"The 2009 Edition of BSF4Rexx"

Rony G. Flatscher (Rony.Flatscher@wu.ac.at), WU Vienna
"The 2009 International Rexx Symposium", Chilworth, England, Great Britain
May 18th – May 21st 2009.

Abstract: *This article describes briefly the functionality made available in the Vienna Version of BSF4Rexx, a package that bridges Rexx and Java. After identifying shortcomings that are linked to missing (concurrency) APIs for Rexx, possible solutions for removing these shortcomings in the Vienna Version of BSF4Rexx that would become possible with the new release of Open Object Rexx 4.0 are discussed and functions with their signatures are devised, that would overcome the identified shortcomings.*

1 Introduction

This article briefly introduces the development of the "Bean Scripting Framework for Rexx (BSF4Rexx)" in section 2 and its various versions since the first package was created in 2001, highlighting the most important new features each of these versions brought along.

Following this, the shortcomings of the "Vienna" version of BSF4Rexx are discussed in section 3, which could only be overcome, if Rexx interpreters would supply APIs that allow for concurrent, direct interaction with running Rexx programs. For 2009 a new version of Open Object Rexx ("ooRexx", version 4.0) was announced which would possess a new kernel, that would allow for concurrent callbacks into running Rexx programs, making it possible to overcome the identified shortcomings (section 4).

From the sketching of the BSF4Rexx shortcomings and the possibilities the announced Open Object Rexx interpreter would allow for, new concepts for BSF4Rexx are devised in section 5, removing the shortcomings of the Vienna version altogether.

2 BSF4Rexx: From Essen to Vienna

Life of BSF4Rexx [W3BSF4Rexx] started as a seminar assignment while staying at the University of Essen, Germany, in the wintersemester 2000/01. The supervised student, Peter Kalender, was given the task to create a minimal proof of concept, that it is in principle possible to create a Rexx engine for the "Bean Scripting Framework (BSF)". BSF was originally created as an open source project by IBM at

the end of the 90s and was later handed over to the Apache Software Foundation (ASF) [W3BSF].

Based on [Kal01]'s initial work the BSF engine for Rexx has since been rewritten and the functionality drastically extended over time by the author [Flat01]. Later, while teaching and researching at the University of Augsburg work on the basic functionality of BSF4Rexx was concluded, yielding the "Augsburg" version of BSF4Rexx in 2003. This version allowed Rexx programs to demand load Java, in order to become able to create and interact with Java objects.

The "Augsburg" version got introduced officially at the "2003 International Rexx Symposium", where Pierre G. Richard's (author of Rexx for Palm) feedback ("need to prepend each argument with its type is unRexxish") caused a new goal: improve the support for Rexx following the "human-centricness" paradigm as set forth by the author of Rexx, Mike F. Cowlishaw [Cow90].

Subsequently the work on BSF4Rexx yielded a version which removed the need to indicate the Java type of arguments supplied by Rexx programmers for Java methods, drastically simplifying the interface for them. This feature and in addition the creation of the Rexx support for OpenOffice.org (OOo) [Flat05a, Flat05b], taking advantage of OOo's Java interfaces, were carried out after returning to the WU Vienna, Austria, yielding the "Vienna" version of BSF4Rexx in 2006.

Using BSF4Rexx, any Java application can use Rexx as its scripting language, being able to supply arguments and fetching return values from the Rexx scripts. Any Rexx program using the BSF4Rexx bridge is able to interact with Java objects and take advantage of all of the functionality that the class library of the "Java Runtime Environment (JRE)" offers. JRE is usually installed on any desktop computer.

3 Shortcomings of the Vienna Version

The implementation of the Rexx engine for ASF's Bean Scripting Framework (BSF) covers practically all features that this framework offers. However, due to the restrictions imposed by the Rexx SAA[1] APIs, which were devised almost 25 years ago, some shortcomings have to be observed that cannot be overcome without additional, new APIs for interfacing with Rexx (objects) concurrently:

1. *Creating ooRexx proxy objects for Java.* It is not possible to allow Java to directly address Rexx objects and to send Rexx messages to them, possibly supplying arguments and fetching return values.

[1] "SAA" is the acronym for IBM's "System Application Architecture" that was created in order to standardize all of IBM's software development, including the user interface design (I.e. the "CUA", common user access, SAA specifications).

