DSC 2003 Working Papers
(Draft Versions)
http://www.ci.tuwien.ac.at/Conferences/DSC-2003/

R: Windows Component Services Integrating R and Excel on the COM Layer

Thomas Baier

Abstract

On the Windows platform a standardized model for component services and interprocess and inter-machine communication has been established by Microsoft—COM. As many applications provide their own or use third party COM services, extending R by means of COM interfaces is a natural way for integration of R into the user's desktop.

This paper discusses the mechanisms available to R users for integration of R into Windows applications and vice versa.

As an example for this integration, we will show how to utilize the COM services provided by Excel from R and how to use R functionality from within the spreadsheet's macro language VBA.

1 COMponent Services

Reusing software components across programming languages has always been a major problem. Even within the same programming language problems will show up when combining compilers from different vendors, mostly due to incompatibilities in the language implementation, the code generation or just the runtime library (e.g. used for dynamic memory allocation).

Different approaches to the problem of component reuse have been defined with different purposes in mind. The CORBA[1] specification has been developed in the early 90's by the Object Management Group (see [4]) and is supported by many vendors on different platforms. On Windows, a very similar—but incompatible—object model has been developed by Microsoft, COM[2]. The infrastructure for COM now is part of the operating system and using this model for sharing components ("sharing implementation") is accepted practice.

[1] Common Object Request Broker Architecture
[2] Component Object Model

Both, COM and CORBA, are very similar. Both define standards for exposing components using public interfaces. The interfaces contain methods (functions) bot do not contain object data. Data has to be exposed via functions (properties, property access functions). The internal implementation of the components themselves are up to the implementor or compiler vendor. Part of the specification of course also deals with invoking methods of an object. Objects can run on the local machine or even remotely, where the component subsystem (COM or CORBA) takes care of the required network communication.

In addition to public interfaces and local or remote function calls, COM and CORBA define mechanisms for object creation and lifetime management and a set of well-defined data types.

The major idea behind COM is to provide a standard (see [2]) for interoperability of components from different software vendors. A COM component (COM object) is provided by a COM server application. COM servers can be implemented in different ways. According to the application requirements one must find a balance between safety and speed and choose from one of these:

- A COM object can be instantiated like any other library object in the memory space (process boundaries) of the client application. This provides the fastest possible access to the COM object with the least possible overhead—similar to calling a virtual function on a C++ object or calling a C function by a function pointer. This kind of implementation is called an *In-Process Server*.
- Next, a COM object can be run in a different process. Calls to this COM object will lead to some interprocess communication (transfer of data from one memory space to another, invoking the function). This provides more safety for both the client and the server application. If the client fails, the server continues to run, if the server fails, the client can handle this, too. This is called an *Out-of-process Server*.
- DCOM[3] also provides mechanisms for accessing COM objects running on remote servers. DCOM takes care of the required network communication and synchronization. This is similar to an out-of-process, but, again, one magnitude of order slower.

In addition to the "plain" COM standard, Microsoft has defined some additional specifications for various purposes. These specifications e.g. cover specific components used in transaction processing (COM+), user interface controls as COM objects (Active X controls) or scripting languages (Automation).

2 Accessing an Application's Services

As shown in the last section, COM objects are provided by a so-called "COM server" application, while they are used by a "COM client".

Using the add-on package "`rcom`", one can—very similar to what is known from Microsoft Office—use R to access services and components provided by other applications and therefore act as a COM client. The approach taken here is very similar to the integration of COM servers into other systems (e.g. XLisp-Stat [6]).

[3]Distributed COM

2.1 Utilizing Excel's COM services

Excel as a spread-sheet application provides very intuitive means for entering data in tabular form. But while Excel is perfectly suited for entering and visualizing data, its' computational engine itself can only be used for very simple computations.

This is contradictory to R, which provides a very strong computational engine and a powerful language for writing your own functionality, but on the other hand is not as well suited for building a GUI. Even if there are some means for entering data, in many cases, Excel just is the "natural" tool for doing data entry and manipulation on the Windows desktop and familiarity with the user interface and the application itself plays a very important role for the users to be comfortable with a user interface.

