From Rexx to NetRexx

A quick impression

René Jansen, 36th International Rexx Symposium, Vienna 2025

The JVM

Java Virtual Machine

- In 1995 Java was ported to IBM's platforms, starting with OS/2, as part of IBM's emerging multi-platform strategy. AIX, Linux, Windows, OS/400 and z/OS followed. NetRexx was part of the VM/ESA product for a while.
- The first (OS/2) port was done by MFC (Mike Cowlishaw), who was then wondering what Rexx would look like if it ran on the JVM. First a translator was produced, and when that worked well (1996), an interpreter was added (2000)
- The NetRexx translator produces Java code which is compiled into .class files
- It runs, and can be built, everywhere where there is a Java Virtual Machine

It looks just like Rexx

but every program is a Class

- `SAY 'HELLO'`
- it has the Rexx built-in functions
  - in oo-notation, like `VARIABLE.LEFT(4)`
  - or in traditional notation, like `LEFT(VARIABLE,4)`
- Case insensitive just like Classic Rexx and ooRexx

```
emacs hello.nrx                                                    13
File Edit Options Buffers Tools Index NetRexx Help
/* This is a very simple NetRexx program */
say 'Hello World!

-UUU:---  F1  hello.nrx      All   (1,0)      (Netrexx) ----------
```

Equivalent, "complete boilerplate" version:

```
emacs hello.nrx                                                    13
File Edit Options Buffers Tools Index NetRexx Help
class hello

  method main(args=String[]) static
    /* This is a very simple NetRexx program */
    say 'Hello world!'

-UUU:**-  F1  hello.nrx      All   (6,0)      (Netrexx) ----------
```

Objects

The Java way

- Inheritance (single- and interface-) and encapsulation
- **Properties Indirect** adds getter and setter methods for class level variables
- There is no labeled function, procedure or 'expose'
  - Everything is in Classes and Methods
    - Which are generated for you for the simple programs

ooRexx

emacs personstructure.rexx 11

```
File Edit Options Buffers Tools Help
p=.person~new("Doe, John", 10500)
say "name: " p~name
say "salary:" p~salary

::class person
::attribute name
::attribute salary

::method init
expose name salary
use arg name, salary

-UU-:---  F1  personstructure.rexx   All   (12,0)
End of buffer
```

NetRexx

emacs Person.nrx 14

```
File Edit Options Buffers Tools Index NetRexx Help
class Person
name
salary

  method Person(nm, sal) -- constructor
    this.name = nm
    this.salary = sal

  method main(args=String[]) static
    p=Person("Doe, John", 10500)
    say "name:" p.name
    say "salary:" p.salary

-UU-:**-  F1  Person.nrx    All   (13,4)    (Netre
```

```
emacs                                                                    12
File Edit Options Buffers Tools Index NetRexx Help
import com.eaio.uuid.UUID
import java.sql.

class Fact
  db = DBAccess

  properties indirect
  pred  = '' -- predicate
  subj  = '' -- subject
  obj   = '' -- object

  method Fact() -- default no-args constructor
    this.db = DBAccess.getInstance()

  method Fact(pred_, subj_, obj_)
    this.db = DBAccess.getInstance()

-UU-:---  F1  Fact.nrx       Top    (17,0)    Git-main  (Netrexx) ---------------------------
```

```
Program Fact.nrx
  === class Fact ===
    constructor Fact()
      signals ClassNotFoundException
      overrides Object()
    constructor Fact(Rexx,Rexx,Rexx)
      signals ClassNotFoundException
    method toString
      overrides Object.toString
    method toSQLInsert
    method toAssertion
    method toRetraction
    method setPred(Rexx)
    method setSubj(Rexx)
    method setObj(Rexx)
    method de_apo(Rexx)
    method write(PrintWriter)
    method writeDB(PreparedStatement)
    method persist(PrintWriter)
    method read(BufferedReader)
      signals IOException
    method readFix(BufferedReader)
      signals IOException
    method getPred
    method getSubj
    method getObj
Compilation of 'Fact.nrx' successful
```

A JavaBean pattern

Small Differences

- All character comparisons are case-insensitive
  - This was planned for Classic Rexx but dropped because of the performance of the computers of the era
- An uninitialized variable is not equal to its variable name like in Classic Rexx
- `PARSE` does not have `VAR` but goes straight to the variable
- Stem variables use [ ] instead of dot (.) notation
- An object is instantiated from a Class by calling its constructor()

