“AUTOMATING OPENOFFICE.ORG WITH OOREXX: ARCHITECTURE, GLUING TO REXX USING BSF4REXX”

Rony G. Flatscher

Wirtschaftsuniversität Wien (WU), Austria

„The 2005 International Rexx Symposium“, Los Angeles, California, U.S.A.,

April 17th - April 21st, 2005.

ABSTRACT

The opensource Microsoft Office clone "OpenOffice" is available on multiple platforms, from Windows, over Linux to OS/2. It can read/write Microsoft office file-formats, such as Word, Excel or PowerPoint. Its scripting architecture is radically different from what Microsoft has come up with and appears to be more systematic, although there is a rather steep learning curve to it.

The architecture of OpenOffice is exposed via the UNO (Uniform Network Objects) interface, which allows C, C++ and Python programs to exploit OpenOffice. On the Windows platform there is an ActiveX/OLE-interface supplied, such that ActiveX-script languages like VBS, JS, as well as ooRexx can be used for scripting purposes, but such programs will lock-in the company into the Windows operating system.

For the programming language Java, OpenOffice supplies a Java interface to UNO, which can also be exploited in rather innovative ways, e.g. using BSF4Rexx (Bean Scripting Framework for Rexx) to bridge between OpenOffice and ooRexx. Such a solution would allow for the driving/scripting of OpenOffice in a platform independent way, allowing customers to eventually break out of possibly undesired lock-ins (e.g. Windows operating system and/or ActiveX/OLE-technology).

This article gives a conceptual overview of OpenOffice UNO and explains in detail how UNO can get instantiated and interfaced with by ooRexx.

Keywords: Object Rexx, ooRexx, Automation, Scripting, OpenOffice, OOo.

1 INTRODUCTION

In the 90ies a German company named "Star Division" developed a portable C++ class library that it named "Star". It became available for MacOS, Unix, OS/2 and Windows. With this infrastructure the company started to develop an integrated office suite which should be portable to all platforms that "Star" was available. It was named "StarOffice".

Due to the success of the Microsoft office (MSO) suite, it was very important for Star Division to create their office suite in a way, which allowed co-operating with MSO users as seemlessly as possible. For that reason powerful import and export filters for MSO have been of paramount importance and in effect allows it today, to import and export most MSO documents into/from StarOffice on any platform Star Office is available.

In addition the StarOffice user interface resembles the MSO user interface, such that it becomes rather feasible for companies to migrate from MSO to StarOffice should they wish to escape a possibly undersired lock-in situation, or should they strategically plan to not be locked-in anymore. Among other benefits such a migration also opens up the possibility to switch operating and hardware system platforms as long as StarOffice is available on the new target.

Finally, the built-in scripting language ("Star Basic") is a look-alike language to Microsoft's Visual Basic, sharing most of the language constructs and properties, thereby making it relatively easy for MSO (end-user) programmers to become productive in StarOffice.

The company Sun, looking for a MSO compatible, powerful office suite for its Unix-based operating system Solaris bought the German company and made it available as a commercial product on Windows as well. In addition, two important developments have taken place since then:

- "StarOffice" has become fully programmable from Java, which is because of Sun's interest to leverage its programming language Java as much as possible,
- Sun released a version of "StarOffice" into the opensource and made it freely available under the name "OpenOffice". The WWW homepage is located at

"OpenOffice.org" and it has therefore become common to abbreviate it as "OOo".

Having an opensource version of StarOffice freely available allows companies and organizations to escape potentially undesired lock-ins by changing to the open-source codebase, either immediately or whenever such a company decides this to be a move in its own interest. In the case of switching from StarOffice to OpenOffice (or the other way round) incurs no switching costs whatsoever. If switching from MSO to StarOffice/OOo switching costs (installation, re-training users) need to be taken into account, but it becomes possible to save by cancelling costful upgrade plans in the long run, and possibly remarkable money by being able to switch away from the Windows operating system altogether. Such switching to opensource software that is free, seems to be quite attractive for public administrations in Europe (e.g. German cities, or the Austrian capital Vienna).

Possibly more important, free software allows schools, Universities, third world countries, private people to employ a powerful, integrated office package for their needs, without cutting into their (most of the time extremely tight) budgets. Having the software not only free, but also in opensource means that maintenance of such software is even possible, if the original writers are not available anymore. Especially in the scholar environment opensource allows analysis, improvements and extensions to such software, because of such a white box approach.

