

ARexx ist das aktuelle Mode-Wort vieler Amiga-Anwender. Eine ARexx-Schnittstelle hat jedes gute Programm. Doch schon wieder eine Sprache lernen? Geben Sie sich einen kräftigen Ruck und schwingen Sie sich auf zum König über Ihren Rechner.

von Karsten Wysocki
und David Göhler

ARexx ist der Zauberstab für den Power-User – so scheint es. Damit lassen sich Applikationen zu Aktionen bewegen, die sonst nicht möglich sind. Programme kann man verbinden und zum Zusammenspiel bringen, deren Autoren nichts voneinander wissen.

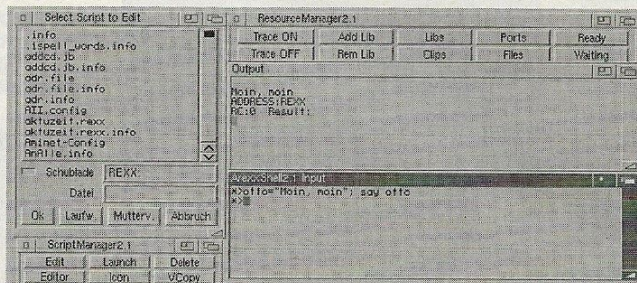
Aber das ist noch nicht alles. ARexx ist auch eine Programmiersprache wie AmigaBASIC. Leicht zu lernen (dafür wurde sie extra so entwickelt), mächtig in den Möglichkeiten und mit allen Raffinessen einer modernen Programmiersprache. Es muß nicht immer C sein – ARexx tut es in vielen Fällen ebenso gut.

Ziel dieses Kurses ist es daher, zu zeigen, wie man ARexx als Sprache und als Hilfsmittel einfach und gezielt einsetzen kann. Mit dem »ARexxManager« (Programm des Monats 9/94) gibt es außerdem eine schöne und einfache zu bedienende Umgebung, in der sich auch einzelne Zeilen eintippen und damit einfach ausprobieren lassen.

Damit der Kurs »lebendig« wird und nicht nur langweilige Textzeilen als Ergebnisse den Bildschirm zieren, bauen wir ARexx mit Public-Domain-Bibliotheken zu einer Profi-Sprache

aus. Dabei entsteht eine schicke grafische Oberfläche für das Archivierungsprogramm »LhA« von Stefan Boberg.

Da der Anfang meist die größte Hürde darstellt, spielt der ARexxManager zu Beginn eine besondere Rolle. Einmal gestartet, können in einer Shell Kommandos einfach eingetippt und die Ergebnisse betrachtet werden. Der Manager unterstützt den Benutzer auch bei der Fehlersuche und ist ein schönes Beispiel dafür, was man in ARexx alles programmieren kann (er selbst ist auch ein ARexx-Programm). Das Programm befindet sich – wie



Der ARexxManager: Er erleichtert die Programmierung von ARexx-Programmen sowie die Fehlersuche und Analyse

auch alle anderen Hilfsprogramme, Libraries und Listings – auf den AMIGA-Magazin-PD-Disketten 9/94 und 7/95.

In die Lehre

Ohne das nötige Handwerkzeug geht auch bei ARexx nicht viel. Deshalb gilt es, sich zuerst mit den grundlegenden Sprach-elementen zu beschäftigen, bevor man überhaupt etwas starten oder steuern kann.

Dieses Kennenlernen ist nicht schwer, da ARexx-Programme nicht kompiliert werden. Man braucht also keinen Compiler, der das Programm erst in eine computergerechte Form bringt. Einzeiler lassen sich einfach eintippen. Anschließend sieht man

gleich, was passiert ist. Mehrzeiler tippt man in eine Datei und übergibt diese dem ARexx-Interpreter, der sie Zeile für Zeile analysiert und ausführt. Dabei kann man ihm sogar sagen, er soll jede Zeile mit ausgeben, die er aus-

mitten im Programmcode ohne vorherige Initialisierung benutzt werden.

