EXTENDING THE WORKPLACE SHELL WITH OBJECT REXX

Rony G. Flatscher
Department of Management Information Systems and Software Engineering at the University of Essen (Germany)

(On leave from: Department of Management Information Systems at the Vienna University of Economics and Business Administration (Austria))
„11th International Rexx Symposium“, Phoenix, Arizona, USA, May 24-26, 2000.

ABSTRACT

IBM’s “Workplace Shell” (WPS) is an object-oriented (OO) user interface (UI) framework developed for and deployed in its PC operating system OS/2. The infrastructure used for building the Workplace Shell is IBM’s “System Object Model” (SOM), which allows for interfacing programs and functions of different programming languages like C, C++, COBOL, PL/I, Smalltalk and Object Rexx using a runtime component. The architecture and features of SOM were introduced into the shaping of the ”Object Management Group” (OMG) standard “Common Object Request Broker Architecture” (CORBA).

This paper introduces the concepts of these technologies and discusses the abilities introduced to a scripting language like Object Rexx by it. Short examples should demonstrate the appliccability of these technologies and may serve as demonstrations what scripting languages are capable of achieving, if a well layed out object-oriented architecture is available. The same principles and application possibilities are available to many scripting languages being employed for driving SOM, DSOM (“distributed SOM”) and CORBA applications in general. With the advent of wrapping up the Windows user interface with object-oriented interfaces via

Microsoft's D/COM and "Windows Scripting Host" (WSH), the same features should become available for scripting languages under the Windows UI in the future.

1 INTRODUCTION

IBM's "Workplace Shell" (WPS) was introduced for the first time with its operating system "operating system/2" (OS/2) for personal computers (PC) in 1992. It has been devised and implemented with IBM's "System Object Model" (SOM) technology, which itself was part of OS/2 2.0 in 1992. IBM's SOM served as a testbed for implementing industrial strength object-oriented infrastructure technology which was also used as IBM's input in helping shape the "Object Management Group" (OMG) "common object request broker architecture" (CORBA). As a matter of fact SOM 2.0 introduced CORBA 1.1 compliance and the last incarnation of IBM's SOM (version 3.0) complied to the CORBA 2.0 specifications and was made available for AIX, Windows and OS/2.

In 1996, meeting demands from IBM's largest international user group, SHARE, IBM had developed and introduced an object-oriented (OO) version of IBM's stratetic "system application architecture" (SAA) procedures language ("scripting language") REXX. This backwardly compatible language is called "Object Rexx" and has been made available to OS/2, Windows 95, Windows 98 and Windows NT, as well as for IBM's AIX and the open source operating system Linux on the Intel platforms.

The object-oriented version of the Rexx interpreter included direct support for SOM and "distributed SOM" (DSOM). Therefore it became possible to interact with this scripting language with all programs employing SOM, effectively allowing Object Rexx to be used for scripting *any* SOM respectively DSOM class! One of the premiere OO frameworks available in IBM operating systems was OS/2's Workplace-Shell, so the Object Rexx developers supplied small scripts which demonstrated how to interact with it via this new OO scripting language.

2 SYSTEM OBJECT MODEL (SOM)

At the end of the 80ies IBM invested heavily in developing state of the art technology to be used in their products. One very important set of technologies was the "System

Object Model" (SOM) with the intention to ultimately allowing for coupling programs written in different computer programming languages with each other. The basic concept was to create a set of interface specifications for programming languages and a runtime environment serving as the mediator between them. Without a mediator, programs written in any of two programming languages need to establish bilateraly interfacing rules in order to be able to invoke functions on either side and to be able to marshall and unmarshall the values of arguments depending on the conventions used by the other language. Figure 1 depicts this classic problem which yields the necessity to create *n*(n-1)* interface and communication infrastructures.

One way to ease this problem is to define a set of neutral (ie. Language independent) interface rules and invocation conventions which need to be defined and implemented for the appropriate programming languages. Now, if any two programs written in different programming languages need to invoke functions at the other side, they follow the same invocation rules (ie. communication protocol) and use the same conventions for encoding ("marhsalling") and decoding ("unmarshalling") of arguments and return values. In the 80'ies a well known set of rules was established, known as "remote procedure call" (RPC) [OrHaEd96]. With the advent of broader acceptance of object oriented technology the need arose for creating a set of invocation rules which followed the object oriented paradigm. The

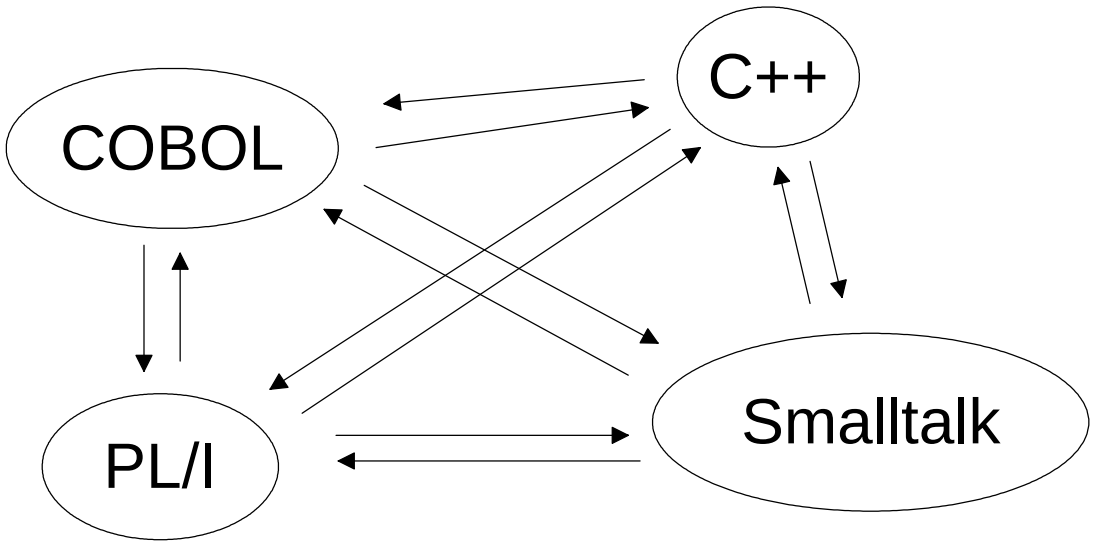

Figure1: Total of n(n-1) interfaces.*

“Object Management Group” (OMG) was established for creating object oriented standards and one of its main purposes has been to define the “Common Object Request Broker Architecture” (CORBA), which targeted exactly this problem of enabling interaction of programs (even written in different programming languages) using an object oriented model. Right from the beginning, one of the big contributors to the definition of the CORBA standard was IBM which created a technology called “System Object Model” (SOM).

