Future NetRexx

René Vincent Jansen
Santa Clara, CA, 20080926

© 2008 IBM Corporation

Disclaimer

- Although I am presently subcontracting for IBM this presentation is made *a titre personnel.* All stated opinions are my own and do not necessarily reflect IBM's position or policies.
- NetRexx is presently classified EWS, the OSSC process must run its course for source code to be published.

I-Bizz IT Services and Consultancy
Amsteldijk 14
1074 HR Amsterdam

Gnosys Systems
Barcadera
Aruba

Java is a trademark of SUN Microsystems Inc.

© 2008 IBM Corporation

Wikipedia

IBM NetRexx
From Wikipedia, the free encyclopedia

NetRexx is IBM's implementation of the Rexx programming language to run on the Java virtual machine. It supports a classic Rexx syntax along with considerable additions to support Object-oriented programming in a manner compatibile with Java's object model. The syntax and object model differ considerably from Open Object Rexx, another IBM object-oriented variant of Rexx which has been released as open source software.

NetRexx is free to download from IBM[1].

© 2008 IBM Corporation

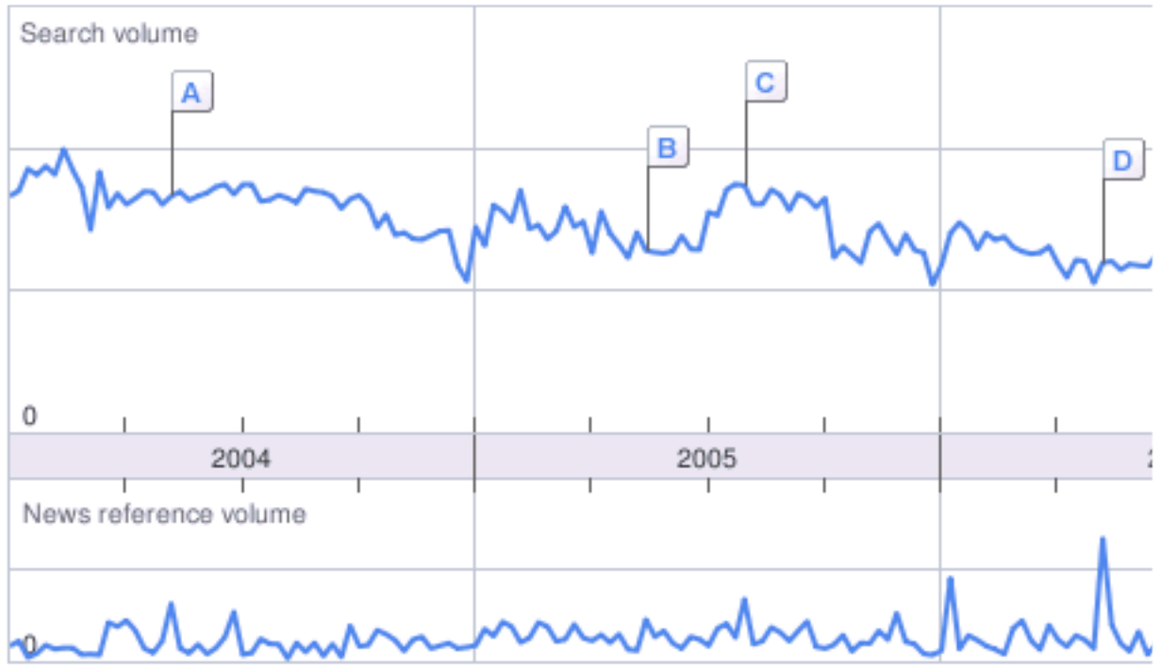
Short History of Rexx

- Mike Cowlishaw, 1979
- Rexx is considered the granddaddy of scripting languages
- Mike made IBM Fellow due to enormous success of language
- Successor of the EXEC2 language for the IBM VM Operating System
- Implemented natively on MVS, OS/2, AS/400, Windows, AIX, Linux, HP/UX, Solaris, MacOSX, etc.

- Three dialects: Classic Rexx, Open Object Rexx, NetRexx
- Creation of NetRexx went along with first Java port by IBM Hursley lab
- Rexx adapted to the Java Object Model
- Easy integration of NetRexx language and Java classes
- Stem. notation casualty of method invocation dot, other syntax cleanups

- Currently a compiler (translator) and an Interpreter
- In existence since 1996, Version 1 released 1997
- Interpreter added in 2000
- Stability in development since Mike C. focused on decimal arithmetic
- Intention to open source in 2008, restart of development

© 2008 IBM Corporation



This is the baseline of this comparison: the modern day status of COBOL in the world.

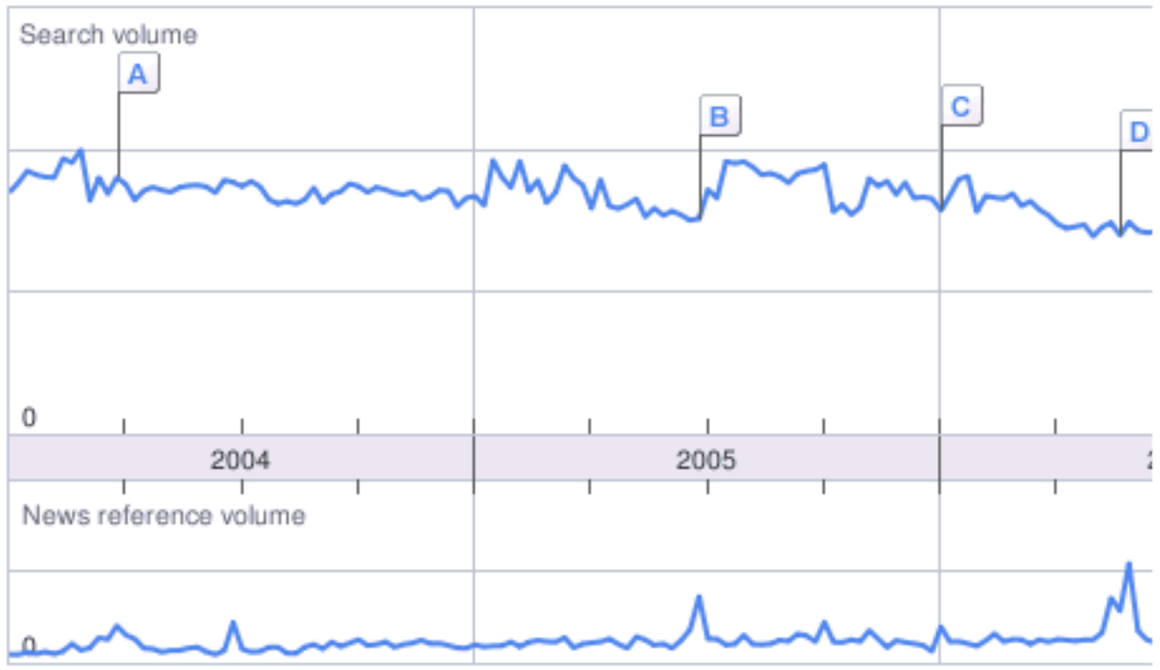
Cities | Regions | Languages

Top cities (normalized)

1. **Kizuki**, Japan
2. **Pune**, India
3. **Trivandrum**, India
4. **Hyderabad**, India
5. **Bangalore**, India
6. **Chennai**, India
7. **Yokohama**, Japan
8. **Chiyoda**, Japan
9. **Shibuya**, Japan
10. **Tokyo**, Japan

OSI certified

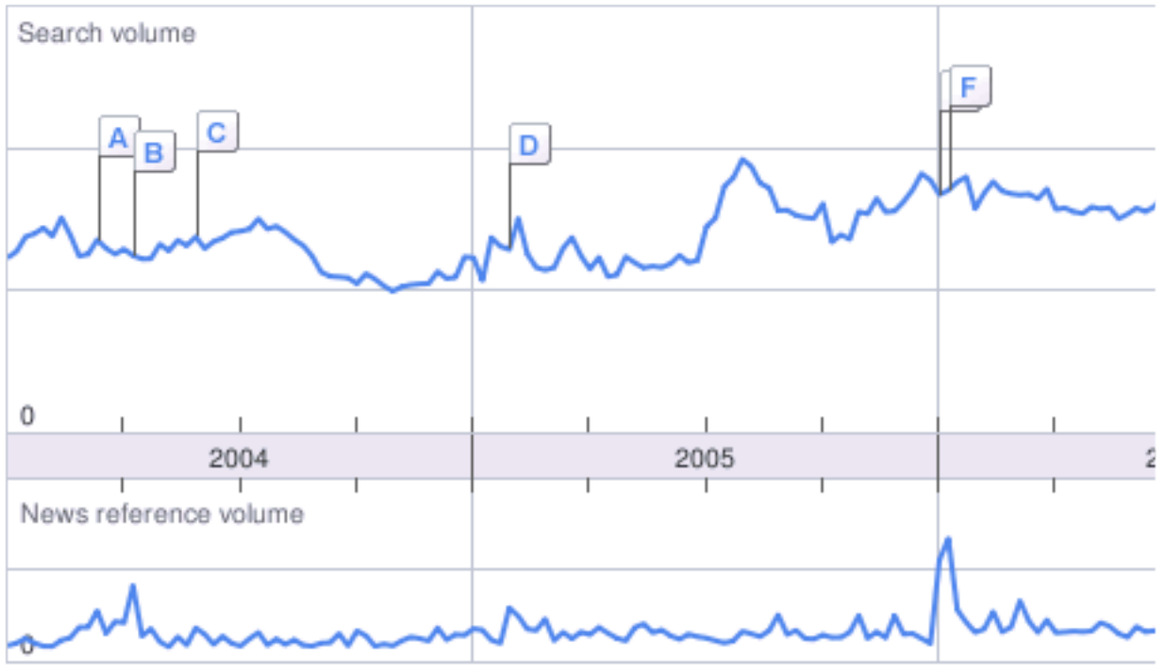
Java is queried most by outsourcing companies in India



