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

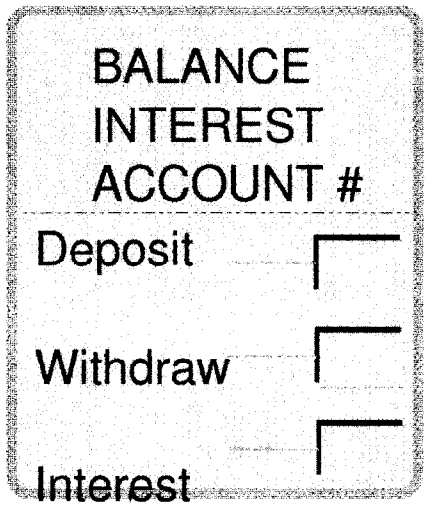
What's New in Object REXX?

- 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)

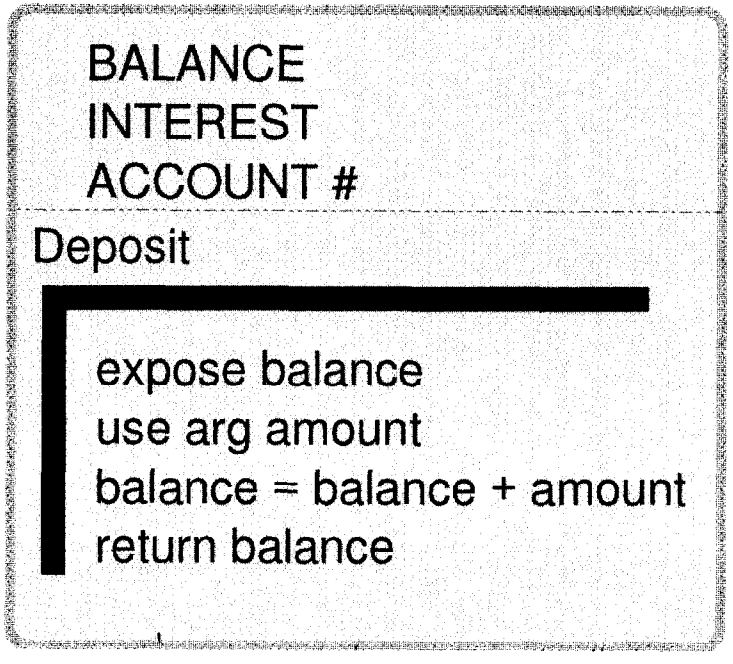
deposit(1457.11)

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

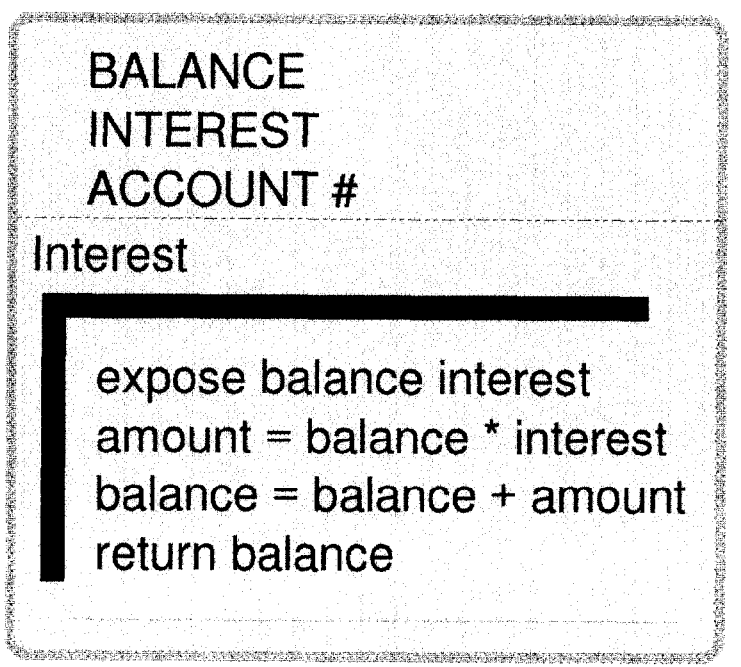
- New syntax:
  - receiver~ message(arguments)
  - receiver~~ message(arguments)
  - receiver[arguments]
- arguments are optional, e.g.:
  - receiver~ message
- May appear as term, instruction, or assignment target
- All REXX operators become messages
  - Can use either syntax

