Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Bachelorarbeit

Benutzerverwaltung in z/OS

Abstract: In dieser Arbeit werden die Konzepte der Benutzerverwaltung und des Ressourcenschutzes unter z/OS mit RACF erläutert und anhand des Zusammenspiels mit TSO konkretisiert. Außerdem werden die Konzepte von RACF mit denen des UNIX-Ressourcenschutz verglichen. Dazu werden auch die UNIX System Services, die UNIX-konforme Schnittstelle von z/OS, analysiert. Mit Hilfe eines REXX-Scriptes werden, aufbauend auf RACF-Befehlen, komplexere Aufgaben der Benutzerverwaltung implementiert.

Leipzig, Oktober 2015

vorgelegt von
Jonas Kreusch
Studiengang Informatik

Betreuender Hochschullehrer:

Prof. Dr. Martin Bogdan
Fakultät für Mathematik und Informatik
Technische Informatik

Danksagung

Ich bedanke mich an dieser Stelle bei Herrn Prof. Dr. Bogdan für die Möglichkeit, an diesem spannenden Thema zu arbeiten. Mein Dank gilt auch meiner Betreuerin Frau M. Sc. Antonia Siegert für ihre Hilfe und ihre vielen Anregungen.

Außerdem danke ich meiner Freundin Jordana und meiner Familie für ihre Unterstützung, meinem Kumpel Flo für seine Ratschläge beim Bearbeiten von Bildern, meinem Chef Stefan für seine Flexibilität hinsichtlich meiner Arbeitszeiten und meinem Cousin Robert für das Korrekturlesen.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Computer sind heute der zentrale Punkt in der Verarbeitung und Speicherung von jeglicher Art von Daten. Damit entsteht das Bedürfnis danach bestimmen zu können, welche Nutzer Zugriff auf diese Daten haben. In IBMs Großrechnerbetriebssystem erledigt dies die „Resource Access Control Facility“. RACF bietet extrem präzise und dabei trotzdem komfortable und zeitsparende Möglichkeiten zu bestimmen, wer auf welche Art auf eine Ressource zugreifen kann. Während solch ausgefeilte Sicherheitsmechanismen auch heute auf Standard-UNIX-Servern fehlen, schützt RACF schon seit Ende der 70er Jahre Daten auf Mainframes. Am Lehrstuhl „Technische Informatik“ der Universität Leipzig können Studenten, durch Praktikumsaufgaben auf einem Mainframe, grundlegende Fertigkeiten im Umgang mit z/OS erwerben. Da das momentane Vorgehen viele Studentenaccounts auf einmal anzulegen umständlich ist, soll eine bessere Möglichkeit geschaffen werden. Außerdem haben sich im Laufe der Zeit viele obsolete Studentenaccounts auf dem Mainframe angesammelt. Es soll ein Skript entstehen, um diese Accounts zu finden und zu löschen. Beides soll in IBMs Skriptsprache „REXX“ implementiert werden.
Die zu Beginn dieser Arbeit vermittelten Grundlagen über RACF sind notwendig um zu verstehen, wie die Skripte Benutzerzugänge anlegen und löschen. Diese Grundlagen umfassen die Benutzer- und Gruppenverwaltungskonzepte, sowie die Möglichkeiten Ressourcen zu schützen. Es folgt ein Kapitel, in dem die wichtigsten Teile der Skriptsprache REXX erläutert werden. In einem Entwurf wird danach festgehalten, wie sich beiden Problemstellungen genähert werden kann. Das Implementationskapitel erklärt, wie die Entwürfe umgesetzt und wie aufgetretene Probleme behandelt wurden. Das Ergebnis der Implementation sind ein Skript mit dem schnell, viele Accounts angelegt werden können und ein Skript, das es erleichtert, nicht mehr benötigte Accounts zu löschen. Mit einer dokumentierten Beispielausführung der Skripte wurden diese ausgewertet und gleichermaßen deren Bedienung erklärt.

2. Grundlagen

Dieses Kapitel behandelt die Grundlagen von z/OS und RACF. Außerdem werden die Konzepte des Ressourcenschutzes in z/OS mit denen von UNIX verglichen. Dies ist zum Verständnis der praktischen Aufgabenstellung integral.

2.1. RACF

Im Mainframebetriebssystem z/OS ist das Subsystem *Time Sharing Option* „TSO“ der Hauptzugriffsweg für alle Nichtendanwender, also vor allem Systemadministratoren und Programmentwickler. Die Interaktion mit TSO erfolgt im Allgemeinen über die Kommandozeile. Das ist ein Texteingabebereich für Befehle zur Steuerung von Software. TSO verarbeitet den Befehl und gibt eine Antwort oder Bestätigung auf der Kommandozeile aus. Mit RACF ist es unter Anderem möglich, den Zugang der Benutzer zu TSO zu kontrollieren. Die Benutzer werden dabei durch ihre User-ID identifiziert und verifizieren durch Eingabe ihres Passwortes, dass sie wirklich der Benutzer sind.
Nachdem ein Benutzer identifiziert und verifiziert wurde, kontrolliert RACF die Interaktion zwischen dem Benutzer und den Systemressourcen. Hierbei muss nicht nur darauf geachtet werden, auf welche Ressourcen der Benutzer Zugriff haben soll, sondern auch auf welche Art, beispielsweise ob er nur lesenden oder auch schreibenden Zugriff haben soll. Der Ablauf der kontrollierten Interaktion ist in Abbildung 2.1 dargestellt. Damit ein Benutzer passenden Zugriff auf Ressourcen erhält, muss ihm zunächst von einem dazu berechtigten Administrator der Zugang gewährt werden. RACF speichert diese Informationen, indem es für jeden Benutzer, jede Gruppe und jede geschützte Ressource ein Profil in der RACF-Datenbank anlegt.

2.1.1. RACF und TSO

Neben dem generellen Systemzugang über TSO kann RACF auch viele andere Subsysteme, Anwendungen und Datenarten schützen. Einen Überblick bietet Tabelle 2.4, Details zu den Ressourcen folgen im Abschnitt 2.3. Zwar wird die

Verwendung von RACF für die Kontrolle des Systemzugangs über TSO empfohlen, es gibt jedoch auch Alternativen. Versucht ein Benutzer, sich in TSO einzuloggen, so können die zugangsbestimmenden Informationen für den Benutzer in der RACF-Datenbank oder auch im Data-Set SYS1.UADS hinterlegt sein. Da diese TSO-eigenen Benutzerinformationen, insbesondere das Passwort, unverschlüsselt gespeichert werden [BCDB+14, Kap. 9.11.1], sollte der Zugriff darauf mit RACF geschützt werden. Es sollten nur Benutzer in SYS1.UADS eingetragen werden, für die dies unbedingt notwendig ist. Zum Beispiel Systemprogrammierer für den Fall, dass RACF zwecks Wartung deaktiviert werden muss.
Mit RACF können auch die Berechtigungen von Benutzern definiert werden bestimmte TSO-Befehle auszuführen, zum Beispiel das Einreichen eines Jobs mit JCL.[IBM11b, Kap.18]

2.1.2. Logging

Die Möglichkeit, Informationen über Ressourcenzugriffe aufzuzeichnen und aus diesen Informationen einen Report zu erstellen, erweist sich als nützlich für die Ressourceninhaber. Auch für Systemadministratoren und -auditoren sind Logs ein wichtiges Mittel, um die Sicherheit und Stabilität des Systems zu gewährleisten. RACF bietet die Möglichkeit, autorisierte und unautorisierte Versuche für alle Ressourceninteraktionen aufzuzeichen. Diese Informationen werden in ein Log in der *System Management Facility* „SMF“ geschrieben. Besonders wichtig ist die Aufzeichung bei Versuchen:

- sich Einzuloggen,
- auf eine RACF-geschützte Ressource zuzugreifen,
- RACF-Kommandos auszuführen,
- Profile der RACF-Datenbank zu verändern.

2.2. Benutzer und Gruppen

Zwei der grundlegenden Elemente in RACF sind die Benutzer des Systems und die Gruppen, in die sie zusammengefasst werden. Dieser Abschnitt erläutert, wie RACF sie mit Hilfe von Profilen verwaltet.

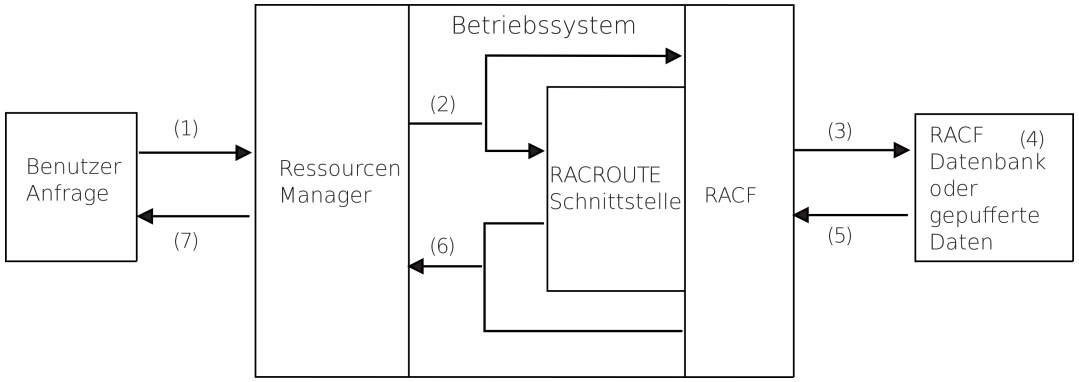

Abbildung 2.1.: Ablauf der kontrollierten Interaktion in RACF (1) Ein Benutzer fordert mittels eines Ressourcen-Managers (z.B. TSO, CICS oder IMS) Zugang zu einer Ressource.
(2) Der Ressourcen-Manager erstellt eine RACF-Anfrage, um zu erfahren, ob der Benutzer auf die Ressource zugreifen darf.
(3) RACF greift auf die RACF-Datenbank (oder eine gepufferte Teilkopie der RACF-Datenbank) zu und...
(4) ... überprüft das entsprechende Ressourcenprofil.
(5) Entsprechend der Informationen aus dem Profil...
(6) ...gibt RACF den Status der Berechtigung (Der Benutzer darf oder darf nicht auf die Ressource zugreifen) an den Ressourcen-Manager.
(7) Der Ressourcen-Manager gewährt oder verweigert dem Benutzer den Zugriff
[IBM13c, Kap.1]

2.2.1. RACF-Benutzer

Aus Sicherheitsgründen ist es sinnvoll, dass jeder Benutzer eine eigene User-ID bekommt. Es gibt jedoch auch Nutzerszenarien, bei denen keine individuelle Zurechnung der Aktionen nötig ist, z.B. öffentliche Endgeräte bei Umfragen. Auch deswegen muss ein RACF-Benutzer nicht immer eine natürliche Person sein, sondern kann zum Beispiel auch ein Programm repräsentieren.
Für jeden Benutzer legt RACF ein Profil in der RACF-Datenbank an, welches Informationen über die Befugnisse des Benutzers enthält. Jedes Profil hat einen Besitzer, das kann ein Benutzer oder eine Gruppe sein, standardmäßig ist es der Benutzer, der das Profil erstellt hat. Der Besitzer des Profils (oder ein Benutzer mit dem Attribut `SPECIAL`, siehe Tabelle 2.2) kann dieses löschen und verändern. Jedes Profil besteht aus einem Basissegment und einer Reihe optionaler Segmente, zum Beispiel für TSO oder OMVS. Jedes Segment besteht wiederum aus Feldern. Die wichtigsten Felder des Basissegments sind in Tabelle 2.1 beschrieben.

Feld	Beschreibung
USERID	Benutzer-ID, 1 bis 8 alphanumerische Zeichen
NAME	Der Name des Benutzers
OWNER	Der Besitzer des Benutzerprofils
DFLTGRP	Die Standardgruppe des Benutzers
AUTHORITY	Berechtigung des Benutzers in seiner Standardgruppe
PASSWORD	Benutzerpassword (Verschlüsselt)

Tabelle 2.1.: Wichtige Felder eines RACF-Benutzerprofils [IBM13c, Kap.3]

Jedem Benutzer können auch Attribute zugewiesen werden. Mit ihnen kann man dem Benutzer zusätzliche Befugnisse oder Limitationen geben. Es gibt Attribute auf System- und auf Gruppenebene. Systemattribute gelten systemweit, Gruppenattribute nur im Kontrollbereich der Gruppe (siehe Abschnitt 2.2.2). Die Benutzerattribute sind in Tabelle 2.2 beschrieben.

2.2.2. RACF-Gruppen

Mit RACF-Gruppen werden RACF-Benutzer zusammengefasst. Idealerweise gruppiert man Benutzer, die weitestgehend die gleichen Rechte haben sollen, etwa eine Gruppe von Praktikanten. Wie Benutzerprofile haben auch Gruppenprofile einen Besitzer. Jeder Benutzer muss mindestens einer Gruppe (seiner Standardgruppe), kann jedoch beliebig vielen angehören. Eine Gruppe kann eigene Ressourcen haben, zum Beispiel Data-Sets, die vor dem Zugriff von

Attribut	Beschreibung
SPECIAL	**Systemebene**: ermöglicht dem Benutzer, alle RACF-Befehle auszuführen und gibt ihm damit volle Kontrolle über RACF-Profile und die RACF-Datenbank.
	Gruppenebene: ermöglicht das Ausführen aller RACF-Befehle für Ressourcen im Kontrollbereich der Gruppe, siehe 2.2.2.
AUDITOR	**Systemebene**: ernennt den Benutzer zum Systemauditor. Ermöglicht, Loggingoptionen von RACF-Profilen einzusehen und zu spezifizieren.
	Gruppenebene: ernennt den Benutzer zum Auditor für Ressourcen im Kontrollbereich der Gruppe.
OPERATION	**Systemebene**: ermöglicht dem Benutzer die Durchführung von Wartungsarbeiten an allen RACF-geschützten Ressourcen, das heißt Katalogisieren, Kopieren, Umstrukturieren und Entfernen von Daten.
	Gruppenebene: ermöglicht Wartungsarbeiten nur an Ressourcen im Kontrollbereich der Gruppe.
CLAUTH	die „class authority “ ist die Berechtigung, Profile einer bestimmten RACF-Klasse zu definieren, siehe 2.3.
GRPACC	wenn vom Benutzer ein Profil für ein Gruppen-Data-Set erstellt wird, haben andere Benutzer der selben Gruppe UPDATE-Zugriff auf dieses Profil, siehe 2.2.2.
ADSP	alle vom Benutzer erstellten permanenten Data-Sets auf einem DASD (Festplatte) werden von RACF geschützt und erhalten ein separates Profil.
REVOKE	hindert den Benutzer daran, sich im System einzuloggen.
RESTRICTED	hindert den Benutzer daran, auf geschützte Ressourcen zuzugreifen, für die er nicht explizit zugelassen ist. Siehe auch UACC 2.3. Hindert den Benutzer nicht daran, auf USS-Dateisystem-Ressourcen zuzugreifen.