2. *Real-time handling of Java events.* Although one can use BSF4Rexx to collect Java event objects and to later analyze the event objects in an event loop, one is not able to forward the firing of an event synchronously to a Rexx object. As a result it is not possible to cancel the handling of Java events right when the event got fired.
3. *Allowing Rexx methods to implement Java interface methods.* Java interfaces allow Java developers to define types that methods need to be implemented using any concrete Java class that claims to implement the interface. It is not possible to implement such methods in form of ooRexx methods. Therefore, Java frameworks that employ Java interfaces (e.g., the Java XML parsing using SAX – cf. package `org.xml.sax` – and DOM – cf. package `org.w3c.dom`, or the interface `java.io.FileFilter`) cannot be combined with Rexx.
4. *Allowing Rexx methods to implement abstract Java methods.* Java abstract classes allow Java developers to define types that abstract methods need to be implemented in subclasses. It is not possible to implement such methods in form of ooRexx methods. Therefore, Java frameworks that employ Java abstract classes (e.g., the Java class `javax.swing.filechooser.FileFilter` for filtering the files to be choosable from a list) cannot be implemented with Rexx.
5. *Communicating Rexx conditions to Java.* Currently, Java can recognize that a Rexx condition has occurred and a lot of effort has been invested in being able to at least communicate the Rexx error message text to Java.[2] It is, however, not possible to make the Rexx condition object available to Java and allow interacting with it by sending Rexx messages from Java.
6. *Throwing Specific Java Exceptions.* Rexx programmers cannot throw specific Java exceptions.

4 New Features of ooRexx 4.0

The new version of Open Object Rexx (ooRexx) 4.0, slated for a release in 2009 possesses a new kernel, which had been developed by the author of IBM's product Object REXX, Rick McGuire, who joined the open source ooRexx project for over five years.

The documentation of the ooRexx APIs in the PDF book "`rexxpg.pdf`" describes in the section "Chapter 10. Classic Rexx Application Programming Interfaces" the Rexx

[2] To be able to fetch the Rexx error message a RXSIO exit had to be created that buffers the last three lines that Rexx outputs to the console. In the case of a Rexx condition this buffer is then used to create an informative text for the Java side.

SAA invocation function `RexxStart()` which had been the only means of invoking Rexx programs (as a whole).

The new ooRexx 4.0 APIs get documented in chapter 9, "Rexx C++ Application Programming Interfaces", documenting among the approximately 130 new functions the following new invocation functions and functions for creating and running Rexx programs as well as sending Rexx messages to individual Rexx objects:

- `RexxCreateInterpreter()`: this function allows to create one or more Rexx interpreter instances, each individually configurable, sharing the `.environment` directory, but having separate `.local` directories. Each interpreter instance is able to concurrently execute any number of Rexx programs.
- `LoadPackage()`, `LoadPackageFromData()`, `CallProgram()`, `NewRoutine()`, `CallRoutine()`: these functions allow the loading of or calling of ooRexx packages (program files) from files or strings at runtime, as well as creating routines from strings and calling them thereafter.
- `SendMessage()`, `SendMessage0()`, `SendMessage1()`, `SendMessage2()`: these functions allow to send messages to ooRexx objects, using either an array of arguments, or for the convenience of the C++ programmer a message without an argument, with one or two arguments.
- `Halt()`, `HaltThread()`: these functions allow to halt all threads of a Rexx instance or a particular Rexx thread.
- `Terminate()`: terminates a Rexx interpreter instance.

With all of the new APIs it becomes possible to use all aspects of the ooRexx interpreter directly from C++. ooRexx 4.0 extends the notion of external Rexx functions to Rexx routines that may be implemented in native code instead of Rexx instruction, and carries this concept further to allow methods in ooRexx classes to be implementable in native code as well.

Given this wealth of new application programming interfaces in the ooRexx interpreter, many new features become possible to be implemented for ooRexx 4.0.

5 Devising New Features for BSF4Rexx

Studying the new APIs of ooRexx 4.0 and analyzing the sthortcomings of the Vienna version of BSF4Rexx, it has become clear, that the shortcomings could be addressed with a new implementation of the kernel of the shared/dynamic link library of BSF4Rexx (written in C++), which would take advantage of the new APIs. Whatever companion logic was needed at the Java side needs to be implemented in all of the

respective Java classes that comprise BSF4Rexx, as well as the camouflaging support in the `BSF.CLS` Rexx package/module.

