Der Director

Bedienung des Directors

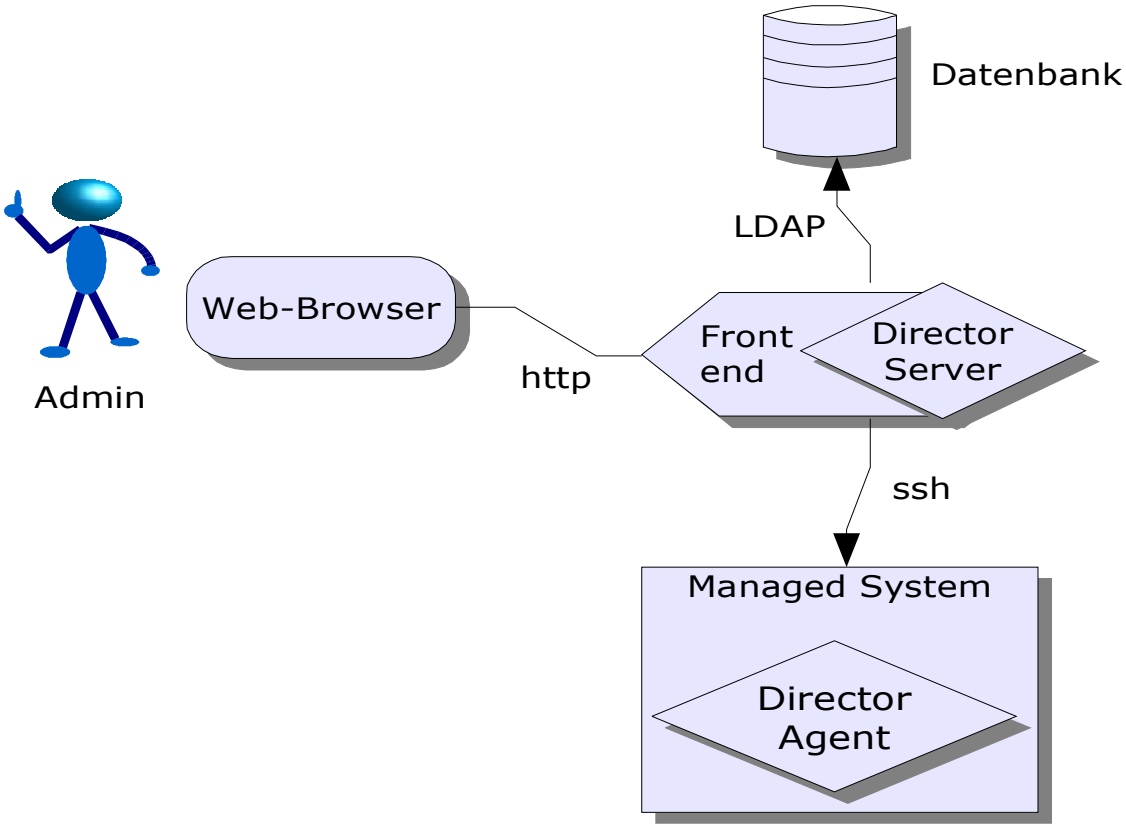
Phinex Informatik AG
Thomas Aeby

Inhalt

Was ist der Director

Der Director ist ein *Systemverwaltungswerkzeug* mit Funktionalität in den Bereichen

- Benutzerverwaltung
- Software-Verteilung
- Konfigurations-Management
- Namensdienste (LDAP, NIS, DNS, etc.)



Die dabei verwaltete Information über die Systemumgebung wird in einem *Repository*, typischerweise einer per LDAP zugreifbaren Datenbank, abgelegt. Das dabei verwendete *Schema* entspricht soweit möglich definierten Standards, andere LDAP-Clients und -Werkzeuge können also in der Regel auf dieselbe Datenbank zugreifen und finden sinnvolle Daten vor.

Auf einem zentralen Rechner läuft das *Director Frontend*. Seine Aufgabe ist es, eine Bedienungsoberfläche für den Systemverwalter zur Verfügung zu stellen. Zur Zeit existiert eine Kommandozeilen- und eine Web-Oberfläche. Ebenfalls auf einem zentralen Rechner, dem *Administrations-Server*, läuft der *Director Server*. Seine Aufgabe ist es, die durch den Systemverwalter über das Frontend angestossenen Aktionen zu verwalten, d.h. den jeweiligen Zielsystemen zur Ausführung zu übermitteln, Abhängigkeiten und Reihenfolgen zu beachten und gegebenenfalls wiederholte Versuche zu machen, falls ein System gerade nicht erreichbar ist. Dazu verwaltet er eine *Queue*, sein inoffizieller Name ist deshalb auch Queueworker.

Um eine Aktion auf einem Zielrechner auszuführen nimmt der Director Server Kontakt auf zum Zielsystem und startet dort einen *Director Agent*. Die Kontaktaufnahme erfolgt typischerweise per SSH, andere Protokolle sind aber möglich. Einmal gestartet empfängt der Agent Befehle und Daten vom Server, führt diese Befehle aus und übermittelt die Resultate. Diese Kommunikation läuft ausschliesslich über denselben Kanal,

der zur ursprünglichen Kontaktaufnahme verwendet wurde, d.h. normalerweise SSH.

Director Server, Director Agent und Director Frontend sind logisch unterschiedliche Komponenten, verwenden aber dasselbe *Binary*, respektive dieselben installierten Dateien. Auf allen beteiligten Systemen kann also eine identische Installation verwendet werden. Die Weboberfläche verlangt zusätzlich die Installation einer *Perl-basierten* Zusatzapplikation (BigClerk), die per CGI mit einem *Webserver*, typischerweise Apache, kommuniziert.

Der Director ist *kein* LDAP-Editor

Beim Director handelt es sich nicht um einen generischen LDAP-Editor. Der Unterschied besteht darin, dass zur Arbeit mit dem Director Konzepte wie Attribute, Objekt-Klassen, DNs, usw. nur unter der Haube verwendet werden und dem Benutzer der Bedienungsoberfläche nicht bekannt sein müssen. Ein Benutzer im Director wird angelegt, indem

- in der Bedienungsoberfläche die Funktion „New Person" (Web) oder „create Person" (Kommandozeile) eingesetzt wird und dieser min. Name, Vorname, Benutzerklasse und Passwort des Benutzers mitgeteilt wird

Im Hintergrund wird automatisch ein passendes LDAP-Objekt in der Datenbank angelegt, nötigenfalls in anderen Namensdiensten bekannt gemacht, ein Homeverzeichnis auf dem passenden Dateiserver eingerichtet, zusätzliche Aktionen wie Ergänzen von Automounter-Maps o.ä. ausgeführt.

Per LDAP-Editor wird das gleiche normalerweise wie folgt passieren:

- Erstellen eines Objektes mit Objektklasse „posixAccount" und „person"
- Einfüllen min. der Attribute sn, cn, uid, uidNumber, gidNumber, homeDirectory, userPassword, loginShell, gecos
- Erstellen des Homeverzeichnisses auf dem passenden Dateiserver
- Aufdatieren von anderen Namensdiensten, in denen benutzerabhängige Daten vorgehalten werden

Unter gewissen Umständen mag es Sinn machen, direkt an den in der LDAP-Datenbank vorgehaltenen Daten zu manipulieren. Dazu kann selbstverständlich ein generischer LDAP-Editor verwendet werden – sofern auf Einhaltung der Schemata geachtet wird, wird der Director problemlos auch dann mitspielen, wenn „seine" Datenbankobjekte durch ein anderes Werkzeug manipuliert werden.

Konfiguration

Nebst dem Inhalt des Repositories („LDAP-Datenbank") verwendet der Director in der Standard-Konfiguration mehrere Konfigurationsdateien. Bis auf die Hauptkonfigurationsdatei sfidirector.conf sollten im Normalfall keine Anpassungen notwendig sein. Zu beachten ist, dass viele Konfigurationsdateien bei einem Update des Directors durch neuere Versionen überschrieben werden. Einige Dateien referenzieren explizit eine Site-spezifische Zusatzdatei (erkennbar an der Zeile „Include" am Ende der Datei). In diesen Fällen werden Ergänzungen am besten in dieser Datei gemacht.