Im weiteren Verlauf kommt der ARexxManager zum Einsatz. Das ist zwar nicht unbedingt nötig, aber einfacher. Als Ersatz können Sie Einzeiler auch in einer Shell mit dem Kommando

```
RX " ... "
```

starten, wobei für die drei Punkte jeweils ein oder mehrere durch ein Semikolon getrennte Befehle stehen. Eine ARexx-Datei kann ebenfalls mit RX gestartet werden. Dazu ist nur der Dateiname hinter dem Befehl anzugeben.

Wenn Sie den ARexxManager noch nicht installiert haben, tun Sie das bitte. Eine Installationsanleitung liegt dem Programm

```
/* IfThen Einfache Auswahlbedingung */

schalter = 1 /* schalter bekommt den Wert 1 */

if schalter = 1 /* wenn schalter = 1 */
then say "Schalter ein" /* dann sage "Schalter ein" */
else say "Schalter aus" /* sonst sage "Schalter aus" */

exit /* Programmende */
```

Listing 1: Abfragen erledigt man in ARexx per If-Then-Else. Hinter »If« muß ein Vergleich stehen

führt. Es ist also sehr einfach, mitzuverfolgen, was gerade passiert.

Wer glaubt, dies müsse ja fürchterlich langsam sein, irrt. Auch wenn ARexx nicht die schnellste Sprache ist, benutzt sie doch häufig Bibliotheksfunktionen, die vorher von einem Compiler übersetzt wurden und recht schnell arbeiten.

Variablen

Wie jede Sprache kennt auch ARexx Variablen. Doch müssen diese nicht vor der ersten Benutzung »erklärt« (definiert) werden. Kann ARexx etwas nicht identifizieren, was eigentlich einen Wert zurückliefern müßte – wie eine Variable – wird kurzerhand diese Variable erzeugt und ihr automatisch ein Wert zugewiesen: der Name der Variablen in Großbuchstaben. Variablen können also

bei. Starten Sie es und aktivieren Sie das Eingabefenster der ARexxShell. Versuchen Sie folgendes in das untere Fenster einzugeben:

```
Say otto
```

Daraufhin erhalten Sie die Ausgabe

```
OTTO
```

Mit der Eingabe »Say otto« haben Sie ARexx mit der Anweisung »say« befohlen, den Wert des Ausdrucks »otto« auszugeben. Weil »otto« jedoch keinen Wert zugewiesen bekommen hat, kann die Sprache die Variable nicht finden und legt sie automatisch an. Als Wert hat sie ihre eigene Bezeichnung in Großbuchstaben erhalten. Geben Sie als nächstes ein:

```
otto = "Hallo" ; Say otto
```

Ein Kurs für wen?

Dieser Kurs ist eine Einführung in ARexx und nicht in die Programmierung selbst. Grundlegende Kenntnisse darüber, was Variablen, Schleifen, Funktionen und Operatoren sind, sollte man mitbringen – allerdings ist das im ersten Teil noch nicht ganz so wichtig.

Die Kenntnisse sind aber nicht schwierig zu erwerben, falls Sie bisher noch überhaupt nicht programmiert haben. Die meisten heutzutage benutzten Sprachen verwenden diese Konzepte, egal ob C, Pascal, BASIC oder ARexx. Alle kennen Variablen und Schleifen. Schulkenntnisse aus einem Informatik-Kurs reichen ebenfalls völlig aus.

Vielmehr ist der Kurs für diejenigen gedacht, die ARexx als mächtiges Werkzeug einsetzen möchten, aber sich bisher dazu noch nicht aufraffen konnten. Der Kurs kommt recht schnell zur Sache, damit Ergebnisse schnell sichtbar werden, man den Spaß nicht verliert und die Kursergebnisse sinnvoll eingesetzt und weiterentwickelt werden können.

(Folge 1)

aufgesetzt

Als Reaktion erscheint nun »Hallo«, also der vorher zugewiesene Wert. Das Semikolon in der Eingabezeile dient zum Trennen von Anweisungen. Es bewirkt, daß ARexx die beiden Anweisungen genauso behandelt, als stünden sie in zwei separaten Zeilen.