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

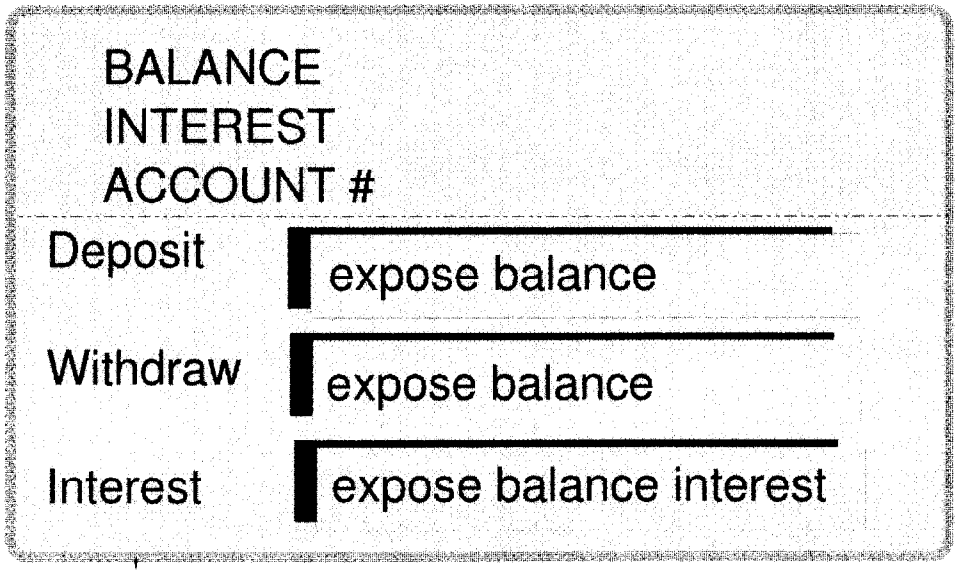
- All variables are references to objects
  - Strings are just one type of object
- Method variables (a.k.a. "local") exist only while method is running
- Object variables last as long as the object does



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...

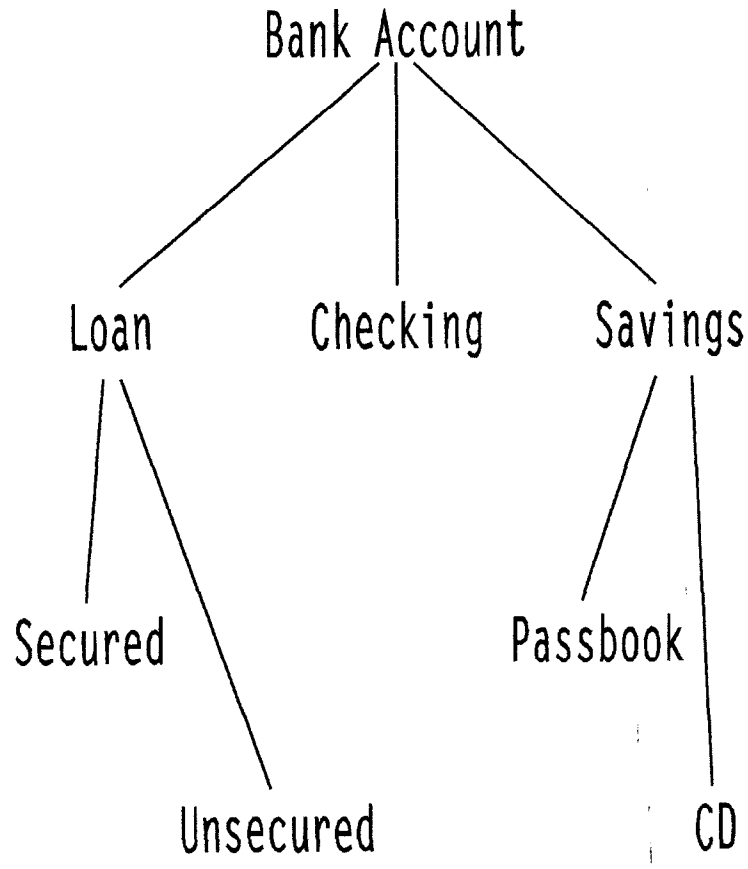
- New user condition support allows users to define own conditions
- New RAISE instruction
  - RAISE condition DESCRIPTION expression
  - "condition" can be any of
    - *rexxcondition*
    - *SYNTAX number*
    - *USER usercondition*
  - "expression" is returned to handler by CONDITION('D')
  - RAISE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

