phinex
informatik ag

Der Director

für Fortgeschrittene

Thomas Aeby, Phinex Informatik AG
13. Dezember 2005

Inhalt

1.LDAP – Zugriff und Datenmodell....2
LDAP-Zugriff mit LDAP-Utils....2
Datenbankgrundlagen....2
Datenbankstruktur....3
Suchoperationen....4
Schemata....4
Authentifizierung und Zugriffskontrolle....4
OpenLDAP-Wartung....5
Backup....5
Indexierung....6
Beziehung zwischen LDAP und dem Director....6
Low-Level DB-Zugriff....6
Objektabstraktion des Director / Editorschemata....6
Spezielle Folder – Inventory-Objekte....7
Uebungen....7
2.Director Server im Betrieb....9
Authentifizierung....9
Director-Komponenten....9
Kommunikation....9
Server-Kommandos auf Port 10371....11
Job-Queues....11
Ablageformat....12
Locking....12
Uebungen....12
3.Passwortsynchronisation mit Microsoft SFU / SSO....14
Funktionsschema SSO....14
4.Director-Konfiguration....16
Hauptkonfiguration sfidirector.conf....16
Events und Aktionen: objevents....16
Event: directorObjectEvent....16
Aktionen: directorTask....17
Ein Beispiel....17
Hartcodierte Aktionen....18
Uebungen....18
Konfigurationsverteilung....18
Dateiformate....19
Konfigurationsverteilung testen....19
Schema und Editorschemata....20
ObjRegistry-Einträge....20
Editorschema....21
Uebungen....24
5.Einführung in den Director-Quelltext....25
Build-Prozedur....25
Modularität mit monolithischem Ergebnis....25
Erste Uebersicht....26
Setup und Tools....27
Einfache Beispiele....28
Dateiformat für die Konfigurationsverwaltung....28

1. LDAP – Zugriff und Datenmodell

Der Director legt Verwaltungsinformation in einer per LDAP ansprechbaren Datenbank ab. Das verwendete Datenmodell orientiert sich an bestehenden Standards und an anderen LDAP-basierten Werkzeugen. Es kann deshalb durchaus auch Sinn machen, die LDAP-Datenbank auch ausserhalb des Directors zu verwenden. Dieses Kapitel geht auf Datenmodelle und Zugriffsmöglichkeiten ein. Der Text geht dabei von der Verwendung von OpenLDAP aus, die meisten Aussagen lassen sich aber analog auf beliebige andere Datenbanken mit LDAP-Zugriff anwenden.

LDAP-Zugriff mit LDAP-Utils

Es existieren viele gute Werkzeuge, um auf LDAP-Datenbanken zuzugreifen. Zu OpenLDAP gehören unter anderem eine Reihe von kommandozeilenorientierten LDAP-Clients, die wir in diesem Dokument verwenden werden.

Am häufigsten wird das Kommando

```
ldapsearch
```

verwendet – damit lassen sich lesende Zugriffe auf die Datenbank machen. Mit einigen weiteren Kommandos lassen sich alle Grundoperationen erledigen:

- ldapadd – Objekt in die Datenbank aufnehmen
- ldapmodify – bestehendes Objekt verändern
- ldapdelete – bestehendes Objekt löschen

Datenbankgrundlagen

Anders als eine klassische relationale Datenbank sind Datensätze in einer LDAP-DB *Objekte* bestehend aus einzelnen *Attributen*, die in einer hierarchischen *Ordnerstruktur* angelegt sind. Jedes Objekt ist teil einer oder mehrerer *Objektklassen*. Die Objektklasse definiert dabei, welche Attribute in einem Objekt dieser Klasse enthalten sein *muss* (MUST) und welche Attribute enthalten sein *dürfen* (MAY). Die Objektklasse selber ist in einem Attribut namens *objectClass* ebenfalls teil des Objektes. Attribute können einen oder mehrere Werte enthalten. Die *Attributdefinition* legt fest, welchen *Datentyp* Werte haben dürfen, ob genau *ein* (SINGLE-VALUE) oder *mehrere* Werte erlaubt sind, und mit welcher Vergleichsoperation Suchoperationen auf dieses Attribut zugreifen sollen. Ein Satz von Definitionen von Attributen und Objektklassen wird als *Schema* bezeichnet.

Jedes Objekt wird in der Datenbank durch einen eindeutigen *Schlüssel* identifiziert, den sogenannten *Distinguished Name* (DN). Der DN ist ein textueller Wert und setzt sich obligatorisch aus mit Komma getrennten *Attribute=Wert*-Komponenten zusammen. Dabei identifiziert das 1. Attribut-/Wert-Paar das Objekt selber, die weiteren dessen Position in der Hierarchie.

Zeit für ein Beispiel. Ein Benutzerobjekt in der LDIF-Schreibweise (LDIF: LDAP Data Interchange Format) kann beispielsweise so aussehen:

```
dn: uid=muster,ou=hosting,ou=people,dc=domain,dc=example
uidNumber: 1028
```

```
cn: Peter Muster
cn: Muster Peter
mailName: Peter.Muster
gecos: Peter Muster
gidNumber: 100
uid: muster
sfipersonClass: Site User
mailAlias: Peter.Muster
homeDirectory: /home/muster
sn: Muster
mail: Peter.Muster@domain.example
givenName: Peter
loginShell: /bin/bash
userPassword: {crypt}assvYTPWirXbY
userPassword: {crypt}$1$B4BQMbUG$i.5nxfjUJ.JRf4ASTNg8f1
objectClass: directorUser
objectClass: posixAccount
objectClass: person
objectClass: shadowAccount
```

Dabei bedeuten:

- dn: verwendet wird das Attribut uid als eindeutiger Schlüssel, das Objekt befindet sich an der Stelle /example/domain/people/hosting/muster in der Hierarchie
- es handelt sich um ein Objekt der Klassen directorUser (Director-spezifisch), posixAccount (RFC2307), shadowAccount (RFC2307) und person (RFC2256)
- diverse Attribute, z.B. cn (Common Name) mit zwei Werten „Peter Muster" und „Muster Peter"

Datenbankstruktur

Eine LDAP-Datenbank ist unter einem *Root-Node* angehängt, damit wird die oberste Hierarchiestufe bezeichnet, die noch zu dieser Datenbank gehört (ähnlich zu einer DNS-Zone).

Unter diesem Root-Node sind die Objekte hierarchisch in einer Folder-ähnlichen Struktur abgelegt. Hierarchiestufen, die der Director anlegt, entsprechen per Konvention immer einem Objekt der Klasse organizationalUnit (ou). Um die Hierarchie der Director-Datenbank anzeigen zu lassen, können wir also folgendes ldapsearch-Kommando absetzen:

```
ldapsearch -x -b dc=domain,dc=example objectClass=organizationalUnit
```

Den Root-Node der OpenLDAP-Datenbank findet sich in der OpenLDAP-Konfiguration /etc/ldap/slapd.conf als Einstellung *suffix*.

Damit bei Aufrufen von ldap-Kommandos nicht jedes Mal per Option -b der Root-Node angegeben werden muss, kann dieser in der Datei /etc/ldap/ldap.conf als Einstellung *BASE* abgelegt werden.

Suchoperationen

Lesender Zugriff auf die Datenbank ist prinzipiell mit dem ldapsearch-Werkzeug möglich. Dieses Werkzeug ist eng an das LDAP-Protokoll geknüpft, das als Haupt-Abfragemethode die Suche (ähnlich wie ein SELECT bei SQL) kennt.

Mittels ldapsearch lässt sich die Datenbank nach Objekten mit bestimmten Werten eines Attributes durchsuchen:

- ldapsearch 'cn=Peter Muster' sucht nach Objekten mit einem cn „Peter Muster"
- ldapsearch 'cn=Peter*' sucht nach Objekten mit einem cn, der mit „Peter" beginnt
- ldapsearch 'uid=*' sucht nach Objekten, für die das Attribut uid definiert ist (also mindestens einen beliebigen Wert enthält)

Suchbedingungen lassen sich logisch verknüpfen:

- ldapsearch '(&(uid=m*)(objectclass=directorUser))' sucht nach Objekten der Klasse directorUser mit einer gesetzten uid, die mit m beginnt
- ldapsearch '(!(uid=*))' sucht nach Objekten ohne uid-Attribut

Eine vollständige Definition der LDAP-Suchausdrücke findet sich in RFC1558.