SOM defined a set of object oriented invocation rules and - what had been unique by then - a runtime system being available at all times to offer relay services among all programs communicating with each other via SOM. The problem of enabling the communication between programs written in different programming languages was effectively reduced to creating the interfacing and communication support from one programming language to SOM only. Every invocation of any routine in some other program looked like it was implemented in the SOM runtime, hence it became unnecessary to know anything else but the SOM interface. Figure 2 depicts this situation and it becomes clear that the problem was reduced to create just *n* interface and communication infrastructures for *n* programming languages.

As SOM follows the object oriented paradigm, it defines a class hierarchy and employs metaclasses for runtime management of metadata. In order for the runtime to know about those classes, methods and their signatures, developers need to

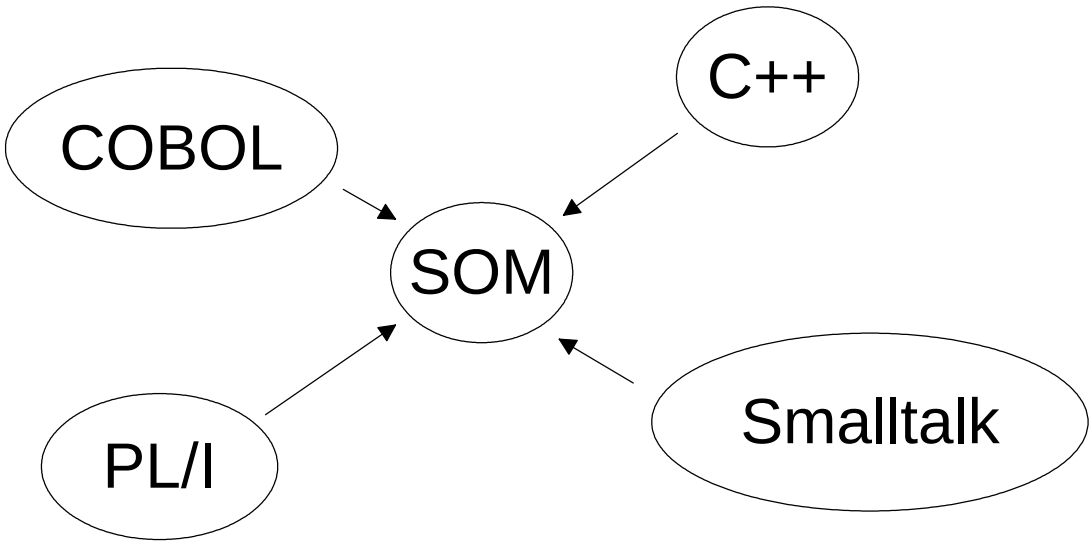

Figure2: Total of n interfaces.

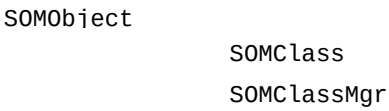

Figure3: SOM class hierarchy.

declare them, usually in the form of an “Interface Defintion Language” (IDL) which then gets used to build the “Interface Repository” (IR) which the SOM runtime can the SOM class hierarchy in which the root is named *“SOMObject”* and the metaclass of SOM is called *“SOMClass”*, a subclass of *“SOMObject”*. In addition three object oriented frameworks were defined for the SOM runtime system: “Interface Repository”, “Metaclass Management” and “Event Management”, which programmers could take advantage of too.

1992 SOM 1.0 was introduced into the market via IBM’s personal computer (PC) operating system OS/2, version 2.0, which was the first incarnation of the 32-Bit version of that operating system.[1)] The initial implementation took advantage of the thread concept available in OS/2 by having the SOM runtime and all SOM classes run within the same process space, dispatching the SOM objects on their own threads. Although this helped with performance problems, when run on slow hardware, the execution of SOM objects was not insulated from the others, thereby opening the risk, that an ill-behaved implementation would hinder or lock-up other SOM objects or the SOM runtime itself. The “workplace shell” (WPS) introduced with the same version of OS/2 was built using the SOM technology.

In 1994 version 2.0 of SOM was introduced with OS/2 3.0 which allowed for executing SOM objects in their own process space, separated from the SOM runtime itself. Because of this new ability, it also became possible to interact with SOM objects in process spaces on physically different machines, even run under different operating systems. IBM produced a separate product called “distributed SOM” (DSOM) to allow for this particular feature. SOM and DSOM were made available to AIX, OS/2 and Windows, and were compliant with OMB CORBA 1.1. It is interesting to note, that the workplace shell which was built on SOM 1.0 could be - according to IBM presentations at that time - switched to use 2.0 without big

1) The portable 32-Bit version of OS/2 to be created by Microsoft was later on renamed to “Windows NT”, after IBM and Microsoft departed from each other.

problems, as most of the interface definitions and communication rules remained backwardly compatible.

Finally, at the end of 1996 IBM released SOM 3.0, which has made SOM fully OMG CORBA 2.0 compliant, allowing SOM to interact with “Object Request Brokers” (ORB) from other vendors. This version has been made available by IBM via Internet to users of AIX, OS/2 and Windows, but concludes IBM development efforts in this technology, although it still has been maintained for its customers up to date.

3 WORKPLACE SHELL (WPS)

With OS/2 2.0 a totally new graphical user interface (GUI) was created by IBM; called “Workplace Shell” (WPS). This user interface was fully created with the object oriented paradigm, defining its own class hierarchy. The WPS is built using SOM and therefore every WPS class is in effect a SOM class. This means that any programming language capable of utilizing SOM, e.g. by sending SOM messages to SOM classes or SOM objects, is able to directly use all of the WPS classes in the same manner.