Übrigens kann man in ARexx-Variablen alles hineinstopfen: Ganze Zahlen, Fließkommazahlen und Texte. Man muß auch nicht vorher festlegen, welche Art Wert eine Variable aufnimmt. Benutzt ein Programm Variablen zum Rechnen, versucht es den Inhalt des Platzhalters als Zahl zu interpretieren. Wird der Inhalt als Zeichenkette (String) benötigt, liefert die Variable den Inhalt als String (auch wenn darin das Ergebnis einer Berechnung steht). Das ist sehr bequem, aber nicht ganz unproblematisch, denn was ist etwa $5 + \text{»Hugo«}$?

Abfragen

Für Abfragen besitzt ARexx mehrere Sprachelemente, die anhand einfacher Beispiele leicht verständlich sind. Die gängige Form mit »if« und »then« steht im Listing 1. Tippen Sie die wenigen Zeilen ab und speichern Sie sie als »TestIfThen«.

Dies geht recht einfach mit dem ScriptManager: Klicken Sie auf »Edit«, um den Editor zu starten. Anschließend ist der Dateiname auszuwählen (hier »Test

IfThen«) und auf »OK« zu klicken. Nach der Eingabe und Speichern des Listings als Datei läßt es sich mit »Launch« (oder RX) starten.

Sie können die mit »/*« und »*/« geklammerten Kommentare

anderen Wert, verzweigt das Programm zu der Anweisung hinter »else«; Probieren Sie es aus!

Zusätzlich zum »If« kennt ARexx noch eine Mehrfachauswahl. Diese beginnt mit der Anweisung »select«. Jede Abfrage beginnt mit »when« (statt mit »if«) und zum Schluß ersetzt ein »otherwise« das »else« für alle »when«-Abfragen. Im Listing 2 ist klar zu sehen, wie es funktioniert: Je nachdem, welchen Wert der Ausdruck »schalter« hat, trifft eine der mit »when« abgeprüften Bedingungen zu und die hinter »then« stehende Anweisung wird ausgeführt. Sind alle Bedienun-

geprüft – das ist aber falsch. Die Schleife wird in jedem Fall einmal ausgeführt und dann die Bedingung am Ende getestet. Im Beispiel führt ARexx die Schleife neunmal durch. Dann hat »schalter« den Wert 10 und die Schleife findet ihr Ende.

ARexx kennt eine Menge Schleifenarten

Wenn im Beispielprogramm der Ausdruck »schalter« zu Beginn den Wert 10 hat, würde die Schleife endlos laufen, da der Wert in der Schleife um eins erhöht wird, also dann 11 ist, und nie mehr den Wert 10 annimmt. Falls Sie das ausprobieren, können Sie die Endlosschleife durch Anklicken des Ausgabefensters und Drücken von <Ctrl c> anhalten (oder Sie starten das Programm »HI«, das automatisch alle ARexx-Programme stoppt).

Die zweite Schleife »DoWhile« besitzt die Anfangsbedingung hinter »do while«. Sie wird vor dem Betreten der Schleife getestet. Im Gegensatz zu »do until« muß die Bedingung erfüllt sein, damit die Schleife durchlaufen wird. Gilt sie schon zu Anfang nicht, springt das Programm gleich zum Ende, ohne die Befehle im Schleifenrumpf auch nur einmal auszuführen.

Wenn also im Beispielprogramm der Ausdruck »schalter« beim Schleifeneintritt den Wert 10 hat, sehen Sie keinen einzigen Wert im Ausgabefenster.

Vorbereitungen

ARexx ist eine sehr anspruchslose Sprache. Für den ersten Anfang braucht man nur zwei Dateien: »RexxMast« und »RX«. RexxMast ist das Programm, dem man ARexx-Texte schickt und der diese dann als Programm abarbeitet. Es sollte im Verzeichnis WBStartup liegen, damit das Amiga-OS es automatisch bei jedem Reset startet.