Cities | Regions | Languages

Top cities (normalized)

1. **Bangalore**, India	
2. **Chennai**, India	
3. **Mumbai**, India	
4. **Singapore**, Singapore	
5. **Chiyoda**, Japan	
6. **Delhi**, India	
7. **Tokyo**, Japan	
8. **Krakow**, Poland	
9. **Warsaw**, Poland	
10. **Austin**, TX, USA	



Ruby is the most hyped oo scripting language at the moment, and the absolute winner at the moment if we measure by book sales. Here we see the trend showing most queries from the US West Coast.

Following the here presented theory this indicates interest from research communities and the fact that is is not yet accepted as proven technology.

Cities Regions Languages

Top cities (normalized)

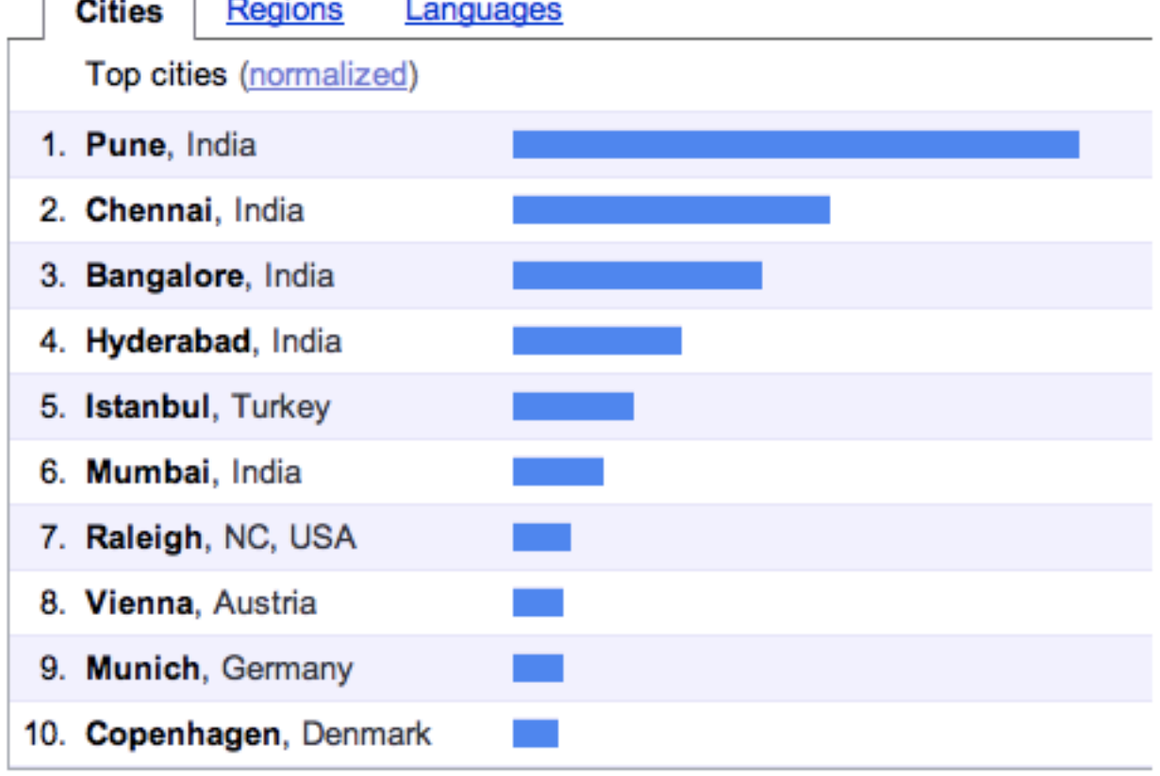
1.	**San Francisco**, CA, USA	
2.	**Pleasanton**, CA, USA	
3.	**Raleigh**, NC, USA	
4.	**Washington**, DC, USA	
5.	**New York**, NY, USA	
6.	**Rochester**, NY, USA	
7.	**Portland**, OR, USA	
8.	**Seattle**, WA, USA	
9.	**Cincinnati**, OH, USA	
10.	**Salt Lake City**, UT, USA	

OSI certified

Rexx is at the same time an established scripting language, with lots of queries from India, doubtlessly for its use as the glue for older, traditional apps that have been offshored, but also leading edge, with the more research-oriented, agile US companies querying it in Google.

My interpretation here is that there is a distinct possibility that the US and Europe queries concern ooRexx, while the offshoring country queries concern mostly Classic Rexx - or Mainframe Rexx.

There is of course no solid proof for this.



OSI certified

Peculiarities of NetRexx

- The Rexx Data type - implicit in other implementations, but not named due to untyped usage
- PARSE
- TRACE
- Arbitrary numeric precision & decimal arithmetic
- Concatenation by abuttal
- No reserved words
- Case insensitive
- Automates type selection and declaration
- Autogeneration of JavaBeans properties (with *properties indirect*)

- Philosophy: a language should be easy for users, not interpreter writers
- ‘No reserved words’ ensures that old programs are never broken by language evolution

© 2008 IBM Corporation

The Rexx Data Type

- This is where statically typechecked meets type-less
- A Rexx instance can be a number or a (Unicode) string of characters
- 3+4 is the same as “3” + “4”
- We can perform (arbitrary precision) arithmetic on it when it is a number
- The Rexx type keeps two representations under the covers (if needed)

- The Rexx way to handle decimal arithmetic ended up in Java and in IEEE 754r, implementation of **BigDecimal** actually written in NetRexx

- Automation inter-conversion with Java String class, char and char[] arrays, and numeric primitives (optional)

You can forego the language and use the Rexx Datatype, from the runtime package, in your Java source

Equally, you can decide not to use the Rexx type at all in your NetRexx source

© 2008 IBM Corporation

Numeric Precision

(**options binary** to avoid this and have Java primitive types as much as possible)

Rexx has arbitrary precision numbers as a standard - implemented in the runtime Rexx Datatype.

say 1/7

```
numeric digits 300

0.142857142857142857142857142857
142857142857142857142857142857142857
142857142857142857142857142857142857
142857142857142857142857142857142857
142857142857142857142857142857142857
142857142857142857142857142857142857
142857142857142857142857142857142857
142857142857142857142857142857142857
142857142857142857142857142857142857
142857142857142857142857142857
```