Optional Arguments

on constructors

```
method charOblong(newwidth, newheight, newprintchar='X')
```

which indicates the third argument, if left out, will be X

if called this way:

```
first=charOblong(5,3) -- make an oblong
```

Seamless integration of JVM classes

- You can call any Java class without any ceremony
- For this purpose, the `IMPORT` statement works a lot like Java's (but has shortcuts)
- Import works on packages which you can add yourself
- You need to be aware of the CLASSPATH environment variable which is used to find classes
- Profits from all performance improvements of the JVM over the years

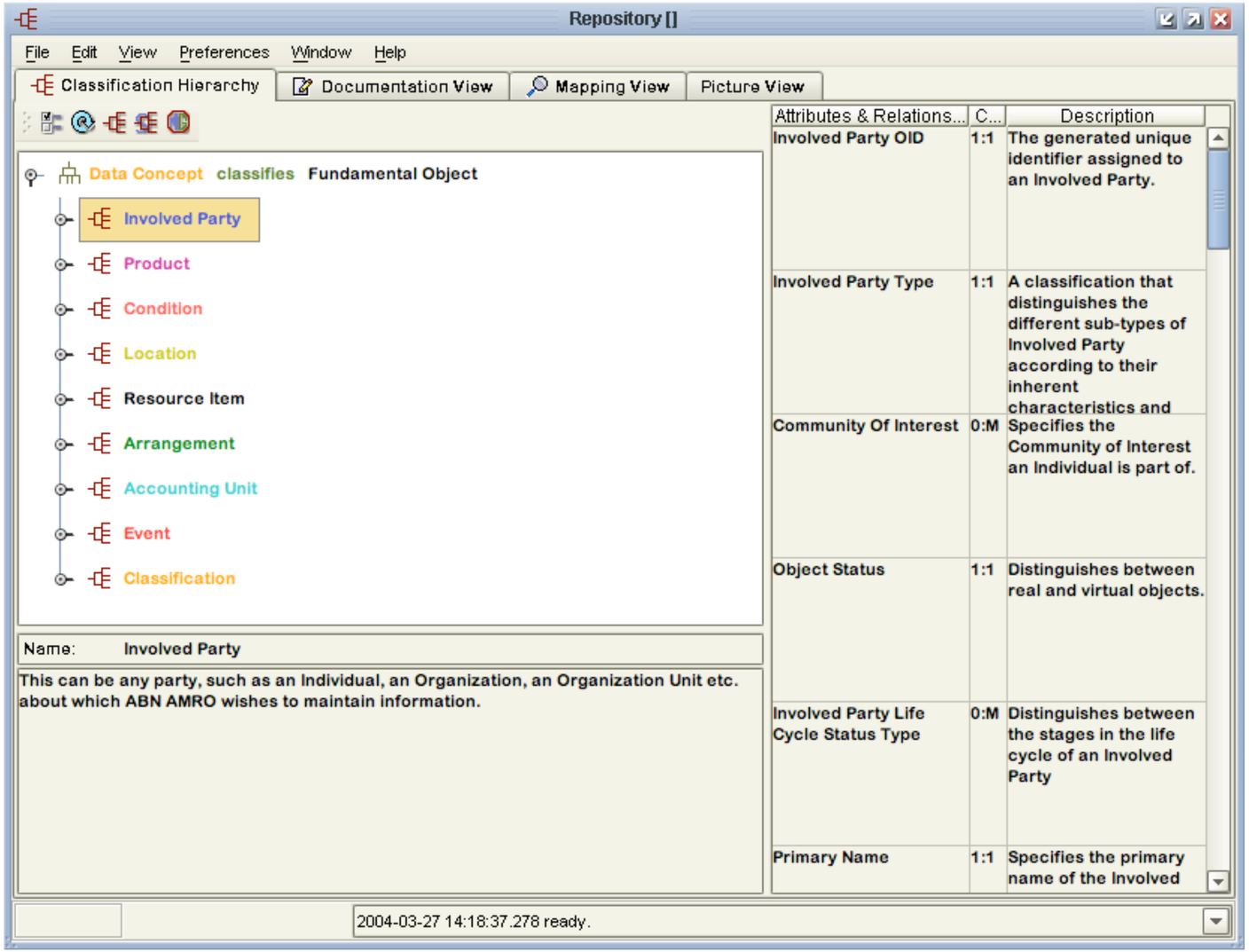
emacs 10

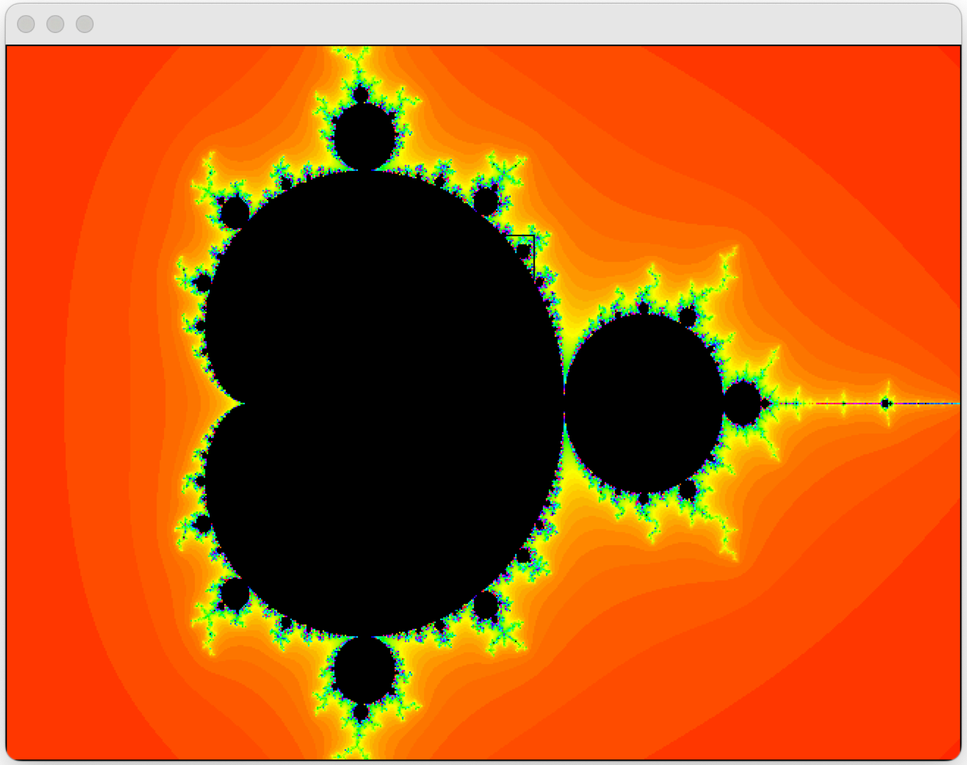
```
File Edit Options Buffers Tools Index NetRexx Help
/**
 * Class IKEAGateway implements commands to the IKEA Zigbee Gateway.
 * <BR>
 * Created on: Wed, 06, Nov 2019 17:03:32 +0100
 */

class IKEAGateway
  properties private
  gateway
  userid
  key
  devices
  devMap = TreeMap()
  istem=''
  estem=''
  /**
   * Default constructor
   */
  method IKEAGateway()
    getProperties()
-UU-:**-  F1  IKEAGateway.nrx   Top   (1,0)     Git-master  (Netrexx) -
```

TreeMap is a Java collection class

... including GUI Framework classes





Unicode

- Rexx Strings in NetRexx are arrays of Java char. A Java char is a Unicode character (UTF-16, but moving to UTF-8 over the years)
- so `"René".length() = 4` and not ~~5~~
- Use of Unicode is very un-problematic

Numeric Digits

unlimited precision, the Rexx way

```
Ready; numeric digits 1000                                                                                             0.794 s
Ready; say 100/81                                                                                                      0.009 s
1.23456790123456790123456790123456790123456790123456790123456790123456790123456790123456790123456790123456790123456790123456790
1234567901234567901234567901234567901234567901234567901234567901234567901234567901234567901234567901234567901234567901234567901
2345679012345679012345679012345679012345679012345679012345679012345679012345679012345679012345679012345679012345679012345679012
3456790123456790123456790123456790123456790123456790123456790123456790123456790123456790123456790123456790123456790123456790123
4567901234567901234567901234567901234567901234567901234567901234567901234567901234567901234567901234567901234567901234567901234
5679012345679012345679012345679012345679012345679012345679012345679012345679012345679012345679012345679012345679012345679012345
6790123456790123456790123456790123456790123456790123456790123456790123456790123456790123456790123456790123456790123456790123456
7901234567901234567901234567901234567901234567901234567901234567901234567901234567901234567901234567901234567901
Ready;                                                                                                                 0.025 s
Ready;                                                                                                                 0.000 s
```