Pfade der Konfigurationsdateien

Datei	***Pfad***	***FHS-Pfad***	***Beschreibung***
sfidirector.conf	PRE/etc/sfidirector	/etc/sfidirector	Haupt-Konfiguration
objevents	PRE/etc/sfidirector	/etc/sfidirector	Objekt-Typen zugeordnete Events
objRegistry.ldif	PRE/etc/sfidirector	/etc/sfidirector	bekannte Objek-Typen
director.schema	PRE/share/sfidirector/etc	/usr/share/sfidirector/etc	Schema-Definition für LDAP-Server
director.schema.ldif	PRE/share/sfidirector/etc	/usr/share/sfidirector/etc	Schema-Definition im LDIF-Format
schema	PRE/share/sfidirector/etc	/usr/share/sfidirector/etc	Formular-Definitionen für den Director
configfiles	PRE/share/sfidirector/etc	/usr/share/sfidirector/etc	bekannte Konfigurationsdateien für Konfig-Verwaltung

Die Bedeutung der Konfigurationsdateien

sfidirector.conf – die Hauptkonfiguration

Die Hauptkonfigurationsdatei legt eine ganze Reihe von Optionen fest, u.a. wo der Director sein Repository findet, wo zusätzliche Konfiguratonsdateien zu finden sind, wie sich der Scheduler verhalten soll, etc.

Die möglichen Konfigurationseinstellungen sind im Kommentar der Datei beschrieben.

objRegistry und Schema-Verzeichnis – Objekte und ihre Attribute

In der Datei objRegistry.ldif steht pro Objekt-Typ, den der Director kennen soll, ein Eintrag. Dieser Eintrag enthält einerseits Informationen darüber, woran der Director Objekte von diesem Typ erkennt (per „objectclass"-Attribut), darüber, wie das Objekt in der Benutzeroberfläche erscheinen soll (Icon, Namen, Kurzbeschreibung, editierbar? kreierbar?) und eine Referenz auf eine Datei im Schema-Verzeichnis, die die Attribute des Objektes beschreibt.

Die Schema-Datei ist nicht zu verwechseln mit einem LDAP-Datenbank-Schema. Vielmehr erinnern die Schema-Dateien an Formular-Beschreibungen: Sie werden von der Benutzeroberfläche dazu benutzt, Inhalte des Objektes darzustellen.

objevents – Objekt-Typen zugeordnete Events

In der objevents-Datei wird für jeden Objekt-Typ definiert, welche Events auf diesen

anwendbar sind und welche Aktionen bei Auftreten eines Events ausgelöst werden sollen.

director.schema / director.schema.ldif – LDAP-Schema

Die Dateien director.schema und director.schema.ldif enthalten die Definition der vom Director verwendeten Objektklassen und Attribute in einem von LDAP-Servern lesbaren Format. Diese Dateien werden durch den Director direkt nicht verwendet.

Bedienung des Directors - Grundfunktionen

Unabhängig davon, ob per Weboberfläche oder Kommandozeile gearbeitet wird, finden ähnliche Bedienkonzepte ihre Anwendung. Aus diesem Grund wird in diesem Kapitel hauptsächlich auf die Kommandozeile eingegangen – wer diese gemeistert hat wird auch die Funktionsweise der Web-Oberfläche verstehen. Letztere sollte für denjenigen, der sich nicht in die Tiefen des Directors einarbeiten möchte für die alltäglichen Aufgaben intuitiv erschliessbar sein.

Grundlagen und Konzepte

Repository

Das Repository ist eine Datenbank, in der der Director seine Informationen über verwaltete Entitäten (z.B. Benutzer, Rechner, Anwendungen) ablegt. Normalerweise liegt das Repository in einer LDAP-Datenbank, in der Datei sfidirector.conf kann allerdings festgelegt werden, wo es liegt. Dies kann für einzelne Teile des Repository getrennt festgelegt werden: So wird standardmässig lediglich der „Startpunkt" des Repository (Einstellung Top:) festgelegt, Rechnerinformationen erwartet der Director dann unter einem „ou=Hosts"-Teilbaum der Datenbank (Hosts: directory:Top:ou=Hosts), Benutzer-Information unter „ou=People" (People: directory:Top:ou=People). Schon in der Standardkonfiguration wird hingegen die JobQueue in einem Spoolverzeichnis statt in der LDAP-Datenbank abgelegt (JobQueue, CmdQueue Einstellungen), da dieser Teil des Repositorys recht intensiv mit Schreiboperationen bombardiert wird und LDAP-Datenbanken in der Regel eher auf Leseoperationen optimiert sind.

Datenbanken

Vielerorts, besonders bei Kommandozeilenfunktionen, tauchen die Begriffe Datenbank und Directory auf. Der Begriff *Datenbank* meint dabei jeweils präziser eine Verbindung auf eine Datenquelle oder eine Datendestination. Dies kann sehr unterschiedliche und vielfältige Datenformate und Protokolle beinhalten, Beispiele für „Datenbanken" sind z.B.:

Datenbank-Bezeichner	***Beschreibung***
`ldap:server:ou=hosts,o=example`	Alle Objekte, die auf dem LDAP-Server „server" unter „ou=hosts,o=example" zu finden sind
`ldiffile:file:/tmp/test.ldif`	Objekte in der Datei /tmp/test.ldif im LDIF-Format
`passwdfile:file:/etc/passwd`	Objekte in der Datei /etc/passwd im klassischen Passwortdateiformat
`jdbc:mysql://localhost/egroupware:table=phpgw_accounts`	Die Tabelle phpgw_accounts in der MySQL-Datenbank egroupware auf dem Server localhost
`ldifdir:/var/lib/sfidirector/cmdqueue`	Director-eigene minimalst-"LDAP"-Datenbank: Jedes Objekt erhält eine eigene LDIF-Datei in einem Verzeichnis
`directory:Hosts:ou=Abteilung 1`	der Teilbaum unter „ou=Abteilung 1" des Host-Directorys (s.u.)

Ueber derartige „Datenbanken" stolpert der Director-Nutzer in der sfidirector.conf-

Konfigurationsdatei, beim Kommandozeilenbefehl „copy" oder bei der Definition von Regeln zur Synchronisation des Director-Repositories mit beliebigen Namensdiensten.

Directories

Ein *Directory* bezeichnet einen logischen Teil des Repositories. Typischerweise sind dies einzelne Objekttypen, z.B. enthält das Directory „Hosts" alle Rechnerobjekte im Repository. Dabei berücksichtigt wird einerseits die zugewiesene Datenbank – für das Hosts-Directory z.B. in der Einstellung „Hosts" in sfidirector.conf konfiguriert – als auch die Objektklassen der durchsuchten Objete. Das Directory Hosts enthält also alle Objekte mit Objektklasse „directorHost" oder „ipHost" aus dem Teilbaum, der für Rechnerinformationen verwendet wird.

Directories tauchen im Kommandozeileninterface bei allen Objektmanipulationsfunktionen auf, z.B. list, modify, create, delete. Directories können auch überall da referenziert werden, wo eine Datenbankreferenz (s.o. Datenbanken) erwartet wird. Die Syntax lautet dann:

```
directory:Hosts
```

oder

```
directory:Hosts:ou=Abteilung 1
```

Im zweiten Fall werden nur Rechner adressiert, die im Teilbaum „ou=Abteilung 1" des Hosts-Verzeichnis liegen.

Jedes Directory weiss, welches Attribut ein Objekt seines Typs identifiziert, dies ermöglicht, beim Zugriff auf Directories einzelne Objekte einfach zu adressieren, z.B. liefert

```
sfidirector list Hosts:solarix
```

den Eintrag zum Rechner mit Namen solarix.

Eine Liste der bekannten Directories liefert das Kommando

```
sfidirector listdirectories
```

Events und Aktionen

Eine Hauptfunktion neben der Verwaltung der Informationen im Repository ist das Ausführen von Aktionen abhängig vom Repository-Inhalt. So soll z.B. beim Anlegen eines Benutzers automatisch ein Home-Verzeichnis angelegt werden, nach dem Zuweisen von Applikationen soll eine Anwendungsverteilung stattfinden, wenn ein Rechner eingetragen wird, soll automatisch DNS nachgeführt werden, usw.