© 2008 IBM Corporation

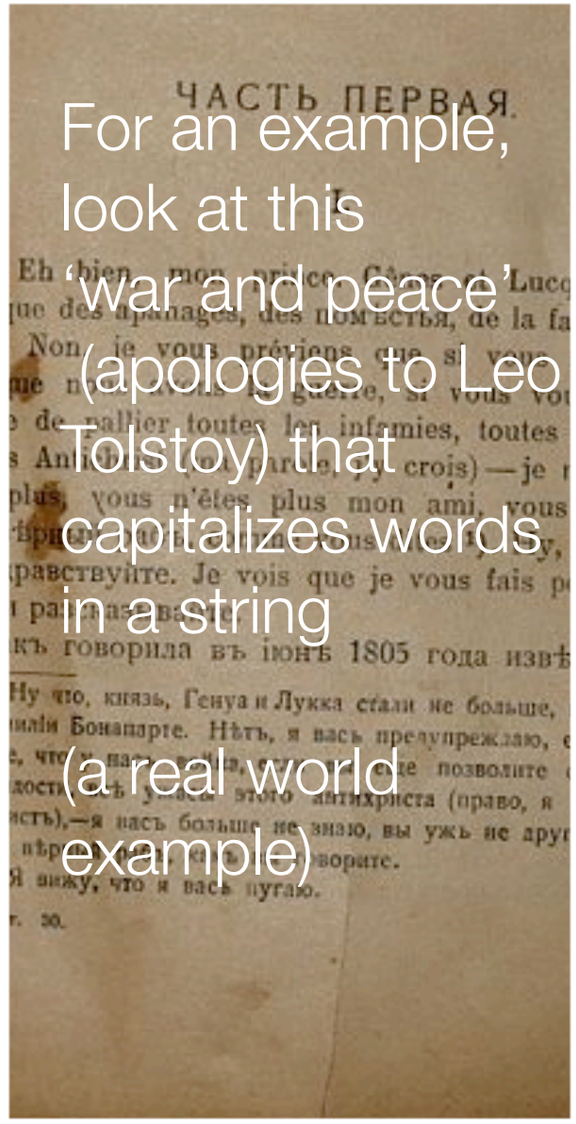
Parse

- not your standard regexp parser - it is template based
- can do lispy things

© 2008 IBM Corporation

```
import java.util.regex.*;

/**
 * Static methods to parse out words from a String
 *
 */
public class Words {

  /**
   * Count the number of words in the String
   *
   * @param str a string containing words that match the regular expression \w+
   * separated by \s+
   *
   * @return int the number of words in the string
   */
  public static int countWords( String str ) {
    String[] words = getWords(str);
    int numWords = words.length;
    return numWords;
  }

  /**
   * Gives a String array containing the parsed out words from the string. The
   * method uses the regular expression \s+ to split the string into words.
   *
   * @param str a string containing words that match the regular expression \w+
   * separated by \s+
   *
   * @return String[] containing the words
   */
  public static String[] getWords( String str ) {
    String[] words = java.util.regex.Pattern.compile("\\s+").split(str.trim());
    return words;
  }
```

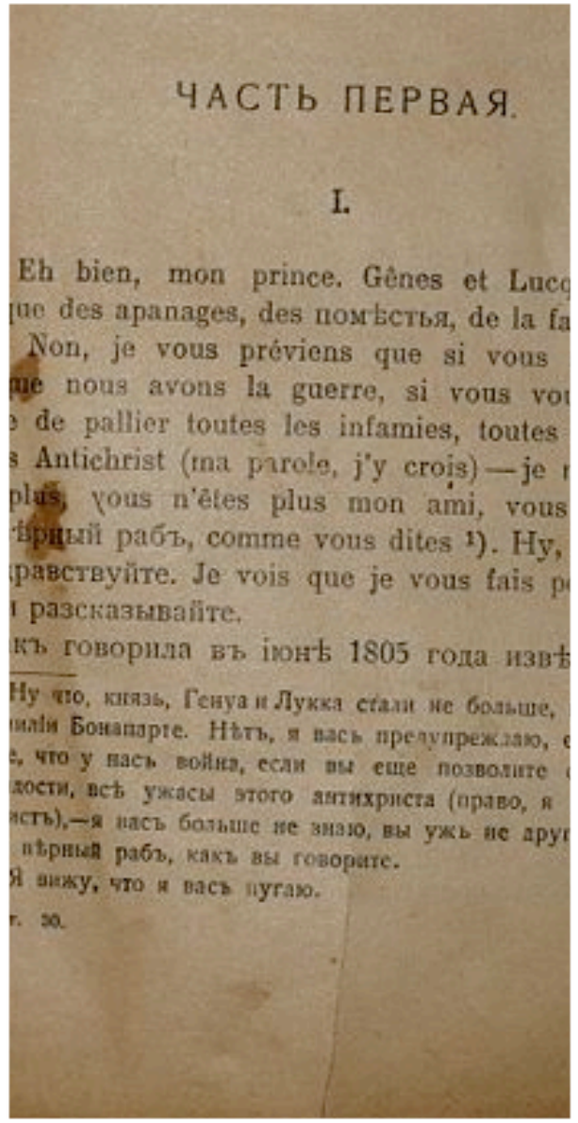
Java can be very ‘wordy’



For an example, look at this ‘war and peace’ (apologies to Leo Tolstoy) that capitalizes words in a string (a real world example)

```
/**
 * Capitalise the first letter of each word in the string. The method uses
 * countWords(String) and getWords(String).
 *
 * @param str a string containing words that match the regular expression \w+
 * separated by \s+
 *
 * @return String containing the original string with the first letter of each
 * word in uppercase.
 */
public static String capitalise( String str ) {
  String capitalised = null;
  int numWords = countWords(str);
  String[] words = getWords(str);
  for (int i = 0; i < numWords; i++) {
    StartingBuffer sb = new StringBuffer(words[i]);
    Character c = sb.charAt(0);
    sb.setCharAt(0, Character.toUpperCase(c));
    words[i] = sb.toString();
    if (capitalised == null) {
      capitalised = words[i];
    } else {
      capitalised = capitalised + " " + words[i];
    }
  }
  return capitalised;
 }

}
```

It goes actually on for another page



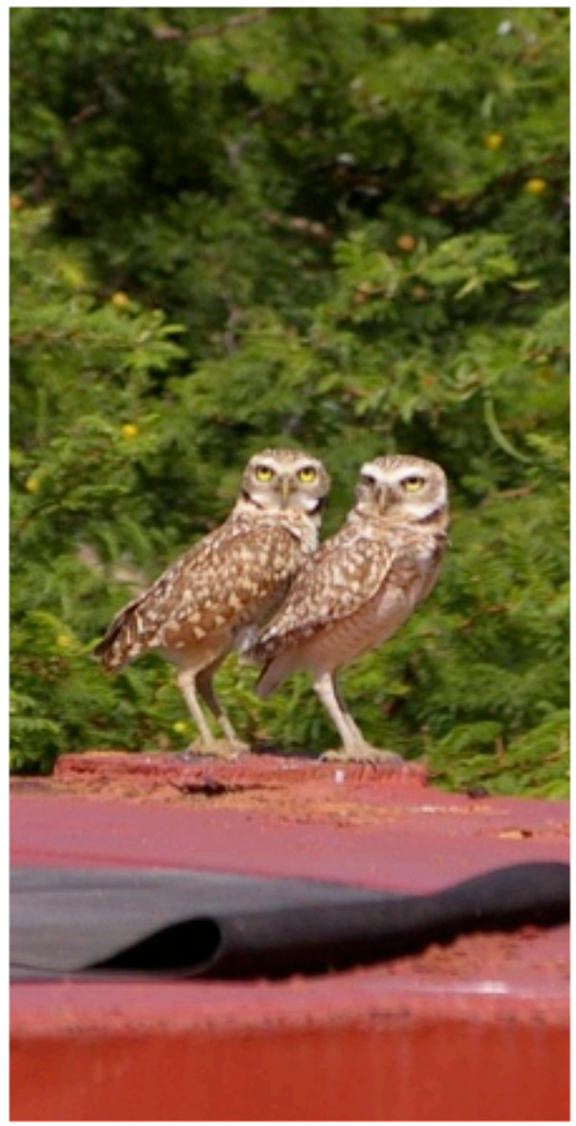
ЧАСТЬ ПЕРВАЯ.

I.

Eh bien, mon prince. Gênes et Luc
ue des apanages, des помѣстья, de la fa
Non, je vous préviens que si vous
ue nous avons la guerre, si vous vo
de pallier toutes les infamies, toutes
s Antichrist (ma parole, j'y crois) — je
plus, vous n'êtes plus mon ami, vous
ѣрный рабъ, comme vous dites [1]). Ну,
дравствуйте. Je vois que je vous fais
разсказывайте.
къ говорила въ іюнѣ 1805 года изв

Ну что, князь, Генуа и Лукка стали не больше,
ліи Бонапарте. Нѣтъ, я васъ предупреждаю,
, что у насъ война, если вы еще позволите
дости, всѣ ужасы этого антихриста (право, я
стъ),—я васъ больше не знаю, вы ужъ не дру
ѣрный рабъ, какъ вы говорите.
Я вижу, что я васъ пугаю.
г. 30.

```
cdr = “foo bar baz”
loop while cdr <> ‘’
    parse cdr car ‘ ‘ cdr
      line = line car.upper(1,1)
end
```



Built-in TRACE

- for you (and me) who still debug best using print statements - saves time
- adds them automatically during compile
- can leave them in and switch off during runtime
- best way to debug server type software
- can ‘trace var’ to keep a watchlist
- or ‘trace results’ to see results of expressions

```
class fact

  method main(args=String[]) static
    factorial(5)

  method factorial(number) static
    trace results
    if number = 0 then return 1
    else return number * factorial(number-1)
```

```
  --- fact.nrx
8 *=*     if number = 0
  >>> "0"
9 *=*     else
  *=*         return number * factorial(number-1)
  >>> "4"
8 *=*     if number = 0
  >>> "0"
9 *=*     else
  *=*         return number * factorial(number-1)
  >>> "3"
8 *=*     if number = 0
  >>> "0"
9 *=*     else
  *=*         return number * factorial(number-1)
  >>> "2"
8 *=*     if number = 0
  >>> "0"
9 *=*     else
  *=*         return number * factorial(number-1)
  >>> "1"
8 *=*     if number = 0
  >>> "0"
9 *=*     else
  *=*         return number * factorial(number-1)
  >>> "0"
8 *=*     if number = 0
  >>> "1"
  *=*                 then
  *=*                     return 1
  >>> "1"
  >>> "1"
  >>> "2"
  >>> "6"
  >>> "24"
  >>> "120"
```

