SLAC-R-95-464
CONF-9505198--

PROCEEDINGS OF THE REXX SYMPOSIUM FOR DEVELOPERS AND USERS

May 1-3, 1995
Stanford, California

Convened by
STANFORD LINEAR ACCELERATOR CENTER
STANFORD UNIVERSITY, STANFORD, CALIFORNIA 94309

Program Committee

Cathie Dager of SLAC, Convener
Forrest Garnett of IBM
Pam Taylor of The Workstation Group
James Weissman

Prepared for the Department of Energy
under Contract number DE-AC03-76SF00515

Proceedings of the 6th International REXX Symposium

Table of Contents

6th International REXX Symposium

Program

Pages 1-6

8:45 **Welcome and Announcements**

Cathie Dager, Stanford Linear Accelerator Center

9:00 **Introduction to REXX Tutorial**

Chip Davis, Aresti Systems

10:00 **Break**

10:15 **Intermediate Programming in REXX Tutorial**

Chip Davis, Aresti Systems

11:15 **Advanced REXX Programming Tutorial**

Chip Davis, Aresti Systems

12:15 **Lunch**

1:30 **Keynote Address: REXX 1995 – The Growth of a Language**

M. F. Cowlishaw, IBM Fellow

Much of the character of REXX today was determined during the first year of its development. In this talk, Mike will take highlights from that first year, and show how the design decisions and user feedback of 1979 have let to steady growth since then and the world-wide use that we see today.

2:30 **Break**

2:45 **The Future of REXX**

Tim Browne, IBM

This presentation discusses IBM's plans for Object REXX including making REXX more pervasive in the industry and aligning with key industry standards. Includes an opportunity for Q&A with the IBM REXX Product Manager.

3:45 **REXX in PC/DOS 7.0**

Dave Gomberg, Experimenta

Another operating system acquired a built-in REXX when IBM shipped DOS 7.0. Although DOS is no longer a cutting edge OS, it now has the latest and greatest shell language available at no extra cost. And legacy users can take advantage of REXX as a shell and prototyping language. This new offering will enhance REXX's claim to be the universal language.

4:15 **Issues and Problems Writing REXX Compilers**

Markus Pelt-Layman, Pelt Industries

This session will cover issues and problems writing REXX compilers.

9:00 **Writing World Wide Web CGI Scripts in REXX**

Les Cottrell & Bebo White, Stanford Linear Accelerator Center

The Common Gateway Interface (CGI) is an interface for running external programs, or gateways, under an information server such as the World Wide Web (WWW). Gateway programs, or CGI Scripts, are executable programs designed to enhance the functionality of a server by providing non-native services. Les and Bebo will describe the operation of CGI and demonstrate how CGI scripts may be written in REXX. In addition, they will point out some of the "gotchas" that SLAC has encountered when using REXX with WWW. This talk will primarily focus on the use of REXX in a Unix environment with the CERN and NCSA WWW servers. Some mention will be made of the VM WWW server written in REXX by Rick Troth.

9:45 **Object REXX Demo**

Rick McGuire, IBM

A demonstration of the latest features in Object REXX, including support for the Workplace Shell, persistent objects, shared objects, and more. See how Object REXX can be used to enhance your OS/2 desktop.

10:45 **Break**

11:00 **Object REXX: Open Doc Support**

Tom Brawn, IBM

11:45 **Report from the X3J18 Committee**

Brian Marks, IBM

The first REXX symposium produced enthusiasm for the idea of a REXX language standard, and offers to participate in the development. The effort started in 1991 and fifteen committee meetings later there is now a proposal for what the standard should say. It is being reviewed by the public. This presentation will cover: choices, corrections, and extensions made by the committee; what is new, what is not, and why; what happens to the proposal next; how you can get a copy; how to understand its more formal parts; what you can do about flaws you detect or perceive in the proposal.

12:15 **Lunch**

1:15 **CenterPiece - An Object-Oriented REXX Development and Runtime Environment**

Sandy Syx, Mantissa Corporation

CenterPiece is a modern, graphical, object-oriented, programmable, distributed, multi-platform, multi-user, interpretive, interactive environment. CenterPiece is suitable for both rapid development and delivery of complex, multi-user, pseudo-realtime applications. CenterPiece was built to be the foundation for developing all types of monitoring and control applications. Mantissa's specific area of interest is in datacenter management solutions. This talk will explore CenterPiece specifically focusing on the object-oriented REXX aspects of CenterPiece.

2:15 **Break**

2:30 **Beyond Client Server**

John Tibbetts, Kinexis

3:30 **Getting Ready for Object REXX**

Rick McGuire, IBM

4:30 SOM: Present and Future

Simon Nash, IBM

IBM's System Object Model (SOM) was introduced in 1992 and is now in its third release, running on eight platforms. A major component of IBM's object strategy, SOM provides a language-neutral object model that allows class libraries to be developed and used in a number of object-oriented and procedural languages (both compiled and interpreted). SOM defines interfaces that allow class libraries to be distributed in binary form and used from other languages than the implementation language, thus enhancing their reusability. Other SOM features include CORBA-compliant object distribution (allowing remote location of objects, transparent to client code) and release-to-release compatibility (allowing new versions of class libraries to be used by unmodified and unrecompiled client code). This talk gives an overview of SOM today, and looks at possible future directions, particularly relating to support for Object REXX and similar dynamic (non-compiled) languages.

8:30 **Rexinda, A REXX Implementation of the Linda Parallel Programming Model**

Stephen Rondeau, AugmenTek

The Linda® parallel programming model was conceived by David Gelernter at Yale University in the 1980's to simplify programming parallel applications. Linda extends a language with four basic functions and two variations on them. These functions put data into and get data from a global, content-addressable data area ("tuple space"). User functions can access that global data and execute in parallel, via multitasking or multicomputing. Since the tuple space and REXX's compound variables are associative, a REXX user may be "comfortable with" Linda. REXX makes it easier to prototype and experiment with hot topics such as data mining and software agents, which can exploit Linda. This presentation will describe Linda and Rexinda in more detail, briefly show how to parallelize a program, elaborate on a simple example and comment on future directions.

9:30 **REXX for CICS/ESA**

Bob Vogel, IBM

10:15 **Break**

10:30 **REXX Changes in OS/2 Warp Version 3**

Dick Goran, C F S Nevada, Inc.

11:15 **S/Rexx for Unix**

David Salthouse, Open Direct

S/Rexx is an implementation of REXX 4.00 for Unix systems with some extensions developed by Robert Benaroya. The presentation discusses the design objectives and gives examples of the benefits. S/Rexx is completely integrated with a Unix Xedit-like editor SEDIT. The principle characteristics are: an interpreter free from size or shape limitations; support for dynamic loading of external procedures which can share global variables with the main procedure; enhanced debugging facilities including a more detailed trace output as well as a Motif based debugger; simplified interface to the Unix platform via a number of additional built-in functions; new ADDRESS environments; extended syntax on a number of instructions.

12:15 **Lunch**

1:15 **A REXX-based Stock Exchange Realtime Client-Server Environment for Research, Educational and Public Relations Purposes: Implementation and Usage Issues**

Martin Misseyer, Free University of Amsterdam

This paper presents the design, development, and implementation of C/S systems from both developer and user views and from both technical and non-technical points of view. Questions addressed include: the difference between quasi-C/S and full C/S; how to develop quasi or full C/S environments in REXX using APIs; REXX portability in C/S environments (designing applications for portability); which REXX programming techniques should be used developing a full C/S environment; which performance and programming techniques should be used in a full C/S environment having large scale database operations and I/O (designing applications for performance); how to create C/S GUIs (specific applications as well as monitors) in REXX.

2:00 **Break**

2:15 **REXX/370 Compiler and Library: What, Why, and How ****

Rick McGuire, IBM

What can the IBM REXX/370 Compiler and Library do for you? What's new with Release 3? This presentation covers performance, other advantages, compatibility, enhancements, supported systems, and tips.

2:45 **REXX and How I Hit the Ground Running in Unix**

Lois White, Stanford Linear Accelerator Center

Possible subtitles for this talk might be something like "uni-REXX, Rx for Making the Transition from VM/CMS to Unix a Little Less Painful" or "How I Learned to Stop Worrying and Love to Type in Significant Mixed Case". Using REXX in Unix really did make it possible for me to move to and, dare I say, "thrive" in the Unix environment. This presentation will show how I use REXX in Unix and how being able to use it there made it possible for me to hit the round running instead of barely crawling.

3:15 **Closing Remarks**

Cathie Dager, Stanford Linear Accelerator Center

**** Session replaced by a general session. Presentation included in the Proceedings.**

REXX 1995 – The Growth of a Language

M. F. Cowlishaw
IBM Fellow

Pages 8-32

Rexx 1995

The Growth of a Language

Mike Cowlishaw

IBM UK Laboratories
Hursley, England

Outline

- The first year
  - Background and context
  - Initial specification, refinement, and evolution
  - Retrospective
- 1980-1995

Reference:

The Early History of Rexx, Mike Cowlishaw
IEEE Annals of the History of Computing,
Vol 16, No. 4, 1994

Whence Rexx?

Rexx grew from two concepts:

1. A *single* macro language for *many* applications (first expounded by Stephenson in 1973)
2. A language designed for the benefit of the *user* (programmer), not the *language implementer*

Traditional macro languages

Macro languages assumed that most of the content of a program would be literal data:

```
&IF &NODE&J ¬= &LOCAL &USER = &STRING OF
&USER&J AT &NODE&J
```

By 1979, programs existed where more than 50% of the tokens began with "&".

The solution:

```
if node.j¬=local then user=user.j 'AT'
node.j
```

March 20-29, 1979

Discussion with EXEC 2 people [March 22]

"... I'm thinking of implementing an experimental EXEC processor to handle a more ... PL/I-like language. ... This is of course the *dual* of the EXEC/EXEC 2 languages, in that literals are identified, rather than variables/control words, but ... EXECs nowadays often seem as complex as programs ... and that therefore literals are often a very small percent of the tokens in an EXEC".

→ first specification for **REX** [March 29]

First specification (1)

- 5 pages of introduction and rationale
- 10-page language description
- 4 pages of examples
- Eleven instructions (IF, DO WHILE/UNTIL, SELECT, QUEUE, PUSH, PULL, SAY, EXIT, RETURN, TRACE ON/ERROR, ERROR)—plus a proposal for REX (INTERPRET)
- 8 special variables (BLANKS, DATE, N, NL, Q, RC, RETCODE, TIME); DATE, Queued, and TIME became functions.

First specification (2)

- There were three example programs (including bugs). For example:

```
/* Send file to a local user */
Pull name fn ft fm;
CP SPOOL PUN name CLASS A;
if rc¬=0 then do; /* check it worked */
  say name is not a valid userid;
  exit 102; end;
PUNCH Fn Ft Fm;
CP SPOOL PUN * CLASS A;
```

etc.

Refinement

- Hundreds of pieces of mail refined the initial specification
- Arguments such as DO...END versus IF...ENDIF
- Version 0.01 to Les Koehler and Ray Mansell [May 21]
- Initial specification had evolved to 30-page reference manual [by June]
- Rapid growth of features, following suggestions (better tracing, hex strings, nested comments, *etc.*)

Key features

- ✦ Control structures
- ✦ Parsing—PULL and decompose into words
- ✦ Fluidity of symbols (multiple uses)
- ✦ Concatenation with blank
- ✦ Alternative quotes for literals
- ✦ Lack of “boilerplate”
- ✦ Case-insensitive comparisons (later removed)
- ✦ Case-preservation for literals (later removed)
- ✦ Tracing

Performance

- ✦ Comparisons with EXEC, EXEC 2, and PL/I
- ✦ Test loop: 3.31 seconds (on S/370 model 155):

```
i=0
do 2000
   i=i+2
   end
```

1995:

- ✦ 0.19s on a 486/33MHz PC

A typical week— the first of 1980

- Requests for a more PL/I-like DO instruction, with the ability to step a control variable
- Requests for subscripts (rejected because, among other things, "... the obvious syntax, using square brackets, is not practical because so few people have brackets on their keyboards")
- A user contributed a draft quick-reference card
- Positive feedback:

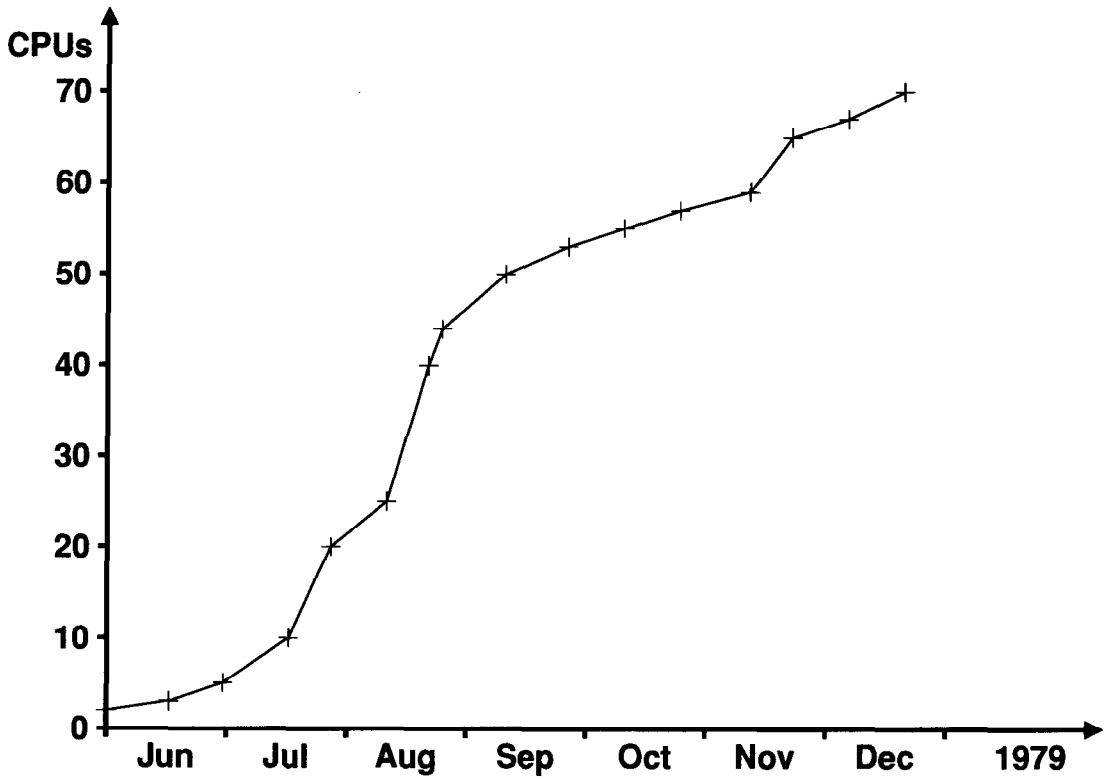
  "REX is getting some really good press around here. People really sit up and take notice, but wonder why someone didn't do it 30 years ago"

Development and usage report

"The value of this communication with other programmers and users cannot be underestimated. Without the communications provided by the network, REX would never have been developed."

- ✦ 10,000 lines of assembler, 5,000 of documentation
- ✦ 27 man-weeks (1000 hours)
- ✦ Only evenings and weekends—when response time was good and interruptions were few.

Growth chart



Retrospective—design errors

- ✦ Comparison should have been case-insensitive
- ✦ DO should have been split into DO...END and LOOP...END
- ✦ Too much emphasis in the External Data Queue
- ✦ Parsing is something of a compromise

Retrospective—successes

- ✦ Deliberate minimizing of "boilerplate" and punctuation, and notations in general
- ✦ Hardware independence and robustness
- ✦ Upgradeable language (keywords only reserved in context)
- ✦ String support (especially "blank operator")
- ✦ Associative arrays (stems)
- ✦ Decimal arithmetic
- ✦ Use of the electronic network for rapid design evolution

1980-1984

- 30 internal releases
- Customers, led by **SLAC**, ask for REX
- Name changed to REXX
- VM/SP 3, with REXX, announced and shipped world-wide (1983)

Help!

There are some **omissions** in the following.

Please let me know of them (and any corrections)—I'll incorporate in a WWW page soon.

2.5

1985-1988

- First non-IBM implementation (Charles Daney, 1985)
- *The Rexx Language* published (1985)
- First Unix implementation (Andy Pierce, IBM, 1985)
- Experimental OS/2 implementation (1986)
- Rexx for VMS VAX (Charles Daney, 1986?)
- IBM SAA has Rexx as "Procedures Language" (1987)
- Amiga Rexx (AREXX, Bill Hawes, 1987)
- Rexx in MVS and TSO/E (1988)
- T-REXX for Tandem (Keith Watts, 1988?)

1989-1990

- IBM and Microsoft agree Rexx is the best scripting language for OS/2 (1989)
- Rexx compiler for VM (IBM Haifa and Vienna, 1989)
- uni-Rexx (The Workstation Group, 1989)
- Rexx 4.00 published (1990)
- First Rexx Symposium (SLAC, 1990)
- Rexx in AS/400 (1990)
- Rexx in OS/2 (1990)

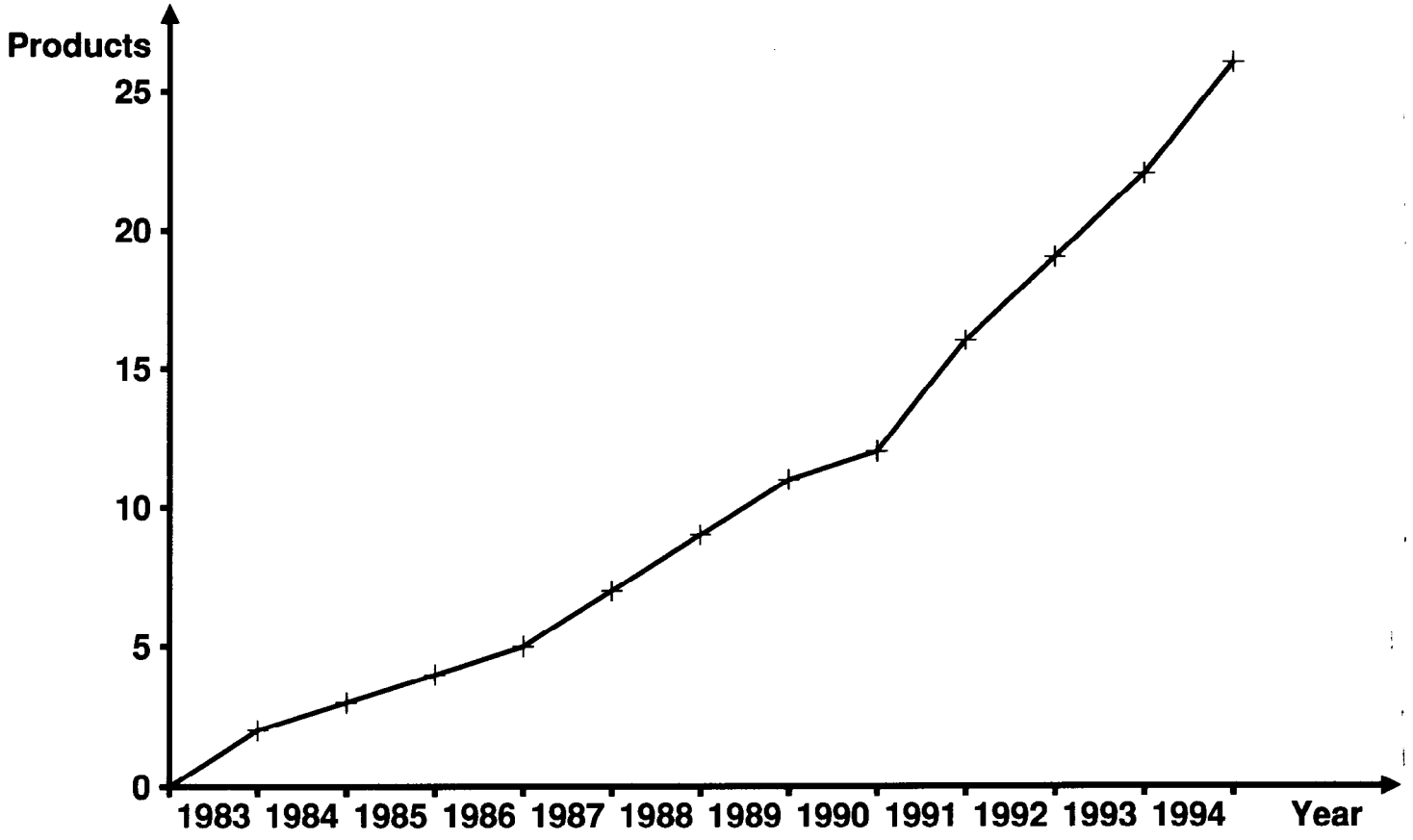
1991-1994

- Work on ANSI standard for Rexx starts (1991)
- Rexx/imc (Ian Collier, 1992)
- Regina Rexx (Anders Christensen, 1993)
- Rexx for VSE (1993)
- Rexx for AIX/6000 (1993)
- Rexx Language Association formed (1994)
- Rexx for Novell NetWare (1994)
- Simware Rexx; Windows, Macintosh, NetWare (1994)
- Rexx for CICS/ESA (1994)

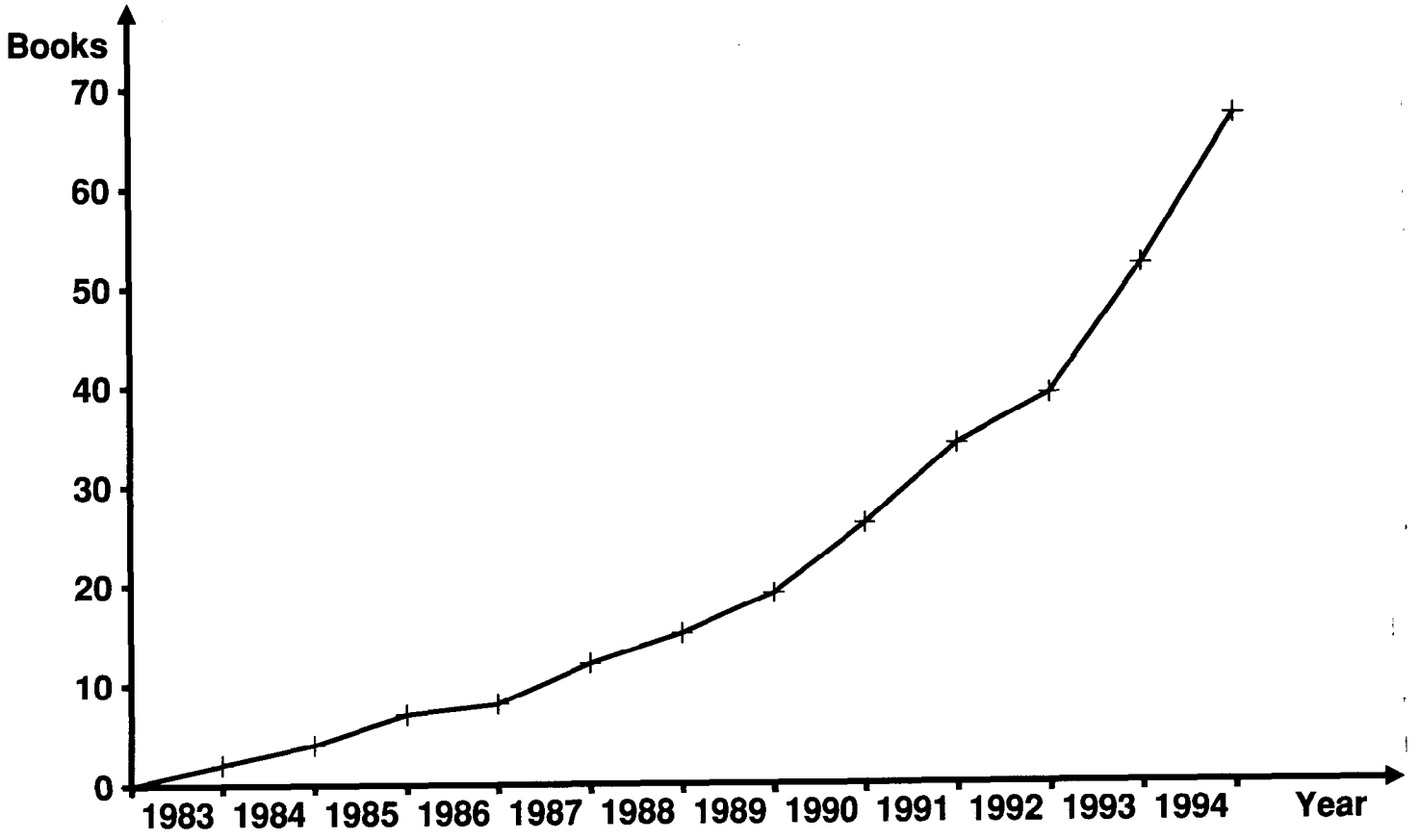
1995

- Rexx in PC-DOS 7, as the "programming language of choice"
- World-Wide Web pages for Rexx; start at:
  - → `http://rexx.hursley.ibm.com/rexx/`
- Object Rexx public beta
- ...and more...

REXX Language Products Available



REXX Books and Manuals



Summary

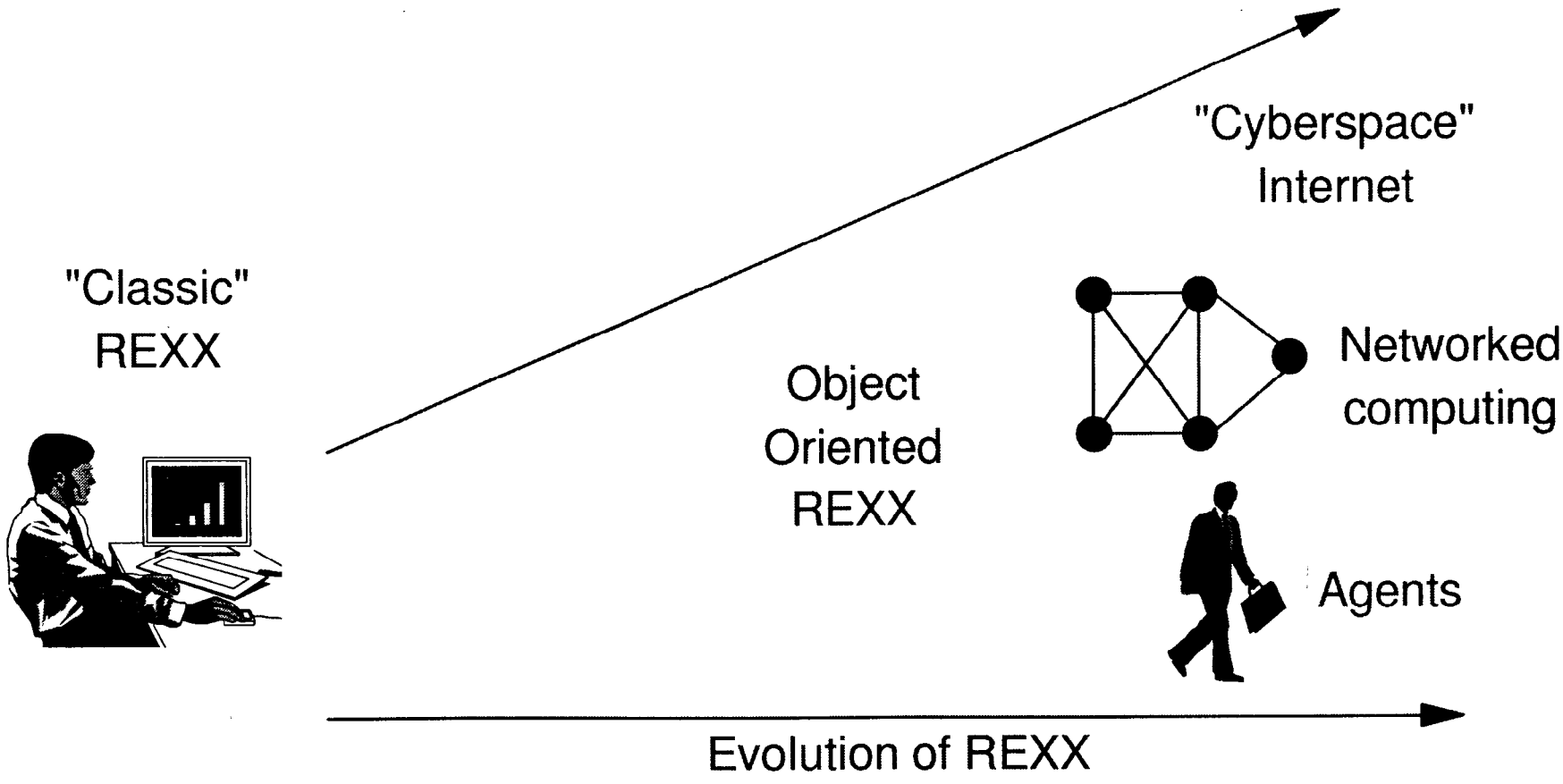
- ✦ Rexx is a carefully designed, purpose-built scripting language
- ✦ Steady growth over 15 years, especially rapid in last 2-3 years
- ✦ Rexx is installed on 15-25 million users' machines
- ✦ Well over 2 million Rexx programmers
- ✦ It wouldn't have been possible without **people**.

The Future of REXX

Tim S. Browne
IBM Endicott

Pages 34-49

The Future of REXX



T. S. Browne 5/1/95 IBM Endicott

REXX Mission

- Continue to support users of "Classic" REXX products
- Continue to evolve "Classic" REXX into Object REXX
- Make Object REXX pervasive across platforms and applications
- Align Object REXX with key technologies and standards
- Facilitate the creation of software agents and the environment they operate in
- Get others to exploit, build, and use our software agent technology

Open Scripting Products

1994 Accomplishments

- OREXX
  - 4/94 OS/2 Beta
  - 10/94 AIX Alpha
  - 11/94 AIX Beta
  - 11/94 OS/2 Developer's Connection
  - 11/94 Windows Alpha
- Classic REXX
  - 3/94 Netware GA
  - 7/94 CICS GA
  - 7/94 DOS GA
  - 12/94 OS/400 GA

REXX Requirements Summary

Requirements#	Description	Current Status	New Status
SOUIZ91007, SALANG90203	REXX for AIX	AIX/6000: Rejected AIX PS/2 Future Objective	AIX/6000: Available as PRPQ#
SALANG90206	Remove 500 char. limit on statement length	Available on OS/2 only. Future objective on other systems	Available on all IBM REXX implentations
SOMVSE93027	IBM should provide a richer suite of debugging tools for REXX	Under study	Future Objective
SAREXX90210	Pull instruction should not type a "?"	Accepted	Available - OS/2 2.0 GA 3/92
SAREXX90211	Document PARSE rules explicity and formally	Accepted	Available
SOCMSX89023	REXX file I/OON Implement CMS REXX	Accepted	Available
SALANG89205	Extend DATE function	Long Range Consideration	Accepted
SALANG89201	Implement REXX File I/O functions that will read/write files both sequentially and randomly	OS/2 Available TSO Long Range Consideration AS/400 Future Objective VM Accepted	VM Available TSO Accepted
SALANG89204, SOCMSX89056	Allow expressions in Compound variables	Long Range Consideration	Accepted
SALAN90204	Share variables between external REXX Functions	Long Range Consideration	Accepted
SAREXX92002	Object Oriented Extensions	Long Range Consideration	Accepted
SAREXX91001	Call through expressions	Long Range Consideration	Accepted
SOCMSX89026	Traverse Stem variables	Long Range Consideration	Accepted
SAREXX91201	Improve Error Messages	Long Range Consideration	Accepted
SOCMSX89008	Passing Stems to Subroutines	Long Range Consideration	Accepted
SAA REXX	Generic Bindings	Long Range Consideration	Accepted
G05CME91003	Date processing in REXX	Long Range Consideration	Accepted

Pervasiveness

Platforms

	Classic	Complier/RTL	OO
OS/2	X		1995
PPC			1995
MVS	X	X	1995
AIX	X		1995
VM	X	X	1996
Windows			1995
LINUX			1995
NetWare	X		
DOS	X		
CICS	X		
AS/400	X		
VSE	X	X (RTL only)	

OEM Platforms

PC DOS
NetWare
Amiga
Windows
DCE/VAX

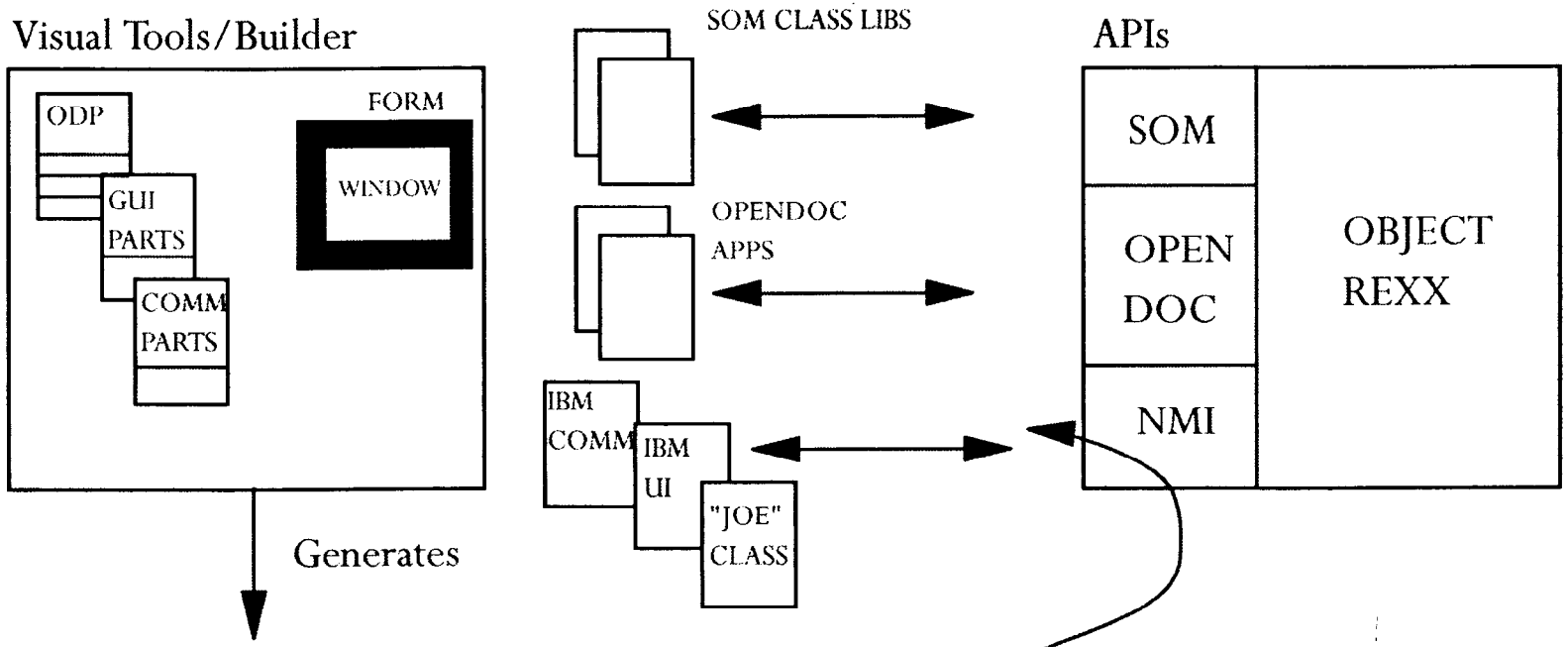
Future Considerations

PDAs

Object REXX Exploits OO Technologies

- Object Oriented Programming
  - Encapsulation, Inheritance, Polymorphism
    - Object REXX supports IBM's SOM
  - Object Oriented Documents
    - Embedded, Composite documents
    - Documents as a new programming model
      - Object REXX is the OPENDOC scripting language
  - Object Oriented Visual Programming
    - Programming via direct manipulation of icons
      - Object REXX support of VisualAge

Object REXX Programming Environment



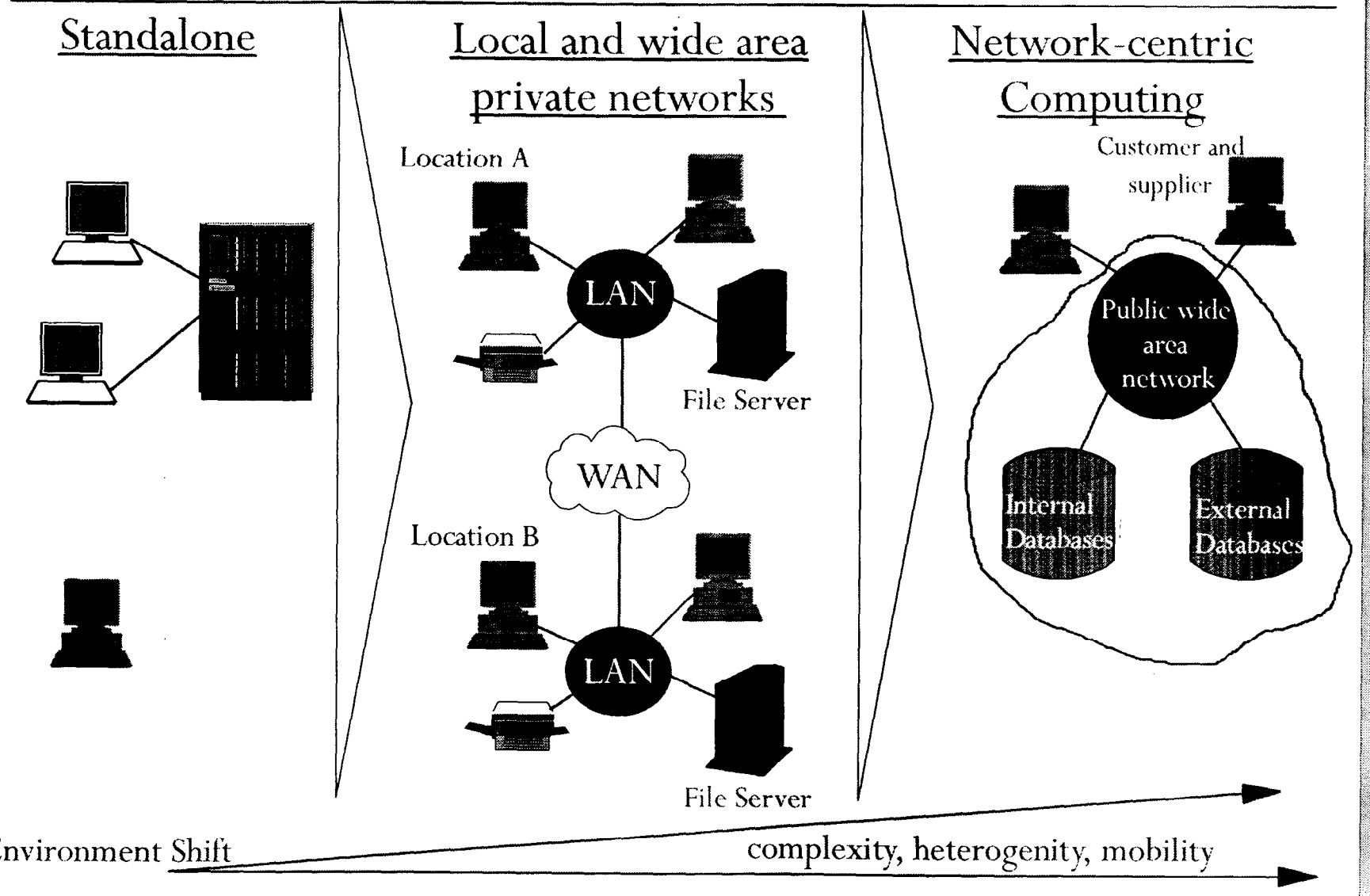
OBJECT REXX SCRIPT

```
VOBJ = .IWINDOWS ~ NEW (...)
VOBJ ~.SHOW ()
VOBJ ~ SIZE (:...)
VOBJ ~ SETBACKGROUND COLOR (..)
        .
        .
```

Functional Requirements - Importance Ratings - Planned Availability

OpenDoc	1.0
Debugger	1.4
Class Browser	1.5
Visual Programming	1.7
Visual Builder	1.8
Export Classes to SOM	2.0
OLE Support for Windows Version	2.8

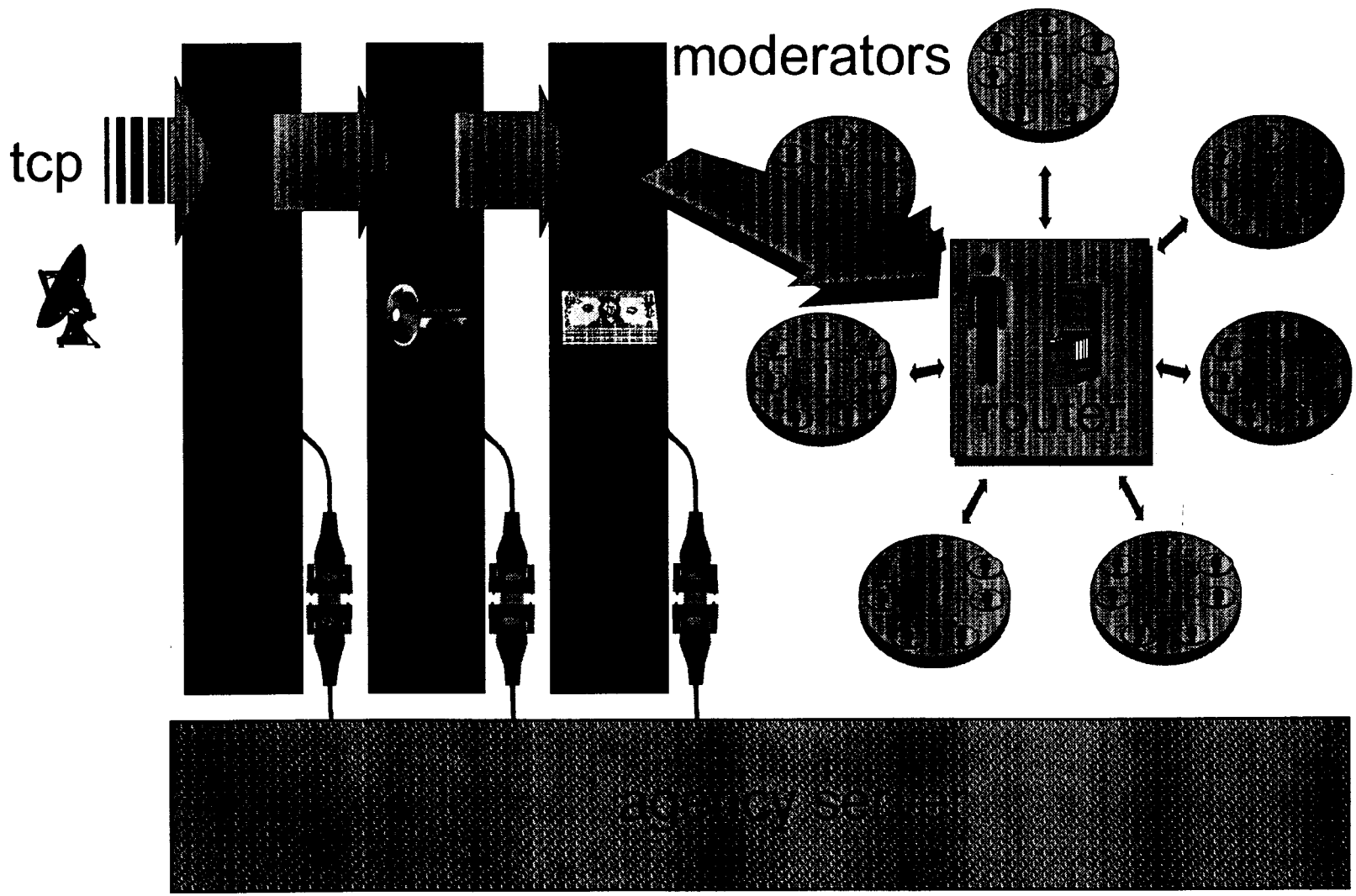
Shift Towards Network Centric Computing



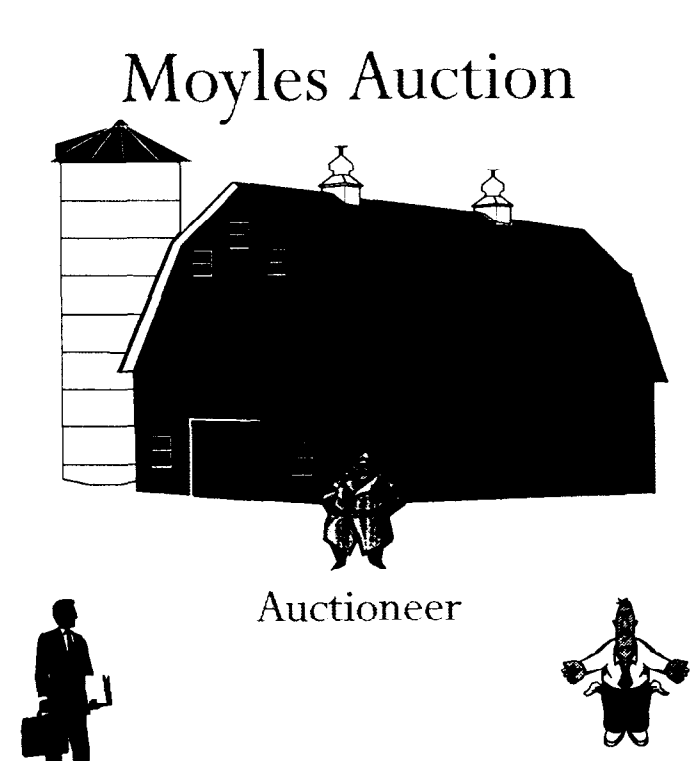
Making "The Customer" Centric in Network Centric Computing

- Agent Services...
  - Personalized Information Retrieval
    - News Services
    - Mail Filtering
    - Internet Surfing
      - Travel Information
      - Medical Information
      - etc.
  - Personalized Event Notification
    - Network Mangement
    - Reservation Alerts
    - Personal Scheduler
  - Personal Transaction Services
    - Banking
    - Business Services
      - Food Services
      - Retail Stores
      - etc.
- Agent "Meeting Places"...
  - Store Fronts
  - Information Databases
  - Trading Floors

A Model for Agents and Agent "Meeting Places"