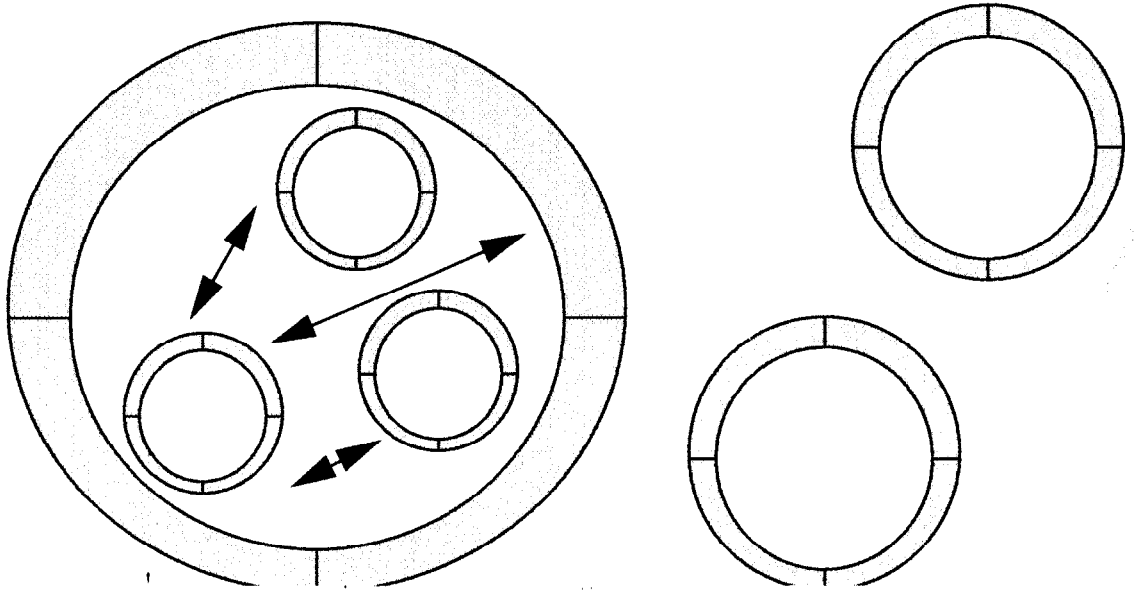
Explicit Concurrency

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

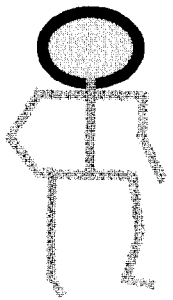
- SOCKET: an OS/2 sockets encapsulation
  - Goal: Clients, Servers without knowing TCP
  - "Server" contains concurrent TCP Objects
    - *"Known Port" socket for service requests*
    - *"Client Sessions" created for each client*
  - "Client" Object(s) request service via TCP



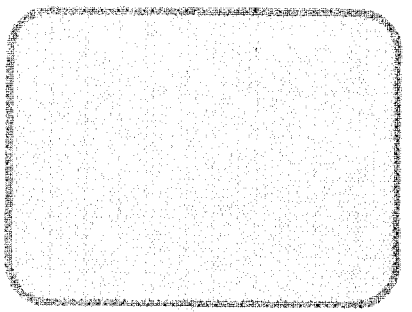
Playing Around, continued

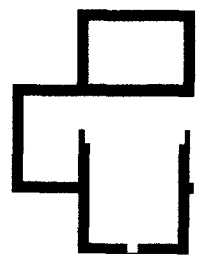
- Socket 'Mirror' TCP C/S Applet:
  - "Framework" classes: 165 lines
  - Client Script: 15 lines
  - Server Script: 27 lines
- Second applet -- 'Toss server':
  - Inherit Socket framework
  - Client Script: 2 changed lines
  - Server Script: 15 new/changed lines

