LDAP-Server mit Tcl bearbeiten

Baum-Pflege

Per Lightweight Directory Access Protocol lassen sich Benutzerdaten zentral in einer Baumstruktur verwalten. Linux, Apache oder E-Mail-Clients greifen zur Authentifizierung oder für Kontaktinformationen darauf zu. Dank der LDAP-Erweiterung bearbeiten auch Tcl-Programme die Daten auf dem LDAP-Server. Carsten Zerbst

Wer als Admin viele Rechner und Benutzer betreut, ist es schnell leid, die Passwortdateien verschiedener Rechner und Dienste zu synchronisieren. Auch um Kontaktdaten zentral zu speichern sind einfache Lösungen gefragt – es muss nicht gleich ein CRM-System sein (Customer Relationship Management). In beiden Fällen hilft ein LDAP-Server, der die Daten zentral anbietet. Das Lightweight Directory Access Protocol (LDAP) wird von vielen Programmen und Betriebssystemen eingesetzt, um Daten über Personen, Organisationstrukturen oder Rechner abzufragen.

Als Server kommt vor allem der Slapd aus dem OpenLDAP-Projekt [1] zum Einsatz, es gibt aber auch kommerzielle Alternativen von Sun oder Netscape. Für die Client-Seite greifen Tcl-Programmierer auf die LDAP-Erweiterung von Tcl zurück, sie automatisieren damit komplexere Aufgaben oder konvertieren und übertragen Daten. Diese Erweiterung gehört bei allen wichtigen Distributionen zum Lieferumfang, die Quellen sind bei [1] zu finden.

Als Einstieg zeigt Listing 1 eine einfache LDAP-Abfrage von Personendaten. Das Beispiel verwendet den öffentlichen LDAP-Server der kalifornischen Chapman-Universität, zu dem das Kommando »LdapBind« (Zeile 9) eine anonyme Verbindung herstellt. Anonyme Verbindungen benötigen keinen Usernamen und kein Passwort, jeweils ein leerer String genügt. Wie bei anderen Datenbankverbindungen auch ist das Ergebnis von »LdapBind« ein Handle, das die Verbindung bei den weiteren Kommandos identifiziert.

Die LDAP-Baumstruktur

Danach fragt »LdapSearch« die Daten eines Eintrags ab. LDAP-Server enthalten baumartig strukturierte Daten, die möglichen Knotentypen und ihre Werte sind durch das jeweils verwendete Schema beschrieben. Standardschemata für verbreitete Knotentypen wie etwa Personen, Organisationen, Gruppen und weitere sind in den RFCs 1274, 2307 und 2798 festgelegt.

Jeder Knoten lässt sich durch den so genannten voll qualifizierten Namen eindeutig wählen. Dieser Name besteht aus den Namen aller Elterneinträge und dem eigenen Namen des Knotens. Der Aufbau einer typischen LDAP-Struktur ist in Abbildung 1 zu sehen: Der Datenbankeintrag »Kontaktdaten« hat zwei Kinder vom Typ »organization« (Firmeneintrag). Beim linken Ast sind mehrere Attribute abgebildet, diese können auch mehrfach vorkommen, wie »objectclass« zeigt. Am rechten Ast ist zusätzlich ein Personeneintrag angehängt.

Anders als von SQL-Datenbanken bekannt starten LDAP-Suchvorgänge immer von einem bestimmten Knoten aus, sie berücksichtigen nicht die darüber liegenden Einträge. Der Anfrager kann die Suchtiefe begrenzen, mögliche Werte sind »base« (nur Attribute aus dem Startknoten), »one« (nur direkte Kinder) oder »sub« (alle Kindknoten). Neben dem Suchbereich lässt sich auch die Anzahl der möglichen Ergebnisse einschränken, »0« steht für beliebig viele. ▶

Tabelle 1: LDAP-Kommandos

Kommando	Erklärung
LdapInit	Initialisiert die Tcl-Erweiterung
LdapBind *Host Port UserDN Passwort*	Gibt die Verbindung zurück
LdapUnBind *Verbindung*	Schließt die Verbindung
LdapDelete *Verbindung DN*	Löscht den Eintrag mit der DN
LdapModify *Verbindung DN Attributliste*	Ändert oder löscht Attribute
LdapAdd *Verbindung DN Attributliste*	Erzeugt einen neuen Eintrag
LdapSearch *Verbindung Suchtiefe Deref Start-DN Max-Results Suchfilter Attributnamenliste*	Sucht Einträge und gibt eine Liste mit Werten zurück