Demonstration



Internet
TCP/IP
WWW Explorer

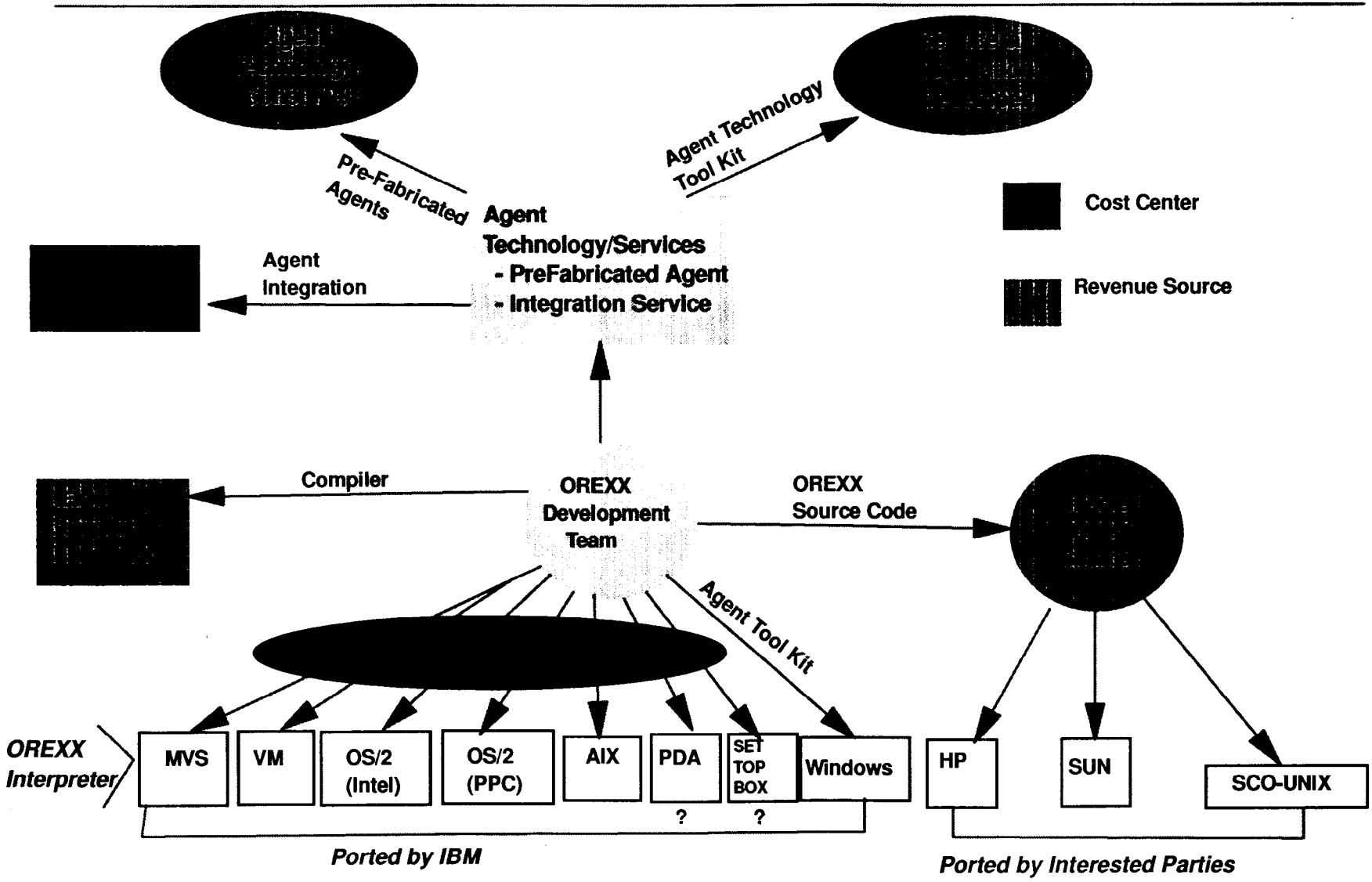
Auction is **Good Test** of Agent Technology

- Agent working on Users behalf
- Agents have guildelines ($100 maximum bid)
- Agent interaction at meeting place (bidding process)
- Value of a moderator (auctioneer)

Demonstrates

- Agent Technology
  - Mobile Agents
- Meeting Place
  - Cooperative Processing

REXX Business



Summary

- REXX has a bright future:
  - Substantial progress made against existing requirements
  - REXX language is being extended to other platforms
  - REXX is evolving it's application development role for network centric computing

Problems "the Shift" Creates

- Access to needed information is:
  - Difficult due to complexity of the environment
    - numerous network services with a multitude of online services
    - mobility in work force increasing
  - Not always timely and/or relevant
    - information overload
    - insufficient technology
  - Labor and effort intensive
    - manual

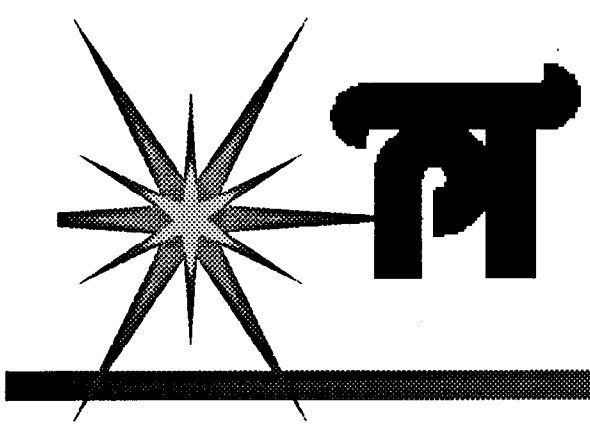
Our Approach to the Problem

- Provide "Agent Technology" where agents act on behalf of end users
- Turn data sources into "Meeting Places" where agents go to perform assigned tasks
- Turn end-users into information consumers rather than information seekers

Problems and Issues Writing REXX Compilers

Markus Pelt-Layman
Pelt Industries

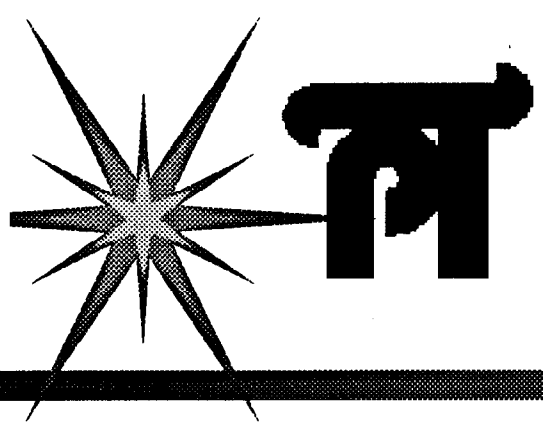
Pages 50-66



Pelt Industries

RexxAnne™

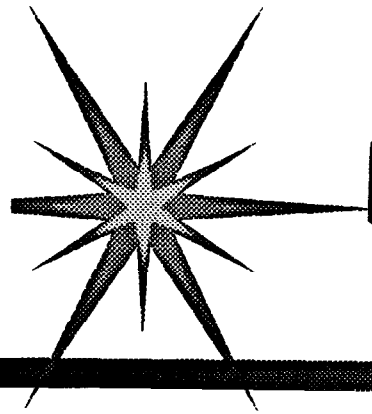
Problems and Issues writing REXX Compilers



Pelt Industries

History

- 1976 **TEACH** language interpreter (Honeywell)
- 1979 **DPS** command language (JES2 look-alike)
- 1985 **Intercept** aka **AF/Operator** command language
  (*TSO CLIST language look-alike for operations automation*)
- 1986 **OPS/MVS** REXX interpreter contract
  (*ADDRESS ISPEXEC, automation rules, cmd/response API*)
- 1987 Windows **Net*Edit** and **Elan Workstation** contracts
- 1989 OPS/MVS REXX **External Product Interface**
- 1992 OS/2 Smalltalk GUI front-end *(REXX EHLLAPI)*
- 1994 ***REXXANNE*** full-time



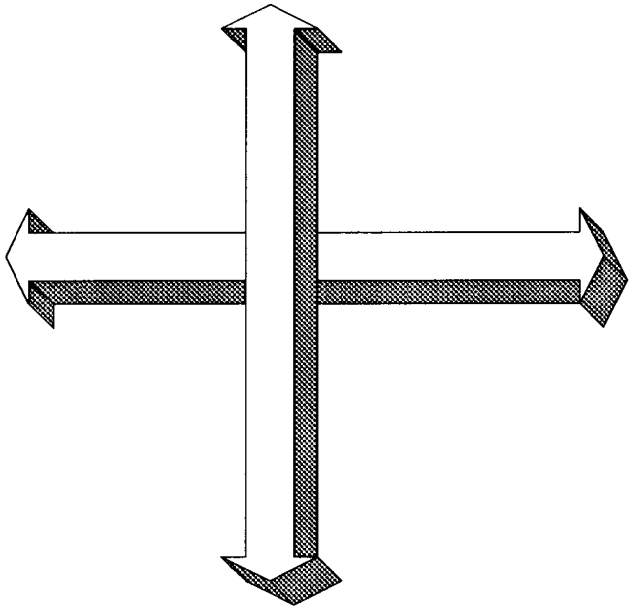
WHY REXXANNE ?

- I love REXX's simplicity and power
- Serious development requires a compiler
- I need
  - **Speed**
  - **Portability**
  - **High-level GUI framework**

SOFTWARE FORCES

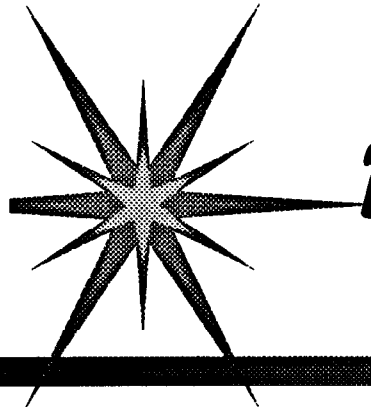
Speed

More hardware/
software platforms



Portability

More APIs



PORTABILITY

I LANGUAGE DEFINITION

TRL/ANSI REXX/Object REXX

II DEVELOPMENT ENVIRONMENT

Portable interpreter/compiler, IDE

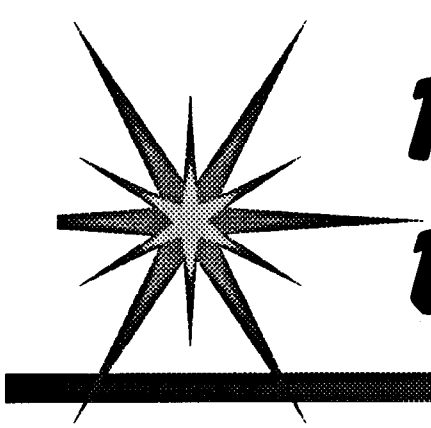
III RUNTIME ENVIRONMENT

Portable run-time lib (built-in functions)

Portable ADDRESS environments (cmd/response)

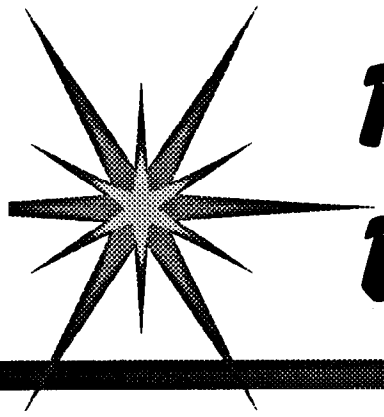
IV OBJECT CLASS HIERARCHY

Portable base class library, GUI class library



REXXANNE DESIGN GOALS

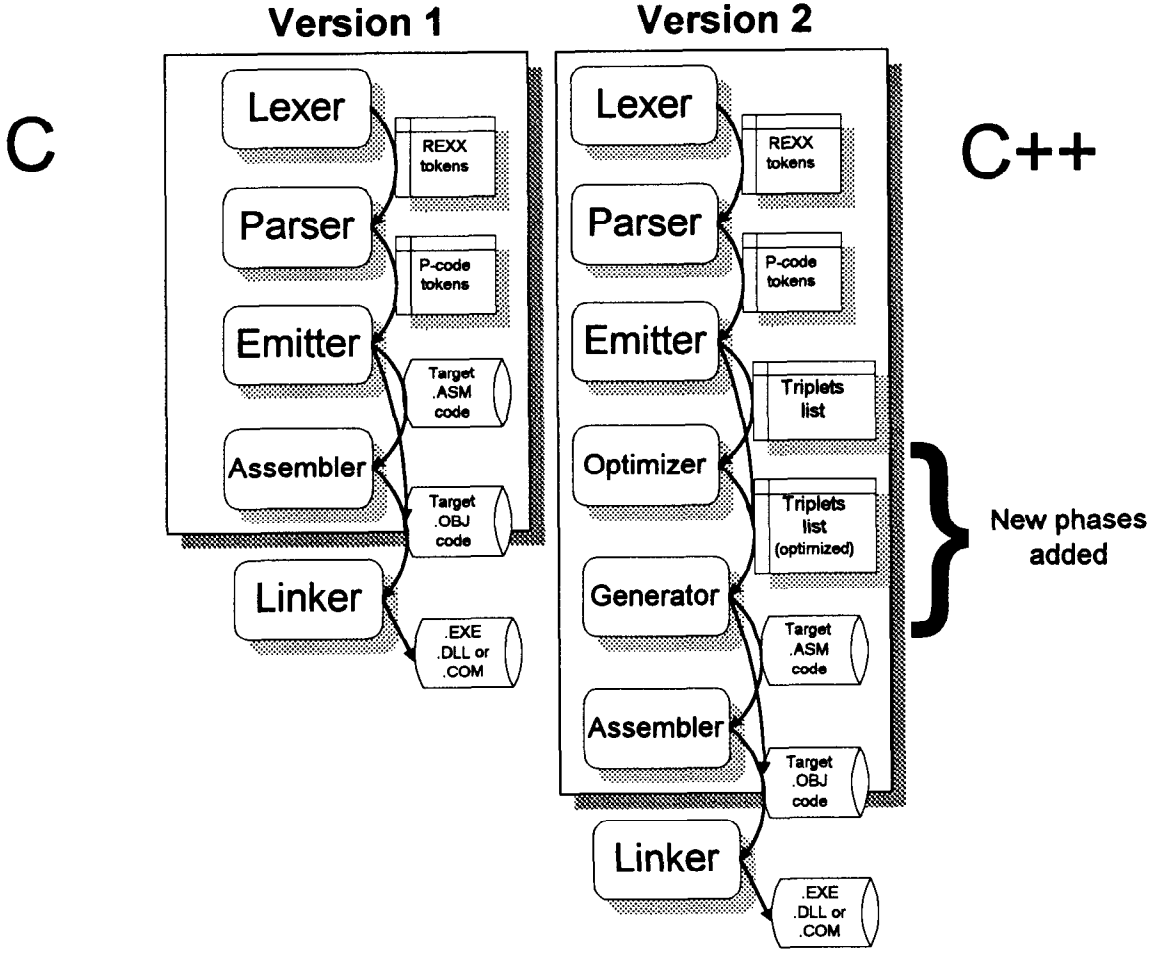
- Shorten development through portability
  (no new learning on new platform)
  - Cross-platform compiler
  - Cross-platform IDE
  - Cross-platform run-time library
  - Cross-platform base class library
  - Cross-platform GUI class library
- Shorten execution time
  through compilation and optimization



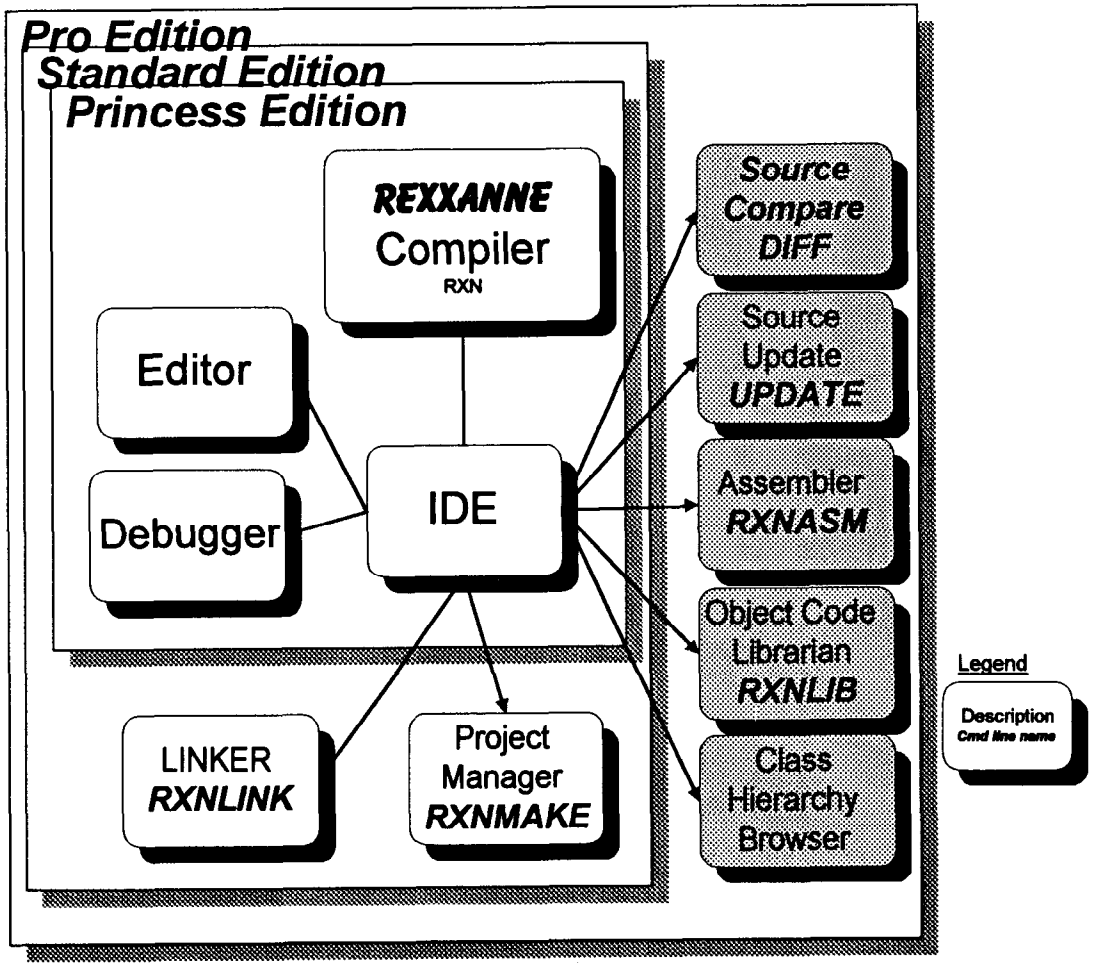
REXXANNE DESIGN ISSUES

- Runtime licensing (INTERPRET)
- Run "interpreted" during development, compiled in production
- GUI portability versus platforms' native look and features (synthesis problem)

REXXANNE CORE COMPILER

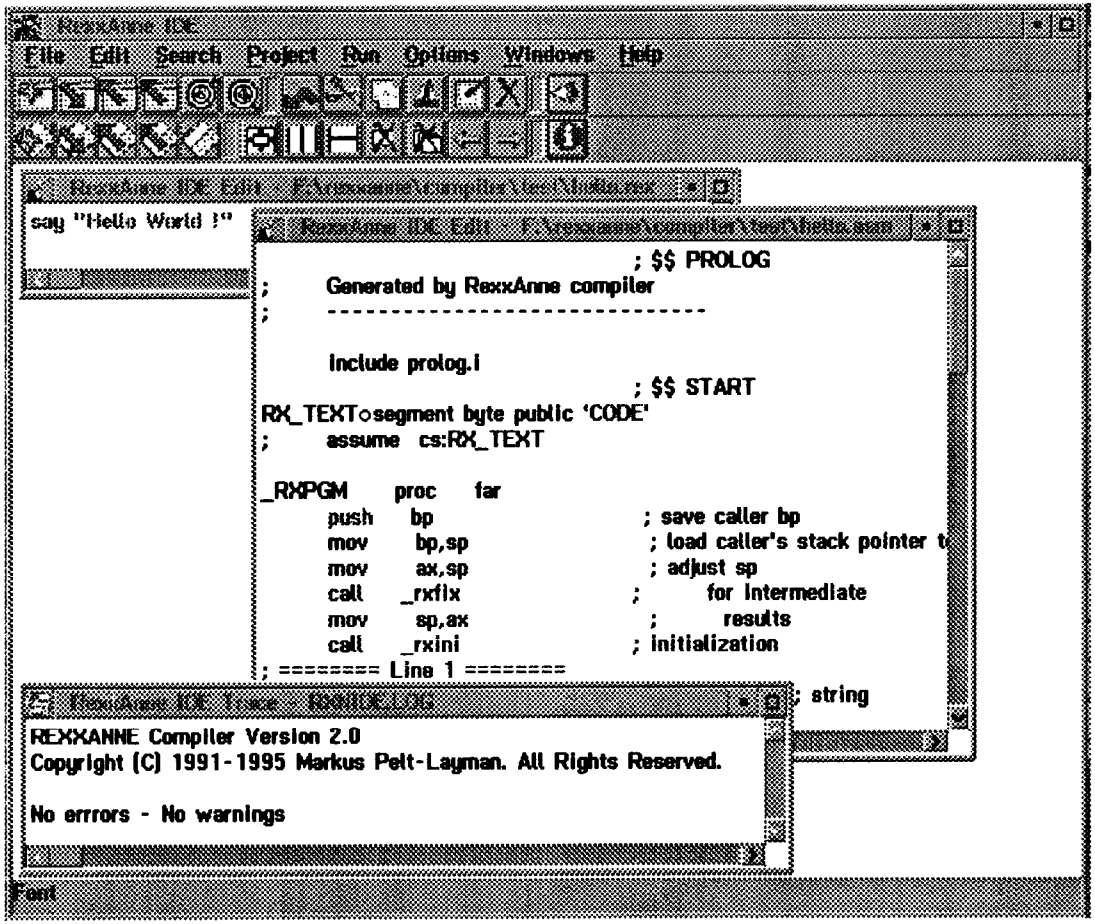


REXXANNE
DEVELOPMENT ENVIRONMENT



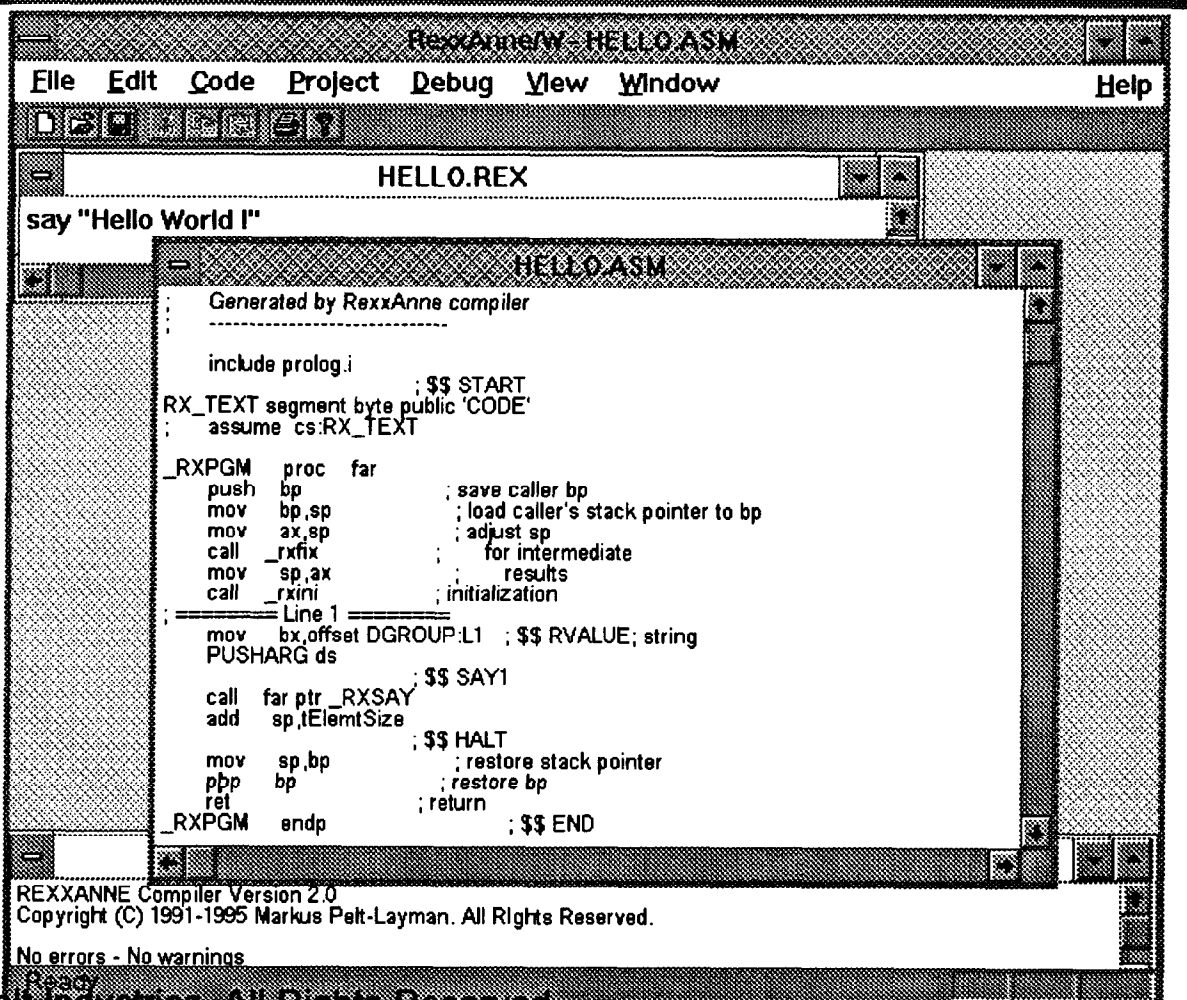
REXXANNE

OS/2 IDE



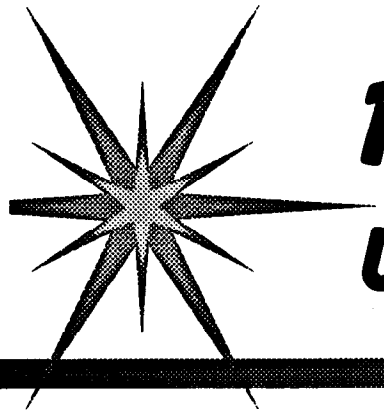
REXXANNE™

WINDOWS IDE



REXXANNE™ CURRENT STATUS

- We are **WAY** behind schedule
  - under-staffed
  - under-capitalized
  - too ambitious a vision ?
- Lack of user interest/support in REXX

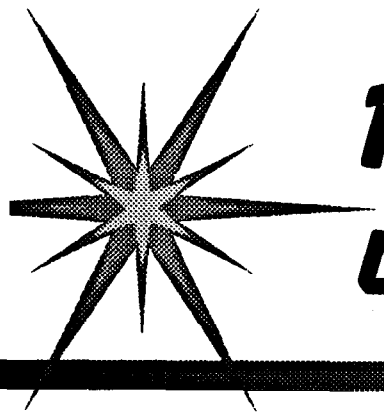


REXXANNE
UNDERSTAFFING

- Task out work
  - Lenny Koff - online tutorial & doc
  - John Kastner - OS/2 IDE
  - Billy Jack - runtime library
  - Find Windows/Windows NT/Windows 95 guru
- Cut down on vision

REXXANNE
UNDERCAPITALIZED

- Generate revenue
  - INTER REXX subscriptions
  - InfoREXX sales
  - Enterprise REXX sales
  - Miscellaneous REXX product sales
  - Beta sales (free upgrade)
- Look for business partners
- Look for capital



REXXANNE

LACK OF USER INTEREST

- Generate interest in REXX language
  - INTER REXX newsletter
  - Get articles published in other magazines
  - Prod IBM and others to better marketing
- Publish information on REXX
  - InfoREXX multimedia help file
  - Novice tutorial
- Generate interest in **REXXANNE**
  - comp.lang.rexx newsgroup participation
  - MSJ advertisements

REXXANNE

FUTURE

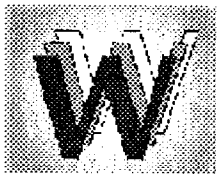
- Continue REXX evangelism
- Release "early bird" version (free upgrade)
- Share development of new features with other vendors (IDE, APIs)
- Get users involved in development
- Continue long-term commitment to REXX

Writing CGI Scripts for WWW Using Rexx

Les Cottrell
Bebo White
Stanford Linear Accelerator Center

Pages 68-99

Writing CGI Scripts For



Using

Les Cottrell and Bebo White, SLAC
6th International REXX Symposium
May 2, 1995

Start

Writing WWW CGI Scripts in REXX

This presentation may be found at:

`http://www.slac.stanford.edu/~bebo/rexx/title.html`

Next

What is WWW?

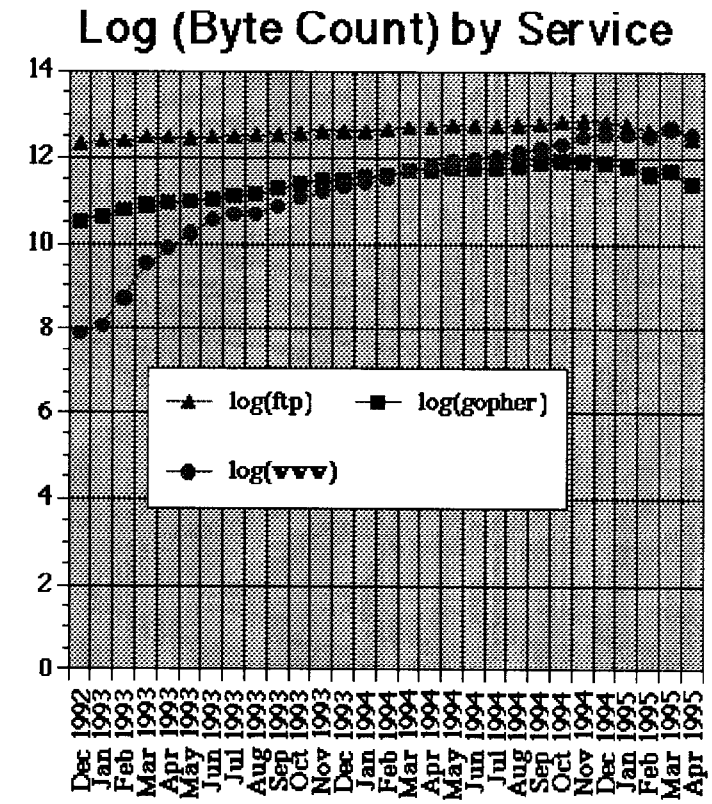
The largest service on the Internet

- The Internet is like the road system
- WWW is like the parcel delivery service

- 27,000 current WWW sites
- Number now doubling every 53 days
- 5 million documents stored in WWW sites

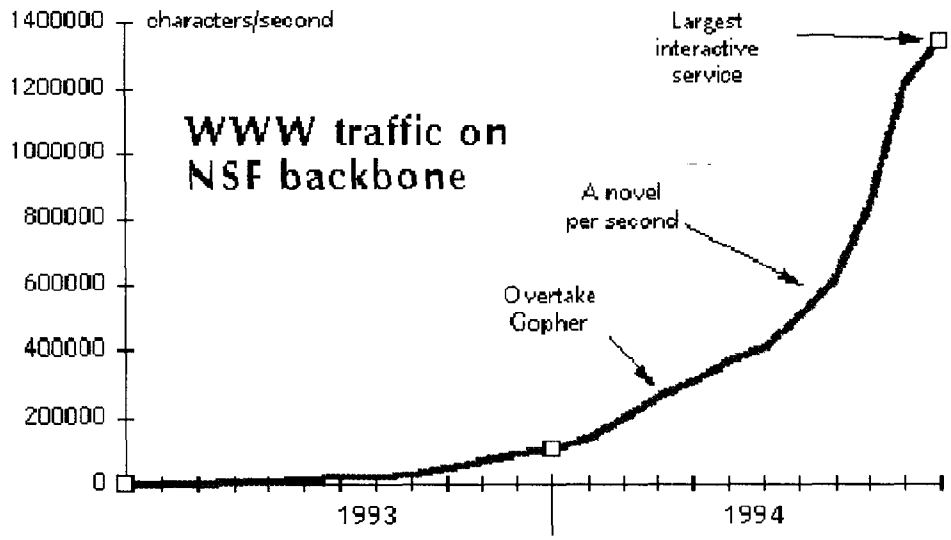
Source:Quoted in *BusinessWeek*, February 27, 1995

[Next]



Statistics provided by Merit NIC Services machine
Graph by: James E. Pitkow, pitkow@cc.gatech.edu

|Next|

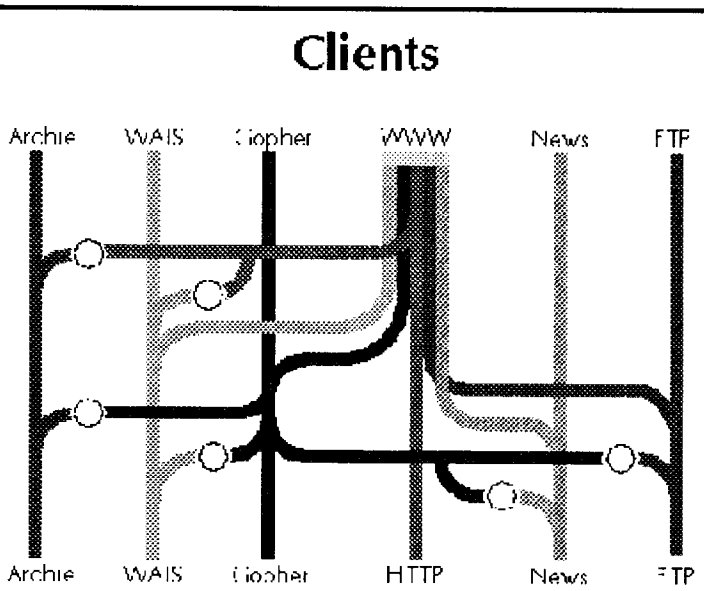


"Over the past three years the traffic on the NSF backbone has increased from 1 TB per month to 18 TB, with a good portion attributable to WWW services." - *Vinton Cerf*

|Next|

Integration of earlier Internet systems

- Gopher, NEWS, Archie, WAIS, ftp, ... are all seamlessly available



[Next]

Format independent

- HTML is *not a format* but a way of structuring documents
- Formatted documents are available through their native applications

---

Multiple Media

- Images
- Sound
- Movies
- Launching of sessions (video, telnet, conferencing, ...

[Next]

Some Neglected Uses of WWW

By the provider:

- CWIS
- Search engines
- Data Base interfaces
- Collaborative work
- Special formats

By the user:

- Home pages
- Structuring of own local information
- Participation (delurking?)

|Next|

What is CGI?

The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.

Gateways are programs (called CGI scripts) that serve data which is not directly readable by a client program, such as a database of high energy physics preprints or personnel information, and convert it to an acceptable form, such as an HTML page, a PostScript file, some images, or a combination of these.

Familiar examples of Web applications which use CGI scripts are fill-out forms, interactive graphics (e.g., banners, maps, and menubars, etc.) and search engines (e.g., Yahoo, Lycos).

[Next]

How Do Gateways Operate?

Gateways can be run by themselves, but are designed to be run by the HTTP server. The server sets up an environment which is

- secure (to prevent willful damage by users)
- informative (to allow communication between httpd server and gateway)
- contained - so all output is sent by the server back to the client

An example of a database query

Another example of a database query

Communication between client, server, and CGI gateway

This is a searchable index. Enter search keywords:

SLAC SPIRES: HEP Preprint database search

Send corrections to: LIBRARY@SLAC.STANFORD.EDU . Use QSPIRES search language (see examples below). Note that there is no possibility for iterative search (yet) in WWW. Therefore, when needed, combine several criteria in a single request. Need more help ?
Examples:

```
show indexes
find author perl, m and title tau and date before 1980
find title prefix supercollid and date 1994
find t so2n+1                                [finds title SO(2n+1)!]
find bulletin-bd  hepex and date-added 9/94
find cn mark-iii and date after march 1991
browse coden physics letters
find c phlta, 70b, 487              [finds citations of a paper!]
find a abe and date 1988 (using wwwcite          [shows citations!]
find author gross, david and journal phys rev
browse affiliation caltech
find af cal tech and date 1994 (result
browse topic higgs
find topic higgs boson or title higgs and date 1-95 (using wwwbrief
browse last ppf
find ppf 9442 (seq rs
```

To learn more about authors, institutions, or acronyms, try **WHOIS, WHEREIS, or WHATIS:**

```
whois ginsparg
whereis cern
whatis sld
```

See also other SPIRES databases, or SPIRES News, or the SLAC home page.

19 April 1995

SLAC Phone Directory: Search Form

SLAC 13 Apr 1995
This fill-out form can be used to search the SLAC phone directory (previously called BINLIST). Fill in the entries you know, leave others blank. Family name, First name and E-mail ID will support truncated searches, using an asterisk (eg: john*).

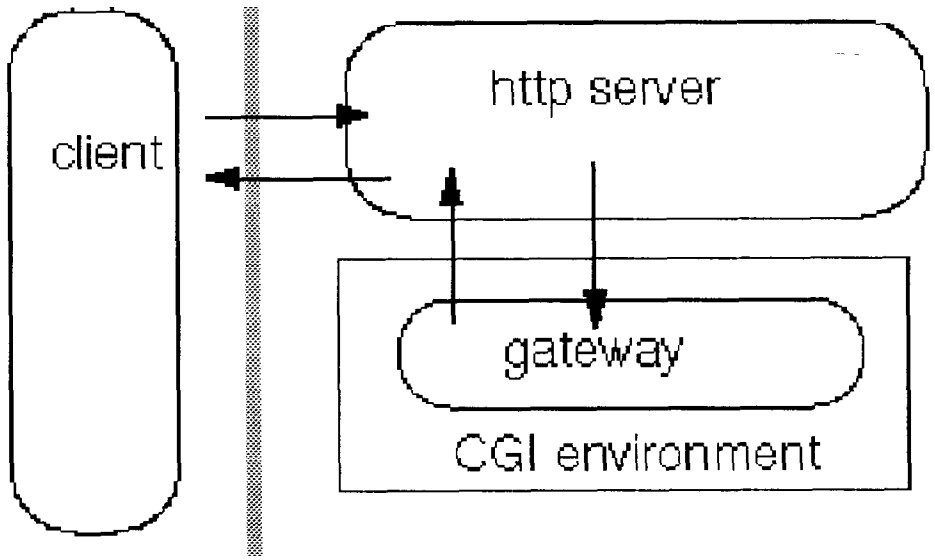
```
    Family name:
     First name:

       E-mail ID:

Work Extension:
```

HELP

The original of this form was created by Evelyn Aviles-Hernandez.

Diana Gregory

Communication Between Client, Server, and CGI Script



Next

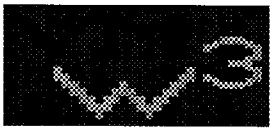
Design of a CGI Script

1. Design the CGI script interface:
   - Use of the <ISINDEX> or <ISMAP> in an HTML page;
   - Design of a fill-out form using HTML;
   - Specify the format of a CGI script-compliant URL which can be used as a link in an HTML page.

2. Design and code a CGI script which performs the following:
   - Argument decoding;
   - Argument validation;
   - Processing (including error handling);
   - Output and cleanup;

INext|

CGI Scripts In REXX

- Webshare - the VM HTTP server 
  - Written entirely in REXX;
  - Supports CGI scripting in REXX;
  - Information at

    `http://ualvm.ua.edu/~troth/software/cmshttpd.html`

- Today's discussion is limited to use with Unix-based servers;

|Next|

Providing Input to a CGI Script

- The `QUERY_STRING` environmental variable
  - `QUERY_STRING` is defined as anything following the first "?" in the script-invoking URL;
  - Generated automatically by the HTML tag <ISINDEX> or a fill-out form (with method=GET);
  - Encoded to include URL information with spaces converted to "+" and special characters according to their hex encoding;
  - Example of `QUERY_STRING` use.

|Next|

Providing Input to a CGI Script

- The **PATH_INFO** environmental variable
  - CGI allows additional context-specific information to be embedded in a URL;
  - This additional information is contained in the **PATH_INFO** environmental variable;
  - The information in **PATH_INFO** is not encoded;
  - Example of `PATH_INFO` use.

[Next]

Providing Input to a CGI Script

- Standard Input (stdin)
  - Used with fill-out forms using method=POST;
  - The environmental variable **CONTENT_LENGTH** contains the amount of data to be read from standard input;
  - Example of `CONTENT_LENGTH` use.

|Next|

Guide to Writing CGI Scripts in REXX or Perl

9 Apr, 1995

Contents

Introduction

This Guide is aimed at people who wish to write their own WWW executable scripts using WWW's *Common Gateway Interface* (CGI). Since there are security and other risks associated with executing user scripts in a WWW server, the reader may wish to first view a document providing information on a SLAC Security Wrapper for users' CGI scripts. Besides improving security, this wrapper also simplifies the task of writing a CGI script for a beginner.

The CGI is an interface for running external programs, or gateways, under an information server. Currently, the supported information servers are HTTP (the Transport Protocol used by WWW) servers.

Gateway programs are executable programs (e.g. UNIX scripts) which can be run by themselves (but you wouldn't want to except for debugging purposes). They have been made executable to allow them to run under various (possibly very different) information servers interchangeably. Gateway programs conforming to this specification can be written in any language, including REXX or Perl, which produces an executable file

Getting the Input to the Script

The input may be sent to the script in several ways depending on the client's *Uniform Resource Locator* (URL) or an *HyperText Markup Language* (HTML) Form:

- `QUERY_STRING` Environment Variable

QUERY_STRING is defined as anything which follows the first ? in the URL used to access your gateway. This information could be added by an HTML ISINDEX document, or by an HTML Form (with the GET action). It could also be manually embedded in an HTML hypertext link, or *anchor*, which references your gateway. This string will usually be an information query, e.g. what the user wants to search for in databases, or perhaps the encoded results of your feedback Form. It can be accessed in REXX by using `String=GETENV('QUERY_STRING')`
or in Perl by using
`$string=$ENV('QUERY_STRING');`

This string is encoded in the standard URL format which changes spaces to +, and encoding special characters with %xx hexadecimal encoding. You will need to decode it in order to use it. You can review the REXX or Perl code fragments giving an example of how to decode the special characters.

If your server is not decoding results from a Form, you will also get the query string decoded for you onto the command line. This means that the query string will be available in REXX via the `PARSE ARG` command, or in the Perl `$ARGV[n]` array.

For example, if you have a URL `http://www.slac.stanford.edu/cgi-bin/foo?hello+world` and you use the REXX command `PARSE ARG Arg1 Arg2` then `Arg1` will contain `"hello"` and `Arg2` will contain `"world"` (i.e. the + sign is replaced with a space).
In Perl `$ARGV[1]` contains `"hello"` and `$ARGV[2]` contains `"world"`. If you choose to use the command line to access the input, you need to do less processing on the data before using it.

- `PATH_INFO` Environment Variable

Much of the time, you will want to send data to your gateways which the client shouldn't muck with. Such information could be the name of the Form which generated the results they are sending.

CGI allows for extra information to be embedded in the URL for your gateway which can be used to transmit extra context-specific information to the scripts. This information is usually made available as "extra" information after the path of your gateway in the URL. This information is not encoded by the server in any way. It can be accessed in REXX by using
`String=GETENV('PATH_INFO')`, or in Perl by using `$string=$ENV('PATH_INFO');`

To illustrate this, let's say I have a CGI script which is accessible to my server with the name `foo`. When I access foo from a particular document, I want to tell foo that I'm currently in the English language directory, not the Pig Latin directory. In this case, I could access my script in an HTML document as:

```
<A HREF="http://www/cgi-bin/foo/language=english">foo</A>
```

When the server executes foo, it will give me `PATH_INFO` of `/language=english`, and my program can decode this and act accordingly.

The PATH_INFO and the QUERY_STRING may be combined. For example, the URL:
`http://www/cgi-bin/htimage/usr/www/img/map?404,451`
will cause the server to run the script called `htimage`. It would pass remaining path information

"/usr/www/img/map" to htimage in the PATH_INFO environment variable, and pass "405,451" in the QUERY_STRING variable. In this case, htimage is a script for implementing active maps supplied with the CERN HTTPD.

- Standard Input

  If your Form has METHOD="POST" in its FORM tag, your CGI program will receive the encoded Form input on standard input (stdin in Unix). The server will NOT send you an EOF on the end of the data, instead you should use the environment variable CONTENT_LENGTH to determine how much data you should read from stdin. You can accomplish this in REXX by using
  ```
  In=CHARIN(,1,GETENV('CONTENT_LENGTH')), or in Perl by using
  read(STDIN,$in,$ENV{'CONTENT_LENGTH'});
  ```

You can review the REXX Code Fragment giving an example of how to read the various form of input into your script.

Decoding Forms Input

When you write a Form, each of your input items has a NAME tag. When the user places data in these items in the Form, that information is encoded into the Form data. The value each of the input items is given by the user is called the value.

Form data is a stream of name=value pairs separated by the & character. Each name=value pair is URL encoded, i.e. spaces are changed into plusses and some characters are encoded into hexadecimal.

You can review the REXX or the Perl code fragment giving examples of decoding the Form input.

Sending Document Back to Client

CGI programs can return a myriad of document types. They can send back an image to the client, an HTML document, a plaintext document, a Postscript documents or perhaps even an audio clip of your bodily functions. They can also return references to other documents (to save space we will ignore this latter case here, more information may be found in NCSA's CGI Primer). The client must know what kind of document you're sending it so it can present it accordingly. In order for the client to know this, your CGI program must tell the server what type of document it is returning.

In order to tell the server what kind of document you are sending back, CGI requires you to place a short header on your output. This header is ASCII text, consisting of lines separated by either linefeeds or carriage returns followed by linefeeds. Your script must output at least two such lines before its data will be sent directly back to the client. These lines are used to indicate the MIME type of the following document

Some common MIME types relevant to WWW are:

- A "text" Content-Type which is used to represent textual information in a number of character sets and formatted text description languages in a standardised manner. The two most likely subtypes are:
  - text/plain: text with no special formatting requirements.

    - `text/html`: text with embedded HTML commands
- An `"application"` Content-Type, which is used to transmit application data or binary data. Two frequently used subtypes are:
    - `application/postscript`: The data is in *PostScript*, and should be fed to a *PostScript* interptreter.
    - `application/binary`: the data is in some unknown binary format, such as the results of a file transfer.
- An `"image"` Content-Type for transmitting still image (picture) data. There are many possible subtypes, but the ones most often used on WWW are:
    - `image/gif`: an image in the GIF format.
    - `image/xbm`: an image in the X Bitmap format.
    - `image/jpeg`: an image in the JPEG format.

In order to tell the server your output's content type, the first line of your output should read:
`Content-type: type/subtype`
where `type/subtype` is the MIME type and subtype for your output.

Next, you have to send the second line. With the current specification, THE SECOND LINE SHOULD BE BLANK. This means that it should have nothing on it except a linefeed. Once the server retrieves this line, it knows that you're finished telling the server about your output and will now begin the actual output. If you skip this line, the server will attempt to parse your output trying to find further information about your request and you will become very unhappy.

You can review a REXX Code Fragment giving an example of handling the `Content-type` information.

After these two lines have been outputted, any output to `stdout` (e.g. a REXX SAY command) will be included in the document sent to the client.