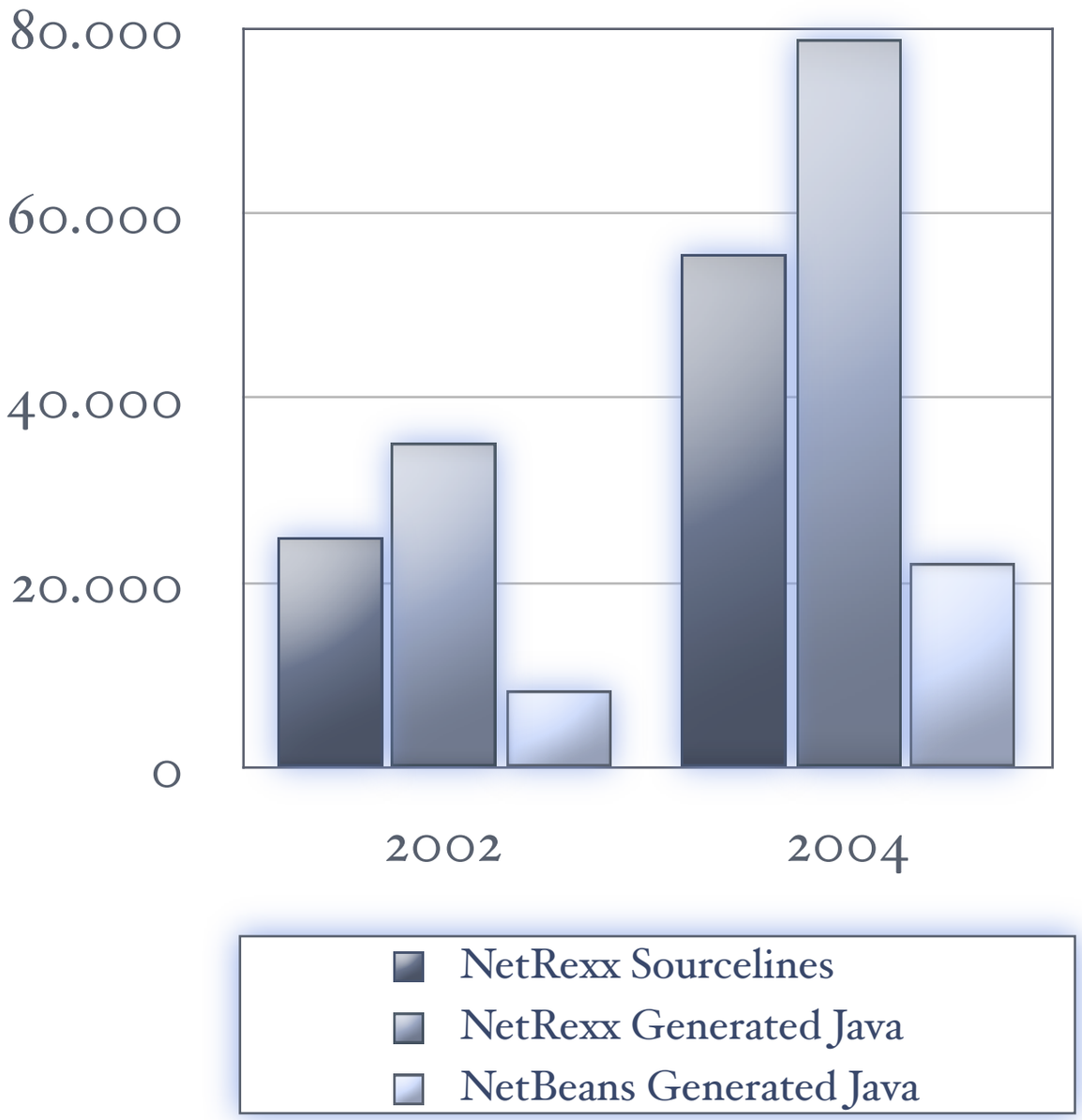

© 2008 IBM Corporation

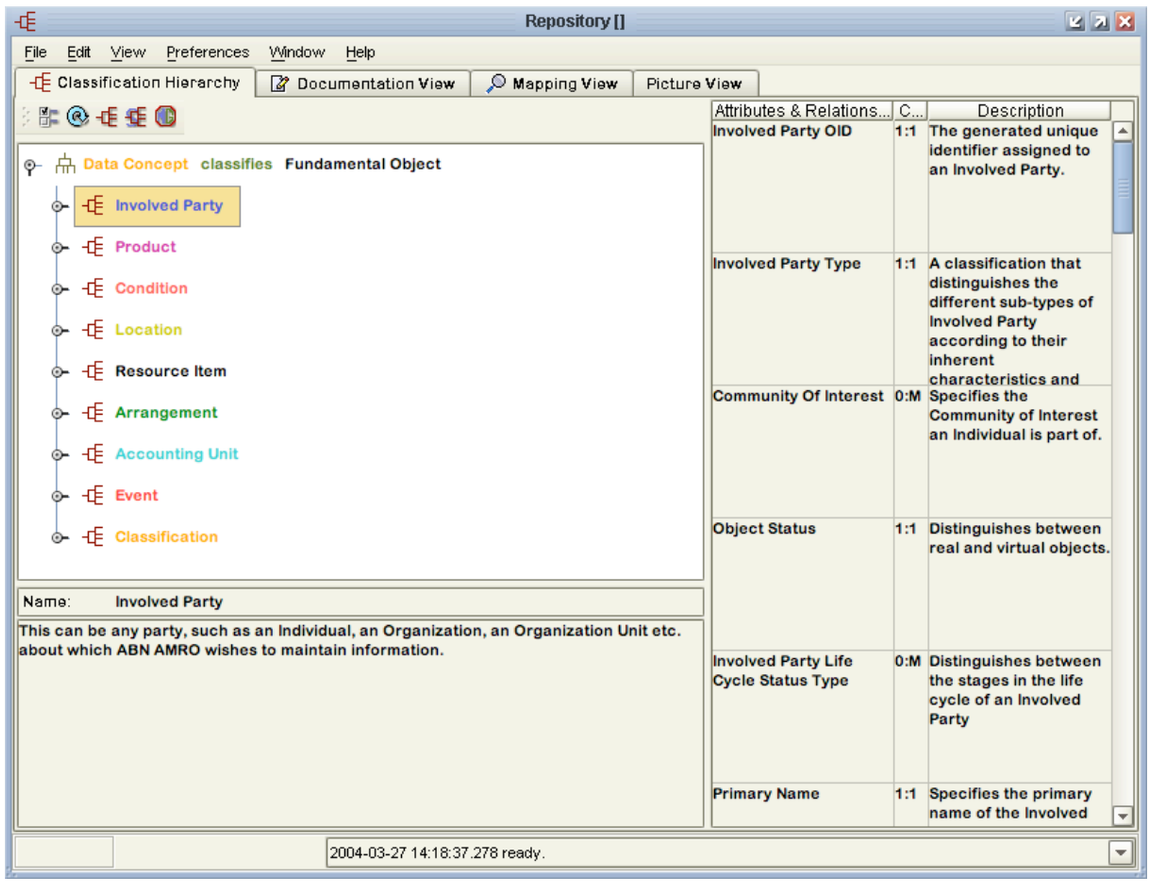
Automated Type Selection and Declaration

```
package com.abnamro.midms.util.ssl
import java.io.
import java.net.
import java.rmi.server.
import javax.net.ssl.
import java.security.KeyStore
import javax.security.cert.X509Certificate
-- mind: if this does not compile you probably do not have jsse.jar on your classpath
class RMISSLServerSocketFactory implements RMIServerSocketFactory, Serializable

  method  createServerSocket(port=int) returns ServerSocket signals IOException
    do
      -- set up key manager to do server authentication
      passphrase = char[] "passphrase".toCharArray()

      ctx         = SSLContext.getInstance("TLS")
      kmf         = KeyManagerFactory.getInstance("SunX509")
      ks          = KeyStore.getInstance("JKS")

      ks.load(ClassLoader.getSystemClassLoader().getResourceAsStream("dmskey"),passphrase)
      kmf.init(ks, passphrase)
      ctx.init(kmf.getKeyManagers(), null, null)
      ssf = ctx.getServerSocketFactory()

    catch e=Exception
      e.printStackTrace()
    end
    return ssf.createServerSocket(port)
```