Dies ist die Aufgabe des *Events-und-Aktionen*-Mechanismus. Jedes Mal, wenn ein Objekt im Repository manipuliert wird, wird ein passendes Event ausgelöst, beim Erstellen eines Objektes ein create-, beim Modifizieren eines Objektes ein alter- und beim Löschen eines Objektes ein remove-Event. Dazu lösen Funktionen wie die Applikationsverteilung eigene Events aus. Beim Abarbeiten eines Events sucht der Director in der objevents-Konfiguration passende Definitionen und führt die darin definierten Aktionen aus.

Zu beachten ist, dass keine Events ausgelöst werden, wenn am Director vorbei direkt an den Datenbankinhalten manipuliert wird. Es werden ebenfalls keine Events ausgelöst, wenn per Director-Befehl direkt auf Rohdaten zugegriffen wird (per „copy"-Kommando).

Klassen

Benutzer- und Rechner können in Klassen eingeteilt werden. Alle Benutzer/Rechner

einer Klasse teilen dabei gemeinsame Eigenschaften. Dabei kommen zwei Mechanismen zum Einsatz

- Templates: Wird ein neues Objekt in einer Klasse erzeugt, so dienen die in der Klasse gültigen Templates als Eingabehilfen. Ein Template kann z.B. festlegen, wie ein Login-Name aus Vor- und Nachname gebildet wird, wie die UID vergeben werden soll, etc.
- vererbte Eigenschaften: Eigenschaften einer Klasse werden an alle Mitglieder vererbt. Wird z.B. in einer Benutzerklasse ein Homedirectory-Server festgelegt, dann ererben alle Klassenmitglieder diese Eigenschaft – es sei denn, für ein Mitglied wird explizit ein anderer Homedirectory-Server festgelegt.

Templates sind nur zur Zeit der Neuerstellung von Objekten wirksam, während vererbte Eigenschaften bei jeder Zugriffsoperation auf das Objekt aufgelöst werden. Nachträgliche Aenderungen von Templates wirken sich also nicht auf bestehende Objekte aus, die Aenderungen vererbter Eigenschaften dagegen schon. Auf Grund von Templates gesetzte Attribute sind explizit im zugehörigen Datenbankobjekt abgelegt, sind also für beliebige LDAP-Clients sichtbar. Vererbung hingegen ist ein Director-eigener Mechanismus, fremde LDAP-Clients ignorieren diese einfach. Vererbung empfiehlt sich also vor allem für Director-spezifische Attribute, z.B. Homedirectory-Server und -Pfade, zugewiesene Applikationen, etc.

Klassen können selber Mitglieder anderer Klassen sein – es lässt sich also eine hierarchische Klassenstruktur aufbauen.

Der geschickte Einsatz von Templates vereinfacht die alltägliche Benutzer- und Rechnerverwaltung sehr, im günstigsten Fall können so z.B. Benutzer durch Eingabe nur von Vornamen, Nachnamen, Klasse und Passwort eingerichtet werden – um die restlichen (Pflicht-)Attribute kümmert sich der Director mittels Templates selbst.

Template-Definitionen finden sich im Abschnitt UI der Klassendefinition.

Das Kommandozeilen-Interface

Alle Kommandozeilenfunktionen des Directors werden mit dem Kommando

```
sfidirector <globale Optionen> <subkommando> <sub-Optionen>
<Argumente>
```

erreicht. Eine Liste der bekannten Subkommandos und ihrer Kurzbeschreibung gibt dabei das Kommando

```
sfidirector help
```

aus. Nützlichste globale Option ist die Option -v (verbose), sie sorgt dafür, dass ausgeführte Kommandos zusätzliche Informationen ausgeben. Gross geschrieben als -V werden noch mehr Infos ausgegeben und kombiniert als -v -V ist der Gipfel erreicht.

Viele Kommandos lösen ein Event mit zugehörigen Aktionen aus. In diesem Fall wartet das Kommando auf Beendigung dieser Aktionen und beendet sich erst mit deren Abschluss. Für die Ausführung von Aktionen ist aber der Director-Server zuständig. Das hat vor allem zwei Konsequenzen:

- Ein Abbrechen des Kommandos (z.B. mit Ctrl-C) stoppt nicht die Aktion
- Per Option -b (batch) kann ein Kommando aufgefordert werden, Events zwar auszulösen, die damit verbundenen Aktionen aber nicht zu beobachten und sofort zu terminieren

Kommandos, die eine Aktion auslösen, geben normalerweise die „job id" dieser Aktion aus. Den Status dieser Aktion kann man jederzeit mittels des

```
sfidirector monitor <job-id>
```

Kommandos prüfen.

Der Rückgabewert (exit status) eines Kommandos gibt über Erfolg oder Nichterfolg Auskunft. Wird ein Kommando mit der -b-Option ausgeführt, dann hat der Erfolg einer eventuell ausgeführten Aktion keinen Einfluss auf den Rückgabewert, da diese ja erst nach der Beendung des Kommandos ausgeführt werden wird.

Rohzugriffe auf Datenbanken

der copy-Befehl / aus der Weboberfläche nicht möglich

Mittels des „copy"-Befehls wird direkt auf Rohdaten des Directors zugegriffen. Der copy-Befehl kann verwendet werden, um Daten ins Repository zu importieren, zu prüfen, welche Rohdaten im Repository liegen, oder um Datenkonversionen durchzuführen. Die Weboberfläche erlaubt zur Zeit keinen direkten Zugriff auf Rohdaten.

Beispiele:

```
sfidirector copy passwdfile:file:/etc/passwd directory:People
```

liest die Datei /etc/passwd, konvertiert alle Einträge in Datenbank-Objekte und speichert diese im Personenteil des Repositorys. Um bestehende Einträge dabei zu überschreiben kann die Option -f (force) verwendet werden.

```
sfidirector copy -m passwdfile:file:/etc/passwd
shadowfile:file:/etc/shadow directory:People
```

liest zusätzlich auch die /etc/shadow-Datei, verheiratet die Einträge aus beiden Dateien (-m=Merge) und speichert die resultierenden Objekte im Repository.

```
sfidirector copy -t directory:People passwdfile:stdio:
```

liest sämtliche Personen-Objekte aus dem Repository, konvertiert sie ins /etc/passwd-Format und gibt sie an die Standard-Ausgabe aus. Die Option -t (Truncate) sorgt dafür, dass die Ausgabedatei vor der Operation geleert wird, ohne -t wird an eine bestehende Datei angehängt.

```
sfidirector copy -s 'ou=Hosts' directory:Top
ldiffile:file:/tmp/hosts.ldif
```

kopiert den Teil des Repositories, in dem sich Rechnerinformationen befinden im LDIF-Format in die Datei /tmp/hosts.ldif. Falls die Datei bereits existiert, werden neue Einträge angehängt. Auch Einträge, die nicht Rechner spezifizieren, aber sich im gleichen Teilbaum befinden, werden mitkopiert – sonst sollte als Quelle „directory:Hosts" gewählt werden.

Objekte manipulieren

list, create, modify, delete, schema / File-New, Add Object, Kontextmenü im Objekt-Browser

Die Subkommandos list, create, modify und delete zeigen Objekte an, erzeugen welche, modifizieren oder löschen einzelne oder ganze Gruppen von Objekten.

In der Weboberfläche sind dieselben Funktionen über den Objektbrowser (Hauptansicht, oder im Menü Go Browse anwählen) erreichbar und verhalten sich ähnlich wie gewohnte Verzeichnisbrowser gängiger Benutzeroberflächen (z.B. Windows-Explorer).

Alle Objekt-Kommandos arbeiten mit Directories (s.o. Directories) und erlauben keine

Angabe von Datenbanken, so ist

```
sfidirector list -a Hosts
```

ein gültiges Kommando, das alle Rechner im Repository auflistet, während

```
sfidirector list -a directory:Hosts
```

ungültig ist.

Zu Debugging-Zwecken ist gelegentlich das spezielle Directory Debug nützlich, so erstellt z.B.

```
sfidirector create Host Debug "Host=testrechner"
```

zwar einen Rechnereintrag für einen Rechner namens testrechner, legt ihn aber nicht im Repository ab, sondern in der Datei debug im aktuellen Verzeichnis. Den Inhalt dieser Datei kann man sich mit

```
sfidirector list -a Debug
```

anzeigen lassen oder natürlich im Rohformat direkt mit einem Texteditor anschauen.