Diagnostics and Reporting Errors

Since `stdout` is included in the document sent to the, diagnostics diagnostics outputted with the SAY command will appear in the document. This output will need to be consistent with the `Content-type: type/subtype` mentioned above.

You can review a REXX Code Fragment giving an example of diagnostic reporting.

If errors are encountered (e.g. no input provided, invalid characters found, too many arguments specified, requested an invalid command to be executed, invalid syntax in the REXX exec) the script should provide detailed information on what is wrong etc. It may be very useful to provide information on the settings of various WWW Environment Variables that are set.

You can review a REXX Code Fragment giving an example of error reporting and Typical Output Generated from such a code fragment.

My First REXX CGI Script

To get your Web server to execute a CGI script you must:

- Write the script and save it somewhere. For example let's say we write a trivial REXX script called `cgi1.rxx` and save it in our home bin directory (e.g. in `/u/sf/cottrell/bin/cgi1.rxx`).
- Make the script executable by your Web server. On Unix this is done using the `chmod` command, e.g.
  - `chmod o+x /u/sf/cottrell/bin/cgi1.rxx`
- Get your *Web-Master* to add a rule to the Web server's rules file to allow the Web server to execute your script. You can look at SLAC WWW Rules to see how SLAC URLs currently map to the UNIX file system.

The Web-Master will want to insure that Security Aspects of your script have been addressed before adding your script to the Rules file.

Acknowledgements

Much of the text on the Common Gateway Interface and Forms comes from NCSA documents. Useful information and text was also obtained from *The World-Wide Web: How Servers Work*, by Mark Handley and John Crowcroft, published in ConneXions, February 1995.

Les Cottrell 15 Jan 1995
[Top | Suggestion Box | Disclaimer]

```
/* ********************************************************************** */
/* Some browsers insert ASCII codes (preceded by a %) for some characters */
/* such as space or +.  We replace them by the appropriate character      */
/* N.B. the encodings maybe in upper or lower case (e.g. %2F=/ or %2f=/   */
/* ********************************************************************** */
Hex='%2b=+&%20=+&%2f=/&%3f=+&'
/*   %2b=+ %20=  %2f=/ %3f=? */
Hex=Hex||TRANSLATE(Hex) /*Allow for upper case*/
IF POS('%',Str)/=0 THEN DO/*Any %s in input*/
 DO UNTIL Hex=''/*Check for hex codes*/
  PARSE VAR Hex Code'='Char'&'Hex; Out=''
  PARSE VAR Str Pre (Code) Str
  DO WHILE (Str /== '')
     Out=Out||Pre||Char
     PARSE VAR Str Pre (Code) Str
  END /*DO*/
 Str=Out||Pre
 END /*DO*/
END /*IF*/
```

Reading HTML Forms Input in REXX

```
PARSE ARG Parms
StdinFile='/tmp/wrap-stdin'_GETPID()/* Get unique name*/
Script=GETENV('SCRIPT_NAME')
...
/* ************************************************** */
/* Read the input from the various possible sources      */
/* Note that we preserve or save all                     */
/* input in case we need to send it to another command.  */
/* If so we can restore the stdin for the called command */
/* by calling it using the REXX command:                 */
/* ADDRESS UNIX script '<' StdinFile                     */
/* ************************************************** */
IF GETENV('REQUEST_METHOD')="POST" THEN DO
   IF LINES()=0 THEN,
      CALL Exit 400, Script': null input from POST method!<br>'
   Line.0=0
   DO L=1 BY 1 WHILE LINES()>0
      Line.L=LINEIN(); Line.0=Line.0+1
      IF L>1 THEN Fail=LINEOUT(StdinFile,Line.L)
      ELSE        Fail=LINEOUT(StdinFile,Line.L,1)
      IF Fail=0 THEN LEAVE L
   END L
   Fail=LINEOUT(StdinFile) /* Close the file*/
END
Path_info=GETENV('PATH_INFO')/*Insert path info in front*/
IF Path_info/='' THEN,
   Cmd=SUBSTR(Path_info,2) Cmd/*Remove leading / from path*/
Temp=GETENV('QUERY_STRING')
IF Temp='' THEN Temp=Parms
IF Temp/='' THEN Cmd=Cmd Temp /* Insert query info at end*/
IF Cmd='' THEN,
   CALL Exit 402, Script': no input provided!<br>'
...
```

Example of Decoding HTML Forms input in REXX

```
/* ************************************ */
/* Input from a form comes in the form:  */
/* name1=value1&name2=value2             */
/* Here we decode the input into an      */
/* array of names and values.            */
/* ************************************ */
DO I=1 BY 1 UNTIL Input=''
   PARSE VAR Input Name.I'='Value.I'&'Input
END I
```

Handling Content-type Info in REXX

```
/* ************************************************/
/* Code fragment to set the Contentype subtype based */
/* on the  file type (as determined by the characters*/
/* following the last period in the filename).       */
/* ************************************************/
FileName='/u/sf/cottrell/public_html/cgi.html'
...
L=LASTPOS('.',Filename); Type=''
IF L>0 THEN DO
  IF LENGTH(FileName)>L THEN
     Type=TRANSLATE(SUBSTR(FileName,L+1))
END

SELECT
   WHEN Type='HTM' | Type='HTML' THEN
      SAY 'Content-type: type/html'
   WHEN Type='PS'                THEN
      SAY 'Content-type: application/postscript'
   WHEN FIND('TXT RXX PL FOR C',Type)/=0 | Type='' THEN
      SAY 'Content-type: type/text'
   ...
   OTHERWISE DO
      SAY 'Content-type: type/html'; SAY ''
      CALL Exit 409, 'Unknown Content-type="'Type'".'
   END
END
SAY ''
...
```

Reporting CGI Diagnostics in REXX

```
/* ************************************ */
/* Code fragment for reporting CGI Script */
/* diagnostic                           */
/* ************************************ */
PARSE SOURCE $ArchName $Efn $Fn .
...
Debug=1
SAY "Content-type: text/html";  SAY
...
IF Debug>0 THEN SAY,
   $Efn': PATH_INFO="'GETENV('PATH_INFO')'".<br>'
```

Reporting CGI Errors in REXX

```
/****************************************** */
/*Code Fragment for REXX CGI Error Reporting*/
/****************************************** */
  ADDRESS 'COMMAND'; SIGNAL ON SYNTAX
  PARSE Arg Parms
  ...
  IF QueryInput='' THEN,
     CALL Exit 490,'No input given!<br>'
  ...
/*****************************************/
/*Rex will jump to this error exit if a    */
/*syntax error occurs. It returns the user */
/*ito the line in the exec with the error. */
/*****************************************/
Syntax:
   PARSE SOURCE Arch . $Fn .
   CALL Exit 499, 'Syntax error on line',
      SIGL 'of' $Fn'. Line="'SOURCELINE(SIGL)'"'
   ...
Exit: PROCEDURE EXPOSE Debug Parms
/* ************************************
| Exit - Assumes Content-type: text/html
| ************************************ */
PARSE ARG Code, Msg
SAY '<title>'GETENV('SCRIPT_NAME')' error' Code'</title>'
SAY '<h2>Error Code' Code 'reported by'
SAY GETENV('SCRIPT_NAME')'.</h2> The WWW utility on'
SAY '<tt>'GETENV('SERVER_NAME')
SAY ':'GETENV('SERVER_PORT')'</tt>that you are using'
...
SAY 'which reports the following error:'
IF Msg/='' THEN SAY '<hr><h1><code>'Msg'</code></h1>'
IF Debug>0 THEN DO
   SAY '<hr>The complete environment follows:<p><pre>'
   ADDRESS Unix "tcsh -c printenv"
   SAY '<br>Arguments="'Parms'".'
   SAY '</pre>'
END
SAY '<hr>['
SAY '<a HREF="/slac.html">SLAC Home Page</a> |'
SAY '<a HREF="/suggestion/cottrell">Suggestion Box</a> ]'
SAY '<address><a HREF="/owner/zaphod">Zaphod</a></address>'
IF Code=0 THEN RETURN
EXIT /*Code*/
```

Error Code 490 reported by cgi-wrap

The WWW utility on `www1.slac.stanford.edu:80` that you are using from your WWW browser (*Mozilla/1.1b1 (X11; international; AIX 2 000003643500)*) on `ATLAS.SLAC.Stanford.EDU` called (using the `GET` method) a Common Gateway Interface (revision `CGI/1.1`) script command wrapper, which reports the following error:

No input given!

[SLAC Home Page | Suggestion Box]
Les Cottrell

Sample Dangerous CGI Script in REXX

```
#!/usr/local/bin/rxx
/* Sample dangerous CGI Script written in */
/* Uni-REXX.                              */

SAY "Content-type: text/plain"; SAY

Query=TRANSLATE(GETENV('QUERY_STRING'),' ','+')

Valid=' abcdefghijklmnopqrstuvwxyz'
Valid=Valid||'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
Valid=Valid||'0123456789-_/.@'
V=VERIFY(Query,Valid)
IF V/=0 THEN DO
   SAY 'Invalid Character('SUBSTR(Query,V,1),
       ')in:"'Query'".'
   EXIT 99
END

ADDRESS UNIX Query

EXIT RC
```

Object REXX: Up Close and Personal

Rick McGuire
IBM Endicott

Pages 100-136

Object REXX (tm): Up Close and Personal

Rick McGuire
Object REXX Development
Endicott, NY

Background

- Work began in 1988
- Prototyped since 1989
- Beta version available on OS/2 Developers Connection Volume 6 (1-800-6DEVCON)
- Complete rewrite of interpreter
- Language architecture "in progress" and subject to change

RexxObject

Why Object REXX?

- Remove limitations of current REXX language
- Bring the power of OO programming to REXX
- Bring the usability of REXX to OO programming
- Extend REXX usage
  - windowing, object manipulation, concurrency, etc.
- Build on large base of existing REXX programs
  - fully upward compatible
- Interact with emerging new technologies such as SOM and OpenDoc

Rexx bject

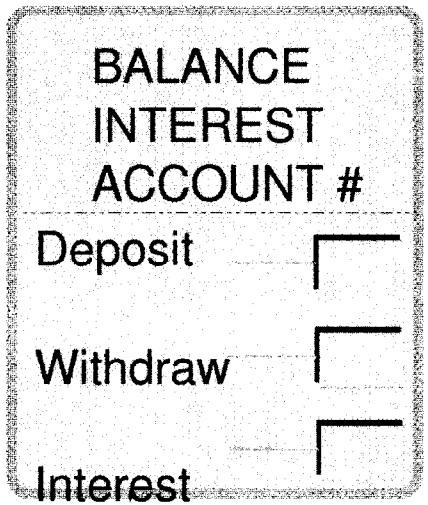
What's New in Object REXX?

- Objects
  - Everything in Object REXX is an object
- Methods
  - Everything that happens in Object REXX is a method
- Messages
  - Everything that happens in Object REXX is caused by a message

Rexx bject

What is an Object?

- Everything in Object REXX!
- Encapsulation of data and code (methods) which operate on data
- Manipulated via messages
  - Code outside object has no direct access to object data
  - Responds to messages by running methods
- Primitive (e.g. string, directory) or programmed
- Automatically reclaimed (garbage collection)



withdraw(156.23)

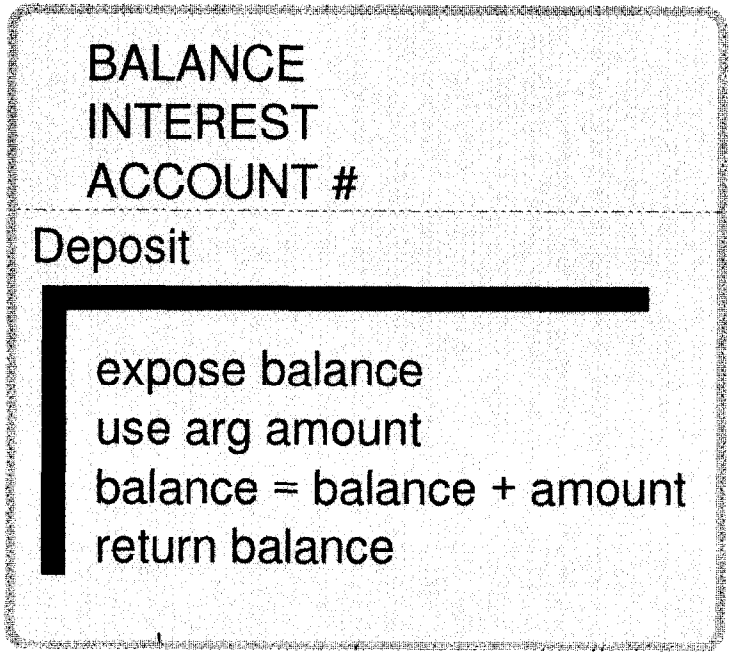
deposit(1457.11)

What is a Method?

- Everything that happens in Object REXX!
- Bits of code that operate on object data
- Similar to subroutines/functions
  - Optionally return results
  - All variables local unless explicitly exposed
- May be private or public
  - Like internal vs. external subroutines/functions
- Defined on object-by-object basis
  - Different objects may have same names for different methods
    - *"Polymorphism"*

What is a Message?

- What causes everything to happen in Object REXX!
- Something "sent" to an object causing the object to run a method
- Message name = method name
- Sender waits for reply
  - Reply may contain returned data



deposit(1457.11)

savings~deposit(1457.11)

1458.11

Messages

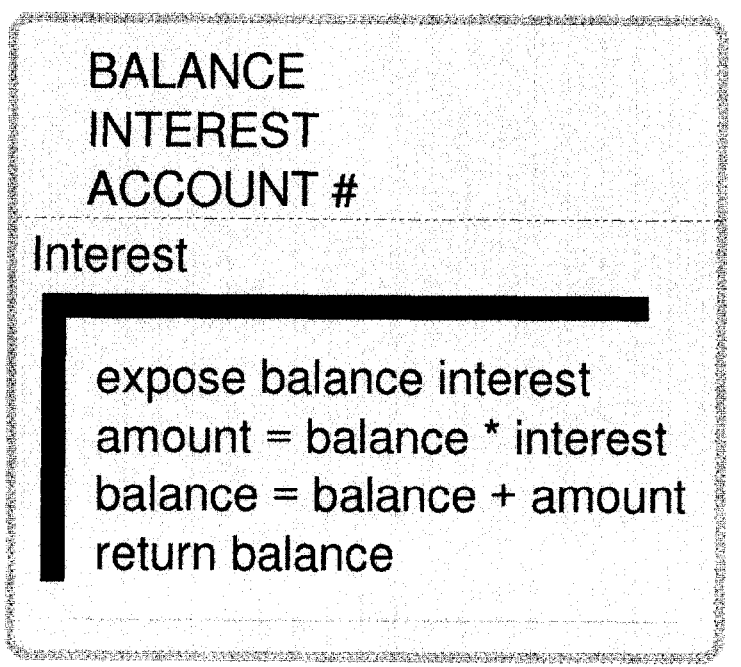
- New syntax:
  - receiver~ message(arguments)
  - receiver~~ message(arguments)
  - receiver[arguments]
- arguments are optional, e.g.:
  - receiver~ message
- May appear as term, instruction, or assignment target
- All REXX operators become messages
  - Can use either syntax

Polymorphism

- Definition: The ability to send the same message to different objects, which may have very different underlying characteristics.
- Powerful feature of object-oriented programming
  - Sender does not need to know internals of receiver
    - ► *Example: "+" method*
  - Allows common usage of common words to improve readability and maintainability
    - ► *Example: PRINT method*

Variables

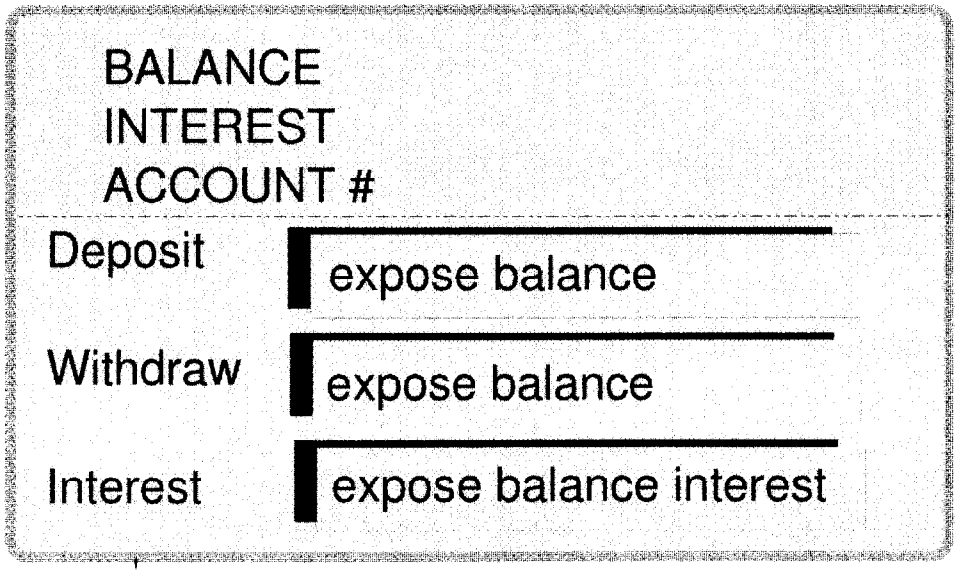
- All variables are references to objects
  - Strings are just one type of object
- Method variables (a.k.a. "local") exist only while method is running
- Object variables last as long as the object does



Rexx bject

EXPOSE Instruction

- Used to expose and create object variables within methods
- Used for sharing between methods, or just for allowing persistence between invocations of same method
- Subsidiary lists also supported
- Dynamically adds to list of object variables
- Must be the first instruction in a method



Rexx bject

Passing Arguments

- Arg and Parse Arg work only with strings
  - All arguments are converted to strings via STRING method
- New instruction: USE ARG name[,name...]
  - Assigns each name to the corresponding object
    - ► *Does not make a copy of the object referred to, only assigns a reference to the variable*
  - This allows a kind of call-by-reference
    - ► *If object can be directly modified (such as stems)*

New Condition Handling

- Significantly enhanced over existing REXX
- New conditions for object oriented needs:
  - NOMETHOD - object cannot find requested method
  - NOSTRING - object with no string value used where string value required
- New ANY condition name for CALL/SIGNAL ON
  - Allows handling of any error not handled by more specific handler
  - Example: NOVALUE raised, no NOVALUE handler ==> ANY trap is invoked

New Condition Handling...

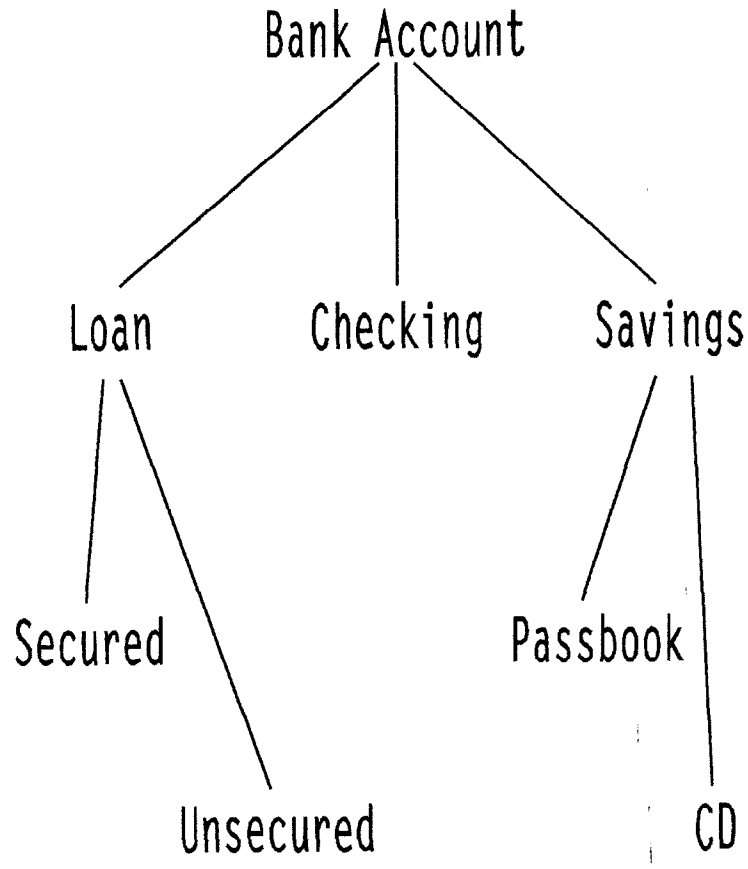
- New user condition support allows users to define own conditions
- New RAISE instruction
  - RAISE condition DESCRIPTION expression
  - "condition" can be any of
    - *rexxcondition*
    - *SYNTAX number*
    - *USER usercondition*
  - "expression" is returned to handler by CONDITION('D')
  - RAISE PROPAGATE passes conditions up to the next call level

Classes

- Need for many objects with same behavior (i.e. methods)
- Use class object to define shared behavior
- Class object is an "object factory"
  - Creates new "instances" with same methods but separate object data
    - ► *e.g. Rick's savings account, Pam's savings account*
- Once created, instances not dependent on classes
  - Methods can be added or replaced per instance
  - Sometimes called "enhanced" objects

Inheritance

- Classes maintained in a hierarchy
- Subclass acquires behavior of superclass and modifies it
- Variables scoped by class
- Allows easy reuse of code
  - programming by differences
- Major benefit of object-oriented programming



Directives

- Purpose: to allow more complex program structures to be contained within a single source file
  - Provides way to identify program entities that previously required separate files
- Object REXX programs can package classes, methods, and routines
  - Routines similar to external functions
- Packages can make objects public
- Programs can identify other programs/packages that they require

Directives

- New packaging directives:
  - ::CLASS classname options -- creates a new class to be used by your program
  - ::METHOD methodname options -- creates methods that are associated with classes
  - ::ROUTINE routinename -- creates functions or subroutines
  - ::REQUIRES programname -- brings in public ::CLASS and ::ROUTINE definitions from another source file

Environment Symbols

Environment

- A look-up table (directory) that is shared among all objects
- Entries created with a name and a value
  - Essentially a global variable pool
- Available via "dot-variables"
  - .array, .true, .false
- Preloaded with Object REXX classes and public objects
  - Public objects include .Input, .Output, and .Environment

Environment Symbols

- Symbols with initial period
- Searches a hierarchy of locations to find a value
  - Classes defined within a program
  - PUBLIC classes accessed via a ::REQUIRES directory
  - The process local directory
  - The global environment directory
- User can explicitly insert entries into environment
  - value(name,object,'')
  - .environment~setentry(name,object)
  - .environment[name] = object

Object-based Concurrency

- Objects are the units of concurrency
- All objects can execute concurrently
- Most object awaiting either a message or a reply
- Actual concurrency achieved via:
  - REPLY instruction
  - START message

Rexx bject

Sequential Execution

Sender

Receiver

Send a message

```
account~deposit(1.98)
```

```
expose balance
use arg amount
balance = balance + amount
return balance
```

Return a result

Processing continues

Concurrent Execution

Sender

Receiver

Send a message

account~deposit(1.98)

Return a result

(Processing continues)

```
expose balance
use arg amount
balance = balance + amount
reply balance
```

(Processing continues)

self~audit('Deposit', amount)

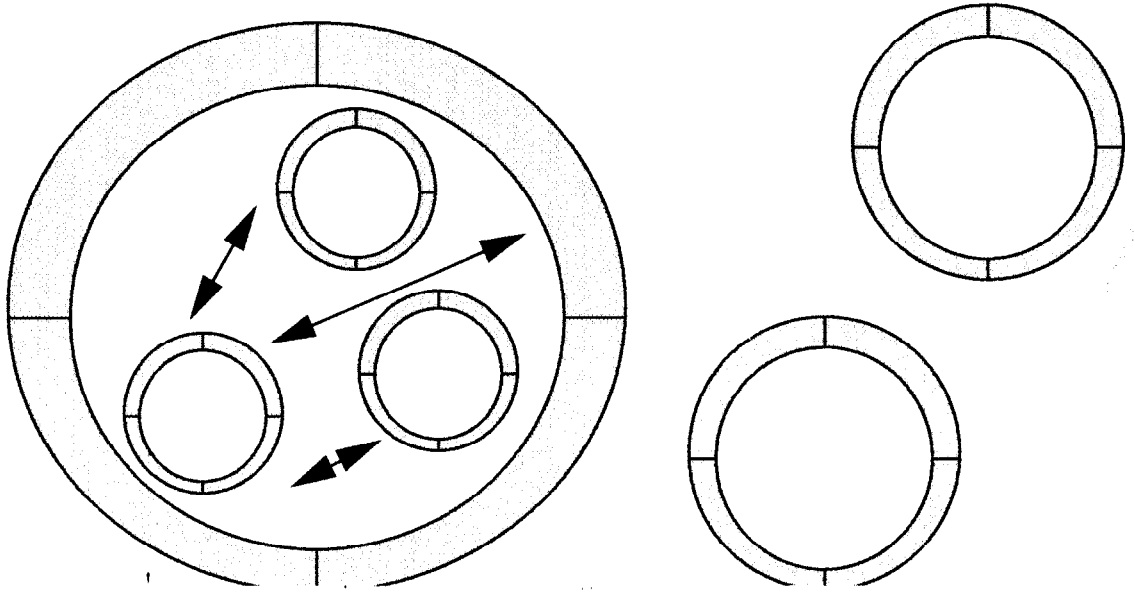
Explicit Concurrency

Sender	Agent	Receiver
agent = account~start('deposit', 1.98)		
Send a message		
Return the agent	account~deposit(1.98)	
Processing continues		
	Send a message	
		expose balance use arg amount balance = balance + amo return balance
balance = agent~result	Return a result	
Request the result		
Return the result		

Rexx bject

Playing Around with Object REXX

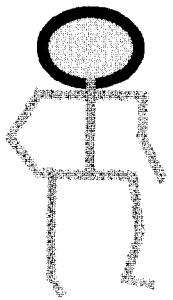
- SOCKET: an OS/2 sockets encapsulation
  - Goal: Clients, Servers without knowing TCP
  - "Server" contains concurrent TCP Objects
    - *"Known Port" socket for service requests*
    - *"Client Sessions" created for each client*
  - "Client" Object(s) request service via TCP



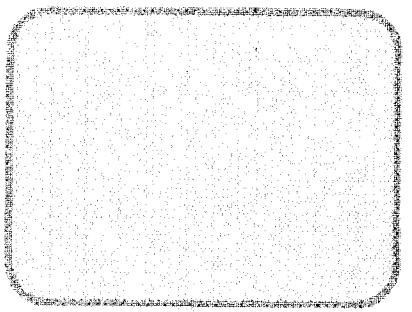
Playing Around, continued

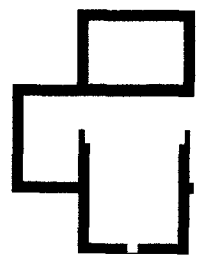
- Socket 'Mirror' TCP C/S Applet:
  - "Framework" classes: 165 lines
  - Client Script: 15 lines
  - Server Script: 27 lines
- Second applet -- 'Toss server':
  - Inherit Socket framework
  - Client Script: 2 changed lines
  - Server Script: 15 new/changed lines

Communications Modes for "Mobile Computing"



- Client Programs
  - Used directly by users
  - Always local
- Server Programs
  - Invoked by client programs
  - May be local or remote
- Agent Programs
  - Work independently for users, even if disconnected

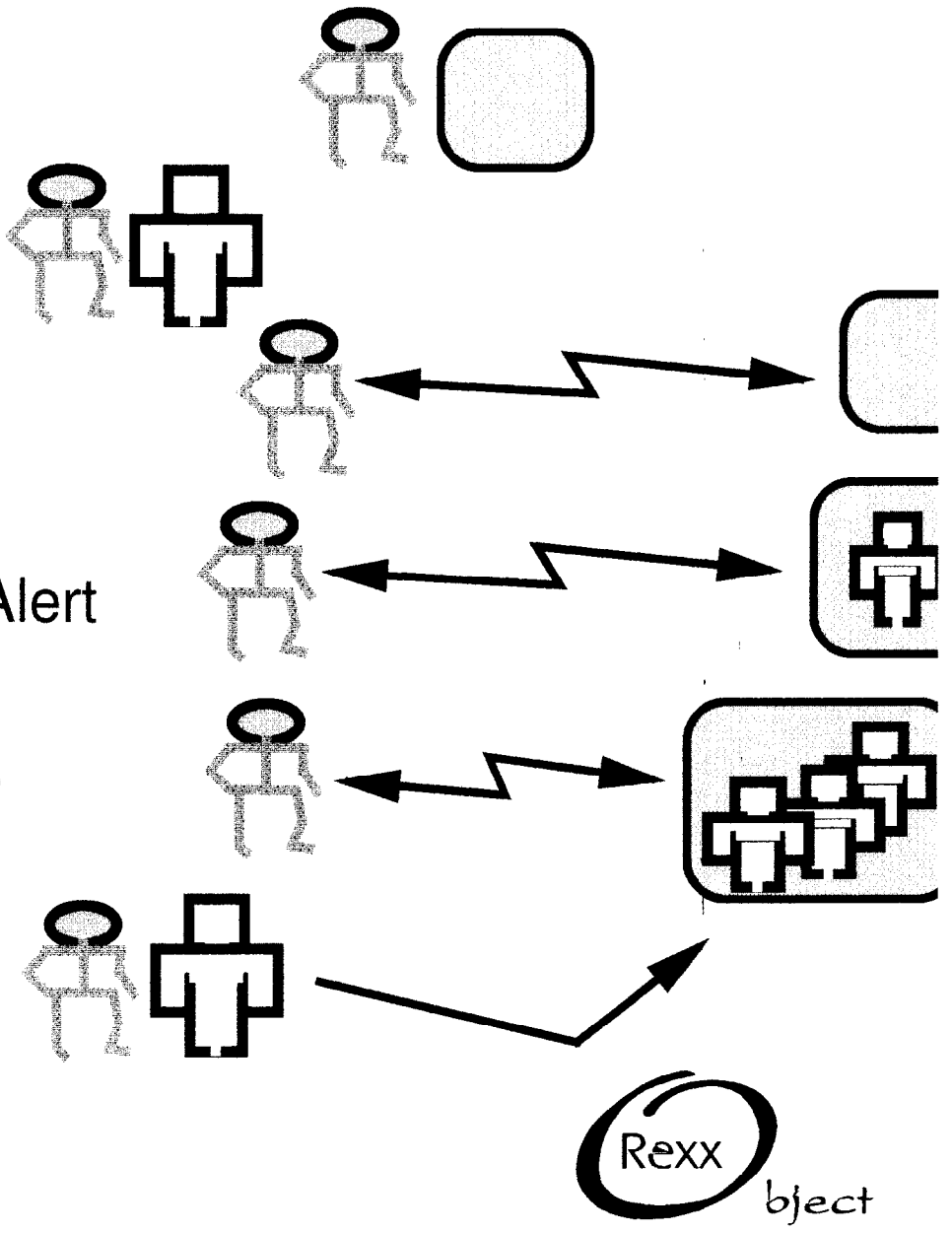




Rexx bject

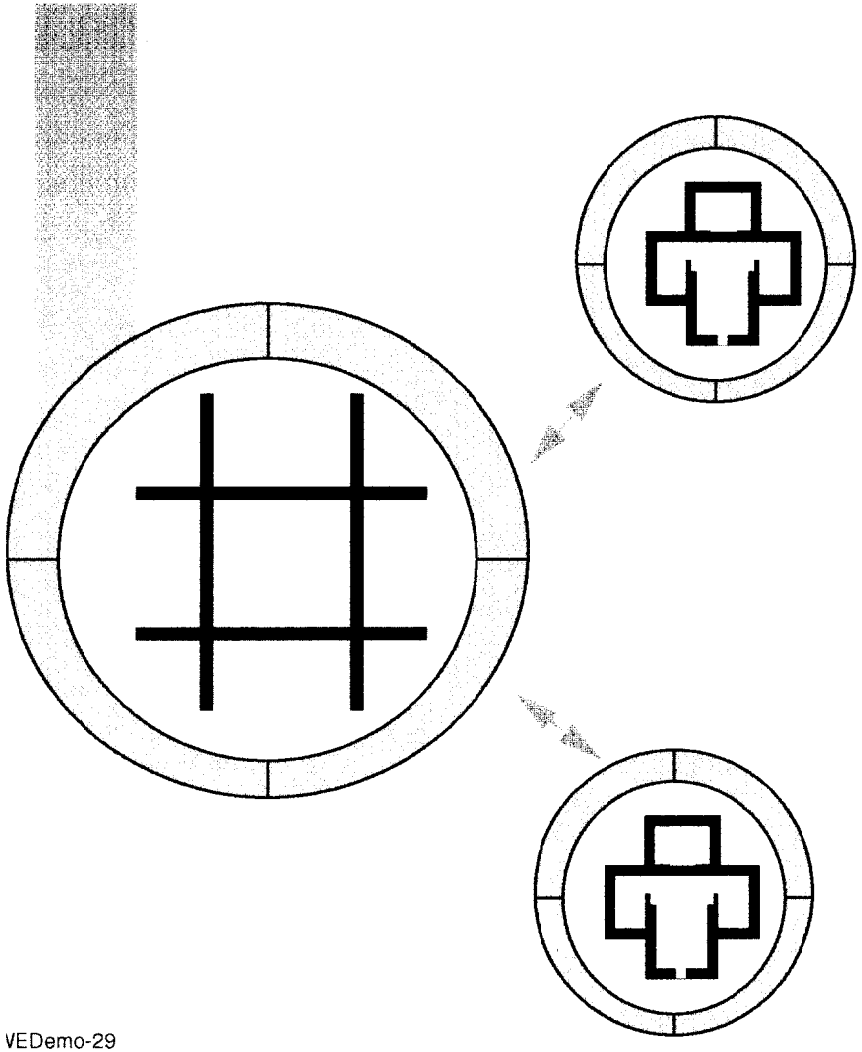
"Mobile Computing": Modes of Communications

- Local Application/Server
  - My word processor
- Local Agent
  - My mail filtering program
- Remote Server
  - My database server
- Remote Server with Agents
  - My Stock Brokerage Auto-Alert
- Remote Interactive Agents
  - Brokers, buyers and sellers
- Wandering Agents
  - Information Scavengers



A Simple Object REXX Program

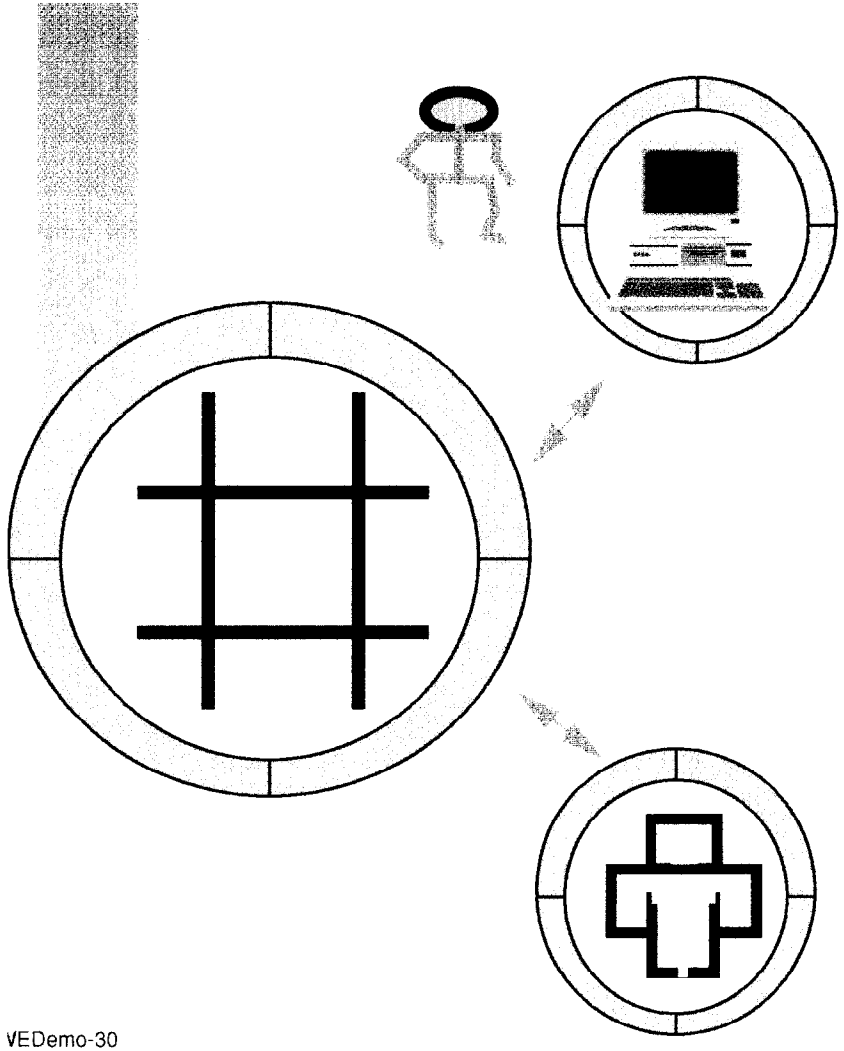
- or, JimBob and Rambo play TicTacToe



"TheGame" manages interacting "Players" on one system

Rexx bject

Adding Interaction to the Game



"Viewer" object

- Same methods as "Players"
- Manages user interface

The Game is now interactive.

OO Jargon: 'polymorphism'

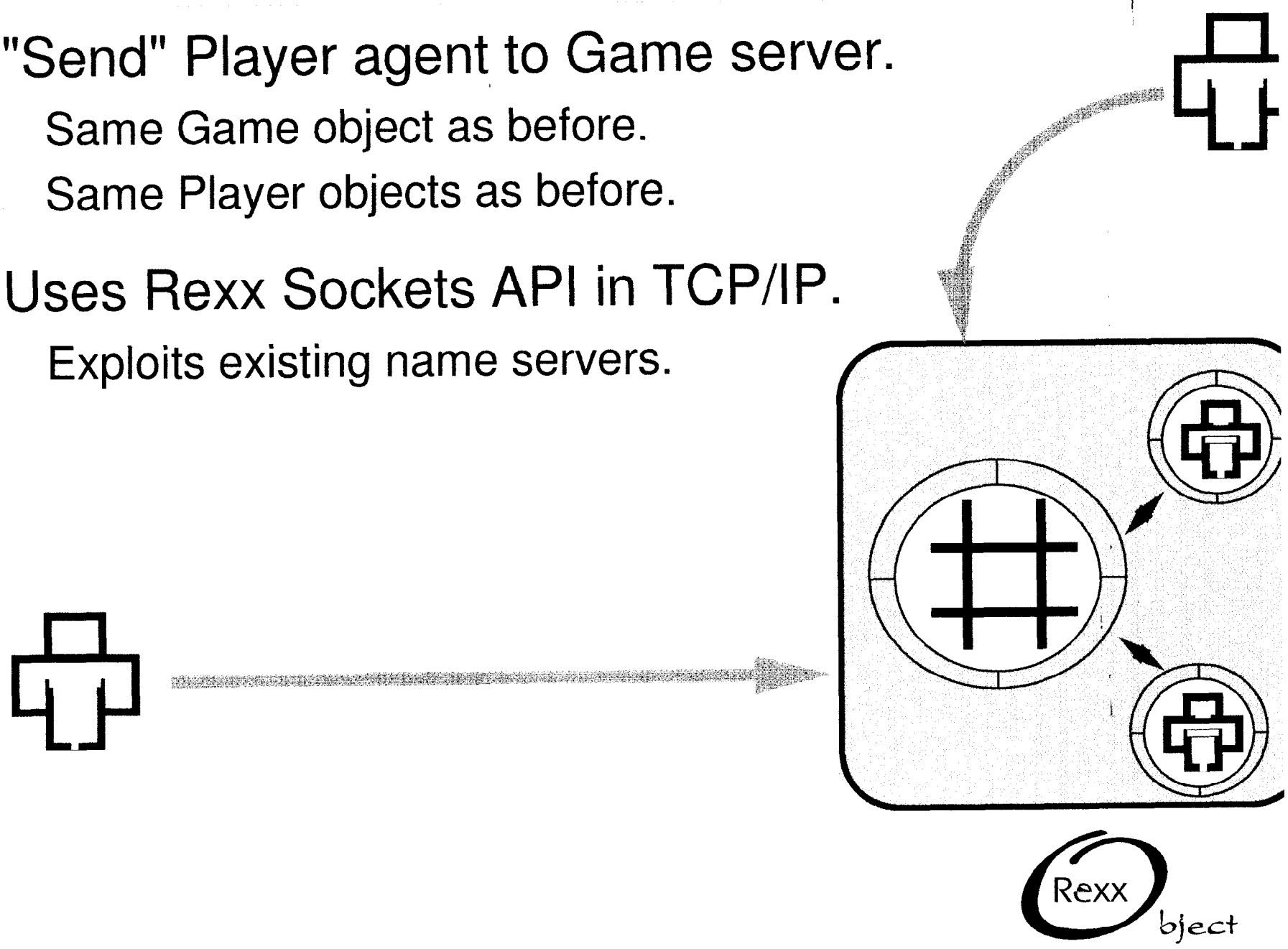
A TicTacToe Agency

"Send" Player agent to Game server.

- Same Game object as before.
- Same Player objects as before.

Uses Rexx Sockets API in TCP/IP.

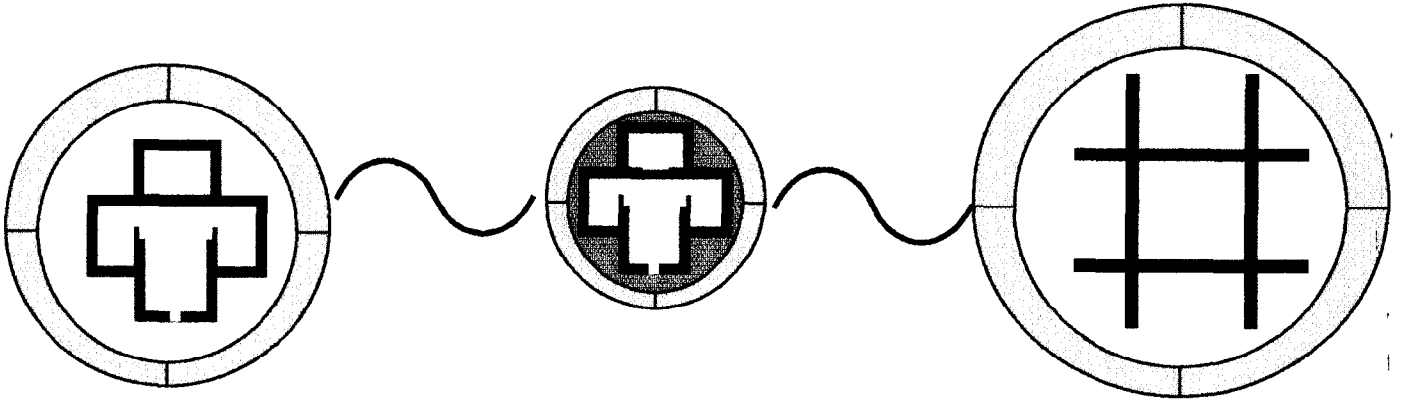
- Exploits existing name servers.



Messaging with Proxies

"Proxy Objects"

- capture messages intended for a target object
- relay message to and response from target
- transparent to sending and receiving objects
- useful for debugging and message tracing and...

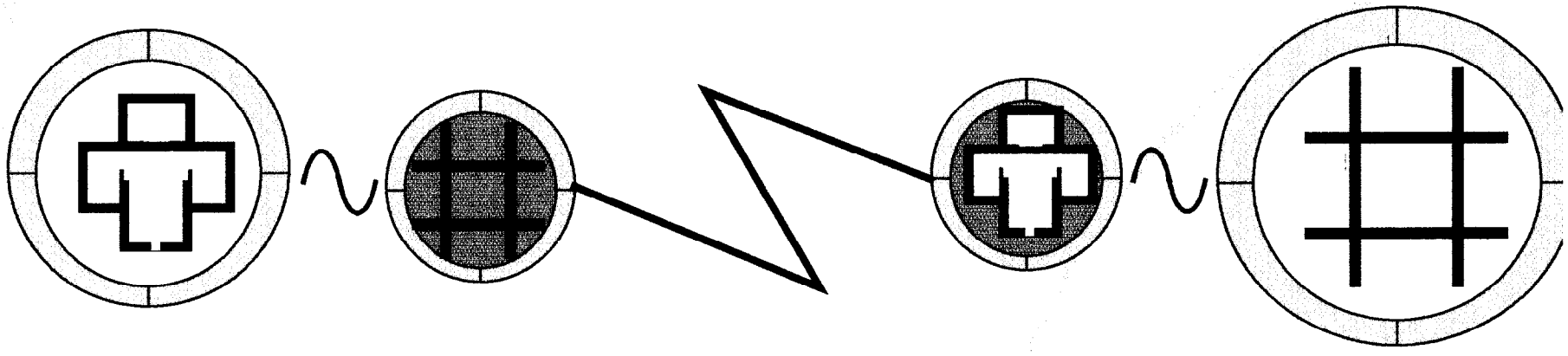


Rexx bject

Communications Proxies

When proxies relay messages over a network connection, the objects appear to be local to each other -- the network is completely hidden.

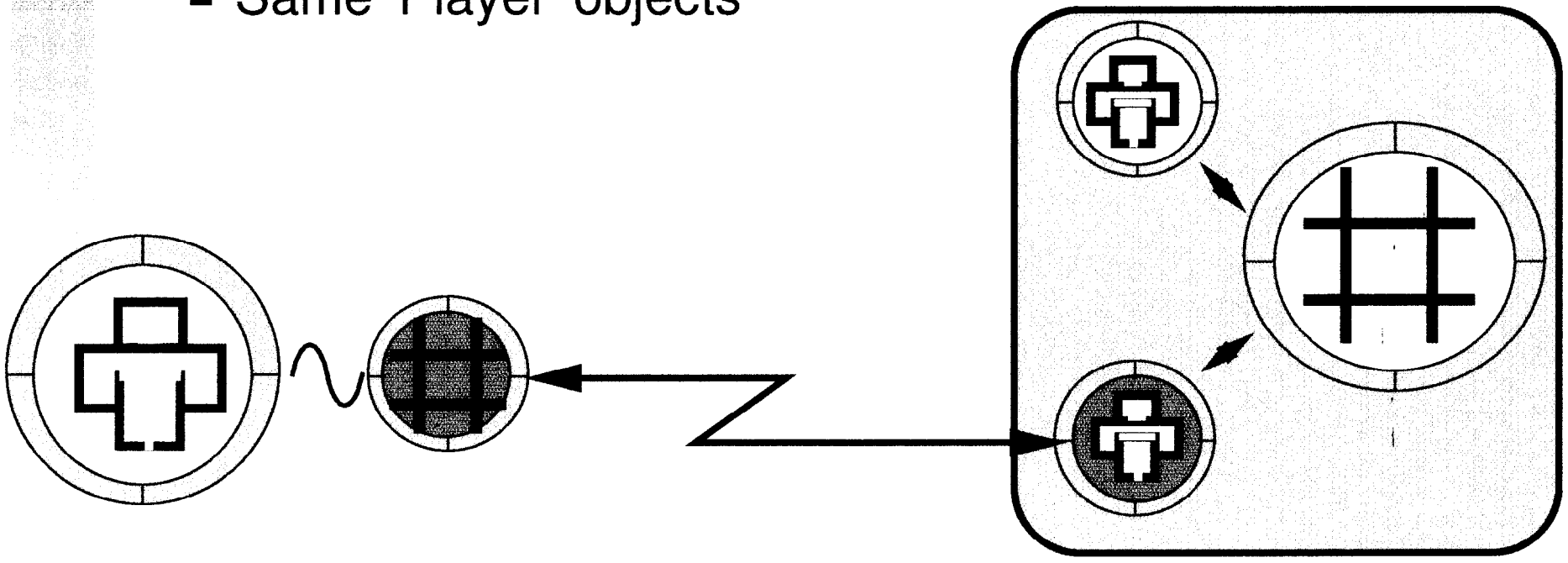
So, communications proxies can network-enable objects that 'know' nothing about networks.