RX ist der Befehl, der die Programmtexte an RexxMast schickt. Ein Aufruf wie

RX Programm.rxx

schickt also »Programm.rxx« zu RexxMast, der die Zeilen abarbeitet. Wenn es nur eine Zeile ist, die abgearbeitet werden soll, kann diese in Anführungszeichen eingeschlossen direkt versendet werden:

RX "Say Hallo"

Bequemer geht es allerdings mit dem »ARexxManager«, der auf unserer PD-Diskette Nr. 2 zu finden ist. Er unterstützt Sie auch bei der Fehlersuche und anderen wichtigen Aktionen.

übrigens weglassen, bis auf den Kommentar in der ersten Zeile, denn der ARexx-Interpreter erkennt ARexx-Programme nur dann als solche an, wenn in der ersten Programmzeile ein solcher Kommentar steht.

Nach dem Start des Programms erscheint in einem Fenster der Text »Schalter ein«, da die Abfrage »schalter = 1« in der fünften Zeile richtig ist und damit die Anweisung hinter »then« ausgeführt wird. Hat »schalter« einen

gen falsch, kommt die Anweisung hinter »otherwise« dran.

Schleifen

Schleifen gibt es in ARexx mehr als in allen anderen Sprachen – vor allem, weil man die verschiedenen Typen kombinieren kann. Darunter gibt es Schleifen mit Anfangs-, End- und Abbruchbedingung, sowie solche mit Zählern. Wie üblich dienen sie dazu, Befehlsabläufe so lange zu wiederholen, bis eine Bedingung erfüllt ist oder ein Zähler einen bestimmten Wert erreicht hat.

Alle Schleifen beginnen mit der Anweisung »do« und enden mit »end«. Hinter »do« folgt die Angabe über die Art, die außer bei Schleifen mit Abbruchbedingung auch Anfangs- bzw. Endbedingungen und die Anzahl der Durchläufe enthält. Im dritten Listing sind einige davon zu sehen (bitte einzeln als Beispiele abtippen und speichern).

Die erste Schleife hat eine Endbedingung, zu erkennen am »do until«. Sie wird solange durchlaufen, bis die hinter »do until« angegebene Bedingung zutrifft. Da die Bedingung hinter »do until« steht, könnte man meinen, sie würde am Anfang der Schleife

```
/* Select Mehrfachauswahlbedingung */

schalter = 1                /* schalter auf 1 */

select                      /* Wähle aus: */
  when schalter = 1         /* Wenn schalter = 1, */
  then say "Schalter auf 1" /* dann sage "Schalter auf 1" */
  when schalter = 2         /* Wenn schalter = 2, */
  then say "Schalter auf 2" /* dann sage "Schalter auf 2" */
  when schalter = 3         /* Wenn schalter = 3, */
  then say "Schalter auf 3" /* dann sage "Schalter auf 3" */
  when schalter = 4         /* Wenn schalter = 4, */
  then say "Schalter auf 4" /* dann sage "Schalter auf 4" */
otherwise
  say "Schalter ungültig" /* Andernfalls sage . */
end                          /* Ende der Auswahlbedingung */

exit                        /* Programmende */
```

Listing 2: Wenn die Abfragen überhand nehmen, bietet sich die Verwendung eines »select-when«-Konstruktes an

Kursübersicht

ARexx in Perfektion, für Leute, die sich bisher nie aufrufen konnten, diese einfache aber mächtige Sprache zu lernen, die zu jedem Amiga gehört: Das ist das Ziel dieses Kurses.

Folge 1: Einführung in ARexx, Variablen-Handling, Schleifen, Abfragen, Blöcke, Nachrichten; »TestMsg« verschickt Nachrichten.

Folge 2: Funktionsaufrufe, Dateizugriffe und die Konsole, Unterprogramme, Steuern anderer Programme, Anbindung von Golded mit Skripten.

Folge 3: Die »apig.library«: grafische Oberflächen mit Schaltern, Fenstern und Requestern; eine Oberfläche für »LHA«.

Folge 4: Tabellenverarbeitung, ein ListView-Gadgets, Fenster-Refresh, TAC- (TransAktionsCode-) Programmieretechnik, weitere ARexx-Funktions-Bibliotheken.