Listing 1: LDAP-Daten per Tcl-Skript abfragen

```
01 package require Ldap
02 LdapInit
03
04 set host ldap.chapman.edu
05 set port 389
06 set userDN ""
07 set passwort ""
08
09 set con [LdapBind $host $port $userDN $passwort]
10
11 set resultat [LdapSearch $con base never \
12    "uid=aanderso,ou=People,o=chapman.edu" 1 \
13    "(objectclass=person)" \
14    [list cn sn uid givenname]
15 ]
16 puts $resultat\n
17
18 set resultat2 [LdapSearch $con sub never \
19    "ou=People,o=chapman.edu" 10 \
20    "(&(objectclass=person)(sn=a*))" \
21    [list cn uid ]
22 ]
23 puts $resultat2
24
25 LdapUnBind $con
```

Listing 2: Array aus einer LDAP-Suche erzeugen

```
01 proc inArray {liste arrayName {usePrefix true}} {
02    upvar $arrayName array
03    set prefix 0;
04
05    set keys {}
06    foreach kv $liste {
07       if {[string match "" $kv]} {
08          incr prefix;
09          continue;
10       }
11       regexp {([[:print:]]+)=
   ([[:print:]\ ]*)} $kv gesamt key value
12       if $usePrefix {
13          set array($prefix,$key) $value
14       } else {
15          set array($key) $value
16       }
17    }
18    set array(length) [incr prefix]
19 }
```

Die »LdapSearch«-Parameter (siehe **Tabelle 1**) Suchtiefe, Deref, Start-DN und Max-Results beschränken den Suchbereich und die möglichen Ergebnisse, der Suchfilter bestimmt die gewünschten Knoten. Hier kommt ein Suchfilter zum Einsatz, wie ihn auch andere LDAP-Werkzeuge verwenden. Optional beschränkt die Attributliste auch die Knotenwerte. LDAP-Datenbanken können sehr groß werden, im produktiven Einsatz sind Verzeichnisse mit mehreren Millionen Einträgen. Die Suchanfrage exakt und eng eingrenzend zu formulieren ist daher in großen Verzeichnissen entscheidend.

In **Listing 1** sind zwei Abfragen enthalten. Die erste (Zeilen 11 bis 15) ermittelt vier Attribute eines bekannten Knotens, die Suchtiefe ist deshalb mit »base« auf null gestellt. Die Ausgabe ist in **Abbildung 2** (oberer Abschnitt) zu sehen, sie besteht aus einer Liste. Für jedes Attribut enthält sie Namen und Wert.

Die zweite Abfrage startet bei einem Or-

Listing 3: User aus »/etc/passwd« übertragen

```
01 package require Ldap
02 package require opt
03
04 LdapInit
05
06 proc importPasswd {con baseDN } {
07    set fd [open "/etc/passwd" r]
08    while { [gets $fd line] >= 0 } {
09       set attribute [list objectclass=top objectclass=posixAccount]
10
11       foreach {uid pw uidNumber gidNumber cn homeDirectory loginShell}
   [split $line :] {
12          if {$uidNumber < 500} continue
13
14          # Eintrag schon vorhanden?
15          if {![catch {LdapSearch $con one never 1 \
16          "(&(objectclass=posixAccount)(uid=$uid))"} err]} {
17             puts stderr "Eintrag für UID $uid schon vorhanden"
18             continue
19          }
20
21          lappend attribute "uid=$uid"
22          lappend attribute "uidNumber=$uidNumber"
23          lappend attribute "gidNumber=$gidNumber"
24          lappend attribute "cn=$cn"
25          lappend attribute "homeDirectory=$homeDirectory"
26          lappend attribute "loginShell=$loginShell"
27
28          # Eintrag anlegen
29          if {[catch {LdapAdd $con "uid=$uid,$baseDN" $attribute} err]}
   {
30             puts stderr "Es trat ein Fehler beim Eintrag von
   $attribute auf, Grund $err"
31          }
32       }
33    }
34    close $fd
35 }
36
37 tcl::OptProc main {
38    {host       "LDAP-Server"}
39    {port -int  "LDAP-Port"}
40    {nutzerBase "Basis-DN für Benutzereinträge"}
41    {?userDN?   "Voll qualifizierter Benutzer-DN"}
42    {?passwort? "Passwort"}
43 } {
44    # Einwahl
45    if {[catch {LdapBind $host $port $userDN $passwort } con]} {
46       puts stderr "Keine Verbindung mit diesen Angaben möglich: $con"
47       exit 1
48    }
49
50    # Basis überprüfen
51    set res [LdapSearch $con one never $nutzerBase 1 "objectclass=top"]
52    if {[regexp "Search failed" $res]} {
53       puts stderr "Nutzer-Basis \u00AB$nutzerBase\u00BB scheint keine
   gültige DN zu sein: $res"
54       exit 1
55    }
56
57    importPasswd $con $nutzerBase
58 }
59
60 if {[catch {eval main $argv} err]} {
61    puts stderr $err
62 }
```