Tabelle 2.2.: Wichtige Attribute eines RACF-Benutzerprofils [IBM13c, Kap.1]

Nichtmitgliedern geschützt werden können. Fügt man einen Benutzer zu einer Gruppe hinzu, so gewährt man dem Benutzer standardmäßig Zugriff zu allen Ressourcen, auf die die Gruppe Zugriff hat. Umgekehrt kann man dem Benutzer diese Privilegien entziehen, indem man ihn aus der Gruppe entfernt. Das macht es dem Administrator einfacher, Berechtigungen zu vergeben und den Überblick über diese zu behalten. Ab einer gewissen Gruppengröße kann es auch sinnvoll, sein einen eigenen Gruppenadministrator einzusetzen.

Group authorities Häufig werden Aufgaben innerhalb einer Gruppe an verschiedene Mitglieder delegiert. Für diese verschiedenen Aufgaben können mithilfe der „Group authorities “ angepasste Berechtigungen innerhalb einer Gruppe vergeben werden. In Tabelle 2.3 werden die Berechtigungen in aufsteigender Wichtigkeit aufgelistet. Jede Berechtigung enthält dabei die Rechte der weniger Wichtigen.

Berechtigung	Beschreibung
USE	Zugriff auf die Ressourcen der Gruppe
CREATE	Gruppen-Data-Sets anlegen und mit RACF schützen, jedoch nicht zu löschen.
CONNECT	andere RACF-Benutzer zur Gruppe hinzufügen und ihnen die Berechtigungen USE, CREATE und CONNECT erteilen.
JOIN	neue Benutzerprofile in Klassen, für die er die „Class authority “ (Siehe 2.2) hat, anlegen und jede Gruppenberechtigung vergeben. Außerdem Subgruppen der Gruppe anlegen, in der er die JOIN-Berechtigung hat.

Tabelle 2.3.: Berechtigungen, die ein Benutzer innerhalb einer Gruppe haben kann.

Kontrollbereich einer Gruppe Benutzerattribute, die auf Gruppenebene vergeben werden, geben dem Benutzer Kontrolle über Ressourcen der Gruppe und Ressourcen und Mitglieder von Subgruppen der eigenen Gruppe und deren Subgruppen. Dies ist jedoch nur der Fall, wenn der Besitzer der Subgruppe die übergeordnete Gruppe und nicht ein Mitglied derselben ist. An der Spitze dieser Gruppenhierarchie steht die automatisch erstellte Gruppe SYS1. Sie ist für die Benutzer mit den größten Privilegien gedacht, beispielsweise den Systemadministratoren und -auditoren.

2.3. Ressourcenschutz mit RACF

Ressourcen im Sinne von RACF sind Orte im System, an denen Informationen gespeichert und verarbeitet werden sowie die Befehle, die dazu nötig sind. Ob und mit welchen Rechten ein Benutzer auf eine Ressource zugreifen kann, entscheiden folgende Faktoren:

- Die Attribute des Nutzers
- Die „Group authorities“des Nutzers
- Die Sicherheitsklassen des Nutzers und der Ressource
- Die „Access Authority“der Ressource

Will man eine Ressource schützen, erstellt man für sie ein diskretes oder generisches RACF-Profil. Darin wird gespeichert, wie auf die Ressource zugegriffen werden darf. Diskrete Profile schützen genau eine Ressource. Der Profilname gibt den Namen der Ressource an. Generische Profile schützen eine oder mehrere Ressourcen mit gleichen Sicherheitsanforderungen. Beginnen zum Beispiel die Namen aller Data-Sets der Abteilung „Technische Informatik“ mit „TI.“, so

schützt das Profil „TI.*“ der Klasse DATASET alle Data-Sets der Abteilung, für die kein diskretes Profil vorliegt.

RACF-Ressourcenklassen Da Ressourcen mitunter sehr verschiedene Attribute besitzen und deswegen manchmal unterschiedlich geschützt werden müssen, werden sie in Ressourcenklassen eingeteilt. Es gibt viele vorgefertigte Klassen und es können eigene definiert werden, indem man sie in die *class descriptor table* „CDT“ einträgt. Typische Ressoucenklassen sind „DATASET“ oder „USER“.

Access authority „Access authorities“ (engl. Zugriffsberechtigungen) legen fest, inwieweit Benutzer oder Gruppen auf eine Ressource zugreifen können. Der Besitzer einer Ressource kann diese für Jemanden freigeben, indem er ihn in die „access list“ des Ressourcenprofils aufnimmt und ihm dort eine Berechtigung zuweist [IBM09, Seite 86]. Für alle anderen Benutzer wird mit der „universal access authority“ eine allgemeine Berechtigung vergeben. Bei Data-Sets kann festgelegt werden, welches Programm der Benutzer für den Zugriff verwenden muss, indem der Programmname dem Eintrag hinzugefügt wird. Die Berechtigungen sind:

- `NONE` Keinen Zugriff
- `EXECUTE` Darf ausgeführt werden (bei Programmen)
- `READ` Lesezugriff
- `UPDATE` Lese- und Schreibzugriff
- `CONTROL` Berechtigung gleicht VSAM-Kontrollpasswort (Nur VSAM-Data-Sets)
- `ALTER` Volle Kontrolle

Einzelne, häufig benutzte Daten (zum Beispiel Hilfeseiten), verursachen unnötig redundante Zugriffe auf die RACF-Datenbank. Um dies zu vermeiden kann der Zugriff durch einen Eintrag in die „global access checking table“ für alle Benutzer geregelt werden. Dabei ist kein erneuter Datenbankzugriff nötig, weil diese Informationen beim Ressourcenzugriff sehr früh geprüft und im Arbeitsspeicher gehalten werden.

2.3.1. RACF-Sicherheitsklassen

Um den Zugriff auf besonders wichtige Daten noch besser kontrollieren zu können, empfiehlt es sich, Benutzer und Daten in Sicherheitsklassen einzuordnen. Folgende Klassifikationen können in RACF-Profilen eingetragen werden:

- *Sicherheits-Level* Ein auf dem System definierter Name, der einem numerischen Grad an Sicherheit entspricht, je höher die Nummer, desto höher das Sicherheitslevel.
- *Sicherheits-Kategorie* Ein auf dem System definierter Name, der eine Abteilung oder einen Bereich innerhalb einer Organisation repräsentiert, die/der in sich ähnliche Sicherheitsanforderungen hat.
- *Sicherheits-Etikett (Label)* Eine auf dem System definierte Kombination aus Sicherheitslevel und beliebig vielen Sicherheitskategorien.

Damit ein Benutzer auf eine klassifizierte Ressource zugreifen kann, muss er mindestens das Sicherheits-Level und die Sicherheits-Kategorie(n) der Ressource innehaben. Die Zugriffskontrolle geschieht dabei mithilfe des Vergleichs von entweder Levels und Klassen oder Labels.

2.4. Vergleich mit Ressourcenverwaltung unter UNIX

Unter UNIX versteht man Mehrbenutzer-Betriebssysteme, die ihren Ursprung im UNIX-System der Firma „Bell Laboratories“ haben oder dessen Konzepte umsetzen [Tan14, Kap.10.1]. Darunter fallen fast alle Open-Source-Betriebssysteme. Einige dieser Konzepte werden in diesem Abschnitt erläutert und mit ihren Äquivalenten unter z/OS verglichen. Die Abschnitte über die Dateisysteme sollen veranschaulichen und grundlegendes Verständnis vermitteln. Sie sind nicht vollständig, zur Vereinfachung wurden vor allem hardwarenahe Details weggelassen.

z/OS-Dateisystem Das z/OS-Dateisystem ist Record-orientiert. Das heißt, auf Speichermedien liegt kein durchgehender Strom von Bytes, der durch Steuerzeichen getrennt werden muss, sondern für jedes Record ist die Anzahl an Bytes festgelegt [EOO05, S.204]. Records sind in Data-Sets zusammengefasst. Diese können sequenziell sein, das heißt, wenn auf eine bestimmte Information zugegriffen werden soll, muss das ganze Data-Set durchgegangen werden,

bis der Eintrag gefunden wird. Oder ein Data-Set ist partitioniert, das heißt, es besteht aus einem „directory“ (engl. Verzeichnis) und einem oder mehreren „members“ (engl. hier: Einträgen). Es kann dann gezielt auf einen bestimmten Eintrag zugegriffen werden. Es gibt viele Methoden des Speicherns und Lesens von Data-Sets („Access Methods“), die Gebräuchlichste ist das sequenzielle Ablegen. Damit leichter auf Data-Sets zugegriffen werden kann, können sie katalogisiert werden. Data-Sets können dann mit ihrem Namen angesprochen werden und es ist belanglos, auf welchem Medium sie gespeichert sind.

UNIX-Dateisystem Dass in UNIX viele Ressourcen über (Pseudo-)Dateien angesprochen werden hat grundlegende Auswirkungen auf den Ressourcenschutz. Wer zum Beispiel Zugriff auf `/dev/cdrom` hat, kann das CD-Laufwerk benutzen (Veranschaulichendes Beispiel, nicht bei allen unixoiden Betriebsystemen verwendet). Auch Festplatten, Netzwerkprotokolle, Drucker und vieles mehr werden so angesprochen. Im Falle von UNIX spricht man nicht von Records, sondern von Files (engl. Dateien). Sie haben eine variable Länge und sind in Ordnern zusammengefasst. Alle Daten und Ordner in einem UNIX-Dateisystem sind logisch als hierarchischer Baum strukturiert, also dem Wurzelverzeichnis "/" untergeordnet (siehe Abbildung 2.2). Auch andere Datenträger müssen als Verzeichnis im Datei-Baum eingehängt werden. Im Gegensatz zu den Namen von Data-Sets in z/OS wird im UNIX-Dateisystem zwischen Groß- und Kleinschreibung unterschieden. Als absoluten Pfad einer Datei bezeichnet man den Pfad im Dateibaum vom Wurzelverzeichnis bis zur Datei. Der relative Pfad bezeichnet den Pfad von einer bestimmten Datei im Baum zu einer anderen.

UNIX-Benutzer werden mithilfe des *username* (engl. Benutzername) oder der *userid* „UID“ identifiziert. Beides wird im Klartext in der Datei `/etc/passwd` gespeichert. Die Benutzerpasswörter werden verschlüsselt in der Datei `/etc/shadow` abgelegt. Beide Dateien können nur vom Superuser verändert werden.

Superuser auch als *root* bezeichnet ist ein Benutzer mit uneingeschränktem Systemzugriff. Die Rechte des Superusers hat jeder Nutzer mit der UID 0 [RAB+06, S.87]. Er kann auf alle Dateien und Programme zugreifen, egal welche Zugriffsrechte dafür vergeben sind. Der Superuser existiert in der Regel auf allen UNIX-Systemen und wird für Aufgaben der Administration und Systempflege genutzt. Seine Befugnisse für Ressourcenverwaltung sind vergleichbar mit einem z/OS-Benutzer mit dem Attribut *SPECIAL*.

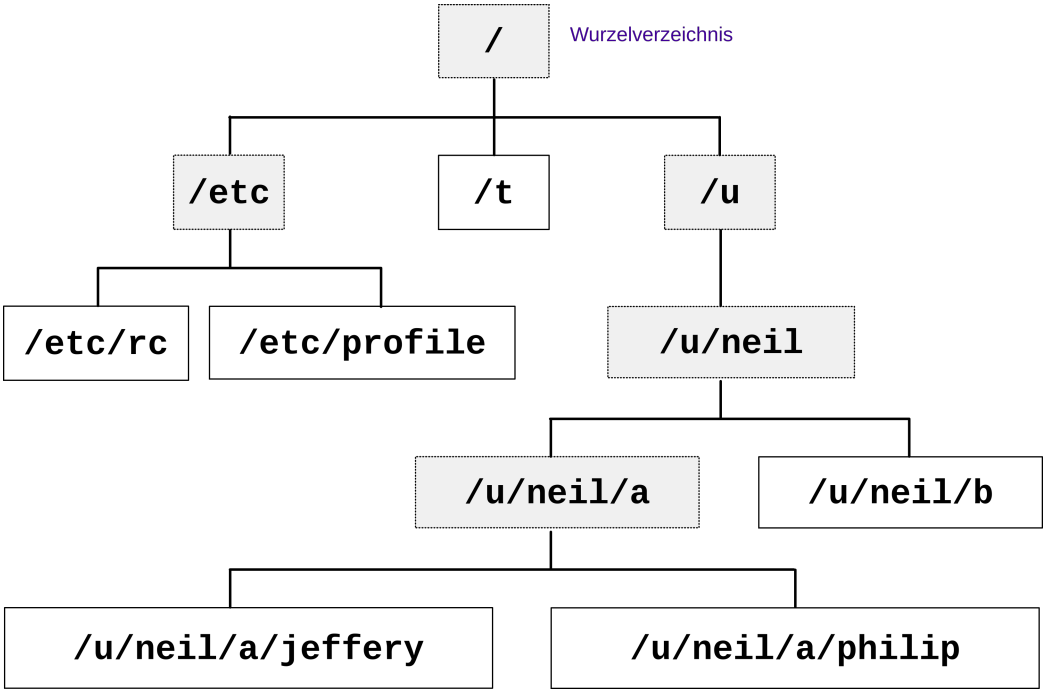

Abbildung 2.2.: Hierarchisches Dateisystem in UNIX Alle Dateien und Verzeichnisse sind dem Wurzelverzeichnis untergeordnet. Helle Boxen sind Dateien, schattierte sind Verzeichnisse.
[RAB+06, Kap.1.1.5]

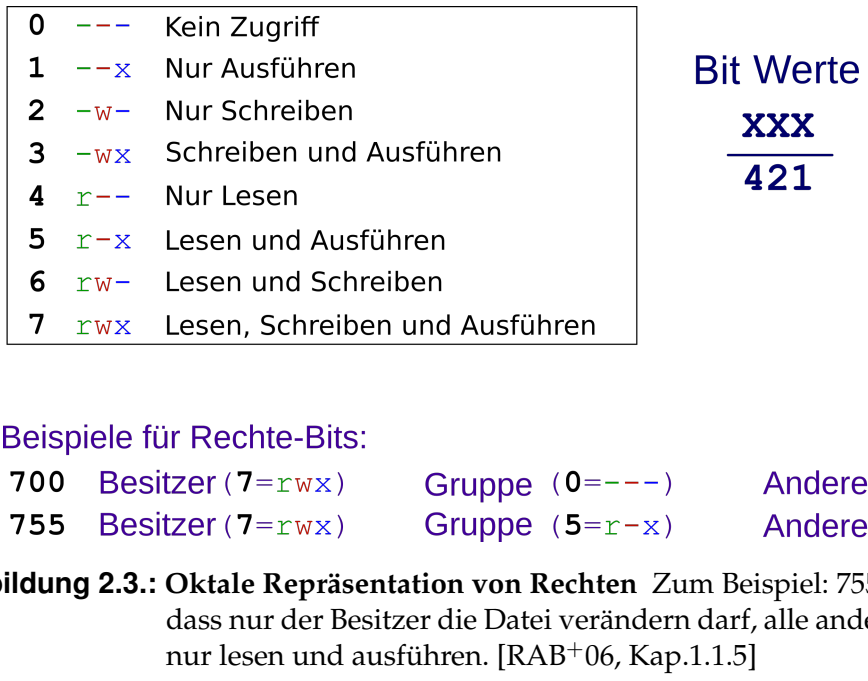

Abbildung 2.3.: Oktale Repräsentation von Rechten Zum Beispiel: 755 repräsentiert, dass nur der Besitzer die Datei verändern darf, alle anderen dürfen sie nur lesen und ausführen. [RAB+06, Kap.1.1.5]

Unix-Gruppen UNIX-Benutzer können in Gruppen eingeteilt werden. Wie in z/OS gehört jeder Benutzer mindestens zu einer Gruppe (seiner Primärgruppe, sie wird in `/etc/passwd` zugewiesen), kann jedoch zu beliebig vielen Gruppen gehören. Gruppen werden über den *group name* (engl. Gruppenname) oder die *Gruppen-ID* „GID “ identifiziert. Diese Informationen werden in der Datei `/etc/group` gespeichert.

