Port-Programmierung

TURTELN MIT AREXX

Viele professionelle Programme erhalten den letzten Schliff erst durch ihre ARexx-Schnittstelle. Wir zeigen, wie eigene Programme mit einer solchen Schnittstelle ausgerüstet werden.

von Bob Malzan

Daß ein ARexx-Port in eigenen Programmen kein Problem ist, zeigt unser Beispielprogramm. Dieses richtet eine minimale ARexx-Schnittstelle ein und wertet die – über ARexx an diese Schnittstelle übermittelten – Befehle aus. Nebenbei springt ein Programm heraus, das die von der Programmiersprache LOGO bekannte »Schildkröte« (Turtle) implementiert und damit zu kinderleichten Grafikexperimenten anregt – gesteuert durch ARexx-Befehle.

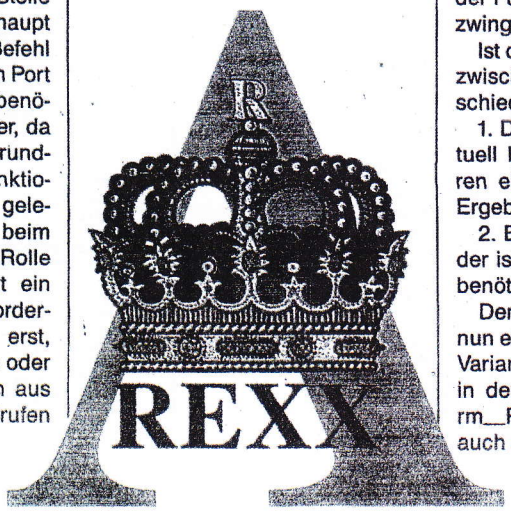
Dreh- und Angelpunkt jedes Datenaustauschs zwischen ARexx-fähigen Anwendungen ist eine Struktur namens RexxMsg. Damit verschickt und empfängt ARexx alle Nachrichten. Das erste Element dieser Struktur ist eine weitere Struktur vom Typ Message. U.a. verwendet auch die Exec-Library diese zum Verschicken Ihrer Nachrichten. Deshalb ist es möglich, ARexx-Messages mit den gewohnten Funktionen GetMsg(), PutMsg() oder ReplyMsg() der Exec-Library zu empfangen, zu verschicken und zu beantworten.

Absender und Empfänger dieser Nachrichten müssen, sozusagen als Briefkasten, zunächst einen Port anmelden. Bei ARexx-Nachrichten ist hierfür eigentlich ein spezieller RexxMsg-Port angesagt, es läßt sich aber auch mit den üblichen Exec-Funktionen arbeiten. Genau das erledigt die Funktion SetupMyPort() im Listing MinRexx.c, die als einzigen Parameter den Namen des zu kreierenden Ports benötigt. Vorher jedoch verhindert diese Funktion, daß der Port-Name nicht schon existiert. Denn zwei Ports mit gleichem Namen – die Guru-Meditation läßt grüßen. Existiert ein Port gleichen Namens jedoch noch nicht, steht dem Öffnen des eigenen nichts im Wege.

Man könnte an dieser Stelle noch prüfen, ob ARexx überhaupt gestartet wurde (mit dem Befehl »rexxmast«), indem nach dem Port »REXX« gesucht wird. Man benötigt ARexx jedoch nicht immer, da die ARexx-Kommunikation grundsätzlich auch ohne ARexx funktioniert. Ja, Sie haben richtig gelesen. Da Exec-Funktionen beim Nachrichtenaustausch eine Rolle spielen, ist nicht unbedingt ein Rexx-Hintergrundprozeß erforderlich. Den brauchen Sie erst, wenn Sie ARexx-Programme oder ARexx-Funktionsbibliotheken aus Ihrer Anwendung heraus aufrufen möchten.

Am zuvor geöffneten Port kann nun mit WaitPort() auf eintreffende Nachrichten gewartet werden. Diese ist mit GetMsg() abzuholen und anschließend zu interpretieren. Für den Empfänger ist in der Nachricht eigentlich nur der STRPTR-Vektor rm__Args[] interessant, und hier auch nur der erste Eintrag rm__Args[0]. Dieser zeigt auf eine Zeichenkette, die das Kommando des Absenders enthält. Alle übrigen Elemente von rm__Args werden nur bei Funktionsbibliotheken aktuell.

