

ARexx ist nicht nur eine Sprache für kleine Hilfsprogramme und zum Steuern von Programmen! Mit einer speziellen Programmieretechnik lassen sich auch größere Vorhaben einfach realisieren. Außerdem erfahren Sie im letzten Teil, wie man einige ARExx-Fallen gekonnt umgeht.

von Karsten Wysocki und
David Göhler

Zuerst noch ein paar nützliche Hinweise zum Umgang mit der im letzten Teil vorgestellten »apig.library«: Die darin enthaltenen Funktionen sind zwar in der beigefügten Dokumentation ausreichend beschrieben, manche Probleme sind aber nicht so leicht zu lösen. Solche kleinen Haken und Ösen sprechen wir deshalb noch einmal kurz an.

Tabellen mit ARExx

Für Listviews (umrahmte Textlisten mit einem Ziehbalken und Pfeilen) ist es ganz nützlich, eine programminterne Tabelle zu erzeugen, die den Inhalt der einzelnen Zeilen der Liste enthält. ARExx benutzt eine recht pfiffige Form der Variablenindizierung, die in anderen Sprachen so nur selten zu finden ist. Auch eine mehrfache Indizierung – für mehrdimensionale Felder oder Tabellen – macht keine Probleme.

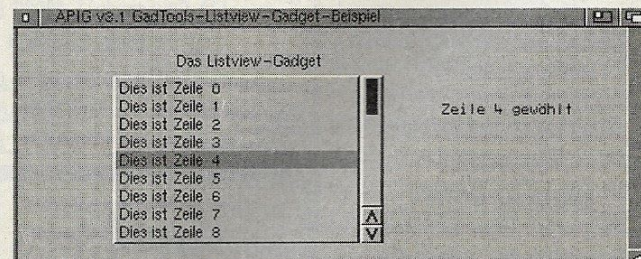
Stellen Sie sich vor, Sie hätten einen Notizblock mit Kästchen. Auf jedem Blatt des Blocks sind jeweils 20 Reihen zu je 15 Käst-

chen, also 20 Zeilen und 15 Spalten. Der Block hat 50 Blätter. Wenn Sie diesen Block mit einer Tabelle erfassen wollen und dann pro Kästchen bestimmte Eintragungen vornehmen möchten, muß die Tabelle dreidimensional sein. Um ein bestimmtes Kästchen zu adressieren, füllen Sie in

weise »Tabelle.1.3.5« heißt. »Tabelle« selbst wird nicht weiter ersetzt, nur die Variablen hinter den Punkten. Der Ausdruck fördert dann den Wert zutage, der in der ersten Reihe, dritte Spalte, auf dem fünften Blatt steht. Um sich die Arbeitsweise der Tabellenverarbeitung zu verdeutlichen, pro-

MOVE« angegeben. Um die Liste jedoch mit den Pfeil-Gadgets kontinuierlich bewegen zu können, (also mit gedrückter linker Maustaste), muß zusätzlich noch »INTUITICKS« angegeben werden (s. Listing 2).

Außerdem sollte man sich den Inhalt der Liste zeilenweise in einer internen Tabelle merken; wenn der Benutzer ein Listenelement auswählt, wird im ersten Message-Argument nur die Nummer der ausgewählten Zeile zurückgeliefert. Um im Programm dann herauszufinden, welcher Text in dieser Zeile steht, kann man diese Nummer als Index benutzen und so das entsprechende Element der internen Tabelle ermitteln. Wichtig: die Nummern der ListView-Zeilen beginnen bei 0 (nicht 1).



Interpretationssache: Ob C oder ARExx, letztlich sieht man dem Fenster nicht an, welche Sprache dahintersteht

die Variablen »zeile«, »spalte« und »seite« die entsprechenden Werte und setzen diese Variablen dann mit einer zusätzlichen Basisvariable (z. B. »Tabelle«) zu einer neuen Variable zusammen. Die einzelnen Elemente trennt ein simpler Punkt:

```
Tabelle.zeile.spalte.seite
```

Dadurch kann der ARExx-Interpreter die Variablen erkennen und auswerten. Zum Ausführungszeitpunkt ersetzt er »zeile«, »spalte« und »seite« durch ihre Werte, behält jedoch die Punkte bei, so daß die Variable beispiels-

bieren Sie doch einmal das Listing 1 aus.