ganisationseintrag und sucht maximal zehn Personeneinträge, die mit dem Buchstaben »a« beginnen. Wie in Abbildung 2 zu sehen ist, kommen auch hier die Ergebnisse als Liste zurück, die jeweils zu einem Eintrag gehörenden Werte werden durch leere Listenelemente »{}« getrennt.

Dieses Ausgabeformat ist allerdings für das weitere Verarbeiten ziemlich unhandlich, deshalb ist in Listing 2 eine Hilfsprozedur angegeben. Sie schreibt die Werte der Liste in ein Array, so wie es von den diversen SQL-Erweiterungen bekannt ist. Folgende Kommandos rufen – am Ende von Listing 1 eingefügt – die Hilfsfunktion auf und geben dann das Array aus:

```
inArray $resultat2 a
parray a
```

Die Ausgabe des »parray«-Aufrufs beginnt wie folgt:

```
a(0,cn)    = Esmael Adibi
a(0,uid)   = adibi
a(1,cn)    = Claudia Alfaro
a(1,uid)   = calfaro
```

Die Ausgabe endet nach dem zehnten Eintrag (Nummer 9). Die Tabelle enthält auch ein Length-Feld, das die Anzahl der Elemente nennt:

```
a(9,cn)    = Mark Axelrod
a(9,uid)   = axelrod
a(length) = 10
```

LDAP-Server sind für viele Abfragen auf relativ statische Inhalte konzipiert, aber irgendwann müssen die Einträge auch in den Server gelangen. Ohne Skriptsprache sind per Hand LDIF-Dateien zu erzeugen und dann mit »ldapadd« einzufügen.

Einfacher geht es mit Tcl, siehe Listing 3. Mit wenigen Zeilen liest das Skript eine »/etc/passwd«-Datei und legt alle neuen Accounts, deren User-ID größer oder gleich 500 ist, im LDAP-Server ab. So lassen sich schnell ganze Rechnerparks auf PAM (Pluggable Authentication Modules) mit einem LDAP-Server zur zentralen Datenhaltung umstellen. Für einen Einstieg in dieses Thema bietet sich [2] an, weitergehende Informationen finden sich bei [3].

Das Opt-Paket wertet die Kommandozeile aus

Um die Verbindungsdaten wie Server, Port und Basis-DN bequem per Kommandozeilenparameter zu erfahren, verwendet das Skript das Opt-Paket (ab Zeile 37). Es gehört seit langem zum Lieferumfang von Tcl, allerdings fehlt immer noch die Dokumentation – eine frühere Feder-Lesen-Folge [4] hat dieses Paket bereits vorgestellt. Anders als Lis-

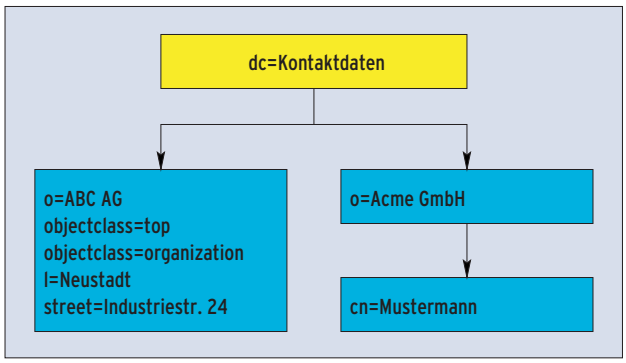

Abbildung 1: Unter dem Startknoten »dc=Kontaktdaten« sind in diesem LDAP-Server Organisationen und Personen gespeichert. Der voll qualifizierte Name von Mustermann lautet »cn=Mustermann, o=Acme GmbH, dc=Kontaktdaten«.

