We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



Chapter 1

MATLAB COM Integration for Engineering Applications

Mariano Raboso, María I. Jiménez, Lara del Val,
Alberto Izquierdo, Juan J. Villacorta and Myriam Codes

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/46471

1. Introduction

COM (Component Object Model) is a Microsoft framework designed for Windows platforms for developing and integrating software components. Software components and reusability techniques have interesting advantages, as component base software engineering has shown through the last years.

The most powerful idea around component-based software, is that components can be implemented by a programmer and reused by others without having knowledge of the source code. Components are binary packages that can be deployed and further integrated with others written on different programming languages. As component selection and integration is usually an easy and well-known process, components are also called COTS (Commercial Off-The-Shelf).

Software components are also very useful for evaluating several implementations for different vendors. Engineers can analyse and compare them in terms of cost, performance and security. Furthermore, component software integration is a key tool for rapid-prototyping software developments.

A component may be implemented with a high specialized language suitable for specific tasks and used by clients written on more general languages. For example, we could be interested on implementing a specialized component in Matlab, and integrating it into a GUI written on Visual Basic or Tcl/Tk. This situation may be comparable to software written on assembly language (specific and low-level) and linked into C programs using libraries or object code.

On the other hand, using software components also involves some risks. Software development must follow its own methodologies, standards and rules that must be taken into account when

INTECH
open science | open minds

© 2012 Raboso et al., licensee InTech. This is an open access chapter distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

integrating external objects made from third parties. Fortunately this can be accomplished by following the rules that the component-based software methodologies suggest.

Microsoft has developed some technologies around COM. OLE (Object Linking and Embedding) and DDE (Dynamic Data Exchange) were the first tools capable of transferring objects between applications and creating links among them. They were available on the earliest Windows versions. In 1996 OLE technology was fused with Internet capabilities and was renamed as ActiveX, providing ActiveX controls, Active Documents and Active Scripting.

COM+ are COM based services first developed for Windows 2000. They extend COM technology with advanced services to manage resource pools, disconnected applications and event publication.

DCOM (Distributed Component Object Model) is a Microsoft technology that enables communication links among distributed components running on network interconnected machines. DCOM extends COM and COM+ using new services based on DCE (Distributed Computing Environment) and RPC (Remote Procedure Call).

Nowadays, Microsoft recommends using .Net technology instead, as it has already integrated COM services. Many powerful programming languages can use COM components; Visual Basic, Matlab, Visual C++, C# and Tcl/Tk are good examples.

The next sections will describe COM technology and related Matlab resources. Section 2 explains COM basis and history, as well as some terminology that will be useful to the reader for better understanding the following concepts. Section 3 describes Matlab COM interface, specifically the COM automation server and interface methods. Some real examples given will be useful to the novel engineer that wants to work with COM technology. In Section 4, a real application called XBDK that makes use of COM services is described. Finally, some conclusions are made to summarize this technology and to give the reader the opportunity to explore deeper inside COM and .Net technology.

2. Microsoft COM technology

A COM component is an instance of the component object class that runs on the COM server and is accessible from a variety of clients. There are several platforms that can serve COM objects and many clients that can use them.

Matlab COM components are very useful to integrate tasks implemented on this language and exported to others applications. These components can be used later in Microsoft Office Applications (for example Microsoft Excel), Microsoft Visual C++, C#, VB, Tcl/Tk, or even other Matlab clients, in local or distributed applications.

2.1. COM interfaces

COM component implementation is hidden to clients through convenient encapsulation, as the only way to access the component is through a public interface. An interface is a set of

public methods, events and properties declarations defining the way an encapsulated object can be accessed. The component manufacturer is responsible for providing the corresponding interface information, with the necessary method details. As this information is all the knowledge of the component, it is necessary to give as much information as possible in order to ensure the component is suitable to be integrated into a third party system. Interface details are usually transparent to users. Applications as Microsoft Visual Studio provide tools for using COM objects and other resources for .Net platforms (Gunderloy, 2001).

There are four basic COM interfaces:

- IUnknown. It is a basic standard interface that is compulsory to every COM object.
- IDispatch. It is a standard interface for obtaining general information about the object and specifically about methods and properties than can be accessible.
- Custom. It is a custom interface. It can be user defined.
- Dual. It is a combination from IDispatch and Custom.

These interfaces are provided by corresponding server types.

2.2. COM clients

A COM client is a program that uses COM Objects that are provided by COM servers. An example can be a spreadsheet built with Microsoft Excel integrated into a Matlab client. Matlab can be used as a COM client or server.