2 ARCHITECTURE OF OPENOFFICE.ORG

Interestingly, OpenOffice.org (OOo) has been designed as a genuine client-/server-application system, which preferably communicates via TCP/IP sockets. A typical installation of OOo installs the client and the server components on the same computer and the client would use the server component on the same system. For this reason it becomes principally possible that an OOo client uses an OOo server which resides on another computer, possibly even running a different operating system installed possibly on a different hardware architecture!

The component model of OOo is called "Universal Network Objects" (UNO), where each component is described in an interface description language (IDL) module.[1] OpenOffice.org is therefore built from a set of UNO components, i.e. the wordprocessor application in effect is nothing else but a set of appropriately combined UNO components, the spreadsheet application another set of appropriate UNO components. Figure 1 depicts the architecture and indicates that the OOo applications are created from UNO components, which may be even shared.

In the case that some UNO components may be usefully employed by different OOo applications, then they are merely re-used. This way, if one learned about automating a particular UNO component in the context of the wordprocessor, that very same knowledge can be directly applied to the spreadsheet application or presentation application and the like. With other words: the UNO components can be freely combined and their functionality can be easily shared among totally different applications!

2.1 UNO Interface Description Language Modules

IDL modules may contain nested IDL modules, where the structure represents a hierarchy having a root module. Identifying a type in this hierarchy of modules is therefore easy, one starts out at the root module and names all nested modules one needs to traverse, leading in and separating the names with double colons (::, c-style) or separating them with a dot only (Java style). Hence the type named "XPrintable" has the fully qualified name "::com::sun::star::view::XPrintable" (C++) or "com.sun.star.view.XPrintable" (Java).

[1] The UNO IDL allows the defining of types (classes, components), structures ("struct") consisting of fields only, exceptions, constants, and enumerations.

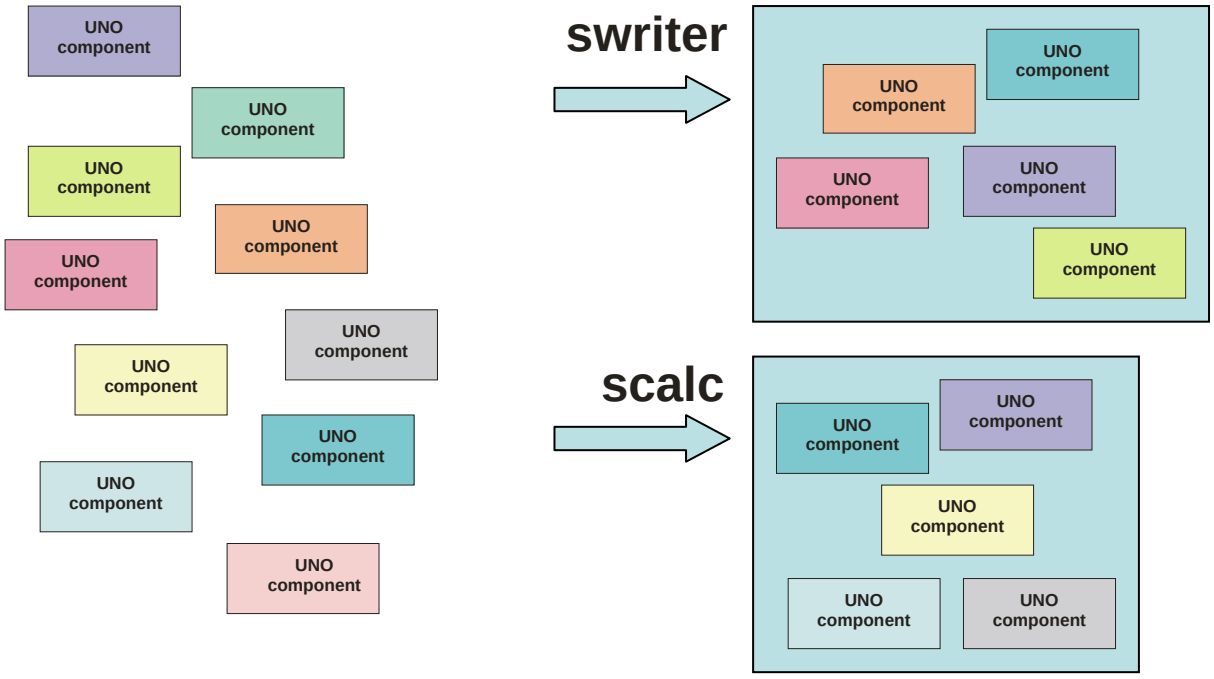

Figure 1: Configuring OpenOffice.org Applications from UNO components.