Remote Messaging via Proxies

'Send' a communications proxy for a Player, and objects on two systems interact around the task of playing the game.

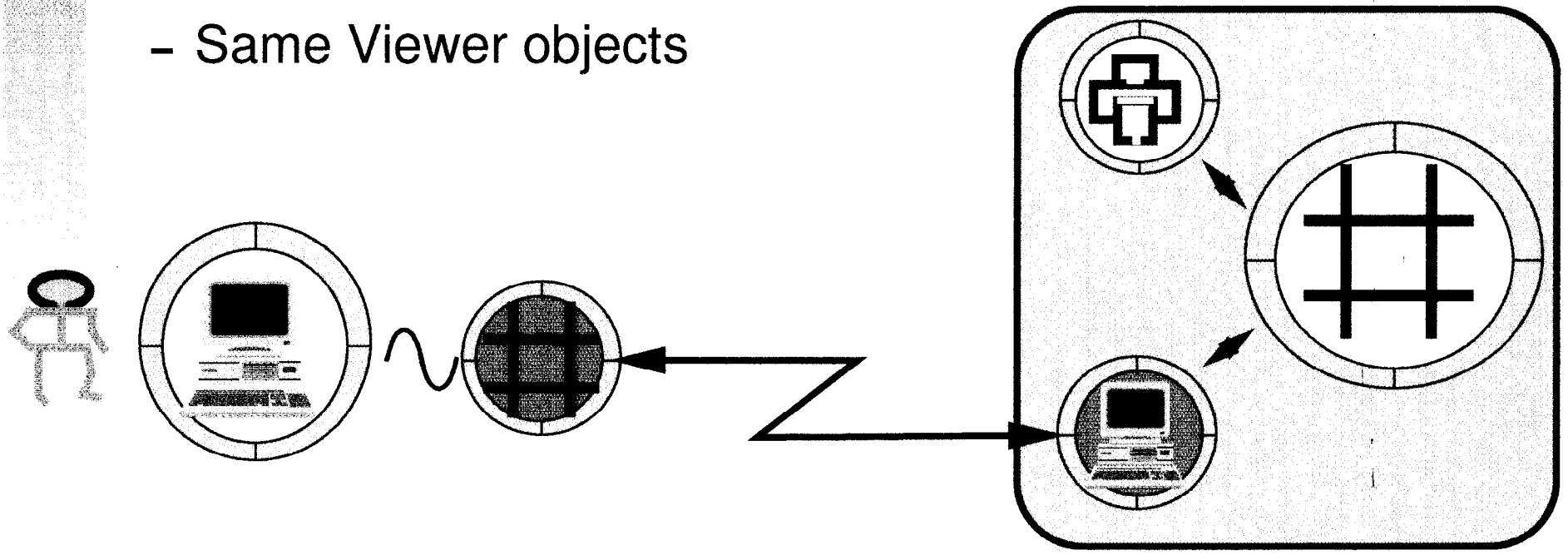
- Same 'Game' object
- Same 'Player' objects



Remote Interaction via Proxies

Send a communications proxy for a 'Viewer' object, and users and objects on three systems interact

- Same Game objects
- Same Player objects
- Same Viewer objects



Rexx bject

What You've Seen

- Multitasking, multi-user TCP/IP servers
- Scripting within, and across systems
- Agent-based and Client/Server computing
- Agents collaborating around a task
- TCP/IP-enabled code without TCP/IP coding
- ...and about 600 lines of Object REXX

Rexx bject

Object Rexx: OpenDoc Support

Tom Brawn
IBM Endicott

Pages 138-142

Object Rexx OpenDoc Support

Thomas Brawn
IBM Corporation
tombrawn@vnet.ibm.com

OpenDoc

- Compound Document Architecture
- Key technologies
  - Parts and Part handlers
  - Bento
  - SOM
  - OSA
- Multi-platform
  - Apple
  - IBM
    - OS/2
    - AIX
    - PPC
  - Wordperfect
    - Windows

OpenDoc
Open Scripting Architecture

- OSA Framework classes
  - Component class
  - Scripting Component class
  - Terminology class
- OSA Event Manager
- Component Manager

Object Rexx
OSA Language Support

- ScriptableApp class (scriptable application proxy)
  - Connect method
- Object Specifiers
  - Identify target within application
- Send OSA Events to scriptable applications

Object Rexx
ScriptableApp Example

```
:
Lotus123 = .ScriptableApp~connect('lotus123')
FirstCell = table[1] cell[1]
Lotus123 FirstCell~setdata(100)
:
```

Object Rexx
OSA Scripting Component

- SOM Class
  - Subclass of OSA Scripting Component class
  - Accessed through Component Manager
- Required functions
  - Execute
  - Load, Store
  - Display, Dispose, GetScriptInfo, SetScriptInfo, ScriptError

Object Rexx Scripting Component...

- Compile functions
  - Compile
  - CopyID, ScriptingComponentName
- GetSource
  - GetSource
- Recording
  - StartRecording, StopRecording

Object Rexx OpenDoc Support - Summary

- Language Support
  - ScriptableApp class, Object Specifiers
  - OSA Events
- Scripting Component functions
  - Required
  - Compile
  - GetSource
  - Recording

Report from the X3J18 Committee

Brian Marks
Formcroft Ltd.

Pages 144-149

History & Status

Enthusiasm at first Rexx Symposium.

First committee meeting 1991.

Fifteenth meeting 1995.

First public review completes May 3rd 1995.

Expected approval as ANSI standard, 1995.

Already an influence on implementations.

Committee continues for maintenance and further versions.

Proposal Document

167 pages including the informal parts.

Has been circulated.

Available commercially.

On the World Wide Web. http://rexx.hursley.ibm.com/rexx/

Available at this symposium.

Method

Backus-Naur Form describes syntax.

Prose describes facilities provided by the "configuration".

Complicated Rexx described in terms of simpler Rexx.

Some "pseudo-Rexx" to glue parts of the definition together.

Content

"The scope of the standard will be the second edition of the Cowlishaw book, plus consideration of implementation experience. The scope may be altered as necessary to promote portability, reliability, maintainability and efficient execution of REXX programs on a variety of computing systems. Both compiling and interpreting REXX programs will be considered."

Not "Design a Rexx for the nineties".
Although Date conversion, error subcodes, command I/O ....

Not "The union of every existing implementation".
Although alternatives for negation and blank characters.

Not "Better wrong than changed"
For example D2C(0), DATATYPE('','B'), 1.0000000003 non-integer.

You can help:

Correct the document.

Talk to your supplier of Rexx about the Standard.

Join the committee for the next version of the standard.

CenterPiece and Object Oriented REXX

Sandy Syx
Mantissa Corporation

Pages 150-173

Mantissa Corporation

CenterPiece and Object Oriented REXX

Sandy Syx - ssyx@mantissa.com
205-945-8930

1

Introduction

- *Mantissa Corporation*
- *Data Center Automation software products since 1981.*

- *RMS "The Report Management System"*
- *OPS "Operations Productivity System"*
- *FYI "Windows/LAN-based Document/Image Management and more"*

2

Agenda

- *CenterPiece Architectural Overview*
- *CenterPiece Built-in Classes*
- *CenterPiece Object-Oriented REXX*
- *REXX Improvements for Complex Problems*
- *Developing CenterPiece Classes*

3

CenterPiece Architectural Overview

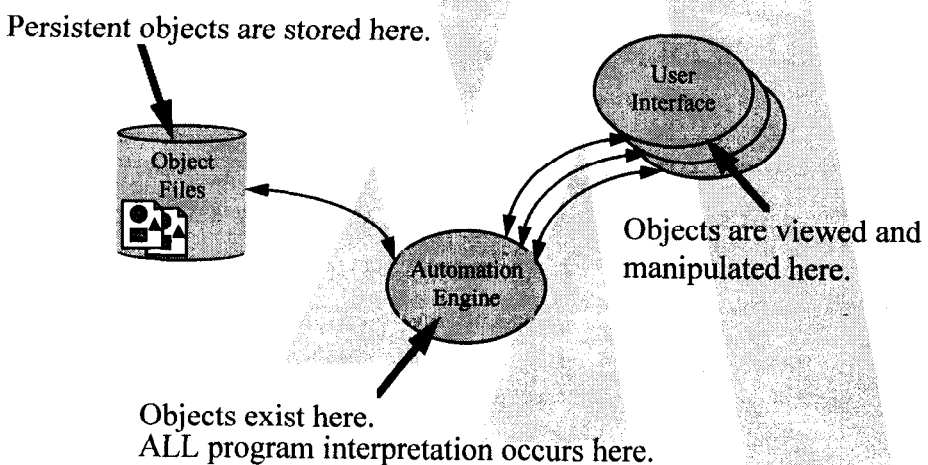
4

What is CenterPiece?

- ***CenterPiece is a Distributed, Multi-platform, Object-Oriented, Interpretive, Development and Runtime Environment.***
- ***Two main Executables:***
  - The Engine - a multi-threaded interpreter that serves objects to multiple simultaneous clients in psuedo realtime.
  - The User Interface - A graphical application that allows one to view and manipulate objects that exist in an "engine".

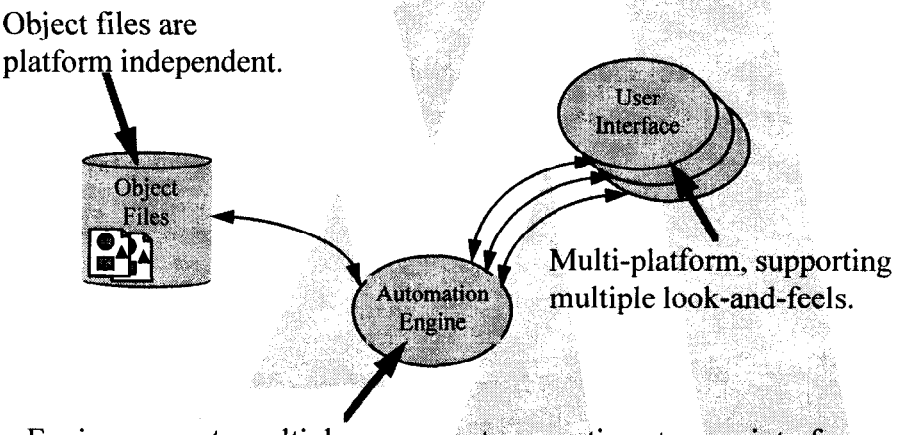
5

CenterPiece Architecture



6

CenterPiece is Multi-platform



7

The Engine Is...

- ***The heart of the system.***
- ***An object server.***
- ***A multi-threaded object oriented REXX executor.***
- ***Basically event driven.***
- ***Responsible for reading and writing object files.***
- ***Not visual.***

8

The User Interface

- *Graphical User Interface*
- *Runs on multiple platforms and window systems (X-Motif, OS/2 Presentation Manager, MsWindows)*
- *Supports multiple look-and-feels (Motif, CUA, Windows)*
- *Very interactive allowing direct manipulation (object menus and drag-and-drop).*

9

CenterPiece Built-in Classes

10

Fundamental Built-in Classes

- *Workspace*
  $2^{1/2}$ Dimensional Visual Container of WorkspaceObjects.
- *WorkspaceObject*
  Gives objects the ability to be on a workspace. (Name,X,Y,Layer, Icon,Workspace, etc...)
- *Class*
  Allows one to create new classes.

11

Programmer's Helper Classes

- *Program*
  Allow interpretation and execution of REXX logic.
- *Thread*
  Instance of executing program.
- *List - Ordered collection of items.*
- *Dictionary - Unordered collection of key/data pairs.*
- *Semaphore - Resource Arbitor*
- *Queue - Object version of REXX queues*

12

Simple Visual Objects

- *Text*
  Floating text. (font, color, angle)
- *Line*
  Line segments. (X2,Y2,width,color)
- *Rectangle*
  Hollow or filled rectangles (width, height,fillcolor)
- *Image*
  2D color images. Can be large and deep.

13

Dialog Objects

- *Button - Action button that runs a "Click" method when pressed.*
- *Checkbox - State selector runs a "Click" method when pressed.*
- *TextEntry - Text entry field, allows multi-line, scrollbars, etc.. Runs a "Changed" method when the text is entered.*

14

More Dialog Objects

- *ListBox - Combination of a List and a ListView. Visual list, allows images and text. Items can be dragged from the list.*
- *Slider - Allows a value to be selected within some range. Runs a "Slide" method when the slider is slid.*
- *RadioGroup - Mutually exclusive group. Runs a "Click" method when the selection changes.*
- *Spinner - Allows spinning or typing in a number from a specified range of values.*

15

Communication Classes

- *MTAServer*
  Message Transport Agent - allows one to create a "server" that will listen for connections from "clients" at any number of access points (tranport,port). Allows telneting into the server if tcp is used.
- *MTAClient*
  Allows one to connect to a server to exchange messages.

16

Object Storage to Disk

- ***ObjectFile***
  Saves all owned objects to a disk file.

17

Application Delivery

- ***UserProfile - This class allows one to secure access to a CenterPiece engine by defining exactly who can connect, and how they connect. Users can be "Developers", "EndUsers" or both. An "EndUser" has a "Connect" method that can be overridden to show the appropriate application dialogs for the user on connection to an engine.***

18

CenterPiece Object-Oriented Extensions to REXX

19

Objects

- *Objects are instances of some Class.*
- *Objects have any number of attributes.*
- *Objects are globally visible.*
- *<u>Every object has a universally unique immutable identifier.</u>*
- *Any object can be made persistent.*

20

Object Ownership

- *Objects can own any number of other objects.*
- *An object can have at most one owner.*
- *When an object is destroyed, all of its owned objects are also destroyed.*
- *When an object is saved, all of its owned objects are saved.*

21

Attributes

- *Attributes act much like REXX variables.*
- *They can be simple or compound.*
- *Object attributes must be defined in some superior class.*
- *Attributes names are case and space preserving, but case and space insensitive.*

22

Referencing Objects

- *Objects have global visibility.*
- *Each object is unique not because of its name, class, nor attribute values, but because of its universally unique immutable identifier (UUID). These are normally just called object identifiers or object-ids.*
- *Objects are referenced by REXX variables that have an object-id as their value.*

23

Attribute Access

Object Attributes are selected with a double-dot (..).

The symbol to the left of the double-dot is translated into a value. The translated value must be an object-id or a class name.

The symbol to the right (up to the next double-dot) is treated exactly like a variable symbol and must reference an object or class member.

24

Attribute Access Examples

Simple Attribute Access

Assume b is an object of the Button class.

```
b..Name = "Press Me"
b..BackgroundColor = "maroon"
```

Multiple Indirections

Assume that b is a Button, and assume that the button has an attribute "Workspace" that references an object of the Workspace class that the button is on. The name of the workspace could be accessed by:

```
b..Workspace..Name
```

the button

the button's workspace

25

Object Creation/Destruction

Accomplished with two new REXX built-in functions:

object_id = ObjectCreate(<classname>)

rc = ObjectDestroy(<object_id>)

For Example,

```
aLine = ObjectCreate( "Line" )
aLine..x = 100
aLine..y = 100
aLine..x2 = 200
aLine..y2 = 200
...
call ObjectDestroy aLine
```

26

Classes

- *Classes define attributes that each instance of the class will have.*
- *CenterPiece allows multiple inheritance.*
- *Classes are objects and are instances of the "Class" class.*
- *Classes are typically used by their name.*

27

Inheritance Model

- *Attributes are inherited dynamically.*
- *A class can be modified "on the fly" with existing instances.*
- *Attribute lookup precedence:*
  1. Local Object Override
  2. Object's Class
  3. Primary Superclass--->Root Class
  4.Secondary Superclass--->Root Class
  In other words: "A Depth first, breadth next search up the class hierarchy".

28

Dropping Attributes

The REXX - DROP instruction can be used to cause an attribute to revert to its class default.

For example, assume that a class "Author" exists which has an attribute named "Name" that has a class default value of "anonymous".

```
anAuthor = ObjectCreate( "Author" )
anAuthor..Name = "Fred Brooks"
say  anAuthor..Name  ==> would print "Fred Brooks"
drop anAuthor..Name
say  anAuthor..Name  ==> would print "anonymous"
```

29

Object Related Built-in Functions

- ***ObjectCreate***
- ***ObjectDestroy***
- ***ObjectClone***
- ***ObjectFindOfClass***

- ***IsObject***
- ***IsObjectOfClass***
- ***ClassOfObject***
- ***ClassIsSubclassOf***
- ***ClassIsDirectSubclassOf***

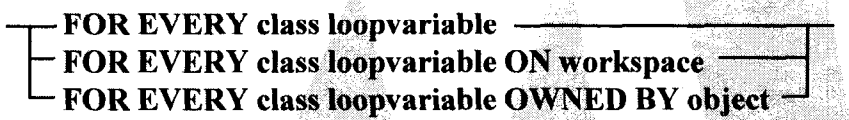
- ***ObjectOpen***
- ***ObjectOpenAsDialog***
- ***ObjectClose***
- ***ObjectGoto***
- ***ObjectGetOwner***
- ***ObjectSetOwner***

- ***ObjectFileOpen***
- ***ObjectFileSave***
- ***ObjectFileClose***

30

Iterating Over Objects

repetitor (extensions):



31

Object Iteration Example

Iterating Over All Objects of a Given Class

```
num_employees = 0

DO FOR EVERY Employee e
   SAY e..Name
   num_employees = num_employees + 1
END
```

32

Object Member Iteration

repetitor (extensions):

33

Composite Objects

- *"An object by itself is intensely uninteresting". - Grady Booch*
- *Object Identifiers behave much like pointers to structures in 'C' or 'C++'.*
- *Any object attribute can contain an object identifier of another object.*
- *Composite objects can be made in which one object references and owns any number of other objects.*

34

Embedded Objects

- *It is possible to embed objects within other objects.*
- *This must be done by adding a class member that references an object of a specified class.*
- *The embedded object will be cloned for each instance of the class.*
- *The embedded object may not be destroyed independently of its owner.*

35

Methods

- *Methods are simply objects of the Program class that are referenced by some attribute of an object.*
- *Method invocation is no different than calling any other REXX function or subroutine. The method is addressed just like any other object attribute, except that it is used where a function or subroutine name would normally be used.*

36

Self Reference In Methods

The double-dot with no prefix is an object self reference inside an object method.

For example, imagine a user interface Button method that runs when the button is clicked.

```
/* begin Button..Click */
   ..Name = "Hello"
return 0
```

In this example the double-dot with no prefix means "this" button.

37

REXX Extensions for Complex Problems

38

Multi-Threaded REXX

- *An additional built-in function, Start(), is provided to allow one thread to start another.*
- *Each thread executes concurrently.*
- *Threads are re-dispatched, basically, after each source instruction.*

39

Unwinding the stack on a Raised Condition

Normal REXX, strangely, doesn't unwind the call stack when a condition (exception) is raised.

We extended the CALL ON and SIGNAL ON instruction to allow them to be prefixed with the keyword UNWIND.

For example,

```
UNWIND CALL ON syntax NAME mysyntaxtrap
...
mysyntaxtrap:
  say "Tarfu"
return
```

40

Developing CenterPiece Classes

41

Modularity

- *Instances do not have to be saved in the same ObjectFile as their classes.*
- *Classes do not have to be saved in the same ObjectFile as their superclasses.*

42

Constructors/Destructors

- *Any class can have a "Create" method. Simply add an attribute named Create and make it a sub-object that is of the Program class.*
- *The method will automatically be run when an instance of the class is created.*
- *Ditto for "Destroy" and "Load" which will be run when the object is destroyed or loaded from an object file, respectively.*

43

User Events

- *Many classes have methods that are run in response to user actions.*
- *These methods are optional, and if not provided, a default built-in action occurs in response to the user event.*
- *Some examples are:*
  WorkspaceObject..Drop or DroppedUpon
  Button..Click
  TextEntry..Changed

44

User Events - Continued

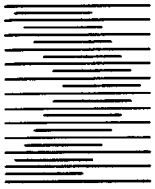
- *The first argument to a user event method is always a Dictionary object that contains entries that indicate what happened.*
- *The attributes present in the context dictionary depend on the event. For example, a Drop event would have the new X and Y locations of the object dropped.*

45

REXX, Distributed Systems and Objects

John Tibbetts
Kinexis

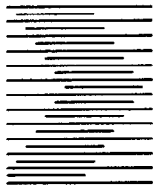
Pages 174-193



Rexx, Distributed Systems and Objects

John Tibbetts
Kinexis
(415) 558-9277
email: john@kinexis.com

Rexx Symposium
May 2, 1995



Rexx, Distributed Systems and Objects

- **Rexx + Client/Server Database**
  - Simple architecture for simple C/S apps
- **ORexx + SOM**
  - Beginning of strong client/server platform
- **Current technology (ORexx)**
  - Functions as SOM requester
  - Adequate for client-side activity
- **Coming technology**
  - Exporting OREXX classes as SOM classes
  - Scripting language for OpenDoc
  - Suitable as server platform

Our approach...

- **Discuss paradigm issues**
  - Evolution of distributed architectures in Four Phases
- **Discuss transaction issues**
  - Agenda of TP
- **Examine Rexx C/S implementation strategies**

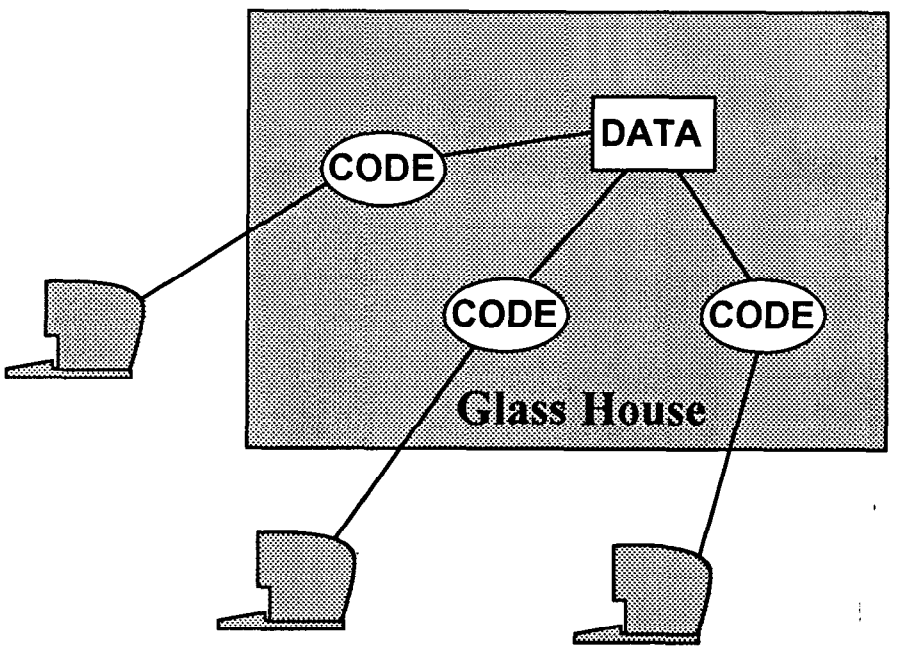
Computing Architecture Phases

1. Centralized
2. Clients to Database Server
3. Clients to Function Server
4. Objects

***Code* and *Data* in varying combinations**

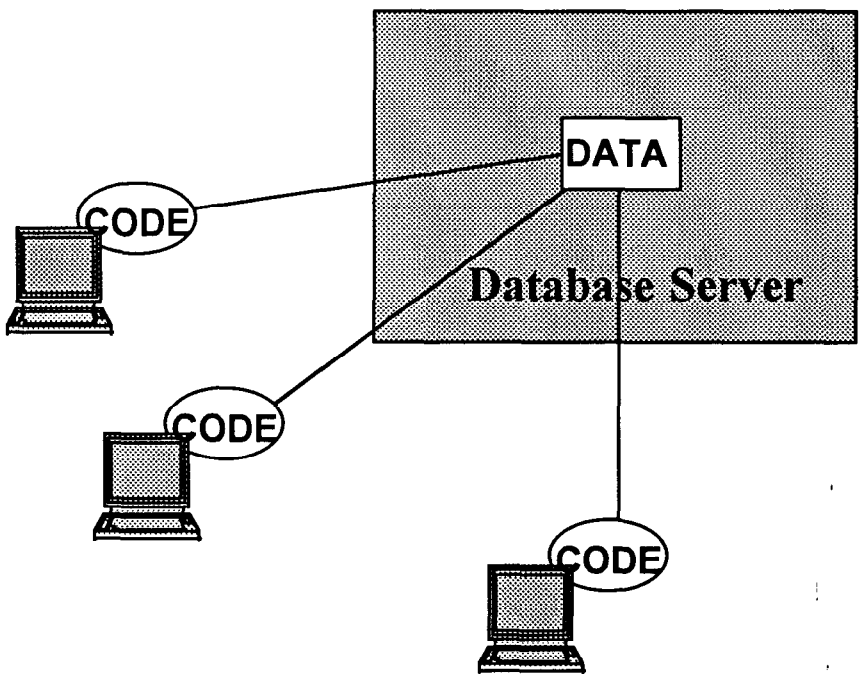
Phase 1. Centralized Computing

- Strong control & manageability
- Good security
- Weak user empowerment
- Weak on distributed computing
- Limits business "reach"



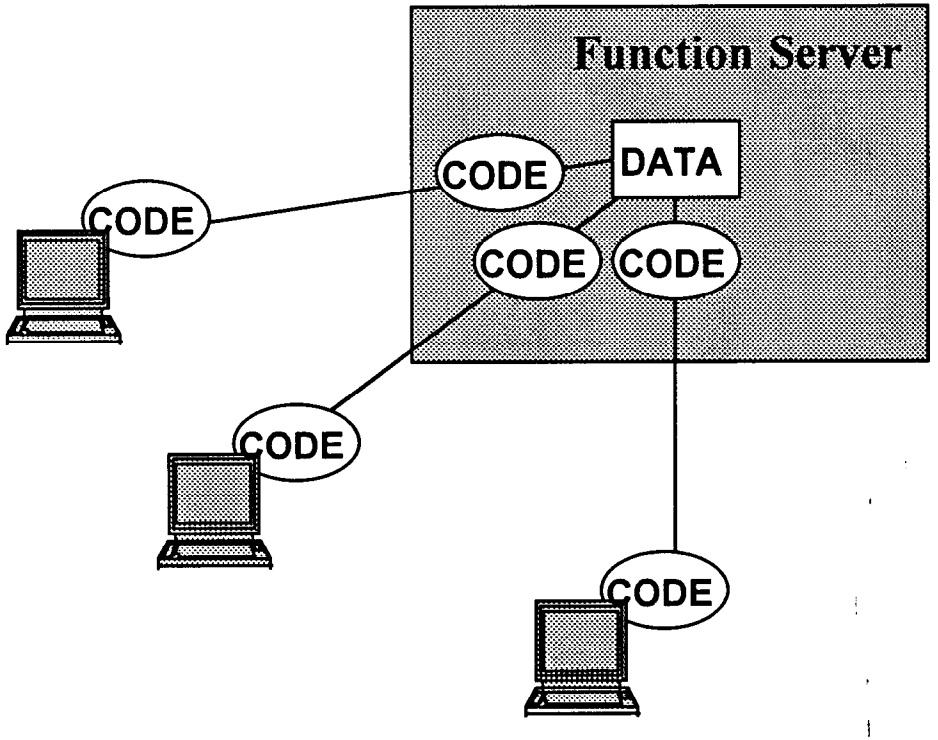
Phase 2. Clients to Database Server

- Power to the user
- Power to the user interface
- Uneven performance and integrity
- Weak 3-tier architecture
- Trust problems



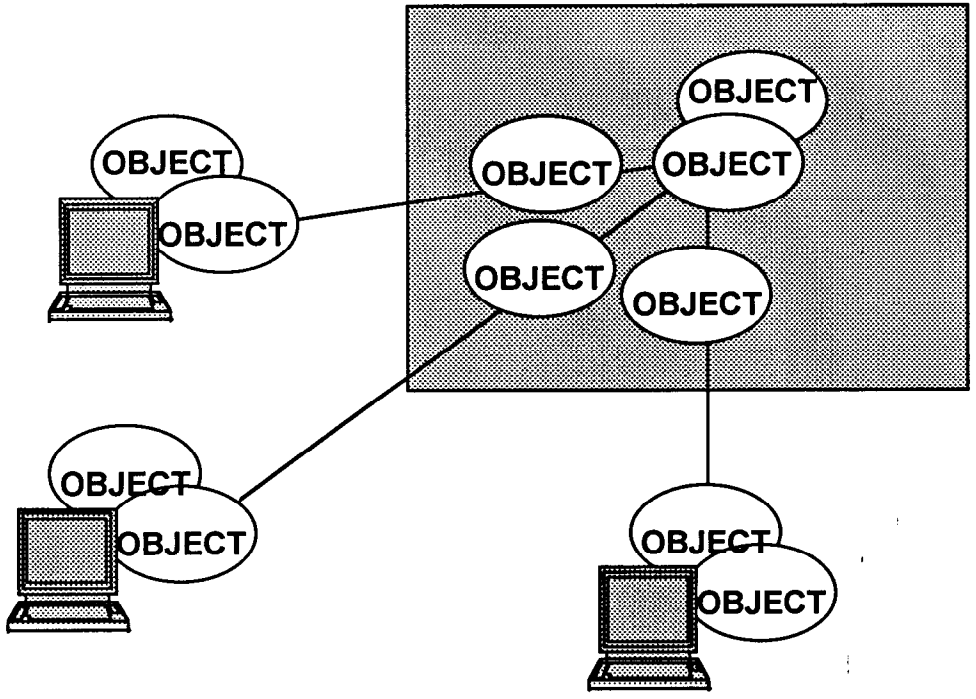
Phase 3. Clients to Function Server

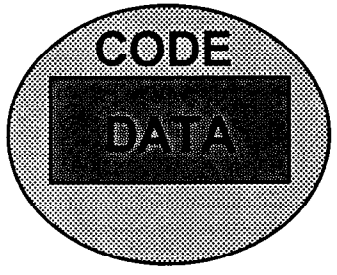
- Improved performance and integrity
- Stronger 3-tier architecture
- Trust tuning
- *But* significant software complexity



Phase 4. And Then There Are Objects...

- Inately partitioned
- Semantic continuity
- Limited transactional awareness





An Object is *Data*
surrounded by a protective layer of *Code*

Transaction = "The Deal"

- **In clay**
  - Baked invoices at Ebla (3rd millenium BC)
- **On paper**
  - Sales orders and invoices
  - Double-entry ledgers
  - Contracts and deeds
- **Online**
  - Reservations for travel, hotels, cars, etc
  - Banking & stock trading documents
  - Order entry, inventory planning, accounting
  - Telephone call setup and billing, email

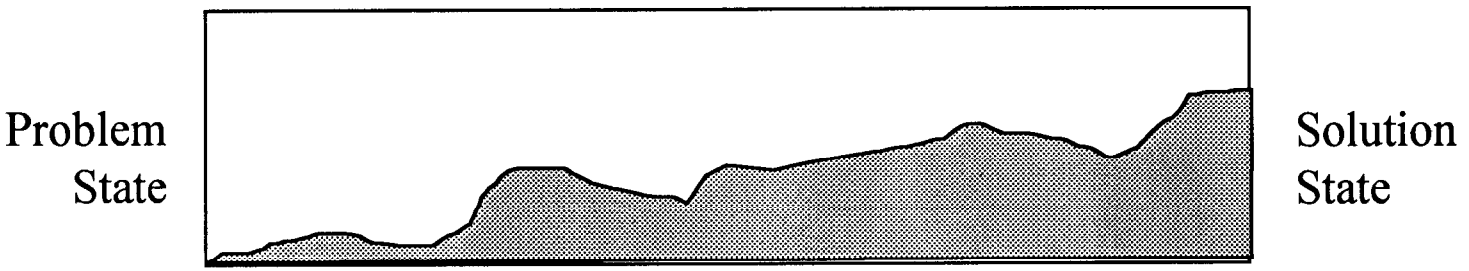
ACID Test for Transactions (And All Deals)

- **Atomicity**
  - Transactions are "all or nothing" (integrity principle)
  - Wedding vows (two-phase commit)
- **Consistency**
  - Transactions are a correct transformation of state
  - Debits = credits
- **Isolation**
  - Concurrent transactions behave as if executed serially
  - Transactions don't see other transactions partial results
- **Durability**
  - Once committed, transactions are not forgotten
  - Bound to honor COMMITments

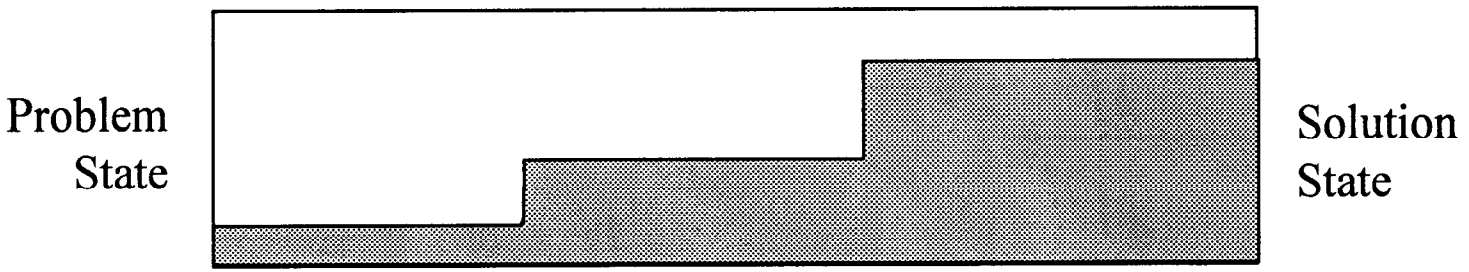
Transactions are the computer equivalent of contract law

The Transactional Discipline

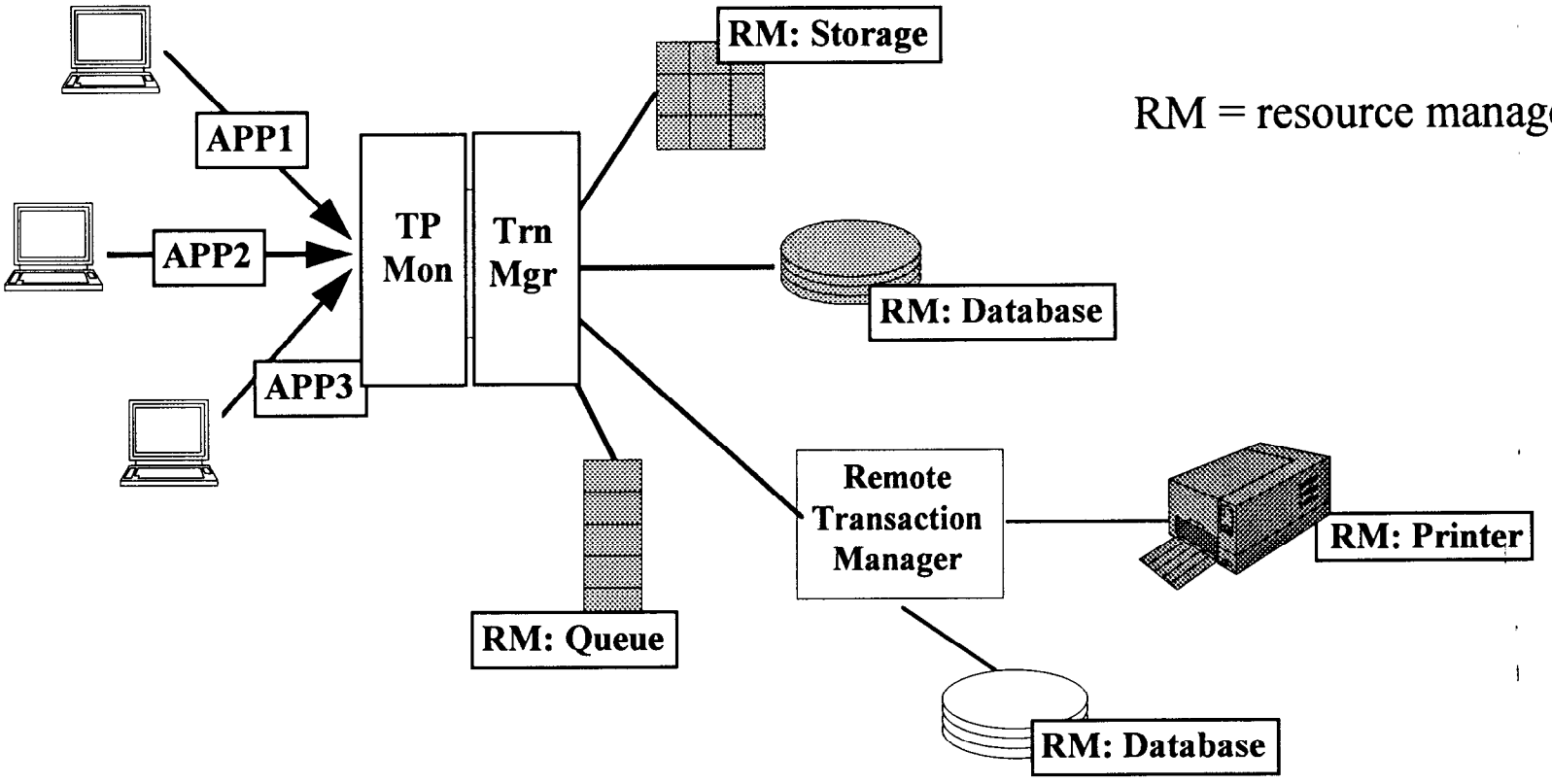
Non-transactional: state changes continuously



Transactional: orderly, coordinated, audited state change

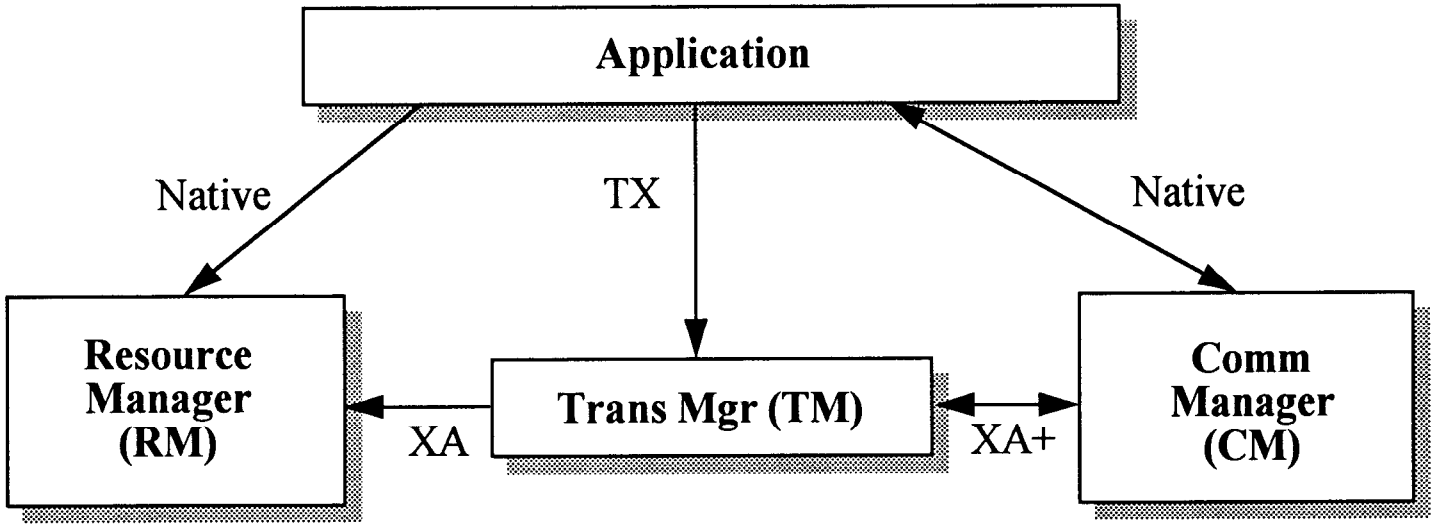


How TP Monitors are organized



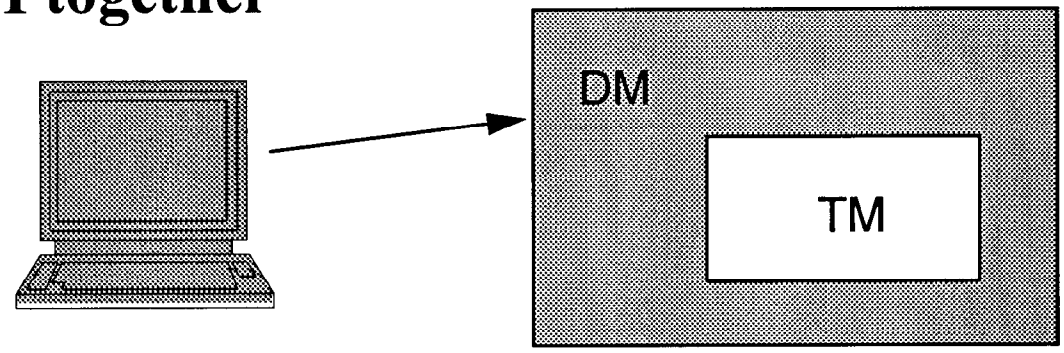
RM = resource manager

Full-Fledged TP: X/Open DTP Model



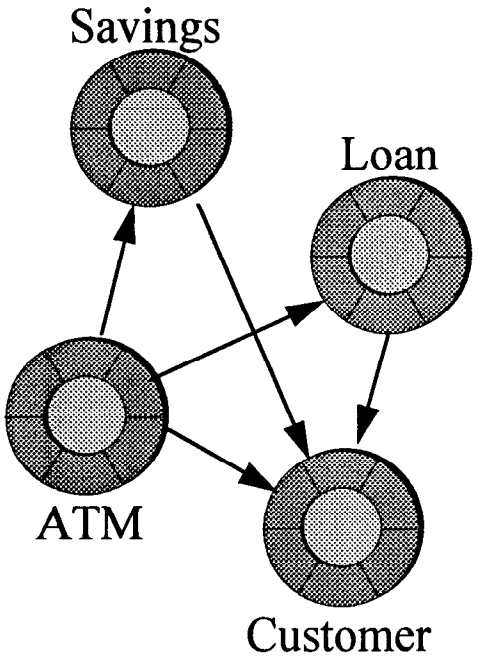
TP-Lite: Transactions Inside Database

- **Today's client/server databases bundle TM and DM together**



- **TM should be *unbundled* for open systems**
  - Coordinate multi-vendor DBMS
  - Coordinate user-written function
  - Coordinate other resources

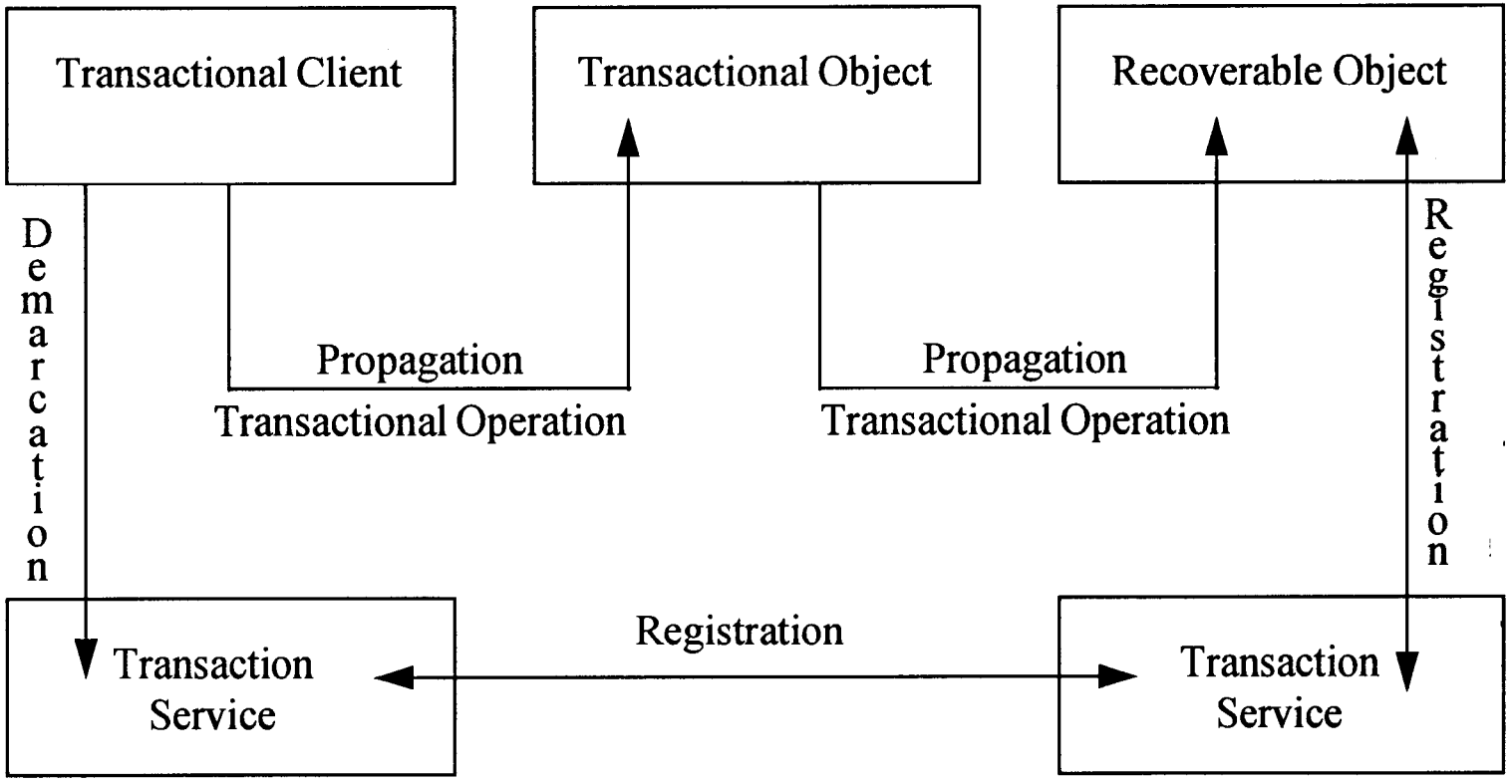
Imagine Transactional Objects



- Objects distributed about network
- Send messages to
  Debit savings acct object
  Credit load account
- Commit changes all object states
- Simultaneous to multiple consumers

Objects are microscopic Resource Managers:
Subsystem driven by a formal API that has state.