Das Listing mit dem Namen »DoForever« ist ein Beispiel für eine Schleife mit Abbruchbedingung, die mit einem »do forever« beginnt. Der Abbruch erfolgt innerhalb per »leave«. Ein Schleifenabbruch mit »leave« kann zusätzlich auch bei anderen Schleifenarten stehen.

Zum Schluß bleibt noch die Schleife mit Durchlaufzähler. Sie wird auch mit »do« eingeleitet, gefolgt von der Initialisierung eines Zählers und der Angabe eines Abbruchkriteriums. Hierbei läßt sich allerhand variieren, weshalb in Listing 3 gleich mehrere Schleifen dieser Art zu finden sind.

ARexx ist eine einfache und mächtige Sprache

Wenn nach der Initialisierung des Zählers das Schlüsselwort »for« und ein numerischer Wert oder Ausdruck folgt, durchläuft das Programm die Schleife so oft, wie hinter »for« angegeben. Statt »for« ist auch ein »to« erlaubt. Damit erreicht man, daß die Schleife so lange läuft, bis der to-Wert erreicht ist.

Normalerweise erhöht sich der Zähler pro Durchlauf automatisch um eins. Dies muß aber nicht so bleiben. Mit »by« als Zusatz kann der Programmierer festlegen, um wieviel der Zähler sich zu ändern hat. Wie das geht, zeigt das letzte Beispiel im selben Listing.

Anweisungsblöcke

Anweisungsblöcke verhalten sich wie Schleifen, die nur einmal durchlaufen werden. Sie beginnen mit dem Schlüsselwort »do« ohne zusätzliche Angaben und enden mit »end«. Sie sind nützlich, wenn hinter einem Schlüsselwort wie »then« nur eine Anweisung stehen darf, man dort aber mehrere unterbringen muß. Denn ARexx faßt das »do« und »end« mit allen Anweisungen dazwischen nur als eine Anweisung auf. Damit lassen sich bei Abfragen also beliebig viele Befehle hinter »then« und »else« ausführen (Listing 4).

Stille Post

Doch genug der grauen Theorie, es hat sich schon genug angesammelt, um ein – sehr nützliches – Programm zu erstellen, das ARexx-Nachrichten empfängt und deren Inhalt auf dem Bildschirm ausgibt. Es agiert damit

wie ein Programm mit ARexx-Schnittstelle und kann dieses eventuell sogar simulieren, was die Fehlersuche stark vereinfacht.

Vor der genauen Analyse des kurzen Listings muß jedoch noch

geklärt werden, was Nachrichten sind und wie man sie zwischen verschiedenen Programmen verschickt. Starten Sie die ARexx-Shell und geben Sie ein:

```
address command list
```

Als Ergebnis Ihrer Eingabe erhalten Sie das Inhaltsverzeichnis der RAM-Disk. Die Anweisung »address« erwartet als Argument den Namen eines »Host«, der Nachrichten empfangen kann. Vorerst reicht es zu wissen, daß ein Host ein irgendwie geartetes Programm ist, das solche Nachrichten empfangen kann.

Wenn nur ein Kommando zu verschicken ist, dann kann man es einfach anfügen, wie hier »list«. Der Host »command« ist nichts anderes als ein Programm, das jedes Kommando als AmigaDOS-Befehlszeile auffaßt (also sich wie eine Shell verhält) und einfach ausführt.

Statt »command« kann man den Namen der ARexx-Schnittstelle eines anderen Programms angeben – vorausgesetzt, das Programm hat einen ARexx-Port.

Das Senden von Nachrichten an Programme ist also recht simpel. Schwieriger wird es, sie in einem ARexx-Programm zu empfangen. Das letzte Listing (Nr. 5) in dieser Folge leistet genau das. Tippen Sie es bitte wie üblich ein und speichern Sie es als »TestMsg«.