Ressourcenschutz Da in UNIX alles eine Datei ist (insbesondere Programme), schützt man Ressourcen, indem man Zugriffsrechte für ihre Daten und Verzeichnisse festlegt. Für jede Datei und jedes Verzeichnis in UNIX können 3 Rechte jeweils für den Besitzer, seine Gruppe und alle Anderen vergeben werden. Die Rechte sind:

- *Lesen* Den Inhalt einer Datei lesen bzw. Inhalt eines Verzeichnisses anzeigen.
- *Schreiben* Den Inhalt einer Datei verändern oder Entfernen bzw. Erstellen und Löschen von Dateien in einem Verzeichnis.
- *Ausführen* Inhalt einer Datei ausführen bzw. in ein Verzeichnis wechseln.

In Abbildung 2.3 wird die oktale Rechte-Repräsentation erläutert, mit der Rechte oft dargestellt werden. Die Rechte werden dezentral für jede Datei abgespeichert. Der Standard-UNIX-Ressourcenschutz bietet, im Gegensatz zu RACF, per

se keine Möglichkeit zur Ressourcen- und Sicherheitsklassifizierung. Mit dem „Security-Enhanced Linux“ (engl. sicherheitsverbessertes Linux, kurz SELinux) existiert jedoch eine Erweiterung für den Linux-Kernel, die diese Funktionen nachliefert. Sie kann in vielen Linux-Derivaten nachinstalliert werden, in manchen ist sie sogar schon enthalten.

2.5. z/OS UNIX System Services

Im Serversegment hat IBM im Laufe der Zeit immer mehr Marktanteile an Linux-basierte Lösungen abgeben müssen, bis IBMs Serversparte schließlich eingestellt wurde. Um es deswegen Administratoren von unixoiden System zu ermöglichen, Mainframes in den Grundzügen zu verwalten, und vor allem auch, um es zu vereinfachen, UNIX-Programme auf dem Mainframe auszuführen, hat IBM die *z/OS UNIX System Services* „USS“ entwickelt. Früher „OMVS“ genannt, ist USS von „The Open Group“, dem Inhaber der Marke UNIX, als vollwertige UNIX-Implementation zertifiziert [RAB+06].
Der Hauptvorteil ist, dass UNIX-Programme von anderen Plattformen häufig nur neu auf dem Mainframe kompiliert werden müssen, um sie auf diesem ausführen zu können. Das ist möglich, da mit USS alle Komponenten eines UNIX-Betriebsystems, wie z.B. Laufzeitumgebungen und Programmbibliotheken in z/OS integriert wurden. Diese systemweite Integration ermöglicht sowohl das Verarbeiten von UNIX-Datein mit z/OS Programmen als auch die Interaktion von UNIX-Programmen mit z/OS Funktionen, beispielsweise Datenbankzugriffe auf DB2. Die für UNIX-Nutzer entwickelte Benutzerschnittstelle von USS ist die „z/OS UNIX Shell“. Diese Kommandozeilenumgebung ist UNIX-Shells nachempfunden, auf denen auch die heute am weitesten verbreitete UNIX-Shell „bash“ fußt [EOO05, S.189]. Um sich in die z/OS UNIX Shell einzuloggen, benötigt man einen TSO-Account mit OMVS-Segment.

Ressource	Beschreibung
JES	An das *Job Entry Subsystem* können Programme oder Skripte zur Ausführung gegeben werden.
IMS	*Information Management System*, ein Transaktionssystem für viele Arten von Informationen. Es bietet eine Datenbanken-Schnittstelle und einen Transaktionsmonitor.
CICS	*Customer Information Control System* Transaktionsverarbeitungsoftware, bietet Schnittstellen, um Transaktionsabläufe zu implementieren, auszuführen und zu überwachen.
SMS	*Storage Management Subsystem* verwaltet automatisch alle Platten- und Bandspeicher
DB2	Relationales Datenbankmanagementsystem. Kann große Datenmengen effizient, wiederspruchsfrei und dauerhaft speichern.
APPC	*Advanced Program to Program Communications* Protokoll mit dem Programme über ein Netzwerk kommunizieren können, vergleichbar mit TCP/IP
Data-Set	Standard-Art der Datenspeicherung in z/OS, eine Datei, die eine oder mehrere Records enthält. Kann katalogisiert werden, um per Name angesprochen zu werden anstatt über die Angabe des Speicherortes. In sequenziellen Data-Sets sind die Records nacheinander gespeichert und es kann nicht gezielt auf ein bestimmtes Element zugegriffen werden. In partitionierten Data-Sets gibt es Members, die ein oder mehrere Records enhalten und ein Verzeichnis, das den gezielten zugriff auf einen Member ermöglicht.
Catalog	Enthält Attribute und Speicherorte von Data Sets. Alle Kataloge und katalogisierten Data-Sets sind direkt oder indirekt durch andere Kataloge im Master-Katalog eingetragen.
DASD	*Direct-access Storage Device* Ein Festplattenlaufwerk, es ermöglicht im Gegensatz zu Bandspeichern den gezielten Zugriff auf einen bestimmten Abschnitt des Mediums.
Transaktionen	z.B. in CICS oder IMS

Tabelle 2.4.: Wichtige Ressourcen, die RACF schützen kann.

3. Stand der Technik

Um einige Arbeiten der Benutzerverwaltung mit RACF zu automatisieren, soll im Rahmen dieser Arbeit ein Skript entstehen, das diese Abläufe übernehmen kann. Das Skript soll in der Sprache *Restructured Extended Executor* „REXX“ geschrieben werden. In diesem Kapitel wird die Skriptsprache in ihren Grundzügen vorgestellt. Aspekte, die hinsichtlich des praktischen Teils der Arbeit wichtig sind, werden vertieft. REXX wurde von IBM für den Gebrauch auf Großrechnern, vor allem unter z/VM, entwickelt. Der Interpreter wurde jedoch auf viele andere Plattformen und Umgebungen portiert.

3.1. Grundlegende Konzepte

Normalerweise werden REXX-Skripte interpretiert, das heißt, es wird der für Menschen lesbare Code und kein übersetzter Programmcode ausgeführt. Jedoch besteht die Möglichkeit REXX-Skripte auch zu kompilieren, wenn das zu lösende Problem besondere Leistung, wie hohen Datendurchsatz oder schnelle Ausführung, erfordert.
Wie bei anderen Programmiersprachen sind viele Befehle in REXX englische Wörter, die den Zweck des Befehls repräsentieren. Ebenfalls sind REXX-Variablen typenfrei, das heißt, beim Anlegen einer Variable muss nicht festgelegt werden, ob sie beispielsweise Ganzzahlen, Reelle Zahlen oder Text speichert. Das erspart mitunter umständliche Typenumwandlungen, wie es sie in anderen Sprachen gibt. Wertzuweisungen haben die Form `Variable = Wert`. Auch sogenannte boolsche Variablen, die Wahrheitswerte `TRUE` (engl. wahr) und `FALSE` (engl. falsch) speichern und in vielen Sprachen Verwendung finden, gibt es in REXX nicht. Werden logische Bedingungen ausgewertet, so wird `TRUE` mit einer 1 und `FALSE` mit einer 0 repräsentiert. Die üblichen logischen Vergleichsoperatoren wie Größer als oder Ungleich stehen auch in REXX zur Verfügung.
Die einzigen andersartigen Variablen sind sogenannte Compound-Variablen (compound: engl. zusammengesetzt). Sie sind ähnlich dem Konzept der „Arrays“ in anderen Sprachen. Das heißt, sie speichern mehrere Werte, indem sie mehrere Variablen in sich vereinen. Namen von Compound-Variablen setzten

sich aus einem Stem (engl. Stamm) oder Präfix und einem Tail zusammen. Beide sind durch einen Punkt getrennt, zum Beispiel „stem.tail1“. Der Stem bleibt dabei stets gleich und identifiziert die Compound-Variable, der Tail identifiziert einen Eintrag oder ein Feld der Compound-Variable. Anders als bei den Arrays in anderen Sprachen müssen Einträge (Tails) nicht durchnummeriert sein. Tails können beliebig mit Zahlen und Buchstaben benannt werden und es gibt keine Reihenfolge der Tails einer Compound-Variable [IBM11c, 21].
Auch gibt REXX keine Formatierungen, wie Einrückungen vor. Da z/OS nicht auf Groß- und Kleinschreibung bei Befehlen achtet, ist dies auch im REXX-Code ohne Bedeutung. Lediglich bei Strings wird diese beachtet.

Funktionen und Subroutinen Eine Funktion ist ein gekapselter Ablauf von Befehlen, der genau einen String als Ergebnis zurückgibt. Subroutinen sind Funktionen, die kein Ergebnis zurückgeben müssen. Funktionen werden mit ihrem Namen, direkt gefolgt von einer öffnenden und einer schließenden Klammer, aufgerufen. Innerhalb der Klammern können Funktionen eine oder mehrere Werte oder Variablen übergeben werden. Damit kann der in der Funktion definierte Ablauf auf verschiedene Werte angewendet werden.
Subroutinen werden mit `CALL` (siehe 3.2) aufgerufen. Man unterscheidet zwischen internen (im selben Skript definierten), eingebauten (siehe 3.3) und externen (siehe 3.4) Funktionen und Subroutinen. Funktionen werden in REXX mit Sprungmarken (siehe 3.2) definiert. Der Unterschied zu reinen Sprungmarken ist, dass nach den Befehlen der Funktion mit dem Keyword `RETURN` zum Code nach dem Funktionsaufruf zurückgesprungen wird. Mit `PARSE ARG Parameter` können in Funktionen übergebene Parameter genutzt und mit `RETURN Rückgabewert` können Ergebnisse aus der Funktion zurückgegeben werden.

3.2. Verwendete Keyword Instructions

„Keyword Instructions“ (engl. Schlagwort-Befehle) sind aus einem oder mehreren vordefinierten Wörtern zusammengesetzte Anweisungen an den REXX-Interpreter. Sie steuern den Programmablauf oder greifen auf externe Schnittstellen oder Programme zu. Manche Keyword Instructions haben Unterwörter, die die Anweisung weiter spezifizieren, und können ineinander oder in sich selbst gekapselt werden.

SAY gibt den darauf folgenden String auf die Konsole aus. Manchmal ist ein String zu lang für eine Zeile im ISPF-Editor. Dann kann der String auf zwei

Zeilen verteilt werden, indem nach dem ersten Teil des Strings ein Komma gesetzt wird. Ein Beispiel dafür findet sich in 5.2.1.

PULL kann eine Eingabe des Benutzers in die dem Befehl folgende Variable speichern. `PULL` gibt dabei eine leere Zeile aus, in die der Benutzer schreiben kann. Seine Eingabe beendet er mit Enter.

DO gruppiert Befehle und strukturiert damit das Skript. Der Befehls-Block muss immer mit einem „END“ abgeschlossen werden, zum Beispiel `DO FOO=1 END`. Es existieren Unterwörter für `DO`, mit denen die Befehlsgruppe nur ausgeführt wird, wenn eine Bedingung erfüllt ist. Diese sind `WHILE` und `UNTIL`.
Bei `WHILE` wird die Gruppe ausgeführt, solange die nachfolgende Bedingung erfüllt ist. Die Bedingung wird vor der Ausführung geprüft. Bei `UNTIL` wird die Gruppe ausgeführt, bis die nachfolgende Bedingung erfüllt ist. Die Gruppe wird erst ausgeführt, dann die Bedingung geprüft.
Auch kann ein `DO`-Block wiederholt werden, indem als Unterwort die Anzahl der Wiederholungen (z.B. DO 5) übergeben wird. Mit `FOREVER` als Unterwort wird der Block unendlich oft wiederholt und kann nur mit dem Befehl `LEAVE` innerhalb des Blocks abgebrochen werden. Außerdem kann ein Block mit einer „Zählschleife“ wiederholt werden.
Zum Beispiel führt `DO INDEX=1 TO 4 BY 1` den Block viermal aus. Dabei wird `DO` eine beliebige Variable (hier `INDEX`) mit einem Startwert angehängt. Mit dem darauf folgenden `TO` wird der Wert festgelegt, den die Zählvariable erreichen muss, damit die Schleife abbricht. Mit `BY` wird die Schrittweite, das heißt der Wert, der bei jeder Wiederholung zur Zählvariable addiert wird, definiert. Ein Wiederholungs- kann mit einem Bedingungsunterwort kombiniert werden, wobei das Wiederholungsunterwort vor dem Bedingungsunterwort stehen muss.

IF bietet eine weitere Möglichkeit zur bedingten Ausführung. Zum Beispiel führt `IF I=1 THEN FOO=1 ELSE FOO=2` die auf `THEN` folgende Instruktion oder `DO`-Gruppe aus, wenn die Bedingung nach `IF` erfüllt ist. Andernfalls wird die Instruktion oder `DO`-Gruppe nach `ELSE` ausgeführt. Die Verwendung von `ELSE` ist optional.

PARSE überträgt Daten aus verschiedenen Quellen anhand gegebener Regeln in eine oder mehrere Variablen. Parse wird eine Quelle und eine Vorlage übergeben. Beispielsweise liefert die Quelle `ARG` die Strings, die dem Skript oder der Funktion beim Aufruf als Parameter übergeben wurden. Als Vorlage werden

Variablen, in denen die Strings gespeichert werden sollen, in der Reihenfolge übergeben, in denen auch die Strings in der Quelle stehen. Alle Quellen und Optionen finden sich in [IBM11c, S.65].

Sprungmarken sind eine Methode der direkten Programmflusskontrolle. Sie ermöglichen es, dass der Rexx-Interpreter an die Stelle der Marke im Skript springen und das Skript von dort aus weiter ausführen kann. Es ist dabei belanglos ob diese Stelle schon einmal oder noch nicht ausgeführt wurde.
Beim Aufruf von Unterfunktionen wird das Skript nach der Ausführung der Unterfunktion genau nach dem Aufruf weiterverarbeitet. Da ein reiner Sprung diese Rückkehr nicht vollzieht, kann er das Verstehen von unbekanntem Code sehr erschweren. Sprünge sind deswegen in den meisten modernen Programmiersprachen nicht mehr vorhanden.

SIGNAL ermöglicht Sprünge zu Sprungmarken. Mit `SIGNAL Sprungmarke` wird direkt zu einer Marke gesprungen. Mit `SIGNAL VALUE Variable` wird zu einer Marke gesprungen, deren Name in „Variable“ gespeichert ist. Mit `SIGNAL ON Fehlertyp NAME Sprungmarke` kann ein aufgetretener Fehler abgefangen werden. Es wird dann keine Fehlermeldung ausgegeben, sondern zur Sprungmarke gesprungen. Mit `SIGNAL OFF Fehlertyp` wird die Fehlerbehandlung für den übergebenen Typen wieder deaktiviert. Die Fehlertypen sind in [IBM11c, S.195] beschrieben.