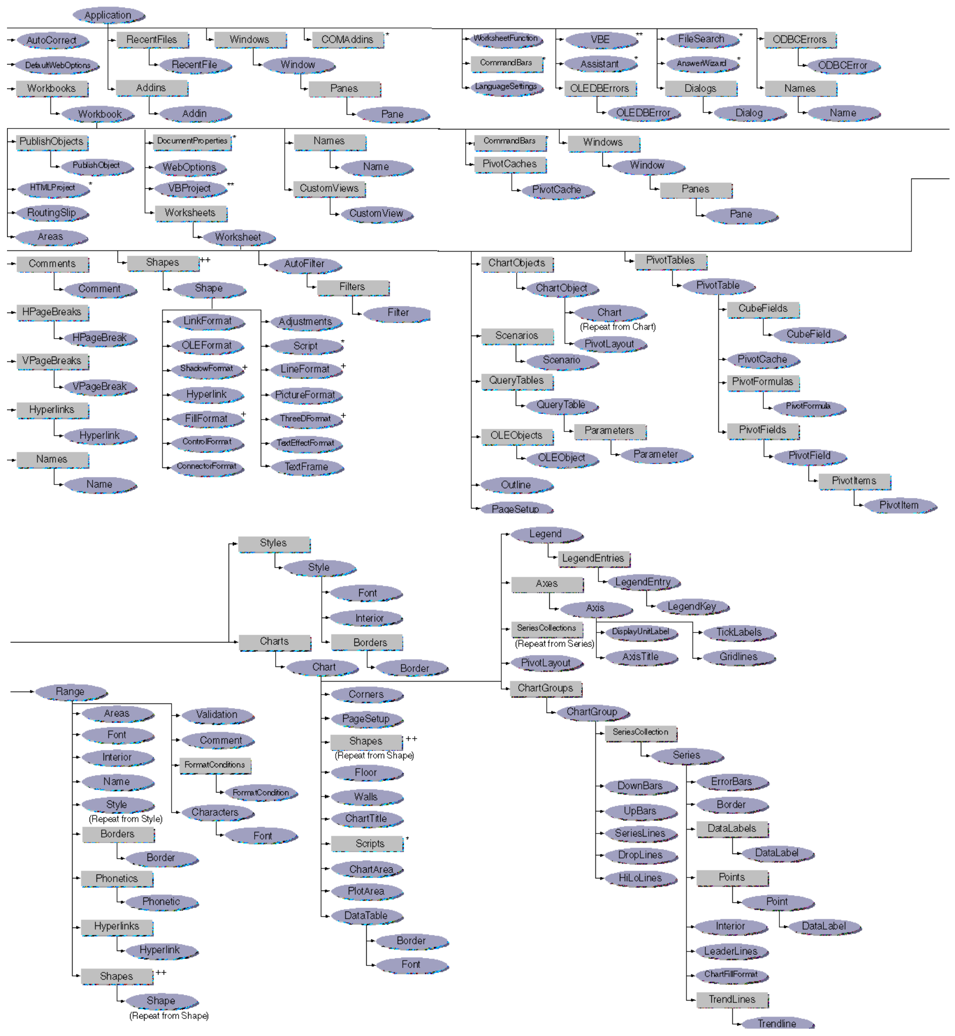

Figure 1: Microsoft Excel Object Model

Fortunately, Excel provides many COM components and services to all applications acting as a COM client. Although in most cases, Excel's object model (see figure 1) is used by VBA[4] scripts running inside Excel itself, by providing this model as COM objects, every custom application (including R) can make use of Excel's functionality.

2.2 Retrieving Data From Microsoft Excel

To find out how to use Excel as a medium for data entry (and presentation) for R, consider a simple example: the user has entered a data vector containing 20 floating point values in the currently active sheet in Microsoft Excel. The values are in cells `A1` to `A20`. We now want to

- read the data vector from Excel into an R variable (`val1`)
- compute the square of each element in `val1` and store the resulting vector in `val2`
- write `val2` (same dimensions as `val1`) back to Excel into cells `B1:B20` and at last,
- create a line plot showing `val1` and `val2` in a single diagram in Excel shown on its own sheet.

