://developer.novell.com/ndk/doc/ocx/ocx__enu/ref/nwidir/
topic_nds_and_ldap_description.htm

[31] Compaq Computer Corporation,
LDAP Directory Synchronizer Online Documentation, 2002
URL: http://www.compaq.com/services/messaging/mg_ldap.html

[32] CCITT Blue Book,
Data Communication Networks: Directory,
Recommendations X.500-X.521, December 1988

[33] Alan O. Freier, Philip Karlton, Paul C. Kocher,
The SSL Protocol Version 3.0,
1996 URL: http://www.netscape.com/eng/ssl3/draft302.txt

[34] *Netscape Directory SDK 3.0 for C Programmer's Guide*,
1998 URL: http://nimbus.ocis.temple.edu/ldap/contents.htm

[35] W. Yeong, T. Howes, S. Kille,
X.500 Lightweight Directory Access Protocol,
RFC 1487, 1993,
URL: http://www.ietf.org/rfc/rfc1487.txt

[36] W. Yeong, T. Howes, S. Kille,
Lightweight Directory Access Protocol,
RFC 1777,1995,
URL: http://www.ietf.org/rfc/rfc1777.txt

[37] M. Wahl, T. Howes, S.Kille,
Lightweight Directory Access Protocol (v3),
RFC 2251, 1997,
URL: http://www.ietf.org/rfc/rfc2251.txt

[38] M. Wahl, A. Coulbeck, T. Howes, S. Kille,
Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions
RFC 2252, 1997,
URL: http://www.ietf.org/rfc/rfc2252.txt

[39] M. Wahl
A Summary of the X.500(96) User Schema for use with LDAPv3
RFC 2256, 1997,
URL: http://www.ietf.org/rfc/rfc2256.txt

[40] T. Howes, M. Smith, B. Beecher,
DIXIE Protocol Specification,
RFC 1249, 1991, URL: http://www.ietf.org/rfc/rfc1249.txt

[41] M. Rose,
Directory Assistance Service,
RFC 1202, 1991,
URL: http://www.ietf.org/rfc/rfc1202.txt

[42] F. Yergeau,
UTF-8, a transformation format of Unicode and ISO 10646,
RFC 2044, 1996,
URL: http://www.ietf.org/rfc/rfc2044.txt

[43] J. Myers,
Simple Authentication and Security Layer (SASL),
RFC 2222, 1997,
URL: http://www.ietf.org/rfc/rfc2222.txt

[44] K. Zeilenga,
OpenLDAP Root Service - An experimental LDAP referral service,
RFC 3088, 2001,
URL: http://www.ietf.org/rfc/rfc3088.txt

[45] A. Gulbrandsen, P. Vixie, L. Esibov,
A DNS RR for specifying the location of services (DNS SRV),
RFC 2782, 2000,
URL: http://www.ietf.org/rfc/rfc2782.txt

[46] E. Stokes, R. Weiser, R. Moats, R. Huber,
Lightweight Directory Access Protocol (version 3) - Replication Requi-

rements RFC 3384, 2002,
URL: http://www.ietf.org/rfc/rfc3384.txt

[47] G. Good,
The LDAP Data Interchange Format (LDIF) - Technical Specification
RFC 2849, 2000,
URL: http://www.ietf.org/rfc/rfc2849.txt

[48] *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane*,
ISO/IEC 10646-1, 1993

[49] T. Dierks, C.Allen,
The TLS Protocol Version 1.0,
RFC 2246, 1999
URL: http://www.ietf.org/rfc/rfc2246.txt

[50] S. Kille, M. Wahl, A. Grimstad, R. Huber, S. Sataluri,
Usind Domains in LDAP/X.500 Distinguished Names,
RFC 2247, 199
URL: http://www.ietf.org/rfc/rfc2247.txt

[51] D. L. Mills,
Network Time Protocol (Version 3),
RFC 1305, 1992,
URL: http://www.ietf.org/rfc/rfc1305.txt

[52] D. L. Mills,
Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI,
RFC 2030, 1996,
URL: http://www.ietf.org/rfc/rfc2030.txt

[53] Massachusetts Institute of Technology,
Kerberos: The Network Authentication Protocol,
URL: http://web.mit.edu/kerberos/www/

[54] D. W. Chadwick,
Understanding X.500 - The Directory,
1996,
URL: http://www.isi.salford.ac.uk/staff/dwc/Version.Web/Contents.htm

[55] Heinz Johner, Larry Brown, Franz-Stefan Hinner, Wolfgang Reis, Johan Westman,
Understanding LDAP,
International Technical Support Organization, 1998,
URL: http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg244986.pdf

[56] Chris Apple, John Strassener,
LDAP Duplication/Replication/Update Protocols (ldup),
Internet Engineering Task Force, 2002,
URL: http://www.ietf.org/html.charters/ldup-charter.html

[57] S. Legg, A. Payne,
LDUP Update Reconciliation Procedures,
Internet Engineering Task Force, 2002,
URL: http://www.ietf.org/proceedings/02jul/I-D/draft-ietf-ldup-urp-06.txt
Dieser Entwurf wurde verworfen und nicht als RFC angenommen..

[58] Ed Reed,
LDAP Replication Architecture,
Internet Engineering Task Force, 2002, URL: http://www.ietf.org/proceedings/02jul/I-D/draft-ietf-ldup-model-07.txt
Dieser Entwurf wurde verworfen und als RFC angenommen.

[59] M. Smith, O. Natkovich, J. Parham,
LDAP Client Update Protocol,
Internet Engineering Task Force, 2002,
URL: http://www.ietf.org/internet-drafts/draft-ietf-ldup-lcup-03.txt
Diese Arbeit ist in der Entwursphase.

[60] *The SLAPD and SLURPD Administrators Guide*,
Universitiy of Michigan, Version 3.3, 1996,
URL: http://www.umich.edu/ dirsvcs/ldap/doc/guides/slapd/tocall.html

[61] Gari Hein,
Active Directory: How Dows It Measure Up?,
NetWare Connection, 2000,
URL: http://www.nwconnection.com/2000_11/mad/

Anhang C

Eidesstattliche Erklärung

Ich versichere, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Ort, Datum Unterschrift