Erwartet der Empfänger Befehle mit mehreren Argumenten, muß die Zeichenkette zunächst in ihre Bestandteile, getrennt durch einfache Leerzeichen, zerlegt werden. Das erste Wort in rm__Args[0] sollte daher immer ein Schlüsselwort sein, aus dem sich ableiten läßt, was es mit den übrigen Parametern auf sich hat. In unserem Programm wird dies in der Funktion InterpretCommand() im Listing turtlehost.c durch einen einfachen Stringvergleich festgestellt. Wer viele Befehle realisieren möchte, sollte sich stattdessen mit dem sog. Hashing-Verfahren auseinandersetzen. Dabei wird das empfangene Schlüsselwort zunächst eindeutig codiert und anschließend direkt die Funktion aufgerufen.



Ist das Schlüsselwort einmal gefunden, muß nur noch die zugehörige Aktion durchgeführt werden. Benötigt der Befehl weitere Parameter, ist zu verifizieren, ob ausreichend Parameter übergeben wurden. Zudem sind diese – benötigt man anstelle von Zeichenketten Zahlen – mit geeigneten Funktionen in ihre numerische Form umzuwandeln. Im Beispiel verwenden wir ausschließlich Gleitkommazahlen, was uns zur Benutzung der Funktion atof() (ASCII to Float) zwingt.

Ist der Befehl abgearbeitet, wird zwischen zwei Alternativen unterschieden:

1. Der Befehl ist korrekt, eventuell benötigte Parameter existieren ebenfalls. Eventuell muß ein Ergebnis zurückgegeben werden.
2. Es trat ein Fehler auf. Entweder ist der Befehl unbekannt oder benötigte Parameter fehlen.

Der Absender des Befehls hat nun ein Recht zu erfahren, welche Variante eingetroffen ist. Dies wird in den Feldern rm__Result1 und rm__Result2 – man bezeichnet sie auch als als Primary- und Secondary-Result – eingetragen. rm__Result1 ist ein Returncode. Trat ein Fehler auf, ist hier ein Wert ungleich 0 einzutragen (empfohlen wird 10). Im Secondary-Result steht dann ein Fehlerschlüssel, der bei ARexx-Programmen jedoch leider nicht angezeigt werden kann. Trat hingegen kein Fehler auf, ist zu prüfen, ob der Absender ein Ergebnis wünscht. Bei ARexx-Programmen wird das mit OPTIONS RESULTS erreicht. RexxMast, der das ARexx-Programm ausführt, setzt daraufhin in allen RexxMsg-Strukturen, die er verschickt, das Bit RXFF__RESULT im Element rm__Action.

Dieses Bit überprüft die Funktion ReplyRexxCommand(). Ist es gesetzt, tragen wir das Ergebnis ins Element rm__Result2 ein. Zu diesem Zweck wird eine Struktur namens RexxArgString erzeugt, die im wesentlichen die Zeichenkette enthält und an den Absender zurückgegeben wird. Wie die Struktur aufgebaut ist, spielt hier keine Rolle. Nur so viel: Der Zeiger auf einen RexxArgString kann unbesorgt wie ein STRPTR verarbeitet werden. Nur ist die mit CreateArgString() erzeugte Struktur mit DeleteArgString() freizugeben (in unserem Fall nicht von uns, sondern vom Absender).

Die beiden letztgenannten Funktionen befinden sich in einer von ARexx ausgiebig genutzten Library, der »rexxsyslib.library«. Die Prototypen und Pragmas für diese Funktionen finden Sie im Listing MinRexx.h.

Sind die entsprechenden Ergebnisse (Primary- und Secondary-Results) eingetragen, beantworten wir die Nachricht mit ReplyMsg().