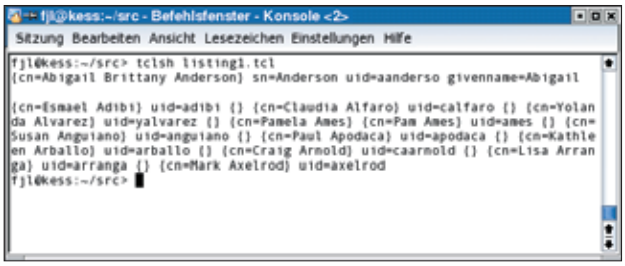

Abbildung 2: Das Ergebnis der Suchanfrage von Listing 1. Die »LdapSearch«-Funktion liefert eine Liste mit den Attributen der gefundenen Einträge. Leere Listenelemente »{}« trennen die Einträge voneinander.

ting 1 kann Listing 2 keinen anonymen Zugang verwenden, da dieser meist keine Schreibrechte hat. Im Zusammenhang mit Authentifizierungsdaten wäre das auch ein gefährliches Sicherheitsloch. Trotzdem sind die Kommandozeilenparameter »userDN« und »passwort« als optional gekennzeichnet (Fragezeichen in Zeilen 41 und 42).

Der größte Teil des Skripts (Zeilen 6 bis 35) dient dazu, die Passwd-Datei auszulesen und in die einzelnen Einträge aufzuteilen. Die Foreach-Schleife in Zeile 11 weist den Inhalt jedes Felds einer Variablen zu. Sie verarbeitet jeweils nur eine Zeile, die Schleife läuft daher immer genau einmal ab. Dennoch ist die Foreach-Syntax sehr vorteilhaft, da sie die Elemente einer Liste einzelnen Variablen zuweist. Die Anweisung »[split $line :]« trennt die Passwd-Zeile an jedem Doppelpunkt auf und gibt sie danach als Liste zurück.

Das »LdapAdd«-Kommando (Zeile 29) erwartet den voll qualifizierten DN (Distinguished Name) des neuen Eintrags sowie eine Liste mit allen Attributen. Die Attribute sind in der gleichen Form anzugeben, wie sie »LdapSearch« als Ergebnis liefert. Das Skript stellt diese Liste in der Variablen »attribute« zusammen, neben dem Verweis auf den richtigen Eintragstyp (Zeile 9) erhält sie die gleichen Felder wie die »/etc/password«-Datei (Zeilen 21 bis 26).

Das Skript zeigt auch die Fehlerbehandlung bei der LDAP-Erweiterung. Jedes Kommando kann Fehler werfen, die das Skript mit »catch« auffangen sollte (Zeilen 16 und 30). Die meisten Fehler sind auf eine gestörte Verbindung oder fehlende Rechte auf dem Server zurückzuführen. Eine weitere Fehlerquelle ist »LdapSearch«: Scheitert die Suche, dann beginnt der Ergebnisstring mit »Search failed« (Zeile 53).

Weitere 30 Zeilen Tcl-Code würden genügen, um auch »/etc/groups« in den LDAP-Server zu füllen. Mit diesen Skripten gestaltet sich der Umstieg auf PAM-Authentifizierung mit LDAP-Anbindung recht bequem.