Um herauszufinden, welche Objekttypen der Director kennt und welche Attribute diese haben, wird das schema-Kommando verwendet. Ueberall, wo eine Attributangabe erforderlich ist, erfolgt diese mittels Attributname=Wert, wobei der Attributname dem Namen entspricht, wie in das schema-Kommando ausgibt.

Für Listenattribute gilt: Soll die Liste erweitert werden, so soll die Syntax Attributname=+Wert verwendet werden, soll ein Eintrag aus der Liste gelöscht werden, so muss die Syntax Attributname=-Wert verwendet werden. Wird vor den Wert weder ein + noch ein - hingeschrieben, so wird der Wert der Liste hinzugefügt.

Beispiel:

```
sfidirector modify Person:muster Classes=-'Standard User'
Classes='+Abteilung 1' Classes='+Gruppe 1'
```

entfernt den Benutzer muster aus der Klasse 'Standard User' und nimmt ihn in die Klassen 'Abteilung 1' und 'Gruppe 1' auf.

Objekte anzeigen mit list

Das list-Kommando zeigt einzelne oder eine ganze Reihe von Objekten an.

Beispiele:

```
sfidirector list -a People
```

listet sämtliche bekannte Personen auf.

```
sfidirector list -c Hosts:solarix
```

zeigt den Eintrag für den Rechner namens solarix an. Die -c-Option bewirkt, dass Klassenzugehórigkeiten dabei aufgelöst werden und auch von Oberklassen vererbte Eigenschaften angezeigt werden.

```
sfidirector list -d 'Abteilung A/Gruppe 1' -a People
```

zeigt alle Personen aus dem Teilbaum „ou=Gruppe 1,ou=Abteilung A" an.

```
sfidirector list -a -t Person -s "Login Name=b*" People
```

schliesslich zeigt alle Personen mit einem Login-Namen beginnend mit einem „b" an.

Objekte erzeugen mit create

Das create-Kommando erzeugt einzelne Objekte. Mittels des schema-Kommandos

lässt sich herausfinden, welche Objekte der Director kennt und welche Attribute diese Objekte anbieten.

Beispiele:

```
sfidirector create Person People 'First Name=Peter' 'Last Name=Muster'
Classes='Standard User' Password='test'
```

erstellt einen Benutzer in der Personendatenbank mit Namen „Peter Muster" in der Klasse „Standard User" und mit Passwort „test" (man lese: „create a Person in People-Database with ..."). Hier zahlt sich die geschickte Verwendung von Klassen aus – so kann die Anzahl Attribute, die man zur Erstellung eines Objektes angeben muss auf ein Minimum reduzieren.

Automatisch wird ein create-Event ausgelöst, so dass je nach Konfiguration ein Homeverzeichnis eingerichtet wird und Namensdienste aufdatiert werden.

Objekte modifizieren mit modify

Die Modifikation von Objekten erfolgt mittels des modify-Kommandos. Die Selektion der Objekte, die modifiziert werden sollen erfolgt dabei wie beim list-Kommando, die Angabe der Attribute, die zu ändern sind wie beim create-Kommando. Es ist also explizit möglich, mehrere Objekte auf einmal zu ändern.

Beispiele:

```
sfidirector modify People:muster Crypt=off
```

löscht für den Benutzer muster das hinterlegte Crypt-Passwort.

```
sfidirector –a –t Group –s 'Group Name=a*' Group 'Members=–muster'
```

entfernt aus allen (-a) Gruppen mit einem Namen, der mit a beginnt, den Benutzer muster.

Automatisch wird ein alter-Event ausgelöst, so dass die vorkonfigurierten Aktionen – typischerweise mindestens ein Update aller Namensdienste – ablaufen.

Objekte löschen mit delete

Mit dem delete-Kommando lassen sich einzelne oder mehrere Objekte aus dem Repository löschen. Der Aufruf erfolgt dabei mit denselben Argumenten wie das list-Kommando.

Automatisch wird ein remove-Event für die gelöschten Objekte ausgeführt.

Schemainformationen abfragen mit schema

Per schema-Kommando lassen sich Informationen über die dem Director bekannten Objekttypen und deren Attribute abfragen. Die so gewonnene Information wird im create- und modify-Kommando benötigt und ausserdem in den -t und -s Optionen aller Objektmanipulationskommandos.

Beispiele

```
sfidirector schema
```

listet alle bekannten Objekttypen auf.

```
sfidirector schema Printer
```

listet die Attribute des Objekttyps Printer auf.

Events manuell auslösen

listevents, event / Weboberfläche Tools und Event-Kontextmenü

Events in der Weboberfläche

Events können in der Weboberfläche manuell über 3 Wege ausgelöst werden

- im Browser per „Event"-Eintrag im Kontextmenü eines Objektes
- ebenfalls im Browser über den „Event"-Eintrag im Edit-Menü
- im Objekteditor eines editierten Objektes sind im Tool-Menü je nach Objekttyp einige ausgewählte Ereignisse vorhanden

verfügbare Events für ein Objekt

Mit dem listevents-Kommando lässt sich für ein gegebenes Objekt herausfinden, welche Events es verstehen wird. Beispiel:

```
sfidirector listevents Hosts:testrechner
```

sucht im Directory Hosts nach einem Eintrag testrechner und zeigt alle Events an, die darauf anwendbar sind.

Event auslösen

Zum Auslösen von Ereignissen wird das event-Kommando verwendet. Es hat die gleiche Syntax wie die Kommandos zur Objektmanipulation (s.o. Objekte manipulieren). Beispiele sind:

```
sfidirector -b event create People:muster
```

führt die Aktion aus, die normalerweise beim Anlegen des Benutzers muster ausgeführt werden würde. Die Option -b hat zur Folge, dass die entsprechenden Aktionen an den Director Server übergeben werden, aber das Kommando nicht abwartet, bis diese abgearbeitet sind, sondern sofort terminiert.

```
sfidirector event -a appinventory Hosts
```

führt auf allen bekannten Rechnern eine Inventur der installierten Software-Pakete aus und speichert die Information im Repository.

Bedienung des Directors – erweiterte Funktionen

Im vorherigen Kapitel wurde erklärt, wie im Director Objekte manipuliert werden. Dies stellt die Grundfunktionalität des Directors dar. Alle weiteren Funktionen basieren auf diesen elementaren Funktionen und alle Funktionen in diesem Kapitel können ebenfalls mit diesen Funktionen ausgeführt werden – z.T. allerdings etwas mühselig.

Rechner unter Director-Verwaltung stellen

SSH-Keys, Invoker, Bootstrap-Pakete, Rechner-Eintrag

Um einen Rechner mit dem Director verwalten zu können müssen mehrere Vorbedingungen erfüllt sein:

- Der Rechner muss im Director-Repository eingetragen sein.
- Der Director muss mit Administrator-Rechten („root") Kommandos auf dem Rechner ausführen können – typischerweise per ssh.
- Die entsprechende Methode, mit der der Director Administratorzugriff erlangt muss im Rechnerobjekt oder einer Oberklasse definiert sein („Invoker"-Einstellung)
- Der Director-Agent muss auf dem Rechner installiert sein (z.B. per „Bootstrapper" - s.u.)

Administrator-Zugang per SSH einrichten

Die empfohlene Methode, um dem Director-Server Zugang zu verwalteten Rechnern zu gewähren ist SSH. Damit die Kommunikation zwischen Director-Server und Director-Agent klappt, muss der Benutzer „root" - also der Benutzer, unter dem der Director-Server läuft – ohne ein Passwort einzugeben auf dem verwalteten Rechner Shell-Kommandos ausführen können.