When browsing Microsoft's documentation for the Excel Object interface (see [3]) we can find out that we have to use

- Excel's `Application` object (as the root of our object model) to get access to the currently active sheet.
- Then we have to define two `Range` objects for our data vectors (`range1` for cells `A1` to `A20`, `range2` for cells `B1` to `B20`). Accessing the data (reading and writing) is just a matter of getting/setting the `Range`'s `Value` property.
- At last we create a new graph for the range object `A1:B20` and set it to display the data as a *Line Graph*.

The following section will show how to achieve this from R code using the package `rcom`.

2.3 Accessing Excel from R

From `rcom`'s point of view, the world of COM consists of objects, functions and properties.

Objects are provided by COM servers (like Microsoft Excel). Properties represent the internal state of an object. Functions on the other hand, provide the functionality of the object.

Properties are the object's variables. Properties can be set to some value or R can retrieve a property's value. Properties can be as simple as scalar values or can represent whole COM objects (e.g. Excel's `Application` object has a property called `ActiveSheet` which holds a *reference* to the currently active sheet edited by

[4]Visual Basic for Applications

the user). Properties also can be indexed by number or by string values to identify a specific part of the property (e.g. see the `Range` property used in section 2.3.2).

Functions (methods) of a COM object can be invoked. You can pass any number of parameters (again, as simple as numbers, strings or as complex as whole COM objects). Functions also can return a result.

2.3.1 Getting a Handle to the Excel Application

When accessing any of Excel's functionality, we have to first get a handle to Excel's `Application` object. `rcom` provides two different means for getting an object handle:

- `com.object.create` will create a new instance of the COM object, whose class name is passed to the function, whereas
- `com.object.get` will search for the system for a handle to a globally registered instance of the passed object class name.

To get a handle to an already running Excel application, we use the function `com.object.get`. Afterwards, the application object is queried for the currently active sheet the user is editing.

```
excel<-com.object.get("Excel.Application")
sheet<-com.property.get(excel,"ActiveSheet")
```

2.3.2 Retrieving a Data Vector from Excel

To retrieve a data vector for cells `A1` to `A20`, Excel requires to use a `Range` object. The range object then is simply queried for its value, which is stored in `val1`.

```
range1<-com.property.get(sheet,"Range","A1","A20")
val1<-as.double(com.property.get(range1,"Value"))
```

Memory management is done automatically. COM's reference-counted memory management scheme is integrated into R's garbage collection mechanism. See section 2.4 for more information on this topic.

2.3.3 Writing Data To Excel

Writing data back to Excel is as simple as reading the data and very similar in code:

```
val2<-val1 * val1
dim(val2)<-c(20,1)
range2<-com.property.get(sheet,"Range","B1","B20")
com.property.set(range2,"Value",val2)
```

2.3.4 Create a New Chart

As now the new data is displayed in Excel, Excel's built-in reporting functionality is used to add a graph displaying both `val1` and `val2` in a simple line graph.

To create a new graph, simply invoke the function `Add` on the active workbook's chart collection.

```
wb<-com.property.get(excel,"ActiveWorkbook")
chart<-com.invoke(com.property.get(wb,"Charts"),"Add")
```

2.3.5 Adjusting the Chart's Properties

Setting the chart's properties is easy as well. The only thing to take care of is that the Excel documentation uses symbolic names for various constants which are available in Excel's VBA implementation. These are currently not available using `rcom`, so you have to directly use the expansions of these constants.

```
com.property.set(chart,"ChartType",65)
```

Finding out which are the numbers the constants are mapped to is possible by using a simple *type library browser* (e.g. `OLE View` provided by Visual Studio) to inspect Excel's data type information (*type library*).

In the above example, e.g. 65 stands for the constant `xlLineMarkers`.