OMG Transactional Object



Mapping Paradigm & Transactionality

	Phase 1	Phase 2	Phase 3	Phase 4
Non-TP	Monolithic program	Any client/server DBMS	RPC Msg Queue Sockets	CORBA, DSOM, COM, DOE
TP	Monolithic program under TP: CICS, IMS, Guardian, ACMS	Any client/server DBMS with RUOW or DUOW	Dist TP: TRPC, TMQ, LU6.2	CORBA (w/OTS), DSOM, (COM)

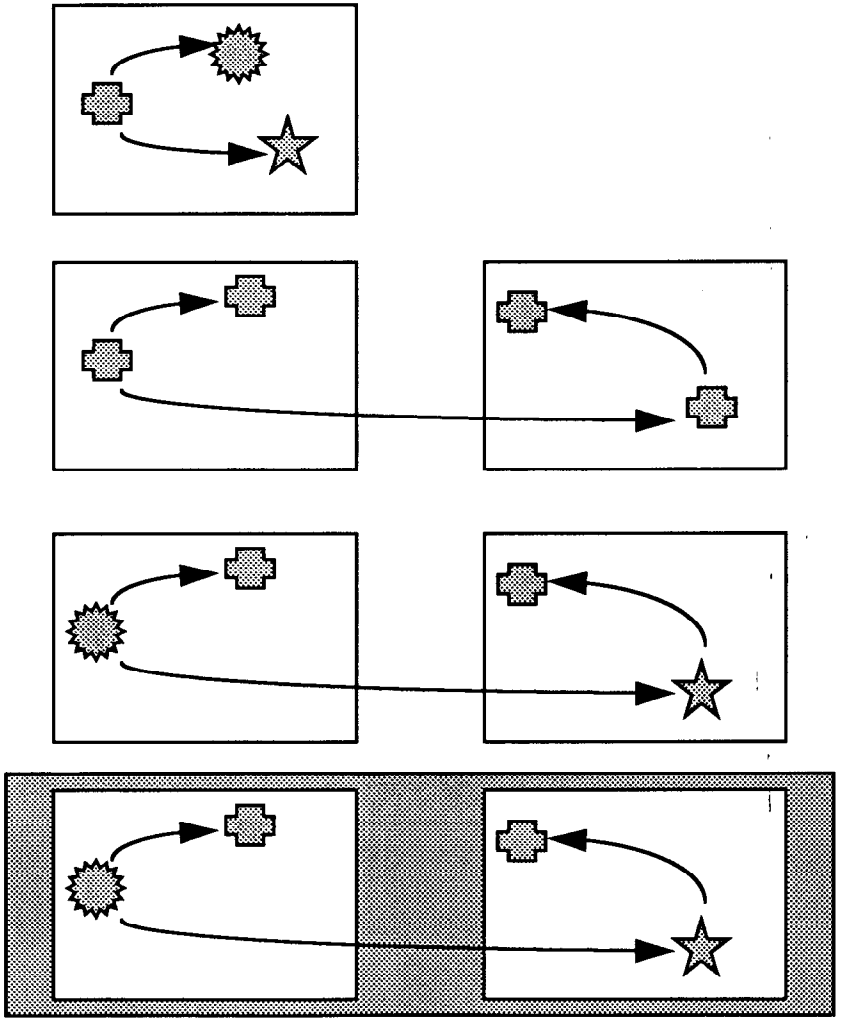
Steps to Distributed, then Transactional, Objects

1. ***Compatibility*** **among differing object models in same machine**
   - CORBA (coarse-grain)
   - SOM (fine-grain)

2. **Distributed homogeneous objects**
   - CORBA
   - DSOM

3. **Distributed heterogeneous objects**
   - CORBA 2.0
   - DSOM

4. **Distributed transactional objects**
   - CORBA w/OTS



Rexx implementation strategies

- **Rexx or ORexx Client to Client/Server database**
  - Phase 2 or Phase 4/2 hybrid
- **Rexx or ORexx Client to Function server**
  - Phase 3 or Phase 4/3 hybrid (non-transactional)
- **Rexx or ORexx Client to TP Monitor (eg. CICS ECI)**
  - Phase 3 or Phase 4/3 hybird (transactional)
- **ORexx Client to DSOM**
  - Phase 4 (non-transactional)
- **ORexx modifying Server behavior**
  - Phase 4 (non-transactional)
- **ORexx Client or Server with ORB transaction services**
  - Phase 4 (transactional)

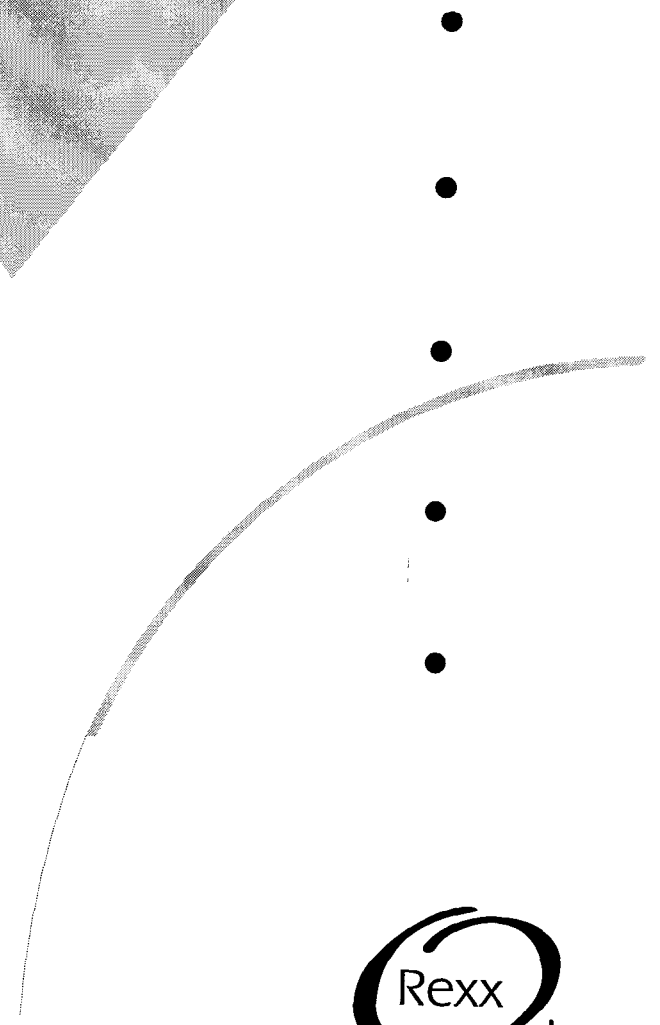
Getting Ready for Object REXX

Rick McGuire
IBM Endicott

Pages 194-218

Getting Ready for Object REXX

Rick McGuire
Object REXX Development
IBM Endicott



Answers to Questions without Answers

- A major goal of Object REXX is removing limitations of the existing REXX language.
- Many of the limitations are seen in some of the most frequently asked (and frequently unanswered) questions on bulletin boards.

Rexx bject

Let's Practice

- Question: How do I convert dates from on e REXX format to another?
- Current Answer: Well, you don't....
- Object REXX Answer: Just specify the input date as the second argument to the Date() function. A third option argument tells Date() what input format you are using:
  - Date('b', '28 Feb 1995')
  - Date('n', '02/28/1995', 'U')

Rexx bject

Passing Stems

- Question: How do I pass a stem to a function or subroutine
- Answer: Just specify the stem in the argument list and access the argument with the USE ARG instruction.

```
call StemSort stem., count
   .
   .
   .
StemSort: procedure
  use arg x., count
   .
   .
   .
  return
```

Returning Multiple Values

- Question: How do I return more than just a single string value from a function?
- Answer: Just return a stem or other "composite" object

```
lines. = ReadFile(filename)
   .
   .
   .
ReadFile:  procedure
parse arg filename
count = 0
do while lines(filename) <> 0
   count = count + 1
   x.count = linein(filename)
end
x.0 = count
return x.
```

Expressions in Compound Tails

- Question: How do I specify that A.i = A.i+1?
- Answer: Specify the variable part of the tail within square brackets ("[]")

```
lines. = ReadFile(filename)
   .
   .
   .
ReadFile:  procedure
parse arg filename
x.0 = 0
do while lines(filename) <> 0
  x.0 = x.0 + 1
  x.[x.0] = linein(filename)
end
return x.
```

Traversing Stems

- Question: How do I traverse all of the tails currently assigned to a stem?
- Answer: Use the DO OVER instruction

```
Do tail over stem.
  say stem.tail
end
```

Packaging Multiple Functions

- Question: Now do I distribute a "bunch" of external functions without creating a file for each function?
- Answer: Package the routines in a "Requires" file

```
::requires sitefunc.cmd


::routine function1 public
   .
   .
   .
::routine function2 public
   .
   .
   .
::routine function3 public
```

Bonus Function

- Requires files can also perform needed global setup

```
/* load required functions */
call rxfuncadd 'a', 'b', 'c'
  .
  .
  .
::routine function1 public
  .
  .
  .
::routine function2 public
```

Sharing Variables Between Programs

- Question: How can I share "global variables" between multiple programs?
- Answer: Access the variables as a REXX "environment" variable

```
.environment~setentry(,
  'MY.PROGRAM',,
  .directory~new

.my.program~name = "xyz"
```


Rexx bject

The "Procedure Expose" Dilemma

- Question: How can I share variables between related subroutines without doing a PROCEDURE EXPOSE for every variable through all of the caller's levels?
- Answer: Structure the related routines as an object and share the variables with the EXPOSE instruction

```
::class data_manager
::method x
expose name time type
  .
  .
  .
::method y
expose time type attributes
  .
  .
  .
::method z
expose attributes
  .
  .
```

Computed CALL instructions

- Question: How do I make a call to a routine whose name is contained in a variable?
- Answer: Use an indirect CALL instruction, placing the routine variable name in parentheses

```
parse arg name, argument
call (name) argument
```

Replacing Common Idioms

- Some common REXX idioms can be made easier using features of Object REXX or by replacing stems with other REXX objects.

Stems vs. Arrays

- A REXX array may be the more appropriate choice
  - Variable size
  - Automatically tracks the size
  - DO OVER traverses in order

```
lines = ReadFile(filename)
   .
   .
   .
ReadFile:  procedure
parse arg filename
output = .queue~new
do while lines(filename) <> 0
   output~add(linein(filename))
end
return output~makearray
```

Stems vs. Directories

- Compound variables can be "vulnerable" to other variable usage in a program

```
employee.name
```

Can fail if name is used as a variable, but

```
employee = .directory~new
employee~name = "Rick"
```

is always safe!

Stems vs. Directories

- Using compound variables as both "collections" and "structures" simultaneously can be awkward

```
employees.i.name = "Rick"
employees.i.salary = "???"
```

vs.

```
employees[i] = nextWorker()
```


Rexx bject

Consider Building You Own Objects

- While many problems can be adequately solved by stems, arrays, directory, etc., consider building your own objects:
  - Hide the processing logic
  - Can be placed in a REQUIRES file for better reuse.

A Common Problem

- Customer wants to process a group of records contained in a flat file, with the data fields organized in columns.
  - Records must be easily accessed, updated, and written out to a new file in the same format.
  - Record formats are subject to change, so updates must be easily performed.
  - Multiple programs will be written to perform updates against the same files.

A Solution

```
::class employee
::method init
expose name id address salary manager
parse arg name 25 id 32 address 100 salary ,
   106 manager 131

::method name attribute
::method id attribute
::method address attribute
::method salary attribute
::method manager attribute

::method string
return left(name, 25) || left(id, 7) || left(address, 68) || ,
   right(salary, 6) || left(manager, 25)
```

A Solution (continued)

```
/* Give everybody a raise! */
parse arg oldFile newFile

do while lines(oldFile) <> 0
  employee = .employee~new(linein(oldFile))
  employee~salary = employee~salary + ,
     employee~salary * .10
  call lineout newFile, employee
end

::requires employ    /* include the employee records */
```

Building New Idioms

- Over the years, many common REXX idioms have been developed
- These idioms are still valid, but...
  - New Object REXX idioms may replace some existing ones
  - New Object REXX programming idioms will be added to existing ones

Rexx bject

For Your Consideration...

- A new Object REXX programming idiom, the "caching directory"
  - Keep a cache of items read from a disk file
  - Caching is done on first reference to an item
  - Subsequent requests pull the item from the cache

The Caching Directory

```
/* Create an employee file caching directory */
cache = .directory~new /* get a directory */
                       /* add an unknown handler
cache~setmethod('UNKNOWN', .methods['UNKNOWN'])
return cache           /* set up is done! */

::method unknown
expose dataFile
parse arg employeeId
if \var(dataFile) then dataFile = .stream~new('emp.rec')
record = dataFile~linein(EmployeeId%100)
record = .employee~new(record)
self[employeeId] = record
return record

::requires employ
```

SOM – Present and Future

Simon Nash
IBM Austin

Pages 220-235

SOM - Present and Future

Simon C. Nash

IBM Corporation
Austin, TX

nash@austin.ibm.com

What is SOM?

- ★ System Object Model
- ★ Part of the OS
- ★ Language-neutral
- ★ Language bindings (toolkit)
- ★ Compiler support (DTS)
- ★ Distributed objects (DSOM, CORBA)

Why SOM?

- ★ OO language interoperability
- ★ Binary format for objects
- ★ Release-to-release binary compatibility
- ★ Procedural language access
- ★ Support for distribution

Platforms

Available:

OS/2, AIX, Windows, Macintosh

Announced:

MVS, AS/400

Other ports in progress

Languages

Available:

C, C++, Smalltalk

Beta:

Object REXX

Announced:

OO COBOL

SOM Releases

1992: SOM 1.0

(C, OS/2 WPS)

1993: SOMobjects 2.0

(C++, IDL, CORBA, DSOM)

1994: SOMobjects 2.1

(Warp, DTS C++)

1995: ???

SOM Components

- kernel
- toolkit:
  - SOM compiler, language bindings
- class libraries:
  - collections
- frameworks:
  - persistence, replication, events, metaclass, IR, emitter
- distribution
  - DSOM (workstation and workgroup)

A SOM Example: stack.idl

```
#include <somobj.idl>

interface Stack: SOMObject
{
  void push (in SOMObject element);

  SOMObject pop ();

  long size ();

  implementation
  {
    SOMObject contents[100];
    long top;
    somDefaultInit: override;
  };
};
```

A SOM Example: stack.c

```
#include "stack.ih"

SOM_Scope void  SOMLINK push(Stack *somSelf,
            Environment *ev, SOMObject* element)
{
    StackData *somThis = StackGetData(somSelf);
    StackMethodDebug("Stack","push");

    _contents[_top++] = element;
}

SOM_Scope SOMObject*  SOMLINK pop(Stack *somSelf,
            Environment *ev)
{
    StackData *somThis = StackGetData(somSelf);
    StackMethodDebug("Stack","pop");

    return _contents[--_top];
}
```

A SOM Example: stack.c

```
SOM_Scope long  SOMLINK size(Stack *somSelf,
            Environment *ev)
{
    iiackData *somThis = StackGetData(somSelf);
    StackMethodDebug("Stack","size");

    return _top;
}

SOM_Scope void SOMLINK somDefaultInit(Stack *somSelf,
            somInitCtrl* ctrl)
{
    StackData *somThis; /* set in BeginInitializer */
    somInitCtrl globalCtrl;
    somBooleanVector myMask;
    StackMethodDebug("Stack","somDefaultInit");
    Stack_BeginInitializer_somDefaultInit;
    Stack_Init_SOMObject_somDefaultInit(somSelf, ctrl);

    _top = 0;
}
```

A SOM Example: test.c

```
#include <stdio.h>
#include <som.h>
#include "stack.h"

void main (void)
{
  Stack *stack1, *stack2;
  Environment *ev;
  SOMObject *obj1;

  stack1 = StackNew();
  stack2 = StackNew();
  ev = somGetGlobalEnvironment();
  _push(stack1,ev,stack2);
  printf("stack1 size is %li\n",_size(stack1,ev));
  obj1 = _pop(stack1,ev);
  printf("stack2 size is %li\n",_size(obj1,ev));
  _somFree(stack1);
  _somFree(stack2);
}
```

A SOM Example: test.exe

```
sc -sc stack.idl

/* code the method implementations */

sc -sh;ih stack.idl

icc test.c stack.c som.lib

test

/* output is:
     stack1 size is 1
     stack2 size is 0
*/
```

A SOM Example: stack.dll

```
sc -sc stack.idl

/* code the method implementations */

sc -sh;ih;def stack.idl

icc /Ge- stack.c som.lib stack.def

implib stack.lib stack.def

icc test.c stack.lib som.lib

test

/* output is:
     stack1 size is 1
     stack2 size is 0
*/
```

SOM Features

- ★ static compile/link time binding
  - — for high performance
- ★ name lookup, programmable dispatch
  - — for flexibility, dynamic languages
- ★ class and metaclass objects
- ★ multiple inheritance
- ★ transparent proxies

SOM Challenges

- ★ scaleability: many fine-grained objects
- ★ performance: approaching native C++
- ★ shared objects: multi-process
- ★ run-time footprint: class metadata
- ★ local/remote transparency
- ★ OMG services, CORBA 2
- ★ dynamic language support

Rexinda

Stephen Rondeau
AugmenTek

Pages 236-251

Rexinda™

Stephen Rondeau
AugmenTek

3606 S. 180th St. C-22
SeaTac, WA 98188
Phone: 206-246-6077
augmentek@acm.org

Agenda

- Linda®
- Rexinda
- Applications
- Future

If enough time...

- Availability
- Parallelization

Linda: Background

-- David Gelernter, early 1980s dissertation

-- Parallel programming model

> coordinate execution and data sharing to solve common problem
> simple to use
> shared memory model
> "tuple space" (global data area) managed by a server

-- C and FORTRAN implementations

-- Several companies

> Scientific Computing Associates, Inc.
> Torque Systems, Inc.
> Others

Linda: Terminology

-- Tuple: like a database record

(field1, field2, field3, ...)

> Examples: ("ball", "color", "red")
("list", {2,4,6,8,10})

-- Tuple Space: unordered collection of tuples, possibly distributed over many processors

> Example:

("ball", "color", "red") ("box", "size", 10)
("list", {2,4,6,8,10}) ("ball", "color", "green")
(27) (4989, 67, 828763) ("box", "size", 10)

-- Matching: uses number of fields, data types, field order, and user values

Linda: Functions

-- Six functions: out(), rd(), in(), eval(), rdp(), inp()

> Examples use Rexinda's syntax

-- out(): put tuple into tuple space

```
Call out "ball", "color", "red"
   > ("ball", "color", "red")

Do i=1 to 5
  number.i = i*2
End
number.0 = 5

Call out "list", "@S number."
   > ("list", {2,4,6,8,10})
```

Linda: Functions (continued)

-- rd(), in(), rdp(), inp(): get values from tuple space by matching

rd() -- wait for match, copy values
in() -- wait, copy values, remove tuple
rdp() -- match not found, return 0; else rd()
inp() -- match not found, return 0; else in()

> Examples given later

-- eval(): start a new process

> Example:
Call eval "sort_result", "C:\SORTRXI"
> ("sort_result", 0) after completion

Linda: Matching Examples

Given the tuple space (TS):
("ball", "color", "red") ("box", "size", 10)
("list", {2,4,6,8,10}) ("ball", "color", "green")
(27) (4989, 67, 828763) ("box", "size", 10)

In order of execution:

Call rd "ball", "color", "? color"
> color="green" -- or "red", TS unchanged

Call in ".C", "size", 10
> One of ("box", "size", 10) is removed

If rdp("box", "size", "?N size") = 0
then Call out "box", "size", 20
else Say "size="size /* 10 in this case */

Call rd "ball", "size", "?N size"
> waits for matching tuple to appear

Rexinda: Rationale

-- Popularize parallel programming

> models world
> requires modularity
> allows recoverability
> offers scalability

-- Follows REXX fairly well

> functions are easy to remember
> associativity similar to stems
> tuple space is global, like default scope

-- Leverage REXX's fast prototyping

-- Extend REXX to handle user-defined events (data appearing in tuple space)

Rexinda: Goals

-- Goals

> C Linda-like (conversion, reference)
> Avoid preprocessing source code
> Extendable
> Automatic datatyping with overrides
> Progressive disclosure philosophy
> Handle errors

-- Basic method: prefix string argument with "markers"

Rexinda: Syntax Markers

-- Needed on out() only to force data type

> "@d" string
where data type d is:
-- "C" or missing for character
-- "N" for a valid REXX number
-- "S" for a stem

-- Input functions really need it:

>"?d varname"
get value of type d and put it in varname

> ".d"
ignore field that has data type d

> "@d" string
force data type d for this string

Applications: Simple Email

Email:

My program:

```
Parse arg name message
Call out "mailto", name,,
         "from", "stephen",,
         message
```

Everyone is running this program:

```
Parse arg my_name .
Do Forever
  Call in "mailto", my_name, ,
          "from", "? sender",,
          "? message"
  Say "Mail from" sender":" message
End
```

Applications: Simple Print Spooler

-- Print spooler client:

```
Parse arg file_name
If rdp("spooler") = 0
 then Call eval "C:\SPOOLER"

Call in "id", "?N id"
Call out "id", id+1

Call out "print", id, file_name
Call in "done", id
Say "Job" id "has printed."
```

-- Print spooler ("C:\SPOOLER"):

```
Call out "spooler"
id = 1
Call out "id", id

Do while rdp("spooler", "quit") = 0
   Select
   When inp("print", id, "? file_name") then do
       Address CMD "@COPY" file_name "/B LPT1"
       Call out "done", id
        id = id+1
     end
    Otherwise Call Delay 1 /* Every inactive second */
    End
End
```

Future: Enhancements

-- Matching:

> Aggregates: match and return values for more than one tuple per call (1.0)
> Counting: count number of matches (1.0)
> Inequalities: allow matches based on <, >, \=, <=, >= a value
> Patterns: allow matches within a field to cause match of tuple

-- Persistence

-- Security

-- Recoverability

-- Transparent data/object access

Availability

-- Rexinda Base (version 0.1): Now
  > Source code
  > No user support
  > Inconvenient and slow
  > Cannot distribute server source code
  > US$20. plus $3. S&H, US Destinations
    (WA residents add 8.2% tax)
    -- Price subject to change without notice

-- Rexinda 1.0: if sufficient interest
  > Function library (DLL) and fast server
  > No source code, compatible with Base
  > Some enhancements (TBD)

-- Rexinda n.0, n > 1: success of v1.0

-- Rexinda Net 1.0: if sufficient interest
  > Network (TCP/IP) version
  > Some enhancements for networking

Parallelization

-- Carriero and Gelernter:
How to Write Parallel Programs, MIT Press, 1991, ISBN 0-262-03171-X

-- Three approaches:

> Result -- the shape of the problem

Example: SQRT(elements of matrix A)

> Specialist -- the makeup of the workforce

Example: send requests to servers

> Agenda -- the tasks to do

Example: many capable workers, list of tasks

REXX for CICS/ESA

Bob Vogel
IBM Dallas

Pages 252-272

REXX Symposium
REXX for CICS

Bob Vogel

May 3, 1995

Contents

Introduction

IBM

- Copyright

(C) Copyright IBM Corporation 1993, 1995

- Trademarks

The following terms used in this paper are trademarks or service marks of IBM Corporation in the United States or other countries:

CICS/ESA, IBM, MVS/ESA, OfficeVision, OS/2

What is "REXX for CICS/ESA"

IBM

- Two products (GA 7/29/94)
  - REXX Development System for CICS/ESA (5655-086)
  - REXX Runtime Facility for CICS/ESA (5655-087)
- REXX language support for CICS/ESA
- EXEC CICS Command support from REXX
- CEDA and CEMT REXX interfaces
- REXX-DB2 Interface
- Native CICS application environment
  - REXX Panel Facility
  - High-level file system & filelist utility
  - Text Editor
  - Interactive shell
  - Open Application Integration facilities
- High-level Client/Server support
- And More

The REXX Language

IBM

- Created by Mike Cowlishaw, at IBM Hursley

- In ANSI X3J18 committee since 1991, target for standard is 1995

- Strengths of REXX
  - Natural / high-level
  - Avoids unnecessary detail
  - Typeless
  - Strong parsing
  - Command and function support
  - Source level interactive tracing
  - Complete set of modern programming constructs
  - Fairly small language, easy to learn
  - Rich set of functions
  - Can be interpreted or compiled

Trends toward REXX popularity

IBM

- Widespread use of REXX under OS/2
  - Now in PC DOS 7.0
- REXX moving aggressively to new platforms
- Shift to very high level languages / devp systems
- Macro support taking off industry wide
- ANSI REXX effort progressing well
- REXX compilers
- Dramatic increases in computing power (improves REXX performance)
- Shift to new system architectures, where REXX is a natural
  - Client/Server computing
  - Workstation GUI to Enterprise data/appls (Visual REXX)
  - Object Oriented (OO REXX)
  - Messaging and Queueing (Workflow Scripts)

Shift to very high level languages

IBM

- Highly competitive times demand higher productivity
- Large numbers of non-DP pros coming on board
- Alignment of programming with business organization
- More complex systems difficult to develop & maintain
- Prototyping Development Methodology has come of age
- Building block approach and code reuse popular
- REXX and BASIC beefed up for serious programming

What were design goals for REXX/CICS

- Deliver a strong productivity tool
- Create a serious REXX-based application environment
- Make REXX work with CICS languages and facilities
- Provide a native prototyping, development and customization environment
- Common REXX support across CICS platforms
- Provide high-level Client/Server interfaces
- Utilize the power of REXX in an open application integration platform

Project history

IBM

- REXX prototype to IBM Program Product
  - From Assembler to PL/X for portability
  - FROM TSO/E REXX base to direct use of REXX kernel
  - From 1 person research project to formal development team

Why Now

- Growing exposure to REXX and its power
- Growing emphasis on productivity
- Product requirements for REXX under CICS
- Opportunity to improve a very important environment
- Enhance customers' large mainframe investment
  - REXX for CICS actually introduces some of the concepts of personal computing into the MVS/CICS environment.

Highlights

- Full REXX 3.48 language support under CICS
- Dynamic EXEC CICS command level support
- REXX interface to CEDA, CEMT
- DB2 Interface (SQL statements & DB2 commands)
- CICS native text editor for REXX execs and data
- High-level VSAM-based REXX file system (RFS)
- Execs may also be run from MVS Partitioned Datasets
- High-level Panel I/O facility
  - Also supports BMS

Function Overview (continued)

IBM

- Support for REXX Subcommands (themselves written in REXX)
- Pseudo-conversational support (conventional and auto)
- System and user profile exec support
- Shared execs in storage (via EXECLOAD & EXECDROP)
- High-level Client/Server interfaces
- Online help and softcopy REXX/CICS manuals
- Improved run-away REXX task management
- Concurrent international language support (English + 6)
  - German, Spanish, French, Canadian French
  - Japanese Kanji, Simplified Chinese

Need for REXX/CICS

- As a tool to streamline support staff activities
  - CICS Systems Programmers and Administrators
  - DB2 Analysts
  - CICS and DB2 testers, other support staff
- More productive CICS application development
  - Native CICS development (simpler)
  - Enjoy the strengths of REXX under CICS
- More flexible, powerful product customization & extension (macros)
- Quick prototyping and procedural language functions
- Preserve REXX investments in migrations
- Needed for products with REXX requirements
- As a script language to automate/streamline development sequences
- Help enable enterprise-wide Client/Server computing
- Better enable CICS end-user computing
- CICS Application Integration
  - Glue language to tie the pieces together
  - Building block support

REXX File System (RFS) Features

- Hierarchical Directory structure (like OS/2, AIX)
- VSAM based
- No need to register most new users
- No need to register individual EXECs
- Import/Export from/to MVS Partitioned Datasets
- Management functions for members (COPY, DELETE, RENAME)
- FLST file directory interface utility
- An EXECIO-like I/O utility (RFS)
- VSAM datasets can be added to a Filepool dynamically
- Number of filepools only limited by DASD

Editor features

- Two personalities
  - XEDIT
  - ISPF
- RFS and PDS file support
- Terminal models 2, 3, 4 & 5 supported
- Customizable
- REXX macro support
- Execs can be run without leaving editor

Security features

- CICS security facilities (via ESM) to control access
- REXX/CICS Authorized Command support
- REXX/CICS Authorized Library support
- REXX/CICS Authorized User support
- Security exits
- RFS AUTH command for directory sharing

Performance

IBM

- REXX/CICS interpreter uses sophisticated performance techniques
- Majority of execution time usually not in language processing
- Shared and Reentrant code / execs
- Performance numbers, courtesy of Steve Ware, University of Florida on WWW (see last page for Web address)
- REXX/CICS run-time support for compiled REXX/CICS execs a possibility

EXEC CICS commands not supported

IBM

- HANDLE ABEND
- HANDLE AID
- HANDLE CONDITION
- IGNORE CONDITION
- PUSH HANDLE
- POP HANDLE

REXX/CICS Summary

- REXX Development System for CICS/ESA much more than another language
- REXX/CICS introduces significant new capability
- REXX/CICS provides new approaches to CICS computing
- REXX/CICS opens CICS to a broader range of uses
- REXX/CICS is a strong productivity tool for devp and support
- REXX/CICS is a good application integration platform
- REXX/CICS is useful for serious programming
- REXX/CICS is natural for Client/Server computing
- REXX/CICS is in step with industry trends (application server)
- CICS and REXX are very synergistic
  - REXX = ease of use, high productivity, native devp env.
  - CICS = production computing and common support

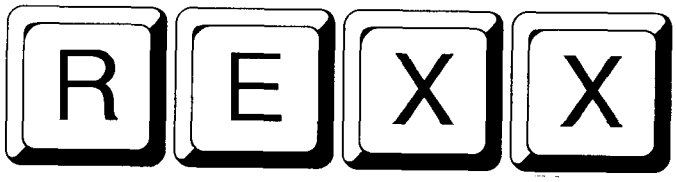
Questions and Wrapup

- Future direction
  - Runtime Lite
  - Compiler Support
  - TCP/IP Sockets
- How to get more information on REXX or REXX/CICS
  - http://rexx.hursley.ibm.com/rexx/
  - http://sfware.nerdc.ufl.edu/rexxcics/rxkixhom.html
  - dshriver@vnet.ibm.com
- Questions

REXX Changes in OS/2 Warp

Dick Goran
CFS Nevada, Inc.

Pages 274-282



Changes in OS/2 Warp

REXX Symposium
May 1-3, 1995
Palo Alto, California

Dick Goran
C F S Nevada, Inc.
953 E. Sahara Avenue, Suite 9B
Las Vegas, Nevada 89104-3012
Voice: 702-732-9616
FAX: 702-732-3847
Email: 71154.2002@CompuServe.com

Warp and REXX

Warp, or OS/2 Warp Version 3.0 as it is officially named, has added 5 new functions to the REXXUTIL application programming interface, or API as IBM likes to call it. Figures 1 through 5 contain a description of these new functions as they appear in the *REXX Reference Summary Handbook*. Unfortunately, IBM did not do a very good job in documenting these new functions.

While copying, moving, or creating shadows of workplace shell objects (WPS) is somewhat intuitive, saving and opening WPS objects with the new functions is not.

SysCopyObject(), **SysMoveObject()**, and **SysCreateShadow()** are fairly straightforward in their purpose. These functions permit a simple means of copying, moving, or creating a shadow of WPS objects from within a REXX program. However, there are some points of special interest when using these three functions.

When an object is copied, no object ID is provided for in the copy. Whether the original object has an object ID or not, the copy will not have an object ID. The only currently available mechanism for assigning an object ID to the copy is with a third-party utility such as ***DeskMan/2***. When a shadow is created, the shadow ID of the newly created object will have the same value as the object ID of the original object.

The purpose of the **SysSaveObject()** function is to force OS/2 to flush the file system objects properties (stored as extended attributes) and the Workplace Shell abstract objects properties (stored in the OS2.INI and OS2SYS.INI files) to disk.

The **SysOpenObject()** function is just as obscurely documented in the online REXX.INF file that comes with Warp. As with many of the other WPS functions contained in REXXUTIL, the IBM-supplied documentation refers to WPS and program manger C^{++} language functions that the average OS/2 user would not have access to without owning the toolkits made for OS/2. The information shown in Figure 1 was compiled from a combination of "bit-digging" research along with some assistance from IBM's Glendale Laboratories - the group responsible for REXX development. The numeric values shown in Figure 1, and used to tell the **SysOpen()** function which view is to be opened, may not be complete. It will take some trial-and-error testing along with independent research to determine what other values may be used. I suggest that users who want to keep up with the latest information, as it becomes available, stay current with the material in the various REXX related fora on CompuServe (OS2DF1, Section 6), IBM's IBMLink and TALKLink (OS2REXX CFORUM), *comp.lang.rexx*on the Internet along with your favorite local BBS.

Development Technologies and Greg Czaja have released version 1.51 of DeskMan/2 with updated REXX functionality as well as interfacing with the WPS functions for

Warp. C F S Nevada, Inc. has released the third edition of the *REXX Reference Summary Handbook* (ISBN 0-9639854-2-6 / IBM SRL & PUBORDER S246-0078-01) with the Warp additions.

One of the other major changes in Warp that is directly related to REXX is the ability to both create and change printer objects (WPPrinter class) and the new LaunchPad (WPLaunchPad) with the REXXUTIL functions. Figure 11 is an example of a REXX program used to replace an existing LaunchPad with one configured within the program.

```
SysSaveObject( object_name, timing_flag )
```

Returns 1 if the WPS object *object_name* was successfully saved; otherwise, returns 0. File system objects (WPFileSystem) are saved in the file system's extended attributes and abstract objects are saved in the OS2.INI (user) file. Transient objects (WPTransient) cannot be saved.

Object_name can be a WPS object ID (the unique string preceded with a '<' and terminated with a '>') assigned to the object when it was created (e.g. <WP_DESKTOP>) or a fully qualified file name.

Timing_flag can be **0** (Boolean false - object is to be saved synchronously) or **1** (Boolean true - object is to saved asynchronously).

Figure 1 - SysSaveObject() function

```
SysCreateShadow( object_name, →
                          → object_destination )
```

Returns 1 if a shadow of *object_name* was successfully created at the specified location, *object_destination*; otherwise, returns 0.

Both *object_name* and *object_destination* can be a WPS object ID (the unique string preceded with a '<' and terminated with a '>') assigned to the object when it was created (e.g. <WP_DESKTOP>) or a fully qualified file name.

Figure 2 - SysCreateShadow() function

```
SysOpenObject( object_name, view, flag )
```

Returns 1 if the WPS object *object_name* was successfully opened on the Desktop; otherwise, returns 0.

Object_name can be a WPS object ID (the unique string preceded with a '<' and terminated with a '>') assigned to the object when it was created (e.g. <WP_DESKTOP>) or a fully qualified file name.

View specifies the view to be opened and can contain either a numeric value or the equivalent string. The function will pass all numeric values to the underlying wpOpen() or wpViewObject() function without testing the value for validity.

0 - DEFAULT
1 - ICON
2 - SETTINGS
3 - HELP
4 - RUNNING
5 - PROMPTDLG
121 - PALETTE

Flag can contain a 1 indicating that an existing view of an object can be opened on top of the Desktop (resurfaced) by calling the wpViewObject method or a 0 indicating that the view specified in *view* is to be opened using the wpOpen method. The following comment originated in the description of the wpOpen method:

"In general, wpViewObject should be used instead of the wpOpen method. This is because wpViewObject takes into consideration the setting in the Object Open Behavior field on the Window page of the Settings notebook for the object. If a view of the object is already open, wpViewObject will depending on the setting of the Object Open Behavior field, either display the existing window for the object or create a new object."

"In contrast, wpOpen always opens a new view of the object. Under certain circumstances this might be called for, but, under most circumstances, wpViewObject should be called instead."

Figure 3 - SysOpenObject() function

```
SysMoveObject( object_name, →
                          → object_destination )
```

Returns 1 if *object_name* was successfully moved to *object_destination*; otherwise, returns 0. If the object already exists in the destination location, it is not moved and a 0 is returned.

Both *object_name* and *object_destination* can be a WPS object ID (the unique string preceded with a '<' and terminated with a '>') assigned to the object when it was created (e.g. <WP_DESKTOP>) or a fully qualified file name.

Figure 4 - SysMoveObject() function

```
SysCopyObject( object_name, →
                        → object_destination )
```

Returns 1 if *object_name* was successfully copied to *object_destination*; otherwise, returns 0. If the object already exists in the destination location, it is not copied and a 0 is returned.

Both *object_name* and *object_destination* can be a WPS object ID (the unique string preceded with a '<' and terminated with a '>') assigned to the object when it was created (e.g. <WP_DESKTOP>) or a fully qualified file name.

Note 01: *The copied object will not have an OBJECTID whether the original object had one assigned or not.*

Note 02: *Some of the object's other properties are not copied along with the object. Specifically, ASSOCTYPE= belonging to the original object does not appear on the copy. This is consistent with what occurs when using drag & drop to copy an object.*

Figure 5 - SysCopyObject() function

Tips on Using REXX and the Workplace Shell

Any changes which are made to an open Settings notebook via **SysSetObjectData()** are not necessarily reflected in that notebook until it is closed and reopened.

If the same key name is specified more than once within a setup string, it generally appears as though the first key name-value pair is the one which prevails; however, that is not always the case.

Where a numeric value of 0 or 1 is used to represent NO or YES respectively; it appears that any numeric value other than 0 will be used as if the value had been 1.

Some of the alphabetic values of the key name=value pairs have been found to be case sensitive with uppercase being required; therefore, all alphabetic values should be created in uppercase.

A new line character, '0A'x, may be used to cause a value such as Title to occupy more than one line. Also, it appears that the occurrence of the escape character, ^, causes a new line to be created; however, 2nd and subsequent escape characters used for this purpose appear to be ignored.

If both ICONFILE and ICONRESOURCE are specified in the same setup string, ICONFILE prevails.

An OBJECTID should not be assigned to an object defined as a template since this would lead to multiple objects with the same OBJECTID.

The object pointer or handle can only be retrieved via the **wpclsQueryObject** method or the **WinQueryObject** function, respectively (neither of which are currently available via REXX).

Prior to Warp, there was no method for altering the background characteristics for a folder other than the bitmap image name (e.g. image vs. color; normal, scaled or tiled image; etc.) using either **SysCreateObject()** or **SysSetObjectData()**. Warp allows all of the characteristics of the Desktop background to be specified.

Prior to Warp, there was no method for altering "Always maintain sort order" using either SysCreateObject() or SysSetObjectData(). Warp introduced the "ALWAYSSORT=YES;" setup string parameter.

If OPEN=SETTINGS is specified, the

program object's notebook is opened; however, if OPEN=DEFAULT is specified, the program object is launched (its icon is cross-hatched) and the program appears in the task list but it does not come to the foreground without either a second call to **SysSetObjectData()** or manual intervention.

Warp / REXX Mystery Failures

There has been a "fix" in Warp Version 3 implemented by IBM that is subtly causing REXX programs, that ran with prior versions of OS/2, to fail. The culprit is the lack of file handles in the OS/2 session where the REXX program is running.

The default number of file handles, a resource required for each open file, is, and has been, twenty. Of the twenty, fifteen are available for user programs with five being reserved for system-related files. Prior to Warp Version 3.0, when multimedia support was installed it changed the default for the number of file handles from twenty to eighty. Apparently, this was being done without the knowledge of the kernel developers and they deemed it necessary to "correct" this problem.

The end result is that if you have a REXX program that inadvertently references file with any of the input/output (I/O) functions: CHARIN(), CHAROUT(), CHARS(), LINEIN(), LINEOUT(), or LINES() or uses any library functions that do not close all of their files (for example - the SysGetMessage() function); these programs may begin to fail. The only way to prevent this from happening is to increase the number of file handles available in the particular session where the program is running. This can be done with the GrowHandles() C function. Quercus Systems has implemented this capability in their latest version of REXXLIB, a commercial product that is available in a fully functional "demo" form from your favorite OS/2 BBS or repository. With the addition of REXXLIB.DLL, you can call the DOSFILEHANDLES() function and specify the number of file handles you want to be available for that session. Once the number of file handles has been increased, the larger number of file handles remain available to that session until the session is closed.

```
/* 9506LS07.CMD (RXLS05.CMD) - Build your own Launchpad */
                                                        /* 0002 */
LaunchPadID = '<WP_LAUNCHPAD>'                          /* 0003 */
location    = '<WP_OS2SYS>'                             /* 0004 */
title       = 'LaunchPad'                               /* 0005 */
class       = 'WPLaunchPad'                             /* 0006 */
                                                        /* 0007 */
/*------------------------*\                            /* 0008 */
|  Setup LaunchPad string  |                            /* 0009 */
\*------------------------*/                            /* 0010 */
parameters =,                                           /* 0011 */
   'CCVIEW=NO;'                                  ||,    /* 0012 */
```

```
      'HELPPANEL=32253;'                                  ||,    /* 0013 */
      'ICONRESOURCE=74 PMWP.DLL;'                         ||,    /* 0014 */
      'LPACTIONSTYLE=OFF;'                                ||,    /* 0015 */
      'LPCLOSEDRAWER=YES;'                                ||,    /* 0016 */
      'LPDRAWERTEXT=YES;'                                 ||,    /* 0017 */
      'LPFLOAT=NO;'                                       ||,    /* 0018 */
      'LPHIDECTLS=YES;'                                   ||,    /* 0019 */
      'LPSMALLICONS=YES;'                                 ||,    /* 0020 */
      'LPTEXT=YES;'                                       ||,    /* 0021 */
      'LPVERTICAL=YES;'                                   ||,    /* 0022 */
      'NOPRINT=YES;'                                      ||,    /* 0023 */
      'FPOBJECTS='                                        ||,    /* 0024 */
         '<WP_DRIVES>,'                  /* 01 */         ||,    /* 0025 */
         '<WPPO_HPLaserJ>,'              /* 02 */         ||,    /* 0026 */
         '<Corel_Draw!>,'                /* 03 */         ||,    /* 0027 */
         '<WP_MEGASCAN^3>,'              /* 04 */         ||,    /* 0028 */
         '<WP_OS2WIN>,'                  /* 05 */         ||,    /* 0029 */
         '<TAPCIS>,'                     /* 06 */         ||,    /* 0030 */
         '<WordPerfect_for_Windows>,'    /* 07 */         ||,    /* 0031 */
         '<WP_GAMES>,'                   /* 08 */         ||,    /* 0032 */
         ';'                                              ||,    /* 0033 */
      'OBJECTID=' || LaunchPadID || ';'                           /* 0034 */
                                                                  /* 0035 */
   call SysCreateObject class,,                                   /* 0036 */
                        title,,                                   /* 0037 */
                        location,,                                /* 0038 */
                        parameters,,                              /* 0039 */
                        'R'                                       /* 0040 */
   if RESULT <> 1 then                                            /* 0041 */
      do                                                          /* 0042 */
         say '   Error creating launchpad'                        /* 0043 */
         exit                                                     /* 0044 */
      end                                                         /* 0045 */
                                                                  /* 0046 */
   /*----------------------*\                                     /* 0047 */
   |  Setup drawer strings  |                                    /* 0048 */
   \*----------------------*/                                     /* 0049 */
   drawer_01 =,                                                   /* 0050 */
      'DRAWEROBJECTS=01,'                                 ||,    /* 0051 */
         '<BackMaster>,'                                  ||,    /* 0052 */
         'L:\WWW,'                                        ||,    /* 0053 */
         ';'                                                      /* 0054 */
                                                                  /* 0055 */
   drawer_02 =,                                                   /* 0056 */
      'DRAWEROBJECTS=02,'                                 ||,    /* 0057 */
         '<WPPO_FxPrint>,'                                ||,    /* 0058 */
         ';'                                                      /* 0059 */
                                                                  /* 0060 */
   drawer_03 =,                                                   /* 0061 */
      'DRAWEROBJECTS=03,'                                 ||,    /* 0062 */
         'c:\os2addon\pmcamera.exe,'                      ||,    /* 0063 */
         ';'                                                      /* 0064 */
```

```
                                                                  /* 0065 */
drawer_05 =,                                                      /* 0066 */
   'DRAWEROBJECTS=05,'                                    ||,     /* 0067 */
      '<WP_WINFS>,'                                       ||,     /* 0068 */
      '<WP_WIN2WIN>,'                                     ||,     /* 0069 */
      '<WP_DOSFS>,'                                       ||,     /* 0070 */
      '<WP_DOSWIN>,'                                      ||,     /* 0071 */
      '<WP_OS2FS>,'                                       ||,     /* 0072 */
      ';'                                                         /* 0073 */
                                                                  /* 0074 */
drawer_06 =,                                                      /* 0075 */
   'DRAWEROBJECTS=06,'                                    ||,     /* 0076 */
      '<IAK_SLIPPM>,'                                     ||,     /* 0077 */
      '<ADV_DIALER>,'                                     ||,     /* 0078 */
      '<WP_OS/2_CIM__>,'                                  ||,     /* 0079 */
      '<WP_INTERNET^C>,'                                  ||,     /* 0080 */
      '<WP_XTALK_^MK_>,'                                  ||,     /* 0081 */
      ';'                                                         /* 0082 */
                                                                  /* 0083 */
drawer_07 =,                                                      /* 0084 */
   'DRAWEROBJECTS=07,'                                    ||,     /* 0085 */
      '<WP_WP_5.1>,'                                      ||,     /* 0086 */
      '<WP_REXX_^HAND>,'                                  ||,     /* 0087 */
      ';'                                                         /* 0088 */
                                                                  /* 0089 */
/*-------------------------------------------------*\             /* 0090 */
|  Add drawers to LaunchPad & open it on Desktop  |              /* 0091 */
\*-------------------------------------------------*/             /* 0092 */
call SysSetObjectData LaunchpadID, drawer_01                      /* 0093 */
call SysSetObjectData LaunchpadID, drawer_02                      /* 0094 */
call SysSetObjectData LaunchpadID, drawer_03                      /* 0095 */
call SysSetObjectData LaunchpadID, drawer_05                      /* 0096 */
call SysSetObjectData LaunchpadID, drawer_06                      /* 0097 */
call SysSetObjectData LaunchpadID, drawer_07                      /* 0098 */
                                                                  /* 0099 */
call SysOpenObject LaunchpadID, 0, 1                              /* 0100 */
exit                                                              /* 0101 */
```

Referenced Resources

REXXLIB - OS/2 REXX API ($20.00 to $50.00)

Quercus Systems
14500 Big Basin Way, Suite E
Saratoga, CA 95070
800-440-5944 orders
408-867-7399 voice
408-867-7489 FAX
408-867-7488 BBS
CompuServe, PCVENA, Sec 11 (GO CIS:QUERCUS - Charles Daney 75300,2450)

REXX Reference Summary Handbook ($27.95) by Dick Goran

C F S Nevada, Inc.
953 E. Sahara Ave, Suite 9B
Las Vegas, Nevada 89104-3012
800-739-9672 orders
702-732-9616 voice
702-732-3847 FAX

Biographical info - Dick Goran:

A veteran of the computer industry for 34 years, Goran is a contributing editor and monthly columnist for *OS/2 Magazine* and serves as one of IBM's OS/2 Advisors on CompuServe. Considered one of the leading authorities on OS/2 REXX, Goran authored the best-selling, award-winning *REXX Reference Summary Handbook*.