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

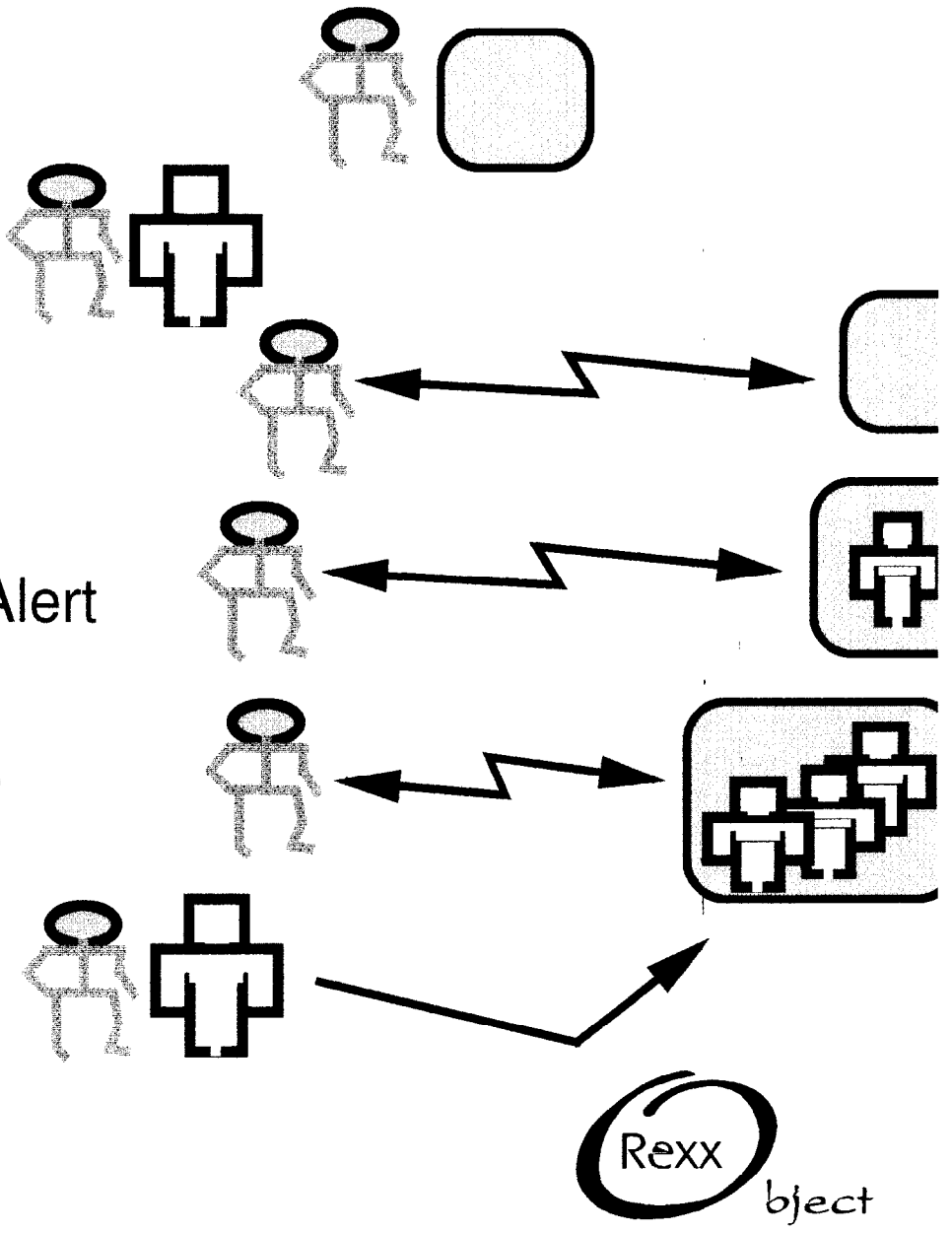




Rexx bject

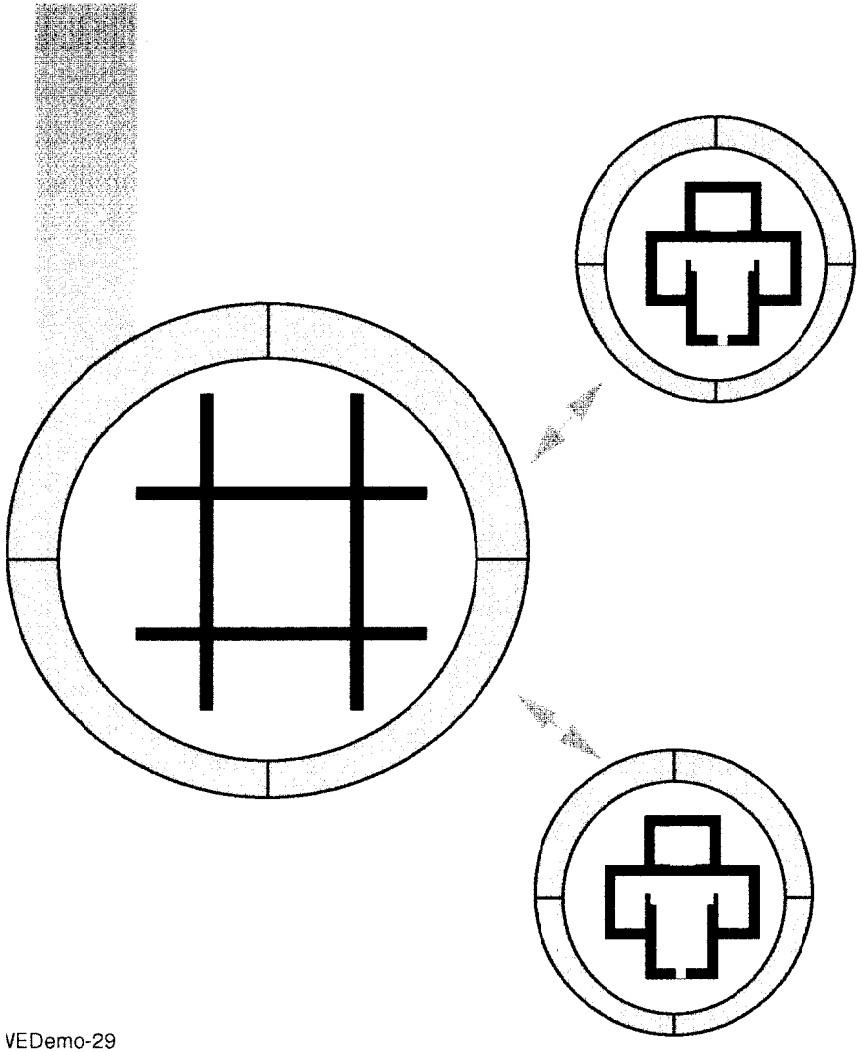
"Mobile Computing": Modes of Communications