Schemata

Eine Sammlung von Attributdefinitionen und Objektklassen bezeichnet man als *Schema*. Ein solches Schema lässt sich in OpenLDAP über die slapd.conf-Konfigurationsdatei einbinden: Die *include*-Anweisung erlaubt es, Schema-Definitionen im gleichen Format, wie sie in RFCs verwendet werden, direkt einzubinden.

Der Director verwendet zusätzlich zu einigen erforderlichen Standardschemata ein eigenes, das einige Director-spezifische Attribute und Objektklassen definiert. Es befindet sich in /usr/share/sfidirector/etc/director.schema.

Vom Netscape Directory Server abstammende LDAP-Server verwenden normalerweise Schemata im LDIF-Format, wie das Schema unter /usr/share/sfidirector/director.schema.ldif.

Authentifizierung und Zugriffskontrolle

Der Zugriff auf ganze Datenbanken, einzelne Teilbäume, einzelne Objekte und auch auf einzelne Attribute einer LDAP-Datenbank lässt sich einschränken. Deshalb kennt LDAP prinzipiell ein Authentifizierungskonzept. Im Zusammenhang mit dem Director wird bisher ausschliesslich die sogenannte *Simple Authentication* verwendet. Bei dieser Authentifizierungsart übergibt der Client dem LDAP-Server die DN eines Benutzerobjektes in der Datenbank und das Passwort des zugehörigen Benutzers.

Damit nicht bei jeder Director-Operation ein Passwort eingegeben werden muss, wird für den Director in der Datei ~/.director/credentials die DN und das Passwort des zu verwendenden LDAP-Benutzers abgelegt.

Wird kein Benutzer/Passwort angegeben, dann erfolgt der LDAP-Zugriff anonym. Auch dem anonymen Benutzer können Zugriffsrechte auf die Datenbank gewährt werden.

Welche Rechte ein bestimmter Benutzer hat, wird für OpenLDAP in der Datei slapd.-conf mittels der *access*-Einstellung definiert. z.B.:

- ```
  access to attr=userpassword,ntpassword,lmpassword by self write
  by dn="uid=root,ou=people,dc=domain,dc=example" write
  by * compare
  ```

  bedeutet, dass auf die Attribute userpassword, ntpassword, lmpassword der jeweilige Benutzer selbst lesend und schreibend, der Benutzer uid=root,... ebenfalls lesend und schreibend und alle anderen (also auch der anonyme Benutzer!) vergleichend zugreifen kann. Ein Benutzer kann also sein eigenes Passwort setzen. Der anonyme Benutzer kann zwar prüfen, ob ein bestimmtes Passwort (genauer: ein bestimmter Hash) mit dem gespeicherten übereinstimmt, er kann jedoch den Hash selber nicht auslesen.
- ```
  access to dn.regex="dc=domain,dc=example" by self write
  by dn.regex="uid=root,ou=people,dc=graeff,dc=com" write
  by * read
  ```

  bedeutet,, dass der Benutzer uid=root,... lesend und schreibend auf alle Objekte unter dc=domain,dc=example zugreifen kann, alle anderen (inkl. der anonyme Benutzer!) nur lesend

Die Art der Berechtigungsvergabe wird im LDAP-Standard nicht festgelegt, da es auf das Kommunikationsprotokoll keinen Einfluss hat und Berechtigungen leider auch nicht per LDAP festgelegt werden können. Andere LDAP-Server haben andere Berechtigungsmodelle.

Neben der Simple Authentication sind auch noch andere Authentifizierungsmethoden definiert, u.a. mittels SSL-Zertifikaten und Kerberos. Diese werden zur Zeit vom Director nicht unterstützt.

OpenLDAP-Wartung

Backup

OpenLDAP legt seine Daten in einem Binärformat ab. Es empfiehlt sich zu Backupzwecken die Daten jeweils im LDIF-Format exportiert auch noch abzulegen. Dies geschieht am einfachsten mit dem Kommando *slapcat*, das einen vollständigen LDIF-Dump der Datenbank ausgibt.

Indexierung

Um die Verarbeitung von Suchoperationen zu beschleunigen verwendet auch OpenLDAP Indexe. Diese müssen allerdings manuell konfiguriert werden. Dazu dient die Einstellung *index* in slapd.conf. Ein Index bezieht sich immer auf ein Attribut. Je nach Suchoperationen, die durch den Index beschleunigt werden sollen, können In-

dexe des Typs *pres* (present, z.B. für Suchen wie uid=*), *eq* (equal, z.B für Suchen wie uid=muster), *subinitial* (z.B. für Suchen wie cn=M*) definiert werden.

Nach Aenderung der Indexkonfiguration müssen die Indexe neu erstellt werden. Dies geschieht mit dem Kommando *slapindex* (bei heruntergefahrenem LDAP-Server ohne Argumente ausführen).

Leider kann es unter gewissen Umständen geschehen, dass ein Index zerschossen wird – das macht sich durch „seltsames" Suchverhalten bemerkbar (d.h. Objekte werden mit speziellen Suchausdrücken nicht gefunden, obwohl sie in der Datenbank noch vorhanden sind). In diesem Falle lohnt sich ein Neuaufsetzen der Indexe mittels slapindex.

Beziehung zwischen LDAP und dem Director

In der sfidirector.conf-Datei wird festgelegt, welche logische Strukturen des Director-Repositories welchen Teilen der LDAP-Datenbank entsprechen. Speziell ausgezeichnet ist dabei die Einstellung Top, die typischerweise dem Root-Node der LDAP-Datenbank entspricht. Andere Teilstrukturen des Directors (z.B. People, Hosts, etc.) sind per Default relativ zur Top-Datenbank definiert.

Die zu verwendeten Authentifizierungsinformationen zur Anmeldung an einen LDAP-Server müssen mindestens für den root-Benutzer in ~root/.director/credentials abgelegt werden. Je nach verwendetem Berechtigungskonzept der Director-Oberfläche muss ausserdem /etc/bigsister/director_credentials analog bestückt werden.

Low-Level DB-Zugriff

Mit dem *copy*-Kommando des Directors kann direkt auf die Datenbank zugegriffen werden. Ein

```
sfidirector copy -t directory:Top ldiffile:stdio:
```

entspricht dabei ziemlich weitgehend einem

```
ldapsearch 'objectclass=*'
```

Jedoch kann mit copy auch schreibend zugegriffen werden, z.B. mit

```
sfidirector copy ldiffile:stdio: directory:People
dn: uid=muster
uid: muster
cn: Peter Muster
...
```

Objektabstraktion des Director / Editorschemata

Um den Director-Benutzer nicht zwingend mit den „rohen" LDAP-Attributen zu konfrontieren, kennt der Director eine weitere Abstraktionsebene: Die sogenannten *Editorschemata* definieren, wie ein LDAP-Objekt angezeigt und editiert werden kann. Diese Editorschemata finden sich unter /usr/share/sfidirector/etc/schema und sind registriert in /etc/sfidirector/objRegistry.ldif.

Die „höheren" Kommandos des Director (z.B. list, modify, delete, die Oberfläche, etc.) verwenden alle die Editorschemata, um LDAP-Objekte Benutzergerecht anzu-

zeigen oder aus der Anzeigeform wieder in ihre LDAP-Repräsentation umzuwandeln.

Nachteil dieses Ansatzes ist, dass sich mit dem Director keine generischen LDAP-Objekte bearbeiten lassen – nur Objekte, für die ein geeignetes Editorschema definiert wurde, sind per Director bearbeitbar.

Werden sitespezifische Schemaerweiterungen gemacht, so müssen also auch die Director-Editorschemata entsprechend erweitert werden.

Spezielle Folder – Inventory-Objekte

Weiter oben wurde behauptet, dass der Director ausschliesslich Objekte der Klasse organizationalUnit verwendet, um Ordnerstrukturen abzubilden. Es gibt eine Ausnahme zu dieser Regel: Inventardaten zu verwalteten Rechnern werden direkt unterhalb des zugehörigen Rechnerobjekts mit der Objektklasse directorInventoryData abgelegt.