2.2 UNO Service Components and their Interfaces

Each UNO component (defined as an IDL type in a module) usually represents a specific "service", consisting of additional services, and possessing properties and interfaces (usual offering methods and properties to a specific aspect of the UNO component) to it. Properties allow the storing of information appertunant to services.

To create instances of service components one employs a "Service Manager", using either its method "createInstance()" or "createInstanceWithArguments()", supplying the fully qualified name of the UNO component.[2] The returned object is called a "service object". Because service managers are used to create service objects, they are also called "factories".

In the context of OOo the term "interface" is very important and describes usually a set of functions/methods belonging semantically together, e.g. the interface

[2] Due to the evolution of UNO the concept of a "context" got introduced which is intended to maintain important runtime information for UNO applications, like prefabricated, fundamentally important service objects etc. As it may become important to create instances of service components with a given context, a service manager may in addition provide the methods "createInstanceWithContext()" and "createInstanceWithArgumentsAndContext()", which are defined in the "com.sun.star.uno.XMultiComponentFactory" interface.

"com.sun.star.view.XPrintable" defines the methods getPrinter(), setPrinter() and print(). If one received a UNO service object, that can be printed, then such a service object will contain the interface "com.sun.star.view.XPrintable". To really print that object, it is mandatory to firstly query the "XPrintable" interface from the service object, which will return an object that in addition to the original one possesses the methods of the requested interface, which then can be invoked.

Although at first it seems odd that one needs to explicitly request interface objects in order to invoke the desired methods in service objects, it has a few uses, e.g.:

- in a client-/server-application it may make sense to have locally a minimal interface object driving a remote service object,
- having functionality organized in interfaces allows for grouping methods which belong semantically and functionally together, makes it rather easy to gain an overview of the composition of service components (starting out with their instances, the "service objects"),
- once a programmer gets used to the most important interfaces, he or she then will know over time, how functionality is organized in OOo; this way it becomes possible to quickly research UNO components and locate desired interfaces.

Programming UNO components therefore is many times focused on querying the needed interfaces, using the appropriate method ("qeryInterface()") of the interface named "com.sun.star.uno.XInterface", which all of the interface components implement.

2.2.1 Communicating With UNO

As UNO employs a client-/server-model the communication is carried out in a remote fashion, employing a protocol ("urp": UNO remote protocol) that is comparable to OMG's (Object Management Group) CORBA[3] (Common Object Request Broker Architecture), and which some claim is faster in aspects than CORBA. urp uses by default TCP/IP sockets for communication.

[3] The OOo developer's guide [W3OOoDev] describes the commonalities and differences.

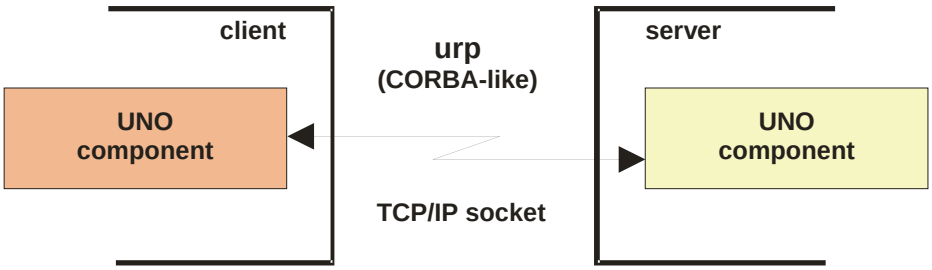

Figure 2: Client-/Server-Communications with UNO Components Using TCP/IP Sockets and the CORBA-like Protocol “urp” (UNO Remote Protocol).

It is interesting to note that whenever there is a need to specify an address of one sort, that UNO expects them to be formatted as URLs (Uniform Resource Locators). This way it becomes a rather simple task for OOo clients to denote the computer they wish to use as the OOo server via UNO.[4]

2.3 Starting Up ("Bootstrapping") the UNO Environment

In order to automate OpenOffice.org (and by the very same token StarOffice) it is necessary to start up and initialize the UNO runtime environment, which then is used to get a service manager from the server.