This new GUI introduced the object oriented paradigm to the user which in the development stage led to an updated set of IBM’s “Common User Access” (CUA), introducing the object oriented notion into the GUI by 1992.

Figure 4 depicts the WPS class hierarchy, having the class *“WPObject”* as its root. It is very interesting to note how the class hierarchy classifies the objects interesting for a GUI:

1) Class *“WPFileSystem”*: objects of this class and its subclasses are reflected in the file system, i.e. their realizations are represented as physical individual files in the file system.

2) Class *“WPAbstract”*: objects of this class and its subclasses are *not* reflected in the file system, i.e. they are *not* represented as files on their own in the file system. Instead, instance data is stored in special system files, called *“INI”*-files.

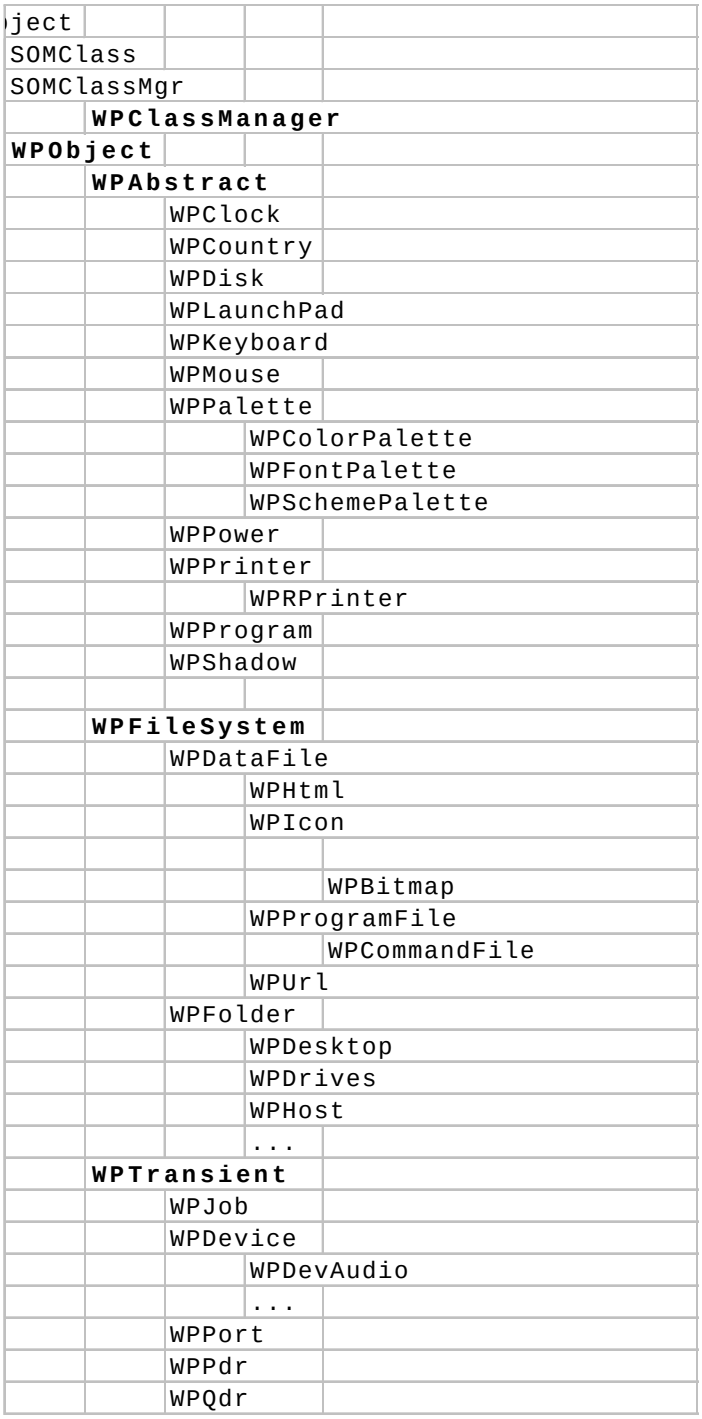

Figure 4: The WPS Class Hierarchy (fragment).

3) Class *"WPTransient"*: objects of this class and its subclasses are *not* reflected in the file system. They exist only while the workplace shell is running and are lost without a trace, after the WPS is shut down.

Studying the WPS class hierarchy allows one to fully understand the individual classes and the behaviour they expose. If it was possible to subclass WPS classes,

then one could in effect influence/change the pre-defined behaviour by maximum re-usage of what has been devised and implemented by the creators. In addition this maximum re-usage of (tested) code which the OO paradigm allows for guarantees from there on, that future enhancements and bug-fixes in the pre-fabricated classes will get re-used right away by the subclasses!

4 OBJECT REXX (ORX)

The programming language Rexx has been created for easying batch programming in IBM's mainframe world and was made available as a product on 20th March 1979 by IBM. Later on it became IBM's"System Application Architecture" (SAA) procedural language and was ported therefore to all IBM operating systems. Outside of IBM the language attracted quite a few open source and commercial developers who produced Rexx interpreters [W3RexxLA]. The language became an ANSI standard ("X3.274") in 1996 [W3Rexx] and serves as a reference for the actual discussions in creating an international standard in decimal arithmetics for Java and the Web [W3DecAri].

The development of Object Rexx began in the beginning of the 90'ies when important IBM customers via the largest IBM customer association, SHARE, requested an object oriented version of Rexx which should be fully backward compatible. In response to these customer requests IBM developed "Object Rexx" (ORX) which was introduced for the first time in 1996 with the new version of OS/2 4.0 (a.k.a. "Warp"). Object Rexx is an interpreted programming language which in principle follows the object model of Smalltalk [GolRob83], enhanced with the ability of multiple inheritance.[2)] Since then Object Rexx has been made available for AIX, Linux, OS/2, Windows 95, Windows 98 and Windows NT.