CALL Mit `CALL Subroutine` kann eine Subroutine aufgerufen werden. Ähnlich wie bei `SIGNAL` ist es möglich mit `CALL ON Fehlertyp NAME Sprungmarke` Fehler abzufangen und anstelle der Ausgabe einer Fehlermeldung eine Subroutine aufzurufen. Mit `CALL OFF Fehlertyp` wird diese Fehlerbehandlung aufgehoben.

Zugriff auf Subsysteme Natürlich können von REXX-Skripten aus Befehle und Funktionen von z/OS Subsystemen genutzt werden. Soll nur ein einzelner Befehl an ein Subsystem gesendet werden, geschieht dies mit `ADDRESS Subsystemname Befehl`. Wenn außerhalb einer Zuweisung ein Ausdruck in Anführungszeichen gesetzt wird oder der REXX-Interpreter einen Ausdruck nicht als REXX-Befehl oder Zuweisung erkennen kann, gibt er den Ausdruck an das „Host Command Environment“ (engl. etwa: übergeordnete Befehlsumgebung) weiter. Standardmäßig ist das TSO.
Mit `ADDRESS Subsystemname` kann das standardmäßige „Host Command Environment“ für die aktuelle Skript-Ausführung geändert werden. Damit ein

Subsystem von `ADDRESS` angesprochen werden kann, muss es in der „Host Command Environment Table“ eingetragen sein.

3.3. Verwendete eingebaute Funktionen

Eingebaute Funktionen sind im REXX-Interpreter vordefinierte Funktionen, die in allen Skripten verwendet werden können.

DATATYPE(Variable,Datentyp) kann feststellen, ob die übergebene Variable den übergebenen Datentyp enthält. Wenn die Variable den Datentyp enthält, gibt `DATATYPE` 1 zurück, sonst 0. Die Auflistung der Datentypen findet sich in [IBM11c, S.95].

SUBSTR(String,n,Länge,Auffüllung) gibt den Teilstring von „String“ zurück, der beim n-ten Zeichen von String beginnt und die Länge „Länge“ hat. Wenn nötig, wird der Teilstring am Ende mit „Auffüllung“ aufgefüllt. Wird keine Länge angegeben, wird der String bis zum Ende zurückgegeben.

POS(Substring,String) gibt die Position von „Substring“ (bezogen auf den ersten Buchstaben) in „String“ zurück. Zum Beispiel: `POS('bc','abcd') = 2`. Gibt 0 zurück, falls „Substring“ nicht in „String“ enthalten ist.

3.4. Verwendete externe Befehle

Externe Befehle werden aus anderen Programmen in das Skript geladen. Die anderen Programme können auch in anderen Sprachen geschrieben sein, sie müssen lediglich eine Schnittstelle für den Zugriff aus REXX bieten. Diese Befehle werden der Einfachheit halber an Beispielen in Abschnitt 5.2 erklärt.

4. Entwurf

Im praktischen Teil dieser Arbeit werden einige Arbeiten der Benutzerverwaltung mit RACF automatisiert. Der Lehrstuhl „Technische Informatik “ bietet Module, in denen Studenten auf dem Mainframe arbeiten. Die entstehenden Skripte sollen das Anlegen und Löschen der Accounts für die Studenten vereinfachen. Im Folgenden werden die wichtigsten Aspekte der Skripte erläutert.

4.1. Grundlegendes

Da das Anlegen und das Löschen von Benutzer-Profilen klar trennbare Arbeitsabläufe sind, ist es sinnvoll auch zwei separate Skripte zu schreiben. Wenn auch absehbar ist, dass sich der Programmcode der Skripte an manchen stellen ähneln wird, erleichtert die Trennung die Bedienung. Das Skript zum Anlegen von Benutzerprofilen soll im Folgenden „ADDUSER“ und das Skript zum Löschen soll „DELUSER“ heißen.

4.1.1. Eingabe

Die Eingabe der Skripte soll mittels einer Kommandozeilenschnittstelle (siehe 2.1) geschehen. Die Skripte werden dem Benutzer aufeinanderfolgend Fragen zur auszuführenden Arbeit stellen. Durch die geführte Eingabe der Informationen soll sichergestellt werden, dass der Benutzer versteht, welches Detail des Arbeitsablaufes er gerade spezifiziert. So können falsche Angaben vermieden werden.
Eine alternative Eingabemöglichkeit wäre das Aufschreiben der Spezifikationen in einer Datei, die dem Skript übergeben wird. Naheliegend wäre es, in jede Zeile ein Paar aus Informationsname und spezifiziertem Wert zu schreiben (Key-Value-Paare). Eine Beispieldatei könnte bereitgestellt werden. Problematisch ist hierbei, dass der Benutzer sich an das vorgegebene Dateiformat halten müsste. Dies mindert den Bedienkomfort und ist sehr anfällig für Eingabefehler. Eine grafische Benutzeroberfläche ist wegen der Geradlinigkeit der Arbeitsabläufe unpassend und der Arbeitsaufwand ist unverhältnismäßig groß.

4.1.2. Ausgabe

Die Ausgabe der Skripte - im Sinne eines Dialogs mit dem Benutzer und Bestätigungen ausgeführter Aktionen - soll ebenfalls auf der Kommandozeile geschehen. Nachdem der Benutzer im Dialog die auszuführenden Aufgaben spezifiziert hat, soll ausgegeben werden, welche Aktionen mit welchen Parametern ausgeführt werden sollen. Der Benutzer wird gefragt, ob er diese Aktionen wirklich so ausführen möchte. Erst nach einer Bestätigung werden die Aktionen ausgeführt. Es ist außerdem vorgesehen, dass alle ausgeführten Tätigkeiten sowie das Datum und der ausführende Benutzer des Skriptes in einer Log-Datei gespeichert werden. Somit soll es später möglich sein zurückzuverfolgen, wer zu welchem Zeitpunkt welche Benutzer-Profile angelegt oder gelöscht hat.

4.2. Das Skript „ADDUSER“

Die grundlegende Funktion des Skripts ist das Anlegen von 1 bis n Benutzer-Profilen mit RACF. In der Vergangenheit bestanden die Benutzerprofile für Studenten aus einer Kombination aus Namenspräfix sowie einer laufenden Nummer, beispielsweise „PRAK031“. Dieses Vorgehen ist für das automatisierte Anlegen von Profilen sehr gut geeignet, da es unerheblich ist, wie die Benutzer-Profile heißen und man deswegen nicht jeden Benutzernamen einzeln definieren möchte. Das Skript soll den Ausführenden demnach nach einem Namenspräfix fragen und die Nummerierung automatisch umsetzen.
Da es möglich sein soll mehrere Benutzer auf einmal anzulegen, muss es eine Möglichkeit geben, die Anzahl zu spezifizieren. Naheliegend wäre es, den Skriptnutzer eine Anzahl N spezifizieren zu lassen und dann die Nutzer „Namenspräfix1“ bis „NamenspräfixN“ anzulegen. Dann müsste jedoch bei jeder Benutzung des Skriptes ein anderes Namenspräfix gewählt werden, da nach dem ersten Ausführen mindestens der Nutzer „Namenspräfix1“ schon existiert. Um das zu vermeiden sollte es möglich sein den Nutzer einen Zahlenbereich spezifizieren zu lassen. Beispielsweise würden für den Zahlenbereich 10 bis 20 die Nutzer „Namenspräfix10“ bis „Namenspräfix20“ angelegt werden.
Üblicherweise erhalten alle Benutzer das selbe Initialpasswort. Nach dem ersten Einloggen werden die neuen Nutzer angehalten, ihr Passwort zu ändern. Der Skriptnutzer sollte also ein Initialpasswort angeben können.
Beim Anlegen von RACF-Benutzern muss eine Standardgruppe angegeben werden. Das Skript sollte den Ausführenden also nach einer Standardgruppe für die Benutzer fragen. Wird keine Standardgruppe spezifiziert, so verwendet RACF die Standardgruppe des Nutzers, der den RACF-Benutzer anlegt. Im Regelfall ist der anlegende Benutzer ein Administrator und seine Standardgruppe die

Administratorengruppe. Da die Benutzer die Rechte ihrer Standardgruppe erben, sollte das Skript eine vordefinierte Gruppe für Praktikanten mit limitierten Rechten verwenden, sodass nicht zu viele Rechte vergeben werden, wenn keine Gruppe spezifiziert wird.
Damit die Benutzer sich in TSO einloggen können, muss in ihren RACF-Profilen das TSO-Segment definiert sein (siehe 2.2.1). Das Skript sollte TSO-Segmente in den Benutzerprofilen anlegen. Da in der Umgebung, in der das Skript eingesetzt werden soll, keine außergewöhnlichen Umstände - wie etwa Speicherknappheit - herrschen, können für die Parameter der TSO-Segmente Standardwerte verwendet werden. Um Data-Sets, die von den Benutzern angelegt werden, vor Zugriffen anderer zu schützen, sollte das Skript ein generisches RACF-Profil der Klasse „DATASET “ (siehe 2.3) für jeden Benutzer anlegen. Damit andere, unprivilegierte Benutzer keinen Zugriff auf die Data-Sets haben, müsste das Skript die UACC (siehe 2.3) in den generischen Profilen auf „NONE“ setzen.

4.3. Das Skript „DELUSER“

Im Praktikumsbetrieb mit Dutzenden neuen Studenten-Accounts jedes Jahr können sich über die Zeit viele obsolete Benutzer-Profile ansammeln. Den Überblick darüber zu behalten, welche Accounts noch benötigt werden und welche gelöscht werden können ist schwer. Das Skript „DELUSER“ soll es erleichtern, obsolete Profile zu finden und zu löschen.
Die Studenten-Accounts sind mit einem Namensmuster benannt. Suchanfragen, bei denen nur ein Teil des Suchbegriffes mit den RACF-Benutzernamen übereinstimmen muss, vereinfachen das Finden von Studenten-Accounts.
Ein Hinweis darauf, dass ein Account nicht mehr verwendet wird ist, dass er sehr lange nicht mehr genutzt wurde um sich mit ihm einzuloggen. Die Suchanfrage des Skriptes sollte also die Möglichkeit bieten nach Profilen zu suchen, mit denen sich lange nicht mehr eingeloggt wurde.
Die gefunden Profile sollten dem Ausführenden des Skriptes aufgelistet und erst nach einer Bestätigung gelöscht werden. Überlicherweise haben die Studenten nur Data-Sets mit ihrem Nutzernamen als High-Level-Qualifier (siehe 2.4) angelegt. Damit nach dem Löschen der Accounts keine unzugeordneten Data-Sets übrig bleiben, sollte das Skript nach dem Löschen der Accounts auch alle dazugehörigen Data-Sets löschen. Ebenfalls haben manche Nutzerprofile die Berechtigung die USS (siehe 2.5) zu benutzen und haben deswegen ein Nutzerverzeichnis in z/OS UNIX. Dieses sollte das Skript nach dem Löschen eines Profils ebenfalls entfernen.

5. Implementierung

Dieses Kapitel behandelt das Resultat der Implementation. Zunächst wird mit Diagrammen der grobe Ablauf der Skripte skizziert und anschließend wird vertiefend der Programmcode erklärt. Abschließend werden aufgetretene Probleme und verwendete Lösungen behandelt.

5.1. Skript-Abläufe

Der grobe Ablauf des Skriptes ADDUSER ist in Abbildung 5.1 dargestellt. DELUSER wird in Abbildung 5.2 verbildlicht.

5.2. Code-Erklärungen

Im Folgenden wird der Skriptcode erklärt, eine vollständige Ablaufbeschreibung findet sich im Kapitel 6. Die Syntax der verwendeten REXX-Befehle wird im Kapitel 3 behandelt, hier ihre Anwendung im Skript. Nicht relevante oder an anderer Stelle erklärte Codestellen werden durch „...“ ersetzt. Manche der Befehle sind funktional zusammenhängend, jedoch im Code getrennt. Zur Erklärung wird der dazwischen stehende Code ausgelagert und mit „Teilcode n“ markiert. Die Erklärung steht jeweils unter den Code-Abschnitten. Der vollständige Skriptcode findet sich im Anhang A.1 und A.2. Da z/OS bei Befehlen nicht auf Groß- und Kleinschreibung achtet, ist dies auch im REXX-Code nicht von Bedeutung. Lediglich bei Strings wird die Groß- und Kleinschreibung beachtet.

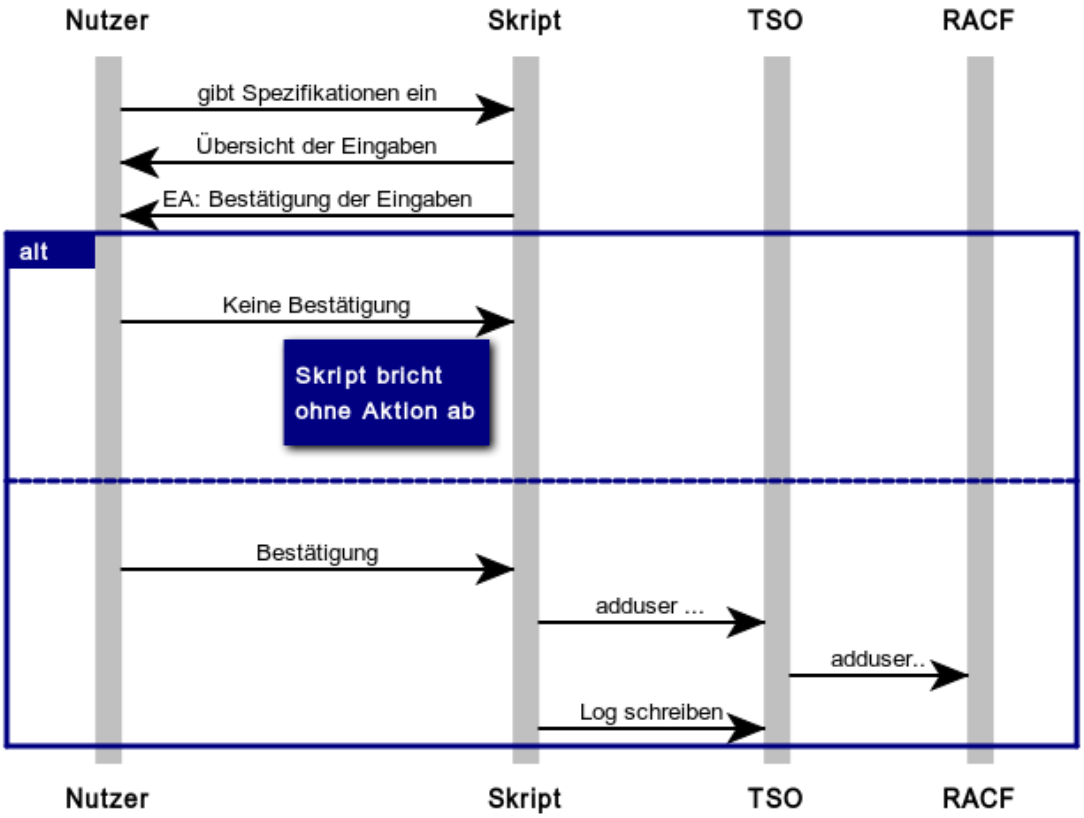

Abbildung 5.1.: Ablauf des Skriptes ADDUSER
Abkürzungen: EA (Eingabeaufforderung), alt (Alternativen)