The service manager is then used to create ("instantiate") all the needed and desired service objects, from which one needs to request specific interfaces in order to arrive at the methods (functions) they supply.

These are the minimal programming steps that are necessary to be carried out, if one wishes to load OpenOffice.org and access the functionality of the different office applications (e.g. “swrite”=wordprocessor, “scalc”=spreadsheet):

(1) The type (class) "com.sun.star.comp.helper.Bootstrap" supplies the static method "bootstrap()" which we can use to create a default UNO runtime environment.
The returned object is of type "com.sun.star.uno.XComponentContext", having all standard UNO services registered and supplying two methods, "getValueByName()" and "getServiceManager()".

(2) Using the method "getServiceManager()" of the context object one receives an object, which represents the local service manager, with a type of

[4] The URL concept is carried forward to filenames as well, which need to be formatted as URLs.

"com.sun.star.lang.XMultiComponentFactory". This is the most important object for interacting with and taking advantage of UNO components. It has three methods: “getAvailableServiceNames()”, “createInstanceWithContext()” and “createInstanceWithArgumentsAndContext()”.

(3) The local service manager object is used to create the OpenOffice.org desktop service object, which is needed to control and drive the OOo applications. From the documentation and the UNOIDL it follows that it has the type (class) name: "com.sun.star.frame.Desktop".

(4) According to the UNO programming model one needs to request the specific desktop interface from the desktop service object, which itself possesses an interface named “com.sun.star.frame.XComponentLoader”.

(5) With the XComponentLoader interface the method “loadComponentFromURL()” becomes available, which is used to create new office documents or load existing[5] ones.

In this scenario a locally installed OpenOffice.org application is addressed.

But as already mentioned, OOo is implemented as a true client-/server-application, communicating via TCP/IP sockets. This makes it possible that an OOo-client uses an OOo-server which resides on a different computer. In order to exploit this particular feature of OOo, one needs to configure the OOo-server to allow for accepting clients at specific ports, either generally via the OOo “share” configuration or specifically for a certain user via the OOo “user” configuration.[6] The OOo-client merely uses the “com.sun.star.bridge.XUnoUrlResolver” interface and supplies an appropriately formatted URL.[7]

[5] As already mentioned, the name of documents must be given in the platform independent URL notation.

[6] It would be also possible, to temporarily start OOo via the commandline and tell it to accept OOo-client connections at a specific port, e.g. (cf. [OOo03], looking for the heading „Starting OpenOffice.org in Listening mode“):

```
soffice -accept=socket,host=localhost,port=8100;urp;
```

[7] Cf. [OOo03], looking for the heading „Importing a UNO Object”.

3 AUTOMATING (SCRIPTING) OF OPENOFFICE.ORG

The UNO components are implemented mostly by C++ programs that adhere to the type descriptions defined in the UNO IDL modules. After Sun bought StarOffice the UNO component technology was made available to the Java language by creating an appropriate proxy/adapter infrastructure into the runtime environment. The Java support for UNO allows the addressing of UNO components such as if they were native Java components. In addition the Java UNO infrastructure allows for creating and implementing UNO components fully in Java as well, leveraging Java knowhow via the UNO infrastructure to C++ and other programming languages that wish to interface and exploit UNO.

Of course, neither C++ nor Java are programming languages that are appropriate for end-user programming. Therefore, OOo comes with a specific programming language that is meant for automating (scripting) OOo components: OOo Basic. This language is modeled after Microsoft's Visual Basic, which is used as the main scripting language in the Microsoft Office suite.

Scripts allow for automating recurrent tasks, which many times would otherwise have to be carried out manually. In addition it becomes possible to combine end-user applications like wordprocessors with spreadsheets in ways that may not have been anticipated by the original creators of such components. Mainly, because of the huge success of the scripting technology in the Windows world, scripting languages have become popular in other domains as well.

Whereas the Windows world adheres to Visual Basic, the Unix based world concentrates on scripting languages with a Unix shell background that can be seen in some concepts incorporated into scripting languages like Perl, TcL, PHP or Python. The scripting language Python is very interesting in the context of OOo as there has been a very active group of developers who have been attempting to create an interface to Python on par with the native OOo Basic support.