4.1 The Built-in Object Rexx SOM Support

The OS/2 implementation of Object Rexx allows direct interaction with SOM and DSOM in a very easy to use manner: D/SOM classes and D/SOM objects look as if they are plain Object Rexx objects. As a matter of fact the entire implementation of

[2)] Unlike Smalltalk there is no default GUI development environment going with it.

```
/* querying the SOM interface repository with Object REXX */
aRepository =.somClassMgrObject~_get_somInterfaceRepository
SAY "repository:" pp(aRepository) "of class:" pp(aRepository~class)
SAY
aContainer = aRepository~contents("InterFaceDef", .true)
SAY "aContainer:" pp(aContainer) "items" pp(aContainer~items)
length = LENGTH(aContainer~items)
i = 0
DO anItem OVER aContainer
    i = i + 1
    SAY RIGHT(i,length) "id:" LEFT(pp(anItem~_get_id),35) "name:" pp(anItem~_get_name)
    anItem~somFree
END
aRepository~somFree
exit 0

::ROUTINE pp
    RETURN "[" || arg( 1 )|| "]"
/* class to get access to SOM */
::CLASS Test PUBLIC EXTERNAL 'SOM SOMObject'
```

Figure 5: Querying the SOM Interface Repository with Object Rexx.

the D/SOM support in Object Rexx makes it transparent for the programmer that he may be addressing D/SOM classes and D/SOM objects!

In the case that some signatures contain INOUT and OUT arguments,[3)] it becomes necessary to supply an interface to the basic standardized (SOM/CORBA) data types.

4.1.1 Interacting with the SOM Runtime System

In order to use Object Rexx directly for interacting with the D/SOM runtime system it is necessary as a pre-requisite, that one learns about D/SOM and its frameworks (and using the gained knowledge to apply it directly to CORBA!). Figure 5 [4)] [5)] depicts an Object Rexx program, which queries the D/SOM interface repository (IR):

3) INOUT and OUT parameters could get changed in their values by the invoked method with datatype representations which are not compatible with Object Rexx' internal handling of it. Therefore it is necessary to use the specialized Object Rexx classes defined in *"\os2\dlfclass.cmd"* in place for the native Object Rexx data types. The supported INOUT and OUT datatypes are: 'Environment', 'Object', 'Number', 'Boolean', 'Char', 'Octet', 'Short', 'UShort', 'Long', 'ULong', 'Float', 'Double' and 'String'. Sequences and arrays are exchanged using the Object Rexx class Array.

4) The Object Rexx message operator is the tilde "**~**", hence identifiers right of a tilde denominate the name of a message, optionally followed by arguments enclosed in paranthesis.

5) Identifiers led in by a point "**.**" are so-called "environment symbols" which the interpreter looks up by

```
repository: [a Repository] of class: [The SOMProxy class]
aContainer: [an Array] items [423]
1 id: [::SOMObject] name: [SOMObject]
2 id: [::Sockets] name: [Sockets]
3 id: [::AnyNetSockets] name: [AnyNetSockets]
4 id: [::Contained] name: [Contained]
5 id: [::AttributeDef] name: [AttributeDef]
6 id: [::BOA] name: [BOA]
7 id: [::SOMEEvent] name: [SOMEEvent]
8 id: [::SOMEClientEvent] name: [SOMEClientEvent]
9 id: [::Context] name: [Context]
10 id: [::ConstantDef] name: [ConstantDef]
11 id: [::Container] name: [Container]
12 id: [::SOMPEncoderDecoderAbstract] name: [SOMPEncoderDecoderAbstract]
13 id: [::SOMPAttrEncoderDecoder] name: [SOMPAttrEncoderDecoder]
... cut ....
122 id: [::TypeDef] name: [TypeDef]
123 id: [::SOMEWorkProcEvent] name: [SOMEWorkProcEvent]
124 id: [::WPObject] name: [WPObject]
125 id: [::M_WPObject] name: [M_WPObject]
126 id: [::WPFileSystem] name: [WPFileSystem]
127 id: [::M_WPFileSystem] name: [M_WPFileSystem]
128 id: [::WPFolder] name: [WPFolder]
129 id: [::M_WPFolder] name: [M_WPFolder]
130 id: [::WPDataFile] name: [WPDataFile]
131 id: [::M_WPDataFile] name: [M_WPDataFile]
132 id: [::WPAbstract] name: [WPAbstract]
133 id: [::M_WPAbstract] name: [M_WPAbstract]
... cut ....
423 id: [::M_OverrideFlWorkerEx] name: [M_OverrideFlWorkerEx]
```

Figure 6: Output (fragment) of Object Rexx Program of Figure 5.

- In order for Object Rexx to load the SOM support at least one class, even if it is not used, needs to be defined as an external SOM class, which is depicted in figure 5 at the very bottom at the class directive[6)].
- The program starts by retrieving a handle to the present SOM repository object.
- Sending the SOM repository object the message *contents()* with an argument of *"InterFaceDef"* yields a container of SOM classes available at that particular system at the time the message got sent. The Object Rexx *DO...OVER* loop iterates over the single SOM objects representing SOM classes and issues the given names.

name in the environment directory available to all Object Rexx programs in a process.

6) Directives are led in by two consecutive colons ("**::**") and are carried out by the interpreter *before* the first line of that program gets executed. Hence, directive definitions are always available at the start of the program.

SOMObject	Defines attributes and methods, available to all subclasses
Animal	Defines attributes: *name, sound, genus, species* Defines methods: *talk(), display()*
Dog	Defines attributes: *breed, color* Overrides methods: *_get_genus(), _get_species(), display()*
BigDog	Overrides method: *talk()*

Figure 7: Class hierarchy of the Animal Example.

- Every access which retrieves a SOM object needs to be followed by an explicit invocation of the *somFree()* method in order for the SOM runtime to know that it is safe to reclaim the appropriate resources.

Figure 6 depicts part of the output of the program in figure 5 on a PC running OS/2 version 4.