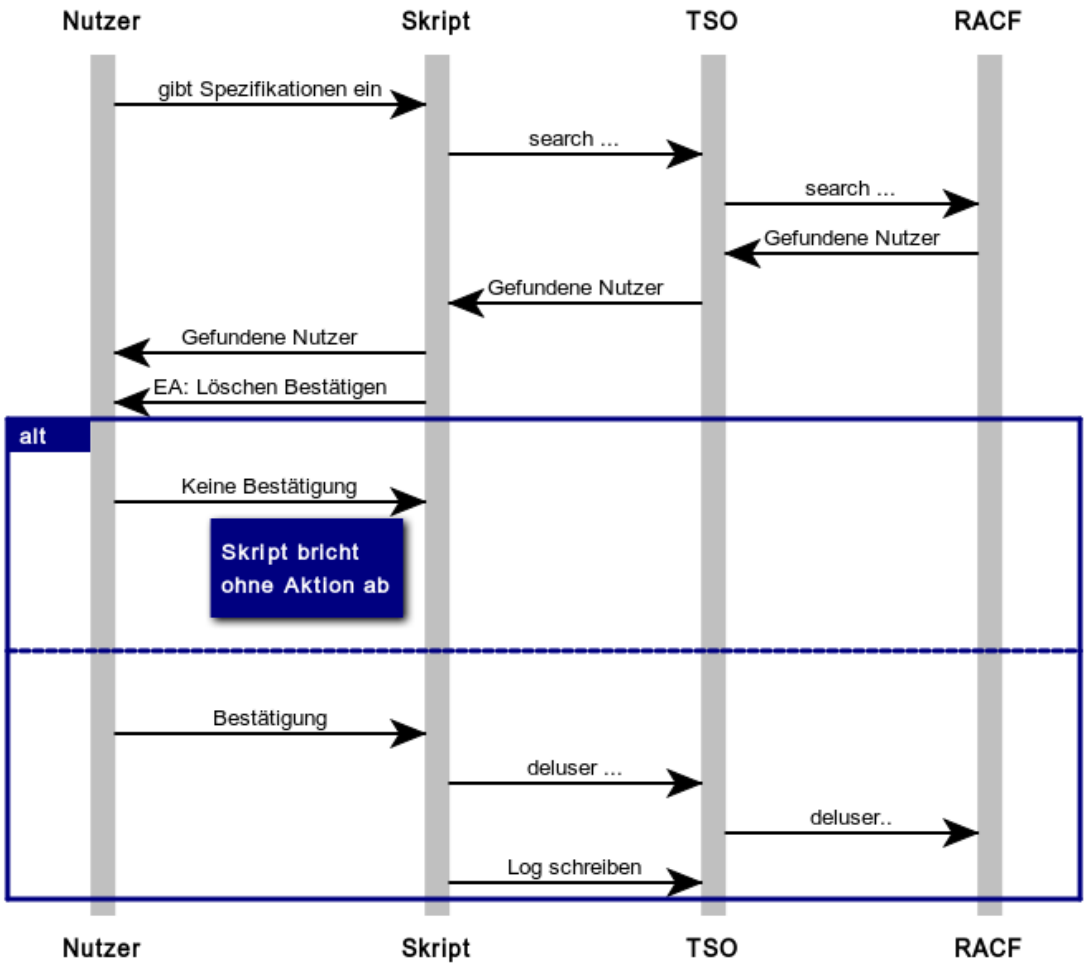

Abbildung 5.2.: Ablauf des Skriptes DELUSER
Abkürzungen: EA (Eingabeaufforderung), alt (Alternativen)

5.2.1. ADDUSER

Im Folgenden wird der Skriptcode des Skriptes „ADDUSER“ erklärt.

```
"PROFILE NOPREFIX"
```

Zu Begin wird das automatische Hinzufügen des eigenen Nutzernamens als „High Level Qualifier“ deaktiviert, siehe 5.3

```
...
FILEERROR:
say 'Bitte einen Dataset-Member fuer das Log angeben: '
...
pull filePath
Teilcode 1
SIGNAL ON ERROR NAME FILEERROR
"ALLOCATE DATASET("FILEPATH") FILE(FILEDS) OLD REUSE"
SIGNAL OFF ERROR
```

Das Skript fordert den Nutzer auf ein Dataset-Member für das Log anzugeben und liest Diesen dann von der Konsole ein. Mit `FILEERROR:` wird eine Sprungmarke gesetzt. Mit `SIGNAL ON ERROR NAME FILEERROR` wird definiert, dass bei einem Fehler vom Typ `ERROR` zu dieser Marke gesprungen werden soll. Wenn nun ein Fehler während des Allozierens des Dataset-Members auftritt, springt der Interpreter zurück zur Marke und ein anderes Dataset-Member kann eingegeben werden.
Mögliche Fehler beim Allozieren sind Tippfehler im Dataset-Member oder dass das angegebene Member bereits mit Exlusivzugriff verwendet wird. Das Allozieren geschieht mit dem TSO-Befehl `ALLOCATE DATASET( FILEPATH ) FILE(FILEDS) OLD REUSE`. Der Befehl ist in Anführungszeichen gesetzt, damit er an TSO weitergeben wird, siehe 3.2. `FILEPATH` ist das zu allozierende Dataset-Member. Die Anführungszeichen werden für die Variable unterbrochen, damit der Interpreter die Variable durch ihren Wert ersetzt und mit an TSO übergibt. `FILEDS` ist der Dateibezeichner, über ihn kann nach dem Allozieren im Skript auf den Member zugegriffen werden. Der Allokations-Typ `OLD` sagt aus, dass das Data-Set bereits existiert und dass der Zugriff exklusiv sein soll. `REUSE` legt fest, dass das Data-Set, falls es gerade verwendet wird, freigegeben und neu alloziert werden soll.
`SIGNAL OFF ERROR` hebt die mit `SIGNAL ON ...` gesetzte, bedingte Sprunganweisung auf. Kommt es danach zu einem Fehler vom Typ `ERROR`, wird nicht wieder zur Eingabe des Dataset-Members gesprungen.

```
if filePath == "" then
  do
    PARSE SOURCE . . . . PATH .
    filePath = PATH||"(LOG)"
    say 'Standard-Dataset-Member wird verwendet: 'filePath
  end
```

Dieser Abschnitt wird an die Stelle von „Teilcode 1“ im letzten Code-Abschnitt eingesetzt. Er sorgt dafür, dass wenn für das Dataset-Member nichts eingegeben wurde (die Variable `filePath` also dem leeren String (““) entspricht), ein Standard-Dataset-Member verwendet wird.
Um Fehler beim Anlegen eines neuen Data-Set zu vermeiden, wird das Log standardmäßig in ein neues Member des Data-Sets, aus dem das Skript geladen wurde, geschrieben. Dazu wird der REXX-Funktion `PARSE` die REXX-Systemvariable `SOURCE` übergeben. Sie enthält Informationen über die Herkunft, Art und Umgebung des ausgeführten Skripts. Von den sechs in `SOURCE` enthaltenen Strings wird nur der fünfte (das Data-Set aus dem das Skript geladen wurde) benötigt. Deswegen wird nur ihm eine Variable (`PATH`) zugewiesen. Für alle anderen ist es nötig einen Punkt als Platzhalter einzufügen. Der Data-Set-Name in `PATH` wird dann mit dem Member-Namen (`LOG`) konkateniert und in `filePath` gespeichert.

```
do until namePrefix \== ""
  say 'Bitte Namensmuster eingeben (z.B. "prak") ',
      'Die Nummer des Nutzers wird ',
      'angehaengt. z.B. prak1'
  pull namePrefix
end
```

Hier wird der zu verwendende Namensprefix vom Nutzer erfragt. Der Dialog wird so lange wiederholt, bis der Nutzer einen Prefix angegeben hat, da dies eine notwendige Angabe ist.

```
say 'Das Skript erstellt Nutzer in einem Zahlenbereich. z.B. prak10',
    'bis prak20.'
do until start > 0 & DATATYPE(start,W) == 1
  say 'Bitte Start des Zahlenbereiches angeben'
  pull start
end

do until ende >= start & DATATYPE(ende,W) == 1
  say 'Bitte Ende des Zahlenbereiches angeben'
  pull ende
end
user.0 = ende-start+1
```

Der Nutzer wird nun aufgefordert den Zahlenbereich, in dem die Nutzer-Profile angelegt werden zu spezifizieren. Der Start des Zahlenbereichs muss größer Null, das Ende des Zahlenbereichs größer oder gleich dem Start sein. Beide müssen ganze Zahlen sein, um Fehler beim Anlegen der Profile zu vermeiden. Dies wird sichergestellt, indem die Größenvergleiche in den Schleifenbedingungen mit der Funktion `DATATYPE` per logischem Und verknüpft werden. Mit dem Übergabeparameter `W` prüft `DATATYPE`, ob die übergebene Variable eine ganze Zahl enthält.
In der Compound-Variable (siehe 3.1) `user.` werden später die Namen der

angelegten Nutzer-Profile gespeichert, um sie in das Log zu schreiben. Hier wird die Anzahl der zu erstellenden Nutzer in das „nullte“ Feld von `user.` geschrieben. Viele Funktionen, die eine Compound-Variable erzeugen, speichern die Anzahl der Einträge in das „nullte“ Feld der Compound-Variable [IBM11c]. Dieses Vorgehen wurde dadurch zur stillen Konvention.

```
say 'Bitte die Standard-Gruppe fuer die Nutzer angeben: ',
  '(Ohne Eingabe Enter druecken, wenn PRAKT als Standardgruppe',
  'verwendet werden soll.)'
pull dfltgrp
if dfltgrp == "" then dfltgrp = "PRAKT"

say 'Bitte Initialpasswort fuer die Nutzer eingeben.',
    'Bei keiner Eingabe ist das Password der Name der Standardgruppe'
pull password
```

In diesem Abschnitt kann der User die Standard-Gruppe und das Initialpasswort für die anzulegenden Profile angeben. Falls der Nutzer keine Standard-Gruppe definiert, wird die Gruppe `PRAKT` verwendet. Das ist eine für Studenten auf dem System „SOLO“der Abteilung „Technische Informatik“ definierte Gruppe.

```
do  until  pos(antwort,'NJ')\==0  &  length(antwort)==1
  say "Es wurde Folgendes spezifiziert: "
  say "Anzulegende Nutzer: " namePrefix||start " bis ",
      namePrefix||ende
  say "Standardgruppe: " dfltgrp
  if password \== "" then say "Password: " password
                     else say "Initialpassword ist der Name der ",
                              "Standardgruppe"
  say "Sollen wirklich " user.0 " Nutzer angelegt werden?(J/N)"
  pull antwort
  antwort = space(antwort,0)
end
```

Alle Spezifikationen der anzulegenden Nutzerprofile werden nun zur Kontrolle aufgelistet und das Skript fragt, ob die Nutzer so angelegt werden sollen. Der Nutzer muss mit `N` für Nein antworten um das Skript abzubrechen. Antwortet er mit `J` für Ja, so werden die Nutzer angelegt. `pos(antwort,'NJ')\==0` sorgt dafür, dass eine gültige Antwort nur aus `N` oder `J` bestehen kann. Die logische Und-Verknüpfung mit `length(antwort)==1` in der Schleifenbedingung legt fest, dass eine gültige Antwort nur aus genau einem Zeichen bestehen darf. Sind nicht beide Bedingungen gleichzeitig erfüllt, so wird der Dialog wiederholt. `space(antwort,0)` entfernt eventuell eingegebene Leerzeichen aus dem Antwort-String.

```
zeiger = 0
if antwort == j then
  do
    do index=start to ende by 1
       zeiger = zeiger + 1
       /*Usernamen fuer das Log speichern*/
       user.zeiger = namePrefix||index
       say 'Erzeuge Nutzer' user.zeiger
       /* Den RACF Befehl zusammensetzten */
       command = "adduser" user.zeiger
       command = command || " TSO(ACCTNUM(ACCT) PROC(DBSPROC)",
       "MAXSIZE(6072) SIZE(5000) MSGCLASS(H) UNIT(SYSALLDA))"
       command = command || " OMVS(UID(1) HOME(/u/"user.zeiger"/)",
       "PROGRAM(/bin/sh))"
       command = command || " DFLTGRP("dfltgrp")"
       if password \== "" then
         do
           command = command || " PASSWORD("password")"
         end
       command
       Teilcode 2
     end
   Teilcode 3
  end
```

Hat der Skript-Nutzer mit `J` geantwortet, werden die Nutzer angelegt. Die innere `do`-Schleife zählt ihre automatische Zählvariable `index` vom Start bis zum Ende des Zahlenbereiches hoch. Mit jedem Schleifendurchlauf wird ein Nutzer angelegt. Damit die Profilnamen später leichter in das Log geschrieben werden können, werden sie in der Compound-Variable `user.` gespeichert.
Um die Profilnamen unabhängig vom Zahlenbereich in die ersten Felder der Variable schreiben zu können, wurde eine zusätzliche Zählvariable `zeiger` eingeführt. Sie wird mit 0 initialisiert und zu Beginn jedes Durchlaufs um 1 erhöht. Der zuerst anzulegende Profilname wird also in `user.1` gespeichert, der nächste in `user.2` und so weiter. Dies geschieht mit `user.zeiger = namePrefix||index`.
Gemäß der Angaben des Skript-Nutzers wird nun in jedem Schleifendurchlauf der RACF-Befehl zum Profilerzeugen zusammengesetzt und zunächst in der Variable `command` gespeichert. Zum Zusammensetzten wird `command` Schritt für Schritt mit einer Konkatenation aus sich selbst und dem nächsten Befehlsabschnitt überschrieben. Zunächst wird der eigentliche RACF-Befehl `ADDUSER` in `command` geschrieben, danach zusätzliche Befehlsoptionen. Diese sind das Segment für TSO, das Segment für OMVS(USS), die Standardgruppe und das Passwort. In USS wird allen Nutzern die UID 1 gegeben, da die automatische Vergabe von UIDs auf dem Testsystem nicht zur Verfügung steht.
Sobald der Befehl fertig zusammengesetzt ist, wird er mit der Befehlszeile `command` ausgeführt. Der Interpreter ersetzt dabei den Variablennamen mit

dem Inhalt der Variable. Da der Inhalt ein String ist, wird dieser an TSO weitergegeben, siehe 3.3.

```
else
  say 'Abgebrochen, keine Nutzer angelegt'
"FREE FILE(FILEDS)"
/*Profil prefix zuruecksetzen*/
"PROFILE PREFIX("USERID()")"
exit
```