Nach erfolgreichem Start gibt das Programm zuerst »Anfang« im erscheinenden Ausgabefenster aus. Anschließend lädt es die »rexsupport.library«, die im Verzeichnis »LIBS:« liegen muß. Diese Library enthält zusätzliche Funktionen wie z.B. »openport()«, »waitpkt()«, »getpkt()«, »getarg()« und »reply()«.

Nachrichten sind mit ARexx schnell verschickt

Danach öffnet es einen ARexx-Port mit der Funktion »openport()«. Da sie einen Wert zurückliefert, kann dieser in einer Variable gespeichert werden: hier ist es »p«. Dies ist vor allem sinnvoll, weil so festgestellt werden kann, ob der Port geöffnet werden konnte. Sonst ließe sich »openport()« auch mit »call openport(portname)« aufrufen. Bei Funktionsaufrufen mit call wird der zurückgelieferte Wert fallengelassen. Ging alles glatt, gibt das Programm den Namen des Ports aus.

Danach initialisiert es die Abbruchvariable »exitme« der äußeren Schleife mit 0. In ihr wird zuerst auf eingehende Nachrichten gewartet. Trifft eine ein, geht es in der inneren Schleife mit dem Ab-

```
/* DoUntil Schleife mit Endebedingung ----- */
schalter = 1 /* schalter auf 1 */

do until schalter = 10 /* Solange bis schalter = 10 */
  say schalter /* Wert von schalter ausgeben */
  schalter = schalter + 1 /* Schalter um 1 erhöhen */
end /* Zurück zum Schleifenanfang */

say "Fertig" /* Zur Kontrolle Verlassen der */
/* Schleife anzeigen */
exit /* Programm beenden */

/* DoWhile Schleife mit Anfangsbedingung ----- */
schalter = 1 /* schalter auf 1 */

do while schalter < 10 /* Solange schalter kleiner 10 */
  say schalter /* Wert von schalter ausgeben */
  schalter = schalter + 1 /* Schalter um 1 erhöhen */
end /* Zurück zum Schleifenanfang */

say "Fertig" /* Zur Kontrolle Verlassen der */
/* Schleife anzeigen */
exit /* Programm beenden */

/* DoForever Schleife mit Abbruchbedingung ----- */
schalter = 1 /* schalter auf 1 */

do forever /* Schleife endlos ausführen */
  say schalter /* Wert von schalter ausgeben */
  if schalter = 10 then leave /* bei 10 Schleife verlassen */
  schalter = schalter + 1 /* Schalter um 1 erhöhen */
end /* Zurück zum Schleifenanfang */

say "Fertig" /* Zur Kontrolle Verlassen der */
/* Schleife anzeigen */
exit /* Programm beenden */

/* DoFor Schleife mit Durchlaufzähler ----- */
do zaehler = 11 for 20 /* Anfangswert = 11, 20 Durchläufe */
  say zaehler /* Wert von zaehler ausgeben */
end /* Zurück zum Schleifenanfang */

say "Fertig" /* Zur Kontrolle Verlassen der */
/* Schleife anzeigen */
exit /* Programm beenden */

/* ForTo Schleife mit Durchlaufzähler ----- */
do zaehler = 11 to 20 /* anfangs 11, Ende wenn 20 erreicht */
  say zaehler /* Wert von zaehler ausgeben */
end /* Zurück zum Schleifenanfang */

say "Fertig" /* Zur Kontrolle Verlassen der */
/* Schleife anzeigen */
exit /* Programm beenden */

/* DoForBy Schleife mit Durchlaufzähler ----- */
anf = 11 /* Anfang bei 11 */
anz = 20 /* 20 Durchläufe */
inc = 2 /* Schrittweite 2 */

do zaehler = anf for anz by inc /* Schleifeninitialisierung */
  say zaehler /* Wert von zaehler ausgeben */
end /* Zurück zum Schleifenanfang */

say "Fertig" /* Zur Kontrolle Verlassen der */
/* Schleife anzeigen */
exit /* Programm beenden */
```

Listing 3: Bei Schleifen hat ARexx eine Vielfalt zu bieten, die sonst keine prozedurale Sprache aufzuweisen vermag

holen der Nachricht (via »getpkt()«) und der Bearbeitung weiter. Letztere erschöft sich im Beispiel darin, die erste Nachrichtenkomponente zu bestimmen und auszugeben.