Uebungen

- Machen Sie sich mit den ldap-Clientkommandos ldapsearch / ldapadd / ldapmodify / ldapdelete vertraut
- Beschaffen Sie sich einen graphischen LDAP-Editor. Was sind Vor- und Nachteile gegenüber der Director-Oberfläche?
- Zeichnen Sie die hierarchische Struktur der LDAP-DB auf. Vergleichen Sie mit der Director-Oberfläche.
- Zeigen Sie mit einem LDAP-Client einige typische Objekte auf dem Bildschirm an. Versuchen Sie, die passenden Objektklassen und Attribute in den Schemadateien zu finden und zu verstehen. Nutzen Sie ebenfalls den Schema Registry Service auf http://www.schemareg.org
- Versuchen Sie mittels ldapadd ein Objekt selber hinzuzufügen.
- Falls Sie Zugriff auf einen LDAP-Server mit vergleichbaren Verwaltungsinformationen wie der Director (z.B. ActiveDirectory, Novell eDirectory, etc.) haben, durchsuchen Sie dieses Verzeichnis mit Ihrem LDAP-Client. Was stellen Sie fest?
- Vergleichen Sie den Zugriff mittels ldapsearch mit dem Zugriff via „sfidirector list“ oder „sfidirector copy“. Was bewirkt die „-c“-Option beim list-Kommando?
- Machen Sie sich mit den im Text erwähnten RFCs vertraut (z.B. über http://www.rfc-archive.org/)
- Sie möchten in Ihrer Umgebung weitere Angaben zu einem Benutzer, z.B. dessen Abteilung oder Kostenstelle mitpflegen. Erweitern Sie das Schema des Datenbankservers. Im Kapitel Schema und Editorschemata auf Seite 18 erfahren Sie, wie Sie dafür sorgen können, dass auch der Director und seine Oberfläche von neuen Attributen erfahren.

2. Director Server im Betrieb

Authentifizierung

s. auch Abschnitt Authentifizierung im Kapitel LDAP. Der Director-Server muss sich gegenüber dem LDAP-Server authentifizieren können. Dazu dient die Datei ~root/.director/credentials, in der die Anmeldedaten für den/die jeweiligen LDAP-Server abgelegt sein müssen.

Soll das Berechtigungskonzept der Director-Oberfläche genutzt werden, muss ausserdem in /etc/bigsister/director_credentials festgelegt werden, mit welchen Anmeldedaten sich die Oberfläche am LDAP-Server anmelden soll. Ist die Datei nicht vorhanden, so wird die Anmeldung an der Oberfläche auch LDAP-seitig verwendet, d.h. der Benutzer, der sich an der Oberfläche anmeldet muss auch Datenbankseitig Zugangsberechtigungen haben.

Director-Komponenten

Der Director besteht aus mehreren Komponenten. Auf dem Director-Server läuft typischerweise der *Queueworker* (meist auch Director-Server genannt, weil er die Kernkomponente darstellt), das *Frontend-Interface* (bietet Dienste an für die Director-Oberfläche) und ein Webserver mit der Directoroberfläche *BigClerk*. Queueworker und Frontend-Interface laufen permanent und werden durch die Init-Skripte *sfidirector* und *frinterface* gestartet.

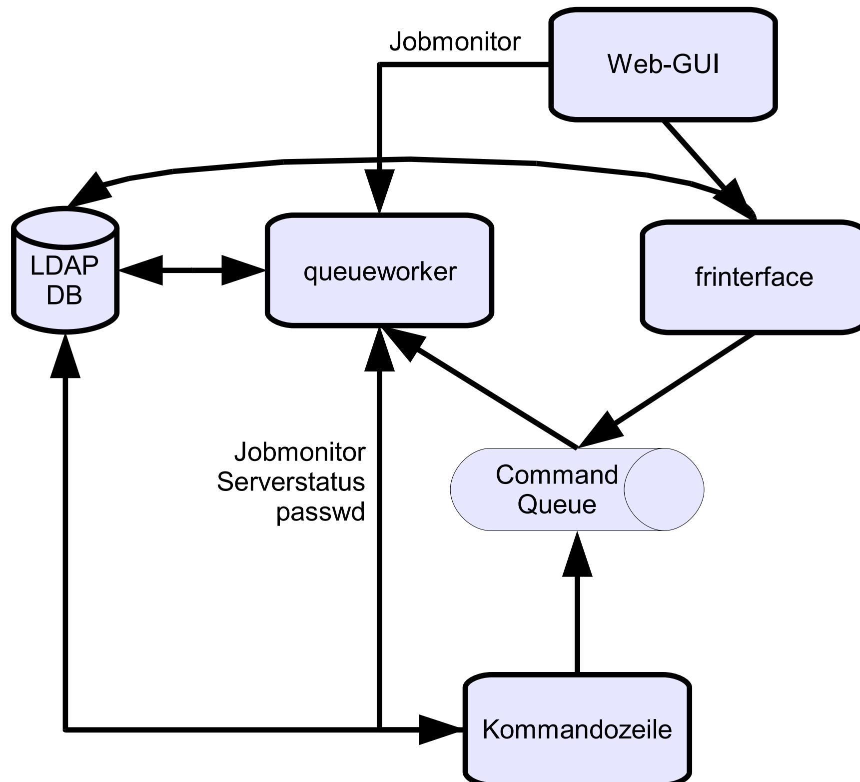
Der Queueworker verwaltet Jobs, die von der Oberfläche, der Kommandozeilenwerkzeuge oder von der Auto-Update-Funktion aufgesetzt werden. Zu ihrer eigentlichen Ausführung startet der Queueworker jedoch jeweils eine Instanz des *Agent* (per Shell oder Remote-Shell). Die Agents laufen also im Gegensatz zu den Serverkomponenten nicht permanent, sondern nur bei Bedarf.

Kommunikation

Kommunikation mit dem Director-Server findet auf mehrere Arten statt:

- *Oberfläche*. Die Web-Oberfläche kommuniziert über das „frinterface"-Protokoll über den Port 10372 mit dem frinterface-Server. Das frinterface-Protokoll ist im Dokument doc/interfaces/frontend (Quelltext-Verzeichnis) beschrieben.
- *Frinterface*. Das Frontend-Interface kommuniziert ausschliesslich über die Command-Queue mit dem Director-Server, über die es Jobs zur Ausführung übergeben kann.
- *Kommandozeile*. Wie das Frontend-Interface kommunizieren Kommandozeilenwerkzeuge normalerweise über die Command-Queue mit dem Director-Server. Ausnahmen s.u.
- *Jobmonitor*. Informationen zu gerade ablaufenden oder bereits bearbeiteten Jobs gelangen an den Director-Server via einem HTTP-Interface auf Port 10371.
- *Serverstatus*. Abfragen des Serverstatus (init-Skript, sfidirector serverstatus) werden ebenfalls über Port 10371 ausgeführt.

- *Passwortänderungen*. Benutzerpasswortänderungen können neben dem direkten Editieren des Benutzerobjektes im Director auch über eine HTTP-Anfrage an Port 10371 ausgeführt werden. Dies wird vom *sfidirector passwd* Kommando verwendet.
- *Agent*. Der Director-Server kommuniziert mit dem Agent über eine Shell bzw. Pipes (z.B. ssh). Das verwendete Protokoll wird nur Director-intern verwendet und es gibt deshalb keine offizielle Dokumentation.



Server-Kommandos auf Port 10371

Auf Port 10371 nimmt der Director-Server (queueworker) einige Kommandos und Statusabfragen entgegen. Das verwendete Protokoll entspricht einem Subset des HTTP-Protokolls und Abfragen lassen sich über einen beliebigen Web-Browser oder das Kommandozeilen-Utility *GET* starten.

Einige ausgewählte Funktionen sind:

- Serverstatus: http://localhost:10371/scheduler/status.txt
- Queue Rescan: http://localhost:10371/scheduler/rescan
- Server auf geändertes Objekt aufmerksam machen: http://localhost:10371/scheduler/modified?directory=<directory>&filter=<suchausdruck>
- Job-Monitor: http://localhost:10371/job/<jobid>.txt

- Passwort-Aenderung: http://localhost:10371/ops/chpw?login=<login>&old_pw=<old-password>&new_pw=<new-password>

Job-Queues

Die Hauptaufgabe des Director-Servers (Queueworker) ist es, die Job-Queues zu verwalten und – natürlich – die darin befindlichen Jobs auszuführen. Dazu unterhält er 3 Queues: Command Queue, Job Queue und Job Archive.