© 2008 IBM Corporation

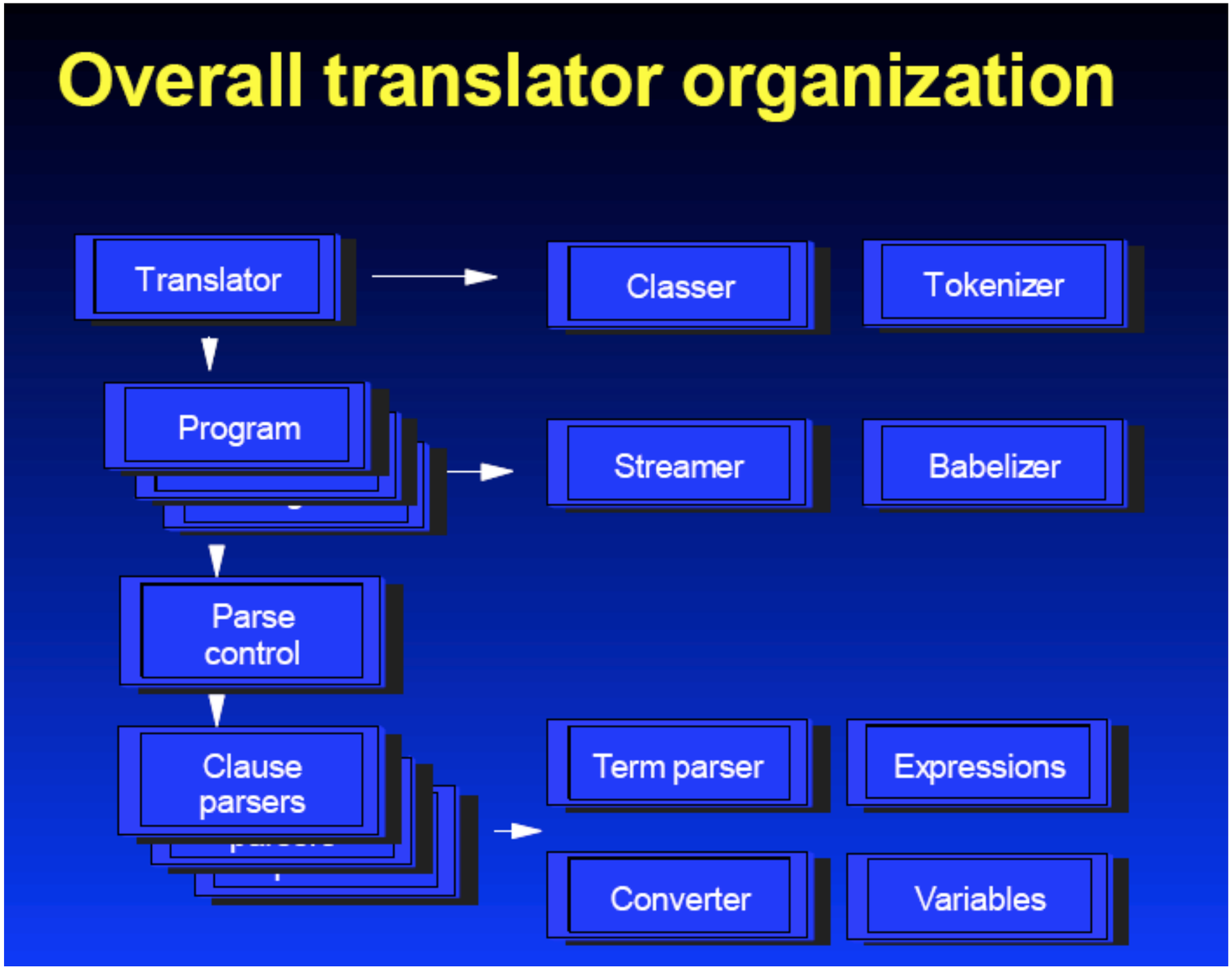
Saving ±40% of lexical tokens in your source





© 2008 IBM Corporation

Translation



© 2008 IBM Corporation

Parsing

- No upfront parsing - handwritten lexer & parser combo does ‘on demand’ parsing
- Parse on a ‘Clause’ base
- Stops quickly after errors in all three phases
  - Clear error messages, ‘land on them’ in IDE’s (or Emacs in my case)

© 2008 IBM Corporation

Cost based conversions - looking for a method match

1. Candidate methods in the class are selected. To be a candidate method:
• the method must have the same name as the method invocation (independent of the case of the letters of the name)
• the method must have the same number of arguments as the method invocation (or more arguments, provided that the remainder are shown as optional in the method definition)
• it must be possible to assign the result of each argument expression to the type of the corresponding argument in the method definition (if strict type checking is in effect, the types must match exactly).
2. If there are no candidate methods then the search is complete; the method was not found.
3. If there is just one candidate method, that method is used; the search is complete.

This seems something other languages also use, and might as well be a part of the JVM functionality

And even this can lead to an ambiguity (which is an error that will be reported)

4. If there is more than one candidate method, the sum of the costs of the conversions from the type of each argument expression to the type of the corresponding argument defined for the method is computed for each candidate method.

© 2008 IBM Corporation

Calling the compiler

- Finding the compiler: here it is a fact of life that sometimes this sits in tools.jar, sometimes rt.jar, sometimes classes.jar
- We are doing some searching for the usual suspects, but for some platforms, in the end, the user needs to know where it is and put it on the classpath (Apple users needed /System/Library/Frameworks/JavaVM.framework/Classes/classes.jar" on the classpath until V3.00, (which is not out yet))
- can use alternative compilers, jikes (is that still around), or ibm jvm compilers

© 2008 IBM Corporation

Use in *scripting* mode

- Compiler adds boilerplate when needed
- and leaves it out when already there

```
say "hello JVM languages Summit!"
```

generates:

```
/* Generated from 'hello.nrx' 22 Sep 2008 19:57:00 [v3.00] */
/* Options: Crossref Decimal Format Java Logo Replace Trace2 Verbose3 */

public class hello{
 private static final java.lang.String $0="hello.nrx";


 public static void main(java.lang.String $0s[]){
  netrexx.lang.RexxIO.Say("hello JVM languages Summit!");
  return;}

 private hello(){return;}
 }
```


© 2008 IBM Corporation

Runtime

NetRexxR.jar is currently 45463 bytes

Contains

- the **Rexx** datatype
- console I/O like **say** and **ask**
- Some Exceptions like BadNumericException - consequence of calling number methods on Rexx strings
- Support for **Trace**
- Support for **Parse**