Witzigerweise läßt aber ARExx nicht nur die Indizierung per Zahl, sondern auch via String zu. Ein einfaches Beispiel:

```
Adresse.1.Name = "Müller"
Adresse.1.Ort = "Hamburg"
Adresse.1.Telefon = "040/1234"
Was = Name
Say Adresse.1.Was
```

Und was gibt das? Die Variable »Was« ersetzt der Interpreter durch »Name« und »Adresse.1.Name« enthält den Text »Müller«. Diese Übung ist in anderen Sprachen in dieser Form überhaupt nicht möglich und erlaubt eine enorm einfache Organisation unterschiedlicher Daten. So könnte man den Benutzer fragen, welchen Teil einer Adresse er benötigt, und die Antwort in die Variable »Was« verfrachten. Anschließend gibt man den gewünschten Teil mit »Say« – wie gezeigt – einfach aus.

Listviews mit »apig«

Daß sich mit der apig.library auch Listviews erstellen lassen, ist in dem apig-Beispiel-Programm »gadtools3.rexx« schon gut zu sehen, der weitere Umgang mit dieser Gadget-Art ist leider unzureichend dokumentiert.

Als »WINIDCMP« wird zum Scrollen der Liste auch »MOUSE-

Fenster-Refresh

Auch mit der apig.library erzeugte Fenster haben normalerweise ein Zoom-Gadget in der Titelleiste. Wenn der Benutzer dieses aktiviert, wird das Fenster normalerweise auf Minimalgröße gebracht; bis dahin ist die Welt noch in Ordnung. Wenn aber das Fenster wieder vergrößert wird, sind zwar einfache Schalter wieder vorhanden, Listviews, mit »pi-text()« erzeugte Texte und einfache Rahmen fehlen jedoch. Diese muß das Programm selbst wieder »hinmalen«. Um überhaupt benachrichtigt zu werden, wenn der Benutzer ein Fenster in der Größe verändert hat, ist beim Öffnen zusätzlich als »windcmp« »NEWSIZE« zu übergeben. Dann kann man im Programm genauso wie z.B. auf »GADGETUP« reagieren:

```
if msgclass = GADGETUP then ...
if msgclass = NEWSIZE then do
/* do the job ... */
call gt_refreshwindow(winptr,0)
end
```

Da das Fenster bei Erhalt der Nachricht bereits auf die neue Größe gebracht wurde, kann man gleich seine Arbeit erledigen (hier im Kommentar »do the job« angedeutet). Um ein ListView wiederherzustellen, reicht es, die Funktion »gt_refreshwindow()« aufzurufen. Texte und Rahmen

```
/* SimpleTab */

/* Befüllen der Tabelle mit Werten */
do n = 1 to 5
  zeile = n
  do i = 1 to 2
    spalte = i
    TABELLE.zeile.spalte = "Zeile" n ",Spalte" i
  end
end

/* Tabelleninhalt zur Kontrolle ausgeben */
do zeile = 1 to 5
  do spalte = 1 to 2
    echo TABELLE.zeile.spalte
  end
end

exit
```

Listing 1: So einfach ist es, mit ARExx Tabellen aufzubauen und zu verarbeiten

müssen genau wie nach dem Öffnen des Fensters mit den Funktionen »pitext()« und »drawbevel-box()« neu gezeichnet werden.