Eine unter vielen Möglichkeiten dazu ist:

- Erstellen Sie auf dem Administrations-Server einen SSH-Schlüssel für den Benutzer „root" mit dem Kommando (als root):

  ```
  ssh-keygen -t rsa
  ```

- Im Homeverzeichnis gibt es nun eine Datei id_rsa sowie id_rsa.pub. Erstellen Sie auf dem Zielrechner im Home des Benutzers „root" ein Verzeichnis .ssh und kopieren Sie den Inhalt der Datei id_rsa.pub in die Datei .ssh/authorized_keys:

  ```
  ssh <zielsystem> mkdir .ssh
  scp .ssh/id_rsa.pub <zielsystem>:.ssh/authorized_keys
  ```

- Ueberprüfen Sie die korrekte Funktion mit:

  ```
  ssh <zielsystem> date
  ```

- Falls so das Datum ohne Frage nach einem Passwort angezeigt wird, dann ist die SSH-Verbindung erfolgreich eingerichtet

Rechner im Director erfassen

Um einen zu verwaltenden Rechner zu erfassen werden die im Kapitel Grundfunktionen vorgestellten Funktionen genutzt. Auf der Kommandozeile erfolgt dies z.B. per

```
sfidirector create Host Hosts Host=rechner1 'IP Address=1.2.3.4'
```

```
Invoker=ssh
```

Dies trägt einen Rechner mit Namen rechner1 und IP-Adresse 1.2.3.4 ein, der Director-Zugriff auf den Rechner erfolgt per ssh.

Director-Agent installieren – Bootstrapper

Um einen Rech ner zu verwalten muss der Director-Server auf diesem den Director-Agent ausführen können. Solange dieser nicht installiert ist, kann der Director folgerichtig den Zielrechner nicht verwalten – eine Installation des Director-Agents z.B. per Softwareverteilung ist also nicht möglich. Selbstverständlich kann der Director-Agent installiert werden, indem manuell oder bei der initialen Rechnerinstallation das entsprechende Software-Paket aufgespielt wird.

Zusätzlich dazu bietet der Director eine simple „Bootstrap"-Funktion, um seinen Agenten auf Zielsystemen zu installieren ohne dass bereits ein Agent verfügbar ist.

Um die Bootstrap-Funktion zu nutzen muss erst ein sog. Bootstrap-Paket eingerichtet werden. Dazu muss im „Applications"-Ast des Director-Repository ein entsprechender Eintrag gemacht werden, z.B.:

```
sfidirector create 'Boot Strap Package' Applications \
'Name=BeispielBootstrapper' \
'Package URL=file:/var/lib/sfidirector/bootstrap.tar' \
'Shell Commands=cd /' \
'Shell Commands=tar xpf -'
```

Wird ein Bootstrap-Vorgang auf einen Zielrechner ausgelöst, dann geschieht nun folgendes:

- Eine ssh-Verbindung zum Zielrechner wird aufgebaut und eine Shell mit Administratorrechten gestartet
- Die Kommandosequenz cd /; tar xpf - wird ausgeführt
- Auf den Standardeingabekanal (STDIN) dieser Kommandosequenz wird die Datei /var/lib/sfidirector/bootstrap.tar vom Administrations-Server kopiert
- Die Kommandosequenz sorgt dafür, dass die Datei als Tar-Archiv im Root-Verzeichnis ausgepackt wird

Die Datei /var/lib/sfidirector/bootstrap.tar muss selbstverständlich auf dem Administrations-Server erstellt werden. Dies geschieht beispielsweise mit

```
cd /
tar cpf /var/lib/sfidirector/bootstrap.tar /etc/sfidirector \
/usr/share/sfidirector /var/lib/sfidirector \
––exclude=/var/lib/sfidirector/*
```

Falls die Architektur des Zielrechners nicht mit der Architektur des Administrations-Servers übereinstimmt, dann darf dieser natürlich nicht als Vorlage verwendet werden.

Selbstverständlich können per Boostrapping auch Software-Pakete im Paketformat des jeweiligen Rechners direkt installiert werden, ein entsprechendes Bootstrap-Paket für RPMs könnte so aussehen:

```
 sfidirector create 'Boot Strap Package' Applications \
'Name=BeispielBootstrapper' \
'Package URL=file:/var/lib/sfidirector/sfidirector-4.0.4b-2.i386.rpm \
'Shell Commands=cat > /tmp/director.rpm' \
'Shell Commands=rpm -i /tmp/director.rpm' \
```

```
'Shell Commands=rm /tmp/director.rpm'
```

Damit der Beispielrechner rechner1 „gebootstrappt" werden kann muss ihm das Bootstrap-Paket zugewiesen werden:

```
sfidirector modify Hosts:rechner1 'Boot Strapper=BeispielBootstrapper'
```

Der Bootstrap-Vorgang selbst wird wie gewohnt per Event-Mechanismus ausgelöst:

```
sfidirector -v event bootstrap Hosts:rechner1
```

Falls die Aktion Erfolg hatte, kann nun per

```
sfidirector -v event ping Hosts:rechner1
```

überprüft werden, ob die Director-Verbindung zu rechner1 funktioniert.

Bemerkung: Selbstverständlich können „Boot Strapper" und „Invoker" jeweils statt im Rechnerobjekt in einer seiner Oberklassen definiert werden – diese Eigenschaften werden vererbt.

Softwareverteilung

Einmal konfiguriert ist die Softwareverteilung recht einfach zu bedienen: In den Rechnerobjekten oder Klassen sind die zu pflegenden Applikationen einzutragen. Die entsprechenden Software-Pakete müssen vorgängig in einem dem Director bekannten Verzeichnis abgelegt werden („Application Repository"). Die Events app_install, app_remove und app_update dienen dann dazu, einzelne Rechner oder ganze Rechnergruppen auf den im Repository erfassten Stand zu bringen. Die Kommandos app_subscribe und app_dist erleichtern diese Arbeit etwas. Eigene Pakete im Director-eigenen Format können mit den Kommandos „create 'Packable Application'"" und app_freeze erstellt werden – der Director kann aber mit Paketen in den Formaten der unterstützten Betriebssysteme umgehen.

Bis dies funktioniert muss allerdings etwas Vorarbeit geleistet werden: Die Verzeichnisse, in denen Software-Pakete abgelegt werden sollen müssen dem Director bekannt gemacht werden und mit Rechnern oder Rechnerklassen verknüpft werden.

Einrichten von Application Repositories

Ein Application Repository ist im Wesentlichen ein Verzeichnis, in dem verteilbare Software-Pakete abgelegt werden. Der Director Server muss auf diese Application Repositories Zugriff haben, um

- einen Index der verfügbaren Applikationen und Pakete zu verwalten
- während einer Applikationsverteilung diese Pakete auszuliefern

Jedem Rechner muss explizit eines oder mehrere Application Repositories zugewiesen sein. Selbstverständlich kann dies auch in seiner Rechnerklasse geschehen, so dass nicht jeder Rechner individuelle Einstellungen benötigt.

Auf dem Rechner werden nur die Pakete installiert oder entfernt, die in „seinem" Repository liegen.

Ein Repository wird eingerichtet, indem im Dateisystem ein passendes Verzeichnis erstellt wird und dann im Director-Repository (nicht zu verwechseln mit dem Application Repository) ein Eintrag für dieses Application Repository erstellt wird, z.B.

```
sfidirector create 'Application Repository' AppRepository Name='My
Repository' URL=file://myserver/export/myrepository Server=myserver
'Path within Server=/export/myrepository'
```

Die URL bestimmt dabei, wie der verwaltete Rechner auf die im Repository abgelegten Pakete zugreifen kann – file://myserver... bedeutet dabei, dass der Director Agent das Paket über den Director Server anfordert – während der Eintrag Server und Path within Server bestimmt, wie der Director Server selbst zugreift.

Mittels

```
sfidirector event scanrepository AppRepository:'My Repository'
```

wird nun der Director Server aufgefordert, das Verzeichnis nach bereitliegenden Software-Paketen abzusuchen. Für alle gefundenen Pakete wird im Repository ein Objekt erstellt. Nach jedem Hinzufügen oder Entfernen von Software-Paketen im Verzeichnis muss diese Operation wiederholt werden.