His company, C F S Nevada, Inc. located in Las Vegas, offers the OS/2 REXX class to the public as well as publishing the *REXX Reference Summary Handbook*. Goran speaks to OS/2 User Groups and other industry associations throughout the country on both OS/2 and the REXX programming language.

Goran returned to the software business in 1991 after having sold his IBM mainframe systems software development business in 1987, retiring, and relocating to Las Vegas from Boston. While in Las Vegas, Goran began hosting an evening radio talk show and has since appeared in several movies.

Goran is highly visible in the OS/2 forums on CompuServe and can be reached via e-mail at *71154.2002@CompuServe.com*. He also maintains an FTP directory at *ftp.netcom.com:/pub/dg/dgoran* where many of his OS/2 REXX utilities are available to the public at no charge.

S/REXX by BENAROYA

David Salthouse
Open Direct

Pages 284-290

S/REXX by BENAROYA

David Salthouse Open Direct
david.salthouse@utopia.fnet.fr

History

1989 SEDIT

1991 S/REXX

1994 S/REXX DEBUGGER

Platforms

AIX
HPUIX
SUNOS
SOLARIS
IRIS
ULTRIX
LINUX

S/REXX VERSION 4 REXX
NO LIMITS ON:-

Procedure Size
Expression complexity
Nesting of Parenthesis
Variables number and content
Recursive function Depth
Argument number and size

LANGUAGE EXTENSIONS:-

Full Function EXECIO
Dynamic Loading of Routines
Multiple Procedure Expose
LEAVE or ITERATE within an INTERPRET
CD
DO name IN expr; ..... End;
OPTION case, setenv, load
LOWER,UPPER
PARSE EXTERNAL and PARSE NUMERIC
{} DO; END
[] SUBSTR

S/REXX VERSION 4 BUILT-IN FUNCTIONS

Dialog Management Openlook or Motif:-
buttons
labels
input fields
toggles

Other BUILT_IN FUNCTIONS:-

ARCH()
CHDIR(),MKDIR(),RM()
LINEIN(),LINEOUT()
UNIX(cmd,stem)
STATUS()
date(date('b')+7)

Programming Interface
add user supplied builtin functions
embed S/REXX into C applications

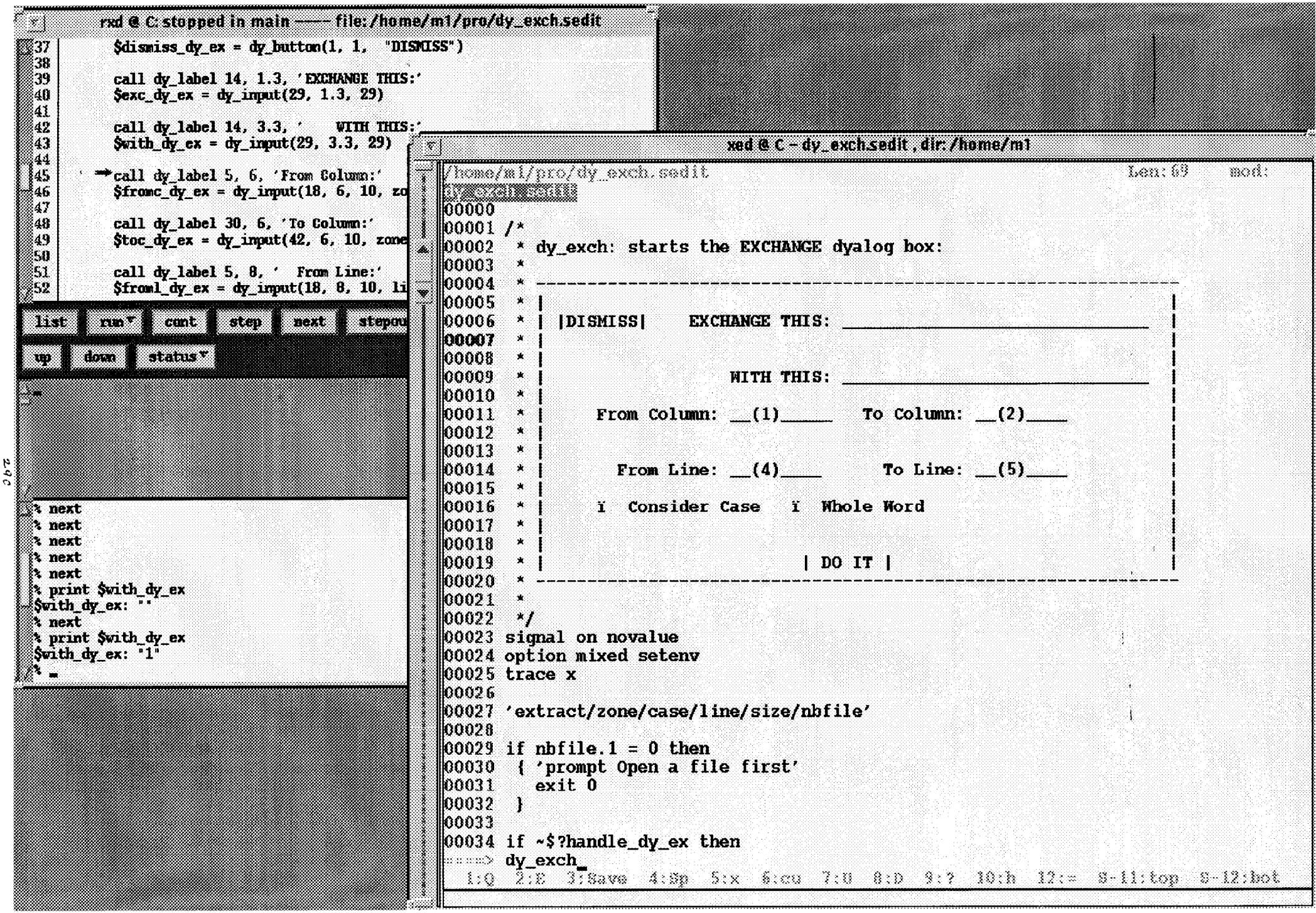
TEST DRIVE SEDIT, S/REXX AND RXD

download from

http://www.sedit.com/sedit

http://www.portal.com/~sedit

OR.....


rxd @ C: stopped in main —— file:/home/m1/pro/dy_exch.sedit
37 $dismiss_dy_ex = dy_button(1, 1, "DISMISS")
38
39 call dy_label 14, 1.3, 'EXCHANGE THIS:'
40 $exc_dy_ex = dy_input(29, 1.3, 29)
41
42 call dy_label 14, 3.3, ' WITH THIS:'
43 $with_dy_ex = dy_input(29, 3.3, 29)
44
45 call dy_label 5, 6, 'From Column:'
46 $fromc_dy_ex = dy_input(18, 6, 10, zo
47
48 call dy_label 30, 6, 'To Column:'
49 $toc_dy_ex = dy_input(42, 6, 10, zone
50
51 call dy_label 5, 8, ' From Line:'
52 $froml_dy_ex = dy_input(18, 8, 10, li
list
run
cont
step
next
stepou
up
down
status
% next
% next
% next
% next
% next
% print $with_dy_ex
$with_dy_ex: ""
% next
% print $with_dy_ex
$with_dy_ex: "1"
%
xed @ C - dy_exch.sedit , dir: /home/m1
/home/m1/pro/dy_exch.sedit Len: 69 mod:
dy_exch.sedit
00000
00001 /*
00002 * dy_exch: starts the EXCHANGE dyalog box:
00003 *
00004 * ----------
00005 * |
00006 * | |DISMISS| EXCHANGE THIS: _____ |
00007 * |
00008 * |
00009 * | WITH THIS: _____ |
00010 * |
00011 * | From Column: __(1)_____ To Column: __(2)____ |
00012 * |
00013 * |
00014 * | From Line: __(4)____ To Line: __(5)____ |
00015 * |
00016 * | ï Consider Case ï Whole Word |
00017 * |
00018 * |
00019 * | | DO IT | |
00020 * ----------
00021 *
00022 */
00023 signal on novalue
00024 option mixed setenv
00025 trace x
00026
00027 'extract/zone/case/line/size/nbfile'
00028
00029 if nbfile.1 = 0 then
00030 { 'prompt Open a file first'
00031 exit 0
00032 }
00033
00034 if ~$?handle_dy_ex then
=====> dy_exch
1:Q 2:E 3:Save 4:Sp 5:x 6:cu 7:U 8:D 9:? 10:h 12:= S-11:top S-12:bot

A REXX-based Stock Exchange Real-time Client/Server Environment for Research, Educational and Public Relations Purposes: Implementation and Usage Issues

Martin P. Misseyer
Lou W. M. Guse
Armoud W. Morsink
Vrije Universiteit Amsterdam

Pages 292-322

A REXX-based Stock Exchange Real-time Client/Server Environment for Research, Educational and Public Relations Purposes: Implementation and Usage issues

Martin P. Misseyer, Lou W.M. Güse, Arnoud W. Morsink

Vrije Universiteit
Faculty of Economic Sciences, Business Administration and Econometrics
Department of Information Systems

Amsterdam, April 1995

Abstract

For many years now the Faculty of Economic Sciences, Business Administration and Economics of the Vrije Universiteit in Amsterdam propagates to impart students of economics scientific 'real market' skills and experience in, for example, portfolio management. Aside from the risk neither the faculty nor most of the students have sufficient means to practice in portfolio management. In the early 1980s the idea evolved at the faculty to develop and use a portfolio management simulation. The Amsterdam Stock Exchange (ASE) granted the faculty in 1983 a free of charge data link with its administrative clearing information system. The data link provided the faculty with real-time data including stocks traded (time, price and volume), exchange news, stock splits and many more. At the time the technology used was relatively simple: the data link consisted of a 1200 bps modem connection using the X-modem protocol to receive data. The received data were put in flat files and were in turn read by a small, written in C, in-house developed portfolio management simulation system named TRANSPAS.

In 1990s it became inevitable for ASE to acquire a modern trading system as more and more trade leaked to more sophisticated foreign exchanges (London, Paris, and Frankfurt). ASE developed a new trade system, entailing major operational and technical changes. In September 1994 a new, fully automated, trade system became operational for complete administration of all ASE transactions (to be) made. Nowadays at the ASE tens of millions of stocks, bonds, futures change hands every day.
The major changes were twofold. First the trade itself was re-engineered by ASE, however this is not discussed in this paper. Secondly, a tremendous change at the technology level was unavoidable. ASE moved from simple analogue asynschronous communication to X.25-based digital synchronous communication (SDLC), anticipating on the need for both much more capacity as data streams increase (re-engineering!) and for full reliable data-link monitoring. Therefore, the faculty was faced with the fact that TRANSPAS as well as its data link became outdated, and implied that the whole system had to be rebuilt.

In 1993 the first author of this paper headed the project team, a composition of colleagues (researchers, graduates, automation staff) and enthusiastic undergraduates, to rebuild TRANSPAS and its ASE data link. The first step the project team took, before setting up several projects, was re-examining the faculty goal. After thorough research, two goals were aimed at. First, the basis of the integrated environment should be based on state-of-the-art relational database technology (the database project was named after the database to be developed: BeursBase) in which all the raw ASE data received would be stored real-time/on-line for a long period of time, preferably three years or more. Secondly, the project team distinguished three major application areas for the database: research, education and public relations. For each of these areas a specific range of application programs

must be developed. Obviously, one of the core application programs is the portfolio management simulation system TRANSPAS, which was renamed into: VUPOS. It became clear that VUPOS could be used in all three areas defined.

Concurrently with the ASE new trade system, the faculty-built REXX-based Client/Server (C/S) became operational. Though already halfway implementation the faculty was informed by the Computer Services Center about its new strategy: the IBM S/390 host facility, the VM Server in the REXX C/S environment (and the basis for both BeursBase and several applications including VUPOS) would be stopped at the end of 1995. It would be replaced by a AIX cluster of 4 very large 590 RS/6000 mid 1994. When the project team was acquainted with the news immediate action was taken. In contrast with the SQL/DS version installed DB2/6000 (AIX) supports full C/S. For several reasons the project team decided to develop in parallel a second, REXX-based C/S environment using the AIX host as Server. This 'new' C/S environment - referred to as the AIX C/S environment - went in operation last January 1995. As the 'old' VM C/S environment was primarily based on REXX, porting the applications to a 'new' AIX C/S environment was relatively simple. Both VM C/S and AIX C/S environments are now fully operational and perform as was planned for, having many similar as well as distinguishing characteristics. The portfolio management Simulation (VUPOS) for the VM C/S environment is written in CSP, and is already used by hundreds of students. Since the AIX C/S does support full C/S the project team was able to develop VUPOS in VX-REXX. Recently the development of this version of VUPOS has entered its final stage. Other applications in VX-REXX, APL/2, VisualGen and VisualAge under construction, range from an import/export facilities to fundamental and technical analysis, and are primarily developed for the AIX C/S environment. The first quarter of 1995 will be used for large scale tests of the system.

This paper presents the design, development, and implementation of these C/S systems from both developer and user views and from both technical and non-technical points of view.

1 General introduction

1.1 Students, portfolio management and TRANSPAS

For many years now the Faculty of Economic Sciences, Business Administration and Economics (FEWEC) of the Vrije Universiteit in Amsterdam (The Netherlands) propagates to impart students of economics scientific 'real market' skills and experience in, for example, portfolio management. Aside from the risk neither FEWEC nor most of the students have sufficient means to practice in portfolio management. In the early 1980s the idea evolved at FEWEC to develop and use a portfolio management simulation. The Amsterdam Stock Exchange (ASE) granted FEWEC in 1981 a free of charge data link with its administrative clearing information system. The data link provided FEWEC with real-time data including stocks traded (time, price and volume), exchange news, stock splits and many more. At the time the technology used was relatively simple: the data link consisted of a 1200 bps modem connection using the X-modem protocol to receive data. The received data were put in flat files and were in turn read by a small, written in C, in-house developed portfolio management simulation system named TRANSPAS.

1.2 Developments at the Amsterdam Stock Exchange

Early in the 90s it became inevitable for ASE to acquire a modern trading system as more and more trade leaked to more sophisticated foreign exchanges (London, Paris, and Frankfurt). ASE developed a new trade system, entailing major operational and technical changes. In September 1994 a new, fully automated, trade system became operational for complete administration of all ASE transactions (to be) made. Nowadays at the ASE tens of millions of stocks, bonds, futures change hands every day.

The major changes were twofold. First the trade itself was re-engineered by ASE, however this is not discussed in this paper. Secondly, a tremendous change at the technology level was unavoidable. ASE moved from simple analogue asynschronous communication to X.25-based digital synchronous communication (SDLC), anticipating on the need for both much more capacity as data streams increase (re-engineering!) and for full reliable data link monitoring. Therefore, FEWEC was faced with the fact that TRANSPAS as well as its data link became outdated, and implied that the whole system had to be rebuilt.

1.3 Rebuilding TRANSPAS: a project plan

In 1993 the first author of this paper headed the project team, a composition of colleagues (researchers, graduates, automation staff) and enthusiastic undergraduates, to rebuild TRANSPAS and its ASE data link. The first step the project team took, before setting up several projects, was re-examining FEWECs goal. After thorough research, two goals were aimed at. First, the basis of the integrated environment should be based on state-of-the-art relational database technology (the database project was named after the database to be developed: BeursBase) in which all the raw ASE data received would be stored real-time/on-line for a long period of time, preferably three years or more. Secondly, the project team distinguished three major application areas for the database: research, education and public relations. For each of these areas a specific range of application programs must be developed. Obviously, one of the core application programs is the portfolio management simulation system TRANSPAS, which was renamed into Vrije Universiteit POrtfolio Simulation, in short VUPOS. It became clear that VUPOS could be used in all three areas defined.

The VUPOS/BeursBase project defined the following phases:
1° Provide a strategy plan, including a re-examination of FEWECs goal(s);
2° Evaluate possible solutions, alternatives for the new system;
3° Design a new organizational setting;
4° Design several subprojects in which the system should be developed and implemented;
5° Provide a maintenance structure based on the strategy plan, including a costs/benefits analysis.

Phases 1°, 2° and 5° lie beyond the scope of this paper, except for a re-examination of FEWECs goal (phase 1° partially) these phases are not discussed. This paragraph elaborates on the subprojects defined (phase 3°). Phase 4°, the design of several subprojects is discussed in the next paragraph. Before continuing with the subsequent section it has to be said that the focal point of this paper is an exemplification and a discussion of the implementation and usage aspects of the REXX-based Client/Server environment developed at FEWEC.

1.4 The strategy plan: a re-examination of FEWECs goal

In the early 1980s TRANSPAS was set up as a kind of 'test' or 'toy' system primarily by students, two lecturers and a system administrator. At the time focus was purely on practice: how can we implement a portfolio simulation system for usage in education? A re-examination of this simple FEWEC goal is shown in figure 1.

At best stock data provided by Beursdata should be stored in a underlying relational database (BeursBase). Then three user application areas can be distinguished: *research, education* and *public relations*. The research and education user applications areas seem more likely than the public relations application area. True, an university position stems from the quality of research and education. The public relations more or less benefit from these facts. At this moment universities in the Netherlands are faced with a significantly declining number of students, partially because of the aging population and partly because of a declining

willingness to study, and therefore competition is high. This is where the public relations part plays a role: FEWEC needs ways to attract potential students. The public relations user application area is distinguished for this goal. FEWEC members, lecturers, researchers and student can aid in application development for this purpose. Other public relations activities are for example finance , i.e. stock and bond investment competitions, school lectures, educational seminars, and media publicity.

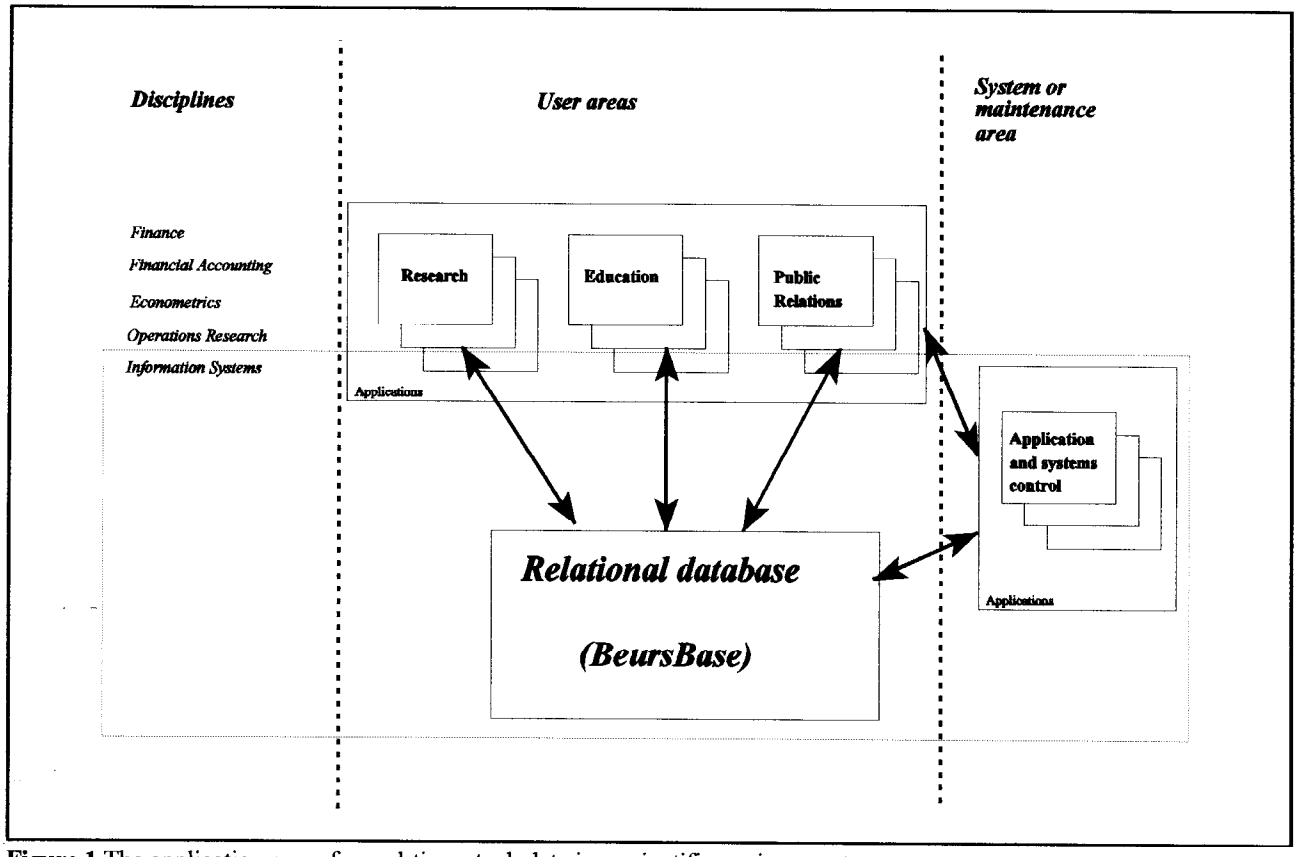


Figure 1 The application areas for real-time stock data in a scientific environment.

The fourth application area is the one for system and maintenance. In this area users, programs and database control rather than stock data are the issue of concern. Finally, each of the user application areas distinguished can be viewed from a variety of economic disciplines. The most relevant disciplines are among Finance, Information Systems and Information Technology, Econometrics, Financial Accounting and Operations Research or Management Science.

The information above can be concentrated into a general purpose framework, a matrix structure in which the user application areas are defined as column and the economic disciplines as rows. For each matrix cell one or more specific users application programs can be developed. Or, if application programs are already available abundantly and are easily adapted in current environment, they should be used instead Furthermore, some application programs if generally developed can be used in more than one area distinguished, and/or in more than one discipline. In that case one can look for or develop application programs for more widespread usage. In table 1, some examples are provided.

The fourth application area forms an exception to the framework. This is primary the area of the Information Systems and Computer Science disciplines and involves more fundamental research on information systems, decision support systems, databases, (tele)communication and networking.

Application area / Discipline	Research	Education	Public Relations
Finance	- Portfolio management and theory; - Theory on market efficiency; - International stock markets; - Development of performance indicators - Analysis of stock market and financial investment data - Application of IS and DSS	- Portfolio management simulation; - Data analysis; - Application of IS and DSS; - Practical classes; - Workshops; - Seminars;	- Financial investment competitions; - Demonstrations; - Financial workshops; - Financial seminars; - Bank and stock market sponsoring; - Media (financial and newspapers);
Information Systems	- Design DSS, ES, KNN systems - Database concepts and theory - System and application development concepts- (Client/Server, Object Orientation) - Application of IS and DSS - Human computer interface	- Data analysis; - Application of IS and DSS; - Practical classes; - Workshops; - Seminars;	- Financial investment competitions; - IS demonstrations; - IS workshops; - IS seminars; - Media (IS and newspapers);
Econometrics	- Statistical (exploratory) data analyses - Longitudinal and time series analysis - Application of IS and DSS - Design of simulation models	- Data analysis; - Application of IS and DSS; - Practical classes; - Workshops; - Seminars;	- Financial investment competitions; - Demonstrations; - Workshops; - Seminars; - Media;
Financial accounting	- Analysis of performance indicators - Analysis of stock and equity issues - Application of IS and DSS	- Data analysis; - Application of IS and DSS; - Practical classes; - Workshops; - Seminars;	- Financial investment competitions; - Demonstrations; - Workshops; - Seminars; - Media;
Monetary economics	- General market theories - Impact stock market on economy - Application of IS and DSS	- Data analysis; - Application of IS and DSS; - Practical classes; - Workshops; - Seminars;	- Financial investment competitions; - Demonstrations - Workshops; - Seminars; - Media.

Table 1 Framework for application program development.

1.5 Criteria for the C/S environment development: in search for processing power

The requirements for a BeursBase and VUPOS platform needed to be specified. Several criteria were defined to determine the amount of processing and database power necessary for BeursBase, VUPOS and other application usage. The project team came up with the following criteria:

- ☐ Number of users in every area distinguished;
- ☐ Number of applications in every area;
- ☐ Type of database processing;
- ☐ In-house experience with systems;
- ☐ Costs/benefits;
- ☐ Designing for flexible systems in terms of portability, efficiency and effectiveness;

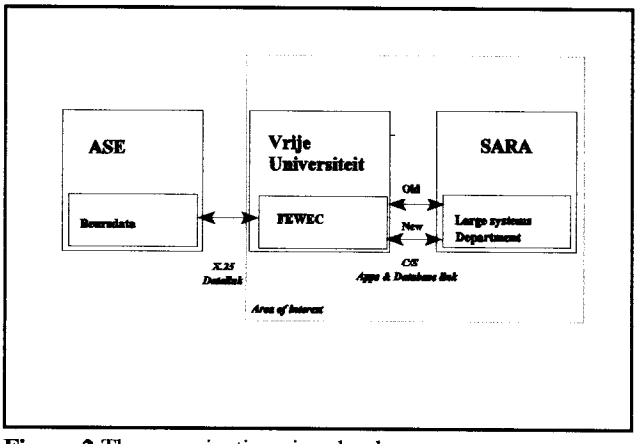


Figure 2 The organizations involved.

Having limited resources available, added to the fact that use of SARAs, the academic computer center of the two universities in Amsterdam, was already paid for up to 1996, the choice was simple. As the department of Information Systems was already using SARAs facilities, the cooperation was intensified. The organizational setting is shown in figure 2. Beursdata, ASEs data vendor provides FEWEC since 1981 with real-time stock exchange data. In 1993 the communication link was upgraded to a X.25 structure. FEWEC, with approximately 3100 students the largest of 15 faculties of the Vrije Universiteit, transforms the real-time ASE data into SQL data-format and puts it into BeursBase (SARA). Ideally, FEWEC members, lectures, researchers, assistants and students should be able to use the data for many different purposes. Data should be available both directly by extracting it (by query) and indirectly by using application programs.

1.6 SARA the computer services center

Since 1987 SARA supports an S/370 facility. During the years it was first expanded from a 3090-150 to a 3090-180 and in 1990 it was replaced by a huge 3090-600VF. The 6 processors were used to run VM, MVS and AIX concurrently. The project team selected the Virtual Machine (VM) operating system to become the BeursBase database server because it came with SQL/Data System installed. The MVS operating system did have DB2 installed, though this facility was not supported for general usage. During the development of the REXX-based Client/Server environment it was found that the SQL/DS installed did not support a server mode. SARA didn't want to invest in a higher SQL/DS level, as it had other plans. But first, the project team decided to develop itself the necessary Client/Server programs, as will be discussed in paragraph 2. Without a 'true' Client/Server environment the project team also decided to develop the first user applications on the VM host (with the IBM development environment Cross Systems Product).

The first signs of strategic movements were already disclosed in 1992 as the 3090-600VF (S/370) was replaced by a 9021-720 (S/390). Developments accelerated in the fall of 1993 when SARA announced it would stop its VM service at the end of 1995. First it reduced the 9021 to a 580 when it stopped the AIX service on this machine. SARA decided that the future role of AIX would become more important, therefore it adopted a new facility, a one of IBMs new developments: a AIX cluster of RS/6000 computers. The cluster installed consists of four 590s and three 980s each equipped with 1 Gb RAM and 6 to 10 Gb disk space. The reason why SARA adopted the new hardware is because it is relatively simple to add processing power to the cluster. One simply adds one or more RS/6000s. Another reason is that an AIX cluster supports 'farming', or distributed processing, which was highly necessary for the high performance computing services offered to the more technical faculties Chemistry and Physics.

Though already in the final stage of development of the VM C/S environment, the project team decided not to wait for more developments to come, but to test the REXX-based C/S environments' flexibility immediately. Because the new AIX facility was provided with IBMs latest version of the DB2 database management system, DB2/6000, now a real C/S environment became an option. The project team decided to develop in parallel to the VM C/S environment an AIX C/S environment in which DB2/6000 would operate as a real database server. It was hoped for that REXXs flexibility would minimize the redevelopment effort.

1.7 Keeping focus

Inorder to avoid problems in discussing the REXX-based C/S environments, the following commentary is necessary. First, ***not one but two*** REXX-based C/S environments were developed. The VM C/S environment is referred to as the 'old' C/S environment, the AIX C/S environment is referred to as the 'new' C/S environment. Secondly, the C/S environments have two levels: a X.25 data link level (ASE-FEWEC) and a database level (FEWEC-SARA). The C/S implementations have a common,

i.e. fixed, X.25 data link (receiving ASE data). With respect to implementation, the area of interest in this paper is primarily the database level (see figure 2). The difference in implementation of the C/S environments stems from the fact that DB2/6000 (the AIX RDBMS) ***is*** and SQL/DS (version 2.2 of the VM RDBMS used) ***is not*** a database server. Thus to establish a VM C/S environment we needed to develop our own C/S environment. From an application and database perspective the VM C/S environment is not a genuine C/S environment because both applications, e.g. VUPOS, and database, i.e. BeursBase, reside at the host. This is in contrast with the AIX C/S environment which fully supports client applications. Thirdly, both systems are fully operational and perform well. Fourthly, with respect to design, development, and implementation issues both VM C/S and AIX C/S implementations are discussed from developer and user views and from both technical and non-technical points of view. Fifth, after VUPOS was implemented further application development for the VM C/S environment was stopped. Thus a discussion about future plans and strategy in this paper, refers to the AIX C/S environment.

2 The design for a real-time Stock Exchange client/server environment

2.1 Moving to a client/server environment: the first level

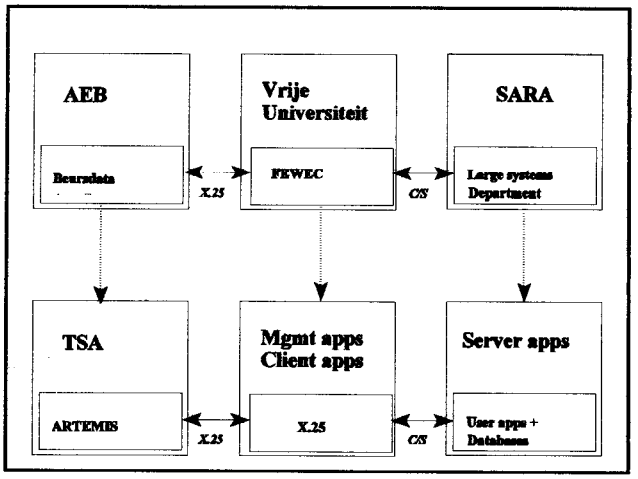


Figure 3 The organizations involved and their systems.

Figure 3 shows the ideal setup of the organizations involved and their systems. The ARTEMIS system of Beursdata is fed on a real-time basis by ASEs trading system TSA. FEWEC has to develop X.25- based communications, a local real-time data link, including a subsystem for temporary data storage. Also FEWEC has to adapt a C/S environment for data and information. Data received should be inserted in a database server (BeursBase); information should be retrieved by client applications from the host database. The C/S environment should be developed using open communication standards.
For manageability reasons the REXX-based Stock Exchange C/S environment project is divided into the following sub-projects, see table 2.

3 Client/Server environments

3.1 The Client/Server concept

Since its introduction, the concept of Client/Server has been discussed among a broad range of disciplines by a large number of people. Scientists, business professionals and many others have been vividly discussing of what one should and should not include in the C/S concepts. As there are so many distinguishable perspectives as opinions, no full-proof C/S definition has been formulated, so far. This paragraph elaborates on our ideas of C/S, hereby avoiding great difficulties and long discussions in placing our C/S environments into general C/S frameworks.

3.2 The five Client/Server levels

C/S can be viewed from both technical and non-technical perspectives. The C/S concept is limited to four aspects, namely, data, database, application programs and users. These four areas include both technical and non-technical perspectives. In general one can distinguish five C/S levels which are exemplified in figure 4. Every higher C/S level inherits lower level functionality.

In the next paragraph first the general levels of C/S are discussed. Then table 2 is further explained by schemes of the architectures of the two C/S environments.

Project	Subproject	Description
A) The X.25 data link	(1) X.25 Communication control	The X.25 data link is logically divided into two parts. The first part receives data packets through a number of logical circuits. These data packets are numbered sequentially per logical circuit. The second part validates received data packets on consistency and sequence number. If one or more data packets are missing or corrupted, this part of the X.25 data link does a retransmission request.
	(2) X.25 data link monitor	In order to have control of the X.25 data link and to be able to automate this part of the C/S environment, it is highly recommended to have a GUI-based monitor. This monitor has to show the status of necessary active communication programs, the logical circuits and information about the amount of data and data packets received, retransmitted and the retransmit status. If one or more hardware, software, X.25 or file errors occur, the GUI-based monitor should be able to restart system components, or the system itself.
(B) The relational database BeursBase	(1) Relational database model	The data received from the Stock Exchange has to be formatted appropriately without losing any information for later data manipulation. In compliance with the ARTEMIS data dictionary a relational data model was developed. This data model is logically divided in two parts. At the one hand all X.25 data received is stored into a submodel. On the other hand application and system programs store their own specific data.
	(2) DBA manipulation, monitoring and authorization programs	Though application programs for database, not data, manipulation, database monitoring and authorization are necessary, the are out of the scope of this paper. At the moment these applications are under development and will be REXX-based.
(C) Client/Server environment	(1) data-SQL converter	Before the data received can be stored into BeursBase, one has to remodel the data to the relational model defined. The function of this program is to create SQL DML statements out of the data received.
	(2) BeursBase link	An application which provides the data and session link layers from the PS/2 to the database (BeursBase) for SQL DML statements execution. In the VM C/S this function is partially implemented by a REXX FTP-based program. In the AIX C/S environment this program is replaced by a C/S software package named Client Application Enabler/2 (CAE/2).
	(3) SQL statements executor	This application program executes the SQL DML statements into BeursBase, in both C/S environments. In the 'old' VM C/S the application runs at the VM host, in the 'new' C/S environment it runs at the pc.
	(4) C/S monitoring program	Like the X.25 Monitor the Client/Server Monitor information about the C/S and BeursBase status is necessary. Like the X.25 environment the C/S environments should run continuously and autonomously. For the 'old' VM C/S environment the monitor is slightly different than for the 'new' AIX C/S environment.

Table 2 An overview of the projects and components of the REXX-based C/S environments.

(A) A Client/Server data link.

If the database as well as user application programs reside on one computer system, one cannot speak of C/S. In our C/S environments, the database residing on this computer system, is fed remotely with data. Therefore we argue that, from a data perspective, this is the most simple form of C/S. In general, if one looks at the user, database or application program perspectives this is considered not to be C/S. The reader should keep in mind that all subsequent C/S levels discussed use this 'C/S level'. As will be shown later, it is this data C/S level which distinguished the two C/S environments developed.

(B) Application program clients - application program/database server level

In the second C/S level simple computers of end-users are the clients populated by several relatively small application programs. These application program make use of a computer system acting as a database server. In addition, the remote host system

is reachable by the end-user too, for larger applications programs, as the local computer system lacks performance and/or the communication line capacity is too small for large scale database I/O. Then, in that case the end-user computer system is used as a dumb terminal as both processing and database I/Os are executed at the remote server.

(C) Application program clients - remote database server level

The next higher level of C/S is when all applications are executed at the clients and the host purely acts as a database server. Thus advanced and complex processing is done at the client level. Three requirements have to be fulfilled. First, faster and more complex client computers. Secondly, a sophisticated communication infrastructure is necessary to cope with large scale database I/O and processing. Thirdly, the application programs developed are by far more complex than in the case of C/S level B.

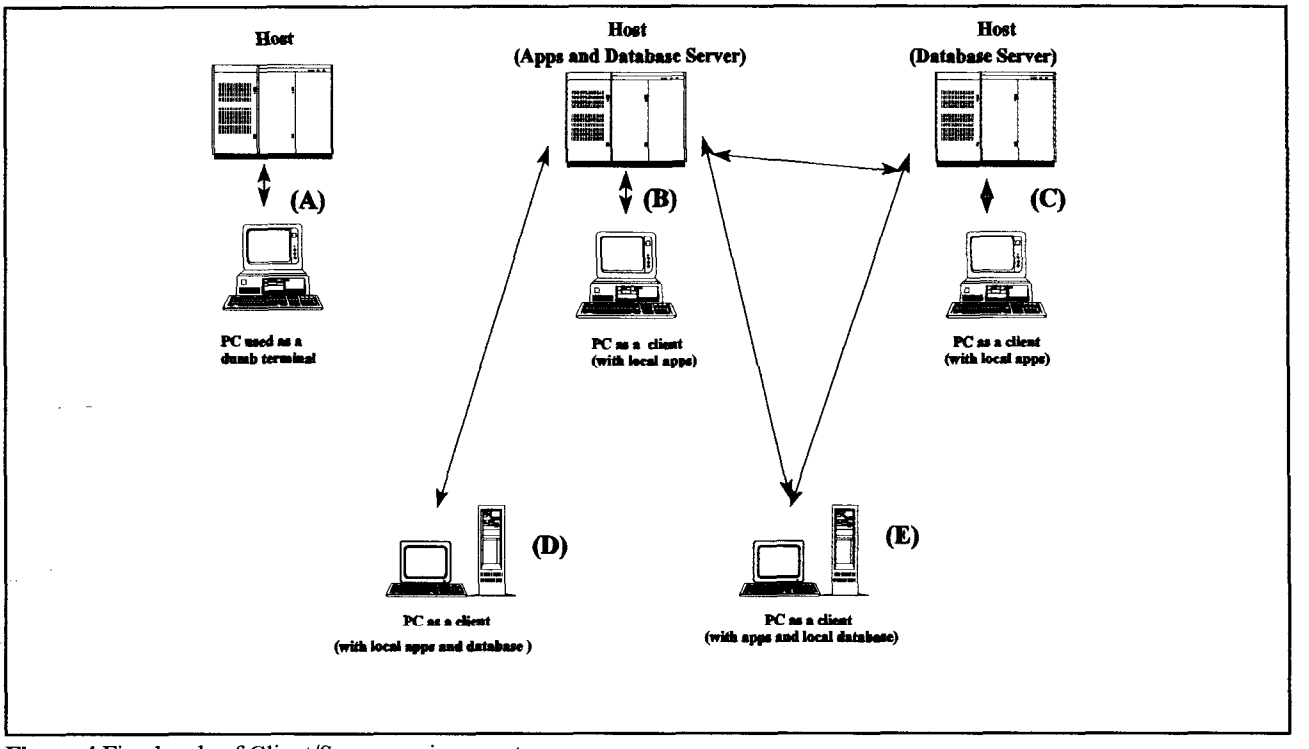


Figure 4 Five levels of Client/Server environments.

(D) The distributed database level

An even more advanced C/S level is established when database processing is distributed and fully integrated in the C/S environment. In this case one has remote as well as local database servers co-operating in a C/S environment. The client computers system can be such a local database server as well. Thus it is not important where the data is stored and where the processing is done. At this level data is stored where it should, for example user specific data is stored as close as possible to the user. In addition, for performance reasons, the distributed database environment decides where to store application specific data. For performance reasons this can be as close as possible to end-users, to some extend introducing data redundancy, or as central as possible at a remote database server. Thus the distributed database environment decides where database processes can be handled at best. At this level client computers can act as application clients as well as local database servers simultaneously. At this C/S

level due to limited client computer system performance (as high performance computing applications programs need) end-users are still able to use application programs directly at some host system, which acts as both database and application server.

(E) The distributed application level

The most sophisticated level of C/S is established when, aside from the distributed database environment, the application programs too are distributed among several clients and servers. Thus the C/S environment decides where to handle data as well as application program processing. For example I/O intensive database operations are executed at a specific database server , CPU intensive calculations are executed on a specific application programs server, small database operations are run on a local server (client) and the remainder of the application programs are executed on another client. In research nor in the business environment has the fifth C/S level been implemented yet.

In summary, the FEWEC-SARA data link in both VM (SQL/DS) and AIX (DB2/6000) C/S environments is purely C/S level A. The AIX C/S environment from an end-user and application programs perspective, is of C/S level B. Before discussing in greater detail the future plans and strategy with respect to the directions the AIX C/S environment will move, the developments of the C/S environments so far, are further discussed.

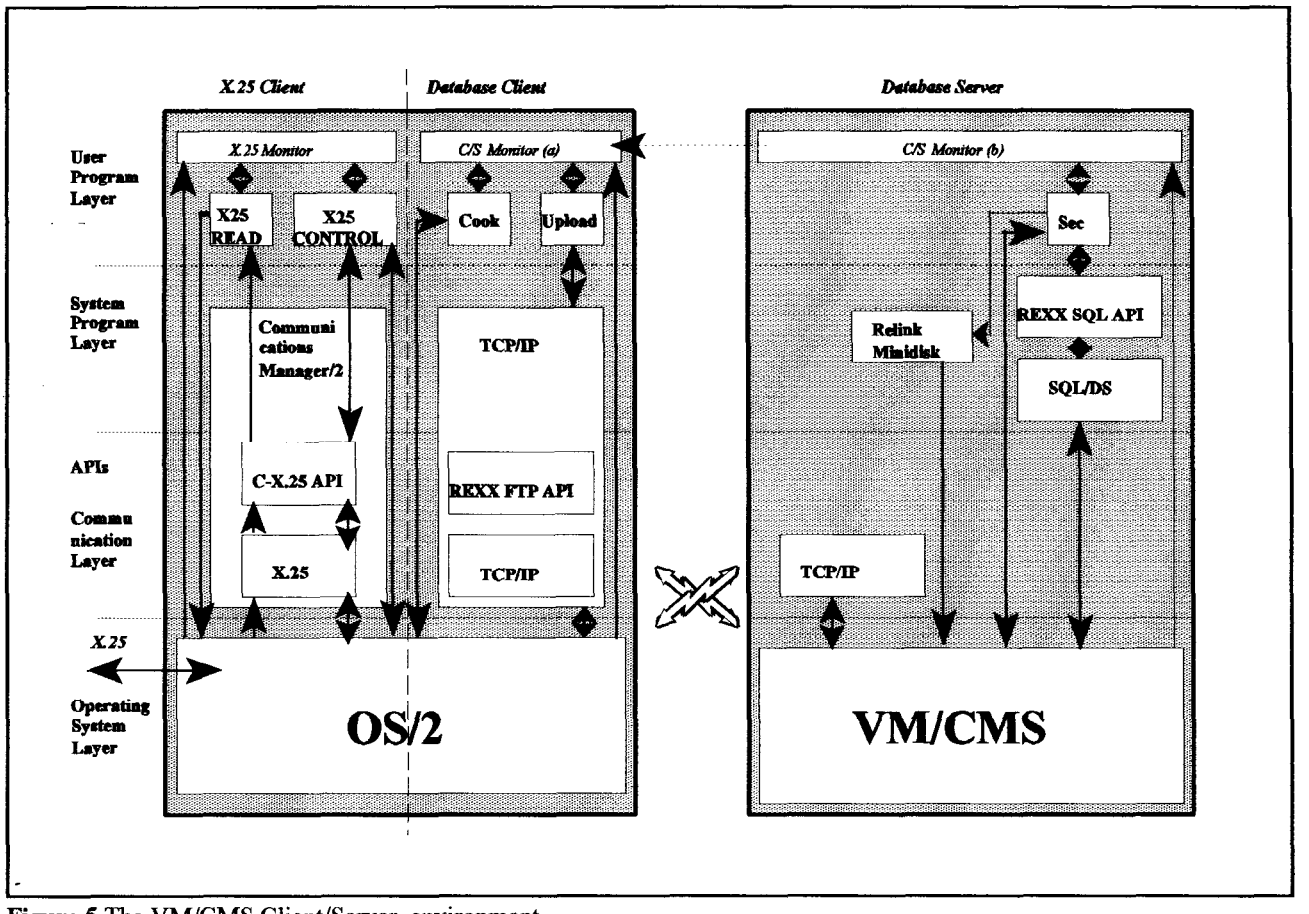


Figure 5 The VM/CMS Client/Server environment.

3.3 The VM-C/S environment

Figure 5 shows the architecture of the VM C/S environment. The data flows distinguished are explained briefly. The X.25 data packets are received by X25READ (arrows up) and written to a file (arrow down to OS/2). X25CONTROL checks this file on validity (packets complete?), consistency (right sequence?) and completeness (all packets received so far?). If not the case retransmission of one or a range of data packets is requested (arrow down) and, when received (arrow up), written to a temporary file (arrow down). The temporary file appended to the first file, is written to a complete data packets file by X25CONTROL (arrows up and down). Reading the complete data packets file (arrow up), Cook decides which lines read have to be converted to BeursBase data model formats (SQL). Lines containing only a synchronizing timestamp, are not used directly as we will see later. Each SQL statement generated is put in a separate file (arrow down), preceded by the parsed aebfcode, timestamp created and some C/S control parameters. Next, Upload sends new files (arrow up) to the host (arrow down) through a FTP connection (TCP/IP). At the host, the SQL statements executor, Sec, sequentially reads received files (arrow up) and first creates a SQL statement based on the parameters (the aebfcode plus timestamp) found at the first line. The result of execution of this SQL DML (table SELECT) statement (our unique fcode) is combined with the SQL statement, containing the actual SQL DML operation (table INSERT, UPDATE or DELETE), found in the file. This SQL statement is subsequently executed into BeursBase (arrow down).

For a continuous autonomous environment, VM file management is quite different compared with other operating systems like UNIX, OS/2 and MS-DOS, a VM file limitation had to be overcome. In contrast with a hierarchical file structure used by most of the operating systems, VM uses a flat file structure. Files are stored with the format <filename, 1 to 8 characters> <file type, 1 to 8 characters> <file mode, 1 character plus 1 digit between 1 and 6> on a so-called virtual minidisk. Minidisks are mapped onto physical storage devices (DASDs). To be used by a program a virtual minidisk has to be linked physically (by a CP LINK command) and has to be logically attached (by a CMS ACCESS command). VM allows multiple links, both in exclusive or shared read and/or write modes. Because links to minidisks are static links, file operations by one program are not 'seen' by another program. Thus, to establish a real-time VM C/S environment, Sec has to refresh its link with the minidisk where Upload writes the SQL DML data files to. This is done every time Sec does not find the next file in sequence (file type = number). To avoid the probability that Sec updates the link to the Upload minidisk continuously, Sec pauses a few seconds when after relinking no new files are found.
Every program mentioned writes a status file to be read by one of the two monitor programs, X.25 monitor and VM C/S monitor. To establish a C/S environment, the VM C/S monitor needs information about Sec. Since it cannot read directly the Sec status file, it has to be frequently downloaded by Upload. In addition , Upload reads the Sec status file frequently too, in order to determine successful execution of SQL statements, and to delete corresponding files by issuing remote (FTP) deletes accordingly.