Einzellauf

ARexx-Programme, die selbst eine ARexx-Schnittstelle zum Empfang von Nachrichten öffnen, können bei Verwendung eines festen Namens nur einmal gestartet werden. Jeder weitere Versuch vor Beendigung des ersten Skripts endet dann mit einer unschönen Fehlermeldung, da es einen Port mit dem gleichen Namen schon gibt. Das läßt sich mit folgender Befehls-Sequenz beim Öffnen verhindern:

```
portname = "ARexxManager"
portlist = Showlist('p')
if find(portlist, portname) > 0
then exit
p = openport(portname)
```

Diese Befehlssequenz bewirkt, daß sich das ARexx-Programm zumindest fehlerfrei beendet, wenn der zu öffnende Port schon existiert.

Um ein ARexx-Programm mehrfach startbar zu machen, bewirkt sich folgende Sequenz:

```
x=1
do forever
  portname = "ARexxManager." || x
  p = openport(portname)
  if p ~= 0 then leave
  else x = x + 1
end
```

Diese Befehls-Sequenz versucht als erstes, den Port »ARexxManager.1« zu öffnen. Wenn das nicht klappt, weil der Port schon existiert, wird die Zahl

Rexx-Oberflächenprogramm ZedREXX

»ZedREXX« ist eine Spracherweiterung von David N. Junod, der bei Commodore das Datatypes-Konzept entwickelt und implementiert hat. Außerdem gehört er zu den Autoren des Style-Guides.

ZedREXX arbeitet aber nicht wie die »apig.library«, sondern stellt – in etwa vergleichbar mit MUI – in ARexx neue Sprachkonstrukte zur Verfügung, um sehr einfach auch komplexe Anwendungen mit den typischen Bildelementen wie Schaltern, Schieberegeln und Listen aufzubauen.

Die Besonderheiten von ZedREXX sind: automatisches Speichern von Fensterpositionen und -ausmaßen, beliebig viele Fenster pro Anwendung, Tastaturkürzel für alle Schaltertypen, automatische Anpassung aller Elemente an die Systemzeichensätze, Unterstützung für Mauspositionsabhängige Soforthilfe, Drag-and-Drop-Hilfen, Ikonifizierung des Fensters, einfache Behandlung von Ereignissen, »Buchhaltung« von angeforderten Systemressourcen und einiges mehr. Damit bietet es mehr als alle anderen Rexx-Erweiterungspakete und erleichtert die Programmierung erheblich. Ein Blick lohnt sich.

Das Schnupperpaket ist im Aminet und auf unserer AMIGA-Magazin-PD-Serie zu finden. Es kann zwar alles, ihm fehlt jedoch die Dokumentation, die man erst nach einer Registrierung erhält. Sie kostet 35 Dollar plus Versand (7 Dollar für Europa).

am Namensende solange erhöht, bis die Aktion erfolgreich ist.

Dynamisches ARexx-Auto-Assign

Für ARexx-Applikationen, die ein eigenes Assign auf das aktuelle Verzeichnis benötigen, muß die Assign-Anweisung nicht unbedingt in der Startup-Sequenz oder in einem CLI-Startup-Skript stehen. Das Assign aufs aktuelle Verzeichnis kann die Applikation selbst mit folgender Befehlszeile anlegen:

```
address command
  'assign X: "`cd`"'
```

»X:« ist dabei der vom Programmierer zu wählende Name, für »cd« wird zur Ausführungszeit automatisch der aktuelle Verzeichnisname eingesetzt. Das bewirkt der Einschluß des Kommandos in Rückwärtshäkchen (über die Taste direkt unter <Esc> zu

erreichen). Die Ummantelung des Kommandos mit doppelten Anführungszeichen oben (<Shift 2>) ist notwendig, da der aktuelle Verzeichnisname durchaus Leerzeichen enthalten kann (man denke nur an die »RAM Disk«). Die gesamte an die Adresse »Command« zu übergebende Zeichenkette ist mit Hochkomma (<Alt â>) einzuschließen.