In die *Command Queue* stellen Director-Clients (Web-Oberfläche/Frontend-Interface, Kommandozeile) neue Jobs. Nach dem Einstellen machen die Clients den Server auf den neuen Inhalt der Queue aufmerksam indem sie ein Queue-Rescan-Kommando (s.o.) an den Kommando-Port des Servers senden. So oder so prüft allerdings der Server regelmässig (Default: 1-mal pro Minute) die Command Queue. Eine Client-Aktion führt typischerweise zu einer ganzen Reihe von Jobs in der Queue, die untereinander verknüpft sind.

Sobald der Server die Kontrolle über Jobs in der Command Queue übernimmt, werden diese in die *Job Queue* übernommen. Diese enthält alle Jobs, die noch auszuführen sind und Jobs, die mit noch auszuführenden Jobs verknüpft sind. Mit dem *sfidirector queuestat* Kommando kann der Inhalt der Job Queue angezeigt werden – dieses Kommando greift allerdings auf die Job-Queue-Datenbank zu, die typischerweise dem Serverstatus ein paar Sekunden hinterherhinkt. Live-Zugriff auf den Server haben hingegen das serverstatus-Kommando und der Jobmonitor.

Abgearbeitete Jobs werden nach einer definierten Zeitspanne (sfidirector.conf: ArchiveMoveTime) ins *Job Archive* verschoben. Dort werden sie zu Loggingzwecken eine weitere Zeitspanne aufbewahrt (ArchivePurgeTime). Weil das Löschen aus dem Job-Archive eine resourcenhungrige Aktivität ist, wird dies nur selten durchgeführt (ArchivePurgeCycle).

Ablageformat

Prinzipiell werden Jobs als LDAP-Objekte gespeichert. Typische LDAP-Server wie z.B. OpenLDAP sind allerdings vor allem auf Leseoperationen optimiert und zumindest OpenLDAP ist mit der Speicherung der intensiv mutierten Queues etwas überfordert. Aus diesem Grund werden die Queues nicht in der normalen LDAP-DB, sondern im Filesystem unter /var/lib/sfidirector in den Verzeichnissen *cmdqueue*, *jobqueue* und *jobarchive* gespeichert.

Locking

Um aufwändige Transaktionsmechanismen zu vermeiden, hat jeweils nur ein Prozess gleichzeitig schreibenden Zugriff auf eine Queue. Um dies zu gewährleisten wird unter /var/lib/sfidirector/locks jeweils eine Lock-Datei angelegt. Locks sind „alternd", d.h. selbst wenn ein Inhaber eines Locks abstürzen sollte ohne seine Locks sauber zu entfernen, so wird dieser nach einer gewissen Zeit (ca. 4 Minuten) automatisch als abgelaufen betrachtet.

Uebungen

- Fahren Sie den Director-Server herunter. Legen Sie nun z.B. mit dem Kommando `sfidirector -v event ping Hosts:einrechner` Jobs in der Command Queue an. Finden Sie die Job-Einträge im Dateisystem und schauen Sie sich an. Starten Sie nun den Director Server und untersuchen Sie, was geschieht.
- Der Director-Server hält Locks auf die Job-Queues. Was geschieht, wenn der Rechner, auf dem der Director Server läuft, einen Stromausfall hat und deshalb die alten Locks beim Hochfahren des neuen Servers noch bestehen? Untersuchen Sie die Lockdateien mehrmals: Wie weiss der neu hochgefahrene Server, dass die alten Locks nicht mehr gültig sind?
- Benennen Sie die Datei ~root/.director/credentials um. Probieren Sie dann ein paar Director-Kommandos wie z.B. `sfidirectory copy` und `sfidirector list` aus. Was stellen Sie fest? Wieso liefern diese Kommandos vermutlich trotzdem noch Resultate? Wie können Sie unterscheiden zwischen den beiden Varianten mit oder ohne credentials-Datei?
- Benutzen Sie einen Packet-Sniffer (z.B. tcpdump), um die Kommunikation zwischen Frontend-Interface und Oberfläche und zwischen Kommandozeilentools und dem Server zu belauschen.

3. Passwortsynchronisation mit Microsoft SFU / SSO

Typischerweise wird das LDAP-Repository des Director als die massgebende Quelle (authoritative) für alle Verwaltungsinformation betrachtet und der Director hilft dabei, diese Information auf sekundäre (oder Slave-)Systeme zu replizieren.

Anhand eines Anwendungsfalles bertrachten wir, wie wir davon abweichen können:

- Eine Linux-Umgebung wird per Director verwaltet
- Die „normale“ Benutzerumgebung basiert vollständig auf Microsoft Windows
- Ein Teil der Benutzer ist auch unter Linux bekannt und diese Benutzer sollen unter Linux ihr Windows-Passwort nutzen können

Die gewählte Lösung in diesem Fall basiert auf den Microsoft Services For Unix. Teil dieser Software-Suite ist der Passwort-Synchronisationsdienst SSO (Single-Sign-On), der es erlaubt, Passwortänderungen auf Windows-Seite an andere Systeme weiterzumelden.

Funktionsschema SSO

Auf allen Windows-Rechnern, auf denen Passworte geändert werden können, muss der Dienst *Passwort-Synchronisation* des SFU installiert sein. Jedes Mal wenn ein Benutzerpasswort geändert wird, wird dieser Dienst dies mitbekommen und das neue Passwort an die in der Konfiguration aufgeführten Rechner schicken.

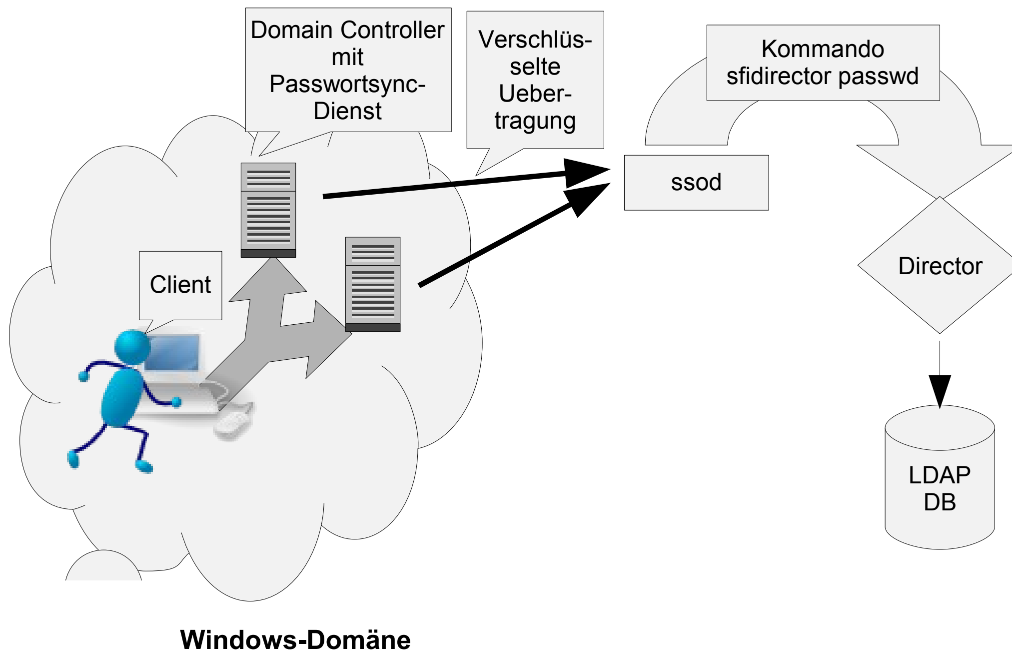
In einer Domänenumgebung – egal ob es sich um eine Active-Directory oder um eine alte NT4-Domäne handelt – muss der Passwortsynchronisationsdienst auf allen Domain Controllern installiert sein.

Auf dem Zielrechner muss ebenfalls der Passwortsynchronisationsdienst installiert sein. Eine geringfügig modifizierte Version unter Linux übermittelt die Passwortänderung durch Aufruf des Kommandos `sfidirector passwd` an den Director-Server, das das Passwort schliesslich über den Director-Server in der LDAP-DB ändert. Falls konfiguriert sorgt ein Auto-Update dafür, dass der Director das neue Passwort auch an weitere Subsysteme weitergibt.

Die Uebertragung zwischen den beiden Instanzen des Passwortsynchronisationsdienstes erfolgt dabei verschlüssselt.

Unter Linux wird die Passwortsynchronisation durch den Daemon *ssod* übernommen. Die Konfiguration dieses Dienstes findet sich in `/etc/sso.conf`.

Die für den Einsatz mit dem Director veränderte Version des ssod kann Passwörter per passwd-Kommando des Directors (Option USE_DIRECTOR in sso.conf) setzen.