2.4 Some Notes on Reference Counting

While memory management inside R is handled automatically by a garbage collector, which will re-use the memory allocated by an object at some time when no more references to this object exist, COM follows a strategy of explicit reference counting.

Every single COM object implements the functions `AddRef()` and `Release()` which are used to increment and decrement the reference count for the object. As soon as the last reference is released and therefore the internal reference count of the COM object reaches 0, the object gets released immediately.

`rcom` integrates COM's reference counting scheme into R's garbage collector. When first a COM object gets known to R, the object's `AddRef()` function is called by the COM framework. As soon as the last reference to the object is released, R's garbage collector is free to release and reuse the memory belonging to the COM object (in R's memory space). When the object is garbage collected, it's `Release()` function is called by the COM framework again, which can cause the object (at server side) to be deleted.

It has to be taken into account, that COM objects accessed by `rcom` are not released immediately when the last reference to the object vanishes. If you want to make sure an object gets deleted properly at some specific point in time, you should take care to manually set the references to 0 for global variables and afterwards invoke R's garbage collector by calling `gc()`.

3 Providing COM Services

The previous section showed, how R can utilize services provided by other applications in the form of COM objects. While this will allow an *R-centric* integration of R with other applications running on the Windows platform, this is only part of the solution.

In the same way as R can get access to internal resources of Excel, using Excel e.g. for data entry and reporting, it is required, that third party applications can get access to R—both its' data and its' functionality.

The example as depicted in section 2.2 on page 4 is probably not the way most users will want to work.

In fact, the most natural way for solving the problem of entering data in Excel and using just R's computational engine would be to hand control over to Excel itself. Excel, as well as many other applications on the Windows platform, is extendible by using a macro language called *Visual Basic for Applications*. One can create *macros* (and much more) in VBA and add these macros e.g. to the user interface of Microsoft Excel. Users then can access these macros by means of e.g. menu items in the main menu, a context menu or even toolbar buttons.

As control is in the hands of Excel, this requires R to act as a COM server, providing COM objects to its clients.

3.1 R COM Server Interface

The general approach to this topic is to define COM interfaces which will allow any clients to utilize R's data space and functionality.

In principle, the following functionality is required

- transfer a data item (scalar or array/matrix) from a client application to R and assign it to some variable/identifier,
- retrieve data from R identified by some variable name,
- call functions running in the name and data space of R.

A COM interface providing the functionality as shown above already has been defined (see [1]). In addition to the functionality shown above, the interface provides functionality e.g. for house-keeping and error-handling.

In the following, most of the functionality provided by the COM interface is shortly discussed. The interface definition of the functions is written in Microsoft's IDL syntax.

3.1.1 Data Exchange

COM defines a generic data type called `VARIANT` which can in principle hold any scalar or compound data type—even COM objects.

As there is no common representation in the COM system of something like a `data.frame` or a `list`, these type are currently not supported.

```
HRESULT GetSymbol([in] BSTR bstrSymbolName,
                  [out,retval] VARIANT* pvData);
HRESULT SetSymbol([in] BSTR bstrSymbolName,
                  [in] VARIANT vData);
```

Variables of all supported data types can be transferred from R to any client application using `GetSymbol()`, where the first parameter represents the name of a symbol in R's global namespace.

Transferring objects (data) from the COM client (e.g. Excel) to R and binding this data with a symbol is achieved using `SetSymbol()`.

3.1.2 Invoking Functions in R

As sometimes R functions/expressions can return a large amount of data, there are two different functions provided for evaluating an expression in the global context of the R interpreter.

```
HRESULT Evaluate([in] BSTR bstrExpression,
                 [out,retval] VARIANT* pvData);
HRESULT EvaluateNoReturn([in] BSTR bstrExpression);
```

The only difference between `Evaluate()` and `EvaluateNoReturn()` is, that the latter function will discard any result of the evaluation, therefore saving the probably time- and memory-consuming transfer of the result to the caller.

Calling one of these functions is like typing the expression at the R command prompt and evaluate it.