Relating to chapter 3, "Shortcomings of the Vienna Version", the following sections introduce features that allow for solving all of the identified shortcomings.

5.1 Feature: Rexx Proxy

In order to allow Java to interact with Rexx objects it is necessary to create an infrastructure that allows Java objects to be created and interacted with that serve as proxies for Rexx objects. To do so, it is necessary to create a registry of Rexx objects in the BSF4Rexx native layer and glue it to a Java peer, which serves as a proxy. To communicate this fact the Java class should be called `RexxProxy` and possess specific methods that makes it easy for Java programmers to send messages to Rexx objects via the `RexxProxy` Java objects.

If a Rexx programmer needs to make a Rexx object available for Java to interact with, then it should be possible to create a Java peer object, supplying the Rexx object it should represent. Then, the Rexx programmer may use such a `RexxProxy` object as an argument to any Java method that can process a `RexxProxy` as an argument.

Analyzing possible use-cases it may be the case, that Rexx programmers may need a means of storing and retrieving information pertaining to the `RexxProxy`. In order to allow for such a functionality an optional Rexx slot argument should be possible that could be any Rexx object and which should be sent back along with any Rexx message that a Java programmer may wish to send to the Rexx object.

5.1.1 Java Class "RexxProxy"

The Java class `RexxProxy` needs to be instantiated at the native layer, i.e. in the shared/dynamic link library bridging Rexx with Java. The class needs at least a field for storing the reference to the Rexx object it represents, one field that allows for storing the optional slot argument and contain at least one method `sendMessage`, which allows sending a Rexx message with optional arguments to the Rexx object.

As a result, if Rexx programs return values to Java, the native layer becomes able to wrap Rexx objects other than string objects as Java `RexxProxy` objects, enabling Java programs to directly interact with the Rexx objects.

Creating explicitly `RexxProxy` objects should be restricted to the Rexx programmer and the native layer as a Java programmer has no direct access to a Rexx object. For Rexx programmers an external Rexx function can be devised that allows for the

creation of RexxProxy instances:

```
BsfCreateRexxProxy(rexxObject[,slot])
```

The external Rexx function BsfCreateRexxProxy() expects a mandatory Rexx object, an optional slot argument, which may be any Rexx object, and returns a Java RexxProxy object representing the rexxObject on the Java side.

Java programmers then are able to send the rexxObject Rexx messages, that may be accompanied with arguments using one of the following methods defined for this purpose in the org.rexxla.bsf.engines.rexx.RexxProxy class:

- sendMessage(String messageName, Object [] arguments): sends messageName to the rexxObject supplying the given arguments.
- sendMessage0(String messageName): convenience method which sends messageName to the rexxObject without an argument (note the trailing "0").
- sendMessage1(String messageName, Object arg1): convenience method which sends messageName to the rexxObject supplyinge one (note the trailing "1") argument.
- sendMessage2(String messageName, Object arg1, Object arg2): convenience method which sends messageName to the rexxObject supplyinge two (note the trailing "2") arguments.
- sendMessage3(...), sendMessage4(...), sendMessage5(...): convenience methods to send messages to rexxObject supplying three, four or five arguments, respectively.

If a slot argument was supplied at RexxProxy creation time, then the messages sent to the rexxObject get an additional argument appended (always the last argument) which is an ooRexx Directory object, containing an entry with an index value of "USERDATA" which refers to the slot argument. Therefore the Rexx programmer is able to use the slot argument to maintain whatever values that should be made available to the invoked Rexx method(s).

Making the Rexx condition object available as a RexxProxy

If a Rexx condition occurs, it becomes possible to wrap the Rexx condition object[3] as a RexxProxy thereby making it available for Java programmers to interrogate all details of a Rexx condition. In order to make this RexxProxy object available in a Java systematic way, a Java exception of type org.rexxla.bsf.engines.rexx.RexxException is defined with a method getRexxConditionObject() that returns the Rexx condition object as a RexxProxy object.

[3] If Rexx raises a condition, the built-in function (BIF) named condition can be used to retrieve a condition object, which is a Rexx Directory object containing indexes that refer to all aspects of the raised condition.