Windows-Domäne

4. Director-Konfiguration

Neben der Haupt-Konfigurationsdatei /etc/sfidirector/sfidirector.conf erlauben eine Reihe weiterer Konfigurationsdateien die Anpassung des Directors und seines Verhaltens an eine Systemumgebung.

Hauptkonfiguration sfidirector.conf

Die Konfigurationsdatei /etc/sfidirector/sfidirector.conf stellt die Haupkonfiguration des Directors dar. Hier wird u.a. definiert, wie auf die Verwaltungsdatenbank zugegriffen werden soll, wo sich andere Konfigurationsinformation findet und eine Reihe von Optionen erlauben es, das Verhalten der einzelnen Komponenten zu beeinflussen. Die einzelnen Konfigurationsoptionen sind in der Datei selber dokumentiert.

Events und Aktionen: objevents

Die Datei /etc/sfidirector/objevents legt fest, welche Events für spezifische Verwaltungsobjekte verfügbar sind und zu welchen Aktionen ein Event führen soll. Sitespezifische Einträge sollten in der Datein /etc/sfidirector/objeventsSite gemacht werden, die Datei /etc/sfidirector/objevents wird typischerweise beim Installieren einer neuen Directorversion überschrieben.

Die Datei liegt im LDIF-Format vor und enthält Objekte der Klasse directorObjectEvent und directorTask.

Event: directorObjectEvent

Ein directorObjectEvent-Objekt definiert ein Event für ein bestimmtes Verwaltungsobjekt. Es muss mindestens einen Namen (cn) tragen, definieren für welche Klassen von Objekten es zuständig ist (sfieventforObjects) und den Kurznamen des Events definieren (sfieventType). Mehrere directorObjectEvent-Objekte dürfen (sollen) den gleichen sfieventType verwenden. Es wird empfohlen, eine Beschreibung des Events im Attribut description und/oder sfimenuName zu hinterlegen – diese werden durch Oberfläche und Kommandozeilentools angezeigt. Wird ein Attribut sfijobname gesetzt, dann tragen Jobs, die auf Grund dieses Events generiert werden diesen Wert als Namen.

SfieventType mit spezieller Bedeutung sind *create*, *alter* und *remove*. Events von diesem Typ werden durch die Oberfläche beim Erstellen, Aendern oder Löschen eines Objektes automatisch ausgeführt.

Das wichtigste Attribut hingegen dürfte das Attribut *sfieventexecutes* sein. Dieses definiert nämlich, welche Aktion(en) beim Auslösen eines Events ausgeführt werden sollen. Einige Aktionen erlauben es, via Attribut *sfioption* näher zu spezifizieren, wie die Aktion ablaufen soll.

Die Werte von sfieventexecutes bezeichnen entweder

- ein directorTask-Objekt mit weiteren Anweisungen
- eine hartcodierte Aktion

Aktionen: directorTask

Ein Objekt der Klasse directorTask definiert eine oder mehrere Aktionen. Das Objekt kann über seinen Namen in anderen directorTask-Objekten oder in directorObjectEvent-Objekten über das Attribut sfieventexecutes referenziert werden.

Ein directorTask-Objekt muss mindestens einen eindeutigen Namen (cn) tragen und macht wenig Sinn ohne ein sfieventexecutes-Attribut. Sfijobname und description tauchen in generierten Jobs als Text auf. Einige sfieventexecutes-Aktionen erlauben es, per Attribut sfioption nähere Angaben zu ihrem Verhalten anzugeben.

Ein Beispiel

Wir stellen uns zur Aufgabe, ein Event zu definieren, das es uns erlaubt, auf einem oder mehreren Zielrechnern den Apache-Webserver neu zu starten.

Als erstes definieren wir ein Event *apacherestart*, das etwa so aussehen könnte:

```
dn: cn=Apache-Restart
cn: Apache-Restart
objectclass: directorObjectEvent
sfieventforObjects: directorHost
sfieventType: apacherestart
sfimenuName: Restart Apache Web Server
```

Das Event soll auf Rechner (directorHost) angewendet werden können. Im Menü soll es unter dem Eintrag „Restart Apache Web Server" erscheinen. Nun müssen wir noch definieren, was für eine Aktion das Event auslösen soll:

```
sfieventexecutes: TApacheRestart
```

Wir entscheiden uns dafür, ein directorTask-Objekt für die eigentliche Aktion zu verwenden und nennen es TApacheRestart.

Als nächstes müssen wir dieses definieren:

```
dn: cn=TApacheRestart
cn: TApacheRestart
objectclass: directorTask
description: Restart Apache Web Server
sfijobname: ApacheRestart
```

Das ist der administrative Teil: Die Aktion kann nun über den Namen TApacheRestart referenziert werden und Jobs werden den Namen ApacheRestart und die Beschreibung Restart Apache Web Server tragen. Nun zur eigentlichen Funktion:

```
sfieventexecutes: Shell
```

Wir werden ganz einfach ein Shell-Kommando auf dem Zielrechner ausführen lassen. Bei der Aktion *Shell* handelt es sich um eine hartcodierte Aktion. Wir müssen dieser allerdings noch mitteilen, auf welchem Rechner sie welches Kommando ausführen soll:

```
sfioption: Host=$cn
sfioption: Command=/etc/init.d/apache restart
```

Als Zielrechner (Host=) soll das Attribut cn des Objektes, auf das sich das Event be-

zieht (also den Namen des Zielrechners) verwendet werden. Auf diesem Zielrechner soll das Kommando `/etc/init.d/apache restart` ausgeführt werden.

Wir können das neu definierte Event nun so austesten:

```
sfidirector -v event apacherestart Hosts:server1
```

vorausgesetzt natürlich, dass server1 ein definierter Rechner ist.

Hartcodierte Aktionen

Leider existiert zur Zeit keine Dokumentation der vorhandenen vordefinierten Aktionen. Die wichtigste Aktion, um auf einfachem Weg zu simplen einfachen Aktionen zu kommen, dürfte aber *Shell* sein, zu der es einige Beispiele im Standard-objevents gibt.

Uebungen

- Benutzen Sie das Kommando listevents, um herauszufinden, welche Events auf ein bestimmtes Objekt definiert sind. Suchen Sie sich eins aus, auf das Sie neugierig sind und untersuchen Sie die zugehörige Definition in der objevents-Datei. Woran erkennen Sie hartcodierte Aktionen? Sind die Einträge trotz des eher kryptischen LDIF-Formats verständlich?

Konfigurationsverteilung

Die per Konfigurationsverteilung verwaltbaren Konfigurationsdateien werden in der Datei /usr/share/sfidirector/configfiles definiert. Eigene Erweiterungen sollten jedoch in /etc/sfidirector/configfilesSite vorgenommen werden, da /usr/sharesfidirector/configfiles bei einem Director-Upgrade überschrieben werden wird.

Die configfiles-Datei ist im LDIF-Format gehalten. Jedes Objekt in dieser Datei entspricht einer Konfigurationsdatei, die die Konfigurationsverteilung verwalten können soll. Jeder Eintrag muss mindestens einen eindeutigen Namen (*cn*) tragen, eine kurze Beschreibung im Attribut *description* wird empfohlen. Das Attribut *sfipackageid* beschreibt, in welchen Konfigpaketen dieser Eintrag enthalten ist.

Um Konfigurationseinstellungen auf die Datei anwenden zu können, muss der Director wissen, in welchem Format die Konfigurationsdatei vorliegt. Dazu dient das Attribut *sfipackagefileformat*.

Den Pfad der Datei entnimmt der Director dem Attribut *sfipackagefilealternative*. Dieses darf auch mehrere Werte enthalten. Bei Ein- und Auspacken wird jedes Mal derjenige Pfad verwendet, unter dem sich auf dem Quell- oder Zielsystem bereits eine Datei befindet.

Nach dem Aufdatieren einer Konfigurationsdatei mag es notwendig sein, einen Dienst neu zu starten. Welche Dienste neu gestartet werden sollen definiert das Attribut *sfirestartservice*. Natürlich soll ein Dienst nur dann neu gestartet werden, wenn er bereits lief. Das Attribut *sfipackagerestartifprocs* definiert, welche Prozesse der Director auf dem System laufend vorfinden muss, damit der Dienst neu gestartet wird. Falls mehrere Prozesse angegeben sind, wird neu gestartet, falls einer davon gefunden wird.