Hat der Skript-Nutzer mit `N` geantwortet, werden keine Nutzer angelegt. In beiden Fällen wird vor dem Ende des Skriptes das Data-Set mit `FREE FILE(FILEDS)` freigegeben und die Vervollständigung von Data-Set-Namen mit `PROFILE PREFIX(USERID())` wieder aktiviert.

```
/* Top Generic Dataset anlegen und Zugriffsrecht geben*/
"ADDSD" user.zeiger||'.*' "UACC(NONE) GENERIC OWNER(ADMIN)"
"PERMIT" user.zeiger||'.*' "ACCESS(ALTER) ID(ADMIN)"
/* OMVS Verzeichnis anlegen und Rechte vergeben*/
unixcmd = "mkdir /u/"||user.zeiger
CALL BPXWUNIX unixcmd
unixcmd = "chown "||user.zeiger||" /u/"||user.zeiger
CALL BPXWUNIX unixcmd
unixcmd = "chmod 700 /u/"||user.zeiger
CALL BPXWUNIX unixcmd
```

Dieser Abschnitt wird an die Stelle von „Teilcode 2“ eingesetzt, hier wird der Data-Set-Schutz eingerichtet und ein USS-Verzeichnis (siehe 2.5) für den Nutzer erstellt. Mit den RACF-Befehlen `ADDSD` und `PERMIT` wird ein generisches RACF-Profil für die Data-Sets des aktuellen Nutzers angelegt und ihm, sowie der Admingruppe, Lese- und Schreibzugriff darauf gewährt.
Mit den folgenden Befehlen wird dem Nutzer ein Verzeichnis im Order für Nutzerverzeichnisse in USS angelegt (`mkdir`), er wird zum Besitzer dieses Ordners gemacht (`chown`) und die Rechte des Ordners werden so gesetzt, dass nur er Zugriff darauf hat (`chmod`, siehe auch 2.3). Dies geschieht, indem die entsprechenden UNIX-Befehle mithilfe von `BPXWUNIX` an USS geschickt werden. Es handelt sich dabei um eine „z/OS UNIX REXX Funktion“. Das sind Funktionen, die USS speziell für REXX-Skripte bereitstellt [IBM10, S.217].

```
/*Logging*/
MSG.1= 'Dies ist eine automatisch erstellte Logdatei.'
MSG.2 = 'Am',
  DATE(),
  'Um',
  TIME(),
  'wurden die nachfolgenden Accounts angelegt.'
MSG.3 = 'Das Skript wurde vom Benutzer',
  USERID(),
  'ausgefuehrt.'
MSG.4 = ' '
```

```
MSG.5 = 'Es wurde der folgende Befehl verwendet: '
MSG.6 = command
MSG.7 = SUBSTR(command,73)
MSG.8 = ' '
MSG.9 = 'Angelegte Nutzer:'
"EXECIO" 9 "DISKW FILEDS (STEM MSG."
"EXECIO" user.0 "DISKW FILEDS (STEM user."
"EXECIO 0 DISKW FILEDS (FINIS"
```

Dieser Abschnitt wird an die Stelle von „Teilcode 3“ eingesetzt, hier wird das Log erstellt. Dazu werden Informationstext, das aktuelle Datum, die Zeit, der Skript-Nutzer und der Befehl um den letzten Nutzer anzulegen in die Compound-Variable `MSG.` geschrieben. Der anschließende Befehl zum Schreiben in das Log setzt sich wie folgt zusammen: der eigentliche Befehl `EXECIO`, 9 ist die Anzahl zu verarbeitender Zeilen, `DISKW` bestimmt schreibenden Zugriff, `FILEDS` ist der Dateibezeichner mit dem das Data-Set alloziert wurde und `(STEM MSG.` legt fest, dass die Compound-Variable `MSG.` verarbeitet werden soll. Es handelt sich dabei um einen sogenannten „TSO-REXX-Befehl“. Das heißt, diese Befehle werden von TSO bereitgestellt und können nur von REXX-Skripten benutzt werden [IBM11c, Kap.10].
Analog dazu werden dann die angelegten Nutzer, die in der Variable `user.` gespeichert sind, in das Log geschrieben. Mit dem ersten schreibenden Befehl wurde das Data-Set implizit geöffnet. Abschließend wird es mit `EXECIO 0 DISKW FILEDS (FINIS` geschlossen. Eine Beispiel-Logdatei ist im Anhang A.3 zu finden.

5.2.2. DELUSER

Im Folgenden wird der Skriptcode des Skriptes „DELUSER“ erklärt. Der Beginn beider Skripte ist sehr ähnlich, weshalb der Anfang hier nicht wiederholt wird. Die Skripte unterscheiden sich, nachdem das Log-Data-Set alloziert wurde.

```
...
do until namePrefix \== ""
  say 'Bitte Namenspraefix (z.B. "prak") eingeben'
  pull namePrefix
end
do until age > -2 & DATATYPE(age,W) == 1
  say 'Zahl der Tage die sich die Nutzer nicht',
  'mehr eingeloggt haben sollen eingeben',
  ' (-1 wenn die Suchoption AGE nicht verwendet werden soll)'
  pull age
end
```

Der Skript-Nutzer wird aufgefordert ein Namenspräfix und die Tage der Abwesenheit der Nutzer für die Suche anzugeben. Da nicht ohne ein Präfix gesucht

werden kann, wird der Dialog so lange wiederholt bis ein Präfix eingegeben wurde. Bei den Tagen der Abwesenheit wird mit der Funktion `DATATYPE` und dem Übergabeparameter `W` sichergestellt, dass es sich um eine ganze Zahl handelt, da es sonst bei der Suche zu einem Fehler kommt.

```
CALL OUTTRAP "LINE.", "*"

/*Suchanfrage an RACF*/
if age \== -1 then
  do
    "SEARCH MASK("namePrefix") CLASS(USER) AGE("age")"
  end
if age == -1 then
  do
    "SEARCH MASK("namePrefix") CLASS(USER)"
  end
```

Mit `CALL OUTTRAP ...` werden alle Ausgaben von Subsystemen, die eine Folge von Befehlen des Skriptes sind, in die Compound-Variable `LINE.` umgeleitet. Jede ausgegebene Zeile erhält ein Feld in der Compound-Variable. Nun wird mit `SEARCH ...` eine Suchanfrage an RACF gestellt. Mit der Suchoption `MASK` wird festgelegt, dass Suchergebnisse mit dem übergebenen String beginnen müssen. `CLASS(USER)` sorgt dafür, dass die Ergebnisse nur Benutzer-Profile sind. Je nach dem ob die Tage der Abwesenheit angegeben wurden, wird der Suchbefehl mit oder ohne die Option `AGE` ausgeführt.

```
CALL OUTTRAP "OFF"
```

Nachdem der Suchbefehl ausgeführt wurde, kann die Umleitung der Ausgaben abgestellt werden und das Ergebnis der Suche steht in `LINE.`.

```
ERRORMSG = "ICH31005I NO ENTRIES MEET SEARCH CRITERIA"
if LINE.1 = ERRORMSG then
  do
    say "Keine Benutzer gefunden"
  end
```

Wird kein Nutzer gefunden, gibt RACF eine Fehlermeldung aus. Entspricht der erste Eintrag dieser Fehlermeldung, kann also eine entsprechende Meldung an den Skript-Nutzer ausgegeben werden.

```
else
  do
    SAY "Es wurden "LINE.0" Nutzer gefunden: "
    do index=1 to LINE.0 by 1
      say "Benutzer " LINE.index " gefunden."
    end
```

Wurden Nutzer gefunden gibt RACF sie Zeile für Zeile aus, jeder Treffer ist in einem Feld von `LINE.` gespeichert. Wie bei REXX üblich, enthält das nullte

Feld der Compound-Variable die Anzahl der Einträge der Variable. Dem Skript-Nutzer wird ausgegeben wie viele und welche Profile gefunden wurden.

```
do until pos(antwort,'NJ')\==0 & length(antwort)==1
  say "Sollen die Nutzer wirklich geloescht werden(J/N)?"
  pull antwort
  antwort = space(antwort,0)
end
```

Das Skript fragt, ob die Nutzer gelöscht werden sollen. Die Ablaufkontrolle ist analog zum Nutzerdialog in „ADDUSER“.

```
if antwort == J then
  do
    do index=1 to LINE.0 by 1
      say "Loesche Nutzer "LINE.index
      "DELDSD "LINE.index||'.*'

      Teilcode 2
      /* OMVS Dateien loeschen*/
      unixcmd = "rm -r /u/"||LINE.index
      CALL BPXWUNIX unixcmd
      "DELUSER "LINE.index
    end
    Teilcode 3
  end
```

Wenn der Nutzer mit `J` geantwortet hat, wird durch die Einträge von `LINE.` und damit über die gefundenen Nutzer iteriert. Für jeden Nutzer werden dann die folgenden Aktionen ausgeführt. Mit dem RACF-Befehl `DELDSD ...` wird zunächst das generische Data-Set-Profil (siehe 2.3) gelöscht. Anschließend wird mit `rm ...` das Nutzerverzeichnis in USS gelöscht. Dieser UNIX-Befehl wird mithilfe von `BPXWUNIX` an USS geschickt. Schließlich wird mit `DELUSER` das Nutzerprofil gelöscht.

```
"ISPEXEC LMDINIT LISTID(listid) LEVEL("LINE.index||'.*'")"
do forever
  "ISPEXEC LMDLIST LISTID(&listid) OPTION(LIST) DATASET(dsn)"
  if rc = 0 then
    do
      say "loesche:'"dsn"'"
      "ISPEXEC LMERASE DATASET('"dsn"')"
    end
  else
    leave
end
```

Dieser Abschnitt wird bei Teilcode 2 eingefügt. Hier werden alle Data-Sets, die den aktuellen Nutzernamen als „High-Level Qualifier“, HQL (also die Data-Sets des aktuellen Nutzers, siehe 2.4) tragen, gelöscht. Bei `ISPEXEC` handelt es sich um den Addressierungsbefehl für das Host Command Environment (siehe 3.3) „ISPF“. Das heißt, ISPF stellt Funktionen bereit, die nur von REXX-Skripten

aus benutzt werden können und mit diesem Befehl angesprochen werden [IBM11c, S.28]. Nun wird der ISPF-Befehl `LMDINIT` verwendet, um eine Liste mit Data-Sets zu erzeugen, die den aktuellen Nutzernamen als HQL tragen. Mit dem Übergabeparameter `LISTID` wird die ID der Liste spezifiziert, mit `LEVEL` das Data-Set-Namensmuster [IBM13a, S.123]. Mit dem an den Nutzer-Namen angehängten Punkt wird der HQL abgeschlossen, der folgende `*` ist ein Platzhalter.
Der ISPF-Befehl `LMDLIST` mit der Option `LIST` gibt nun mit jedem Aufruf das nächste Element der Liste aus, bis alle Elemente ausgegeben wurden. Übergeben werden die ID der Data-Set-Liste und mit `dsn` die Variable, über die auf das aktuelle Listenelement zugegriffen werden kann. Die Liste ist dabei ein Objekt in ISPF, nicht in REXX. Deswegen muss beim Zugriff auf die Liste der Referenzoperator `&` aus der ISPF-Kommando-Syntax verwendet werden.
In der speziellen REXX-Variable `RC` steht der Rückgabewert des zuletzt ausgeführten Befehls. Wenn `LMDLIST` ein Element aus der Liste ausgegeben hat, ist `RC`=0. Konnte jedoch kein Element ausgegeben werden, ist `RC`=8. Wurde ein Data-Set aus der Liste ausgegeben, so wird es mit `LMERASE` gelöscht. Ist das Ende der Liste erreicht, so wird die Dauerschleife abgebrochen.

In Teilcode 3 wird das Log erstellt. Der Ablauf ist analog zum Erstellen des Logs in „ADDUSER“und wird deswegen nicht wiederholt.

```
    else
      say 'Abgebrochen, keine Nutzer geloescht'
end
```

Antwortet der Nutzer mit `N` werden keine Nutzer gelöscht.

```
"FREE FILE(FILEDS)"
/*Profilprefix zuruecksetzen*/
"PROFILE PREFIX("USERID()")"
exit
```

In beiden Fällen wird vor dem Terminieren des Skriptes das Data-Set-Member für das Log freigegeben und das automatische Vervollständigen von Data-Set-Namen wieder aktiviert.

5.3. Probleme bei der Implementation

Dieser Abschnitt behandelt die Probleme, die während der Implementation auftraten und wie sie gelöst wurden.

Kein automatischer Zeilenumbruch beim Schreiben in Data-Set-Member Der heutige Standard-Zugriffsweg auf TSO sind Emulatoren des IBM 3270 Terminals [EOO05]. Die gängigste Bildschirmauflösung ist dabei 80 Zeichen breit und 45 Zeichen hoch. Dadurch kann der z/OS Standard-Editor „ISPF-Editor“ nur 72 Zeichen pro Zeile darstellen. Im Skript „ADDUSER“ wird der über 72 Zeichen lange Befehl zum Nutzer anlegen in das Log geschrieben. Da der Befehl zum Schreiben in ein Data-Set `EXECIO` keinen automatischen Zeilenumbruch bietet, werden alle Zeichen nach dem zweiundsiebzigsten verworfen. Um den gesamten Befehl zu speichern, werden mithilfe der REXX-Funktion `SUBSTR` alle Zeichen ab dem dreiundsiebzigsten in die nächste Zeile geschrieben.

Unvollständige Data-Set-Namen werden vervollständigt Data-Set-Namen gelten als „fully-qualified“ (engl. hier: vollständig), wenn sie aus einem Nutzernamen als Präfix, einem vom Nutzer spezifizierten „qualifier“ (engl. hier: Kennzeichen) und einem beschreibenden Kennzeichen bestehen[IBM13d, S.22]. Zum Beispiel: `JOHN.REXX.EXEC` . Wenn Data-Set-Namen nicht aus 3 Kennzeichen bestehen, geht TSO davon aus, dass auf ein eigenes Data-Set zugegriffen werden soll und fügt den eigenen Nutzer-Präfix hinzu.
Im Skript „DELUSER“ wird nach allen Data-Sets der Nutzer gesucht, deswegen wird zur Suche nur der Namens-Präfix der zu löschenden Nutzer und ein Platzhalter verwendet. Der gesuchte Data-Set-Name ist also nicht vollständig und wird von TSO um den Namens-Präfix des Skript-Nutzers ergänzt, wodurch die gesuchten Data-Sets nicht gefunden werden können. Das automatische Vervollständigen kann mit dem TSO-Befehl `PROFILE NOPREFIX` deaktiviert werden[IBM13d, 169].

6. Auswertung

In diesem Kapitel wird untersucht, ob das Ziel der Implementation erreicht wurde. Mithilfe von Bildschirmausschnitten von Testläufen wird die Funktionalität der Skripte gezeigt. Gleichzeitig stellt die Erklärung der Abläufe eine Bedienungsanleitung dar.

6.1. Anwendungen

Anhand von Bildschirmausschnitten aus ISPF wird im Folgenden die Funktionalität der Skripte gezeigt. Die rote Schrift ist hierbei die Ausgabe des Skriptes, die grüne Schrift die Nutzereingaben. Alle Befehle werden vom „ISPF Command Shell"-Bildschirm (ISPF-Menupunkt 6) aus ausgeführt. Die eingegebenen Befehle werden an TSO weitergegeben, jedoch stehen vom Skript „DELUSER" verwendete ISPF-Programmbibliotheken zur Verfügung. Es wird davon ausgegangen, dass sich die Skripte im Data-Set `REXX.EXEC` befinden.