5.1.2 Creating a RexxProxy for Java Interfaces

The Java class `java.lang.reflect.Proxy` can be used to create Java objects that can be used as arguments for those Java methods that expect an object implementing one or more Java interface(s). Any invocation of an interface method will then be forwarded to the Java object that has to be supplied to the `java.lang.reflect.Proxy` constructor and which must implement the `java.lang.reflect.InvocationHandler` interface. If the `RexxProxy` class implementation would also implement the `InvocationHandler` interface, then `RexxProxy` objects could be employed as invocation handlers as well. This would result in forwarding the invocation of Java interface methods to the denoted `RexxProxy` object which forwards the invocation to its `rexxObject` allowing in effect the implementation of Java methods in Rexx!

The following syntax allows to create a `RexxProxy` object from Rexx that can be used with one or more Java interface types:

```
BsfCreateRexxProxy(rexxObject,[slot], interface1 [,interface2]...)
```

The Java interface types can be indicated by either using their respective class objects or their fully qualified names.

The resulting `RexxProxy` object can then be used as an argument to any Java method expecting one of the listed Java interface types. Any invocation of an interface method and any supplied arguments will be forwarded to the `rexxObject`. There will be an additional argument appended to the Rexx message which is a directory containing the case-preserved name of the Java method name (index `"METHODNAME"`), the Java method object (index `"METHODOBJECT"`), and an optional entry (index `"USERDATA"`) if the `slot` argument was given when creating the `RexxProxy` object .

This mechanism will cause the Rexx methods implemented in the `rexxObject` to be invoked, whenever on the Java side an interface method gets invoked.[4]

Handling Java events synchronously

`RexxProxy` objects can be used for event object listeners, and the invocation of the listener methods will occur synchronously ("real time invocation") allowing to return values that may control the execution path to be taken from then on (e.g. cancelling event handling and the like).

Figure 1 gives an example of how easyly one can put this infrastructure to work: after the Java GUI has been set up the Rexx program gets intentionally blocked with the statement `rexxObject~waitForExit` until either the frame window is closed or the button is pressed. In the meantime Java is able to create and dispatch Java events on a separate Java thread, which will be the thread used for the callback into the

[4] Of course one may take advantage of the ooRexx `UNKNOWN` mechanism.

```
  /* create a RexxProxy to serve two Java interfaces                    */
rexxObject=.eventHandler~new   /* create an event handler object         */
proxy=BsfCreateRexxProxy(rexxObject, , "java.awt.event.ActionListener", -
                                       "java.awt.event.WindowListener")

  /* create a Java awt window and a Java awt button                     */
tmpWin = .bsf~new('java.awt.Frame', 'Hello World!')  /* create a frame   */
tmpWin~addWindowListener(proxy) /* add our event handler to button       */

tmpBut = .bsf~new("java.awt.Button", "Press me!")   /* create a button    */
tmpBut~addActionListener(proxy) /* add our event handler to button       */

  /* prepare window and show it, using cascading messages               */
tmpWin ~~add(tmpBut) ~~pack ~~show ~~toFront

say "waiting for exit..."
rexxObject~waitForExit   /* waits until control attribute has been reset  */

/* ------------------------------------------------------------------- */
::REQUIRES BSF.CLS       /* requires the BSF-class                      */
/* ------------------------------------------------------------------- */
::CLASS "eventHandler"
::method init            /* constructor: set bExitProgram attribute value */
 expose bExitProgram           /* get direct access to control attribute */
 bExitProgram=.false           /* set control attribute value            */

::method unknown         /* intercept any other event: do nothing       */

::method windowClosing   /* an event from java.awt.events.WindowListener */
 expose bExitProgram           /* get direct access to control attribute */
 bExitProgram=.true            /* clear bExitProgram attribute           */

::method actionPerformed /* sole event from java.awt.events.ActionListener */
 expose bExitProgram           /* get direct access to control attribute */
 bExitProgram=.true            /* clear bExitProgram attribute           */

::method waitForExit     /* wait on bExitProgram control attribute      */
 expose bExitProgram             /* get direct access to control attribute */
 guard on when bExitProgram=.true /* block until bExitProgram turns .true */
 say "waitForExit, lock released, returning ..."
```

Figure 1: Using a RexxProxy for Implementing Java Interface Methods with ooRexx.

Rexx program. Whenever Java creates a `windowClosing` or `actionPerformed` event, the respective Rexx method will get invoked changing the value of the control attribute `bExitProgram` to `.true` and thereby unblocking the call to `waitForExit` which causes the main Rexx program to continue to run, closing down the program gracefully.