Falls die Implementierung des angegebenen Dateiformats Konfigurationseinstellungen unterstützt, so definiert das Attribut *sfipackageconfigprefix* den Präfix, den alle Einstellungen für diese Konfigurationsdatei tragen müssen.

Dateiformate

Leider sind die implementierten Dateiformate noch nicht dokumentiert. Allerdings hilft das Directorkommando *listconfig* etwas weiter: Es zeigt nämlich nicht nur die konfigurierten Konfigdateien an sondern auch die zur Verfügung stehenden Dateiformate. Alle Einträge mit der Anmerkung „No packages = file format handler only“ unter Packages stellen Dateiformate dar.

Einige ausgewählte Formate sind:

- Plain: Datei ohne Director-bekanntes Format, keine Konfigurationseinstellungen unterstützt.
- SimpleKey: Datei mit Einträgen der Form

  ```
  Schlüssel     Wert
  ```

  zwischen Schlüssel und Wert befinden sich Leerzeichen oder Tabs.
- SimpleKeyEqual: Wie SimpleKey, aber Schlüssel und Wert sind durch ein „=“ getrennt.
- SimpleKeyColon: Wie SimpleKey, aber Schlüssel und Wert sind durch einen „:“ getrennt.
- SimpleKeyEqualSpaceTolerant: Wie SimpleKeyEqual, aber vor und nach dem „=“ dürfen sich ausserdem Leerzeichen oder Tabs befinden.

Neben diesen generischen Dateiformaten sind allerdings auch einige spezielle Formate nicht durch einen configfiles-Eintrag sondern hartcodiert implementiert (das war für eine Weile die einzige Möglichkeit, um Konfigdateien in die Konfigurationsverteilung aufzunehmen). Die dazugehörigen Dateiformattreiber können auch in Einträgen in der configfiles-Datei verwendet werden.

Konfigurationsverteilung testen

Selbstvertändlich kann nach Eintrag einer Konfigurationsdatei in configfilesSite direkt eine Konfigurationsverteilung versucht werden. Eine einfachere Möglichkeit zum Testen stellen allerdings die *saveconfig*- und *restoreconfig*-Kommandos dar. Mittels

```
sfidirector savconfig -f /tmp/test package1 package2
```

lassen sich die in den Paketen package1 und package2 enthaltenen Konfigdateien in eine Datei einpacken. Das verwendete Format der Ausgabedatei ist zwar ein Director-spezifisches, lässt sich aber problemlos mit einem Texteditor ansehen, um zu prüfen, ob die erwarteten Dateien tatsächlich enthalten sind.

Das so erzeugte Archiv lässt sich auf einem Zielsystem mit

```
sfidirector restoreconfig -f /tmp/test
```

wieder auspacken und die Konfigurationsdateien werden „wie bei der richtigen Konfigurationsverteilung“ installiert, d.h. allenfalls auch Dienste neu gestartet. Um Konfigurationseinstellungen zu testen kann ausserdem ein Objekt angegeben werden, dessen Konfigurationseinstellungen verwendet werden sollen:

```
sfidirector restoreconfig -f /tmp/test -c Hosts:rechner1
```

wird die Konfigurationseinstellungen des Rechnerobjekts rechner1 verwenden um die Konfigdateien zu entpacken.

Die -c-Option verlangt, dass das restoreconfig-Kommando Zugang zur entsprechenden Datenbank hat, wird also in der Regel nur auf dem Adminserver selber ohne weiteres funktionieren.

Schema und Editorschemata

Das verwendete LDAP-Schema befindet sich unter /usr/share/sfidirector/etc/director.schema, in einem Format für vom Netscape Directory Server abstammende LDAP-Server (z.B. SunOne Directory Server, RedHat Director Server) auch unter .../director.schema.ldif.

Es sollte vermieden werden, direkt an diesem Schema Modifikationen vorzunehmen, da diese beim nächsten Director-Upgrade überschrieben werden. Statt dessen sollten eigene Definitionen in eine separate Datei – die natürlich ebenfalls in slapd.conf referenziert werden muss – gelegt werden. Dies bedeutet allerdings, dass bestehende Objektklassen nicht erweitert werden sollen, sondern statt dessen besser eine (nicht-*Structural*) optionale eigene Objektklasse definiert werden sollte.

Der Director selbst kümmert sich nicht um den Inhalt von director.schema. Zu einem Teil verlässt er sich darauf, dass Objekte im LDAP-Repository in einem standardisierten Format vorliegen (z.B. p*osixAccount* für Benutzer). Anzeige- und Editierfunktionen hingegen nutzen die Editorschema-Dateien unter /usr/share/sfidirector/etc/schema. Welche Schema-Datei dabei welchen LDAP-Objekten entspricht definiert die Datei /etc/sfidirector/objRegistry.ldif. Auch hier existiert wieder eine Datei mit der Ergänzung Site, also /etc/sfidirector/objRegistrySite.ldif, die für eigene Erweiterungen vorgesehen ist.

ObjRegistry-Einträge

Wie zu vermuten war ist auch die objRegistry-Datei im LDIF-Format gehalten. Jeder Eintrag entspricht einem Objekt-Typ, der mit dem Director erzeugt oder bearbeitet werden kann.

Jeder Eintrag trägt einen eindeutigen Namen (*cn*). Es ist explizit erlaubt, in objRegistrySite einen Namen zu verwenden, der bereits in objRegistry benutzt wurde – in diesem Fall wird der Eintrag aus objRegistry unwirksam. Neben dem Namen sollte eine Beschreibung (*description*) und ein Anzeigename (*sfidisplayName*) definiert werden, die in Menüs oder auf der Kommandozeile als Objektbeschreibung erscheinen können. Der *sfiobjContext* gibt der Director-Oberfläche einen Hinweis, an welcher Stelle Objekte dieses Typs Sinn machen könnten. Gültige Werte sind zur Zeit: Application, Classes, DSRules, People, Services. Das Attribut *sfifeature* (sofern definiert) ist bei Objekttypen gesetzt, die nur dann zur Verfügung stehen

sollen, wenn bestimmte „Features“ in den User-Preferences von BigClerk eingeschaltet sind. Das Attribut *sfiicon* definiert, welches Icon in der Oberfläche für Objekte dieses Typs zu verwenden ist.

Per *sfioption*-Eintrag wird festgelegt, ob Objekte von diesem Typ bearbeitet (*editable*) und/oder erstellt (*createable*) werden können. Der vorwiegende Teil der Objekte wird als editable und createable eingetragen werden.

Die beiden Attribute *sfiregmatchClass* und *sficontainsClass* legen fest, mit welchen LDAP-Objektklassen ein Objekt-Typ in Beziehung steht: Für jede Klasse eines Objektes werden alle Objekt-Typen gesucht, die einen passenden Eintrag in sfiregmatchClass tragen. Ergibt dieser Vorgang mehr als einen möglichen Objekttyp, so wird sficontainsClass als Entscheidungskriterium benutzt: Der Objekttyp mit den meisten passenden sficontainsClass-Einträgen macht das Rennen.

Das Attribut *sfiregschema* schliesslich legt fest, welches Editorschema aus /usr/share/sfidirector/etc/schema verwendet werden soll.

Beispiel:

```
dn: cn=host
cn: host
sfidisplayName: Host
sfioption: editable
sfioption: createable
sfiregschema: Host
sfiicon: host.gif
sfiregmatchClass: directorHost
sfiregmatchClass: ipHost
sfiobjcontext: Hosts
objectclass: directorObjectRegistry
```

definiert einen Objekttypen *host*, Objekte von diesem Typ lassen sich *erzeugen* und *editieren* und machen im Zusammenhang (Context) mit *Hosts* Sinn. Als Icon wird *host.gif* verwendet. LDAP-Objekte der Klassen *ipHost* und *directorHost* entsprechen diesem Objekttyp. Als Editorschema zum Bearbeiten eines solchen Objektes wird /usr/share/sfidirector/etc/schema/*Host* verwendet.

Editorschema

Jedes Editorschema entspricht einer Schema-Datei unter /usr/share/sfidirector/etc/schema. Eine Schema-Definition kann weitere Schema-Dateien per *import*-Anweisung referenzieren.