Einer bestehenden Klasse „Standard Rechner" kann nun das eben eingerichtete Application Repository zugewiesen werden mit

```
sfidirector modify HostClass:'Standard Rechner' Repositories='My
Repository'
```

Um Netzwerkbandbreite zu sparen kann der Director mit *Mirrors* umgehen. Um einen Mirror einzurichten geht man gleich vor wie um ein normales Application Repository einzurichten. Zusätzlich setzt man aber im Eintrag für das Repository

```
'Mirror of=My Repository'
```

Der Director weiss nun, dass das neu erstellte Application Repository eine identische Kopie des bestehenden My Repository darstellt. Es muss natürlich gewährleistet sein, dass dies tatsächlich der Fall ist. Der Director bietet zwar eine Funktion dafür, allerdings tun hier Mirroring-Werkzeuge wie rdist oder rsync einen deutlich besseren Job.

Als URL für den Zugriff auf das Application Repository sollte nun natürlich nicht mehr file://myserver/... angegeben werden – sonst läuft der Zugriff nach wie vor über den Director Server – vorausgesetzt, der Director hat überhaupt direkten Zugriff auf das Verzeichnis. Zur Zeit muss zwingend eine file:/...-URL verwendet werden, da andere Protokolle noch nicht unterstützt sind. Ein file:/net/fileserver/mirror/myrepository bewirkt, dass die verwalteten Systeme direkt in ihrem Filesystem unter /net/fileserver/mirror/myrepository nach Paketen suchen.

Um Pakete in einem Application Repository aufzunehmen werden ganz einfach die Paketdateien ins angegebene Verzeichnis kopiert, dann per „scanrepository" der Director darauf aufmerksam gemacht, dass Aenderungen passiert sind. Eine spezielle Variante dessen ist die Verteilung von Paketen via des APT-Paketsystems (Debian). Hier ist es nicht notwendig, die jeweiligen Pakete direkt ins Application Repository zu kopieren, da APT darauf basiert, dass jeder Rechner „weiss", wo er sich Pakete besorgen kann. Statt dessen wird für APT-Pakete lediglich eine (oder mehrere) Dateien beliebigen Namens mit der Dateiendung .apt im Repository-Verzeichnis erstellt und lediglich eine Liste der Anwendungsnamen, die der Director verwalten soll, eingetragen, also z.B.:

```
cd /export/myrepository
echo dhcp3-server >> packages.apt
echo bind9 >> packages.apt
sfidirector -v event scanrepository AppRepository:'My Repository'
```

Der Director kennt nun die Pakete dhcp3-server und bind9 und weiss, dass diese per apt-get-Kommando zu installieren oder entfernen sind. Dies setzt allerdings ein funktionierendes APT-System sowohl auf dem Rechner, auf dem das Application Repository eingerichtet ist, als auch auf allen Zielrechnern voraus.

Anwendungen zuweisen, installieren und löschen

app_dist, app_subscribe, app_unsubscribe

Pakete aus den eingerichteten Application Repositories können Rechnern zugeordnet und verteilt werden. Prinzipiell sind beide Funktionen über die bereits bekannten Grundfunktionen *modify* und *event* zugänglich. Pakete werden nämlich zugeordnet, indem die entsprechenden Attribute (Subscribed, Remove) im Rechner- oder Rechnerklassenobjekt gesetzt werden und die Applikationsverteilung wird wie gewohnt per Event-Mechanismus angestossen.

Um dies ein wenig zu vereinfachen wurden die Kommandos *app_subscribe* und *app_dist* eingeführt. Mittels app_subscribe wird der Director aufgefordert, die Applikationszuweisungen an Rechner oder Rechnerklassen zu modifizieren, z.B. wird mit

```
sfidirector app_subscribe –a samba rechner1
```

dem Rechner rechner1 das Paket samba zugewiesen. Mittels der Optionen -v, -p und -m kann beeinflusst werden, welche Version eines Paketes installiert werden soll, falls sich im Repository mehrere befinden. Ansonsten wird jeweils das mit der höchsten Versionsnummer installiert werden.

Um Pakete ganzen Rechnerklassen zuzuordnen wird app_subscribe z.B. so benutzt:

```
sfidirector app_subscribe –a samba –C 'Standard Rechner'
```

Schliesslich dient das Kommando *app_unsubscribe* dazu, zugewiesene Pakete wieder zu entfernen, also z.B.:

```
sfidirector app_unsubscribe –a samba rechner1
```

Die Kommandos app_subscribe und app_unsubscribe führen nie direkt automatisch zu einer Installation. Alle Modifikationen bei der Paketzuordnung können also in Ruhe gemacht werden. Erst wenn man sich seiner Sache sicher ist, wird mit app_dist die eigentliche Applikationsverteilung angestossen:

```
sfidirector –v app_dist –h rechner1
```

prüft, welche Pakete auf rechner1 installiert sind, welche laut Informationen im LDAP installiert sein sollten und installiert gegebenenfalls die fehlenden nach oder macht Upgrades.

Wird app_dist zusammen mit der Option -r benutzt kann app_dist auch Pakete löschen – die nämlich, welche explizit mit app_remove ähnlich wie bei app_subscribe als zu löschende Pakete zugeordnet wurden.

App_remove ist erst ab der Director-Version 4.0.6 enthalten. Bis dahin funktioniert die Paketdeinstallation per modify-Befehl. Alle Pakete, die – falls installiert – von einem Rechner entfernt werden sollen, können so z.B. mit

```
sfidirector modify Hosts:rechner1 Remove=ethereal Remove=portmap
```

dem Rechner rechner1 zugeordnet werden.

Selbstverständlich kann auch app_dist auf einzelne Rechner, alle Rechner oder Rechnerklassen angewendet werden.

Damit der Director entscheiden kann, welche Pakete zu installieren oder zu löschen sind, muss er selbstverständlich wissen, welche Pakete denn nun auf dem jeweiligen Zielrechner bereits installiert sind. Dazu macht er ein Paketinventar und legt dieses in seinem Repository unter dem Rechnerobjekt ab. Während jeder Applikationsverteilung wird automatisch ein Inventar neu erstellt. Trotzdem können die Inventardaten gelegentlich von der Realität abwechen – dann nämlich, wenn auf dem Rechner manuell oder per anderem Tool als dem Director Pakete nachinstalliert, aufdatiert oder entfernt

wurden. Ebenfalls fehlt vor dem ersten Applikationsverteilungsversuch das Inventar gänzlich – was sich durch eine entsprechende Fehlermeldung bemerkbar macht. Trotz der Fehlermeldung wird auf jeden Fall eine Inventarisierung durchgeführt, so dass ein zweiter Versuch korrekt Anwendungen verteilen sollte.

Um Probleme durch inkorrekte Inventardaten zu vermeiden kann selbstverständlich eine Neu-Inventarisierung auch automatisch angestossen werden:

```
sfidirector -v event appinventory Hosts:rechner1
```

wird die Inventardaten des Rechners rechner1 auf den neuesten Stand bringen.

In der Weboberfläche findet die Zuweisung von Paketen über den Editiermechanismus statt (Abschnitt Applications, Felder Subscribed und Remove in der Editiermaske zu Rechnern oder Rechnerklassen), die Verteilung kann entweder ebenfalls in der Editiermaske im Tools-Menü oder aber über den Eventmechanismus angestossen werden.

Bemerkung: Die Anwendungsverteilung wird jeweils nur Pakete installieren oder löschen, die im Application Repository verfügbar sind. Vor allem letzteres ist zu beachten, da auf den ersten Blick nicht unbedingt einleuchtend.

Konfigurationsverteilung

In vielen Fällen sind Konfigurationsdateien gewisser Dienste identisch oder nahezu identisch auf verschiedenen Maschinen eines Netzes. Die Konfigurationsverteilung des Directors ist ein einfaches Mittel, um solche Konfigurationen zentral zu verwalten, ohne auf die aufwändigere, aber dafür ebenfalls taugliche Softwareverteilung zurückzugreifen.

Jedem Rechner und jeder Rechnerklasse lassen sich eine Reihe von *Paketen* zuweisen. Diese Paket-Zuweisungen sagen aus, welche *Konfigurationsdateien* aus welchen *Templates (Vorlagen)* installiert werden sollen, wobei ein Template selbst wiederum ein definiertes *Verzeichnis* auf einem definierten *Rechner* bezeichnet. Die Templates selber werden während einer Konfigurationsverteilung normalerweise 1:1 auf den jeweiligen Zielrechner kopiert, jedoch können sie in gewissen Grenzen durch den Director über die *Paketkonfiguration* an den Zielrechner angepasst werden.

