"Automating OpenOffice with ooRexx: Architecture, Gluing to Rexx Using BSF4Rexx"

2005 International Rexx Symposium
Los Angeles, California, U.S.A. (April 2005)

Rony G. Flatscher (Rony.Flatscher@wu-wien.ac.at)
Wirtschaftsuniversität Wien, Austria (http://www.wu-wien.ac.at)
Agenda

• "OpenOffice.org" ("OOo")
  – Overview Architecture
  – "UNO", "urp"
• BSF, BSF4Rexx
  – Architecture
• Making ends meet
  – Gluing of OOo with ooRexx
• Roundup and Outlook
Sources of figures, examples and hints


• Mr. Augustin's paper "Erweiterung der Skriptfähigkeit von OpenOffice.org durch BSF und JSR-223" at the "WU Wien", cf. http://www.matt.at/oo_examples

OpenOffice.org
Brief History, 1

- StarOffice
  - Originates in Germany
  - Portable C++ class library ("Star")
    - Allow creation of a portable integrated office suite
    - Goal: compatibility to MS Office
  - 90'ies
    - OS/2
    - Windows
    - Explored Macintosh, Unix
OpenOffice.org
Brief History, 2

• StarOffice, continued
  – Bought by Sun
    • Development transferred to the U.S.A.
  – Solaris
    • Allowed MS Office compatible office suite
  – Opensource
    • In parallel to commercial version "StarOffice"
    • "OpenOffice.org" (OOo)
      – Linux, Macintosh, OS/2, Solaris, Windows, …
OpenOffice.org
Developer's Bird Eye's View, 1

• Set of services to create and maintain documents

• All common functionality of all types of documents is extracted and organized as a set of interfaces
  – E.g. Loading, saving, printing documents

• For each type of document the specific functionality is extracted and organized as a specialized set of interfaces
  • E.g. TextCursors ("write"), Cell-Manipulation ("calc"), etc.
Client/Server Architecture

- Employing distributable components ("UNO")
  - Server can run on any computer in the world!
  - Operating system of server as well as that of the client is irrelevant!

- Communication
  - TCP/IP sockets
  - Named pipes, if available

- Client can run on the same machine as the server
OpenOffice.org

Building Blocks, 1

- "UNO"
  - Universal Network Objects
  - Distributable, interconnectible infrastructure
  - All functionality is organized in the form of classes
    - "UNO classes"
- "urp"
  - "UNO remote protocol"
    - CORBA-like protocol
OpenOffice.org

Building Blocks, 3

• "Service Managers"
  – Supplied by servers
  – Can be used to request services from the server
  – Returned service allows access to a part of the "office" functionality, E.g.
    • com.sun.star.frame.Desktop
    • com.sun.star.configuration.ConfigurationProvider
    • com.sun.star.sdb.DatabaseContext
Illustration 2.1: Service manager
• "Services"
  – Can be comprehensive
  – Are organized in partitions named
    • "Interfaces" (group of functions/methods) and
    • " structs" (group of related properties only)
  – Depending on the desired task you need to request the appropriate interface, e.g.
    • com.sun.star.view.XPrintable
    • com.sun.star.frame.XStorable
    • com.sun.star.text.XTextDocument
• An example
  – Two services with seven interfaces exposed
    • There are more available
  – "OfficeDocument"
    • Four interfaces
  – "TextDocument"
    • Three interfaces
• Client needs to get in touch with the server
  – URL-style connection string
  – Server creates an object to interact with and returns a handle for it to the client
OpenOffice.org
Programming languages

• OOo version 1.1
  - C++
  - StarBasic
    - Scripting language
    - **Java**
  - Python

• Upcoming OOo version 2 in addition
  - BeanShell (interpretable Java)
  - JavaScript
OpenOffice.org
Java, 1