Am Beginn der meisten Schema-Dateien steht eine *interface*-Abschnitt, der mit dem Schlüsselwort *interface* beginnt und mit dem Wort *end* abgeschlossen wird. Das Interface definiert, wie Editorobjekte mit dem entsprechenden LDAP-Objekt zusammenhängen. Jeder Eintrag entspricht der Form

```
Importer   argument1=wert1 ... argumentN=wertN
```

Jedes LDAP-Objekt wird – bevor es durch den Director bearbeitet werden kann – importiert. Beim Import können Attribute des LDAP-Objektes in editorinterne Repräsentationen umgewandelt werden. Wird der Editor beendet, so werden die inter-

nen Attribute wieder *exportiert*. Dabei wird das Objekt auch um berechnete Attribute wie z.B. die DN oder das objectclass-Attribut ergänzt. Editorattribute, die mit einem „_“ beginnen, werden sofern im Interface-Abschnitt nicht anders definiert beim Exportieren verworfen, alle anderen unverändert übernommen.

Definierte Importer sind z.B.:

- DN: dient zum Berechnen der DN, unter der das Objekt abgelegt wird. DN kennt zwei Argumente: *dn* und *prefix*. Das Argument dn enthält eine Strichpunktseparierte Liste von Attributen, die als Schlüssel dienen sollen. Für jeden Eintrag in dn muss ein entsprechender Eintrag im prefix-Argument (Strichpunktsepariert) vorhanden sein, der definiert, wie das Attribut LDAP-seitig heisst.
- Expand mit den Argumenten *field* und *expr* berechnet den Inhalt eines Attributes (field) beim *Exportieren* aus dem Ausdruck in expr.
- List spaltet eine Werteliste beim Importieren auf mehrere interne Attribute auf, beim Exportieren werden diese wieder zu einer Werteliste zusammengesetzt. Das Argument *external* enthält den Namen des LDAP-Attributes, das aufgespaltet werden soll. Das Argument *internal* enthält eine kommaseparierte Liste von internen Attributnamen. Der erste Wert des externen Attributes erhält beim Importieren den ersten Wert, usw. Vor dem letzten Eintrag in internal darf ein „*“ stehen – in diesem Falle werden alle übriggebliebenen Werte diesem Attribut zugewiesen.
- Migrate dient dazu, beim Importieren „alte“ Attribute durch ihre neue Repräsentation zu ersetzen. Das Argument *deprecated* muss dabei den Namen eines veralteten Attributs enthalten. Dieses Attribut wird beim Importieren umbenannt in das im Argument *new* benannte neue Attribut.
- ObjectClass wird verwendet, um das objectClass-Attribut automatisch zu bestimmen. Das einzige Argument *classes* enthält eine kommaseparierte Liste von Objektklassen, die ein Objekt von diesem Typ annehmen kann. Jede Objektklasse ist gefolgt von einem Doppelpunkt. Hinter diesem Doppelpunkt folgt ein Attributname. Ist dieses Attribut im Objekt gesetzt, so wird im objectClass-Attribut die genannte Objektklassen aufgenommen. Viele Objektklassen benötigen mehrere obligatorische Attribute – in diesem Fall darf hinter dem Doppelpunkt eine Liste von Attributnamen verknüpft mit dem Zeichen „&“ stehen.
- PasswordHash analysiert beim Importieren Passwortattribute (userpassword, lmpassword, ntpassword) im LDAP-Objekt und setzt je nach Typ der vorhandenen Passworthashes interne Attribute. Im Argument *prefix* wird ein Präfix festgelegt, der vor alle internen Namen gesetzt wird. Ist beispielsweise im LDAP-Objekt userpassword auf „{crypt}abcd“ gesetzt und der prefix „_pwd_“, so wird ein internes Attribut _pwd_crypt mit Wert „on“ gesetzt. Falls beim Exportieren ein Attribut *_password* existiert, dann werden alle Hashes, für die ein internes Attribut mit dem Wert „on“ existiert, aus diesem Klartextpasswort berechnet.

Attributnamen im Editorschema sind im Gegensatz zu LDAP-Attributnamen case-sensitive, müssen also innerhalb des Editorschemas immer gleich geschrieben sein.

Der Rest der Schema-Datei beschreibt die Editorattribute, resp. die Art und Weise,

wie der Editor mit Ihnen umgehen soll. Die Attribute werden prinzipiell in „Panel" gruppiert. Die graphische Repräsentation eines Panels ist dem Editor überlassen – BigClerk verwendet dazu per default Register.

Für jedes Attribut muss eine Zeile der Form

```
attributname    FeldTyp    argument1=wert1 argument2=wert2
```

im Schema stehen.

Einige Argumente finden für alle Feldtypen Verwendung, andere sind vom Feldtyp abhängig. Zu ersteren zählt z.B. *label* (Anzeigenamen des Attributs), *editline* (mehrere Attribute mit derselben editline werden bevorzugt nebeneinander dargestellt), feature (dient zur Angabe, bei Einschalten welcher Features in den User-Preferences das Feld überhaupt erscheint, s. auch S. 18), *template* (ein Ausdruck, der den Defaultwert des Feldes beschreibt, identisch mit der Vorlagenfunktion in Benutzerklassen) oder das verwandte *default* (Defaultwert).

Auch hier gilt leider wieder, dass die verfügbaren FeldTypen nicht dokumentiert sind, es finden sich jedoch viele Beispiele in den vorhandenen Schemata. Einige ausgewählte Basis-Typen:

- Text: Ein einfaches Textfeld
- List: Ein Listenfeld. Im Argument *of* muss der Feldtyp der einzelnen Listeneinträge festgelegt werden.
- CheckBox: Eine Checkbox, Argumente *true* und *false* legen die Attributwerte fest, die einer eingeschalteten oder ausgeschalteten Checkbox entsprechen.
- Selection: Bietet eine Auswahl von Werten an. Das Argument *choice* enthält eine strichpunktseparierte Auswahlliste, das Argument *values* enthält eine Liste der Attributwerte, die bei Auswahl der entsprechenden Option aus choice im Attribu erscheinen soll.

Bis zur Verbreitung von BigClerk als Oberfläche war es Pflicht, dass jeder verwendete Feldtyp hartcodiert (d.h. mit einer entsprechenden Java-Klasse im Director-Code implementiert) war. Seit einiger Zeit ist allerdings jetzt ein Konstrukt wie

```
attributname    SomeCustomType  master=Text label=whatever
```

erlaubt. In diesem Fall braucht der Director den Feldtypen SomeCustomType selbst nicht zu kennen. Alle Editoren, die SomeCustomType nicht kennen, werden statt dessen einfach ein Feld vom Typ Text anzeigen.

Uebungen

- Erweitern Sie das Schema für Benutzerobjekte um umgebungsspezifische Attribute wie z.B. Abteilung, Büronummer oder Kostenstelle und machen Sie diese Attribute editierbar.
- Sie möchten spezielle Folder mit einer Funktion versehen. Definieren Sie z.B. ein spezielles Objekt „Abteilung". Dieses soll in der Directoroberfläche als Unterverzeichnis erscheinen (muss also die Klasse *organizationalUnit* tragen), aber trotz-

dem soll das Objekt gesondert in Menüs erscheinen, ein eigenes Icon tragen und einige zusätzliche Attribute kennen.

- Definieren Sie ein Event für das soeben konfigurierte Abteilungs-Objekt: Beim Erstellen eines Abteilungsobjektes soll auf Ihrem Fileserver automatisch ein Unterverzeichnis zum Datenaustausch zwischen Mitgliedern dieser Abteilung erstellt werden.

5. Einführung in den Director-Quelltext

Build-Prozedur

Der Director wird im Unterverzeichnis *director* der Quellen gebildet. Verwendet wird eine klassische Autoconf-basierte Prozedur, d.h. der Director lässt sich im Wesentlichen mit

```
./configure
make
make install
```

kompilieren und installieren. Vorausgesetzt wird hauptsächlich GCC ab Version 3.2. Die verfügbaren offiziellen Binaries sind allerdings (zu diesem Zeitpunkt) mit einer gepatchten Version von GCC 4.0.1 erstellt, da leider in anderen GCC-Versionen unterschiedliche Fehler existieren, die die Stabilität und teilweise die Funtionen des Directors beeinträchtigen können. Configure kennt einige Optionen, über deren Einsatz sich nachzudenken lohnt, die Binärpakete werden zur Zeit mit

```
./configure --disable-java --enable-FHS --enable-natural –enable-infrappliance
```

gebildet.