4.1.2 Interacting with SOM Applications

With Object Rexx there are some tutorial programs enclosed emphasizing different aspects of the programming language. One set of examples deals with the very

```
/* derived from IBM's animal.cmd example */
spot =.dog~new
Say "spot's default name:" spot
say "spot's ClassName: " spot~somGetClassName
say "display"; spot~display
say "now talk, spot:"; spot~talk
say
sadie =.bigDog~new /* Create new Big Dog Object */
sadie~setup('Sadie', 'German Shepard', 'black and tan', 'Steve')
say "sadie's default name:" sadie
say "sadie's ClassName: " sadie~somGetClassName
say "display:"; sadie~display
say "now talk, sadie:"; sadie~talk

/* import some SOM Classes to use from Object Rexx*/
::Class Dog Public EXTERNAL 'SOM Dog'

::Class BigDog Public EXTERNAL 'SOM BigDog'
    ::method setup /* setup object */
        expose owner
        use arg name, breed, color, owner /* Owner assign on use Arg.... */
        self~_set_name(name) /* Set the SOM attribute */
        self~_set_breed(breed)
        self~_set_color(color)
        self~objectName = name /* set up the object's name to be the name as well */

    ::method display /* display attribute values */
        expose owner
        say 'The Big <'self~_get_color'> Dog <'self~_get_name'> is owned by <'owner'>'
```

Figure 8: Object Rexx Programm Using the SOM Classes "Dog" and "BigDog".

```
spot’s default name: a Dog
spot’s ClassName:    Dog
display
The animal named (Genus: Canis, Species: Familiaris) says:
        <Unknown>
It’s breed is and its color is .
now talk, spot:
        <Unknown>
sadie’s default name: Sadie
sadie’s ClassName:    BigDog
display:
The Big <black and tan> Dog <Sadie> is owned by <Steve>
now talk, sadie:
        WOOF WOOF
        WOOF WOOF
        WOOF WOOF
        WOOF WOOF
```

Figure 9: Output of Object Rexx Programm of Figure 8.

ability of Object Rexx to transparently interact with SOM classes written in any supported programming language.

This section introduces the supplied SOM example with an Object Rexx program of the author, which itself got derived from IBM’s sample set.

```
#include "animal.idl"
    interface Dog : Animal
    {
        attribute string breed;
        // The breed of this Dog.
        attribute string color;
        // The color of this Dog.
#ifdef __SOMIDL__
    implementation {
        releaseorder: _get_breed, _set_breed, _get_color, _set_color;
        //# Class Modifiers
        functionprefix = dog_;
        callstyle = oidl;
        dllname = "animals.dll";
        //# Attribute Modifiers
        breed: noset;
        color: noset;
        //# Method Modifiers
        _get_genus: override;
        _get_species: override;
        display: override;
        somInit: override;
        somUninit: override;
        somDumpSelfInt: override;
    };
#endif /* __SOMIDL__ */};
```

Figure 10: SOM IDL File for Class “Dog”, a Subclass of “Animal”.

```
#include "dog.idl"
    interface BigDog : Dog
    {
#ifdef __SOMIDL__
        implementation {
                //# Class Modifiers
                functionprefix = bigdog_;
                callstyle = oidl;
                dllname = "animals.dll";

                //# Method Modifiers
                talk: override;
    };
#endif /* __SOMIDL__ */};
```

Figure 11: The SOM IDL for Class "BigDog", a Subclass of "Dog".

```
/*
* This file was generated by the SOM Compiler and Emitter Framework.
* Generated using:
* SOM Emitter emitctm: 2.7
*/
#define BigDog_Class_Source
#include <bdog.ih>
SOM_Scope void SOMLINK bigdog_talk(BigDog *somSelf)
{
    /* BigDogData *somThis = BigDogGetData(somSelf); */
    string sound;
    BigDogMethodDebug("BigDog","bigdog_talk");

    somPrintf(" WOOF WOOF " );
    somPrintf(" WOOF WOOF " );
    somPrintf(" WOOF WOOF " );
    somPrintf(" WOOF WOOF " );
    if \(sound = Animal__get_sound(somSelf\))
    somPrintf("(and sometimes: %s) " , sound);
}
```

Figure 12: The Implementation of Class "BigDog" in C.

Figure 7 shows a class hierarchy of SOM classes built with some language with SOM support (the concrete examples are built using C). The class *"BigDog"* specializes the class *"Dog"*, which itself got implemented in C. Figure 8 shows the Object Rexx program *"animal.cmd"* which contains a class definition for *"BigDog"* and overrides some of the *"BigDog"* methods from within Object Rexx. In addition it accesses to SOM class *"Dog"* too, which behaviour does not get changed. The output of the Object Rexx program of figure 8 is shown in figure 9.

For the SOM runtime to be able to resolve these interactions it is necessary to get information about the involved parties. In the case of the prefabricated and installed C programs realizing the *Animal*, *Dog* and *BigDog* classes, IDL definitions have to be supplied and compiled for the SOM interface repository. Example IDLs are shown in figures 10 *(Dog)* and 11 (*BigDog*), an implementation in C for *BigDog* is depicted in figure 12, showing that the implmentation merely overrides method *talk()* by issuing big capital letter “WOOF”s. This is the method for instances of the SOM class *BigDog* which gets invoked in return of receiving the message *talk*, as is the case in the Object Rexx program with the instance called *“sadie”*.

There are two more noteworthy points to be made with respect to the the Object Rexx program in figure 8 with its output as shown in figure 9:

- the default implementation of the method *“display”* as used by instances of class *Dog* use *“<unknown>”* for method *talk*,
- the Object Rexx implementation overrides the method *“display”* for instances of class *BigDog* and therefore gives totally different information. In addition it invokes the *talk* method of the *BigDog* class which has an implementation in C, giving capitalized “WOOF”s.

The Object Rexx support for SOM and DSOM allows therefore to easily interact with any D/SOM classes. This way it becomes possible to solve recurrent actions by automating them with Object Rexx, in addition it is possible to use any D/SOM class for scripting purposes using the easy to learn language Object Rexx.[7)]

[7)] In principle, the same functionality can be employed with the Windows version of Object Rexx and IBM’s support for OLE/ActiveX-Automation as made available via [...??? ORX-homepage]. If IBM manages to make Object Rexx available to the “Windows Scripting Host” (WSH) [...???...] comparable functionality to D/COM is made available to Object Rexx under the Windows set of operating systems. In effect, it may then be possible to control/script every aspect of Windows and Windows applications with the easy to learn and easy to use Object Rexx, even allowing interactive interpretations at runtime (e.g. for exploring/debugging the D/COM environment).