In this chapter, we are concerned with developing COM server objects using Matlab. Therefore, Matlab client details will be omitted.

2.3. COM server types

COM model defines three types of servers, depending on the interfaces implemented (The Mathworks, 2012):

- **Automation.** This type of server can be accessed by all clients. It supports the OLE Automation standard, and servers are based on IDispatch interfaces.
- **Custom.** These servers are used when a special client requires specific and faster access.
- **Dual.** It is a combination of the above types.

Depending on the locations of the component and client the server can be one of these two types:

- **In-Process server.** In this configuration, the client accesses the components using a DLL (dynamic link library) or an ActiveX. Both client and server run in the same process so they share a unique context.
- **Out-Process server.** The component is implemented as an independent executable (.exe) file. The client can access this program with either local or remote configuration. It depends on the location of the client and server. For local access, performance can be reduced compared to an In-Process configuration. For remote access configuration, it can be only accomplished on systems supporting DCOM (Distributed COM).

2.4. Programmatic identifiers

A programmatic identifier is a unique string that identifies an instance of a COM object. It is usually defined by the vendor.

Users can access different services using different identifiers. For example, Matlab provides three identifiers with several versions to provide such services. If there were more than one version of the software installed, each one would have its own identifier.

Figure 1 shows how a programmatic identifier can be found. It corresponds to an object available in the Windows registry. Besides finding out the existing programmatic identifiers, explore the registry (regedit32.exe) is a good way to verify if the COM servers are properly installed. Another option is to use the Microsoft Visual Studio object explorer or reference explorer.

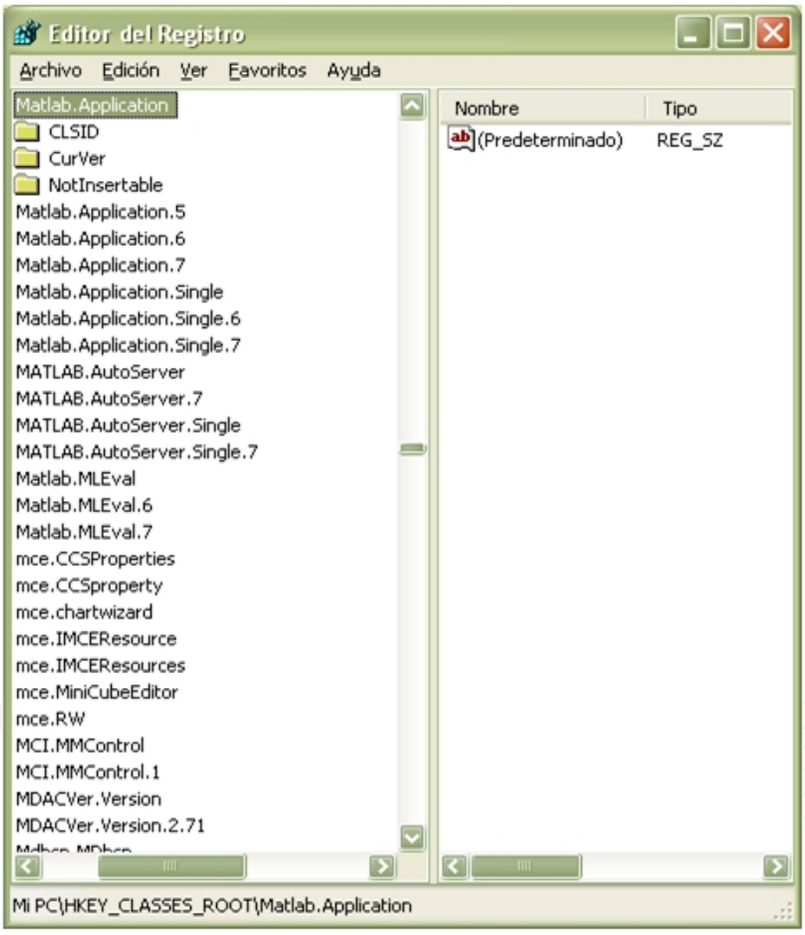


Figure 1. Matlab programmatic identifiers for different package versions.

This is very useful if the client must start an instance of a specific version of the software, because each version has its own programmatic identifier. The identifier has also a suffix with a major, followed by a minor, version to identify it (for example 7.3). If no identifier is used, the most recently version will be selected. This situation is not very common, but it can be used to compare the different behaviour of the different versions.