5.1.3 Implementing Abstract Methods in Rexx

In Java there are abstract Java classes that are expected to be extended by concrete Java classes implementing the abstract methods. If the external Rexx function `BsfCreateRexxProxy` would allow the on-the-fly creation of extended Java classes that forward invocations of (on-the-fly implemented) abstract methods to the supplied

```
fileChooser=.bsf~new("javax.swing.JFileChooser")   /* create a JFileChooser   */

   /* create an instance of the RexxFileFilter                                */
description="*.rex;*.rxj;*.rxo;*.jrexx;*.java"     /* info for user feedback  */
filter=".rex .rxj .rxo .jrexx .java"   /* blank delimited list of extensions  */
rexxObject=.RexxFileFilter~new(description, filter)/* create the filter       */

   /* create a RexxProxy, serving as the implementation for the abstract class*/
proxy=BsfCreateRexxProxy(rexxObject, , "javax.swing.filechooser.FileFilter")

fileChooser~setFileFilter(proxy)             /* set the filter               */

if fileChooser~showOpenDialog(.nil)=0 then   /* if accept button was pressed  */
   say "selected file:" pp(fileChooser~getSelectedFile~getCanonicalPath )
else
   say "dialog was cancelled!"

/* ------------------------------------------------------------------------- */
::REQUIRES BSF.CLS      /* get the ooRexx Java support                       */
/* ------------------------------------------------------------------------- */
/* Rexx class that implements the abstract Java methods "getDescription" and
   "accept" of the abstract Java class "javax.swing.filechooser.FileFilter".
*/
::class RexxFileFilter
::method init              /* constructor: set bExitProgram attribute value   */
  expose description filter   /* attributes to hold description and filter    */
  use strict arg description, filter /* assign arguments to attributes        */

::attribute description    /* description for the filter instance             */

::attribute filter         /* filter                                          */

   /* implementations of the abstract Java methods                            */
::method getDescription    /* implementation of "getDescription"              */
  expose description            /* access attribute directly                  */
  return description            /* return value attribute refers to           */

::method  accept           /* implementation of "accept"                      */
  expose  filter               /* access attribute directly                   */
  use arg fileObject           /* get java.io.File object to work with        */

  if fileObject~isDirectory then /* a directory in hand?                       */
     return .true              /* always include a directory                  */

  fName=fileObject~getName     /* get the filename                            */
  pos=fName~lastpos('.')       /* get last dot in name                        */
  if pos=0 then                /* no extension?                               */
     return .false             /* do not accept file                          */

  fExt =fname~substr(pos)      /* extract file extension without dot          */
  return filter~caselessWordPos(fExt) > 0 /* return .true, if found, 0 else   */
```

Figure 2: Using a `RexxProxy` for Implementing Java Abstract Methods with ooRexx.

`rexxObject`, one could in fact realize abstract methods in Rexx! The following syntax allows to create and instantiate an on-the-fly Java class that forwards abstract method invocations to the supplied `rexxObject`:

```
BsfCreateRexxProxy(rexxObject,[slot], javaAbstractClass [,argument1]...)
```

In the case that there are constructors expecting arguments, one is able to supply them, following the `javaAbstractClass` argument, using commas as delimiters. The returned Java object can then be used as an argument to any Java method expecting an instance of an extension of the given `javaAbstractClass`. Any invocation of an abstract method and any supplied arguments will be forwarded to the `rexxObject`.

There will be an additional argument appended to the Rexx message which is a directory containing the case-sensitive name of the Java method name (index `"METHODNAME"`), the Java method descriptor in Java documentation and ("internal") type signature form (index `"METHODDESCRIPTOR"`), the Java object that forwarded the message (index `"JAVAOBJECT"`) and an optional entry (index `"USERDATA"`), if the `slot` argument was given when creating the `RexxProxy` object.

This mechanism will cause the Rexx methods available in the `rexxObject` to be invoked, whenever on the Java side an abstract method gets invoked.[5]

Figure 2 gives an example of how easyly one can put this infrastructure to work: whenever the Java side invokes either the method `accept` or `getDescription`, the respective Rexx methods will get invoked (one determining whether to accept the Java supplied file object for the file chooser, the other one returning the string to be shown to the user).