- Full implementation for UNO
  - "Java UNO"
- Every UNO component/class can be directly used by Java
- UNO components can also be developed in Java
- C++ UNO and Java UNO are fully interoperable!
XComponentContext xLocalContext =
com.sun.star.comp.helper.Bootstrap.createInitialComponentContext(null);
// initial serviceManager
XMultiComponentFactory xLocalServiceManager = xLocalContext.getServiceManager();
// create a URL resolver
Object urlResolver = xLocalServiceManager.createComponentInstanceWithContext(
"com.sun.star.bridge.UnoUrlResolver", xLocalContext);
// query for the XUnoUrlResolver interface
XUnoUrlResolver xUrlResolver =
(XUnoUrlResolver) UnoRuntime.queryInterface(XUnoUrlResolver.class, urlResolver);
// Import the object
Object rInitialObject = xUrlResolver.resolve(
"uno:socket,host=localhost,port=2002;urp;StarOffice.ServiceManager");
// XComponentContext
if (null != rInitialObject) {
    System.out.println("initial object successfully retrieved");
} else {
    System.out.println("given initial-object name unknown at server side");
}
• No direct support for ooRexx in OOo
• No external Rexx functions available for OOo
• BUT
  – **If** there was a way to bridge ooRexx with Java and then use Java to bridge to UNO, **then** it would be **possible** to team OOo with ooRexx!
  – … and there **is** a means available for that:

  **BSF4Rexx**!
• Bean Scripting Framework
  – A Java framework, making it easy for Java to invoke scripts in non-Java scripting languages
    • E.g. JavaScript, NetRexx
  – Originally developed by IBM as open source
    • Part of IBM's WebSphere to allow scripts to be deployed within Java Server Pages (JSP)
  – Fall 2003 handed over to jakarta.apache.org
    • Used e.g. in ant, xerces
BSF4Rexx

• BSF with a Rexx engine
  – Allows the usage of Rexx from BSF
    • Any Java program can invoke Rexx
    • Rexx scripts are able to communicate with Java objects, if made available by the Java program
  – Allows Java to be used as a huge Rexx function library
    • The public methods and public fields of every Java object and Java class object can be used by Rexx
    • If necessary, Java can be started up by Rexx
/* classic Rexx version, querying the installed Java version */

/* load the BSF4Rexx functions and start a JVM, if necessary */
if rxFuncQuery("BSF") = 1 then /* BSF() support not loaded yet ? */
do
  call rxFuncAdd "BsfLoadFuncs", "BSF4Rexx", "BsfLoadFuncs"
  call BsfLoadFuncs /* registers all remaining BSF functions */
  call BsfLoadJava /* loads Java */
end

say "java.version:" bsf('invoke', 'System.class', 'getProperty', 'java.version')

Yields, e.g.:

java.version: 1.4.2
BSF4Rexx Architecture

http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/

BSF()
BsfDropFuncs()
BsfInvokedBy()
BsfLoadFuncs()
BsfLoadJava()
BsfQueryAllFunctions()
BsfQueryRegisteredFunctions()
BsfUnloadJava()
BsfVersion()
BSF4Rexx, Typing Issue, 1

"Strict"

• A newer version than the "Augsburg" version of BSF4Rexx
  – Will be named "Vienna" or "WU", still under development
  – Beta version can be downloaded from http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/
  – Allows to omit type information usually needed for Java
    • Java is a strongly typed programming language, Rexx is not!
  – "strict" allows to supply explicit type information
    • Needed under rare circumstances where Java methods of the same name and same number of arguments exist, but differ in the type of their arguments only
"Strict"

- "Type indicators" preceed the argument in BSF()- subfunctions containing the word "Strict"
- "Type indicators" are one of the following strings
  - **BOolean**, **BYte**, **Char**, **Double**, **Float**, **Int**, **Long**, **Object**, **SHort**, **String**
  - Only bold and uppercase letters need to be given
  - Java type information is given in the HTML documentation
  - "BOolean", "Byte", "Char", "Double", "Float", "Int", "Long", "SHort", "String" are the Java "primitive" data types
  - "Object" is *any* Java object
Camouflaging Java, 1

BSF.cls

- "BSF.cls"
  - An ooRexx package
  - Defines routines, classes and methods which hide the procedural interface from ooRexx programs
  - Wraps all BSF()-subfunctions into ooRexx Methods
  - Allows to import Java classes explicitly into ooRexx in the form of ooRexx proxy classes
  - Allows to create ooRexx proxy objects which interact with the appropriate Java objects
Camouflaging Java, 2