3. Matlab COM server

Matlab COM server is included in the default software installation. With COM enabled, developers can implement COM components and use other COM objects. Developers may insert Matlab applications on web (html code) pages or use Microsoft Excel spreadsheets to present numerical results.

Integration of COM Matlab objects can also be distributed, so components may be run from remote machines. This is mainly used for developing parallel solutions, which are typical on many engineering applications, specifically in digital signal processing.

The next sections will explain how Matlab COM server works. Furthermore, COM objects are illustrated with some real examples.

3.1. Matlab programmatic identifiers

As shown above in figure 1, Matlab provides three programmatic identifiers to access COM servers:

- **Matlab.Application.** This identifier is used to start a Matlab Automation server on an independent window. A command window will appear to enter commands. Note that if several versions exist, the most recently Automation Server version run will be selected.
- **Matlab.Autoserver.** A Matlab Automation server is started on an independent command window. The most recent install version is selected if no version is specified.
- **Matlab.Desktop.Application.** With this configuration, a Matlab full desktop is started. The most recently installed version will run, if no version is specified.

By running the registry tool in Windows, we will be able to find out the identifiers for accessing the COM servers (see figure 1):

- **Matlab.Application**
- **Matlab.Application.Single**
- **Matlab.AutoServer**
- **Matlab.AutoServer.Single**

The *single* attribute will make the server to run on an exclusive mode. Otherwise the server will starts on a shared configuration.

3.2. Matlab Client/Server architecture

Matlab client-server architecture defines how client and server relationships can be established. Four different models are available:

- Matlab client (In-Process Server). Using this architecture, Matlab clients access services by ActiveX controls or DLL libraries. Both client and server run on the same process. Communication is extremely efficient as both share the same context. If a DLL is used, it runs on a separate window.

- Matlab client (Out-Process Server). Client accesses server resources using an executable (.exe) file. The component instantiates on a separate process. Communication is not as effective as in the In-Process configuration.
- Matlab application and Matlab server (Automation Server). Using this configuration, a client application (called controller) accesses the services provided by the Automation server. The services make possible to run commands and transfer variables into Matlab workspace. The server may run locally or on a remote machine using DCOM services (see section 2.3). This configuration is affected from networks issues, specifically by bandwidth bottlenecks and latency problems.
- Client application and Matlab server (engine server). Matlab offers a faster interface called IEngine to be used with C, C++ and Fortran clients.

3.3. Linking references to COM servers

When using developing tools as Microsoft Visual Studio, some previous configuration must be made before writing the code. Specifically, COM references must be included into the project configuration. Figures 2 and 3 show how these references can be added:

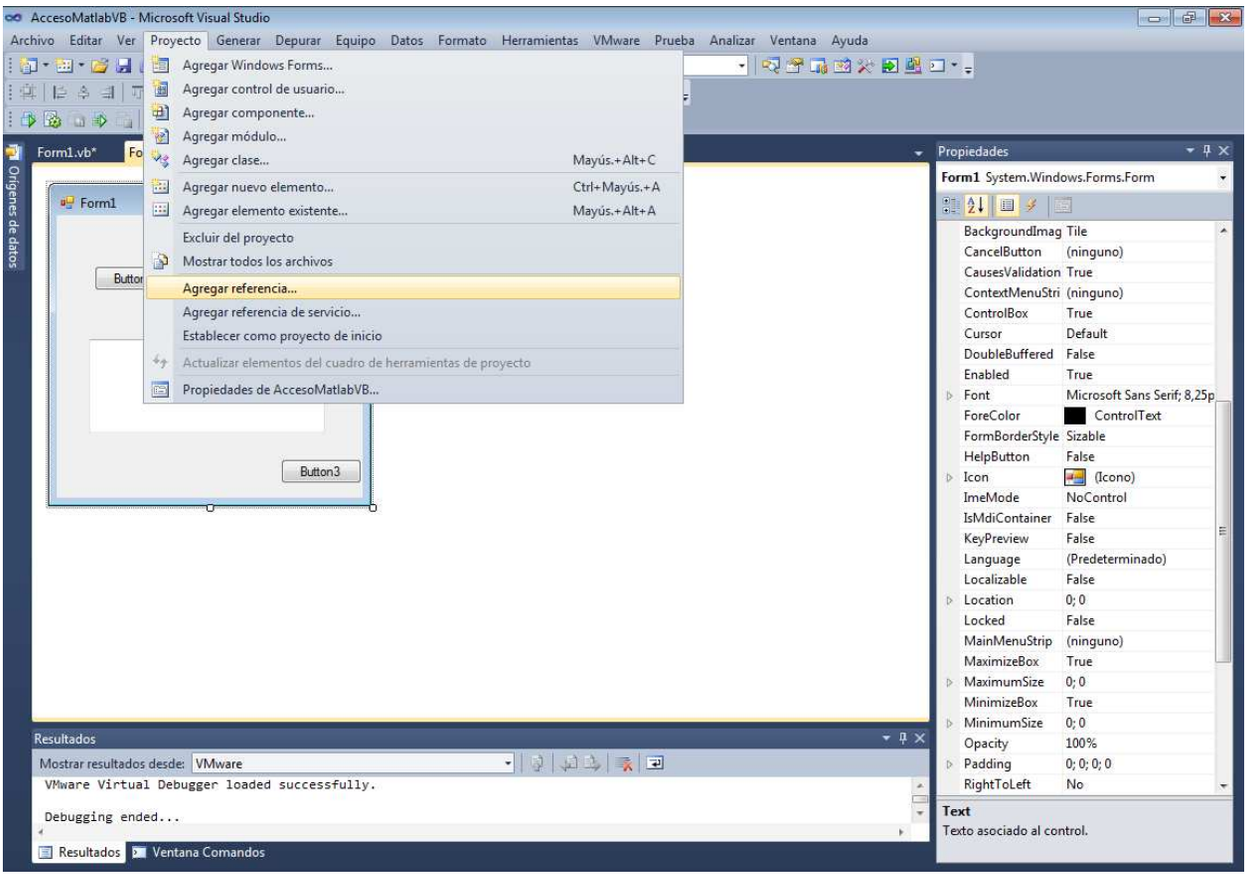


Figure 2. Visual Studio tool for adding external references.

By clicking on project tab and selecting "add reference", the references window appears, and shows the applications that provides COM servers (figure 3).

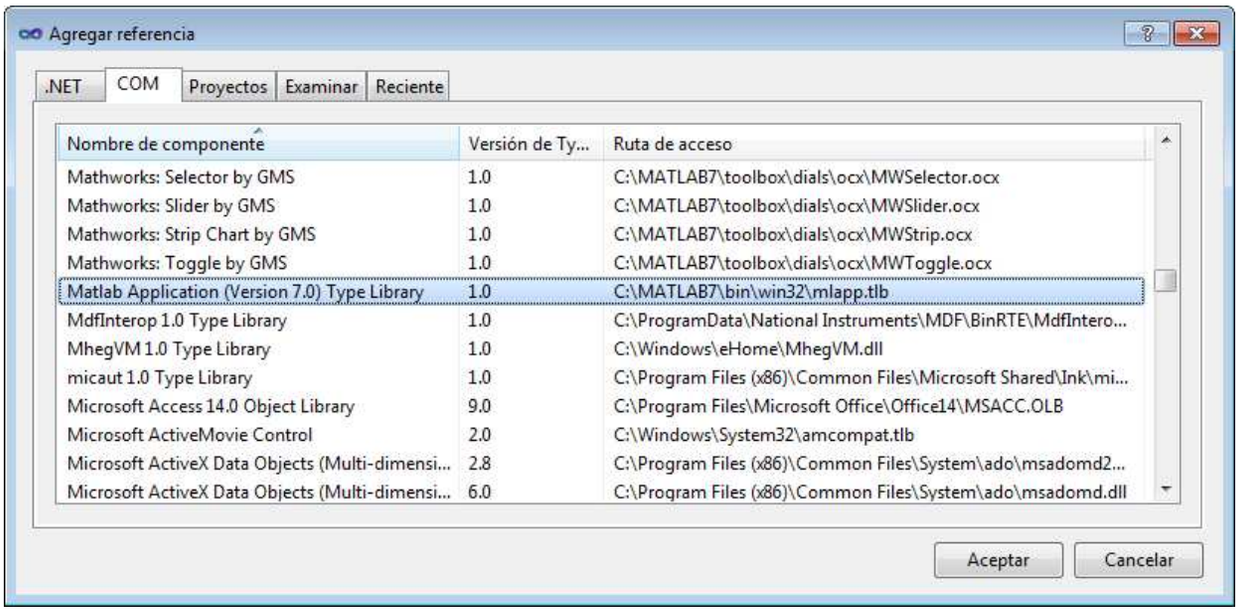


Figure 3. Adding Matlab COM references.