This is still a reasonable size for applet support

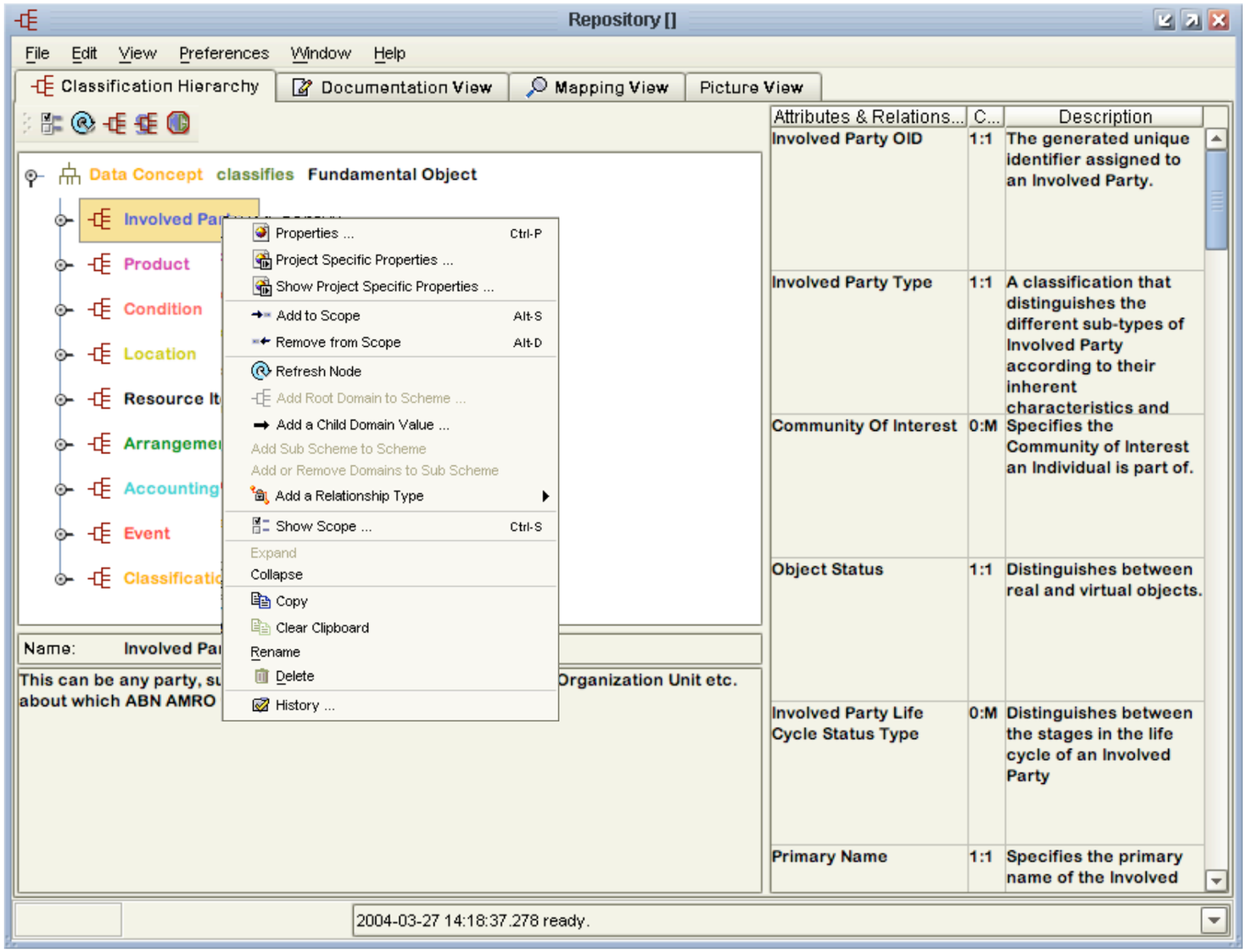
© 2008 IBM Corporation

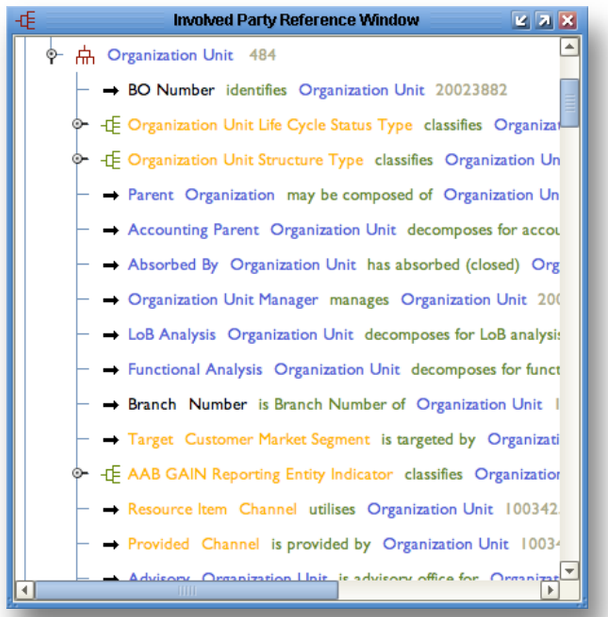
Easy integration with all existing Java infra

- Successfully and easily use
  - Java Collection Classes
  - NetBeans
  - Antlr
  - Hibernate
  - JSF
  - You name it.

© 2008 IBM Corporation

Swing GUIs using NetBeans





© 2008 IBM Corporation

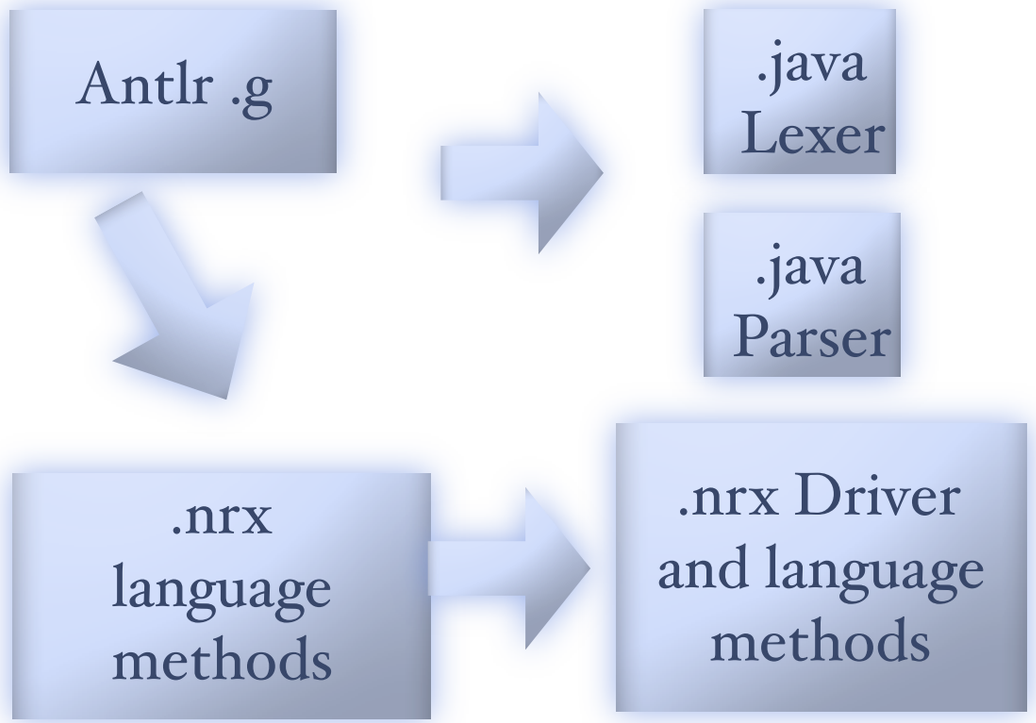
Antlr - specifying the interpreter for a DSL (called ***bint***)

bint.g

```
header
{
    package com.abnamro.midms.bint;
}

{
    import java.io.*;
    import java.rmi.*;
}

class bintParser extends Parser;
options {
  k = 2;
  exportVocab=bint;
  codeGenMakeSwitchThreshold = 2;
  codeGenBitsetTestThreshold = 3;
  buildAST = false;
}

{
 public bintInterface bintInstance ;
}

program
    : (statement)+ EOF
    ;

statement
```

bint.nrx

```
class bint implements bintInterface

  method parseFile(s=InputStream)
    do
      lexer = bintLexer(s)
      parser = bintParser(lexer)
      parser.bintInstance = this;
      parser.program()

    catch e = Exception
      System.err.println("parser exception: " e)
      e.printStackTrace()
      errorText = e.getMessage()
      this.rollBackAndQuit()
    end -- do
    return
```



© 2008 IBM Corporation

Java Server Faces

```
<f:view>
<h:form id="AddSystemForm">

  <h:inputText id="PrimaryNameIn" value="#{addSystem.primaryName}" />
<b>  <h:outputText id="PrimaryNameOut" value="System Name"/></b><br>

  <h:inputText id="UriIn" value="#{addSystem.systemAddress}" />
<b>  <h:outputText id="UriOut" value="IP Address or DNS Name"/></b>

  <h:commandButton action="#{addSystem.add}" value="Add System" /><br><br>
  <h:commandButton action="#{addSystem.goback}" value="Return to Subscription" /><b
r>
</h:form>
</f:view>
```

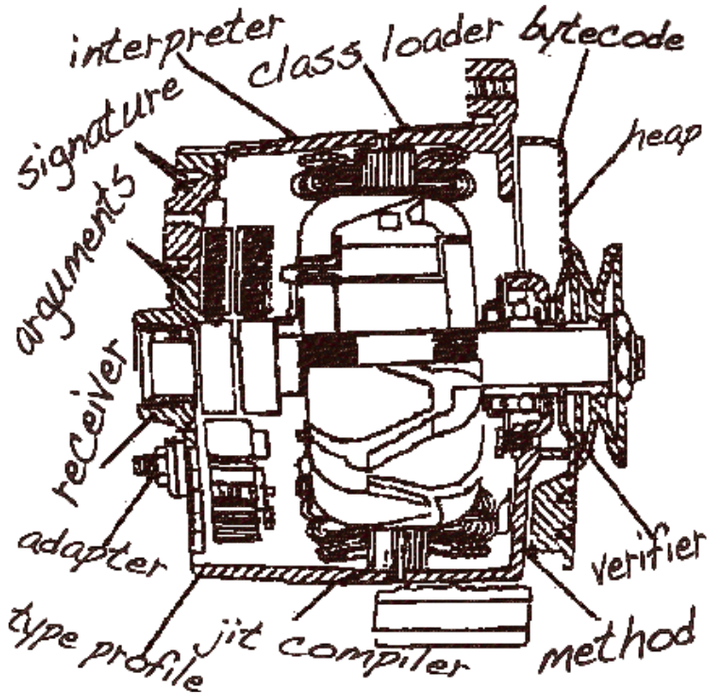
AddSystem.nrx: /Volumes/Workspace/src/com/abnamro/crd/admin/AddSystem.nrx

```
method setSystemAddress(s=String)
  this.systemAddress = s

method getSystemAddress() returns String
  return this.systemAddress

method add() returns String
  transaction = this.session_.beginTransaction()
  uri_ = UniformResourceLocator()
  if this.getSystemAddress() = null then this.setSystemAddress('localhost')
  uri_.setPrimaryName("http://"this.getSystemAddress()":8080/invoker/JNDIFactory
")
  this.session_.save(uri_)
  sip = SystemImplementationService()
  if this.getPrimaryName= null then this.setPrimaryName('dummy name')
  sip.setPrimaryName(this.getPrimaryName())
  sip.setUniformResourceLocator(uri_)
  this.session_.save(sip)
  transaction.commit()
  say CRDutil.getUser() 'added a SystemImplementationService at' this.getSystemA
ddress()
  return "success"
```