3.1.3 Initialization and Termination

In case R is started from a client application, there is are explicit functions clients use to initialize and terminate a session with R.

```
Init([in] BSTR bstrConnectorName);
HRESULT Close();
```

Before calling `Init()`, the client must not do any data transfer or evaluation in the context of R. The same holds true after calling `Close()`.

The parameter passed to `Init()` makes it possible to optionally pass parameters to the R process wich would normally have been passed on startup (using command line switches).

3.1.4 Error Handling

As using R as a COM server can cause many non-trivial problems for a client application (e.g. failure of some COM function-calls), there are some functions in the interface especially dedicated to error-handling. Error-handling is a critical part of this interface, to some extent because the strings passed to `Evaluate*()` or `Get/SetSymbol()` cannot be checked at “compile time” for syntactic or semantic errors.

```
HRESULT GetErrorId([out,retval] LONG* pulErrorId);
HRESULT GetErrorText([out,retval] BSTR* pbstrErrorText);
```

Every single function can return one of the pre-defined error-codes (e.g. “symbol not found”). For use with automation clients (e.g. *VBScript* etc.) the last error id can also be retrieved by calling `GetErrorId()`. Additionally, a string in english language describing the error can be retrieved by calling `GetErrorText()`.

This minimum diagnostics interface will allow the developer/user to find the source for most errors and fix the problem. See table 1 on page 9 for some of the more common error codes and the descriptive text for these.

3.2 R as an Invisible Computational Backend

One of the advantages of COM is, that it allows *interface-based* programming. The COM client-applications don’t directly “program” against a library or something similar, but they use a *COM interface*. This interface defines a set of functions (and properties) every single implementation of the interface must provide.

The interface is defined independently from the implementation and the COM system allows multiple implementations for the same interface.

Code	Error Text
0x80040010	installation problem: interpreter interface version mismatch
0x80040011	installation problem: interpreter version mismatch
0x80040012	installation problem: unable to load interpreter interface
0x80040013	installation problem: unable to load connector
0x80040014	installation problem: invalid connector library
0x80040015	installation problem: interpreter initialization failed
0x80040016	invalid connector name specified
0x80040020	unexpected fatal error in back-end implementation. release the object!
0x80040007	symbol not found
0x80040008	invalid expression
0x80040009	incomplete expression
0x8004000a	unsupported data type requested
0x8004000b	evaluation stopped because of an error
0x80040001	invalid argument passed in function call
0x80040005	interpreter already initialized
0x80040006	interpreter not initialized

Table 1: Common Error Codes returned by R COM Server

This is a feature of COM which is used by R's COM services. While just a single interface (`IStatConnector`—excerpts from the interface definition are shown in section 3.1)—is defined, which will then provide any COM client with the possibility to access R's data and functionality, different implementations have been created for different applications.

As said before, the client application programs against the object's interface, and it is up to the client application, which implementation—which COM server—to choose then. The concrete implementation of the interface to be used is identified by its *class id*. This is a unique identifier associated with every single COM object. In addition to the class id, which is unique, but not really an identifier in a human-readable form, every COM server is also identified by a so-called *program id*, which is a string and can easily be remembered by the programmer.

On the one hand, there is a stand-alone COM server, which has already proven successful in many applications (e.g. see [5]). It is provided as an add-on package to R for Windows and has to be installed separately. This implementation is available from CRAN at `http://cran.r-project.org/contrib/extra/dcom/` and can provide full integration of R into a custom application.

The goal of this implementation is to fully embed R into a custom application, e.g. into Microsoft Excel or any other application acting as a COM client.

Some examples of concrete applications of this COM server I know of are:

R/Excel Interface This interface has been done by Erich Neuwirth (see [1]). It provides full embedding of R into Microsoft Excel and makes it possible to use R as the computational back-end while still preserving Excel as the only user interface.