ARexx ist die universelle Programmiersprache

Das ganze geht allerdings auch anders. Die Funktion:

```
dir = pragma(D)
```

liefert als Ergebnis ebenfalls das aktuelle Verzeichnis und ist dem relativ aufwendigen Konstrukt mit »cd« vorzuziehen. Wie man sieht, sind die Möglichkeiten unter ARexx sehr vielfältig und es gibt meistens mehr als nur einen Weg, um auftretende Probleme zu lösen. Doch nun zur bereits erwähnten professionellen Programmierertechnik:

TAC-Programmierung

Aufgrund der Multitasking-Fähigkeit des Amiga und den besonderen Eigenschaften der Programmiersprache ARexx ist es möglich, Software-Designs auf Basis der Transaktions-Code-Technik aufzubauen, wie dies unter UTM auf BS2000-Großrechnern üblich ist (nicht erschrecken!). Die Besonderheit von Transaktions-Codes (TACs) ist, daß jeder TAC ein eigenständiges Programm ist, das nach dem Prinzip »Eingabe – Verarbeitung – Ausgabe« arbeitet und damit mit den Amiga-Tasks vergleichbar ist.

Dieses Prinzip in Verbindung mit TACs wird fast ausschließlich auf Großrechenanlagen eingesetzt, weil auf Kleincomputern die Voraussetzungen nicht gegeben sind (man benötigt dazu echtes, Hardware-unterstütztes Multitasking und kein Windows-Pseudo-Zeug). Auf dem Amiga ist es jedoch ohne weiteres möglich, dieses Prinzip umzusetzen.

Man entwickelt dazu ein Steuerprogramm, das z.B. mit den apig.library-Funktionen einen Bildschirm öffnet. Dieses Programm muß einen eigenen ARexx-Port öffnen und danach ein weiteres ARexx-Programm (wie ein Hauptmenü) mit

```
address command "run
  sys:rexcc/rx > NIL: prgname"
```

starten. Wenn das Steuerprogramm den Bildschirm geöffnet hat, und das Hauptmenü ein Auswahlfenster auf diesem Screen erzeugen soll, muß vor dem Start des Hauptmenü-TACs der Screen-Pointer hinterlegt werden. Das geht z.B. mit ARexx-Clips:

```
call setclip("Screen",
  ,screenpointer)
```

Alle weiteren vom Steuerprogramm gestarteten TACs können sich dann den Screen-Pointer mit

```
screenpointer = getclip("Screen")
```

abholen. Nach dem Starten des ersten TACs wartet das Steuerprogramm in einer Schleife mit »waitpkt()« auf eingehende Nachrichten. Es ist im Prinzip genauso aufgebaut wie das Beispielprogramm »TestMsg« aus dem ersten Kursteil.

Kursübersicht

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

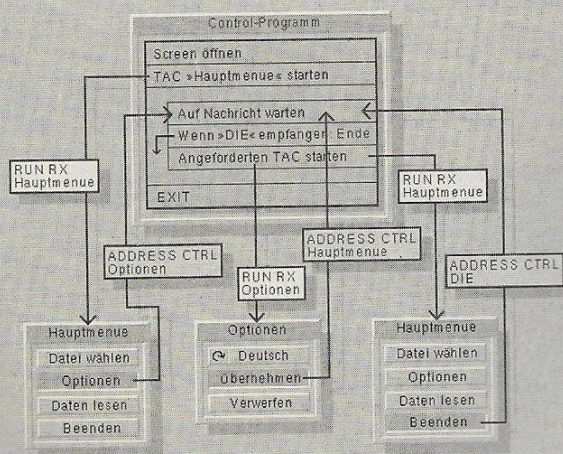
Folge 1: Einführung in ARexx, Variablen-Handling, Schleifen, Abfragen, Blöcke, Nachrichten; »TestMsg« verschickt Nachrichten.

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-Gadget, Fenster-Refresh, TAC- (Transaktions-Code-) Programmierertechnik, weitere ARexx-Funktions-Bibliotheken.