Doch nun zu unserer Schildkröte. Sie soll als Task im Hintergrund

TURTLE-BEFEHLE

Befehl	Wirkung
CLEAR <Farbe>	Löscht den Fensterinhalt mit der angegebenen Farbe.
HOME	Turtle in Ausgangszustand (betrifft auch DIMENSIONS).
PEN UP	Stift anheben. Bei den nachfolgenden Bewegungen wird nichts gezeichnet.
PEN DOWN	Stift senken. Bei den nachfolgenden Bewegungen wird in der aktuellen Farbe gezeichnet.
PEN COLOR <Farbe>	Legt die aktuelle Stiftfarbe fest (0 bis 4).
AHEAD <Anzahl>	Die Schildkröte bewegt sich <Anzahl> Schritte in der aktuellen Richtung vorwärts.
DIMENSIONS <X> <Y>	Legt die virtuelle Fenstergröße fest; voreingestellt ist <1> <1>.
TURN <Winkel>	Dreht die Schildkröte um den angegebenen Winkel (im 360-Grad-Kreis).
DIE	Programmende

auf ARexx-Nachrichten bzw. Befehle warten und diese ausführen. Gestartet wird das Programm mit

```
run Turtle Links Oben Breite Höhe
```

Links, Oben, Breite und Höhe bestimmen hier die Dimensionen des Workbench-Fensters. Danach lassen sich Befehle an den Port mit dem Namen »REXX_TURTLE« senden. Vom CLI/Shell geschieht das z.B. mit

```
rx "address rexx_turtle turn 90"
```

Hierzu allerdings muß ARexx in jedem Fall aktiv sein. In der Tabelle finden Sie alle von der Schildkröte akzeptierten Befehle.

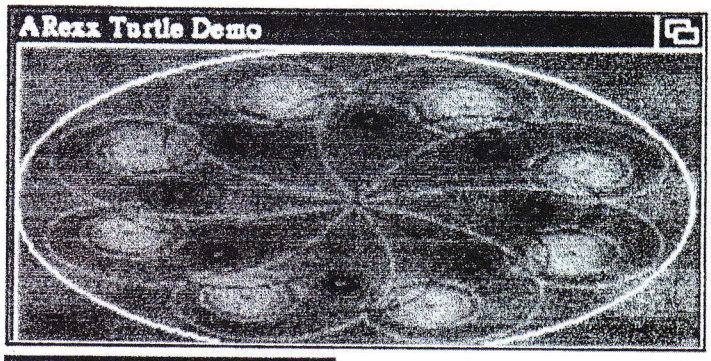
Die Programme wurden mit dem Manx Aztec-C-Compiler erstellt. Übersetzen Sie zunächst die Listings »TurtleHost.c« und »MinRexx.c« mit den Compiler-Optionen »-f8 -c2 -so«. Anschließend müssen beide Objektdateien mit:

```
ln -o Turtle TurtleHost.o
MinRexx.o -lmf -lc
```

zum fertigen Programm »Turtle« gebunden werden. *rz*

Literatur:

[1] Bob Malzan: ARexx-Programmierung auf dem Amiga, Markt & Technik Verlag AG, ISBN 3-87791-035-1, 59 Mark



Hübsch anzuschauen Solche Grafiken lassen sich mit unserem Beispielprogramm Turtle realisieren

Programm: MinRexx.h

Bemerkung: Nur Include-Datei

```
#ifndef MINREXX_H       Programmautor: Bob Malzan
#define MINREXX_H       ------------------------
#include <functions.h>
struct RexxMsg {
   struct Message rm_Node;
   APTR      rm_TaskBlock,rm_LibBase;
   LONG      rm_Action,rm_Result1,rm_Result2;
   STRPTR    rm_Args[16];
};
#define RXCOMM     0x01000000
#define RXFUNC     0x02000000
#define RXFF_NOIO     (1<<16)
#define RXFF_RESULT   (1<<17)
#define RXFF_STRING   (1<<18)
#define RXFF_TOKEN    (1<<19)
#define RXFF_NONRET   (1<<20)
extern void *RexxBase;
STRPTR
CreateArgstring(char *String,long Length);
void   DeleteArgstring(STRPTR String);
struct RexxMsg *CreateRexxMsg(
       struct Port *ReplyPort,
       char *Extension,char *Host);
struct RexxMsg
*DeleteRexxMsg(struct RexxMsg *Msg);
struct MsgPort *SetupMyPort(char *Name);
void FreeRexxCommand(struct RexxMsg *RxMsg);
void ReplyRexxCommand(struct RexxMsg *RxMsg,
 long primary,long secondary, char *result);
#pragma amicall(RexxBase,0x66,
       CreateArgstring(a0,d0))
#pragma amicall(RexxBase,0x6c,
       DeleteArgstring(a0))
#pragma amicall(RexxBase,0x78,
       CreateRexxMsg(a0,a1,d0))
#pragma amicall(RexxBase,0x78,
       DeleteRexxMsg(a0))
#endif /* MINREXX_H */
```