**BSF.cls**

- "BSF.cls"
  - Supports Java array objects as ooRexx array proxies
    - Allows using Java array objects as if they were ooRexx array objects
    - Hence indexing of proxy arrays starts with 1 (and not 0)!
/* ooRexx version */

say "java.version:" .bsf4rexx~system.class ~getProperty('java.version')

::requires "BSF.cls" -- loads the ooRexx (camouflaging) support

Yields, e.g.:

java.version: 1.4.2
Camouflaging Java, 3
Architecture

Java Program

BSF (Java)

BSF Registry

JNI

BSF4Rexx
(C++)

ooRexx environment
(e.g. ".bsf4rexx")

BSF.cls

ooRexx scripts

RexxAndJava
Pre-register Fundamental Java Class Objects

- ooRexx directory ".BSF4Rexx"

1) .bsf4rexx~Class.class
2) .bsf4rexx~Object.class
3) .bsf4rexx~Method.class
4) .bsf4rexx~Array.class
5) .bsf4rexx~String.class
6) .bsf4rexx~System.class
7) .bsf4rexx~Boolean.class
8) .bsf4rexx~boolean
9) .bsf4rexx~Byte.class
10) .bsf4rexx~byte
11) .bsf4rexx~Character.class
12) .bsf4rexx~char

1) .bsf4rexx~Double.class
2) .bsf4rexx~double
3) .bsf4rexx~Integer.class
4) .bsf4rexx~int
5) .bsf4rexx~Long.class
6) .bsf4rexx~long
7) .bsf4rexx~Float.class
8) .bsf4rexx~float
9) .bsf4rexx~Short.class
10) .bsf4rexx~short
11) .bsf4rexx~Void.class
12) .bsf4rexx~void
Proxy Class BSF

- Execute "BSF.cls" either with **call** or **::requires**
  ```
  call "BSF.cls"
  ::requires "BSF.cls"
  ```

- Allows to import Java classes and interact with them as if they were ooRexx classes
  ```
  .bsf~import(rexxName, javaName)
  .bsf~import("javaFrame", "java.awt.Frame")
  f=.javaFrame~new("hi!")~~show~~toFront~~setSize(200,100)
  ```

- Allows to create Java objects
  ```
  .bsf~import("javaFrame", "java.awt.Frame")
  f1=.javaFrame~new("hi!") -- using an imported Java class
  -- or directly via class .BSF
  f2=.BSF~new("java.awt.Frame", "hi!") - using .BSF directly
  ```
Proxy Class **BSF**

- Procedural BSF()-subfunctions available as (mangled) instance methods:
  1. `bsf.addEventListener`
  2. `bsf.exit`
  3. `bsf.invoke`
  4. `bsf.invokeStrict`
  5. `bsf.getFieldValue`
  6. `bsf.setFieldValue`
  7. `bsf.setFieldValueStrict`
  8. `bsf.getPropertyValue`
  9. `bsf.setPropertyValue`
  10. `bsf.setPropertyValueStrict`

- Procedural BSF()-subfunctions available as class methods:
  1. `exit`
  2. `sleep`
  3. `lookupBean`
  4. `pollEventText`
  5. `getStaticValue`
  6. `postEventText`
  7. `wrapArray`
  8. `createArray`
  9. `wrapEnumeration`
  10. `setRexxNullString`
-- create a two-dimensional (5x10) Java Array of type String
arr=.bsf~createArray(.bsf4rexx~string.class, 5, 10)

arr[1,1]="First Element in Java array."  -- place an element
arr~put("Last Element in Java array.", 5, 10)  -- place another one

do i over arr  -- loop over elements in array, ooRexx style!
say i
end

Yields:
First Element in Java array.
Last Element in Java array.
Making Ends Meet
Setting Up, 1

• Install BSF4Rexx
  – Follow the instructions coming with BSF4Rexx
  – Run the supplied test/nutshell programs

• Configure the OOo Java archives
  – Make sure OOo is enabled for Java
    • Check "Tools → Options… → Security → OpenOffice.org → Java → Enable"
  – Add the following OOo "jar"-files (in ...\program\classes) to the environment variable "CLASSPATH"
    • jurt.jar, jut.jar, javaunohelper.jar, ridl.jar, classes.jar, sandbox.jar
    • juh.jar, unoil.jar
Making Ends Meet
Setting Up, 2