The reference selected is identified by a version number, and a path for the library location. If the server is located on a remote machine, then, the identifier will contain the path to the server having Matlab installed. The next figure shows a remote server accessible by the path "\\hiseuibd01\home$\matlab2\bin":



Figure 4. Matlab COM references for different package versions and locations.

References to other different hosts make it possible to access Matlab remote servers using DCOM. With this configuration Matlab software must be also installed on the local host, because some Matlab components must be accessible locally.

Once the references are added, Matlab objects links can be used in the source code. This is usually made by the "new" construct or "CreateObject" function. If the interface IMLApp is used, the methods available can be shown by the source code interactive help system. Otherwise methods can be viewed using the object explorer menu option (figure 5 and 6):

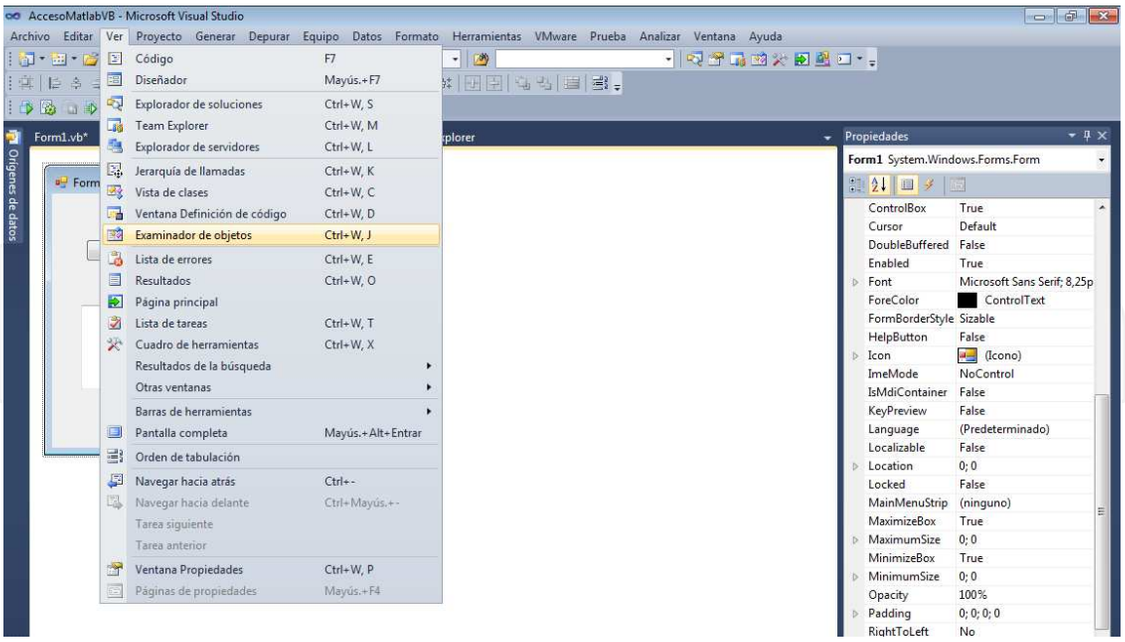


Figure 5. Visual Studio object explorer window.