6.1.1. ADDUSER

Beispielhaft wird im Folgenden der Nutzer „TEST1" angelegt.

```
  Menu  List  Mode  Functions  Utilities  Help
------------------------------------------------------------------------------
                              ISPF Command Shell
Enter TSO or Workstation commands below:

===> LISTUSER TEST1

ICH30001I UNABLE TO LOCATE USER    ENTRY TEST1
***
```

In der Ausgangssituation ist kein Nutzerprofil mit dem Namen „TEST1" vorhanden. Der RACF-Befehl `LISTUSER` zeigt den Inhalt von RACF-Nutzerprofilen

an. Da das angefragte Profil nicht existiert, antwortet RACF mit einer Fehlermeldung.

```
  Menu  List  Mode  Functions  Utilities  Help
 ------------------------------------------------------------------------------
                              ISPF Command Shell
 Enter TSO or Workstation commands below:

 ===> EXEC REXX.EXEC(ADDUSER) EXEC

 Mit diesem Skript koennen RACF-Benutzer angelegt werden
 Bitte einen Dataset-Member fuer das Log angeben:
 Format: HLQ.LLQ(Membername) Enter druecken fuer Standart Dataset

 Standard-Dataset-Member wird verwendet: KREUSCH.REXX.EXEC(LOG)
 Bitte Namensmuster eingeben (z.B. "prak")  Die Nummer des Nutzers wird angehaen
 gt. z.B. prak1
TEST
 Das Skript erstellt Nutzer in einem Zahlenbereich. z.B. prak10 bis prak20.
 Bitte Start des Zahlenbereiches angeben
1
 Bitte Ende des Zahlenbereiches angeben
1
 Bitte die Standard-Gruppe fuer die Nutzer angeben:  (Ohne Eingabe Enter druecke
 n, wenn PRAKT als Standardgruppe verwendet werden soll.)

 Bitte Initialpasswort fuer die Nutzer eingeben. Bei keiner Eingabe ist das Pass
 word der Name der Standardgruppe

 Es wurde folgendes Spezifiziert:
 Anzulegende Nutzer:  TEST1  bis  TEST1
 Standardgruppe:  PRAKT
 Initialpassword ist der Name der  Standardgruppe
 Sollen wirklich  1  Nutzer angelegt werden?(J/N)
```

Um das Skript aus ISPF heraus zu starten geht man in den „Command“ Subscreen, indem man 6 in die Optionszeile eingibt und mit Enter bestätigt. Dort gibt man folgenden Befehl in die Befehlszeile ein und bestätigt mit Enter: `EXEC REXX.EXEC(ADDUSER) EXEC`. Nach einer kurzen Information über die Funktion des Skriptes wird man aufgefordert, ein Data-Set-Member für die Logdatei zu spezifizieren. In der Logdatei werden zur Nachvollziehbarkeit alle relevanten Informationen über die Ausführung des Skriptes gespeichert.
Auf manchen ISPF-Bildschirmen beginnt die Ausgabe des Skriptes am unteren Ende des Bildschirmes. Bevor das Data-Set-Member wie aufgefordert spezifiziert werden kann, muss dann zunächst der Bildverlauf mit Enter weitergeschoben werden. Dies wird mit den Zeichen `***` in der untersten Zeile des Bildschirmes signalisiert.
Nun wird das Data-Set-Member im Format HLQ.LLQ(Membername) eingege-

ben. Es ist auch möglich ohne eine Eingabe Enter zu drücken. Dann wird in dem Data-Set, in dem sich das Skript befindet ein Member namens „Log“ angelegt und verwendet. In beiden Fällen gibt das Skript den Namen des verwendeten Data-Set-Members in einer Kontrollmeldung aus.
Als nächstes fordert das Skript auf, ein Namensprefix für die anzulegenden Nutzer zu spezifizieren. Die Nutzernamen setzten sich aus diesem Prefix und einer fortlaufenden Nummer aus einem Zahlenbereich zusammen, beispielsweise prak1 bis prak9. Nachdem ein Namensprefix eingegeben wurde, muss nun nacheinander der Beginn und das Ende des Zahlenbereiches spezifiziert werden, in dem die Nutzer angelegt werden sollen. Danach fordert das Skript auf, eine Standardgruppe für die Nutzer anzugeben. Wird ohne eine Eingabe Enter gedrückt, so wird eine vordefinierte Gruppe mit limitierten Rechten verwendet. Anschließend fordert das Skript auf, ein Initialpasswort für die Nutzer zu vergeben. Wird ohne eine Eingabe Enter gedrückt, so wird der Name der Standardgruppe der Nutzer als Initialpasswort verwendet.
Nachdem damit alle Informationen erfasst wurden, gibt das Skript diese zur Kontrolle noch einmal aus und fragt, ob die spezifizierten Nutzer wirklich angelegt werden sollen.

```
n
 Abgebrochen Keine Nutzer angelegt
 ***
```

Wenn keine Aktion ausgeführt werden soll, muss `N` eingegeben werden. Das Skript terminiert dann ohne Nutzer anzulegen.

```
j
 Erzeuge Nutzer TEST1
 ***
```

Wenn die Aktionen ausgeführt werden sollen, muss `J` eingegeben werden. Das Skript legt dann die Profile an und bestätigt dies jeweils. Alle relevanten Informationen über den Vorgang werden in die Logdatei geschrieben und das Skript terminiert.

```
  Menu  List  Mode  Functions  Utilities  Help
------------------------------------------------------------------------------
                              ISPF Command Shell
Enter TSO or Workstation commands below:

===> LISTUSER TEST1

USER=TEST1  NAME=UNKNOWN  OWNER=KREUSCH   CREATED=15.280
 DEFAULT-GROUP=PRAKT    PASSDATE=00.000 PASS-INTERVAL=180 PHRASEDATE=N/A
 ATTRIBUTES=NONE
 REVOKE DATE=NONE   RESUME DATE=NONE
 LAST-ACCESS=UNKNOWN
 CLASS AUTHORIZATIONS=NONE
 NO-INSTALLATION-DATA
 NO-MODEL-NAME
 LOGON ALLOWED    (DAYS)          (TIME)
 ---------------------------------------------
 ANYDAY                          ANYTIME
  GROUP=PRAKT     AUTH=USE      CONNECT-OWNER=KREUSCH   CONNECT-DATE=15.280
    CONNECTS=    00  UACC=NONE     LAST-CONNECT=UNKNOWN
    CONNECT ATTRIBUTES=NONE
    REVOKE DATE=NONE   RESUME DATE=NONE
SECURITY-LEVEL=NONE SPECIFIED
CATEGORY-AUTHORIZATION
 NONE SPECIFIED
SECURITY-LABEL=NONE SPECIFIED

OMVS INFORMATION
----------------
UID= 0000000000
HOME= /u/TEST1
PROGRAM= /bin/sh
CPUTIMEMAX= NONE
ASSIZEMAX= NONE
FILEPROCMAX= NONE
PROCUSERMAX= NONE
THREADSMAX= NONE
MMAPAREAMAX= NONE

***
```

Nun wird erneut mithilfe von `LISTUSER` das Profil von „TEST1“ aus der RACF-Datenbank angefragt. Die Ausgabe zeigt, dass das Profil erstellt wurde. Wie spezifiziert ist `PRAKT` die Standardgruppe des Nutzers. Außerdem ist sichtbar, dass dem Nutzer ein OMVS-Segment samt USS-Nutzerverzeichnis angelegt wurde.

```
  Menu  List  Mode  Functions  Utilities  Help
------------------------------------------------------------------------------
                              ISPF Command Shell
Enter TSO or Workstation commands below:

===> LISTDSD DATASET('TEST1.*')

INFORMATION FOR DATASET TEST1.* (G)

LEVEL  OWNER    UNIVERSAL ACCESS   WARNING   ERASE
-----  -------- ----------------   -------   -----
 00    ADMIN           NONE          NO      NO

AUDITING
--------
FAILURES(READ)

NOTIFY
--------
NO USER TO BE NOTIFIED

YOUR ACCESS  CREATION GROUP  DATASET TYPE
-----------  --------------  ------------
   ALTER        ADMIN          NON-VSAM

GLOBALAUDIT
-----------
 NONE

NO INSTALLATION DATA
***
```

Die Ausgabe des Befehls `LISTDSD` zeigt, dass ein generisches Profil (siehe 2.3) für Data-Sets des Nutzers angelegt wurde. Das Profil schützt Data-Sets des Nutzers vor Zugriffen anderer Nutzer, da die UACC (siehe 2.3) auf `NONE` gesetzt wurde.

6.1.2. DELUSER

Ausgangspunkt ist das Ende von Abschnitt 6.1.1. Das heißt, es wurde ein Nutzerprofil mit dem Namen TEST1 und ein dazugehöriges generisches Profil für Data-Sets „TEST1.*“ angelegt. Beide werden nun mithilfe des Skriptes „DELUSER“ gelöscht.

```
  Menu  List  Mode  Functions  Utilities  Help
 ------------------------------------------------------------------------------
                              ISPF Command Shell
 Enter TSO or Workstation commands below:

 ===>  EXEC REXX.EXEC(DELUSER) EXEC
 Mit diesem Skript koennen automatisiert Nutzer geloescht werden
 Bitte einen Dataset-Member fuer das Log angeben:
 Format: HLQ.LLQ(Membername) Ohne Eingabe wird das Standard Data set verwendet.

 Standard-Dataset-Member wird verwendet:  KREUSCH.REXX.EXEC(LOG)
 Bitte Namensprefix (z.B. "prak") eingeben
TEST
 Zahl der Tage die sich die Nutzer nicht mehr eingeloggt haben sollen eingeben
 (-1 wenn die Suchoption AGE nicht verwendet werden soll)
-1
 Es wurden 1 Nutzer gefunden:
 Benutzer  TEST1  gefunden.
 Sollen die Nutzer wirklich geloescht werden(j/n)?
```

Um das Skript „DELUSER“ aus dem „Command“ Subscreen heraus zu starten, gibt man `EXEC REXX.EXEC(DELUSER) EXEC` ein und bestätigt mit Enter. Wie bei „ADDUSER“ wird nach einem Dat-Set-Member für das Log gefragt. Anschließend wird man aufgefordert ein Namensprefix für die Suche nach Nutzerprofilen anzugeben. Alle Nutzer, deren Nutzername mit diesem Namensprefix beginnt, werden in der Datenbank gefunden.
Danach gibt es die Möglichkeit die Suche mit der Option `AGE` in Tagen zu spezifizieren. Es werden dann nur die Nutzer gefunden, die sich für die Anzahl der angegebenen Tage nicht mehr eingeloggt haben. Wenn diese Suchoption nicht verwendet werden soll, muss für `AGE` -1 eingegeben werden. Anschließend durchsucht das Skript die RACF-Datenbank. Wenn keine Nutzer gefunden werden, gibt das Skript eine entsprechende Nachricht aus und wird beendet. Werden Nutzer gefunden, so werden sie aufgelistet und das Skript fragt, ob die gefunden Nutzer gelöscht werden sollen.

```
n
 Abgebrochen, keine Nutzer geloescht
 ***
```

Wenn keine Aktion ausgeführt werden soll, muss `N` eingegeben werden. Das Skript terminiert dann ohne Nutzer zu löschen.

```
j
 Loesche Nutzer TEST1
 Nutzer erfolgreich geloescht
 ***
```

Wenn die Aktionen ausgeführt werden sollen, muss `J` eingegeben werden. Das Skript löscht die Profile, deren Data-Sets und bestätigt dies jeweils. Alle relevanten Informationen über den Vorgang werden in die Logdatei geschrieben und das Skript terminiert.

```
  Menu  List  Mode  Functions  Utilities  Help
------------------------------------------------------------------------------
                              ISPF Command Shell
Enter TSO or Workstation commands below:

===> LISTUSER TEST1

ICH30001I UNABLE TO LOCATE USER    ENTRY TEST1
***
```

Die Ausgabe des Befehls `LISTUSER` zeigt, dass das Nutzerprofil „TEST1“ gelöscht wurde, da es in der RACF-Datenbank nicht gefunden wurde.

7. Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit sollten Lösungen entstehen, um Accounts für Studenten in z/OS besser automatisiert anzulegen und um obsolete Accounts zu finden und zu löschen.
Dazu wurden zunächst die Grundlagen von RACF herausgearbeitet. RACF legt für jede Ressource, die es schützen soll, jeden Benutzer und jede Benutzergruppe ein Profil seiner Datenbank an, welches die Zugriffsinformationen enthält. Ressourcen im Sinne von RACF sind alles, was auf einem Mainframe schützenswert ist. Das sind vor allem Data-Sets, Transaktionen und der Zugriff auf die Subsysteme von z/OS. Ressourcen können dabei in Ressourcenklassen eingeteilt und einem Sicherheitslevel zugewiesen werden.
Darauf folgend wird die für die Skripte verwendete Sprache „REXX“ beleuchtet. Besonders interessant war dabei, wie aus einem Skript heraus auf die Subsysteme von z/OS zugegriffen werden kann. Einerseits stellen viele Subsysteme spezielle Befehle nur für REXX bereit, zum anderen können Befehle, die der Nutzer sonst in der Konsole eingibt, aus einem Skript heraus gesendet werden.
Im folgenden Kapitel wurden wichtige Entwurfsentscheidungen für die Skripte festgehalten. Es wurde beispielsweise entschieden, dass die Skripte per Kommando-Zeilen-Schnittstelle bedient werden sollen. Außerdem wurde festgelegt, dass der Skriptnutzer beim Anlegen von Accounts einen Namenspräfix und einen Zahlenbereich angeben soll. Die Accountnamen setzen sich dann aus dem Präfix und einer Zahl aus dem Bereich zusammen.
Danach wird mit groben Ablaufdiagrammen und erklärten Code-Ausschnitten die Umsetzung der Skripte erläutert. Gesondert erklärt werden Probleme, die bei der Implementierung auftraten und wie sie gelöst wurden.
Es folgt eine dokumentierte Beispielausführung, die gleichermaßen die Funktionalität der Skripte zeigt und eine Bedienungsanleitung liefert.

7.1. Ausblick

Das entstandene Skript zum Anlegen von Nutzerprofilen vergibt nur grundlegende Berechtigungen wie den Zugriff auf TSO und OMVS an die Nutzer. Eine folgende Arbeit könnte das Skript um Optionen erweitern, wie beispielsweise Profil-Segmente für andere Subsysteme wie dem Transaktionsmonitor „CICS“ und das Datenbankmanagementsystem „DB2“ anzulegen. Beim Anlegen des OMVS-Segmentes wird allen Nutzern die UID 1 gegeben, da die automatische Vergabe von UIDs auf dem Testsystem nicht aktiviert werden konnte. Dies liegt vor allem daran, dass die RACF-Datenbank dafür in mehreren Schritten in die passende Form überführt werden müsste. Dies ist ein komplizierter Prozess (siehe [IBM11a, S.222]), der zusammen mit den anderen Verbesserungen Teil einer Folgearbeit sein könnte.

Glossar

Auditor Der Auditor ist dafür verantwortlich zu überprüfen ob Sicherheitsmechanismen funktionieren. Das bedeutet insbesondere zu kontrollieren, ob Benutzer Zugriff auf Ressorucen haben, auf die sie keinen Zugriff haben sollten. Dazu werden Nutzeraktionen in Logs protokolliert. Es obliegt auch dem Auditor Umfang der Protokollierung zu spezifizieren [IBM13b].