Medieval Chant Analysis This is software system supporting the analyses of manuscripts for medieval chants done by Brian D. Ripley and Ruth M. Ripley. The system is written in Visual Basic and provides an easy-to-use front-end to inexperienced computer users. The complexity of both a database system behind the application and R are completely hidden from the user.

To really achieve the goal of embedding R as the computational engine into custom applications, the COM interface also defines some additional functions to be used here:

- Active X controls for displaying R text output (but not input) can be put inside your application. R then can perform text output directly at some reserved place (e.g. this is then where the output of `print()` statements will show).
- Another Active X control is provided as an area for R to display graphics output. Put on a form, R's graphical capabilities (e.g. `plot()`) can be used and the output is redirected to a place inside the custom application.

3.2.1 Accessing R from Excel

From the Excel point of view, R has to act as a COM server. To implement the example depicted in section 2.2 letting Excel control R, we have to use VBA. What we have to do is

- get a handle to an R interpreter
- transfer the Range object for cells `A1` to `A20` to R
- compute the results in R and get back these results
- the results then are stored in cells `B1` to `B20`

Creating the graph in Excel is not considered here, but is similarly easy as shown before.

3.2.2 Getting a Handle to R

To access R, we have to create a COM object representing R. This is very similar to what we did before (see section "Getting a Handle to the Excel Application" on page 5) to access Excel.

```
Dim R As StatConnector

Set R = New StatConnector
R.Init "R"
```

The code shown above is in Excel's macro language *Visual Basic for Applications*. This language is very similar to *Visual Basic* and provides a full-featured development environment for Excel Macros (including e.g. a dialog editor, a debugger etc.).

First, an object for accessing R is created (the variable name referring to this object is `R`, the object's class is `StatConnector`), then this object is initialized. After the call to `Init()`, the R interpreter can be used.

3.2.3 Writing Data to R

To transfer the data from the first range to R and store it into `val1` we have to use the following code

```
Dim range1 As Range
Dim val1(1 To 20) As Double
Dim i As Integer

Set sheet = Application.ActiveSheet
Set range1 = sheet.Range("A1", "A20")

For i = LBound(val1) To UBound(val1)
  val1(i) = range1(i).Value
Next i
R.SetSymbol "val1", val1
```

Unfortunately, the `Value` property of the `Range` object returns an array of `VARIANT`s, which is currently not directly supported by the COM server's `SetSymbol()` function. Therefore, the explicit conversion to an array of `Double`s (see the `For ... Next` loop) has to be applied.

3.2.4 Computing the Result and Transferring Back to Excel

Next, `EvaluateNoReturn()` is used to compute the result as before and store it in `val2`.

```
Dim range2 As Range

Set range2 = sheet.Range("B1", "B20")
R.EvaluateNoReturn "val2<-val1 * val1"
R.EvaluateNoReturn "dim(val2)<-c(20,1)"
range2.Value = R.GetSymbol("val2")
```

The call to `GetSymbol()` then transfers the data back to Excel and directly stores the values in the `Range` object.

3.3 R as COM Server and Client Side by Side

The full integration of R into a custom application undoubtedly has many advantages. One of the most appealing fact is, that R—both as a system and its complexity for many users—are completely hidden from the user's point of view. The user mostly even doesn't know, that the real work behind the scenes is done by R.

On the other hand, it can be desireable for advanced users to have both COM client and COM server running side by side. E.g. one could use Excel for data entry and send data from Excel to R using R's COM server, then switches over to R and types some commands at the command line to use the data.

This is possible using a second implementation of R's COM interface which has been shown in section 3.1. The previously mentioned package **`rcom`** (see section 2 on page 2) not only provides functions for R to act as a COM client but also automatically registers an additional COM server for the running R process.

As soon as the package is loaded, R's internal COM server is registered at the system and R can be accessed by other applications.

Using the stand-alone COM server described in section sec:statconnector normally every single client application starts its own COM server and gets its own instance of R. Two different clients then will then have a different data and function space to use.

The internal COM server behaves different. This server is *shared* for all clients. The first R process loading `rcom` will register itself as a COM server for this machine. All client applications using this COM server will have access to the same data and function space.