»TestMsg« – ein ARexx-Host in ARexx geschrieben

Die innere Schleife hat die Aufgabe, Nachrichten solange vom Port abzuholen, bis keine mehr vorliegen. Das erkennt das Skript daran, daß »getpkt()« den hexadezimalen Wert »0000 0000« zurückgibt. Ist dies der Fall, wird die innere Schleife verlassen und die äußere Schleife fortgesetzt.

In der inneren Schleife wird außerdem noch geprüft, ob die eingegangene Nachricht »DIE«

gramm beendet, sonst auf die nächste Nachricht gewartet.

Wenn dies Programm läuft, sollten Sie im »ResourceManager« einmal auf das Gadget »Ports« klicken. In der daraufhin erscheinenden ARexx-Port-Liste können Sie den von Ihrem Programm geöffneten ARexx-Port »TestMsg« wiederfinden. Schließen Sie die Port-Liste, wechseln Sie in die ARexxShell und geben Sie als Befehl ein:

```
address TestMsg Hallo
```

Erstaunlicherweise reagiert unser Testprogramm darauf nicht. Warum? Weil die von Ihnen angegebene Adresse »TestMsg« intern nicht als Variable gefunden wurde und ARexx eine angelegt hat, die als ersten Inhalt den Namen in Großbuchstaben erhält (Sie erinnern sich?). Dies ist auch der Ausgabe der ARexxShell zu

Operatoren

Wie jede vernünftige Sprache besitzt auch ARexx eine Reihe Operatoren, die zum Rechnen gedacht sind. Wie in der Mathematik gilt dabei Punktrechnung vor Strichrechnung. Vor jede Zahl kann man ein Minus- oder Pluszeichen schreiben. Mit Klammern lassen sich Ausdrücke gruppieren.

Operator	Funktion
Arithmetisch	
+	Addition von zwei Zahlen
-	Subtraktion der zweiten von der ersten
*	Multiplikation von zwei Zahlen
/	Division (Teilen) der ersten Zahl durch die zweite
%	Ganzzahlergebnis einer Division
//	Rest einer Division von zwei Zahlen
**	Potenzierung der ersten Zahl mit der zweiten
Logisch	
~	logisches Nicht
&	logisches Und zweier Wahrheitswerte
	logisches Oder zweier Wahrheitswerte
! oder &&	logisches Exklusiv-Oder
Bei Vergleichen	
gelten für Zahlen und Zeichenketten	
==	auf exakte Gleichheit testen
!=	auf exakte Ungleichheit testen
=	auf Gleichheit testen
>	auf Ungleichheit testen
> oder <	auf größer als vergleichen
<	auf größer gleich testen
< oder >	auf kleiner als vergleichen
<= oder >=	auf kleiner gleich testen

heißt. Sie sorgt dafür, daß sich das Programm beendet. Hierzu muß es die Abbruchvariable »exitme« auf 1 setzen.

Anschließend ist noch mit »reply()« die empfangene Nachricht an den Versender zurückzuliefern. Dies ist nötig, weil die sendende Task auf eine Empfangsbestätigung wartet und erst weiterarbeiten kann, wenn sie eingetroffen ist.