Aufräumarbeiten

Niemand ist perfekt, irgendwann wird man Einträge ändern und löschen müs-

Listing 4: Daten im LDAP-Verzeichnis ändern

```
01 package require Ldap
02 package require opt
03
04 LdapInit
05
06 source listing2.tcl
07
08 proc säubern {con nutzerBase} {
09    set res [LdapSearch $con one never $nutzerBase 0
   "objectclass=posixAccount" [list uid uidNumber] ]
10    if {[regexp "Search failed" $res]} {
11       puts stderr "Nutzer-Basis \u00AB$nutzerBase\u00BB enthält keine
   Posix-Accounts: $res"
12       return
13    }
14    inArray $res accounts
15    parray accounts
16    for {set i 0 } {$i < $accounts(length)} {incr i} {
17       if {$accounts($i,uidNumber) >= 500} {
18          continue
19       }
20
21       puts "Lösche uid $accounts($i,uid), uidNumber
   $accounts($i,uidNumber)"
22       LdapDelete $con "uid=$accounts($i,uid),$nutzerBase"
23    }
24 }
25
26 proc userIdVergrößern {con nutzerBase} {
27    set res [LdapSearch $con one never $nutzerBase 0
   "objectclass=posixAccount" [list uid uidNumber] ]
28    if {[regexp "Search failed" $res]} {
29       puts stderr "Nutzer-Basis \u00AB$nutzerBase\u00BB enthält keine
   Posix-Accounts: $res"
30       return
31    }
32    inArray $res accounts
33    for {set i 0 } { $i  < $accounts(length)} {incr i } {
34       puts "Wandle uid $accounts($i,uid), uidNumber
   $accounts($i,uidNumber)"
35
36       set attribute [list "uidNumber=[expr {500+$accounts($i,uidNumber)}]"]
37       LdapModify $con "uid=$accounts($i,uid),$nutzerBase" $attribute
38    }
39 }
40
41 tcl::OptProc main {
42    {host       "LDAP-Server"}
43    {port -int  "LDAP-Port"}
44    {nutzerBase "Basis-DN für Benutzereinträge"}
45    {?userDN?   "Voll qualifizierter Benutzer-DN"}
46    {?passwort? "Passwort"}
47 } {
48    # Einwahl
49    if {[catch {LdapBind $host $port $userDN $passwort } con]} {
50       puts stderr "Keine Verbindung mit diesen Angaben möglich: $con"
51       exit 1
52    }
53
54    # Basis überprüfen
55    set res [LdapSearch $con one never $nutzerBase 1 "objectclass=top"]
56    if {[regexp "Search failed" $res]} {
57       puts stderr "Nutzer-Basis \u00AB$nutzerBase\u00BB scheint keine
   gültige DN zu sein: $res"
58       exit 1
59    }
60
61    userIdVergrößern $con $nutzerBase
62    säubern $con $nutzerBase
63 }
64
65 if {[catch {eval main $argv} err]} {
66    puts stderr $err
67 }
```

Das Neueste

Die Tcl-Entwickler haben ihren Weihnachtsurlaub offenbar nicht nur unter dem Tannenbaum verbracht, zumindest Csaba Nemethi hat auch fleißig programmiert. Neue Versionen von gleich drei Paketen sind der Lohn der Mühe: das Widget-Callback-Paket, das Multi-Entry-Widget und die Multi-Column-Listbox [5]. Die ersten beiden Pakete lösen Eingabeprobleme, die nach mehr als dem einfachen Entry-Widget verlangen.

Das Widget-Callback-Paket lässt den Programmierer Benutzereingaben abfangen und verändern, bevor sie im Eingabefeld landen. Vorteil: Der Wert im Eingabefeld und in einer eventuell damit verkoppelten Variablen ist immer gültig, Formatierung oder Wertebereich lassen sich bequem einschränken. Darauf aufbauend löst das Multi-Entry-Widget komplexere Eingabeprobleme, seien es eine gültige Ethernetadresse in Hexadezimalnotation oder andere Eingaben mit vorgegebenen Formaten.

Schöne Tabellen dank mehrspaltiger Listbox

Das dritte Paket nimmt sich eines ganz anderen Problems an, der ansprechenden Darstellung von Tabellen (Abbildung 4). Es hinterlegt die Zeilen mit beliebigen Farben, sortiert auf Mausklick die Daten anhand einer Spalte und lässt den Benutzer mit Hilfe beliebiger Widgets einzelne Zellen ändern. Dabei ist das Widget rein in Tcl/Tk implementiert.

Die aktuelle Tcl-Version 8.4.5 ist kaum erschienen (Ende November 2003), schon nähert sich Version 8.5. Sie wird viele kleinere Verbesserungen enthalten, etwa vollständigen IPv6-Support und das lange ersehnte Alpha-Blending im Canvas (Abbildung 5). Bereits neu erschienen ist Activestates neues Entwicklungspaket Tcl Dev Kit 3.0 [6]. Es unterstützt auch die neueren Entwicklungen, etwa virtuelle Dateisysteme oder Tclkit mit einfachen GUI-Tools.

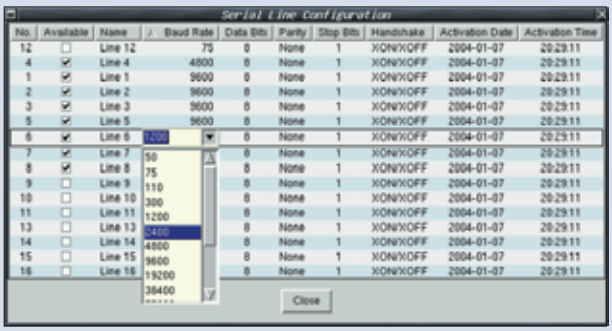

Abbildung 4: Csaba Nemethis Tablelist-Widget ist rein in Tcl/Tk implementiert. Damit lassen sich eigene Programme um ansprechende Tabellen ergänzen, ohne Binary-Erweiterungen zu installieren.