In addition to that, adding graphics devices or text/error output devices (the ActiveX controls provided with the stand-alone server) is silently ignored at the moment.

Translating our previous example from using the stand-alone COM server to using the internal COM server provided by `rcom` will follow this strategy:

- The user enters data in Excel in cells `A1` to `A20`. Then the user sends the data from Excel to R's internal COM server
- Afterwards the user switches to the R window, computes the result and sends this result back to Excel.

3.3.1 Coding in Excel: Sending Data to R

The solution described above is implemented partially in VBA:

```
Dim R As StatConnector
Dim range1 As Range
Dim val1(1 To 20) As Double
Dim i As Integer

Set R = New InternalConnector
R.Init "R"
Set sheet = Application.ActiveSheet
Set range1 = sheet.Range("A1", "A20")

For i = LBound(val1) To UBound(val1)
  val1(i) = range1(i).Value
Next i
R.SetSymbol "val1", val1
```

The code looks nearly the same as before. The call to `Init()` can be removed. If not, it will return a success code (no error) `S_FALSE`.

How does Excel know, which COM server to use? The code looks the same, only creating the COM server (`New InternalConnector`) is different.

In Excel, a reference to the COM server to use is added in the GUI part of the system. In these two cases, just different references to different COM servers are assumed.

Additionally, one can use the server's *program id* to directly specify in code which COM server to use. For example, you can replace the creation of the COM server with something like

```
Set R = CreateObject("StatConnectorSrv.StatConnector")
```

if you want to use the stand-alone COM server. To use R's internal server (provided by `rcom`), type

```
Set R = CreateObject("RCOMServerLib.InternalConnector")
```

The rest of the client application is still the same.

3.3.2 Computing and Sending Back the Results

After the VBA code shown in the last section has been executed, in R's symbol space there's a global symbol `val1`. Now we can switch over to the R window and use this symbol there. The result gets computed and is sent back to Excel using R's COM client functionality:

```
excel<-com.object.get("Excel.Application")
sheet<-com.property.get(excel,"ActiveSheet")
val2<-val1 * val1
dim(val2)<-c(20,1)
range2<-com.property.get(sheet,"Range","B1","B20")
com.property.set(range2,"Value",val2)
```

4 TODOs for a Better Integration

This paper has described some low-level mechanisms provided for R to act as both a COM client and a COM server. While this is all required for connecting R to any COM client or server, the interfaces are not what user's will expect.

As an example, I would like to look at an integration of R and Excel. The work done by Erich Neuwirth to integrate R's COM servers into Microsoft Excel's user interface (see [1]) clearly shows the right direction for integration. The same is currently work in progress for the R side, where an Excel-specific package is created, providing high-level functionality to access Excel. The interface `rcom` currently provides is not as user-friendly as an R-Excel interfaces should be.

References

[1] Thomas Baier and Erich Neuwirth. Embedding R in standard software, and the other way round. In Kurt Hornik and Friedrich Leisch, editors, *DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing*, http://www.ci.tuwien.ac.at/Conferences/DSC-2001/, 2001. ISSN: 1609-395X.

[2] Microsoft Corporation and Digital Equipment Corporation. The component object model specification. Technical Report 0.9, Microsoft Corporation, October 1995. Draft.

[3] Microsoft Corp. Microsoft Excel 2000. In *Microsoft Office 2000 Developer Object Model Guide*, MSDN Library. Microsoft Corp., July 2001.

[4] Inc. Object Management Group. Common object request broker architecture: Core specification. Specification 3.0, Object Management Group, Inc., Nov. 2002.

[5] B.D. Ripley and R.M. Ripley. Applications of R clients and servers. In Kurt Hornik and Friedrich Leisch, editors, *DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing*, http://www.ci.tuwien.ac.at/Conferences/DSC-2001/, 2001. ISSN: 1609-395X.

[6] Luke Tierney. Connecting Lisp-Stat to COM. *Journal of Computational and Graphical Statistics*, 9(3):459–469, September 2000.