- 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

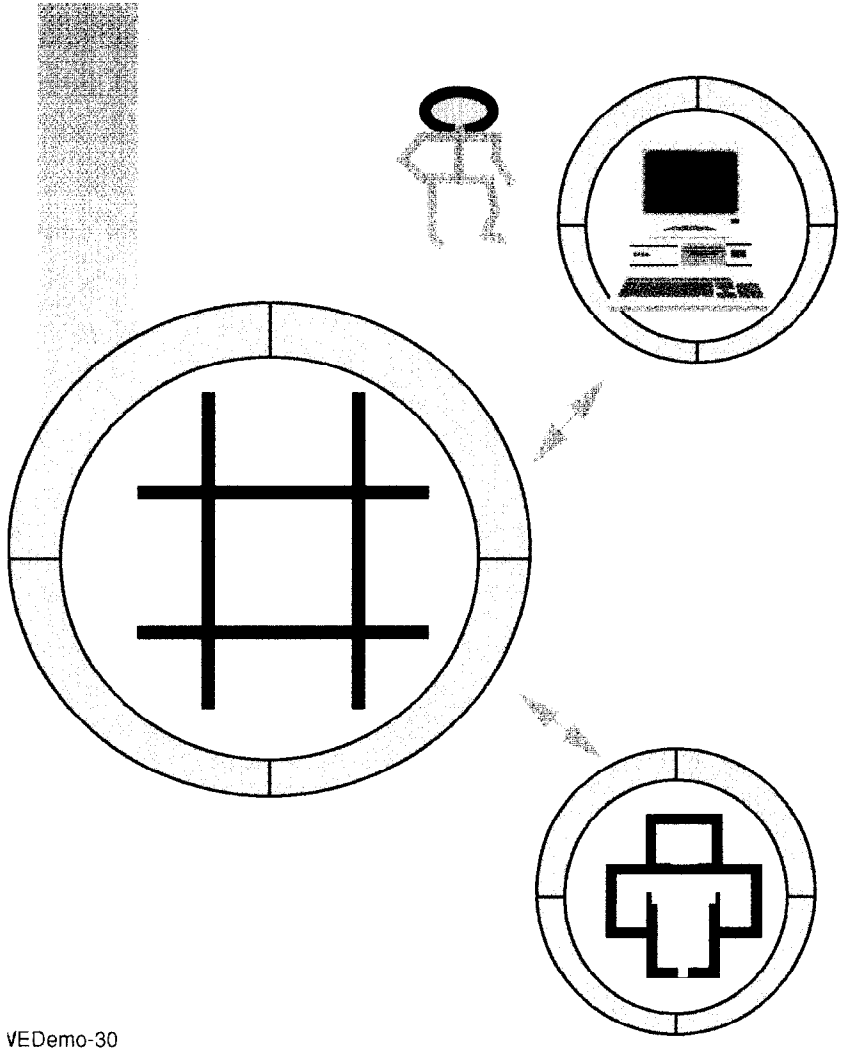
- or, JimBob and Rambo play TicTacToe



"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'