5.2 Feature: RexxException

Throwing Java exceptions or errors from Rexx is probably only meaningful, if a callback from Java has been executed and the Rexx program is about to return (to Java) and wishes to signal some extraordinal condition with the means of Java by throwing a particular Java exception.

In order to throw any `java.lang.Throwable` (an exception, an error) on the Java side a new external Rexx function can be devised as follows:

```
BsfJavaException("throw", javaThrowableObject)
BsfJavaException("check"|"clear"|"occurred")
```

The first argument to this external Rexx function can be:

- `"throw"`: throws the `javaThrowableObject` supplied as the second argument, needs to be invoked as the last call before returning to the Java side (no exception/error free interaction with Java would be possible from this moment on),
- `"check"`: returns `.true`, if a Java exception is pending, `.false` else,

[5] Of course one may take advantage of the ooRexx `UNKNOWN` mechanism.

```
   /* get the OpenOffice.org/StarOffice desktop and its XDesktop interface */
xDesktop=uno.createDesktop()~XDesktop

   /* create a proxy that can be used as an OOo-XTerminateListener         */
rexxObject=.TerminateListener~new   /* create terminateListener object     */
proxy=BsfCreateRexxProxy(rexxObject, ,"com.sun.star.frame.XTerminateListener")

xDesktop~addTerminateListener(proxy)   /* add the TerminateListener object */

/*
   ... do something ...
*/

   /* stop vetoing terminating OOo: remove TerminateListener              */
xDesktop~removeTerminateListener(proxy)

/* ------------------------------------------------------------------------ */
::requires UNO.CLS      /* get the OpenOffice.org support for ooRexx       */
/* ------------------------------------------------------------------------ */
/* This listener will throw a "com.sun.star.frame.TerminationVetoException"
   to force OOo to remain active.                                           */
::class TerminateListener

::method unknown         /* intercept any other message: do nothing        */

::method queryTermination  /* may be invoked by xDesktop on an own thread  */
   /* create a com.sun.star.frame.TerminationVetoException object          */
  t=.BSF~new("com.sun.star.frame.TerminationVetoException")
  call BsfJavaException "throw", t  /* throws the TerminationVetoException */
```

Figure 3: Raising a Java Exception to Veto the Termination of OpenOffice.org/StarOffice.

- `"clear"`: clears a pending Java exception,
- `"occurred"`: returns the Java Throwable object, if any, ohterwise `.nil`.

Figure 3 depicts an excerpt from a Rexx program that uses OpenOffice.org (OOo) and wants to make sure that while it is running at least one instance of OOo remains active (the user could inadvertently close all open OOo documents otherwise). If one registers a termination listener with the OpenOffice.org/StarOffice desktop object, then this listener will be invoked by the desktop object when it is about to close the last instance of an OpenOffice.org/StarOffice document and therefore is about to terminate the application. If the invoked listener method `queryTermination` throws a `com.sun.star.frame.TerminationVetoException` then the desktop object will not close the last document, thereby not terminating the application.

6 Roundup and Outlook

The implementation of the Vienna version of BSF4Rexx has not been perfect, as the Rexx APIs would not allow concurrent interaction with running Rexx programs, nor sending Rexx objects concurrent messages.

In order to remove the shortcomings from BSF4Rexx, they needed to be identified first. Next it was necessary to realize the new kernel features and the new APIs that come along with the new version 4.0 of Open Object Rexx (ooRexx), introducing the concurrent APIs briefly.

Devising new APIs for BSF4Rexx, which still remain simple to understand and to use for Rexx programmers, following the "human centricness" of the Rexx language as postulated and mandated by the author of Rexx, Mike F. Cowlishaw, has been a big and ongoing challenge. The core the challenge has been about bridging a strictly typed, compiled language (Java), with a dynamically typed, interpreted language (Rexx), such that Rexx programmers are alleviated of the need to know any of the involved complexities that BSF4Rexx must address.

This article introduced two external Rexx functions, `BsfCreateRexxProxy()` and `BsfJavaException()` which allow to alleviate the identified shortcomings in a manner, that adheres closely to the Rexx language philosophy, yet making a very important area of Java usable for Rexx programmers, that in the past was simply not possible.

With the new ooRexx 4.0 APIs it becomes feasible to create additional, Java-related infrastructures that would e.g. allow to create OpenOffice.org components that are fully implemented in Rexx.