4.2 The Built-in Object Rexx WPS Support

As the Workplace Shell is implemented using the SOM technology, it is possible for Object Rexx to address and interact with it using the built in D/SOM support. As the WPS is of such an utmost importance to the OS/2 operating system, a comprehensive support has been implemented in addition, making it easier for Object Rexx programs to interact with the WPS. The specific Object Rexx support consists of:

- a mandatory Object Rexx WPS-SOM DLL which needs to get installed via invoking *"\os2\wpsinst.cmd +"*, and
- optional WPS related definitions put into the Object Rexx environment, by invoking the programs: *"\os2\wpconst.cmd"* and *"\os2\wpsysobj.cmd"*.

In addition *"\os2\wpfind.cmd"* allows to search for files recursively thru all available Workplace Shell folders. Figure 13 depicts the Object Rexx program carrying out the search, demonstrating the usage of the native WPS class methods, which by convention all start with the lowercase letters "wp". This subroutine uses the title of the file object to look for and the starting WPS folder, which together with its WPS subfolders will get analyzed recursively until the file object is found, in which case the WPS object representing it gets returned or finally the value *.nil*, to indicate that the sought for file object was not found.

In order to use the WPS programming support it is necessary to understand the Workplace Shell and to study its architecture, the classes, their behaviour and how they are employed in the GUI. Therefore all examples available with WPS programming are helpful for exploring this architecture.

Due to the technology in place with OS/2, namely SOM, WPS and Object Rexx, it becomes possible to even extend the workplace shell itself with classes written in Object Rexx. In order for such classes to be available to the SOM it is important to load them at the time the WPS starts up initially. It is at this time that the Object Rexx supplied WPS-SOM-DLL gets invoked which in turn looks for an Object Rexx program by the name *"\wpuser.cmd"*, which can be used to load all WPS class extensions created with Object Rexx which then get registered with SOM.

```
Find: procedure /* edited extract from "\os2\wpfind.cmd" */
    use arg title, folder
    say 'Searching' folder~wpQueryTitle 'for "'title'"'
        /* populate the folder with all objects, not just folders. */
    folder~wpPopulate(0, folder~wpQueryTitle, .false)
    if \result then do /* error calling wpPopulate? */
        say 'Populate error for' folder~wpQueryTitle /* yes, report error. */
        return .nil
    end
        /* Get the 1st item in this folder */
    first = folder~wpQueryContent(folder,0 /*QC_FIRST*/)
        /* and the last item in the folder */
    last = folder~wpQueryContent(folder,2 /*QC_LAST*/)
    this = first /* we start with the 1st object */
    do while this \= .nil
        if this~wpQueryTitle = title then return this /* found! return it */
        else
        do
                prev = this /* keep reference to prev object */
                if this = last then /* nope, are we at end of list? */
                        this = .nil /* indicate at end. */
                else /* otherwise, get next item in folder. */
                        this = folder~wpQueryContent(this, 1 /*QC_NEXT*/)
                prev~wpUnLockObject /* unlock previous object. */
        End
    end
    return .nil /* object not found. return nil */
```

Figure 13: Find procedure of "\os2\wpfind.cmd".

The following two sub-chapters explain how one could extend the workplace shell by defining Object Rexx classes which are derived from WPS classes.[8)]

[8)] The "locked workplace shell folder" example used in this article goes back to an example posted in "news:comp.lang.rexx" by the then chief developer, Rick McGuire, and has been adapted a little bit.

4.2.1 Password Protected WPS Folder: Requirements

If one wishes to extend the Workplace Shell with a password protected folder, one needs to first analyze the requirements, study the present implementation of the *WPFolder* class and then think about possibilities on how to devise a new class which would behave accordingly.

These are the requirements, which should get implemented:

- A password protected folder must only be usable if a user was authenticated as being allowed to access its content,
- if using a password scheme it should be possible to change the password at any time,
- all features and behaviour of the present implementation of a folder with the class *WPFolder* should be fully available with the password protected folder.

Analyzing the implementation of the WPS class *WPFolder* it becomes clear, that there is a message sent every time a user wishes to open a folder, either by using the popup menu option or by double-clicking the icon representing the folder object. If one is able to intercept that particular message *wpOpen*, then it would be possible to ask the user at that moment for a password and if it is not correct, merely suppressing the forwarding of that particular message to the superclass *WPFolder*.

```
/* source: Rick McGuire (appr. 1996/1997), adapted: 2000-03-04;
   ---rgf, wuw; (using VREXX.ZIP and changing from WPDLF to DLF-data type classes)
      using VREXX.ZIP, ews from Steve Lamb (IBM)
*/

call RxFuncAdd 'VInit', 'VREXX', 'VINIT'

if Vinit() = "ERROR" then       /* error loading VRexx-functions */
do
   call VExit                   /* clean-up */
   raise syntax 40.1 array ("VREXX.Vinit()")    /* abort program        */
end
.local~lock_icon = STREAM( "REXX.ICO", "C", "QUERY EXISTS")
.environment~WPLockFolder = .WPLockFolder       /* make class available */

::REQUIRES DLFClass             /* needs the support for INOUT/OUT datatypes */
```

Figure 14: Loading the Rexx Function Package “VREXX”.

```
::CLASS WPLockFolder SUBCLASS WPFolder INHERIT VXPWPrompts
 ::METHOD wpclsQueryTitle CLASS
   return 'LockFolder'

 ::METHOD init
   expose password
   self~init:super      /* let superclass initialize 1t */
        /* Create object to allow PW Change */
   .smpPwChange~new('Change Password', 'ICONFILE=' || .lock_icon || ';', self, 1)
   if \var('PASSWORD') Then     /* PW initialized via SetupString? */
      password = ''     /* Nope, give default '' */

::METHOD wpOpen
  expose password
  use arg handleContainer, view, params
  if password == '' then        /* no password set? */
     return self~wpOpen:super(handleContainer, view, params)    /* go ahead and open this*/
        /* Ask user for password. */
  enterpw = self~ask4Password('Locked Folder Password', 'Enter Password')
  if password = enterPw then Do /* Was correct password entered */
        /*Yup, forward to WPFolder top Open */
     return self~wpOpen:super(handleContainer, view, params)
  End
  else Do       /* Incorrect pw entered. */
     reply .false       /* Return failure, and return to WPS */
     guard off          /* Now display error to user. */
     self~displayError('LockFolder Error! [should be: "' || password || "]" )
  End
```