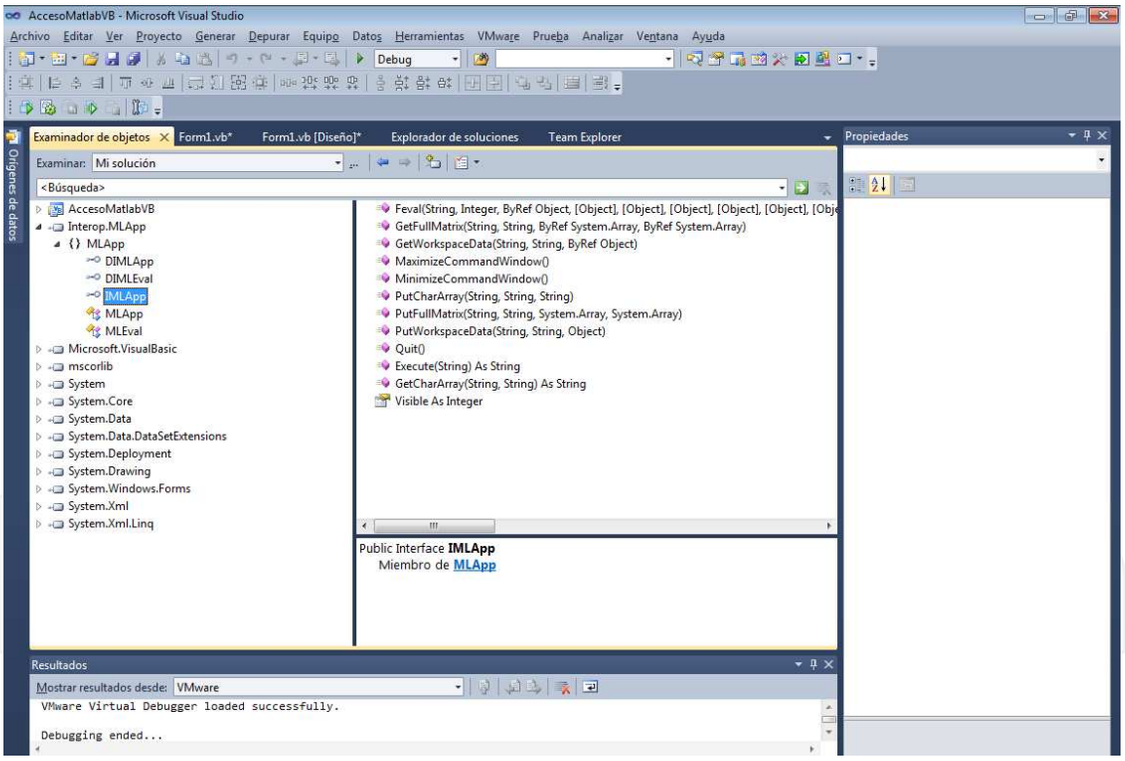


Figure 6. Matlab MLApp interface methods available.

Figure 6 shows the available interface methods. The most useful are:

- MLApp.Execute(). This method executes a Matlab command on the COM server.
- MLApp.Feval(). This method evaluates a Matlab function.
- MLApp.GetFullMatrix(). This method is useful to copy a matrix from Matlab workspace.
- MLApp.PutFullMatrix().This method is useful to copy a matrix into Matlab workspace.

- MLApp.PutWorkspaceData(). This method is used to put variables into Matlab workspace.
- MLApp.GetWorkspaceData(). This method is used to read variables from Matlab workspace.
- MLApp.Quit(). It must be used to force exit when the host application finalizes.

3.4. Accessing Matlab resources from the source code

The next sections will describe how to create and reference COM objects using different languages.

3.4.1. Creating COM objects

Microsoft .Net technologies provide resources to access COM objects from different languages. For convenience, the following examples are written in Visual Basic. Users that prefer other languages may use COM objects and references on the same way.

The following example makes use of the easy graphical user interface that provides Visual Basic. Matlab specific routines are accessed by adding references into the Visual Basic code. The next code is an example of running a Matlab server that exchanges data from the workspace. The application simply runs an interactive tool to access some Matlab resources using COM services.

```
Public Class Form1
    'Dim Matlab as Object and use CreateObject method
    'use MLApp library and New MLApp method to see the methods available
    Dim MatlabCommand As String
    'Dim Matlab As MLApp.MLApp
    Dim Matlab As Object
    Dim ArrayA() As Double = {1, 2, 3, 4}
    Dim ArrayB() As Double = {5, 6, 7, 8}
    Dim ArrayC() As Double = {0, 0, 0, 0}
    Dim ArrayImg() As Double = {0, 0, 0, 0}
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Matlab = CreateObject("Matlab.Application")
End Sub
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
        Matlab.Quit()
        Me.Close()
End Sub
```

Table 1. GetFullMatrix() and PutFullMatrix() methods example.

Note that before using COM objects, some declarations must be made in order to instantiate the objects.

Figure 7 shows how the “execute()” method can be used for running commands inside the Matlab environment. Methods “GetFullMatrix()” and “PutFullMatrix()” are used in the example to exchange data between the application and the Matlab workspace.