Starting with OOo version 2.0, expected sometimes in 2005, a Java based scripting framework is put into place, which allows Java based scripting languages like BeanShell ("interpretative Java") and Rhino (a Java implementation of JavaScript) to be plugged into OOo. The scripting framework allows the usage of its scripting languages everywhere where OOo Basic can be used from within OOo.

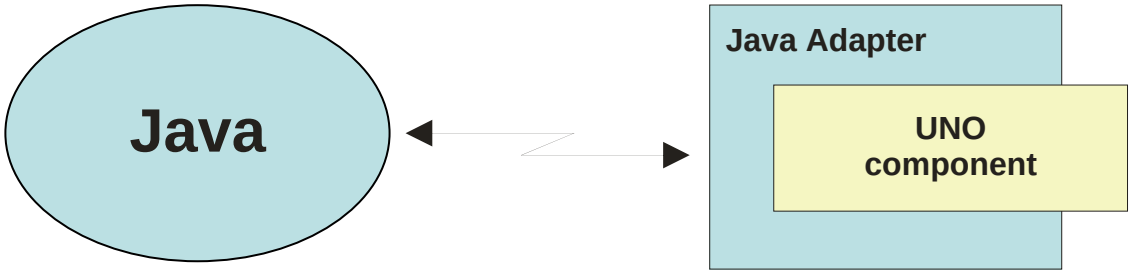

Figure 3: The Java UNO Support.

Because of this it should be possible to employ REXX and Object REXX, if one uses a Java bridge like "BSF4Rexx" (cf. [Flat01], [Flat03], [Flat04]).

3.1 UNO Programming with Java

Sun has made the UNO architecture available to Java programmers by supplying helper and adapter classes. Because of this, UNO can be fully exploited by Java and therefore OOo as well. Figure 3 depicts the interfacing of Java programs with UNO via a Java adapter.

It must be noted, however, that it may be necessary to activate the Java support via "Tools -> Options -> Security -> OOo -> Java -> Enable" (OOo 1.1.4). Also, one needs to set up the Java CLASSPATH according to the Developer Guide in order to use the Java classes that allow the interfacing with UNO.[8]

Studying and experimenting with the Java support for UNO, it becomes clear that one can really take advantage of all of UNO via Java. Figure 4[9] shows a Java program that starts up the UNO environment following the layout given in section 2.3 above, creates an OOo desktop service and uses that to create an empty word processor document (“swriter” document).

Studying the Java program the following steps need to be sequentially taken in order to create an OOo word processor document:[10]

(1) have OOo create a “context” object by bootstrapping the UNO runtime environment,

[8] Cf. [OOo03], looking for the heading „Use Java UNO class files” documents the location and the names of the Java archive files you need to have access to.

[9] Excerpt from the Java example “HardFormatting.java” coming with the OOo development kit.

[10] The paragraph numbers in the following list relate to the numbers in parenthesis that are inserted into the program’s code in figure 4 in form of line comments.

```
class Test {
   public static void main (String args[]) {
         // excerpted from "HardFormatting.java" from the OOo development package
     com.sun.star.frame.XDesktop xDesktop = null;
     com.sun.star.lang.XMultiComponentFactory xMCF = null;
     try {
       com.sun.star.uno.XComponentContext xContext = null;

       // (1) bootstrap the UNO runtime environment
       xContext = com.sun.star.comp.helper.Bootstrap.bootstrap();

       // (2) get the service manager
       xMCF = xContext.getServiceManager();
       if( xMCF != null ) {
         System.out.println("Connected to a running office ...");
         // (3) start up an instance of office
         Object oDesktop = xMCF.createInstanceWithContext(
             "com.sun.star.frame.Desktop", xContext);
         // (4a) get the XDesktop interface object
         xDesktop = (com.sun.star.frame.XDesktop)
             com.sun.star.uno.UnoRuntime.queryInterface(
                 com.sun.star.frame.XDesktop.class, oDesktop);

         // (4b) get the desktop's component loader interface object
         com.sun.star.frame.XComponentLoader xComponentLoader =
             (com.sun.star.frame.XComponentLoader)
                 com.sun.star.uno.UnoRuntime.queryInterface(
                     com.sun.star.frame.XComponentLoader.class, xDesktop);

             // create an empty text ("swriter") document
         com.sun.star.beans.PropertyValue xEmptyArgs[] = // empty property array
             new com.sun.star.beans.PropertyValue[0];
         // (5) create an empty word processor ("swriter") component (document)
         com.sun.star.lang.XComponent xComponent =       // text document
                xComponentLoader.loadComponentFromURL( "private:factory/swriter",
                    "_blank", 0, xEmptyArgs);
       }
       else
           System.out.println("Not able to create desktop object.");
     }
     catch( Exception e) {
         e.printStackTrace(System.err);
         System.exit(1);
     }
     System.err.println("Successful run.");
     System.exit(0);
   }
}
```

Figure 4: Java Program which Bootstraps the UNO Runtime Environment and Loads the OpenOffice.org Desktop Service to Load a New, Empty Word Processor Document.