Char Char, kurz für Character (engl. Zeichen), repräsentiert genau einen Buchstaben, eine Zahl oder ein Zeichen. Auch Sonder- und Steuerzeichen wie das Leerzeichen sind Chars.

Interpreter Skripte (wie zum Beispiel mit REXX geschriebene) werden nicht kompiliert, sondern vom Interpreter zur Zeit der Ausführung interpretiert.

iterieren Eine Liste schrittweise durchgehen.

Job Im Kontext von z/OS ist ein Job die Ausführung eines Programms. Dies ist vergleichbar mit einem Prozess unter UNIX-artigen Betriebssystemen.

kompilieren Das Übersetzen von menschenlesbarem Programmcode in ausführbaren Maschinencode vor der Ausführung.

Mainframe Ein Großrechner aus IBMs System Z Serie.

Open-Source Programme werden als Open-Source bezeichnet wenn der Quellcode, aus dem sie erzeugt wurden öffentlich zugänglich ist. Es gibt verschiedene Lizenzmodelle, die zum Teil auch vorsehen, dass jeder den Code verändern und dann weiterverbreiten darf.

Record Logische Einheit zusammengehörender Daten, Wörter oder Felder. z/OS Verwendet ein record-orientiertes File System, Windows und UNIX ein Byte Stream File System. Bei letzteren muss das Ende eines Records deswegen durch ein spezielles Zeichen gekennzeichnet werden [EOO05].

Skript Skript bezeichnet hier eine Auflistung von Befehlen einer Skriptsprache. In der Regel sind Skripte nicht kompiliert, sondern werden interpretiert.

String String bezeichnet in vielen Programmiersprachen eine Aneinanderreihung von Chars.

System Eine bestimmte Instanz von z/OS.

Verzeichnis Synonym für Ordner. Kann Dateien enthalten.

z/VM IBM Großrechnerbetriebssystem für den parallelen Betrieb fast beliebig vieler virtueller Maschinen.

Literaturverzeichnis

[BCDB+14] BUECKER, Axel ; CHAKRABARTY, Boudhayan ; DYMOKE-BRADSHAW, Lennie ; GOLDKORN, Cesar ; HUGENBRUCH, Brian ; NALI, Madhukar R. ; RAMALINGAM, Vinodkumar ; THALOUTH, Botrous ; THIELMANN, Jan: *Security on the IBM Mainframe: Volume 1 A Holistic Approach to Reduce Risk and Improve Security*. Bd. 1. 2014 http://www.redbooks.ibm.com/redbooks/pdfs/sg247803.pdf 3

[EOO05] EBBERS, M. ; O'BRIEN, W. ; OGDEN, B.: *Introduction to the New Mainframe: z/OS Basics*. 2005 http://www.redbooks.ibm.com/redbooks/pdfs/sg246366.pdf. – ISBN 0738435341 9, 13, 35, 45

[IBM09] IBM: *Security Server RACF General User's Guide*. 2009 http://publibz.boulder.ibm.com/epubs/pdf/ich2a100.pdf 8

[IBM10] IBM: *Using REXX and z/OS UNIX System Services*. 2010 http://publibz.boulder.ibm.com/epubs/pdf/bpxzb6a0.pdf 30

[IBM11a] IBM: *Security Server RACF System Programmer's Guide*. 2011 http://publibfp.dhe.ibm.com/epubs/pdf/ichza2c0.pdf 44

[IBM11b] IBM: *TSO/E Customization*. 2011 http://publibz.boulder.ibm.com/epubs/pdf/ikj4b4c0.pdf 3

[IBM11c] IBM: TSO/E REXX Reference. (2011). http://publibfp.dhe.ibm.com/epubs/pdf/ikj4a3a0.pdf 16, 18, 19, 28, 31, 34

[IBM13a] IBM: *ISPF Services Guide*. 2013 http://publibz.boulder.ibm.com/epubs/pdf/isp2sg00.pdf 34

[IBM13b] IBM: *Security Server RACF Auditor ' s Guide*. 2013 http://publibz.boulder.ibm.com/epubs/pdf/ich2a800.pdf 45

[IBM13c] IBM: *Security Server RACF Security Administrator's Guide*. 2013 http://publibz.boulder.ibm.com/epubs/pdf/ich2a700.pdf 4, 5, 6

[IBM13d] IBM: *TSO/E User's Guide*. 2013 http://publibz.boulder.ibm.com/epubs/pdf/ikj2o200.pdf 35

[RAB+06] ROGERS, Paul ; ANTOFF, Theodore ; BRUINSMA, Patrick ; HERING, Paul-Robert ; KÜHNER, Lutz ; O'CONNOR, Neil ; SOUSA, Lívio: *UNIX System Services z/OS Version 1 Release 7 Implementation*. 2006 http://www.redbooks.ibm.com/redbooks/pdfs/sg247035.pdf 10, 11, 12, 13

[Tan14] TANENBAUM, Andrew S.: *Modern Operating Systems*. Fourth Edi. 2014. – ISBN 978–0133591620 9

Anhang A.

Anhang

A.1. Skript „ADDUSER“

```
/*REXX*/
/* autoprefix fuer nicht vollstaendige dsnamen entfernen*/
"PROFILE NOPREFIX"
/*Script zum anlegen eines oder mehrerer Nutzer*/
say 'Mit diesem Skript koennen RACF-Benutzer angelegt werden'
/*Dataset fuer das Log spezifizieren*/
FILEERROR: /*Sprungmarke*/
say 'Bitte einen Dataset-Member fuer das Log angeben: '
say 'Format: HLQ.LLQ(Membername) Enter druecken fuer Standart Dataset'
pull filePath

if filePath == "" then
  do
    PARSE SOURCE . . . . PATH .
    filePath = PATH||"(LOG)"
    say 'Standard-Dataset-Member wird verwendet: 'filePath
  end

SIGNAL ON ERROR NAME FILEERROR /*bei Fehler: Sprung zu Marke */
/* Dataset allozieren*/
"ALLOCATE DATASET("FILEPATH") FILE(FILEDS) OLD REUSE"
SIGNAL OFF ERROR

/*Eigabe der Spezifikationen*/
do until namePrefix \== ""
  say 'Bitte Namensmuster eingeben (z.B. "prak") ',
      'Die Nummer des Nutzers wird ',
      'angehaengt. z.B. prak1'
  pull namePrefix
end
say 'Das Skript erstellt Nutzer in einem Zahlenbereich. z.B. prak10',
    'bis prak20.'
do until start > 0 & DATATYPE(start,W) == 1
  say 'Bitte Start des Zahlenbereiches angeben'
  pull start
end
```

```
do until ende >= start & DATATYPE(ende,W) == 1
  say 'Bitte Ende des Zahlenbereiches angeben'
  pull ende
end
user.0 = ende-start+1

say 'Bitte die Standard-Gruppe fuer die Nutzer angeben: ',
  '(Ohne Eingabe Enter druecken, wenn PRAKT als Standardgruppe',
  'verwendet werden soll.)'
pull dfltgrp
if dfltgrp == "" then dfltgrp = "PRAKT"

say 'Bitte Initialpasswort fuer die Nutzer eingeben.',
    'Bei keiner Eingabe ist das Password der Name der Standardgruppe'
pull password

do  until  pos(antwort,'NJ')\==0  &  length(antwort)==1
  say "Es wurde Folgendes spezifiziert: "
  say "Anzulegende Nutzer: " namePrefix||start " bis ",
      namePrefix||ende
  say "Standardgruppe: " dfltgrp
  if password \== "" then say "Password: " password
                     else say "Initialpassword ist der Name der ",
                              "Standardgruppe"
  say "Sollen wirklich " user.0 " Nutzer angelegt werden?(J/N)"
  pull antwort
  antwort = space(antwort,0)
end

zeiger = 0
if antwort == j then
  do
    do index=start to ende by 1
       zeiger = zeiger + 1
       /* Usernamen fuer das Log Speichern */
       user.zeiger = namePrefix||index
       say 'Erzeuge Nutzer' user.zeiger
       /* Den RACF Befehl zusammensetzten */
       command = "adduser" user.zeiger
       command = command || " TSO(ACCTNUM(ACCT) PROC(DBSPROC)",
       "MAXSIZE(6072) SIZE(5000) MSGCLASS(H) UNIT(SYSALLDA))"
       command = command || " OMVS(UID(1) HOME(/u/"user.zeiger"/)",
       "PROGRAM(/bin/sh))"
       command = command || " DFLTGRP("dfltgrp")"
       if password \== "" then
         do
           command = command || " PASSWORD("password")"
         end
       command
       /* Top Generic Dataset anlegen und Zugriffsrecht geben*/
       "ADDSD" user.zeiger||'.*' "UACC(NONE) GENERIC OWNER(ADMIN)"
       "PERMIT" user.zeiger||'.*' "ACCESS(ALTER) ID(ADMIN)"
       /* OMVS Verzeichnis anlegen und Rechte vergeben*/
       unixcmd = "mkdir /u/"||user.zeiger
       CALL BPXWUNIX unixcmd
```

```
        unixcmd = "chown "||user.zeiger||" /u/"||user.zeiger
        CALL BPXWUNIX unixcmd
        unixcmd = "chmod 700 /u/"||user.zeiger
        CALL BPXWUNIX unixcmd
      end
    /*Logging*/
    MSG.1= 'Dies ist eine automatisch erstellte Logdatei.'
    MSG.2 = 'Am',
      DATE(),
      'Um',
      TIME(),
      'wurden die nachfolgenden Accounts angelegt.'
    MSG.3 = 'Das Skript wurde vom Benutzer',
      USERID(),
      'ausgefuehrt.'
    MSG.4 = ' '
    MSG.5 = 'Es wurde der folgende Befehl verwendet: '
    MSG.6 = command
    MSG.7 = SUBSTR(command,73)
    MSG.8 = ' '
    MSG.9 = 'Angelegte Nutzer:'
    "EXECIO" 9 "DISKW FILEDS (STEM MSG."
    "EXECIO" user.0 "DISKW FILEDS (STEM user."
    "EXECIO 0 DISKW FILEDS (FINIS"
  end
else
  say 'Abgebrochen, keine Nutzer angelegt'
"FREE FILE(FILEDS)"
/*Profil prefix zuruecksetzen*/
"PROFILE PREFIX("USERID()")"
exit
```

A.2. Skript „DELUSER“

```
/*REXX*/
say 'Mit diesem Skript koennen automatisiert Nutzer geloescht werden'
"PROFILE NOPREFIX"

FILEERROR:
say 'Bitte einen Dataset-Member fuer das Log angeben: '
say 'Format: HLQ.LLQ(Membername) Ohne Eingabe wird das Standard Data',
    'set verwendet.'
pull filePath
/* Standard Dataset parsen*/
if filePath == "" then
  do
    PARSE SOURCE . . . . PATH .
    filePath = PATH||"(LOG)"
    say 'Standard-Dataset-Member wird verwendet: ' filePath
  end
```

```
SIGNAL ON ERROR NAME FILEERROR
/* Dataset allozieren (reservieren)*/
"ALLOCATE DATASET("filePath") FILE(FILEDS) OLD REUSE"
SIGNAL OFF ERROR

do until namePrefix \== ""
  say 'Bitte Namenspraefix (z.B. "prak") eingeben'
  pull namePrefix
end
do until age > -2 & DATATYPE(age,W) == 1
  say 'Zahl der Tage die sich die Nutzer nicht',
  'mehr eingeloggt haben sollen eingeben',
  ' (-1 wenn die Suchoption AGE nicht verwendet werden soll)'
  pull age
end
/*Rueckgaben von Kommandos an das Host Environment in die Variable*/
/* LINE umleiten*/
CALL OUTTRAP "LINE.", "*"

/*Suchanfrage an RACF*/
if age \== -1 then
  do
    "search mask("namePrefix") CLASS(USER) AGE("age")"
  end
if age == -1 then
  do
    "search mask("namePrefix") CLASS(USER)"
  end

/*Umleitung der Rueckgaben beenden*/
CALL OUTTRAP "OFF"

ERRORMSG = "ICH31005I NO ENTRIES MEET SEARCH CRITERIA"
if LINE.1 = ERRORMSG then
  do
    say "Keine Benutzer gefunden"
  end
else
  do
    SAY "Es wurden "LINE.0" Nutzer gefunden: "
    /*Die Suchergebnisse ausgeben*/
    do index=1 to LINE.0 by 1
      say "Benutzer " LINE.index " gefunden."
    end

    do  until  pos(antwort,'NJ')\==0  &  length(antwort)==1
      say "Sollen die Nutzer wirklich geloescht werden(J/N)?"
      pull antwort
      antwort = space(antwort,0)
    end

    /*Wenn die Nutzer geloescht werden sollen*/
    if antwort == J then
      do
        /*User loeschen*/
```

```
        do index=1 to LINE.0 by 1
          say "Loesche Nutzer "LINE.index
          /* Generisches Profil des Nutzers loeschen */
          "DELDSD "LINE.index||'.*'
          /* Alle Datasets des Users loeschen */
          "ISPEXEC LMDINIT LISTID(listid) LEVEL("LINE.index||'.*'")"
          do forever
            "ISPEXEC LMDLIST LISTID(&listid) OPTION(LIST) DATASET(dsn)"
            if rc = 0 then
              do
                say "loesche:'"dsn"'"
                "ISPEXEC LMERASE DATASET('"dsn"')"
              end
            else
              leave
          end
          /* OMVS Dateien loeschen*/
          unixcmd = "rm -r /u/"||LINE.index
          CALL BPXWUNIX unixcmd
          /* Eigentliches RACF-Nutzerprofil loeschen */
          "DELUSER "LINE.index
        end
        /* In die Datei schreiben */
        MSG.1 = 'Dies ist eine automatisch erstellte Logdatei.'
        MSG.2 = 'Am',
          DATE(),
          'Um',
          TIME(),
          'wurden die nachfolgenden Nutzer geloescht.'
        MSG.3 = 'Das loeschende Skript wurde vom Benutzer',
          USERID(),
          'ausgefuehrt.'
        MSG.4 = ' '
        MSG.5 = 'Geloeschte Nutzer:'
        "EXECIO" 5 "DISKW FILEDS (STEM MSG."
        "EXECIO" LINE.0 "DISKW FILEDS (STEM LINE."
        /*Datei schliessen und dann freigeben*/
        "EXECIO 0 DISKW FILEDS (FINIS"
        say 'Nutzer erfolgreich geloescht'
      end
    else
      say 'Abgebrochen, keine Nutzer geloescht'
end
"FREE FILE(FILEDS)"
/*Profilprefix zuruecksetzen*/
"PROFILE PREFIX("USERID()")"
exit
```

A.3. Beispiel-Log

```
Dies ist eine automatisch erstellte Logdatei.
Am TT Oct 2015 Um HH:MM:SS wurden die nachfolgenden Accounts angelegt.
Das Skript wurde vom Benutzer XXX ausgefuehrt.

Es wurde der folgende Befehl verwendet:
adduser TEST1 TSO(ACCTNUM(ACCT) PROC(DBSPROC) MAXSIZE(6072) SIZE(5000) M
SGCLASS(H) UNIT(SYSALLDA)) DFLTGRP(PRAKT)

Angelegte Nutzer:
TEST1
```

Erklärung

„Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann“.

Ort Datum Unterschrift