Debian- und RPM-Pakete lassen sich auch direkt aus dem Sourcetree bilden mit

```
./configure
make rpm
make deb
```

(als root oder mit fakeroot). Voraussetzung ist, dass die jeweiligen Paket-Toolchains für RPM oder DEB vorhanden sind. Empfohlen ist in jedem Fall der Einsatz des Kommandos *fakeroot:*

```
./configure
fakeroot make rpm
fakeroot make deb
```

Es ist möglich, eine Java-Bytecode-Version des Directors zu bilden. Dazu wird der GCC nicht benötigt, sondern ein beliebiger Java-Compiler. Wegen diverser JVM-spezifischer Probleme wurde entschieden, die GCC-kompilierte Binary-Version zu bevorzugen.

Modularität mit monolithischem Ergebnis

Als Ergebnis der Build-Prozedur erhält man im Wesentlichen ein einziges, etwas gross geratenes Binary. Bei der Festlegung der Architektur wurde dieses Modell gewählt, um eine möglichst einfache Installation zu ermöglichen: Im Extremfall reicht das Installieren des „sfidirector"-Binaries etwa aus, um ein System unter Director-Verwaltung zu stellen.

Trotz dieses „fetten" Ansatzes ist der Code selber selbstverständlich modular aufgebaut.

Erste Uebersicht

Prinzipiell ist der Director-Quelltext in einige Java-Packages mit entsprechenden Funktionen aufgeteilt:

Package	***Zweck***
util	Klassen, die quer über alle Packages hinweg genutzt werden.
Cli	Implementation aller Shell-Kommandos, pro Shell-Kommando existiert eine CLI-Klasse
System	Systemnahe Funktionalität. Das Systempackage enthält auch einige in C geschriebene„native"-Bibliotheken und das Hauptprogramm (ebenfalls C) für das GCC-kompilierte Binary.
Retriever	Enthält Klassen zum Anfordern und Schreiben von Dateien (lokal und remote).
Jobs	Job Management. Serializieren und De-Serializieren von Jobs, Ablaufsteuerung, Job-Monitor
Repository	Zugriff auf Datenbanken (LDAP und andere) und Directory Services, Datenkonversionen
Repository.config	Zugriff auf logische Teile der Verwaltungsdatenbank (z.B. directory:Hosts) und Vererbung von Personen- und Rechnerklassen
Repository.transform	Implementation der in DSRules verwendeten Transformationsfunktionen
Schedule	Vom Queueworker verwendete Scheduler zur Queueverwaltung
Invoker	Lokale und „remote" Ausführung von Jobs (resp. Kommunikationsprotokolle dafür)
Executor	Ausführung von Job-Objekten
Gui	Urspr. Graphische Oberfläche, jetzt nur noch einige Klassen für Editorfunktion
Gui.fields	Editor-Feldtyp-Implementationen
Gui.importers	Editor-Importer-Klassen (Interface-Section in Editorschema)
Com.graeff.dbedit	Editorimplementation
Com.graeff.dbedit.fields	Editor-Feldtyp-Implementationen
Com.graeff.dbedit.importers	Editor-Importer-Klassen
Com.graeff.pwencode	Passworthash-Generatoren
Application	Die eigentliche Anwendungslogik, z.B. Funktionen wie

Package	*Zweck*
	Answendungsverteilung, Frontend-Interface, Directory-Service-Updates, etc.
Application.scripttask	Diese Klassen implementieren hartcodierte „Tasks‟ (s. sfieventexecutes in der objevents-Konfiguration). Die meisten ScriptTask-Klassen fungieren auch als executor (s.u.)
Application.executor	Die auf Agentseite über Jobs ausführbaren Funktionen.
Application.config	Konfigurationsverwaltung, Treiber für Dateiformate
Application.objevents	Eventmechanismus

Neben diesen Paketen existiert noch ein Minmum an zusätzlichem Code (Angabe: Verzeichnis):

- gcj3: Klassen, die nur bei Verwendung eines GCC mit Versionen 3.x verwendet werden (typischerweise Workarounds für GCC-Bugs).
- gcj: Klassen, die nur beim Bilden der Binärversion mit GCC verwendet werden.
- Java: Re-Implementationen von Java-Bibliotheksklassen (typischerweise ersetzen diese fehlerhafte Klassen aus der Standardbibliotheke).
- ../3rd: Einige Bibliotheken wie z.B. PostgreSQL und MySQL-Clients können in die Binärversion eingebunden werden, falls die passende .jar-Datei in diesem Verzeichnis existiert.
- ../jldap: Die Novell'sche (Freeware) Bibliothek für LDAP-Zugriffe, die mittlerweile in Bevorzugung zu SUNs JNDI verwendet wird.

Setup und Tools

Die beiden Klassen sfi.director.util.Setup und sfi.director.util.Tools werden nahezu quer über den ganzen Quelltext hinweg verwendet.

Die Tools-Klasse enthält eine Sammlung von Prozeduren und Funktionen von allgemeinem Interesse wie z.B. kontrolliertes Ausführen von Shellkommandos, Logging, Debugging, etc.

Die Setup-Klasse hingegen dient dazu, Konfigurations- oder Siteabhängige Informationen zur Verfügung zu stellen. Sie liest die sfidirector.conf-Datei ein, baut die Datenbankverbindungen auf, initialisiert Treiberstrukturen, etc. In vielen Packages wird ein Treibermodell verwendet (z.B. repository), bei dem einzelne Treiberklassen eine Funktion (z.B. Zugriff auf einen DB-Typ) repräsentieren. Diese Treiberklassen müssen zentral registriert werden. Den Verwendern der Treiber ist nur die Registrationsklasse bekannt, die anhand eines Namens, einer URL oder eines anderen Bezeichners die passende Treiberklasse findet.

Einfache Beispiele

Dateiformat für die Konfigurationsverwaltung

Im Kapitel Konfigurationsverteilung auf Seite 16 haben wir gesehen, wie durch einen Eintrag in der Konfigurationsdatei configfiles weitere Dateien unter die Verwaltung des Director gestellt werden können. Dabei ist die Frage offen geblieben, wie neue Dateiformate zu implementieren sind, und welche Dateiformate denn nun genau existieren.

Für jedes Dateiformat existiert im Verzeichnis application/config eine von *ConfigFileHandler* abgeleitete Klasse. Diese Klasse bietet eine ganze Reihe von Möglichkeiten, aber auch ein vernünftiges Defaultverhalten für die meisten Dateiformattreiber.

Wer „lediglich“ ein generisches Dateiformat implementieren will, das in configfiles verwenden werden soll, der orientiert sich am besten am Beispiel *SimpleKeyHandler*. Nach diesem Beispiel sollten mindestens folgendes implementiert werden:

- Konstruktoren für () und (String, String [], String[])
- Eine Methode processLine(RestoreConfigEnvironment, String) - das Defaultverhalten von ConfigFileHandler ist es, eine Datei zeilenweise zu verarbeiten. Für jede gelesene Zeile wird dabei processLine() aufgerufen, in dessen Verantwortung es liegt, die Zeile gegebenenfalls umzuformen und in die Zieldatei zu schreiben. Wer die Schreiberei nicht selber erledigen will, ruft die processLine()-Methode der Superklasse auf, die genau dies erledigt.
- Eine Method emit(String, String) - diese wird unter Angabe eines Variablennamens und des zugehörigen Wertes aufgerufen. Emit() kann aus processLine() aufgerufen werden. Für alle Konfigurationseinstellungen, die am Ende der Datei noch nicht emit()-et wurden, wird ConfigFileHandler emit() vor dem Schliessen der Datei aufrufen.

Ist unsere Klasse geschrieben, muss noch irgendwo festgehalten werden, dass sie jetzt existiert: Dies geschieht durch einen Eintrag in der Liste *handlers* der Klasse *ConfigEngine*.

Nun muss der Director neu kompiliert werden. Welche Klassen zu kompilieren sind, wird durch das *configure*-Skript herausgefunden. Nach Hinzufügen oder Löschen einer Klasse reicht es also nicht aus, ein *make* durchzuführen, sondern es muss ein configure mit anschliessendem make durchgeführt werden.

Das resultierende Binary liegt nach einem make im Verzeichnis build und kann von dort ausgeführt werden:

```
build/sfidirector.static listconfig
```

In der Ausgabe müsste jetzt unser neues Dateiformat erscheinen.

> *Je nach configure-Optionen kann das Binary anders heissen: sfidirector.dynamic, sfidirector.static, sfidirector.extrastatic, sfidirector.mostlystatic ...*