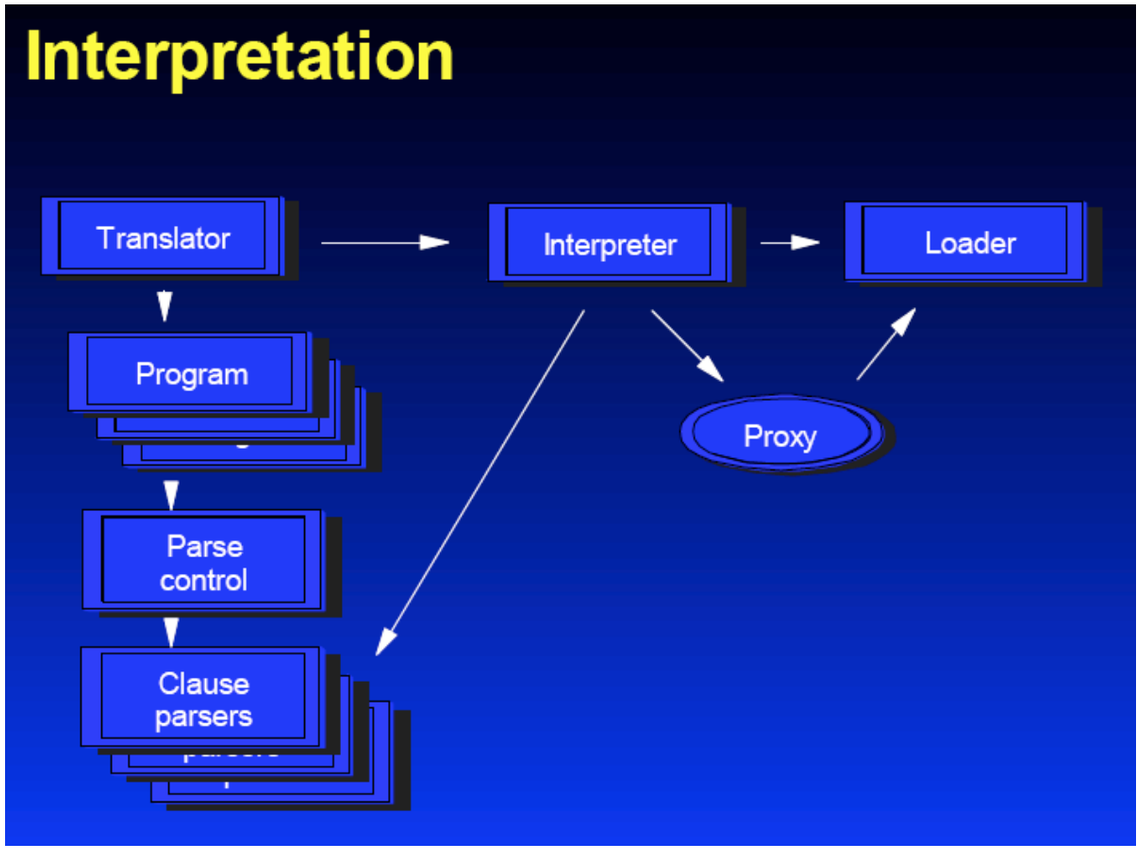

© 2008 IBM Corporation

Interpreter

- All statement translator classes have an **interpret** method.



© SUN



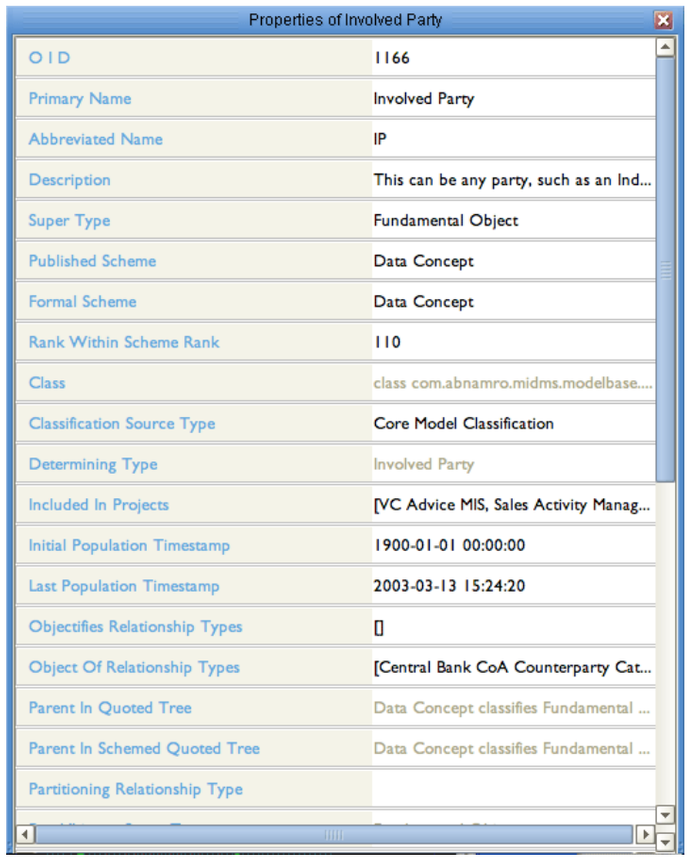
In the interpreted mode, for each class a proxy (‘stub’) is created, that contains method bodies that just return, and the properties like in a ‘real’ class. The proxy is constructed from a byte array;

When the method is called (through reflection) the interpreter executes its body as read from source.

© 2008 IBM Corporation

Tied to Java object model and staticness

```
do
  /* check whether the getter returns an object instance. if it
   * does, we pass it an EditorVisitor instance that handles the
   * editing this of course polymorphically with double dispatch
   * on the indirect object.
   */
  invokeResult = this.globalGetter.getMethod().invoke(this.globalObject, null)
  /* if the result from the Getter invocation is null, we
   * instantiate a new object and also have it accept an
   * editorvisitor.
   */
  if invokeresult = null then
    do
      do
        cls = Class.forName(this.globalGetter.getMethod().getReturnType().getName())
        clz = cls.newInstance()
        (Visited clz).accept(edV)
        edV.getEditor.setFont(this.dialogFont)
        ppp.validate()
        this.panel.validate()
      catch Exception
        say "Exception instantiating object-to-be-edited"
      end
    end
  else
    do
      (Visited invokeResult).accept(edV)
      edV.getEditor.setFont(this.dialogFont)
      ppp.validate()
      this.panel.validate()
```



A generic object editor

© 2008 IBM Corporation

Visitor - static typing at its most verbose

When we hit this manifestation of static typing we clearly liked to have a more dynamic language

On the other hand, we would not like to have the performance loss that seems to be the price when we look at the dynamic language implementations for some other JVM languages.

Something more advanced for multiple dispatch would be most welcome

"mechanical and boring transformations of code"

```
class Visitor abstract
  method visit(a  = AbbreviatedName) abstract
  method visit(c  = Classification) abstract
  method visit(c  = ClassificationSet) abstract
  method visit(c  = TypeSet) abstract
  method visit(s  = SchemeSet) abstract
  method visit(s  = SubSchemeSet) abstract
  method visit(d  = Descriptor) abstract
  method visit(d  = DomainOnly) abstract
  method visit(e  = com.abnamro.midms.modelbase.Ever
  method visit(f  = FundamentalObject) abstract
  method visit(i  = Identifier) abstract
  method visit(i  = InstanceOnly) abstract
  method visit(i  = InstanceOnlyScheme) abstract
  method visit(i  = InvolvedParty) abstract
  method visit(c  = CommunityOfInterest) abstract
  method visit(i  = Individual) abstract
  method visit(e  = EmploymentPosition) abstract
  method visit(o  = Organization) abstract
  method visit(o  = OrganizationUnit) abstract
  method visit(s  = StaffMember) abstract
  method visit(is = IndividualSet) abstract
  method visit(es = EmploymentPositionSet) abstract
  method visit(is = InvolvedPartySet) abstract
  method visit(os = OrganizationUnitSet) abstract
  method visit(u  = Userid) abstract
  method visit(r  = RecommendedIdentifier) abstract
  method visit(e  = Password) abstract
  method visit(i2 = ISOCountry2CharacterCode) abstra
  method visit(i3 = ISOCountry3CharacterCode) abstra
  method visit(m  = Measure) abstract
  method visit(m  = ModelComponent)
  method visit(n  = Name) abstract
  method visit(n  = com.abnamro.midms.modelbase.Numb
  method visit(o  = Oid) abstract
  method visit(o  = com.abnamro.midms.modelbase.Obje
```


© 2008 IBM Corporation

Performance

- Without going into microbenchmark discussions, NetRexx is a lot faster than the competition - probably as a result of using plain Java source (so leveraging javac) and a minimal runtime without any proxying of objects, and the ‘binary’ option, which even leaves much of the Rexx runtime untouched if Java primitive types can be used

- The interpreter is a bit slower, but not much so - and we win that back in development cycle turnaround.