Figure 16: Class “WPLockFolder” with the Methods “init” and “wpOpen”.

```
::METHOD wpSetup
   use arg setupString
   maxLength = 64
   strLength = .DLFULong~new(maxLength) /* Will allow for up to 64 char PW */
        /* Get INOUT String parm */
   str = .DLFString~new~~_set_maxSize(maxLength)

        /* see if setup string has PW */
   if self~wpScanSetupString(setupString, 'PASSWORD', str, strLength) then
      self~password = str~asString      /* Yup, set password. */

   return self~wpSetup:super(setupString)       /* Superclass does remainder. */

 ::METHOD scrollTitle unguarded         /* unguarded, want to run concurrently*/
   title = self~wpQueryTitle            /* Get current title */
   do 2         /* Will scroll twice. */
      do i = 1 to title~length          /* For length of title. */
         self~wpSetTitle(right(left(title, i), title~length)) /* display 1st 1 titlechars*/
      end
   end
```

Figure 17: Methods “wpSetup” and “scrollTitle”.

This would be a simple but effective behaviour for a new password protected folder

```
::CLASS VXPWPrompts mixinclass object
 ::METHOD ask4Password          /* ask for a password */
   use arg title, prompt
   buttons  = 3         /* use "OK"- and "CANCEL"-buttons */
   prompt.0 = 1;                        /* prompt */
   if arg(2, "E") then prompt.1 = prompt
                  else prompt.1 = 'Password'
   width.0  = 1; width.1  = 64          /* widths in character units */
   hide.0   = 1; hide.1   = .true       /* don't echo PW */
   answer.0 = 1; answer.1 = ''          /* default value: empty string */
   call VDialogPos 50, 50       /* center message box on screen */
   button = VMultBox(title, "prompt", "width", "hide", "answer", buttons)

   if button = 'OK' then return answer.1        /* return entered password      */
   return .nil          /* "CANCEL" pressed; indicate no PW entered     */

 ::METHOD displayError
   use arg msg
   do i=1 to 10 while msg <> ""
      pos = length(msg)
      if pos > 80 then          /* VRexx allows 80 chars per msg-line only */
      do
         pos = lastpos( " ", msg, 80)   /* try to break at a blank      */
         if pos = 0 then pos = 80       /* no blank in first 80 chars, force break */
      end
      msg.i = substr(msg, 1, pos)       /* assign chunk to msg-stem */
      msg = substr(msg, pos+1)
   end
   msg.0 = i                    /* assign message */
   call VDialogPos 50, 50       /* center message box on screen */
   return VMsgBox('Important error message!', "msg", 1) /* show OK-button only */
```

Figure 15: The Class for Prompting the Password.

class, *if* this new class would specialize the pre-fabricated WPS class *WPFolder*! In addition, if using specialization one could fully re-use all of the behaviour implemented (and tested) in all superclasses, which means, that one needs not to implement any of the behaviour of *WPFolder*, but rather re-uses it. Should future enhancements be implemented in the WPS, then these enhancements would be available to the subclass as well, without the need to adjust anything in the new password protected folder.

4.2.2 Password Protected WPS Folder: Implementation in Object Rexx

The implementation of a password protected WPS folder class in Object Rexx should be feasible, as all the needed infrastructural access is available to that programming language. Figures 14 through 19 constitute the entire Object Rexx program and their contents would reside in the same program.

```
::METHOD password ATTRIBUTE

::METHOD wpSaveState                    /* Save the password data */
  self~wpSaveString(self~ somGetClassName, 1, self~password)
  return self~wpSaveState:super         /* Let parent save any state. */

::METHOD wpRestoreState
  self~initButtons                      /* make sure OREXX side initialized. */
  size = .DLFULong~new                  /* Get DLFUlong for size query. */
       /* Retrieve size of string for restore */
  self~wpRestoreString(self~ somGetClassName, 1, .nil, size)
       /* Create DLFString large enough to contain the string, plus NULL */
  str = .DLFString~new~~_set_maxSize(size~_get_value + 1)
       /* Now get saved password */
  self~wpRestoreString(self~ somGetClassName, 1, str, size)
  self~password = str~asString          /* Save password state value. */
       /* let parent restore state. */
  return self~wpRestoreState:super(arg(1))
```

Figure 18: Methods "password", "wpSaveState" and "wpRestoreState".

The implementation will do the following:

- It will use a freeware package "VREXX" for prompting a password from the user with the help of a popup-window. This package was created by the IBM employee Steve Lamb and has been published with IBM's "employee written software" (EWS) program for OS/2 (cf. figure 14 which shows the loading of this Rexx function package).

- For prompting for a password the Object Rexx class *"VXPWPrompts"* is created. It controls and executes the popup-window for inputing the password (cf. figure 15).

- There will be one Object Rexx class *"WPLockFolder"* which subclasses the WPS class *"WPFolder"* inheriting all the features and behaviour of regular WPS folders (cf. figure 16). This class intentionally demonstrates the ability to employ multiple inheritance by declaring this class to have in addition *"VXPWPrompts"* as its other superclass. Therefore messages sent to an instance of *"WPLockFolder"* could be ultimately addressed to the methods available in *"VXPWPrompts"*. In addition, a method *wpOpen* is supplied which overrides the *"wpOpen"* message. At this point the user is prompted for a password. If it is correct, then the message is forwarded to the WPS class

```
::CLASS SMPPWChange SUBCLASS WPAbstract
 ::METHOD wpOpen
   use arg handleContainer, view, params
   if view \= 2 & view \= 3 then Do     /* Opening Default view? Dbl-click */
      lockf = self~wpQueryFolder        /* Get our containing lock folder  */
        /* Ask for new password */
      newpw = lockf~ask4Password('New LockFolder Password', 'Enter New Password' )
      if newpw \= .nil Then Do          /* Get a new password? */
         lockf~password = newpw         /* Yup, set new pw. */
         lockF~wpSaveImmediate          /* Save object state (PW) */
      End
      return 0
   End
        /* Forward wpOpen to super class to handle. */
   forward class (super)
```

Figure 19: Class "SMPPWChange" for Changing Existing Password.