(2) request a “service manager” object from the context object,

(3) use the “service manager” object to create the OOo “desktop” service object,

(4) (a) use the “desktop” service object to query (and receive) the “desktop interface” object, and

(b) use the “desktop interface” object to query (and receive) the interface object

representing the “component loader”,

(5) use the “component loader” object to load an empty instance of the word processor component.

3.2 BSF4Rexx: Bridging ooRexx with UNO via Java

In the past years an external Rexx function package has been created that allows interfacing Rexx with Java and Java with Rexx ("BSF4Rexx", cf. REXX (cf. [Flat01], [Flat03], [Flat04]). Because of this enabling technology it should be possible to drive UNO - and as a result OOo - via Rexx as well, using Java as the bridge.

Employing the object-oriented version of Rexx, ooRexx, and its specific support coming with "BSF4Rexx")[11], the code to drive UNO and OOo can be mapped directly from the Java program above (cf. figure 4), mapping the statements almost in a 1:1 fashion from the Java program. Looking at the resulting code (cf. figure 5) and comparing it with the original Java program it becomes clear that the ooRexx version is quite smaller and much more ledgible than the Java code. The main reason for this is the typelessness of Rexx compared to Java, as there is no need in ooRexx to cast to a specific type.

Figure 5 depicts an ooRexx program taking advantage of the BSF4Rexx ooRexx support (module “BSF.CLS”) which carries out exactly the same functions as the Java program in figure 4. This program has been tested with OOo 1.1.4 and runs as successfully as the above Java program, thereby serving as a proof of concept, that it is principally possible to interface ooRexx with OOo (UNO).

Due to BSF4Rexx it is possible to create ooRexx programs that can drive OpenOffice.org via the Java UNO interfaces in a platform independent manner. ooRexx scripts therefore should be able to drive OOo unchanged on different operating systems like Windows and Linux. ooRexx programmers who wish to develop OOo

[11] For IBM's Object REXX and its opensource and free sibling "ooRexx" there is an ooRexx module named "BSF.CLS" supplied with BSF4Rexx, which camouflages Java as Object REXX (cf. [Flat04]). This support makes it possible to import Java classes into ooRexx and interact with them as if they were ooRexx classes. Java objects can be treated as if they were ooRexx objects, e.g. by sending ooRexx messages to them using the ooRexx message operator (the tilde, ~, in ooRexx also called "twiddle"). In essence this support allows to use all of Java as if it was a huge ooRexx class library, that is already ported to all platforms ooRexx runs on.

```
-- (1) bootstrap the UNO runtime environment
xContext   = .bsf~bsf.import("com.sun.star.comp.helper.Bootstrap")~bootstrap
unoRuntime = .bsf~bsf.import("com.sun.star.uno.UnoRuntime")

-- (2) get the service manager
xMCF = xContext~getServiceManager();
if xMCF<>.nil then
do
   say "Connecting to office ..."
      -- (3) start up an instance of office
   oDesktop = xMCF~createInstanceWithContext("com.sun.star.frame.Desktop", xContext)
      -- (4a)get the XDesktop interface object
   xDesktop = unoRuntime~queryInterface( -
      .bsf~bsf.import("com.sun.star.frame.XDesktop"), oDesktop)

      -- (4b) get the desktop's component loader interface object
   xComponentLoader = unoRuntime~queryInterface( -
            .bsf~bsf.import("com.sun.star.frame.XComponentLoader"), xDesktop)

      -- create an empty Java array of type "com.sun.star.beans.PropertyValue"
    PropertyValueClass=.bsf~bsf.import("com.sun.star.beans.PropertyValue")
    xEmptyArgs=bsf.createArray( PropertyValueClass, 0 )  -- empty property array

      -- (5) create an empty word processor ("swriter") component (document)
    xComponent = xComponentLoader~loadComponentFromURL( "private:factory/swriter", -
               "_blank", 0, xEmptyArgs);
end
else
    say "Not able to create desktop object."

::requires BSF.CLS   -- get Java support (using "BSF": Bean Scripting Framework)
```

Figure 5: ooRexx Program Mapped from the Java Program in Figure 4.

scripts in ooRexx can draw from the many OOo code examples written in Java, in OOo Basic and in Python.