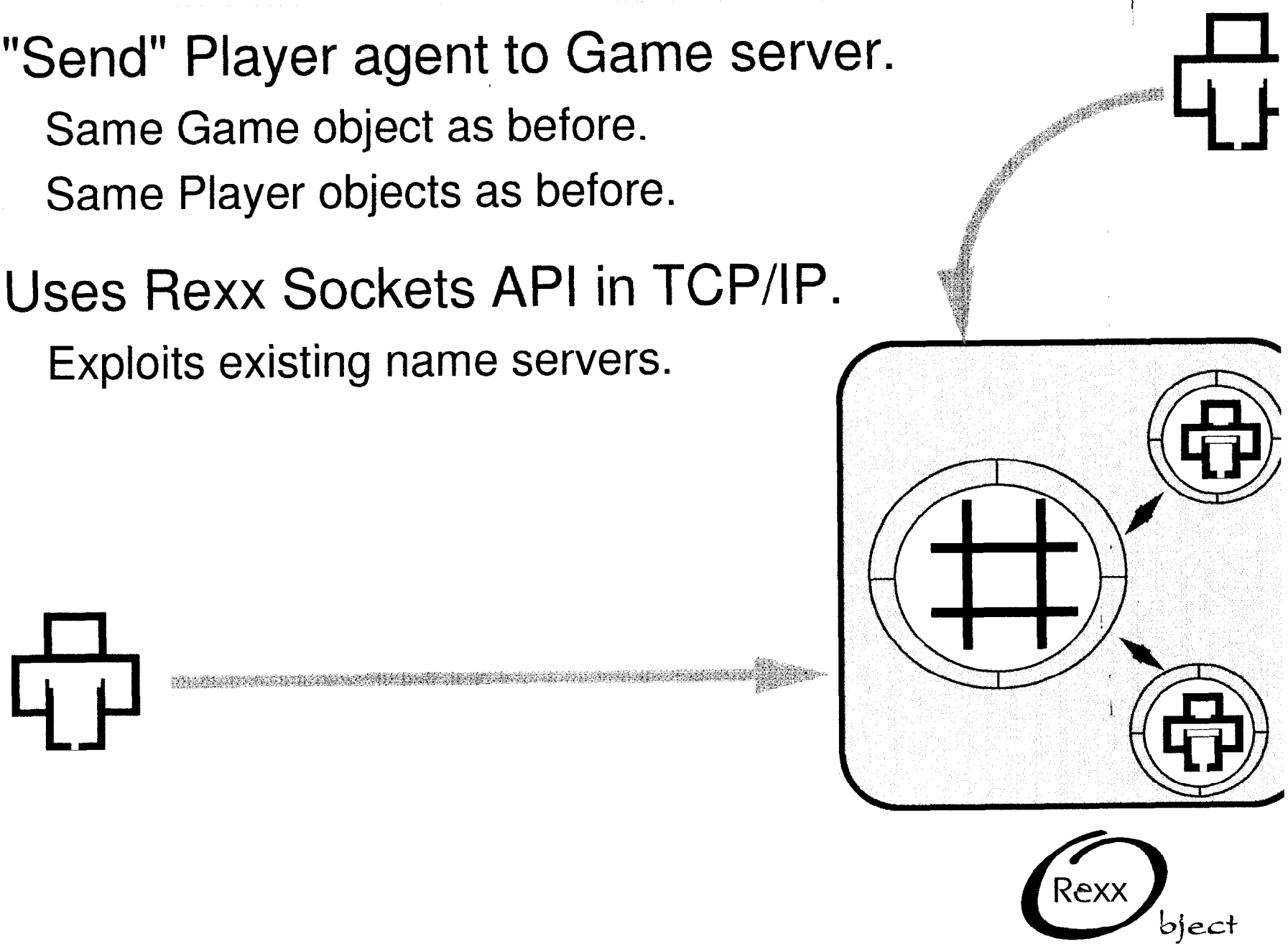
A TicTacToe Agency

"Send" Player agent to Game server.

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

Uses Rexx Sockets API in TCP/IP.