JNI - The Java Native Interface

- External, native (to the instruction set of the platform) functions can be called through the JNI, the Java Native Interface
- Normal external functions would be written in NetRexx or Java and the JNI is reserved for specialist work

ADDRESS

- ADDRESS works like in Classic Rexx with some of the extensions of the Rexx Standard which it shares with ooRexx
- ADDRESS WITH can write from and to files and stem variables

JDBC - Java Database Connectivity

- Your program works on all database engines that have a JDBC driver (Db2, Oracle, Postgres, SQLite, MySQL, etc, etc including even MS Excel.
- This portability is a great bonus; your app works one day on z/OS with DB2 and the next day on Linux with PostGreSQL - unchanged!

- But for smaller programs: you can also just `ADDRESS` the database cli

emacs DBAccess.nrx 12

File Edit Options Buffers Tools Index NetRexx Help

```
  * Method getInstance returns the (only) instance of this class when
  * it initialized, or constructs an instance when it is not
  */
 method getInstance() returns DBAccess static protect
   if instance <> null then return instance
   instance = DBAccess()
   url_ = System.getenv("INVENTORY_URL")
   Class.forName("org.duckdb.DuckDBDriver")
   do
     jdbcCon = Connection DriverManager.getConnection(url_)
   catch e = SQLException
     printException(e)
   end -- do

   return instance

 /**
  * Method getDescriptorsForID returns the set of descriptor relationships
  * to an object, as in the DESC/SCHEME dichotomy
  * @param id is a Rexx String
  */
 method getDescriptorsForID(id) returns ArrayList
   l = ArrayList()
   do
     sqlstmt = "select subj " -
               "from inv " -
               "where pred = '89184770-1A5C-11E3-9DFC-0A0027000000' " -
     " and obj ='"id"'" -
     " order by 1"
     stmt = Statement this.jdbcCon.createStatement()
     rs = ResultSet stmt.executeQuery(sqlstmt)
```

-UU-(DOS)--- F1 DBAccess.nrx 5% (18,0) Git-main (Netrexx) ----------

Singleton Pattern

Connect to driver

Create and execute statement

emacs 12

File Edit Options Buffers Tools Help

```
/* rexx for writing all presentations for a year to a .tex file */
year=directory().substr(directory().lastpos('/')+1)
say date() time() 'starting writeyear for' year
lineout('presentations.tex','% presentations for 'year,1)

-- get the location
i6stem=''; i6stem=i6stem
i6stem[0]=2
o6utstem=''
i6stem[1]='connect rexxla;'
i6stem[2]='select location, startdate, enddate, isbn from event where year ='year';'
address system 'mysql' with -
  input stem i6stem -
  output stem o6utstem

parse o6utstem[2] location'\t'fromdate'\t'todate'\t'isbn
if location.pos('Online') > 0  then coupling = ''
else coupling = 'in'
if location.word(1)='Aruba' then coupling = 'on'
```

-UUU:--- F1 writeyear.rexx Top (1,0) Git-master (REXX) ----------

JPMS: The Module System

- NetRexx works on the JPMS, and tolerates its use
- This enabled NetRexx to run on Java 9 and higher
- Applause to Marc Remes for pulling this off

emacs

File Edit Options Buffers Tools Index NetRexx Help

```
      otherwise
        say 'RxJrt : Walks the JPMS jrt:/ file system and modules provided in --module-path'
        say ' Optional arguments'
        say '  [-a | -all]     show all'
        say '  [-m | -module]  show module'
        say '  [-p | -package] show package (actually a directory..)'
        exit 2
      end
    end

    rx = RxJrt()
    if \isJrt then do
      exit 1
    end
    else do
      rx.RxJrtTree()
      rx.RxModPath()
      exit 0
    end

-- constructor
-- check if running >= JDK9, special case CSR JDK-8227076

  method RxJrt
    v = NrVersion()
    say '# 'v.getLogo() v.getFullVersion() v.getProcdate()

    o = Object.class.getResource('Object.class')   -- check for 1.1.8+
    say '# Found Object.class at 'o
    os = o.toString()
    if os.startsWith(jrtprefix) then do
      isJrt = 1
      c = jrtprefix||os.substring(jrtprefix.length())
      p = Paths.get(URI.create(c))
```