Wird die innere Schleife verlassen, prüft die äußere, ob das Abbruchkriterium gültig ist (per »do until«). Wenn ja, wird die äußere Schleife verlassen und das Pro-

```
/* TestMsg © Karten Wysocki 1995 ----- */

say "Anfang" /* Programmstart melden */

call addlib("rexsupport.library",0,-30,0) /* rexx...lib laden */
/* Diese Library enthält weiter unten benötigte Funktionen */
portname = "TestMsg" /* Portnamen festlegen */
p = openport(portname) /* ARexx-Port öffnen */

if p = "0000 0000"x /* konnte nicht geöffnet werden ? */
then do /* dann folgendes ausführen: */
    say "Port konnte nicht geöffnet werden - Abbruch!"
    exit /* und Programm verlassen */
end /* Ende des Blocks */
else
    say "Address =" portname /* Portnamen anzeigen */

    exitme = 0 /* Abbruch auf ungültig setzen */
    do until exitme = 1 /* Anfang der äußeren Schleife */
        x = waitpkt(portname) /* Warten auf Empfang einer Message */
        do forever /* Anfang der inneren Schleife */
            msg = getpkt(portname) /* Message aus Port lesen */
            if msg = "0000 0000"x /* Wenn Message-Block-Ende */
            then leave /* dann innere Schleife verlassen */
            arg0 = getarg(msg,0) /* Erstes Msg-Argument lesen */
            say "Msg-Arg 0:" arg0 /* Message-Argument ausgeben */
            say " " /* Leerzeile ausgeben */
            if arg0 = "DIE" /* Wenn Message-Argument = "DIE" */
            then exitme = 1 /* dann äußere Schleife abbrechen */
            x = reply(msg,0) /* Zurückliefern der Message */
        end /* Zum inneren Schleifenanfang */
    end /* Zum äußeren Schleifenanfang */

say "ende" /* Programmende anzeigen */

exit /* Programmende */
```

Listing 5: Ein kompletter ARexx-Host als ARexx-Programm. Damit wird die Fehlersuche zum Kinderspiel.

entnehmen. Dort wird als Adresse »TESTMSG« angezeigt. Da unser Programm jedoch die Adresse »TestMsg« hat (mit Kleinbuchstaben), müssen Sie eingeben:

```
address "TestMsg" Hallo
```

Darauf reagiert das Programm mit der Ausgabe:

```
Msg-Arg 0: HALLO
```

Es hat also die Message »HALLO« in Großbuchstaben erhalten. Daß es nicht »Hallo« heißt, liegt wiederum daran, daß die von Ihnen eingegebene Nachricht als nicht initialisierter Ausdruck behandelt wurde. Wenn Sie

nun »Hallo« auch in Häkchen setzen, bekommen Sie das gewünschte Ergebnis.

Experimentieren Sie ruhig ein wenig mit dem pfiffigen Programm. Statt des absoluten Wertes »"Hallo"« kann man auf folgende Weise auch andere Ausdrücke einsetzen:

```
msg = "Otto";address "TestMsg" msg
```

Das Programm kann nach Abschluß Ihrer Experimente mit folgender Message beendet werden:

```
address "TestMsg" DIE
```

Danach können Sie das Ausgabefenster schließen.

Das soll fürs erste genug sein. Experimentieren und probieren Sie ruhig noch ein wenig Der nächste Teil widmet sich Dateizugriffen, dem Öffnen von Fenstern zur Texteingabe und Unterprogrammtechniken. Als Beispiele gibt es nützliche ARexx-Programme, die den Texteditor »Gold« steuern, der als Shareware frei verfügbar ist. dg

Literatur:

- [1] Amiga-OS 3.1 ARexx, Commodore Electronics Limited. Erhältlich bei Village Tronic Marketing GmbH, Sarstedt
- [2] Michael Metz: ARexx, Eine Einführung und mehr ..., CompuStore Handelsgesellschaft, ISBN 3-93073300-5

```
/* DoEnd Anweisungsblöcke ----- */

schalter = 1 /* schalter bekommt den Wert 1 */

if schalter = 1 /* wenn schalter = 1 */
then do /* dann führe Anweisungsblock aus */
    schalter = "ein" /* schalter bekommt neuen Wert */
    say "Schalter" schalter /* Sage "Scha..." & schalter-Wert */
end /* Ende des Blocks, sonst */
else do /* führe Anweisungsblock aus */
    schalter = "aus" /* schalter bekommt neuen Wert */
    say "Schalter" schalter /* Sage "Scha..." & schalter-Wert */
end /* Ende des Anweisungsblocks */

exit /* Programmende */
```

Listing 4: Anweisungsblöcke bündeln mehrere Befehle und Zuweisung so, als sei es eine Anweisung