Das Prinzip der TAC-Programmierung



TAC-Programmierung: Für größere ARexx-Anwendungen ein nützliches und sehr modulares Programmierprinzip

```

/* GadTools3.rexx -- ein ListView-Beispiel */
/* dieses Skript würde etwas modifiziert */
x = addlib("apig.library",0,-30,0)
call addlib("rexxsupport.library",0,-30,0)
call SET_APIG_GLOBALS()

/* Zeiger auf Workbench-Screen holen */
scr = LOCKPUBSCREEN("Workbench")
scrinfo = GETVISUALINFO(scr)
scrfont = GETVALUE(scr,40,4,'p')
glistpointer = ALLOCVEC(4,MEMF_CLEAR)
conxgad = CREATECONTEXT(glistpointer)

previousgadget = conxgad; myid = 10

/* Zuerst ein NewGadget erzeugen und initialisieren */
newgad = MakeNewGadget(scrinfo,scrfont,65,32,185,120,
    , "Das ListView-Gadget", PLACETEXT_ABOVE, myid, null())

/* Dann eine Exec-Liste reservieren und initialisieren */
myexeclist = build_me_a_list()
/* we now create the gadget, a LISTVIEW gadget */
previousgadget = CREATEGADGET(LISTVIEW_KIND,
    , previousgadget, newgad, GTLV_LABELS, myexeclist,
    , LAYOUTA_SPACING, 1, TAG_DONE, 0)

wintitle = "APIG v3.1 GadTools-ListView-Gadget-Beispiel"
winidcmp = CLOSEWINDOW+GADGETUP+GADGETDOWN+MOUSEMOVE+INTUITICKS,
    +NEWSIZE
winflags = WINDOWCLOSE+WINDOWDRAG+WINDOWSIZING+WINDOWDEPTH,
    +GIMMEZEROZERO+ACTIVATE

portname = "apiggadt3_port"; p = openport(portname)

/* Fenster öffnen, conxgad zeigt auf unsere Gadgets */
w1 = OPENWINDOW(portname,0,30,500,200,0,1,winidcmp,winflags,
    ,wintitle,scr,0,null(),null(),conxgad)

call GT_REFRESHWINDOW(w1,null()) /* ist nötig! */
rpw1 = GETWINDOWRASTPORT(w1) /* zum Zeichnen von Text */

exitme = 0; clicked = -1
do while exitme = 0
    x = waitpkt(portname)
do forever

```

```

msg = getpkt(portname)
if msg = '0000 0000'x then leave
msgclass = getarg(msg,0)
msgcode = getarg(msg,1)
x = reply(msg,0)
select
when msgclass = CLOSEWINDOW then exitme = 1
when msgclass = NEWSIZE then do
    if clicked >= 0 then
        call pitext(rpwl,290,50,("Zeile" clicked "gewählt"),
            ,1,0,JAM2,null())
        call GT_REFRESHWINDOW(w1,null()) /* ist nötig! */
    end
when msgclass = GADGETUP then do
    clicked = msgcode
    call cleareol(rpwl,290,50)
    call pitext(rpwl,290,50,("Zeile" clicked "gewählt"),
        ,1,0,JAM2,null())
    end
otherwise nop
end
end
end

call CLOSEWINDOW(w1); call UNLOCKPUBSCREEN(null(),scr)
call FREEVISUALINFO(scrinfo); call FREEGADGETS(conxgad)
call FREETHIS(newgad); call FREEVEC(glistpointer)
call FREE_EXEC_LIST(myexeclist,,1)
exit

build_me_a_list:
/* our list structure always use ALLOCMEM */
/* so it can be freed properly. */
mylist = ALLOCMEM(14,MEMF_CLEAR)

call NEWLIST(mylist)
do x = 1 to 30
    txt = "Dies ist Zeile " x-1
    ptr_to_node_added = ADD_LIST_NODE(mylist,txt)
end

return mylist

```

Listing 2: Dieses Listing erzeugt ein Fenster mit ListView. Es ist die verbesserte Version von »GadTools3.rexx«