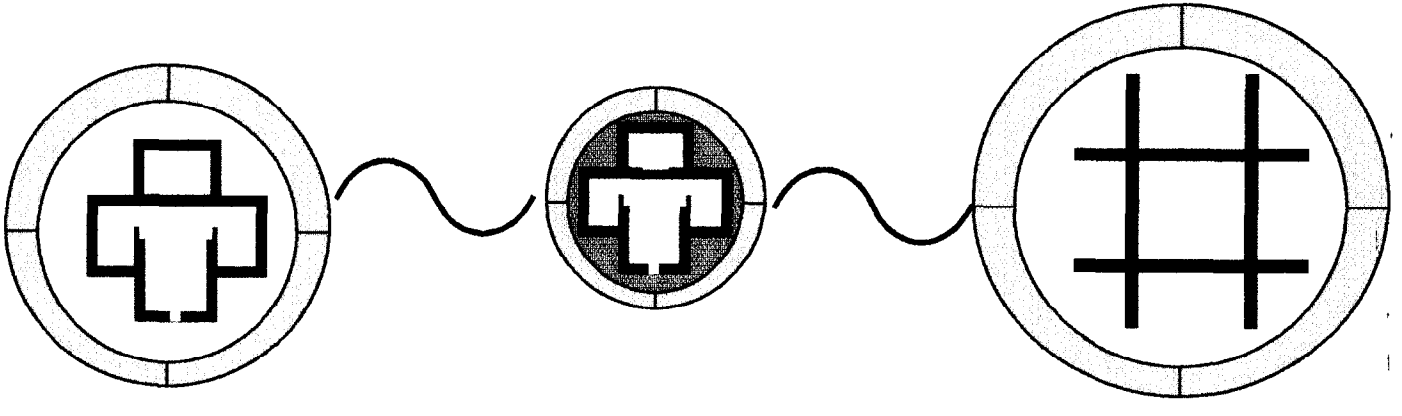
- Exploits existing name servers.



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...