Der Director verwendet – anders als die meisten bekannten Template-Mechanismen – keine spezielle Templatesyntax mit eigener Makrosprache. Typischerweise ist ein Template eine Konfigurationsdatei, wie sie auf einem laufenden System einsetzbar ist. Im einfachsten Fall wird als Template so auch direkt die aktive Konfiguration eines Rechners als Vorlage für gleichartig konfigurierte Rechner verwendet.

Konfiguration verteilen

configure-Event

Vorausgesetzt, dass die Konfigurationsverteilung konfiguriert ist, lässt sich die Konfiguration eines Rechners rechner1 per

```
sfidirector -v event configure Host:rechner1
```

aufdatieren, die Konfiguration einer ganzen Rechnergruppe (Klasse) klasse1 mittels

```
sfidirector -v event configure HostClass:klasse1
```

Das „configure"-Event überschreibt dabei nur diejenigen Konfigurationsdateien, die mit dem zu installierenden Original nicht übereinstimmen, bereits auf dem korrekten Stand befindliche Dateien werden nicht berührt. Falls notwendig wird nach dem Ueberschreiben einer Konfigurationsdatei der zugehörige Dienst neu gestartet.

Bemerkung: Kopiert werden nicht nur die Dateiinhalte, sondern auch Berechtigungen und Zeitstempel. Eine überschriebene Konfigurationsdatei erhält also insbesondere als Datum der letzten Aenderung nicht das aktuelle Datum und Uhrzeit, sondern dasjenige der letzten Aenderung an der Vorlage!

Templates erstellen und Pakete/Templates zuweisen

Die zugewiesenen Pakete/Vorlagen werden im Abschnitt System Config des jeweiligen Rechner- oder Rechnerklassenobjektes vermerkt. Mit der folgenden Anweisung

```
sfidirector modify HostClass:klasse1
'Configuration Template=+ntp://server1/'
```

wird beispielsweise der Rechnerklasse klasse1 die NTP-Konfiguration zugewiesen, die auf dem Rechner server1 unter / zu finden ist – das entspricht also der aktiven NTP-Konfiguration auf server1.

Wird statt / ein beliebiges anderes Verzeichnis verwendet, z.B.

```
sfidirector modify HostClass:klasse1
'Configuration Template=+ntp://server1/var/configs/klasse1'
```

dann erwartet der Director, dass unter dem Verzeichnis /var/configs/klasse1 auf server1 die auf die Rechner der klasse1 zu verteilenden NTP-Konfigurationsdateien zu finden sind. Erwartet wird weiterhin, dass sie in einer Verzeichnis-Struktur angelegt sind, die einer aktiven Installation entspricht, dass also beispielsweise eine Datei

```
/var/configs/klasse1/etc/ntp.conf
```

existiert. Dateien, die nicht zum jeweiligen Paket (hier „ntp") gehören, werden ignoriert. Es ist folglich möglich, eine komplette Struktur mit Konfigurationsdateien für diverse Dienste unter dem Verzeichnis anzulegen und mehrere Pakete daraus zu verteilen:

```
sfidirector modify HostClass:klasse1
'Configuration Template=+ntp://server1/var/configs/klasse1'
'Configuration Template=+resolvconf://server1/var/configs/klasse1'
'Configuration Template=+nsswitch://server1/var/configs/klasse1'
```

Zu Konfigurationsdatei passende Paketnamen finden

Um eine Konfigurationsdatei verteilen zu können, muss ein passender „Configuration Template"-Eintrag im Rechner- oder Rechnerklassenobjekt eingetragen werden. Eine Liste der bekannten Konfigurationsdateien und der Pakete, von denen diese Teil sind, gibt das Kommando

```
sfidirector listconfig
```

aus. Die jeweiligen Paketnamen entsprechen der Ausgabe „Packages". Jede Konfigurationsdatei kann in einem oder mehreren Paketen enthalten sein, so kann die NTP-Konfigurationsdatei /etc/ntp.conf sowohl – zusammen mit anderen Dateien – per Paket „ntp" zugewiesen werden, aber ebenfalls über das Paket ntp.conf, das ausschliesslich die ntp.conf-Datei enthält.

In vielen Paketen sind die Konfigurationsdateien nicht strikt fest verdrahtet und mehrere Alternativen werden berücksichtigt. So ist in der Definition von ntp.conf enthalten, dass die ntp.conf-Datei je nach System entweder unter /etc/ntp.conf oder /etc/ntp/ntp.conf gefunden wird. Dabei kann übrigens der Quell- und Zielpfad unterschiedlich sein, d.h. eine /etc/ntp.conf vom Quellsystem geholt und auf dem Zielsystem unter /etc/ntp/ntp.conf installiert werden. Berücksichtigt wird sowohl beim Lesen als auch beim Schreiben, wo bereits eine entsprechende Datei existiert.

Pakete lokal testen, packen, entpacken

saveconfig, restoreconfig

Die Director-Kommandos saveconfig und restoreconfig erlauben es, lokal, d.h. ohne Einbezug des Director-Servers einzelne Pakete in ein Archiv zu schreiben oder aus dem Archiv wieder zu entpacken – dies entspricht dem Mechanismus, den auch das „configure"-Event verwendet.

Mittels

```
sfidirector saveconfig -f datei.config ntp resolvconf nsswitch
```

werden beispielsweise alle Konfigurationsdateien, die zu den Paketen ntp, resolvconf und nsswitch gehören in die Datei datei.config gesichert. Mittels

```
sfidirector restoreconfig -f datei.config
```

wird die mit saveconfig gesicherte Konfiguration wieder entpackt und installiert.

Mit der Option „-c" ist es möglich, beim Entpacken ausserdem die Paketkonfiguration eines bestimmten Rechners oder einer Rechnerklasse anzuwenden (s. unten, Paket-konfiguration):

```
sfidirector restoreconfig -f datei.config -c Hosts:rechner1
```

wendet beim Entpacken die Paketkonfiguration des Rechners rechner1 an.

Eigene Konfigurationsdateien definieren

Es gibt zwei verschiedene Arten von Definitionen, die vom Director berücksichtigt werden:

- „Built-in": Diese Definitionen sind im Director-Binary fest eingebaut und lassen sich nicht ohne Neukompilation verändern
- „konfiguriert": Diese Definitionen werden aus der Datei configfiles (s. Pfade der Konfigurationsdateien, Seite 5) geholt

Eigene Definitionen lassen sich am einfachsten über die zweite Alternative einbringen. Die configfiles-Datei referenziert eine Datei „configfilesSite" (normalerweise unter /etc/sfidirector, je nach Installation). Diese ist für eigene Erweiterungen vorgesehen. Beispiele dafür, wie Einträge darin aussehen, finden sich in „configfiles".

Paketkonfiguration – Konfigurationsdateien beim Verteilen verändern

Oft sind Konfigurationen auf mehreren Rechnern nahezu identisch, aber nicht vollständig identisch. Mal mag eine Samba-Konfiguration bis auf die WORKGROUP-Einstellung identisch sein, mal man in einer NTP-Konfiguration eine andere Peer-Liste erforderlich sein, etc. Trotz dieser kleinen Unterschiede ist die restliche Konfiguration identisch und sollte zentral verwaltbar sein.

Der Director erlaubt es, über die Paketkonfiguration Konfigurationsdateien während der Verteilung geringfügig ans Zielsystem anzupassen. Dazu werden im Abschnitt „System Config" unter „Package Configuration" eines Rechners oder einer Rechnerklasse Variablenzuordnungen definiert, z.B.:

```
sfidirector modify Hosts:rechner1
'Package Configuration=samba.global.workgroup=TEST'
```

Das Beispiel sorgt dafür, dass während der Verteilung der Samba-Konfiguration im Abschnitt [global] der smb.conf-Datei der Workgroup-Eintrag auf den Wert „TEST" gesetzt wird.