Programm: TurtleHost.c

Aufruf: cc -f8 -c2 -so Turtlehost.c

```
#include <exec/types.h>
#include <exec/ports.h>
#include "minrexx.h"
#include <intuition/intuition.h>
#include <pragmas.h>
#include <math.h>
#define MY_NAME "REXX_TURTLE"
#define TWO_PI (3.141592653589793*2)
#define ERR_NUM_ARGS {
       Primary=10; Secondary=17; }
#define ERR_BAD_TOKEN {
       Primary=10; Secondary=8; }
#define ERR_INVALID_ARG {
       Primary=10; Secondary=18; }
#define WFLAGS
WINDOWDRAG|WINDOWDEPTH|GIMMEZEROZERO
struct NewWindow NewTurtleW = {
  0,0,0,0,0,1,NULL, /* IDCMP-Flags */
  WFLAGS,     /* Flags */
  NULL, NULL,
  (UBYTE *)"ARexx Turtle Demo", /* Title */
  NULL, NULL,
  20, 20, -1,-1,
  WBENCHSCREEN /* Screen Type */
};
struct Window         *TWindow=NULL;
struct MsgPort        *MyPort=NULL;
struct IntuitionBase  *IntuitionBase=NULL;
struct GfxBase        *GfxBase=NULL;
void                  *RexxBase=NULL;
struct RastPort       *RPort;
double Angle=0.0, PosX=0, PosY=0;
double ScaleX=1.0, ScaleY=1.0;
BOOL  PenUp=TRUE;
LONG PenColor=0;
USHORT Width,Height;

BOOL SetupTurtle(int NumArgs,char **Args) {
  NewTurtleW.LeftEdge=atoi(Args[1]);
  NewTurtleW.TopEdge =atoi(Args[2]);
  Width=atoi(Args[3]);
  Height=atoi(Args[4]);
  NewTurtleW.Width   =Width+8;
  NewTurtleW.Height  =Height+14;
  if (Width<=20 || Height<=20)
    return(FALSE);
  TWindow=OpenWindow(&NewTurtleW);
  if (!TWindow)
    return(FALSE);
  RPort=TWindow->RPort;
  Move(RPort,0L,0L);
  return(TRUE);
}
void CleanExit(char *ErrMsg) {
  if (TWindow) CloseWindow(TWindow);
  if (IntuitionBase)
    CloseLibrary(IntuitionBase);
  if (GfxBase)
    CloseLibrary(GfxBase);
  if (MyPort) DeletePort(MyPort);
  if (RexxBase) CloseLibrary(RexxBase);
  if (ErrMsg) {
    puts(ErrMsg);
    exit(10);
  } else exit(0);
}
BOOL InterpretCommand(struct RexxMsg *Msg) {
  char *Result="1"; /* 1 heißt 'alles OK' */
  long Primary=0,Secondary=0;
  int i,CmdCount,Pos;
  char Command[4][20];
  BOOL NotDead=TRUE;
  Pos=0;
  for (i=0,CmdCount=0;
       i<strlen(Msg->rm_Args[0]); i++) {
    char c=Msg->rm_Args[0][i];
    if (c==' '){
      Command[CmdCount++][Pos]='\0';
      Pos=0;
    }
    else Command[CmdCount][Pos++]=
              (isupper(c) ? c : toupper(c));
  }
  Command[CmdCount++][Pos]='\0';
  for(i=CmdCount;i<4;i++) *Command[i]='\0';
  if (strncmp(Command[0],"CLEAR",5) == 0L) {
    if (*Command[1]) {
      SetAPen(RPort,atoi(Command[1]));
      RectFill(RPort,0,0,Width,Height);
      SetAPen(RPort,PenColor);
    } else ERR_NUM_ARGS;
  }
  else if(strncmp(Command[0],"HOME",4)==0L){
    PosX=PosY=Angle=0.0;
    ScaleX=ScaleY=1.0;
  }
  else if(strncmp(Command[0],"PEN",3)==0L) {
    if (*Command[1]) {
      if(strncmp(Command[1],"UP",2) == 0L)
        PenUp=TRUE;
      else if(strncmp(Command[1],"DOWN",4)
                                    == 0L)
        PenUp=FALSE;