• Either
  – Start OOo ("soffice.exe") with the following command line
    ```
    soffice -accept=socket,host=localhost,port=8100;urp;
    ```

• Or
  – Configure OOo to always listen on the given socket and communicating with 'urp' as explained in the OOo Developers Guide, p. 31ff
  – Start one instance of OOo
    • Possible to start an explicit server instance of OOo!
Making Ends Meet
Get the Ball Rolling, 1

• Get in contact with the server and request access to OOo using Java UNO
  – Create a local (client-side) OOo context and get its ServiceManager from it
    • Get a URLResolver service from the local ServiceManager
    • Use the URLResolver service to establish a connection to the server returning the RemoteContext
    • Request the remote ServiceManager from the received RemoteContext
Making Ends Meet
Get the Ball Rolling, 2

– With the help of the remote ServiceManager request the "Desktop" service on the server
  • Of all of the interfaces defined for the "Desktop" service, request the interface "XComponentLoader"
    allowing the loading (creation) of components (documents)
  • Use the functionality of the XComponentLoader to load (create) an empty text document
/* initialize connection to server, get its Desktop-service and XComponentLoader interface */
CALL "BSF.cls"

xComponentContext = .bsf~new("com.sun.star.comp.helper.Bootstrap") –
    ~createInitialComponentContext(.nil)

xUrlResolver = xComponentContext~getServiceManager() –
    ~createInstanceWithContext("com.sun.star.bridge.UnoUrlResolver", xComponentContext)

unoResolverName = .bsf4rexx~Class.class~forName("com.sun.star.bridge.XUnoUrlResolver")
unoRuntime = .bsf~new("com.sun.star.uno.UnoRuntime")
urlResolver = unoRuntime~queryInterface(unoResolverName, xUrlResolver)

unoUrl = "uno:socket,host=localhost,port=8100;urp;StarOffice.NamingService"
rInitialObject = urlResolver~resolve(unoUrl)
namingServiceName = .bsf4rexx~Class.class~forName("com.sun.star.uno.XNamingService")
rName = unoRuntime~queryInterface(namingServiceName, rInitialObject)

rXsmgr = rName~getRegisteredObject("StarOffice.ServiceManager")
msfName = .bsf4rexx~Class.class~forName("com.sun.star.lang.XMultiServiceFactory")
xMsf = unoRuntime~queryInterface(msfName, rXsmgr)

-- Retrieve the Desktop object, we need its XComponentLoader interface
-- to load a new document
aDesktop = xMsf~createInstance("com.sun.star.frame.Desktop")
xDesktop = .bsf4rexx~Class.class~forName("com.sun.star.frame.XDesktop")
oDesktop = unoRuntime~queryInterface(xDesktop, aDesktop)
xComponentLoaderName = .bsf4rexx~Class.class~forName("com.sun.star.frame.XComponentLoader")
xComponentLoader = unoRuntime~queryInterface(xComponentLoaderName, oDesktop)
-- ... continued ...

/* Open a blank text document */

/* No properties needed */
propertyValueName = .bsf4rexx~Class.class~forName("com.sun.star.beans.PropertyValue")
loadProps = .bsf~createArray(propertyValueName, 0) /* 0=no elements, i.e. empty Java array */

/* load an empty text document */
xWriterComponent = xComponentLoader~loadComponentFromURL("file:///c:/docs/aFile.sxw", "_blank", 0, loadProps)

file:///c:/docs/aFile.sxw
http://www.RexxLA.org/aFile.sxw
ftp://www.RexxLA.org/aFile.sxw

scalc
swriter
simpress
sdraw
• OOo
  – Opensource, openplatform
  – UNO, urp
    • C++, Java
  – Client/server architecture
• ooRexxx
  – BSF4Rexx as bridge
• Full openplatform control by ooRexxx
  – Not restricted to C++, Java, StarBasic or Python!
Roundup and Outlook, 2

- Creating an ooRexx package
  - Simplifying recurring tasks, like establishing a connection with a server
  - Simplifying access to components, e.g. making it easier to manipulate cells of the spreadsheet

- With the advent of OOo 2.0
  - Devise a plug-in for BSF4Rexx, allowing ooRexx to be dispatched from within OOo
  - Will make it possible to use ooRexx wherever StarBasic is used!