"WPFolder" and that folder gets opened, otherwise this forwarding is inhibited and an error poup-window is concurrently displayed, which in this particular example would (intentionally) unveil the correct password.

- The password is stored as part of the instance data of this class, using the setup-string functionality, which by default is implemented in the *WPObject* method *wpSetup* (cf. figures 17 and 18, methods *wpSetup*, *wpSaveState*, *wpRestoreState*).[9] For the first time the user is prompted for a password at the creation of an instance of the *"WPLockFolder"* class and later on whenever this folder should get opened.

- In order for the user to change a password an abstract object will be placed into the password protected folder and by double-clicking it the appropriate popup-window displays prompting for a new password for that particular folder. That object is defined as an Object Rexx implemented WPS class, named *"SMPPWChange"* and depicted in figure 19.

As one can infer from these examples, it is rather easy to create a password protected folder in Object Rexx, given that all relevant information is put together.

---

[9] The class *"DLFSTring"* is defined in *"\os2\dlfclass.cmd"* as mentioned eralier, and allows to use INOUT and OUT parameters in SOM. In these cases the called methods set the content of such arguments, which after the call can get retrieved by Object Rexx programs.

5 SUMMARY AND OUTLOOK

This article introduced the reader to the IBM developed technologies SOM, DSOM, WPS and Object Rexx, which all have been available as products for years. Especially (corporate) users of OS/2 have employed these state of the art products. This article researched and attempted to put together all dispersed information necessary to employ the SAA scripting language Object Rexx to solve interesting problems in a manner, which is simply not possible (yet) with other platforms and technologies. Although IBM has been maintaining D/SOM and WPS for their customers, no new features have been added to them in years, rather much of its investments w.r.t. software development technologies has been directed towards Java.[10)]

The state of Object Rexx is different: since its inception 1996 with OS/2 4.0 it has been made available to AIX, Linux, Windows 95, Windows 98 and Windows NT. The Windows version got an addition with a so-called “developer edition” allowing Object Rexx to natively use the Windows graphical user interface to create user interface windows. As the Windows user interface has not been fully implemented in an object oriented style like the WPS, it is still not possible to extend the Windows UI with Object Rexx defined classes. With Object Rexx Windows-based feature enhancements like the OLE/ActiveX automation support one can conclude, that eventually one will be able to achieve these tasks with the Windows environment sometimes in the future. This goal would be even more likely, if IBM planned to enhance the Object Rexx platform support on Windows operating systems with Microsofts “Windows Scripting Host” (WSH) facility, which would allow for using the scripting language Object Rexx as a native Windows scripting (batch) language as well as a fully supported macro language for most Windows applications.

[10)] Interestingly, it has become almost impossible to research the Internet about information and books dealing with D/SOM and WPS, neither on IBM websites nor outside. This indicates that the world wide web is not feasible to be used as an archive of documents which originally were posted via it. This has also been a motivation to create this article, because in this form researched knowledge remains available to the interested researcher.

6 REFERENCES

[BiDiSh97] Bitterer A., Dijkstra V., Shingarow B.: “VisualAge for Smalltalk Handbook - Volume 2: Features”, Redbook SG24-2219-00, IBM 1997. Available online, URL (2000-05-19): `http://www.redbooks.ibm.com`

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

[Flat96a] Flatscher R.G.: "Local Environment and Scopes in Object REXX", in: Proceedings of the "7th International REXX Symposium, May 12-15, Texas/Austin 1996", The Rexx Language Association, Raleigh N.C. 1996.

[Flat96b] Flatscher R.G.: "Object Classes, Meta Classes and Method Resolution in Object REXX", in: Proceedings of the "7th International REXX Symposium, May 12-15, Texas/Austin 1996", The Rexx Language Association, Raleigh N.C. 1996.

[Flat96c] Flatscher R.G.: "ORX_ANALYZE.CMD - a Program for Analyzing Directives and Signatures of Object REXX Programs", in: Proceedings of the "7th International REXX Symposium, May 12-15, Texas/Austin 1996", The Rexx Language Association, Raleigh N.C. 1996.

[Flat97] Flatscher R.G.: "Utility Routines and Utility Classes for Object Rexx", in: Proceedings of the "8th International Rexx Symposium, April 22nd-24th, Heidelberg/Germany 1997", The Rexx Language Association, Raleigh N.C. 1997.

[GolRob83] Goldberg A., Robson D.: “Smalltalk-80 - The Language and Its Implementation”, Addison-Wesley, Reading 1983.

[OrHaEd96] Orfali R., Harkey D., Edwards J.: “The Essential Distributed Objects Survival Guide”, John Wiley & Sons, New York 1996.

[TurWah97] Turton T., Wahli U.: "Object Rexx for OS/2 Warp", Prentice-Hall, London 1997.

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

[WahHolTur97] Wahli U., Holder I., Turton T.: "Object REXX for Windows 95/NT, With OODialog", Prentice Hall, London 1997.

[W3DecAri] Homepage of the decimal arithmetic initiative, URL (2000-05-19): http://www2.hursley.ibm.com/decimal/decimal.html

[W3Hobbes] WWW-Repository for useful Rexx programs originally developed under OS/2, URL (2000-05-19): http://hobbes.nmsu.edu/

[W3MS_WSH] Microsoft Windows Scripting Host information, URL (2000-05-19): http://msdn.microsoft.com/scripting/windowshost/default.htm

[W3ObjRexx] Object Rexx homepage of IBM, URL (2000-05-19): http://www.ibm.com/software/ad/obj-rexx/

[W3Rexx] Rexx homepage of the creator of the language, the IBM fellow Mike Cowlishaw, URL (2000-05-19): http://www2.hursley.ibm.com/rexx/

[W3RexxLA] Rexx homepage of the “Rexx Language Association”, URL (2000-05-19): http://www.RexxLA.org