Die TACs müssen selbst keinen AREXX-Port öffnen, es sei denn, sie benutzen z.B. die »apig.library« für Fenster und Schalter. In den TACs wird dann nach Aufbau der Oberfläche eine Benutzereingabe abgewartet und danach entschieden, ob eine TAC-interne Verarbeitung erfolgt oder ob ein neuer TAC (z.B. »Optionen«) zu starten ist. Die Entscheidung über den Start eines neuen TACs wird dem Steuerprogramm per

address controlprg botschaft

mitgeteilt. Danach beendet sich der aktuelle TAC. Das stellt sicher, daß immer nur ein TAC aktiv ist und kein Durcheinander entsteht. Daten an nachfolgende TACs kann man über Dateien im RAM:, der PIPE: und über AREXX-Clips weiterreichen.

In der Hauptschleife des Steuerprogramms wird nichts weiter getan, als auf eingehende Nachrichten zu warten und die gewünschten Programmteile (TACs) zu starten. Zusätzlich sollte es sich auf eine bestimmte Nachricht hin (»DIE«) selbst beenden. Dies

könnte das Hauptmenü-TAC ans Steuerprogramm versenden, wenn der Benutzer im Hauptmenü auf »Quit« klickt.

Diese Programmieretechnik hat den Vorteil, daß eine größere Applikation in mehrere kleine, leicht überschaubare Teile zerlegt wird. Das verringert die Fehlermöglichkeiten und garantiert einen stabilen Ablauf. Auch die Testmöglichkeiten werden dadurch enorm verbessert, denn man kann die einzelnen TACs separat testen.

Ein weiterer angenehmer Nebeneffekt ist, daß sich die ein-

zelnen TACs z.B. bei Versions-Updates leicht austauschen lassen. Das im dritten Kursteil vorgestellte Programm »LHACControl« wurde nach diesem Prinzip entwickelt, des geringen Programmfangs wegen jedoch auf ein Steuerprogramm verzichtet – es ist im Listing »LHACControl« integriert.

Und zu guter Letzt erhält man über TACs mit der Zeit und bei vorausschauender Programmierung automatisch Module, die sich immer wieder in neuen Programmen einsetzen lassen, ohne daß man etwas an ihnen ändern müßte.

Wer nach diesem Prinzip größere Anwendungen entwickeln möchte, dem sei »AREXX-ManagerProfessional« empfohlen. Es enthält für die TAC-Programmierung spezielle Erweiterungen wie den »HostManager«. Damit lassen sich einzelne TACs und das Steuerprogramm bequem mit der Maus manipulieren und testen. Auch der Aufbau eigener Funktionsbibliotheken und das automatische Einbinden eigener Funktionsblöcke in AREXX-Listings wie dort von den Funktionen »LinkF« und »ShowF« unterstützt. Fertige Applikationen können kompiliert werden und verhalten sich anschließend wie übersetzte Programme anderer Sprachen.

AREXXManagerProfessional beinhaltet ein deutsches Handbuch, zusätzliche Funktionsbibliotheken sowie zahlreiche Beispiel-Applikationen und ist bei »Wolf Software & Design« erhältlich. 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

Nützliche AREXX-Libraries

Im Public-Domain-Bereich sind zahlreiche Funktionsbibliotheken für AREXX erhältlich. Viele dieser Bibliotheken enthalten jedoch die gleichen Funktionen, arbeiten nicht so systemkonform wie die »apig.library« oder sind nur auf Amiga-OS 1.3 abgestimmt. Deshalb wurde in diesem Workshop bisher auch nur die »apig.library« vorgestellt. Sie enthält allerdings nicht alles, was das AREXX-Programmierherz begehrt, so daß man nicht umhin kommt, zur Lösung mancher Probleme weitere Bibliotheken heranzuziehen. Deshalb hier die wichtigsten zur Ergänzung:

Name	Funktion	Quelle
owndevunit.library	Ermöglicht Zugriff auf Devices (z.B. serial.device)	Fish 643
rexxmathlib.library	Mathematische Funktionen	Fish 227
rexxRMF.Library	Ermöglicht Aufbau von Datenbank-Strukturen auf Basis von binären Bäumen	Fish 629