Welche Variablen der Director versteht ist abhängig von der Art der Konfigurationsdatei. Prinzipiell fangen alle Variablennamen mit einem Präfix an. Diesen listet das listconfig-Kommando unter „Prefix" auf. Typischerweise folgt nach dem Präfix ein Name, der der jeweiligen Einstellung in der Konfigurationsdatei entspricht, also z.B.

```
ntp.conf.peer=192.168.1.12 192.168.1.15
```

um in der NTP-Konfiguration 192.168.1.12 und 192.168.1.15 als Peers einzutragen. In künftigen Director-Versionen wird das listconfig-Kommando Informationen zu den möglichen Konfigurationen jeder Datei ausgeben können mit z.B.:

```
sfidirector listconfig ntp.ntpconf
```

Bekannte Konfigurationseinstellungen

Eine unvollständige Liste der bekannten Konfigurationseinstellungen:

Name	***Bedeutung***
samba.global.*	Einstellung in der [global]-Section von smb.conf
ntp.keys.*	Schlüssel mit Nummer '*' in NTP-Keydatei setzen
ntp.tickers.server	Servereintrag in ntp step-tickers Datei
ntp.conf.peer	Liste der Peers in ntp.conf (Leerzeichen-getrennt)
ntp.conf.server	Liste der Server in ntp.conf
ntp.conf.*	beliebiger Eintrag in ntp.conf
squid.*	beliebige Einstellung in squid.conf, sofern nur einmal in der Konfigurationsdatei vorkommend (keine Einstellung von komplexen Optionen wie acls)
sendmail.cw.names	Liste der Namen in sendmail.cw-Datei
sendmail.masquerade.-domains	Liste von Namen in masquerade-domains Datei
sendmail.conf.smarthost	DS-Eintrag in sendmail.cf
sendmail.conf.hub	DH-Eintrag in sendmail.cf
sendmail.conf.local	Namen in C-Klasse (lokale Namen)
sendmail.access.*	Eintrag in sendmail access-Datei
uucp.call.*	User und Passwort für einen Eintrag in der UUCP-Calls-Datei
uucp.dialcode.*	Wähl-Prefix in UUCP-Dialcode-Datei
inittab.*	inittab-Eintrag mit Label '*'
resolvconf.nameserver	Liste der Nameserver in /etc/resolv.conf
resolvconf.domain	Domain in /etc/resolv.conf
logindefs.*	Logindefs-Eintrag

Namens- und Verzeichnisdienste

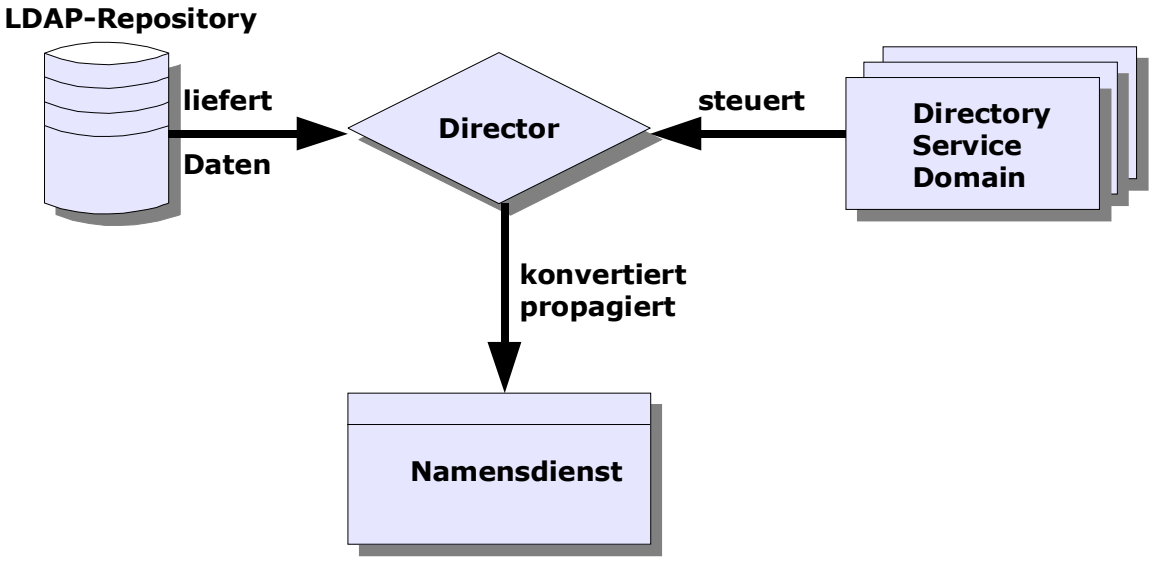
Eine der Grundfunktionen des Directors ist die Konversion der Datenformate diverser Namens- und Verzeichnisdienste Datenformate ineinander. Während der „copy"-Befehl (s. Seite 10, Rohzugriffe auf Datenbanken) es erlaubt, einzelne Dateien oder Datenquellen zu konvertieren, kennt der Director mit den sogenannten *Directory Service Domains* und den zugeordneten *Directory Service Rules* einen mächtigen Mechanismus, um automatisiert Verwaltungsdaten in diverse Namensdienste zu propagieren.

Funktionsprinzip

Eine *Directory Service Domain* fasst eine oder mehrere *Directory Service Rules* zu

einer Einheit zusammen. Letztere beschreiben Datenquellen und -ziele, und wie Daten vom Quell- ins Zielformat konvertiert werden müssen. Typischerweise wird eine Datenquelle ein Teil der vom Director verwendeten LDAP-Datenbank sein, während Datenziele nahezu beliebige Konfigurationsdateien oder Datenbanken eines Namensdienstes sein können.

Jeder *Directory Service Domain* können einzelne Rechner oder Rechnerklassen zugeordnet werden. Diese Zuordnung bedeutet, dass die Datenziele sich beim Propagieren der Namensdienste auf den jeweiligen Rechnern befinden.



Um diese sehr abstrakte Einleitung etwas zu illustrieren folgen hier Beispiele:

- *DHCP*. Im Director erfasste Rechner sollen automatisch in der Konfiguration des ISC-DHCP-Server auf Rechnern in der Rechnerklasse „DHCP-Server" eingetragen werden.

  Dazu weisen wir der Directory Service Domain „DHCP" die Rechnerklasse „DHCP-Server" zu. Diese Domain enthält zwei Directory Service Rules: 1. Rechnerinformation wird auf dem Director-Server aus dem Directory Hosts (directory:Hosts, s. Directories, Seite 8) geholt; 2. Die Rechnerinformation wird in das Format der DHCP-Server-Konfiguration konvertiert und auf dem Zielrechner in der Datei /etc/dhcpd.conf eingefügt. Nach dem Aufdatieren der DHCP-Konfiguration wird automatisch der DHCP-Dienst neu gestartet. Die Daten werden jeweils dann abgeglichen, wenn dies manuell angestossen wird oder dann, wenn per Director Rechnerinformation modifiziert wird.

- *Passwd*. Um Server von Namensdiensten unabhängig zu machen, sollen die Benutzerkontodaten direkt in den Dateibasierten Tabellen, d.h. /etc/passwd, /etc/shadow und /etc/group, auf den Systemen liegen. Lokale Benutzer und Gruppen sollen weiterhin direkt auf den Zielsystemen verwaltet werden und daran erkannt werden, dass deren uid/gid kleiner als 5000 oder grösser als 50000 sind.

  Dazu weisen wir der Directory Service Domain „Local Users Files" die entsprechenden Server zu. Diese Domain enthält eine ganze Reihe von Directory Service Rules:

1. Benutzerkonten werden aus dem Directory „People“ auf dem Director-Server geholt
2. Gruppen werden aus dem Directory „Groups“ auf dem Director-Server geholt
3. Auf dem Zielrechner werden aus /etc/passwd und /etc/shadow alle Benutzereinträge extrahiert, deren uid kleiner als 5000 oder grösser als 50000 ist
4. Auf dem Zielrechner werden aus /etc/group alle Gruppen extrahiert, deren gid kleiner als 5000 oder grösser als 50000 ist
5. Auf dem Zielrechner werden die Konten aus 1. und 3. zusammengefügt und die Datei /etc/passwd neu geschrieben
6. Auf dem Zielrechner werden die Konten aus 1. und 3. zusammengefügt und die Datei /etc/shadow neu geschrieben
7. Auf dem Zielrechner werden die Gruppen aus 2. und 4. zusammengefügt und die Datei /etc/group neu geschrieben