-UU-:--- F1 RxJrtApi.nrx 23% (69,0) Git-master (Netrexx) ----------

Functional Programming

- Added later to the Java language
- NetRexx can make use of this

emacs functional.nrx

File Edit Options Buffers Tools Index NetRexx Help

```
wordstring=String "Just a bunch of words to test for killer items containing a k"
-- convert the string into a Java List (a Collection):
alist=ArrayList(Arrays.asList(wordstring.split(" ")))
-- now run a filter stream operation on the list
-- using a hard coded Predicate class for a filter instead of a Java lambda expression:
-- (the filter just selects words containing the letter 'k')
sa=alist.stream.filter(Pred()).toArray
-- print the results for verification:
loop y over sa
  say y
end
-- now run a foreach operation on a stream
-- using a hard coded Consumer class instead of a Java lambda
--     the consumer here just prints inputs with some surrounding brackets
alist.stream.foreach(Eatem())
class Pred implements Predicate
  method test(s=Object) returns boolean
    return Rexx(s.toString).pos('k')>0
class Eatem implements Consumer
  method accept(s=Object)
    say ">>"s"<<"
```

-UU-:--- F1 functional.nrx All (22,0) (Netrexx) ----------

```
killer
k
>>Just<<
>>a<<
>>bunch<<
>>of<<
>>words<<
>>to<<
>>test<<
>>for<<
>>killer<<
>>items<<
>>containing<<
>>a<<
>>k<<
```

Interpret

- Version 5.01 adds INTERPRET in interpreted (ha!) and compiled versions
- The full force of meta-interpretation is available

emacs modify.rexx

```
File Edit Options Buffers Tools Help
/* REXX */
code = "say 'this is step 1'; code = 'say ''this is step 2'''"

loop for 2
  interpret code
end

-UU-:---  F1  modify.rexx    All   (7,0)     Git-master  (REXX) ----------
```

rvjansen@Algol:~/test/interpret

```
$ nrc modify.rexx
NetRexx portable processor 5.01-GA build 320-20250501-1930
Copyright (c) RexxLA, 2011,2025.   All rights reserved.
Parts Copyright (c) IBM Corporation, 1995,2008.
Program modify.rexx
Compilation of 'modify.rexx' successful


Ready; T=0.813/0.813 23:39:53 nrc modify.rexx
[23:39:58] rvjansen ~/test/interpret [master  *]
$ modify.class
this is step 1
this is step 2


Ready; T=0.434/0.434 23:40:02 modify.class
[23:40:30] rvjansen ~/test/interpret [master  *]
$
```

Text Blocks

- Version 5.01 add multiline text blocks
- “”””Starts a multiline block”””
- Very convenient for Interpreted blocks and multiline SQL queries
  - This was present in embryonic REX but was dropped because of granularity of error messages.

emacs DBAccess.nrx 12

File Edit Options Buffers Tools Index NetRexx Help

```
      istmt.executeUpdate()

      return 0
    catch e = SQLException
      printException(e)
      exit
    end -- do

  method prepareInsertStatement() protect returns PreparedStatement
    do
      insert_st = """
      insert into inv( subj, pred, obj )
      values (?,?,?)
      """
      istmt = PreparedStatement this.jdbcCon.prepareStatement(insert_st)
      return istmt
    catch e = SQLException
      printException(e)
      return null
    end -- do

  method closeInsertStatement(istmt=PreparedStatement) protect
```

-UU-(DOS)--- F1 DBAccess.nrx 75% (251,1) Git-main (Netrexx) ----------

NetRexx Pipelines

- A very complete implementation of CMS Pipelines
- Multithreaded and multistream, top performance
- Like on CMS, callable from (Net)Rexx and your own Pipeline Stages can be written in (Net)Rexx

emacs ⌥⌘9

```
File Edit Options Buffers Tools Help
pipe < instructions.txt | sort 8 | specs /insert into instruction VALUES(''/ 1 1-6 next /'','/ next 8-18 \
strip next /'','/ next 18-38 next /'','/ next 39-100 next /');/ next | > insert_ins_dec.sql

-UU-:**-  F1  demo.njp       All   (2,0)      (Pipes) ----------------------------------------------------
```

Stream I/O

- The Rexx 4.0 ANSI I/O package that never made z/OS
- Added to NetRexx for larger compatibility with other Rexx'en
- Even more improvements in NetRexx 5.01

Complete documentation

- The NetRexx Language Definition, ISBN 978-94-648-5133-5
- The NetRexx Programming Guide
- The NetRexx Pipelines User Guide and Reference