If the ooRexx language gains APIs for debugging and profiling Rexx programs, then BSF4Rexx could be adapted to allow their instrumentation from Java, making it possible to use e.g. the Eclipse infrastructure and its wealth of plugins to devise an integrated, fully portable ooRexx development environment, that ooRexx has been seriously lacking.

7 References

[Cow90] Cowlishaw, M.F.: "The REXX Language", Prentice-Hall (Second edition), 1990.

[Flat01] Flatscher R.G.: "Java Bean Scripting with Rexx", in: Proceedings of the „12th International Rexx Symposium“, Raleigh, North Carolina, USA, April 30th - May 2nd, 2001. WWW (as of 2009-10-31): `http://wi.wu.ac.at/rgf/rexx/orx12/JavaBeanScriptingWithRexx_orx12.pdf`

[Flat03] Flatscher R.G.: "The Augsburg Version of BSF4Rexx", in: Proceedings of the „The 2003 International Rexx Symposium“, Raleigh, North Carolina, USA, May 4th - May 7th, 2003. WWW (as of 2009-10-31): `http://wi.wu.ac.at/rgf/rexx/orx14/orx14_bsf4rexx-av.pdf`

[Flat05a] Flatscher R.G.: "Automating OpenOffice.org with ooRexx: Architecture, Gluing to Rexx Using BSF4Rexx", in: Proceedings of the „The 2005 International Rexx Symposium“, Los Angeles, California, USA, April 17th – April 21st, 2005. WWW (as of 2009-10-31): http://wi.wu.ac.at/rgf/rexx/orx16/2005_orx16_Gluing2ooRexx_OOo.pdf

[Flat05b] Flatscher R.G.: "Automating OpenOffice.org with ooRexx: ooRexx Nutshell Examples for Write and Calc", in: Proceedings of the „The 2005 International Rexx Symposium“, Los Angeles, California, USA, April 17th – April 21st, 2005. WWW (as of 2009-10-31): http://wi.wu.ac.at/rgf/rexx/orx16/2005_orx16_NutShell_OOo.pdf

[Fos05] Fosdick H.: "Rexx Programmer’s Reference", John Wiley & Sons, ISBN: 0-7645-7996-7, URL (as of 2009-10-31): http://www.wrox.com/WileyCDA/WroxTitle/productCd-0764579967.html

[Kal01] Kalender P.: “A Concept for and an Implementation of the Bean Scripting Framework for Rexx”, Seminar paper, University of Essen, MIS and Software Engineering Department, February 2001.

[VeTrUr] Veneskey G.L., Trosky W., Urbaniak J.J.: "Object Rexx by Example", Aviar. URL (as of 2007-10-31): http://www.oops-web.com/orxbyex/

[W3BSF] Homepage of the Apache Software Foundation (ASF) project Bean Scripting Framework(BSF), (as of 2009-10-31): http://jakarta.apache.org/bsf

[W3BSF4Rexx] Homepage of the BSF4Rexx project (as of 2009-10-31): http://wi.wu-http://wi.wu.ac.at/rgf/rexx/bsf4rexx/current/

[W3OORC] Code repository for ooRexx (as of 2009-10-31): http://oorexx.cvs.sourceforge.net/oorexx/

[W3ooRexx] URL (as of 2009-10-31): http://www.ooRexx.org

[W3Rexx] URL (as of 2009-10-31): http://www.Rexx.org

[W3RexxInfo] URL (as of 2009-10-31): http://www.RexxInfo.org

[W3RexxLA] URL (as of 2009-10-31): http://www.RexxLA.org

Date of Article: 2009-10-31

Published in: Proceedings of the „The 2009 International Rexx Symposium“, Chilworth Manor, Winchester, UK, May 18th - May 21st, 2009. The Rexx Language Association, Raleigh N.C. 2009.

Presented at: „The 2009 International Rexx Symposium“, Chilworth Manor, Winchester, United Kingdom, May 18th, 2009.

The new (2009) version of BSF4Rexx is now called "BSF4ooRexx" to denote that it can only be used with the new version of ooRexx 4.0 and higher. Consequently, a new download location has been created for the "BSF4ooRexx" package, which implements the functionality as set forth in this article:

http://wi.wu.ac.at/rgf/rexx/bsf4oorexx/current/

Rony G. Flatscher, Vienna, Austria. 2009-10-31