3.4 The AIX-C/S environment

At a first sight the architecture of the AIX C/S environment shown in figure 6 is almost identical to its VM C/S counterpart. However, what seem 'minor' differences in design, with respect to the VM C/S environment, results in tremendous improvements. First, because a standardized C/S interface (Client Application Enabler/2) is applied, Upload is dropped. In addition, because Sec executes the SQL DML statements from the PS/2 (client), no special file operations (minidisk link refresh) are needed. Thus, this C/S environment gives advantages from both control and integration perspectives.
At the one hand, better control (effectiveness) because all status (file) information is directly available to the C/S monitor program, without necessary tricks. On the other hand, integration of functions is enhanced (efficiency) because programs involved run concurrently on one computer.

In the next paragraph we discuss the C/S architectures in greater detail, especially with respect to REXX and its interfaces used in the C/S environment.

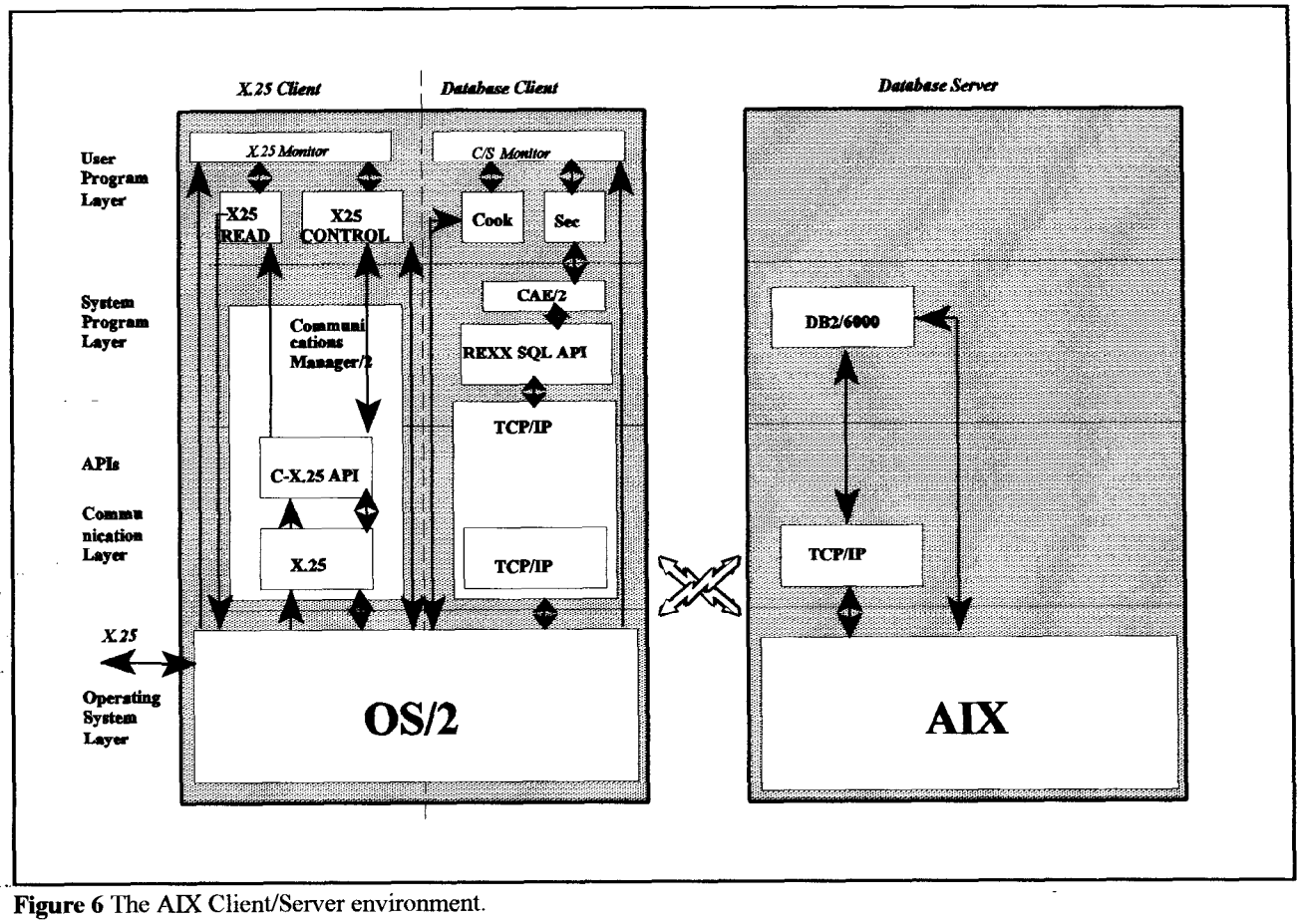


Figure 6 The AIX Client/Server environment.

4 REXX and the interfaces used in the VM and AIX C/S environments

4.1 Introduction

The basis of this paper is to elaborate on the impressive role REXX performed in development and implementation of the VM and AIX C/S environments. In particular, programming techniques, tips and tricks applied form the main subject for this paragraph.

Basically, the answer on the question why REXX forms the core in the development of the VM and AIX C/S environment, is visualized in figure 7. This figure shows that REXX, compared to other programming languages and development environments, is exceptional with regard to supported interfaces. True, REXX is not the exclusive language having so many different interfaces, in this respect for example is C of equal quality, though it's the ease of us which makes REXX unique. In addition, the fact that REXX programs cab be both interpreted and compiled, makes REXX special.
For each interface shown in figure 7 some general and REXX programming concepts applied are discussed.
Unfortunately, to avoid lengthy discussions, only some glimpses are provided.

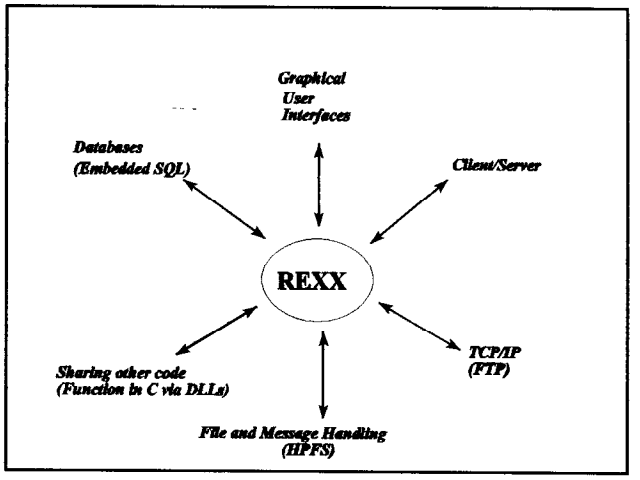


Figure 7 REXX interfaces.

4.2 REXX flexibility in C/S environments: designing for both performance and portability

Our experience is that REXX can be used very effectively in C/S development. One should be aware of the programming power which comes with REXX. Like any language one can benefit tremendously if one is cautious about performance and flexibility. First, develop REXX programs as universal, i.e. system-independent as possible for portability reasons. Secondly, do not make use of operating system specific functions, unless there is no alternative available. Thirdly, code well-documented, though as compact as possible. This is especially true if one does not use compiled code. Fourthly, when necessary and if possible, test programs on different hardware as soon as one can. Don't wait until there's no way back.

4.3 REXX and embedded SQL: the REXX-SQL interface

Aside from some exceptions, most of the administrative information systems in the economics discipline, especially in the business environment, are characterized by only a few fundamental functions. These comprise data storage, data manipulation and information retrieval and presentation. Since its introduction as a general purpose query language, its popularity is growing. Nowadays use of SQL, an acronym for Structured Query Language, is widespread. For years now, SQL has a solid place in the FEWEC IS curriculum. Its adoption in the C/S development was inevitable.

The REXX-SQL programming interface is available for all IBM database management systems (SQL/DS, and several DB2 versions). Despite some minor differences the REXX SQL programming interface is implemented uniformly for all database management systems. The usage of the REXX-SQL programming interface in the C/S environment is explained by the following example. The example shows how SQL database manipulation language (DML) statements are generated by an data-SQL converter, which we called ***Cook***, and how the are interpreted by ***Sec***.

Each line from the checked data packets file is examined by Cook. If Cook finds relevant data, it 'cooks' the corresponding SQL DML statement, based on the specified action on the data line. Lines not of interest should not be converted into SQL DML statements, though there is one exception: Cook does not use lines containing control data consisting of synchronizing timestamp for the X.25 connection. However, lines containing ASE trade volumes data do not have a timestamp. In that specific case Cook uses the timestamp found in previous line processed, which sometimes contain such control data.

```
1 1887ANFA36058NL0000360584     8104811 NLG2      4040            55500
3  864AOFC22311NL0000223113      220000    2
1 1888AOFC37750NL0000377505       19655   15
2  783ANFA00208NL0000002087     8104824 NLG2     10040
1 1889ANFA00921NL0000009215    53104826 NLG2      6000           245136
1 1890ANFA34948NL0000349488    30104826 NLG2      1680           975360
8 1665AQFL34948NL0000349488      104828 NLG2      1680           19800          1690          103800
```

Example 1a Small portion of the checked data file.

The body of Cook is a huge four-level select, which corresponds with four alphanumeric characters found on positions 8 to 11 of the parsed data line. The *parse* command, like *select* another powerful REXX feature, has been applied in Cook many times. This increase flexibility and maintainability significantly, in contrast with direct (static) usage of the BeursBase data definitions. The last line of example 1a shows the code **'ANFA'**, which stands for **'ASE'**, **'Price'**, **'Stock'**, **'New'**, meaning that ***at the Amsterdam Stock Exchange a new price for a stock transaction*** has been established. Based on the ARTEMIS data dictionary Cook builds a SQL DML INSERT statement for the NOTERINGEN table (example 1b).

```
1995-02-20-16.39.13 O 00173 0
INSERT INTO V67CVPOS.NOTERINGEN VALUES (:fcode, '1995-02-20-16.39.13',    5, 'K', '0', 99.9, 'H', ' ', ' ')
```

Example 1b SQL DML: INSERT statement generated by Cook preceded by timestamp, C/S control parameter 1, ASE fcode and C/S control parameter 2.

The SQL DML statement is preceded by the ASE timestamp, a C/S control parameter, the ASE stock code (aebfcode), and a second C/S control parameter. Only the ASE parameters were found in the raw data file line, the other two parameters were added by Cook. In the SQL DML statement the host variable **:fcode** is put in place of the ASE stock code, because the ASE stock code is not unique over time. First Sec parses the SQL DML statement from the file which is preceded by the creation timestamp, a C/S control parameter, the ASE stock code (aebfcode), and a second C/S control parameter found in the raw data file. Then Sec generates a SQL DML statement to retrieve the unique FEWEC stock code using the ASE stock code and the creation timestamp (example 1c).

```
command = "SELECT fcode FROM" InstrumentenTabel,
  "WHERE aebfcode = :oldfcode",
  "AND ts_intro  <= :timestamp",
  "AND ts_extro  IS NULL"
SQLS = 'SQLGETOLD'
CALL PrepareS
```

Example 1c SQL DML: SELECT statement generated by Sec to search for the unique FEWEC fcode.

The characters preceding the SQL DML code are stripped away and the found FEWEC stock code is put in place of the host variable :fcode. Now this SQL DML statement is executed by Sec.
Stocks are characterized by both a moment of introduction and a moment of extroduction. The majority of stocks once introduced exist permanently, however, there is always a possibility that a stock may be extroduced. Stocks can be extroduced for many reasons. In case of a management buy-out, a stock split or a bankruptcy of the firm, trade is ended, and sometimes a new stock will be introduced. In contrast with stocks, bonds are always extroduced. ASEs policy is that after stock extroduction the stock code comes free and is re-usable. If one intends to store all stock prices ever listed, like in BeursBase, one has to introduce

```
command = "SELECT MAX(fcode) FROM" InstrumentenTabel
SQLS = 'SQLGETNEW'
CALL PrepareS
```

Example 1d SQL DML: SQL DML statement to retrieve the maximum FEWEC fcode.

instead a unique stock code. Thus every time ASE introduces a new stock accompanied with their 'unique' ASE stock code (aebfcode), we have to map this stock code to a time-independent one. A simple solution is to use the maximum FEWEC stock code found in the stock table, plus one (figure 1c). For identification purposes and to keep track of all stocks, one needs to store both stock codes as well as both dates of introduction and extroduction. If stocks are still tradable their extroduction remains empty, in SQL we set the data value to NULL.The generated unique fcode is subsequently used in a SQL DML INSERT statement to add the stock introduced to the INSTRUMENTEN table (example 1e). Identical to the SQL DML insert statements update and delete SQL DML statements are generated, except for the INSTRUMENTEN (securities) table from which a delete is not allowed.

Many additional remarks can be made, however, three essential are discussed. First, it is never the case that a stock price precedes stock announcements. Thus the introduction of a new stock (or a renewed stock, for example, due to a stock split) happens always before new stock prices arrive and therefore there exists always a unique FEWEC stock code (the SQL statement generated is an INSERT into the stocks table). Secondly, official stock prices send can be modified or even withdrawn. Thirdly, if the X.25 connection is closed in the evening, a final file is generated. In contrast with the other files uploaded, this file does not contain a SQL DML statement but an 'End-Of Day' message. This way Sec is notified that no more files will be send that day.

```
SQL1 =      "INSERT INTO" InstrumentenTabel "VALUES (",
    ":fcode,",                          /* FCODE              */   FEWEC 'unique' stock code
    ""fondsnaam",",                     /* FNAAM              */   ASE security name
    ""fondstype",",                     /* FTYPE              */   FEWEC security type (bond, stock, warrant, et cetera)
    ""timestamp",",                     /* TS_INTRO           */   ASE introduction date
    "NULL,",                            /* TS_EXTRO           */   ASE extroduction date
    ""isincode",",                      /* ISINCODE           */   Official International Standard Identification Number
    fondscode+0",",                     /* AEBFCODE           */   ASE stock code
    rentoperc",",                       /* RENTE              */   Interest (bonds)
    ""marktcode",",                     /* MARKT_CODE         */   Market code (official market, non-official market, et cetera)
    ""noteringswijze",",                /* NOTERING           */   Current price
    ""valnotering",",                   /* VAL_NOTERING       */   Valuta traded
    ""valafrekening",",                 /* VAL_AFREKENING     *    Valuta paid
    ""swaif",",                         /* FONDS_SW_AIF       */   ASE
    ""omzetcode",",                     /* OMZET_CODE         */   Amount traded code
    ""cdatum1",",                       /* CDATUM1            */   Coupon date 1 (bonds)
    ""cdatum2",",                       /* CDATUM2            */   Coupon date 2 (bonds)
    eenheid",",                         /* EENHEID            */   Number traded
    "0,",                               /* NOMINAAL           */   Nominal value
    ""fondstype2",",                    /* FONDSTYPE          */   ASE security type (bond, stock, warrant, et cetera)
    ""swvoorlopc",",                    /*                    */   Definition 1 in official ASE trade newspaper
    ""swdefopc",",                      /*                    */   Definition 2 in official ASE trade newspaper
    ""symbol",",                        /*                    */   Symbol
    ""kleinstecoupure",",               /*                    */   Smallest number of instrument available
    ""coupondate",",                    /*                    */   Dummy coupon date
    interestfrequentie",",              /*                    */   Interest frequency
    ""intereststart"",                  /*                    */   Interest start
    ")"
```

Example 1e SQL DML: INSERT statement from Cook regenerated by Sec with the unique FEWEC fcode.

4.4 REXX and GUIs

REXX on IBM platforms doesn't come with a sophisticated GUI. One reason one can think of lies in the diversity of systems, in terms of hardware and software to be supported. The GUI is the most hardware and software, i.e. operating system, dependent of all software components. For example, mainframes support primarily text-based character terminals. Personal computers work with the OS/2 GUI known as the Presentation Manager. AIX based RS/6000s use the widely accepted UNIX GUI Xwindows extended with OS/F Motif. Then making it even more complex some GUIs are supported on multiple operating systems for instance Xwindows is supported at the PS/2, the RS/6000, the S/390 and SP families. Finally other hardware manufacturers have adopted REXX onto their systems like SUN (Sparc), Hewlett Packard (HP xxxx) and Commodore (Amiga), for instance. IBMs strategy not to support a platforms wide REXX GUI is the only choice.
Thus for software (especially programming and development tools) manufacturers specializing merely one or two platforms REXX-supported GUIs can be profitable market. For PS/2s there are two REXX-based GUIs available. At the one hand there is VX-REXX marketed by Watcom, and VisPro REXX marketed by VisPro. The first REXX GUI FEWEC acquainted with, VX-REXX, was bought. Not to undervalue VisPro-REXX, FEWECs choice was not a poor one.

4.5 REXX and TCP/IP: REXX FTP API

In the VM C/S environment the REXX-FTP interface plays an vital role. Without this interface the quasi C/S environment could not be established with REXX. The REXX-FTP interface, written by several IBM employees, became available as freeware, add-on product of TCP/IP for OS/2, in 1993. The first author of this paper was first acquainted with this product in June 1994. After solving installation and operation problems, we acknowledgement several persons in the REXX and TCP/IP community, as the REXX-FTP interface now works excellent. As an elaboration on section 3.3, our utilization of the REXX-FTP interface is discussed next.
At the PS/2 Cook reads the file containing the X.25 received data packets. Other lines with data packets with synchronizing timestamps for the X.25 connection, are only processed for these timestamps. Cook converts each relevant line into a SQL DML statement and subsequently writes it to a OS/2 HPFS file with a naming convention of <yyyymmdd.#>, where # stands for a sequence number. When started Upload starts an FTP session with the VM host via the REXX-FTP interface with the ftpuser function. The core of Upload is a loop in which several actions are programmed. Upload pauses until new files have been 'cooked' (Cook), then uploads them individually to the host using the ftpput function. If successful uploaded, Upload deletes the local file. Another action within the Upload loop is the frequently download of the Sec status file using the ftpget function (table 3). The download frequency is set based on a fixed number of uploads. Every day the FTP connection is closed using the ftpclose function and Upload is ended. Early next day Upload is started again by the VM C/S Monitor. Cook (packets processed) pauses when the Upload (packets uploaded) delay exceeds a certain threshold (200 files). This is necessary because the number of files in a local directory is negatively correlated to Uploads speed. Analogous to local file deletion, Upload frequently erases remote files, successfully executed by Sec, using the ftpdelete function. For the VM C/S Monitor, Upload frequently writes its operational status to a local file (table 3).

Program name	Filename used	Description
X	PVC.raw*	all data packets received, but not checked on consistency, validity or sequence;
	PVC.status	a status file containing information about the PVCs. This information is displayed in the X.25 monitor;
*XX**	HERTRANS.tmp	a buffer for the retransmitted data packets;
	CHECKED.raw**	the complete checked set of data packets so far;
	CHECKED.status	a status file including information about the PVCs (number of retransmissions, total number per PVC) and the CHECKED.raw file offset which happens to be the same as its filesize). This information too is displayed in the X.25 monitor.
	yyyymmdd.CHECKED	at 11.00pm the CHECKED.raw file is renamed with the current date at the beginning of the filename.
Cook*	Cook.status	a status file containing information about Cook to be displayed in the C/S monitor;
	yyyymmdd.#***	every SQL DML statement written to a file starting with the date and ended by a sequence number (#).
	yyyymmdd.#S***	stop file
	yyyymmdd.Cook.Log	Dependent on the loglevel chosen, every relevant operation (reading CHECKED.raw, creating a SQL DML, filing the SQL DML) is logged into this file.
	yyyymmdd.Cook.Status	all information needed for the C/S monitor is written to this file.
Upload* (VM C/S!)	Upload.Status	a status file containing information about Upload to be displayed in the C/S monitor
	yyyymmdd.Upload.Log	every operation (FTP connect, FTP put, FTP get, connect status, current activity) is written to this file.
	yyyymmdd.Upload.Status	all information needed for the C/S monitor is written to this file.
Sec*	Sec.Status	a status file containing information about Sec to be displayed in the C/S monitor
	VM: yyyymmdd.Log AIX: yyyymmdd.Sec.Log	every SQL DML statement file read and executed (SQLCA) is written to this file.
	VM: yyyymmdd.Status AIX:yyyymmdd.Sec.Status	all information needed for the C/S monitor is written to this file.

Table 3 Some filenames used by the X.25 data link programs.
*,** and *** mean that the files referred to are used as input for the particular program.

4.6 REXX and X.25

Although it was not used in the development of the C/S environments, the X.25 interface to the OS/2 Communications Manager (CM/2) is a good example of the importance of REXX in communications. The CM./2 X.25 interface is provided for a variety of programming languages such as C, COBOL, FORTRAN, Assembler and REXX. The main reason not to use REXX for X.25 based communications is in order to control the X.25 data link in terms of priority scheduling and multithreading and to secure application performance, one would be better of with C. As will be discussed later in this paper, action is taken to take a more detailed look to port the C-based X.25 programs to REXX.

4.7 REXX and file handling

From maintainability and flexibility perspectives it is wise to develop REXX programs using High Performance File System (HPFS). HPFS, which comes standard with OS/2, should be preferred over the 8.3 character limitation of the DOS File Allocation Table file system (FAT). In our REXX-based programs we benefitted from the HPFS features which allows using long filenames

which can be as 255 characters long. In table 3 the most important filenames used in the X.25 data link are shown. In the development of the C/S environments several advantages of using HPFS over FAT were exploited.
First, if a system crash occurs HPFS file are almost always recovered. Secondly, it is preferable to use semantically sound filenames. With the 8.3 character FAT limitation it is impossible to name files appropriately.

4.8 REXX and C programming

In the development of the C/S environments, like REXX, C plays an important role too. As discussed earlier, C was used to develop the specific X.25 programs. In addition, some general unctions , implemented with C, were developed for C/S simulation and to overcome some REXX limitations. One of these general C-functions, xread, was to overcome the REXX file-sharing limitation. REXX programs have to find external functions in a so-called Dynamic Link Library, or DLL. Such a DLL has to be developed and compiled by C.
This paragraph is concluded with table 4 listing the major hardware and software components used in the development of the C/S environments. In summary, at the one hand REXX was partially used to develop some of the components required for a C/S environment (Cook, Upload, Sec). On the other hand, for those components not developed with REXX (X25READ, X25CONTROL, external functions), REXX was the 'glue' to integrate these components (X.25 and C/S Monitors) with the REXX-based components. This paragraph focussed on more technical aspects of the C/S environments developed, especially the role of REXX within the C/S environment.

Subsystem: / Components:	X.25 data link	New AIX Client/Server environment	Old VM Client/Server environment
Computer system	PS/2	RS/6000 Cluster (4 x 590)	ES/9021-580
Operating System	OS/2	AIX	VM CMS
RDBMS	DB2/2 (small tests)	DB2/6000	SQL/DS
Client/Server application	CAE/2 CAE/DOS (MS-Windows)		
Client/Server programming	REXX, C Set/++, WorkFrame	REXX	REXX
Communication	TCP/IP CM/2 (X.25)	TCP/IP	TCP/IP
API	C-X.25 API (CM/2)	REXX SQL API CAE/2 and CAE/DOS C-REXX API	REXX SQL API (RxSQL) REXX FTP API (TCP/IP) C-REXX API
Application programming:	VX-REXX C/S	Server (AIX): - APL2/6000 - QueryManager/ 6000 Client:(PS/2s): - Watcom VX-REXX C/S - APL2/2 (special) - VisualGen (4 GL) - VisualAge(OO) - DB2/2 QueryManager - Any ODBC Windows pack-age	CSP/370 QMF/370 VX-REXX C/S

Table 4 The software packages used for design of the C/S environments.

Once established a C/S environment, one can develop a broad range of user and system application programs using BeursBase. The next paragraph, using section 4.4 (REXX and GUIs) as a starting point, discusses the value of REXX for application development. We will elaborate on important aspects of information visualization, templates and object-orientation using REXX. The following applications are discussed: system applications (the X.25, VM C/S and AIX C/S monitors) and user area applications like the AIX version of VUPOS.

5 REXX User programs for the C/S environments

5.1 Database connectivity

In the AIX C/S environment, C/S concepts used are much more sophisticated. For instance, applications at the client, preferably GUI-based, are able to show more information about the C/S environment. In this way a user can be more actively involved. Figure 8a shows an object with user and database information. Depending on the authorization level a user is or is not allowed to modify the information displayed. The object in figure 8a is retrieved through object 8b, by a click on the 'Wijz' pushbutton. If the user is not allowed to do so, the pushbutton is set to not-clickable and its color is changed. If the information displayed is correct or modified, the user can click on pushbutton 'Con.' to establish a connection. In the case of a simple end-user an auto-connect is pursued. Status information about the necessary REXX DLLs as well as status information about the database connection.

5.2 Object Orientation and REXX

Figures 9a thru 9c exemplify a way how to define and use objects in VX-REXX. Figure 9a shows the complex object which consists of a listbox and four entry fields. The object used to display tabular or graphical information is showed after activating the 'Genereer grafiek' pushbutton in figure 8b. At first the object is shown as in figure 8a. Then, based on the user selection from the listbox., one or more properties are changed by message passing. If the user selects 'Real-time', no additional information is necessary to show the tabular of graphical information requested, because the system date can be used. In contrast, if 'Meerdere dagen' (= more than one day) is selected, begin and end date are required. Also if 'Meerdere dagdelen' is selected begin and end time is required too. Using a GUI this way keeps the user's eye focussed onto the display and does not provide the user irrelevant information.

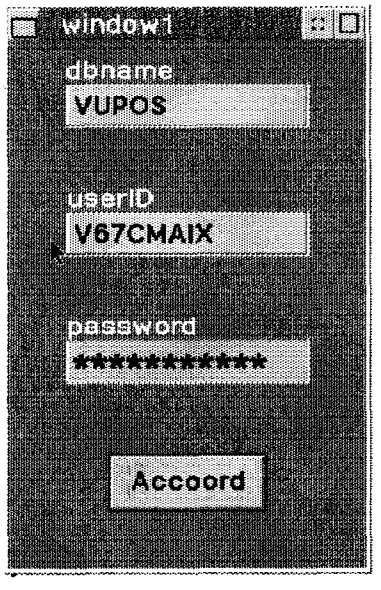


Figure 8a Database connect information.

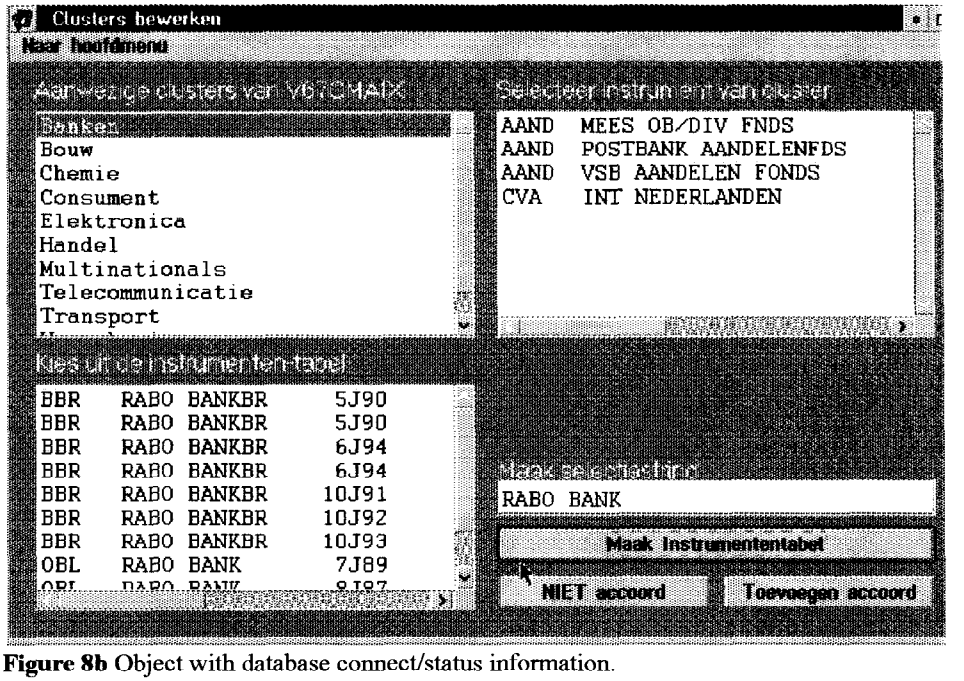


Figure 8b Object with database connect/status information.

Figure 9a Object with hidden entry fields.

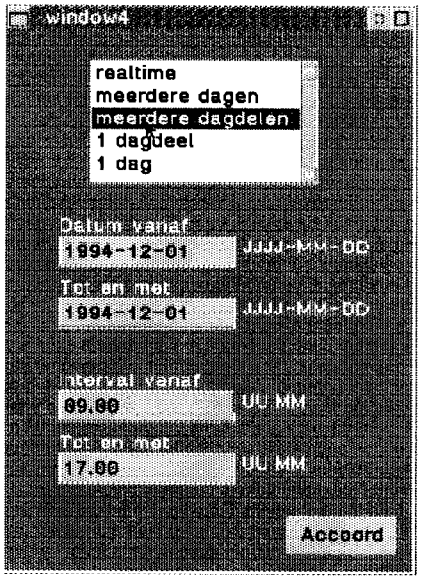


Figure 9b Object with all entry fields shown.

Figure 9c Object with some entry fields shown.

5.3 Developing general parts and templates

A sophisticated way of programming is the use of objects in the form of templates. Object orientation facilitates the use of general parts or templates. An example of a general part is shown in figure 10. This object is especially useful for debugging. Every SQL statement parsed is checked on its SQLCA. If the SQLCA is not equal to 0 (success) or 100 (no more rows certified the conditions specified), further program execution stops and the SQLCA object is displayed.
An earlier example of a general part was already discussed. Figure 8a, the object for displaying database information, is used in every application developed so far.

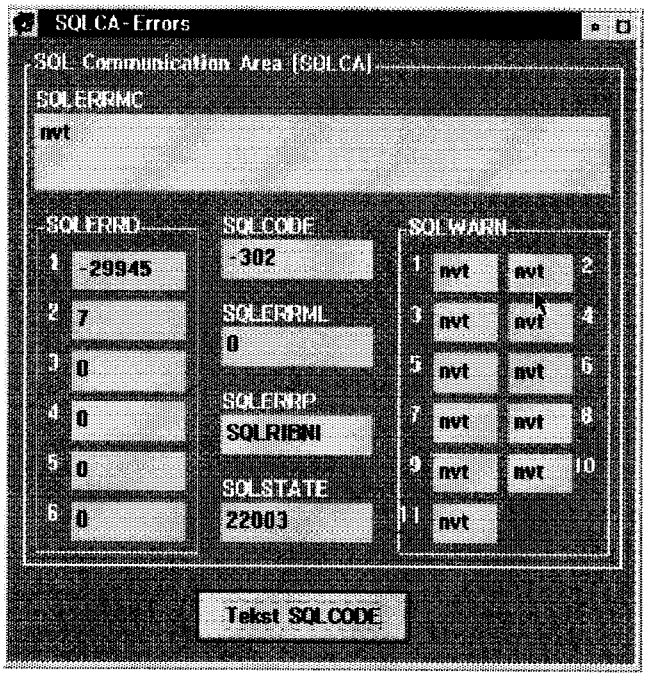


Figure 10 Object for presentation of SQL Communication Area Descriptor.

5.4 GUI design and usage: monitoring

The X.25 monitor
A monitor should be kept simple and should give direct information. The X.25 monitor, developed for control of the X.25 data link displays all the information necessary. Information includes the # of packets received, retransmitted, and the # of retransmissions per logical circuit (PVC group).

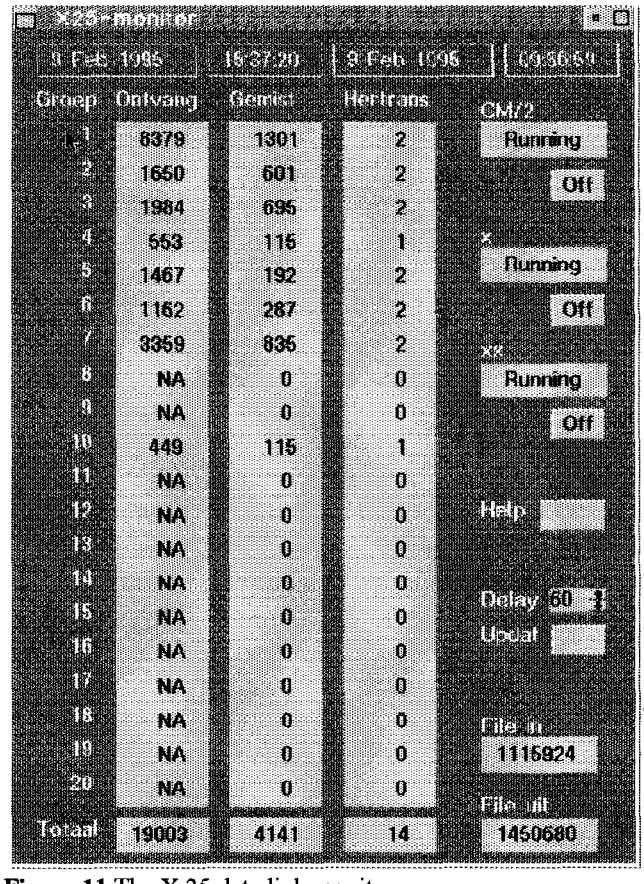


Figure 11 The X.25 data link monitor.

The X.25 and C/S monitors are design to be displayed simultaneously. The information of two monitors, X.25 and AIX C/S are integrated as follows: the difference between the number of received data packets (X.25) and the packets cooked (AIX C/S) is the first packets delay. The packets cooked means the number of SQL DML statements created. The difference between this number and the packets executed (AIX C/S) gives the second delay.

The VM C/S monitor

Notice that the VM C/S monitor is very different from its AIX C/S counterpart. Since the VM C/S environment is user written, information about the underlying program (upload) is displayed. The information of two monitors, X.25 and VM C/S are integrated as follows: the difference between the number of received data packets (X.25) and the packets cooked (VM C/S) is the first packets delay. The packets cooked means the amount of SQL DML statements (files) created. The difference between this number and the packets uploaded (VM C/S) results in the second delay.

In addition vital program information is provided. Finally the sizes of the input (unchecked) and output (checked) files, where the raw data (packets) are stored, is displayed.

Several features are added to the monitor. First, to be certain the screen update process does not consume too much time, a timer event updates the screen periodically. The time period can be set between 10 and 60 seconds in steps of 10 seconds. If an immediate screen update is required, one simply clicks the update pushbutton.

The AIX C/S monitor

Like the X.25 monitor the AIX C/S monitor provides direct information about the underlying C/S processes. As the AIX C/S environment supports full C/S, all the processing and programs can be run at the client. Information about the various stages of processing and programs is displayed. In contrast with the user applications the database information is integrated in the monitor window. The reason is to have all the information displayed in just one screen.

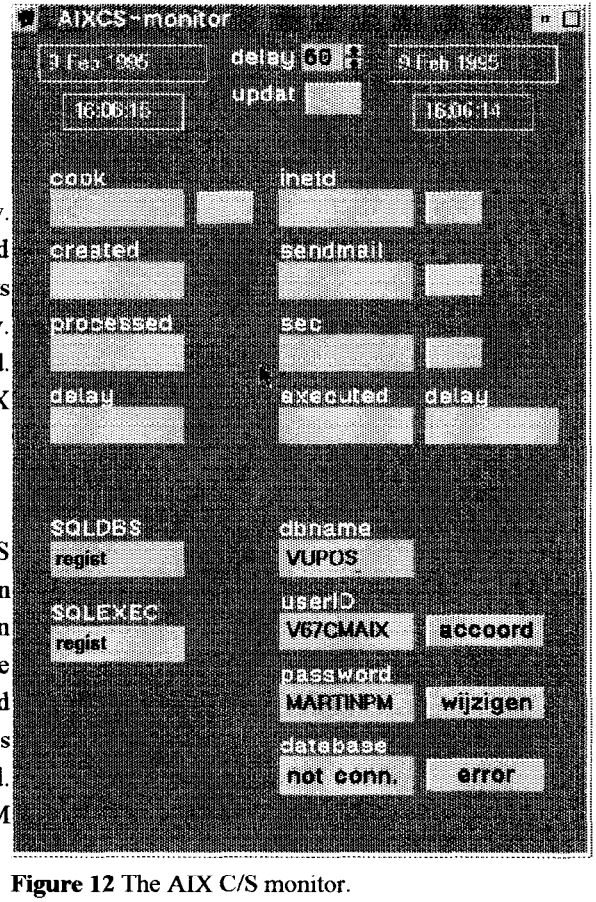


Figure 12 The AIX C/S monitor.

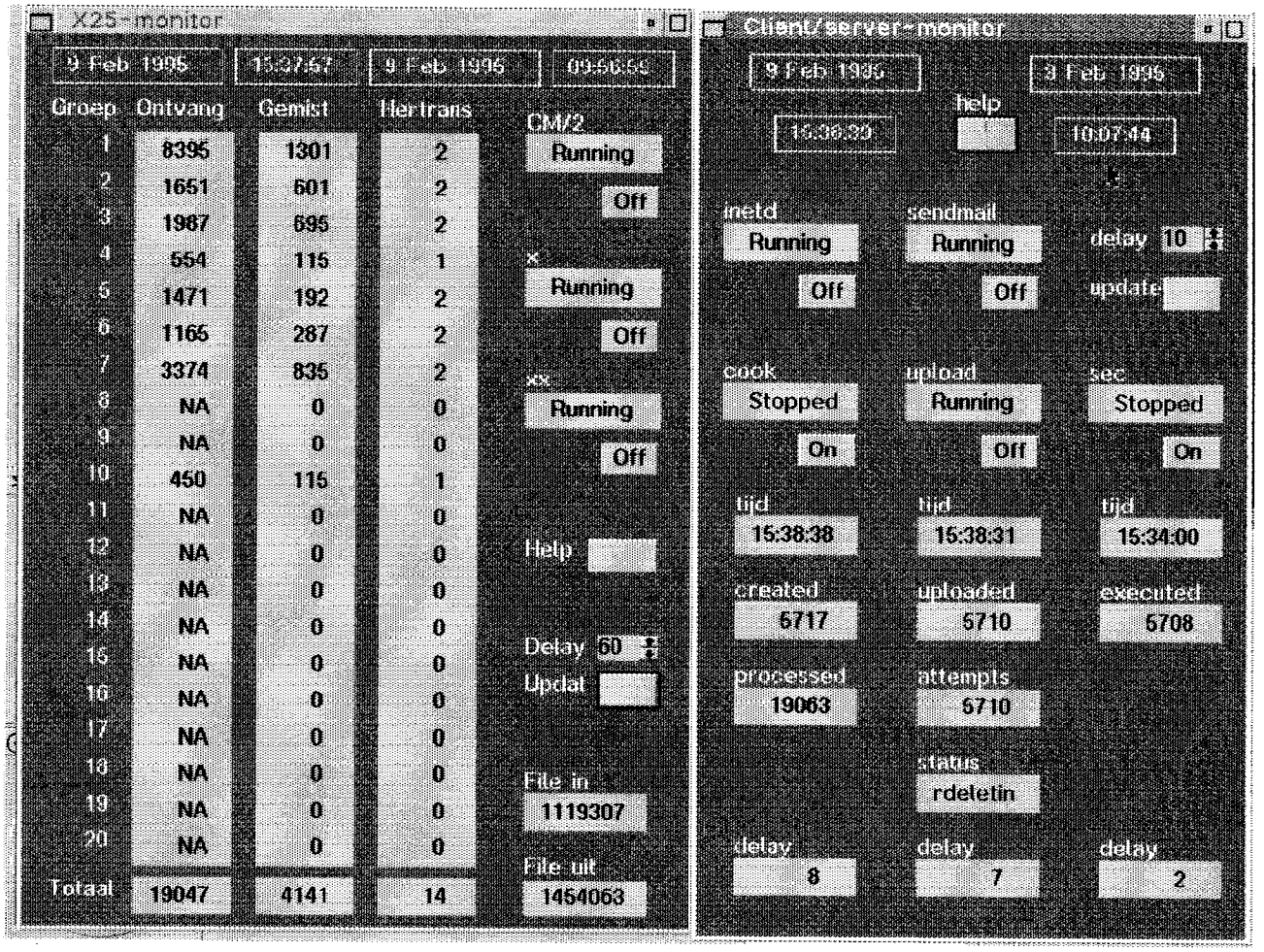


Figure 13 A full screen display of both the X.25 monitor (left) and the VM C/S monitor (right).

Finally, the difference between the packets uploaded (VM C/S) and packets executed (VM C/S) gives the third delay. In figure 13 both the X.25 monitor and VM C/S monitor are displayed.

5.5 Other graphical user-interface features

One of the issues that was ignored for a relatively long time, because the development of the C/S environments did have so many interesting research, design, implementation and usage issues that the project team almost forgot about a neat menu structure. Application of the menu structure is simple because much of the preparation for it has already been done. First, the framework for application development was designed long before. Secondly, database control and maintenance can be well-defined. Thirdly, every application program is characterized by some general functions which can easily be specified. Fourthly, the information generated can be visualized in a limited number of formats: on the computer display, on paper and on magnetic and optical media. The general menu structure to be designed should be a shell in which every application developed can have a place. Then based on user and program authorization, users can or cannot select the specific application program. The general purpose menu structure is shown in figure 14.

5.6 VX-REXX specific objects

Of the many special objects, the timer object is a very convenient one, especially if one designs and implements autonomous monitors. For control purposes, one has to have feedback frequently, and if necessary one has to maintain the C/S environment dynamically. Using timer objects one can frequently poll the environment (exception management) and retrieve information

(status files) to be displayed.

6 Future strategy of the C/S environments

6.1 Extending the matrix framework with (REXX) application programs

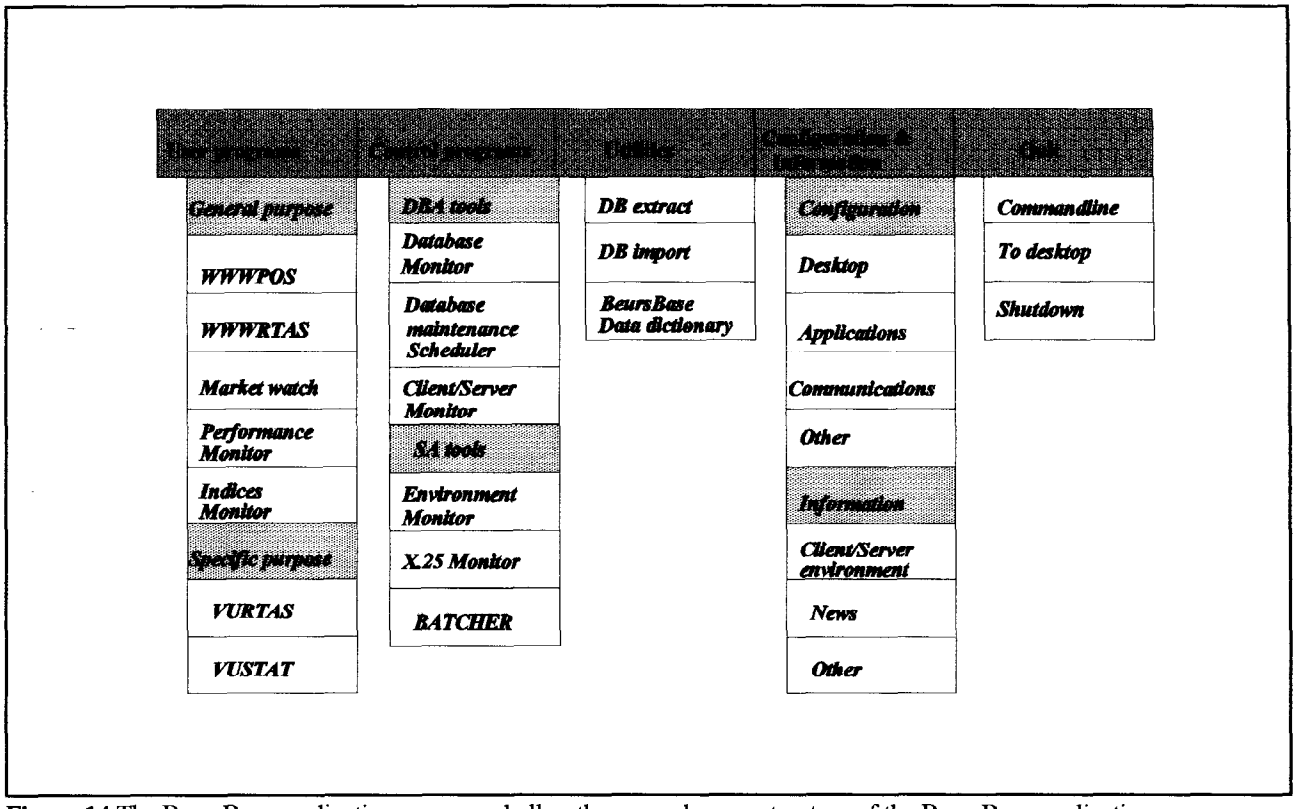


Figure 14 The BeursBase application program shell or the general menu structure of the BeursBase application programs.

VUPOS itself being ported from the mainframe to the personal computer. In contrast with the Cross Systems Product mainframe version, the personal computer version, implemented in VX-REXX is provided with a graphical interface. Compared with mainframe graphics using GDDM (CSP-ADMCHART interface), the VX-REXX graphical user interface is much more sophisticated. Though the most important advantage using a GUI painter is the significant reduction of development time. Especially for students it is very convenient to be able to generate usable output in a short time. In figure 15 an example of a simple XY-chart is shown. This XY-chart displays stock price development on December the first 1994 of the largest multinational of the Netherlands at the Amsterdam Stock Exchange known as '*Koninklijke Nederlandse Petroleum Maatschappij*' or '*Royal Dutch*'. To the general

public the multinational is known as Shell. A minimum of additional information is provided, like ASEs trading period ((9.30am to 4.30pm), the number of transactions found (159), and both maximum (188.60) and minimum (190.70) prices of the day. It still is an art and a science area to build good information displays. It is simple to expand the above example to a chart which shows, for example, the development of the portfolio of a user over a longer period of time.