- Arguably, missing dynamic language features like open classes is a pain, specially in regard of the full support of this in Open Object Rexx

© 2008 IBM Corporation

NetRexx is completely written in NetRexx

The language is bootstrapped (starting from Classic Rexx)

A working compiler is needed to compile the compiler - save one!

Disadvantages

- Some problems become slightly non-trivial, like changing package names of the compiler - opportunities to shoot oneself in the foot
- For example the compiler package is called **COM.ibm.netrexx.process** - Sun told us to uppercase domain suffix in the early days.

Advantages

It can be built on every platform where there is a working Java

- Structure of the language translator is clear - interpreter like - and readable. Especially if you are into writing NetRexx

© 2008 IBM Corporation

Building mixed source and JavaDoc

```
COMPILE_COMMAND = java -Xms128M -Xmx256M COM.ibm.netrexx.process.NetRexxC

.nrx.class:
        $(COMPILE_COMMAND) $< -comments -sourcedir -time -keep -replace -pfpn -format -warnexit0
        mv $*.java.keep $*.java

NRX_SRC          := $(wildcard *.nrx)
NRX_OBJS         := $(NRX_SRC:.nrx=.class)
JAVA_SRC         := $(wildcard *.java)
JAVA_OBJS        := $(JAVA_SRC:.java=.class)
DOCPATH          :=
DOCCLASSPATH     :=
DOCTITLE         :=
WINDOWTITLE      :=

.SUFFIXES: .nrx .nry .njp .class .skel .xsl .java .pl

#
# target all compiles the netrexx and java code
#
all::   $(NRX_OBJS) $(JAVA_OBJS)

#
# target clean removes compiled products
#
.PHONY: clean
clean:
        rm -f *.class
        rm -f *.crossref
        rm -f *.bak
        find . -name "*.nrx" | awk '{$$2 = $$1 ; sub ( /\.nrx/,".java",$$1 ) ; print $$1 }' | xargs rm -f

.PHONY: doc
doc:
        mkdir -p $(DOCPATH)
        javadoc -J-Xmx128M -classpath "$(DOCCLASSPATH)" -private -author -version -breakiterator -use -d $(DOCPATH
) -bottom $(BOTTOM) -header $(HEADER) -windowtitle $(WINDOWTITLE) -doctitle $(DOCTITLE)
```


© 2008 IBM Corporation

Open Sourcing

Follow IBM's open sourcing process - OSSC

Prepare source code for release

Tidy up & Package, build procedure, arrange testing suite

Formal handover to RexxLA - The Rexx Language Association

© 2008 IBM Corporation

Recent Additions

As the Scala compiler does, the compiler will have an option to stay resident in memory

This is in the form of an compiler server, now integrated in the main compiler source

Need to solve the problem of changed interfaces captured in compiler server state

Runs now on a lot more OS Platforms/ JVM implementations without classpath changes

© 2008 IBM Corporation

Possibles

IDE Support (but I don't use them much)

Annotations - this is really missed for integration with other tools
- I wish it was just put in comments and not added to Java syntax

Support for Generics - nah. Well, maybe later - maybe without syntax

Direct to Bytecode compiler - if enough benefits and/ or necessities are seen

(cross language-)Dependency cycle resolving - like javac does for .java

Getting closer to Open Object Rexx

- Especially the dynamic aspects - adding classes, methods at runtime
- Might need to use the InvokeDynamic stuff here - this will bump up the minimum Java version req from 1.1.2 to 7!

© 2008 IBM Corporation

Object Rexx on the JVM

Presented this last year at RexxLA

Unfortunately, this was an Object Rexx talk - in todays NetRexx they are equally hard to use

- metaclasses
- duck typing
- the unknown method

Planned for somewhere in 2010

© 2008 IBM Corporation

Publications

Mike Cowlishaw, **The NetRexx Language**, Prentice Hall, ISBN 0-13-806332-X

Heuchert, Haesbrouck,Furukawa, Wahli, **Creating Java Applications Using NetRexx**, IBM 1997, http://www.redbooks.ibm.com/abstracts/sg242216.html

Mike Cowlishaw, **The NetRexx Interpreter**, RexxLA/WarpTech 2000, (netrexxi.pdf)

© 2008 IBM Corporation

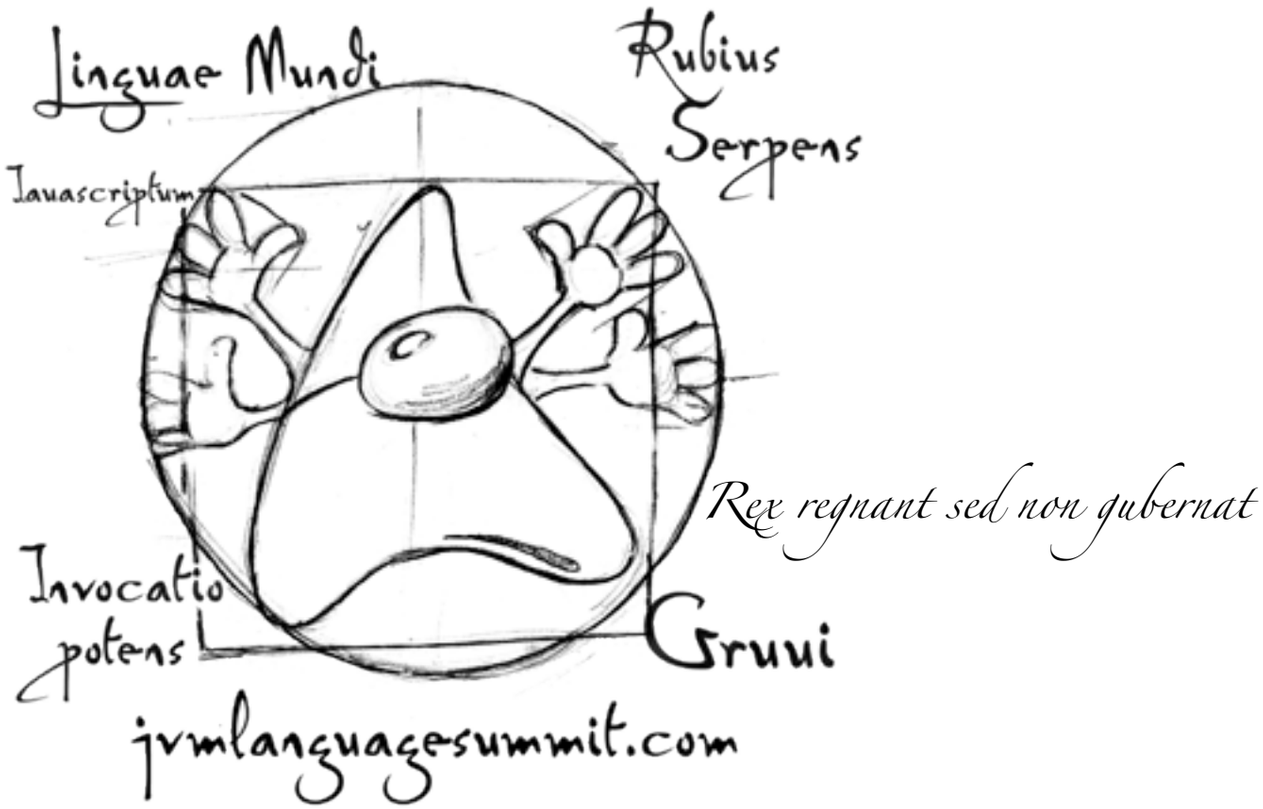
Contact

www.netrexx.org will be reactivated - starting with serving tools (emacs & vi modes), documents and other information - watch that space!

rvjansen@xs4all.nl

rene.vincent.jansen@nl.ibm.com

Thanks for your attention!



“strong typing does not need more typing”

© 2008 IBM Corporation