4 SUMMARY AND OUTLOOK

This article introduced the OpenOffice.org (OOo) client/server architecture, which is based on UNO (Universal Network Object) components, that communicate via a CORBA like protocol, urp (UNO remote protocol), with each other. There exists a comprehensive support for Java, which enables Java programs to fully interact with UNO components and also makes it possible to implement UNO components in Java that can be controlled with other programming languages, namely C++, which is the original implementation language for OOo.

Matching the opensource office suite with the opensource scripting language “Open Object Rexx” (ooRexx) is possible right away, if using BSF4Rexx as a bridge for ooRexx to Java. This support enables ooRexx to interact with UNO components by addressing the Java interfaces for UNO.

This article demonstrated a proof of concept that this bridging is indeed possible (in a straight-forward manner!) adding with ooRexx a new programming (scripting) language to drive OOo besides C++, Java, the builtin OOo Basic and Python.

Analyzing the C++ and Java programs from the OOo development kit there are quite a few recurring tasks to be carried out, like querying interfaces to service objects at all times, which could be simplified, if an appropriate support with the means of ooRexx is created.

With the advent of OOo 2.0 a new Java-based scripting framework will be introduced for which a plug-in for BSF4Rexx could be devised, allowing ooRexx to be used from within OOo for scripting purposes. In effect, this would allow ooRexx to be used in contexts, where so far only OOo Basic has been available!

5 REFERENCES

[Aug05] Augustin A.: “Examples for Open Office Automation with Scripting Languages”, Bachelor course paper, Wirtschaftsuniversität Wien, January 2005.

[Ende97] Ender T.: "Object-Oriented Programming with REXX", John Wiley & Sons, New York et.al. 1997.

[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.

[Flat03] Flatscher R.G.: “The Augsburg Version of BSF4Rexx”, in: Proceedings of the „14th International Rexx Symposium“, Raleigh, North Carolina, USA, May 5th - May 7th, 2003.

[Flat04] Flatscher R.G.: “Camouflaging Java as Object REXX”, in: Proceedings of the „2004 International Rexx Symposium“, Böblingen, Germany, May 3rd - May 6th, 2004.

[Flat05] Flatscher R.G.: “Automating OpenOffice with ooRexx: ooRexx Nutshell Examples for Write and Calc”, in: Proceedings of the „The 2005 International Rexx Symposium“, Los Angeles, California, U.S.A., April 17th - April 21st, 2005.

[OOo03] N.N.: “OpenOffice.org 1.1 - Developer’s Guide”, PDF download from [W3OOo], June 2003.

[VeTrUr96] Veneskey G., Trosky W., Urbaniak J.: "Object Rexx by Example", Aviar, Pittsburgh 1996.

[W3BSF] Homepage of Apaches “Bean Scripting Framework” (BSF), URL (2005-05-17): `http://jakarta.apache.org/bsf`

[W3BSF4REXX] Beta test and release candidate site of the “BSF4Rexx” package, URL(2004-05-01): `http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/`

[W3ObjRexx] Open Object Rexx homepage, URL (2005-05-17): `http://www.ooRexx.org`

[W3OOo] OpenOffice.org homepage, URL (2005-05-17): `http://www.OpenOffice.org/`

[W3OOoAPI] OpenOffice.org API homepage, URL (2005-05-17): http://api.OpenOffice.org/

[W3OOoUDK] OpenOffice.org UDK (UNO development kit) homepage, URL (2005-05-17): http://udk.OpenOffice.org/

[W3RexxLA] Rexx homepage of the “Rexx Language Association”, URL (2005-05-17): http://www.RexxLA.org

Date (latest version) of Article: 2005-05-17.

Published in: Proceedings of the „2005 International Rexx Symposium“, Los Angeles, California, U.S.A., April 18th - April 21st, 2005, The Rexx Language Association, Raleigh N.C., 2005.

Presented at: „2005 International Rexx Symposium“, Los Angeles, California, U.S.A., April 20th, 2005.