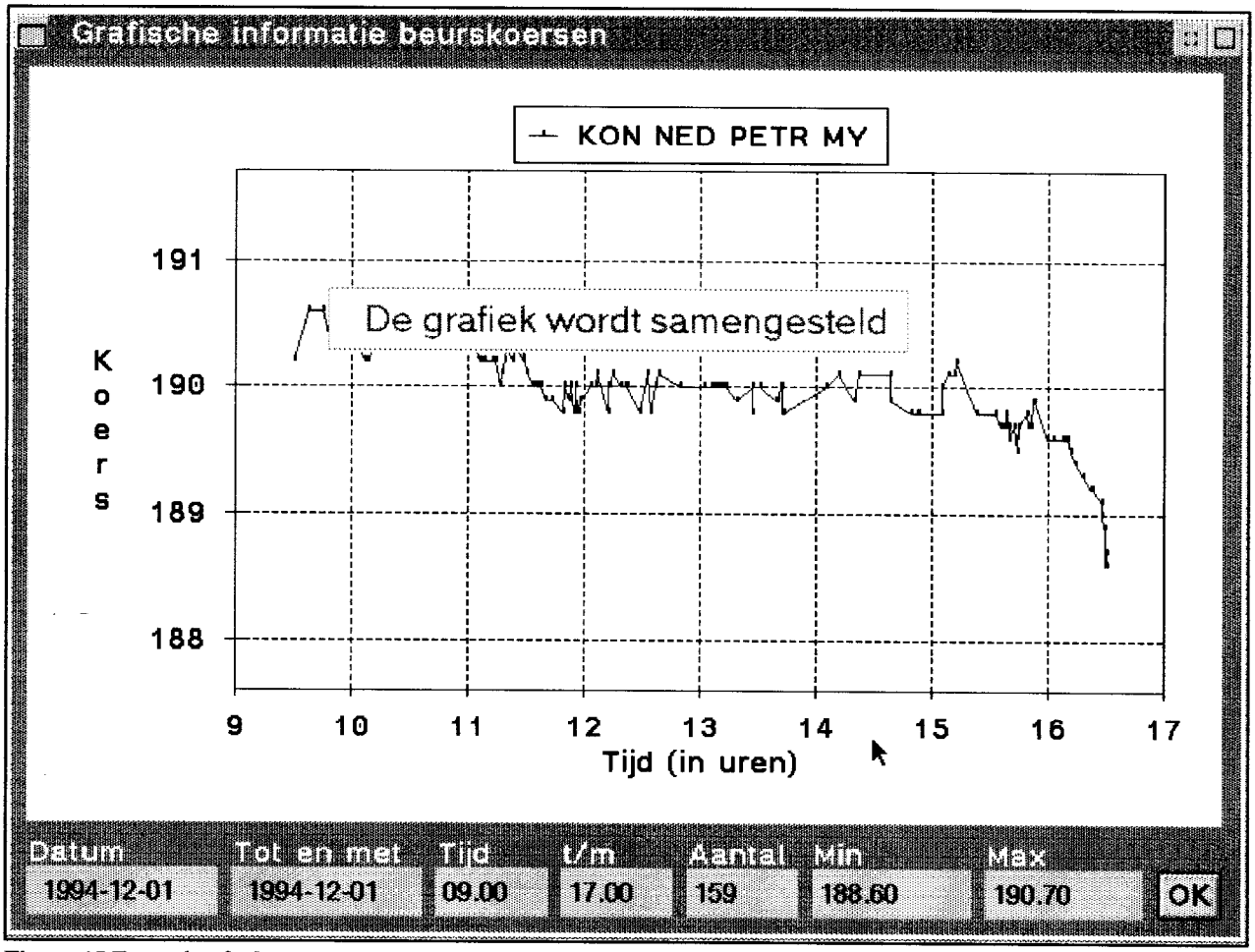


Figure 15 Example of a VUPOS screen.

6.2 Designing user (DBA) programs for database monitoring and control

As the number of users is increasing rapidly as well as the set of user application programs, the need for control and maintenance structures emerges. Though kept in mind this was set at a low priority. The following kinds of control are desired:

Authorization control: users and application programs

There are several possibilities of how to deal with multiple users and multiple programs. First, in C/S environments authorization can be placed at the user level. In this case every user obtains a userid. For maintenance purposes not very attractive, since it entails much administration (users) and a lot of authorization (programs). Secondly, as an alternative authorization can be placed at the program level. In that case every program is provided with a userid, or better, program id. Now administration

is reduced and authorization is simplified tremendously. It should be clear that in this situation program users do not have direct access to the database directly. In practice, a combination of userids and program ids is used. This requires sophisticated control. Up to now this has been done manually. An appropriate tool is under development.

Database control: database performance and tuning

BeursBase grows on a real-time basis. At the one hand Exchange data is inserted continuously, and at the other hand users and user programs add data as well. The largest tables in BeursBase contain hunderds of thousands of tuples and will rapidly grow up to tens of millions. It is not the physical size of the data that limits database operations, the gigabytes range will not be reached for years, but the performance and optimization issues. Reorganizing dbspaces, tables and indexes form a burden on database operations and database efficiency. This makes BeursBase so interesting for IS research. Therefore developments for several applications for performance measurement and database tuning have been taken.

Database maintenance

Usually database management systems are not provided with user friendly maintenance programs. Especially for BeursBase, database maintenance issues are of vital importance. Though there are lots of ideas on this topic, the have not yet been implemented in practice.

6.3 Improving the X.25 data link

A point of weakness in the BeursBase project remains the X.25 data link, because only one single communication is in use. In order to minimize the risk of data loss a secondary X.25 data link is necessary. Figure 16 shows the ideal implementation. Because ASE has two separate PTT Telecom (the national telecom operator) nodes. Therefore FEWEC can be provided with two separate X.25 links. Although the SARA link is implemented with a single line, no data will be lost when this communication line is goes down. For almost 100 percent reliable X.25 data link with the ASE, the following actions have to be taken.

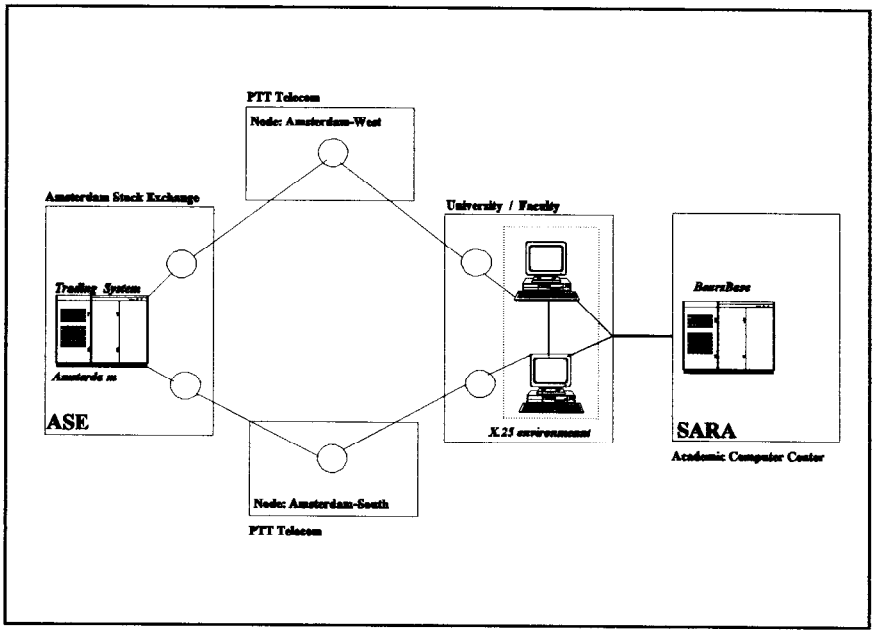


Figure 16 Two independent X.25 environments.

- Configuration of a shadow or backup X.25 connection, consisting of:
  - a second PS/2 with an X.25 Coprocessor expansion card;
  - a second ASE communication link, physically independent from the first one. This means that this X.25 link uses different nodes. At the ASE multiple independent nodes are available (Figure 16);
  - between FEWEC and the ASE, a second 19.2 Kbps PTT leased line;
  - An uninterruptable power supply (UPS) for the two X.25 PS/2s;

- A REXX-based control structure, as applied on FEWEC servers will be installed on both X.25 PS/2. This intelligent mechanism, a two PS/2s based monitoring system, should take appropriate action if one or more connections are lost;

- A hardware mechanism will be constructed to be able to cold-boot the PS/2 remotely. If the responsible system operator is not at FEWEC s/he should be able to reboot the system by a remote connection. (If one cannot telnet to the PS/2, it should be possible by connection via one of the FEWEC servers). Using a cold-boot procedure suffices, as the systems can start necessary processes automatically.

In the first half year of 1995 these investments are planned. In addition, several application programs are needed to create autonomous control between X.25 PS/2s and the FEWEC LAN servers. Two identical REXX-based programs, running at both X.25 PS/2s, have to check each other operations (file sizes and contents of the status files). If needed the C/S data link to BeursBase has to be changed, from one PS/2 to the other, without any delay and appropriate action has to be taken to get both PS/2s in operation again. Another program, residing at the FEWEC LAN server to check frequently if both X.25 PS/2 are running. Again, if some exception is found, appropriate action has to be taken. If, for some reason , one or more systems do not response, say within 15 minutes or so, FEWECs computer support staff is notified by email or screen messages.

6.4 Moving to the third and higher Client/Server levels

Eventually the AIX C/S environment has to move to the third, fourth and even fifth C/S level. Why? During the development processes of our C/S environments we encountered many performance problems. The larger tables where used by many programs and even more users concurrently. At the same time these tables are maintained (inserted, updated, deleted) on a real-time basis, resulting in many (dead) lock situations.

Looking over and over at the data, it was decided to split them into several time categories, based on their anticipated usage (figure 17). The first category, real-time (today's) data, is stored in a separate table, as this data has the highest priority. The second category, this years data, is accumulated into a second table. This data too is used very frequently (its size spans obviously maximally a year). The third category contains data from yesterday to a year before. The fourth and last category is historical data and is actualized to the end of last year. For performance reasons the last two tables are stored in dbspaces without leaving space free and records are stored by stock by timestamp. The first two tables are stored in dbspaces with small data and index buffers. Every night except for the weekend, the following batch processing is done: the today table is emptied in this year's table, also, at the end of December the last year table is emptied into the history table. A more intelligent solution would be a dynamically calculated optimal database performance with a minimum of data redundancy. Such an system area application program will be developed this year.

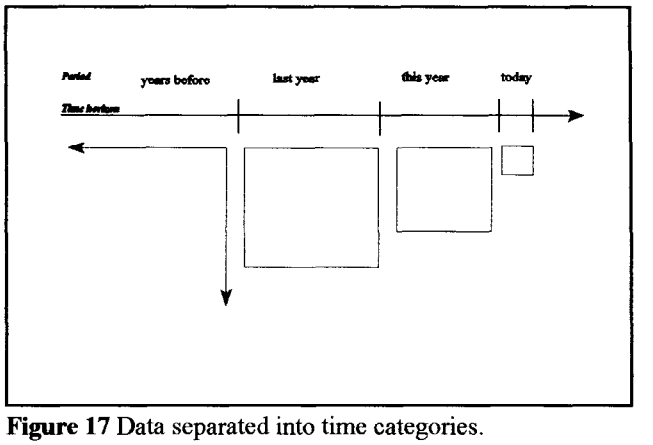


Figure 17 Data separated into time categories.

Moving to the third Client/Server level

Now for the C/S aspects one can imagine that the small real-time data table (today) used at a local database server would be significantly faster than a host database. So the next phase in the development of the AIX C/S environment is to establish a local database server for real-time data (IBMs DB2/2 for OS/2?). This will increase both application and database performance

for user applications like VUPOS tremendously. In addition, a local database server is more suitably equipped to support batch processing with the large database server as remote host.

Moving further to the fourth and fifth Client/Server levels

The third C/S level is relatively simple to reach, though should thoughtfully be implemented. User applications which need data for processing which is not available at the local server have to send requests to the remote database server. At least two problems, i.e. bottle necks are obvious: first if the amount of data is relatively large, database I/O is heavy and all data requested has to be send over the communication line which consumes much time. Secondly, local client processing is, compared to an AIX or VM host, though cheaper relatively slow. To overcome these bottlenecks, for some user applications it would be interesting to split up applications in several modules. Then these modules can be distributed across several systems, based on the CPU and database performance required. For example the portfolio simulation VUPOS or a technical analysis application has an econometrics/statistical feature which calculates several stock performance indicators over a certain time period (from actual real-time to historical data). Partially, the indicators requested are calculated at the client using the local database server (real-time) data, partially the indicators are calculated at the host using the host database server (historical) data. Finally the client program uses some algorithm to glue the partial outcomes into meaningful information. Even if the host requires real-time data (which is stored at the local database server) in total the results are generated much faster compared to the alternative in which the client does all the processing. Clearly this is another interesting area for research and education. The fourth and fifth C/S levels are within reach, though it will take at least two years before users can benefit from these ideas.

6.5 Speeding up REXX programs More extensive utilization of REXX compilers

Much can be written on the subject of REXX in terms of interpretation versus compilation. For the most part in the development of the C/S environments REXX is used by interpretation, except for some application programs which have been written in VX-REXX. At the point where speed is of primal importance, up to this moment the choice has been for programming in C. Using C as programming language implies using compiled programs. The main reasons for this choice are the flexibility of C, the clear X.25-C interface as well as the robustness and performance of C programs once they are compiled into executable code. An advantage in C too is the easiness with which multithreaded high priority programs can be developed. In addition there was no experience with the combination of X.25 and REXX or with REXX compilers which can generate fast executable code.

For manageability and maintenance reasons it would be wise to move from C to REXX. Currently research is conducted to take this step forward. Moving from C to REXX is necessary because the experience with C programming is and should be limited at FEWEC. The higher the programming, or more appropriate development, level the better. Thus one of the major concerns is to make more extensively use of REXX under the constraints that X.25 support, priority scheduling and compiling are supported and easy to implement.

6.6 Sophisticated application programs: a World Wide Web future?

Though a detailed discussion about user applications should not be included in this paper, one serious idea is worth mentioning, namely a portfolio simulation accessible through World Wide Web. As already discussed, the VUPOS portfolio simulation is being redesigned for the AIX C/S environment. This application will be used for all the user areas defined by several of the identified disciplines. For several reasons is the usage of VUPOS restricted to FEWEC members and students. First the application is relatively complex and one has to have detailed knowledge of the underlying processes. For students of economics and FEWEC members this should be no problem. If FEWEC intends to use VUPOS as a public relations tool and make it

available to others, for example high schools or other interesting populations, this will cause several difficulties. First, as said VUPOS is a complex program. Secondly, it takes several screens to manage ones portfolio. Thirdly, processing is done interactively which makes very large to huge scale application very costly and performance dramatically slow. Fourthly, a high-end OS/2 computer is highly recommended. Such systems are not so widespread in use.

Based on the rapid adoption of Internet, granted by the development of global information systems like World Wide Web, it is realistic to conclude that high schools and other interesting populations are able to connect to Internet without having to pay insurmountable costs. Thus a realistic alternative would be a stripped version of VUPOS, reduced to three to five screens, supporting batch processing once or twice every day. This simplified or stripped version of VUPOS should be provided with a World Wide Web interface. Installed at the FEWEC WWW server anyone, or if FEWEC so wishes, a selected audience will be able to use the program in, for example, a large stock investment competition. The investment results of the competition as well as the system usage itself can be input for research and education.
Anyway FEWEC is currently developing in cooperation with the ASE a WWW service which includes a real-time graphical display of the AEX, the index of both Exchanges in The Netherlands. We consider this as a first step towards an Internet based simulation, because this WWW facility will be used to research several interfaces, forms and displays for such a simulation.

6.7 Comparative application development

Once one has an AIX C/S environment as described in this paper, it is very interesting to examine and compare various development environments are available for C/S application development. Today numerous development environments available. This year we will use for example APL2/2 and VisualAge (both IBM), and CASE environments like ADW from KnowledgeWare and IEF from Texas Instruments/James Martin. Probably programming languages like C, C++, Smalltalk (via VisualAge) and COBOL will be tested too with respect to the trade-off (compared to the 4GL development environments) between an increased development time and run-time performance.

7 General conclusions and recommendations

7.1 Advantages of using REXX in a Client/Server environment

The project team of FEWEC concludes, based on the experiences so far, that the REXX programming language is, in particular in combination with the interfaces discussed in this paper, suited to develop not only C/S programs but a C/S environment itself too. Even without having much experience in developing C/S environments, implementing a REXX-based C/S was relatively simple. The project team benefitted from REXXs flexibility and portability as an AIX C/S environment was easily developed using code from the VM C/S environment.
Especially in the design and implementation stages it is advantageous to have an interpreted language in stead of having to compile code every time changes have been made to it. Finally, it has been proved that REXX user application programs can be gorgeous from the outside, efficient from within, and effectively fast in general without being to complex. This can hardly be said of programs developed in for example C or C++. Therefore in the IS curriculum of FEWEC REXX is preferred over C or C++.

7.2 Problems using REXX in a Client/Server environment

To temper the over enthusiastic mood of the project team, some problems or vague issues have to be dealt with too. Most of the problems encountered are probably heard before, but one cannot overemphasize their importance. Some REXX interfaces lack sufficient documentation, examples and example code. In the view of the project team every time the interface is used, the wheel is re-invented. Though there are for example many good REXX books available, most of them don't go beyond the introductory level. Also, voluminous manuals are not necessarily of high quality. Especially the REXX FTP, REXX SQL and REXX X.25 interfaces can gain popularity when documentation is extended.

Secondly, problems occur when one needs to share files between REXX programs in OS/2. Files in OS/2 REXX are used exclusively thus they cannot be shared among application programs. A solution was found in developing a C-based DLL function, as we found that it was possible to share the file this way. Using the C-function, OS/2 still results in DOS read errors, although it works fine. If one lacks the specific experience of writing DLLs, it is a burden to find out how things are being done. Ideally, a REXX compiler should be equipped with a tool enabling developer (student), without low-level programming, to put functions to be shared in a DLL.

Thirdly, more effort should be put in REXX benchmarking and tool evaluation. The project team encountered difficulties in choosing the 'right' REXX development tool and interface. Not usage of a REXX development was the problem, but which environment should be used. Fortunately, talking in hindsight acceptable choices were made. Fourthly, sharing information using status files is not the most sophisticated way of communication. However, no neat alternative was available. Especially the issue of file corruption and system crashes makes the usage of files in interprogram communication volatile.

7.3 Overall conclusions

During the last two years development of the VM and AIX C/S environments was successful. Both systems are operational and future plans and strategy promise to generate a massively used AIX C/S environment, high quality applications and a sound scientific environment for research, educational and public relation purposes. Though the power of REXX has already been proved in the current environment, as the AIX C/S environment moves further towards higher levels of C/S, REXXs position as an advanced C/S development language and environment shall be indisputable.

7.4 Recommendations

Finally, based on the REXX experiences, the project team comes with a few practical recommendations for developers, manufacturers and users as well. First, everyone wants better application performance. Though some other issues are of equal importance. One of the severe limitations of OS/2 REXX is that files cannot be shared among application programs. Working around the problem is not the preferred solution. In an advanced environment as OS/2, file sharing between OS/2 REXX application programs cannot be missed. Secondly, REXX still lacks some sophisticated interface toolboxes, especially for general and C/S specific monitoring. Thirdly, however fully accepted in CASE development, REXX code re-usability, re-engineering and a central repository, are far from reality. This definitely has to change if REXX is to be used in large scale application development. Fourthly, the basis for REXX-based GUIs has been set a few years ago, REXX development tools can be extended with more specific and more appropriate GUI features. Finally, as a fifth recommendation one should develop more detailed REXX programming handbooks with realistic examples.

8 References

[ASE94] Amsterdam Stock Exchange (1994), ***Amsterdam Real-Time Market Information System (ARTEMIS): user manual***, version 3.0 (in Dutch), Beursdata B.V., Amsterdam.

[Buys93] Buys, E.O (1992), ***TRANSPAS:Tthe Next Generation*** (in dutch), Graduation project, Free Univeristy, Amsterdam.

[Cowl84] Cowlishaw, M. (1984), ***The design of the REXX Language***, IBM Systems Journal, vol. 23, no. 4, pp. 326-335, IBM, New Jersey.

[Cowl90] Cowlishaw, M. (1990), ***The REXX Language***, 2nd edition, Prentice Hall International, Englewood Cliffs, New Jersey.

[Date92] Date, C.J. (1992), ***An introduction to Database Systems***, volume I, 6th edition, Addison-Wesley Publishing Company, Reading, Massachusetts.

[Davi85] Davis G.B., M.H. Olsen (1985), ***Management Information Systems***, McGraw-Hill Book Company, New York.

[Deit90] Deitel, H. (1990), ***Operating Systems***, 2nd edition, Addison-Wesley Publishing Company, Reading, Massachusetts.

[Deit92] Deitel, H., M.S. Kogan (1992), ***The design of OS/2***, Addison-Wesley Publishing Company, Reading, Massachusetts.

[Enge95] Engel, J.P. (1995), *ARTEMIS: the Amsterdam Stock Exchange and electronic information services* (in dutch), Graduation project, Vrije Universiteit, Amsterdam.

[Germ94] H. German (1994), ***The REXX handbook: BASICS, APPLICATIONS and TIPS***, Van Nostrand Reinhold, New York.

[Güse95] Güse, L.W.M. (1995), ***BeursBase project: operation's manual*** (in dutch, in preparation), Free University, Amsterdam.

[Horn90] Horne, J.C. Van (1990), ***Financial Management and Policy***, 8th edition, Prentice Hall International, Englewood Cliffs, New Jersey.

[IBM92] IBM (1992), ***VM/System Product Interpreter SQL/Data System Interface, program description/operations manual***, New York.

[IBM93] IBM (1993), ***OS/2 REXX : From Bark to Byte,*** IBM International Technical Support Centers, Boca Raton Center.

[IBM94] IBM (1994), ***Client Application Enabler/2, User's Guide*** version 1.2, IBM Canada Ltd. Laboratory: Information Development, North York, Ontario.

[Miss94a] Misseyer, M.P. (1994), ***From Stockdata to real-time Exchange Information***, in: Landelijk BIK blad, vol. 1, no. 1, pp. 25-27, Amsterdam.

[Miss95] Misseyer, M.P. et. al. (1995), ***TRANSPAS is death: Long live VUPOS and BeursBase***, Project proposal of VUPOS and BeursBase, Vrije Universiteit, Amsterdam.

[Mors94] Morsink, A.W. (1994), *Receiving, transforming, converting and adding real-time Amsterdam Stock Exchange data to BeursBase* (in dutch), Graduation project, Vrije Universiteit, Amsterdam.

[Orfa93] Orfali, R., D. Harkey (1993), ***Client/Server programming with OS/2 2.1***, 3rd edition, Van Nostrand reinhold, New York.

[Rudd94] Rudd, A.S. (1994), ***Application Development using OS/2 REXX***, Wiley-QED.

[Stal90] Stallings, W. (1990), *Business Data Communications*, MacMillan Publishing Company, New York.

[Tops94] Tops, I. (1994), *The Design and Implementation of a Real-Time Stock Exchange Simulation and Performance Monitoring System* (in dutch), Graduation project, Vrije Universiteit, Amsterdam.

[Turb95] Turban, E.F. (1995), ***Decision Support Systems and Expert Systems***, 4th edition, Prentice-Hall International, Englewood Cliffs, New Jersey.

REXX/370 Compiler and Library 1995

Pages 324-358

IBM Rexx/370 Compiler and Library

1995

1995 May 1..3

Rexx Symposium
Stanford, California

IBM Rexx/370 Compiler and Library
Service and Development

rexxcomp@vnet.ibm.com

Disclaimer

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "As Is" basis without any warranty either expressed or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

In this document, any references made to an IBM licensed program are not intended to state or imply that only IBM's licensed program may be used; any functionally equivalent program may be used instead.

Any performance data contained in this document was determined in a controlled environment, and therefore the results which may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

It is possible that this material may contain references to, or information about IBM products (machines and programs), programming or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming or services in your country.

Permission is granted to the Rexx Symposium for Developers and Users to publish this presentation paper in the Proceedings of the Rexx Symposium for Developers and Users.

Products

- Compiler:
  - IBM Compiler for SAA Rexx/370, Release 3
    - Program number 5695-013
    - CompID 569501301 FMID HWK0130 (MVS)
    - CompID 569501302 FESN 0463773 (VM)
- Library:
  - IBM Library for SAA Rexx/370, Release 3
    - Program number 5695-014
    - CompID 569501401 FMID HWJ9130 (MVS)
    - CompID 569501402 FESN 0463776 (VM)
  - Rexx/VSE Library, Release 2
    in Rexx/VSE, Version 1 Release 1
    - Program number 5686-058
    - CompID 568605802
  - Rexx/VSE Library, Release 2
    in VSE Central Functions, Version 6 Release 1
    in VSE/ESA, Version 2 Release 1
    - Program number 5686-066
    - CompID 568606612

Operating Systems

- MVS
  - TSO/E V2R3M1 or later on MVS/ESA SP V4R1 or later
  - TSO/E V2R4 or later on MVS/ESA SP V3R1
  - NetView V2R2 or later with above
- VM/CMS
  - VM/ESA V1R1 or later
  - VM/XA SP R2 or later
  - VM/SP R5 or later
  - VM/HPO R5 or later
- VSE (Library only)
  - Rexx/VSE V1R1 or later on VSE/ESA V1R3 or later
  - VSE/ESA V2R1 or later
    (Rexx/VSE integrated into base)

Language Levels

The Rexx language level accepted is:

- 4.00 on VM/ESA V1R2.1 and later
  including stream I/O[R3]
- 3.48 everywhere else
  including Trace[R3] and Interpret[R2]

With Release 3, the Rexx Compiler and Library now supports the entire classic Rexx language

Compiler and Library Publications

IBM Compiler and Library for SAA Rexx/370, Release 3:

- Licensed Program Specifications (GH19-8161-02)
- Introducing the Next Step in Rexx Programming (G511-1430-02)
- User's Guide and Reference (SH19-8160-03)
- User's Guide and Reference (Japanese) (SH88-7187-03)
- Diagnosis Guide (SH19-8179-01)
- User's Guide and Reference and Diagnosis Guide (SK2T-1410-00)
  included in IBM Online Library Omnibus Editions:
  - MVS Collection (SK2T-0710-10)
  - VM Collection (SK2T-2067-06)
  - VSE Collection (SK2T-0060-05)

Program Directories

- MVS Compiler: PRGDDIR820P, October 1994
- MVS Library: PRGDDIR817P, October 1994
- VM Compiler: PRGDDIR83F2, March 1995
  (replaces PRGDIR822P, October 1994)
- VM Library: PRGDDIR82F2, March 1995
  (replaces PRGDIR818P, October 1994)

Other Pubs About Using The Compiler

- TSO Extensions Version 2
  - Rexx/MVS Reference (SC28-1883-06)
  - Rexx/MVS User's Guide (SC28-1882-04)
  - Customization (SC28-1872-07)
- VSE/ESA V2R1
  - Rexx/VSE Reference (SC33-6642-00)
  - Rexx/VSE User's Guide (SC33-6641-00)
  - Rexx/VSE Diagnosis Reference (LY33-9189-00) (available August 1995)
- Rexx/VSE V1R1
  - Reference (SC33-6529-00)
  - User's Guide (SC33-6528-00)
  - Diagnosis Reference (LY33-9144-00)
  - Getting Started (GG24-4192-00)
- Book
  - The Rexx Handbook
    Gabriel Goldberg, Philip H. Smith III
    1992, McGraw Hill (SB20-0020-00)

Communicating

- Service: USREXX,182 or WTREXX,182
  - 569501301 R130 MVS Compiler
  - 569501302 R130 VM Compiler
  - 569501401 R130 MVS Library
  - 569501402 R130 VM Library
- Electronic
  - IBM TalkLink: RexxComp CForum
  - VMSHARE: Memo RexxComp
  - VMSHARE: Prob RexxComp
  - VMSHARE: Note RexxComp
  - ListServ: RexxComp@bitnic.cren.net
  - EMail: RexxComp@vnet.ibm.com
- Readers' Comment Form
  - Internet: pubrcf@vnet.ibm.com
  - IBMLink: GDLVME(PubRCF)
  - IBM Mail: USIB2L8Z@IBMMail
  - Fax: USA 607-752-2327

Release History

Short Name	Program Number	Rel	First Avail.	End of Service
CMS Comp & Libr	5664-390	1	89Jun30	95Sep22
CMS Library	5684-124	1	89Nov17	95Sep22
Rexx/370 Compiler	5695-013	1	91Aug30	93Nov28
Rexx/370 Library	5695-014	1	91Aug30	93Nov28
Rexx/370 Compiler	5695-013	2	93May28	95May07
Rexx/370 Library	5695-014	2	93May28	95May07
Rexx/VSE V1R1 Libr	5686-058	2	93Sep17	
Rexx/370 Compiler + Alternate Library	5695-013 + PN48006(MVS)	2	93Nov04 PN48015(VM)	95May07
Rexx/370 Compiler	5695-013	3	94Nov07	
Rexx/370 Library	5695-014	3	94Nov07	
Rexx/VSE V2R1 Libr	5686-066	2	95Apr21	
Rexx/VSE V2R1 Libr	5686-066 +	3	95Oct27	

Determining Levels

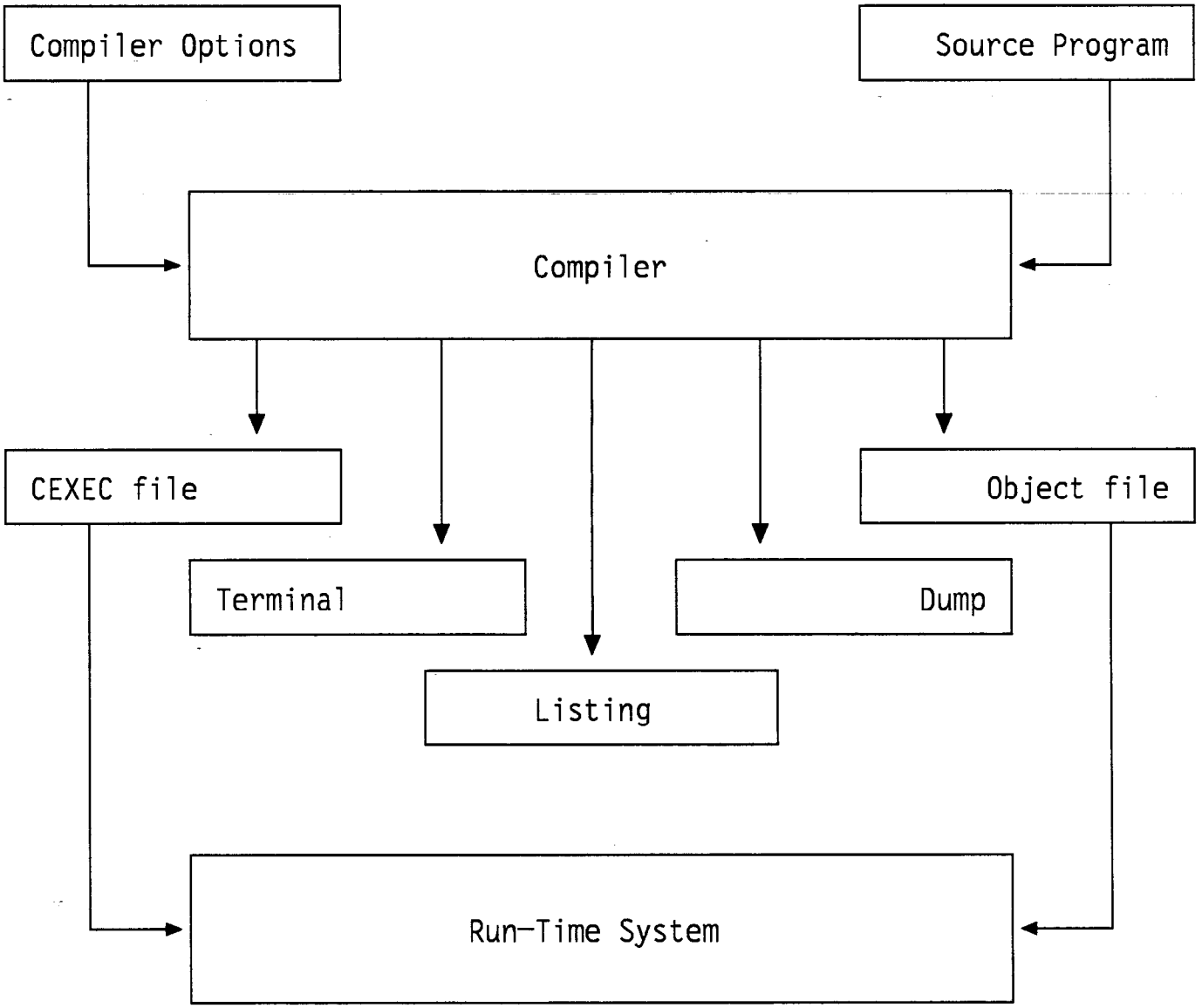
- Compiler
  - From program listing: Release, PTF
- Library

  Offset from beginning of first EAGRTLIB in file

Release	+9..+13
PTF	+19..+25
Date	+37..+44
Time	+46..+50
- Compiled program

Field	CExec file	Object file
Release	rec 1 cols 36..40	rec 2 cols 52..56
Compilation Date	rec 1 cols 43..54	rec 2 cols 60..70
Compilation Time	rec 1 cols 56..63	rec 2 cols 72+ rec 3 cols 17..23
Compilation System	rec 1 cols 65..67	rec 3 cols 25..27
Language Level	rec 1 cols 78..81	rec 3 cols 38..41
Compiler Date	rec 1 cols 83..93	rec 3 cols 43..53
Compiler PTF	rec 1 cols 99..105	rec 3 cols 59..65

Compilation



Note: No compiler work files, everything kept in virtual storage

Compiled Rexx Files

- CExec and Object files contain the same information, except for one bit indicating what kind of file it is, but are formatted differently
  - CExecs are used the same way Execs are used
  - Object files are used the same way other high-level language compiler outputs are used (link-edit)
- Contain
  - Executable S/370 instructions
  - Invocations of Library routines
  - Symbol tree, with names and descriptors
  - Control blocks
- Are reentrant, relocatable, and XA (31-bit) capable
- Are execution operating system independent
- Can use any Library at a release level at least as great as the Compiler
- Don't contain the program source (unless compiled with SLine option)

Rexx Is Hard To Compile

- Dynamic program structure
  - No conventional block structure
  - Start a procedure by executing Procedure instruction
  - End a procedure by executing Return instruction
- Signal can transfer control most anywhere
- No data types but some operations content dependent
- Variables
  - Are not declared
  - Can change attributes dynamically
  - Come and go dynamically
  - Can be shared with external programs
  - Names can be computed
  - Size limited only by storage
  - Arithmetic precision can be set dynamically
- Program text can be created dynamically

Assumptions That Make Compiling Worthwhile

- Assignments appear often
- Simple arithmetic appears often
- Control constructs appear often
- Do loops appear often
- Interpret not used often
- Storage management is expensive

Performance

Compiled programs that include many	Run this much faster
Arithmetic operations	6 to 10 times
String and word processing	6 to 10 times
Constants and variables	4 to 6 times
References to procedures and built-in functions	4 to 6 times
Changes to values of variables	4 to 6 times
Assignments	2 to 4 times
Reused compound variables	2 to 4 times
Host commands	Minimal improvement

Optimizations

- No tokenizing/parsing at run-time
- Address simple variables and stems directly
- Compiler optimizations
  - Common subexpressions
  - Constant folding
  - Value propagation
  - Less general code generation with knowledge about state of variables, Numeric Digits setting, and types of operands
  - Not load addresses already in register
- Fast linkage to library routines
- Optimized storage management for several kinds of use
- Binary arithmetic
- String arithmetic optimized for large numbers
- Avoid string movements, reuse string storage
- Lookup for compound variable access not always from top
- Cache compound variable addresses
- Optimized for compound variable integer tails

Optimization stoppers

- Interpret instruction
- Trace compiler option
- Numeric Digits < 9 suppresses binary arithmetic
- Numeric Digits unknown suppresses binary arithmetic
- Integers coded in exponential notation, with decimal points, or in strings with non-digit characters suppress binary arithmetic (1e0, 1., '1 ', ' 1' vs '1', 1)
- Labels stop compound variable access optimizations
- Referenced labels may stop other optimizations
- Labels within loops require run-time checks for jumps into loop
- More than three numeric tails suppresses numeric tail optimizations

Note: A program compiled with the Trace[R3] option is fully interpreted by the run-time Library and will perform better than when interpreted by the system interpreters

Optimizing programs

- Quoted strings perform better than variable names
- Assignment of quoted strings perform best
- TestHalt slows down loops (especially on MVS)
- Compiled assignment is faster than Parse
- Assignment preserves binary value
- Simple variables are faster than compound variables
- Exposing stem is faster than exposing compound variable
- Binary representation can be forced (a+0)
- Preallocating strings faster than extending strings
- DLinked modules perform best
- Object compiler output can be used in function packages (which can be DLinked)

Extensive Error Reporting

- 232 compile-time message numbers
  - Detailed static syntax analysis of entire program
  - Marks probable cause of error in listing
  - Cross-reference can be used to
    - find misspelled and similarly spelled names
    - find variables never assigned a value
  - Can flag non-SAA language elements
- 182 run-time message numbers
  - Issues standard Rexx error messages
  - Plus more detailed messages for each error
- Messages can be translated to other national languages (Japanese available)
- Both compiler and library have internal diagnostic facilities to help isolate internal errors

Program Listing

- On every page
  - program identifier
  - compiler release and PTF level
  - compilation date and time
- Compilation summary
  - Compilations status
    (number of messages, severity code)[R2]
  - Each compilation option with specified or default value
  - If ETMode in effect[R2]
- Source listing (optional)
  - Nesting levels for If, Do, Select
  - Program line numbers and record and file numbers[R3]
  - Messages interspersed with markers to probable cause on line

- Cross-references (optional)
  - Grouped by
    - Labels, built-in functions, external routines
    - Constants (optional)
    - Simple variables
    - Stems and compound variables
  - Include
    - The item
    - Attributes
    - Line references
    - Where set and for labels: valid definition, reference to undefined, duplicate
  - Host commands in source[R3] (optional)
- Compilation statistics[R2]
  - Number of source lines
  - Size of compiled program
  - Message statistics
  - Flagged source line numbers
  - Included files names[R3]

Alternate Library (R2+PTF)

- Run compiled execs without the Library product
- Can be distributed freely, without charge
- Can be packaged with compiled Rexx applications
- Uses interpreter so no performance advantage
- Alternate and SLine compiler options required
- Condense option may be used
- Can be used for either CExec or Object files
- Compiled execs can use actual Library if available

Condense (R1)

- Compiled programs larger than source
- Condensed programs usually smaller than source, even when source lines included
- Expansion occurs when program invoked
- Advantages
  - Less disk space
  - Less I/O when read into storage
  - After expansion at start-up, no performance degradation
  - Source scrambled, including host commands and constants, even when source lines included
- Disadvantages
  - More storage when running (both condensed and expanded versions remain in storage)
  - More processor time to expand when invoked
  - Can't use DLink option

- Use Condense when
  - I/O is the bottleneck and storage isn't
  - Program resides on disk or non-shared storage
  - Program is large
  - Program is long-running
  - Program is seldomly invoked
  - Source or constants need protection
  - DLink not required

Copyright (R2+PTF)

- Control directive — /*%Copyright ... */
- Inserts notice as visible text in compiled file
- Inserted notice is the concatenation of all Copyright directives in a program
- Treated as a comment by Rexx interpreters

Margins (R3)

- Can specify left and right text bounds of source files
- Only text within margins is compiled
  Compiler listing contains complete record
- SLine and IExec output contain only text within margins
- On MVS, file sequence numbers detected and removed before margins applied

Include Files (R3)

- No longer necessary to have entire program in 1 source file
- Control directive — /*%Include file_id */
  - Inserts included file immediately following the */
  - Includes may be nested
  - Included files may be members of libraries
  - Treated as a comment by Rexx interpreters — but ...
- IExec compiler option
  - Generates a single file with all program source, %Included or otherwise
  - Contains only text within specified margins
  - Can be used to interpret programs composed of include files or with non-Rexx text outside of margins

Object

- Use Rexx program as would other high-level language programs
  - Build modules
  - Command or program search order
  - Use various MVS/VSE parameter passing conventions
    - TSO/E command
    - Rexx external routine
    - Either TSO/E or Rexx external routine
    - MVS program
    - VSE program
    - TSO/E Called command
- Build function packages
- Combine with routines written in other languages
- Same file content as CExec, just different format
- Get external symbol and relocation information with DLink option

DLink (R1)

- Combine external functions and subroutines into 1 executable module
- Direct linking instead of searching
  - Can be very significant performance improvement
- Can create self-contained modules
- No name clashes with user's environment
- No behavioral changes due to changes to external routines
- Select which routines are included — doesn't have to be all routines (generates weak external references)

Possibilities?

- Object Rexx
- More, better optimizations
- Better error reporting by recognizing bifs and operand types at compile time
- ANSI flag option — flag non-ANSI syntax
- NoExecComm option — assume no ExecComm interface, means better optimization possible
- WDB/WDBLang debugger support — generate needed side files
- AutoSLine option — include source only if SourceLine bif used
- SLine option ranges — include only selected source
- Scramble imbedded source — improve security
- Compiler dump range option — reduce dump volume
- Page width option — support wider lines
- Indicate minimum runtime level required on listing and via utility and function

- Error number cross reference option
- Add column numbers to messages and list of flagged lines
- Print DCB parameters in options list
- Support alternate DD names
- Include invalid hex and binary strings in cross reference listing
- Print hex and binary strings as they appear in source
- Spilt source lines at more sensible places in listing
- More dump data — unsorted symbol table, environment interface, lister
- User specified placement of TestHalt hooks
- Ability to build single executable that doesn't require runtime library
- OS/2 syntax checker, lister
- Source reformatter — indent by nesting level, etc.

- Classic Rexx compiler and library for
  - OS/2, WARP
    - Intel
    - PowerPC
  - AIX, UNIX
  - WindowsNT, Windows95
  - AS/400
  - CICS/MVS (library only, both)
  - VSE (compiler)
  - PC DOS
  - Other

- Classic Rexx compiler and library for
  - OS/2, WARP
    - Intel
    - PowerPC
  - AIX, UNIX
  - WindowsNT, Windows95
  - AS/400
  - CICS/MVS (library only, both)
  - VSE (compiler)
  - PC DOS
  - Other

How REXX Helped Me Hit the Ground Running in UNIX

Lois White
Stanford Linear Accelerator Center

Pages 360-362

How REXX Helped Me Hit the Ground Running in UNIX

or

How I Stopped Worrying & Learned to Love Typing in Significant Mixed Case

Lois White
SLAC Computing Services
Stanford Linear Accelerator Center, Stanford, California

6th International REXX Symposium
Stanford Linear Accelerator Center, Stanford, California
May 3, 1995

Since the early 80's, REXX has been my language of choice in VM/CMS for a system of execs and SAS programs which manipulate and store data and produce daily, month-to-date, and month-end reports on resource utilization and performance. Actually, I created and still maintain three VM service machines which keep track of utilization and some performance statistics for VM/CMS, a VAX cluster, and SLAC's telephone system.

About three years ago, it was announced that UNIX would be the future direction for physics computing at SLAC. In order to prepare for the coming of UNIX, I took an introductory course and was appalled...shell scripts??...no REXX??? I looked at bourne, korn, and c shell scripting and said to myself "I thought we had progressed beyond EXEC and EXEC2". Then someone mentioned that **perl** was the scripting language to use in UNIX. When I looked at **perl** I found it to be powerful and concise, but nearly impossible to decipher without comments on each line! I was looking at a steep learning curve here!

As the new RS6000 machines began arriving, it quickly became evident that I would need to start accounting for resource utilization on them and get an idea of how much they were being used and by whom. The accounting software included with the RS6000 Base Operating System was minimal, but it produced most of the information we needed. However, it didn't store data in a form that is readily used for reporting purposes. It also didn't clean up after itself very well, allowing directories to grow indefinitely. A great deal of manual intervention was required in order to save data and produce reports on a regular basis and to keep directories from filling up. It was clear that I needed to have more than just crontab entries and SAS software and I needed it soon!

Writing **perl** or shell scripts would have accomplished the task but I estimated that it would take some time, perhaps months, to become proficient enough to do what was needed in a lot less time. In early 1993, I learned that The Workstation Group's uni-REXX had arrived at SLAC and I felt as if I'd been saved. All subsequent references to REXX in UNIX in this paper refer specifically to the use of The Workstation Group's uni-REXX.

In the VM/CMS world I had already developed techniques for data storage, data manipulation,

and report production which I could now use in the UNIX world with the arrival of REXX. Close to ten years of experience using VM/CMS REXX meant that I wouldn't have to spend three to six months achieving the skills I needed before I could start managing data and producing reports for the rapidly multiplying UNIX machines. Furthermore, the uni-REXX manuals were written for users making the transition from VM/CMS to UNIX. With all this encouragement, I jumped in and wrote my first REXX script in UNIX which copies AIX disk accounting data from the file where it gets replaced each time disk accounting runs to another directory where it is saved and used later to analyze disk usage on a long term basis.

Since then I have developed a system of crontab entries and REXX and SAS programs which store and manipulate UNIX accounting and performance data and produce daily, month-to-date, and monthly UNIX resource utilization and performance reports. On each of the (now) sixty-four RS6000s where we collect accounting data, a REXX program executes daily which copies the locally stored data to a generally accessible directory where accounting data for all sixty-four machines is stored. After that, another crontab entry on one machine executes a REXX program which initiates the daily data processing and subsequent analysis reports by executing SAS and additional REXX programs. This daily program's decisions on which processes to start are based on the current date, day of the week, and other criteria. There are several other cron-initiated REXX programs which take care of data copying, directory cleanup, and daily checking of all automatic processes. In addition, there are several REXX programs which are executed manually to rerun processes which failed and to perform tasks such as large scale data backup.

Outside of the accounting and performance area, I have used the same techniques to develop a system of crontab entries and REXX and SAS programs to produce regular reports and graphs analyzing network performance data for our UNIX systems.

In conclusion, REXX enabled me to "hit the ground running in UNIX" because it has:

Familiarity:
I had many years of experience writing REXX code.

Portability:
I was able to transfer several large pieces of REXX code from VM/CMS to UNIX and use them, often without modifications. These included useful algorithms for cleaning up old files, finding dates, creating file names, etc. to use as parameters for execs and SAS programs.

Communication with UNIX:
It is possible to issue UNIX commands from REXX programs. By using the POPEN instruction or function, it is possible to read output from UNIX commands and to test return codes. Sometimes I've found that using a UNIX command is more efficient than doing the same task with REXX code, e.g. file editing using the sed utility instead of REXX's linein/linout functions.

Readability:
REXX code is relatively easy to understand and usually doesn't require adding comments on every line in order to remember "*Why* did I do *that*?!"