      else if(strncmp(Command[1],"COLOR",5)
                                    == 0L) {
        if (*Command[2]) {
          PenColor=atoi(Command[2]);
          SetAPen(RPort,PenColor);
        } else ERR_NUM_ARGS;
      } else ERR_BAD_TOKEN;
    } else ERR_NUM_ARGS;
  }
  else if(strncmp(Command[0],"AHEAD",6)==0L){
    if (*Command[1]) {
      double Distance=atof(Command[1]);
      if (Distance>0.0) {
        PosX+=cos(Angle)*Distance*ScaleX;
        PosY+=sin(Angle)*Distance*ScaleY;
        if (PenUp) Move(RPort,PosX,PosY);
        else Draw(RPort,PosX,PosY);
      } else ERR_INVALID_ARG;
    } else ERR_NUM_ARGS;
  }
  else if (strncmp(Command[0],
        "DIMENSIONS",10) ==0L) {
    if (*Command[1] && *Command[2]) {
      double Dx=atof(Command[1]);
      double Dy=atof(Command[2]);
      if (Dx>0.0 && Dy>0.0) {
        ScaleX=Width/Dx;
        ScaleY=Height/Dy;
      } else ERR_INVALID_ARG;
    } else ERR_NUM_ARGS;
  }
  else if(strncmp(Command[0],"TURN",4)== 0L){
    if (*Command[1])
     Angle+=(atof(Command[1])/360.0*TWO_PI);
    else ERR_NUM_ARGS;
  }
  else if (strncmp(Command[0],"DIE",3)== 0L)
    NotDead=FALSE;
  else ERR_BAD_TOKEN;
  ReplyRexxCommand(Msg,Primary,
                   Secondary,Result);
  return(NotDead);
}
void main(int argc, char **argv) {
  BOOL          NotDone=TRUE;
  struct RexxMsg *RxMsg;
  if (argc==0) CleanExit(NULL); /*Nur CLI*/
  if (argc!=5)  CleanExit("Aufruf: Turtle\
                Left Top Width Height!");
  RexxBase=
      OpenLibrary("rexxsyslib.library",0L);
  if (!RexxBase)
    CleanExit("Wo ist die Rexxsyslib?");
  IntuitionBase=
    OpenLibrary("intuition.library",0L);
  GfxBase=
    OpenLibrary("graphics.library",0L);
  if (!SetupTurtle(argc,argv))
    CleanExit("Window ist nicht zu öffnen!");
  MyPort=SetupMyPort(MY_NAME);
  if (!MyPort)
    CleanExit("Kann Port nicht öffnen!");
  while(NotDone) {
    WaitPort(MyPort);
    while( NotDone && (RxMsg =
      (struct RexxMsg *)GetMsg(MyPort))) {
      /* Rexx-Befehl erhalten, jetzt
       * interpretieren */
      NotDone=InterpretCommand(RxMsg);
    }
  }
  CleanExit(NULL);
}
```

Programm: MinRexx.c

Aufruf: cc -f8 -c2 -so MinRexx.c

```
#include <exec/types.h>
#include <exec/ports.h>
#include "minrexx.h"
static struct MsgPort *RexxPort = 0L;
struct MsgPort *SetupMyPort(char *Name) {
  struct MsgPort *Port;
  Forbid();
    if (FindPort(Name)) { /* irgendwo? */
      Permit();
      return(NULL); /* abbrechen */
    }
    Port = CreatePort(Name,NULL);
  Permit();
  return(Port);
}
void ReplyRexxCommand(struct RexxMsg *RxMsg,
long primary,long secondary,char *result) {
  if(!primary && (RxMsg->rm_Action &
    RXFF_RESULT) ) {
    secondary =
    (result ? (LONG) CreateArgstring(result,
     strlen(result)) : NULL);
  }
  RxMsg->rm_Result1 = primary;
  RxMsg->rm_Result2 = secondary;
  ReplyMsg((struct Message *)RxMsg);
}
```