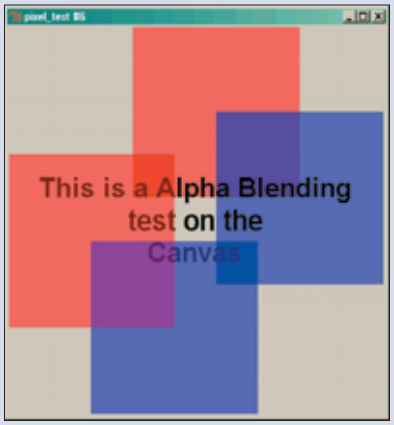

Abbildung 5: In der kommenden Version 8.5 unterstützt das Tk-Canvas-Widget Alpha-Blending. Die Canvas-Elemente werden dabei transparent, darunter liegende Elemente schimmern durch.

sen. Listing 4 enthält die passenden Beispiele. Die Prozedur »säubern« (Zeile 8) löscht mit »LdapDelete« (Zeile 22) alle User, deren ID kleiner als 500 ist. Die Funktion bedient sich des »inArray«-Kommandos aus Listing 2, um bequem auf die gefundenen Einträge zuzugreifen. Das Array enthält dann alle Posix-Accounts, die »LdapSearch« in Zeile 9 (Listing 4) aus dem LDAP-Server abgerufen hat.

Die Prozedur »userIdVergrößern« schiebt die User-ID der verbleibenden Accounts um 500 nach oben. Solche Aufgaben lassen sich ohne Skriptsprachen nur mit aufwändiger Handarbeit erledigen. Die Abfragesprache von LDAP allein wäre nicht ausreichend, ihre Möglichkeiten sind stärker eingeschränkt als die Datenbank-Abfragesprache SQL.

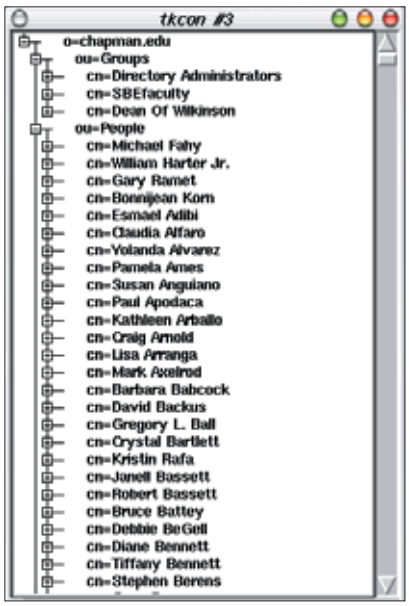

Abbildung 3: Das Tree-Widget von BWidgets stellt die Daten des öffentlichen LDAP-Servers der Chapman-Universität dar. Dieser einfache LDAP-Browser ist in wenigen Zeilen Tcl-Code implementiert.

Alles dabei

Diese vier Beispiele zeigen alle wichtigen Grundlagen, um Einträge im LDAP-Server zu suchen, neu anzulegen, zu ändern und zu löschen – sei es für ein schnelles Skript zwischendurch oder als Basis komplexer Werkzeugen. Bei den Programmquellen für diesen Artikel [7] ist auch ein einfacher LDAP-Client zu finden. Das Programm stellt die Baumstruktur passend mit einem Tree-Widget dar (siehe Abbildung 3). Nicht mal 100 Zeilen sind erforderlich, um die LDAP-Strukturen mit Hilfe der BWidgets abzubilden. *(fjl)* ■

Infos

[1] OpenLDAP: [http://www.openldap.org]
[2] LDAP and OpenLDAP: [ftp://ftp.kalamazoolinux.org/pub/pdf/ldapv3.pdf]
[3] Padl: [http://www.padl.com]
[4] Carsten Zerbst, „Verborgene Schätze – Nützliche Funktionen in Tcl und der Tcllib“: Linux-Magazin 02/02, S. 104: [http://www.linux-magazin.de/Artikel/ausgabe/2002/02/feder/feder.html]
[5] Csaba Nemethi: [http://www.nemethi.de]
[6] Activestate: [http://www.activestate.com/Products/Tcl_Dev_Kit/]
[7] Listings: [ftp://ftp.linux-magazin.de/pub/magazin/2004/03/Feder-Lesen/]

Der Autor

Carsten Zerbst arbeitet bei Atlantec an einem PDM-System für den Schiffbau. Daneben beschäftigt er sich mit dem Einsatz von Tcl/Tk.