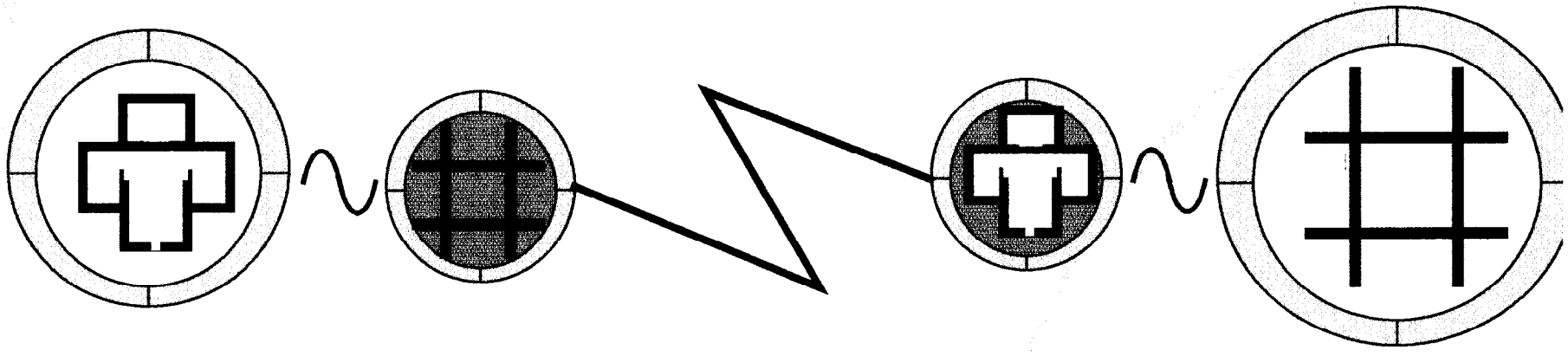


Rexx bject

Communications Proxies

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

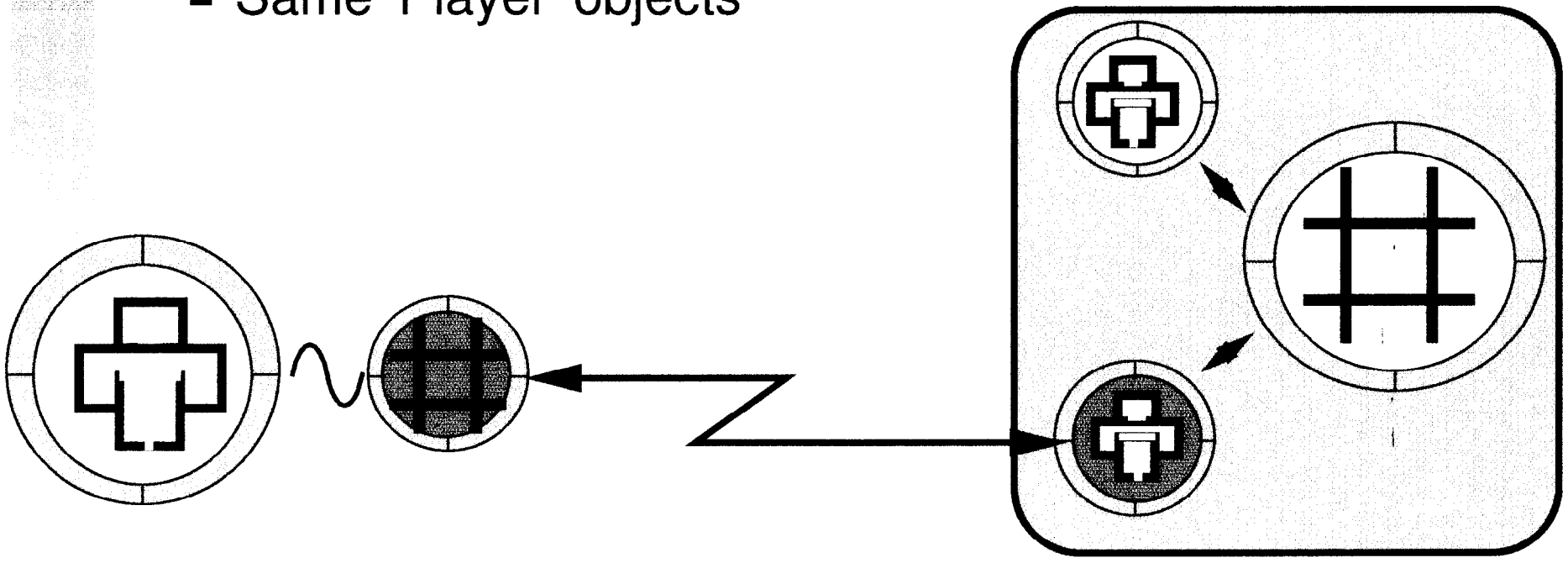
So, communications proxies can network-enable objects that 'know' nothing about networks.



Remote Messaging via Proxies

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

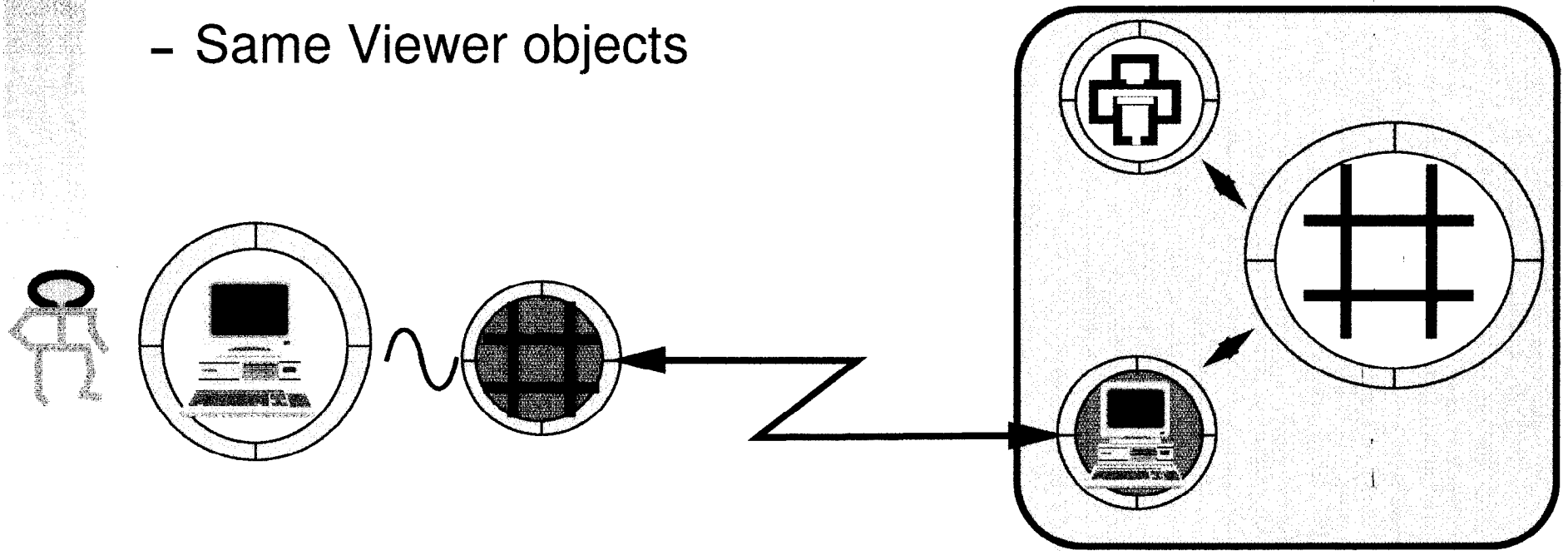
- Same 'Game' object
- Same 'Player' objects



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