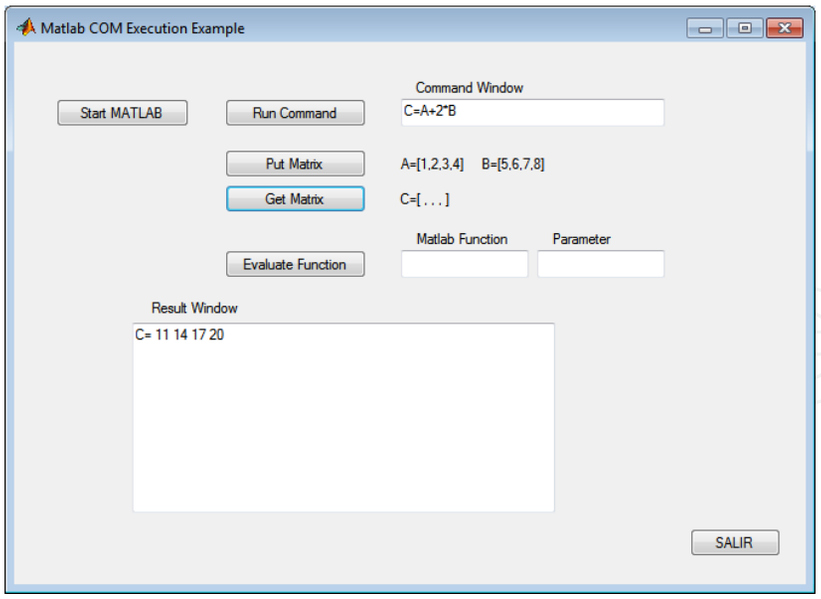


Figure 7. Matlab Execute(), PutFullMatrix() and GetFullMatrix() interface methods example.

Matlab Execute() method provides a powerful tool to use Matlab server. It is probably the most frequent method used. The source code associated with the task shown in figure 7 is the following:

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        Dim StringResult As String
        MatlabCommand = CommandBox.Text
        StringResult = Matlab.Execute(MatlabCommand)
        ResultBox.Text = StringResult
End Sub

Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
        Dim ArrayA() As Double = {1, 2, 3, 4}
        Dim ArrayB() As Double = {5, 6, 7, 8}
        Dim ArrayC() As Double = {0, 0, 0, 0}
        Dim ArrayImg() As Double = {0, 0, 0, 0}

        Matlab.PutFullMatrix("A", "base", ArrayA, ArrayImg)
        Matlab.PutFullMatrix("B", "base", ArrayB, ArrayImg)
End Sub

Private Sub Button5_Click(sender As System.Object, e As
System.EventArgs) Handles Button5.Click
        Dim ArrayC() As Double = {0, 0, 0, 0}
        Dim ArrayImg() As Double = {0, 0, 0, 0}

        Matlab.GetFullMatrix("C", "base", ArrayC, ArrayImg)
        ResultBox.Text = "C= " + ArrayC(0).ToString() + " " +
ArrayC(1).ToString() + " " + ArrayC(2).ToString() + " " +
ArrayC(3).ToString()
End Sub
```

Table 2. Matlab Execute(), GetFullMatrix() and PutFullMatrix() methods example.

Figure 8 shows how the Feval() method can be used to evaluate a trigonometric function inside Matlab environment. This method allows parameter exchange, so you can pass or receive the necessary arguments (sin(π) in the example).

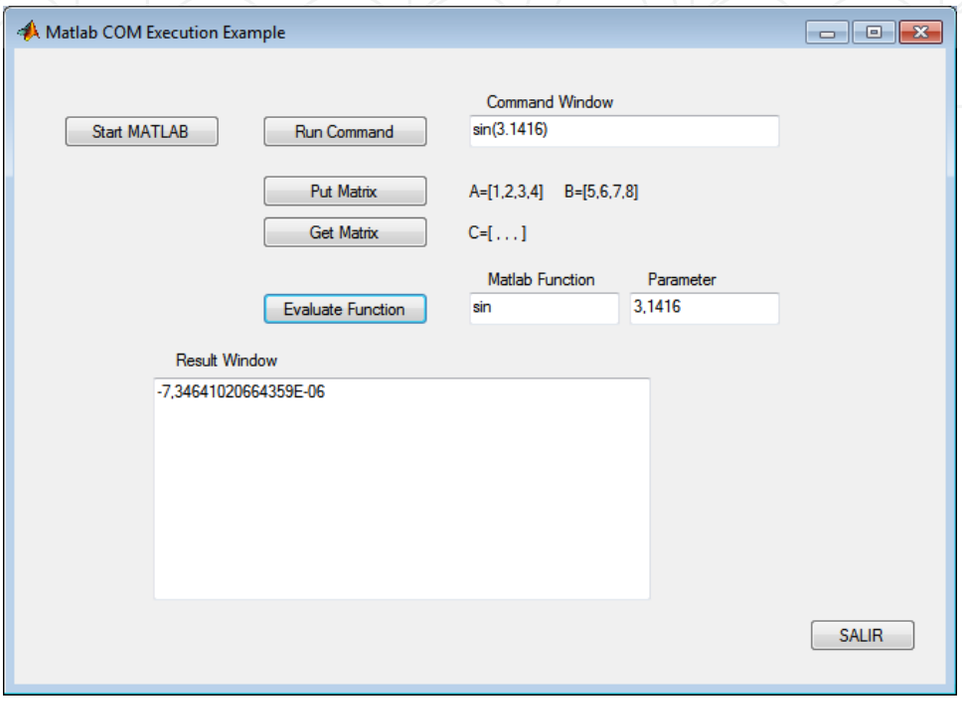


Figure 8. Feval() interface method.

The source code fragment for the application that it is shown in figure 8 is the following:

```
Private Sub Button6_Click(sender As System.Object, e As
System.EventArgs) Handles Button6.Click
        Dim MatlabFunction As String
        Dim out As Object
        Dim x As Double

        x = Parameter.Text
        MatlabFunction = FunctionBox.Text
        Matlab.Feval(MatlabFunction, 1, out, x)
        ResultBox.Text = out(0).ToString
End Sub
```

Table 3. Feval() method example.

Finally, the next code fragment shows an example for transferring data between the application and Matlab workspace, using "PutWorkspaceData()" and "GetWorkspaceData()" interface methods:

```
Private Sub Button7_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Button7.Click
        Dim nombre As String
        Dim matriz(,) As Integer = New Integer(,) {{0, 1}, {2, 3}}
        Dim S2X2(,) As Short = New Short(,) {{5, 6}, {7, 8}}

        nombre = TextBox5.Text
        Matlab.PutWorkspaceData(nombre, "base", matriz)
End Sub

Private Sub Button6_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Button6.Click
        Dim resultado As Object
        Dim nombre As String

        nombre = TextBox5.Text
        Matlab.GetWorkspaceData(nombre, "base", resultado)
        TextBox7.Text = resultado
End Sub
```

Table 4. Example using PutWorkspaceData() and GetWorkspaceData() interface methods.

Both examples transfer variables using the default "base" workspace and custom variable names read from a text box.

3.4.2. Creating COM objects using Tcl/Tk

Tcl (Tool Command Language) is a very powerful and easy language to learn. It is useful for automation, test, programming embedded systems, web applications and database access (Tcl community, 2009).

It was developed in 1988 by John K. Ousterhout at the University of California, and later maintained by Sun Microsystems Laboratories by the group SunScript. One of its great advantages is that it is multiplatform. There are versions for Windows, Mac OS X and most versions of Unix: Linux, Solaris, IRIX, AIX, BSD...

The language is constantly evolving, so new versions of interpreters are frequently released, along with extensions that extend the functionality of the language.

Tcl is flexible and open source, so any developer can investigate the language details, include new features, modify existing ones, and even develop new commercial versions. Extensions are usually of the same type of license, although it depends on each developer.

Associated with Tcl, there is a toolkit for developing graphical user interface called Tk (Tool Kit), which is the most popular Tcl extension. Tk was also developed by John Ousterhout and provides an interpreter that adds Tcl commands and others capable of creating graphical user interface components such as buttons, panels, combo boxes and dialog boxes. It is usually distributed into a package called Tcl/Tk.

Recent versions of Tcl/Tk include a package to access COM objects through a very extensive API, replacing the traditional interpreter by a compiler that translates source code to bytecodes, which then runs another interpreter (Huang, 2006). Although this improvement

allows substantial increase of the execution speed, it cannot be still compared with other compiled languages.

The following code shows how Matlab COM server can be created using a Tcl/Tk script:

```
package require tcom
set application [::tcom::ref createobject  "Matlab.Application.7"]
```

Table 5. Example creating a COM reference from Tcl.

Once the reference is created, all methods will be accessible. The next lines show how to run a Matlab .m program and get the result (Matlab diagram variable) from the "base" workspace, into the local variable "diagrama".

```
$application Execute "run('C:/smi.m')"
$application GetWorkspaceData "diagram" "base" "diagrama"
```

Table 6. Example using Execute and GetWorkspaceData interface methods for Tcl.

4. XBDK, a TCL/TK platform using Matlab and other COM objects

XBDK stands for XML-Based Beamforming Development Kit. It is a software platform designed for aiding the engineer through the beamforming development process (Raboso et al., 2003, 2007, 2009). It is a CASE (Computer Aided Software Environment) tool that integrates several interesting applications related to software modelling, simulation and XML parsing and manipulation. The applications are written in Java and Matlab, and are integrated using a script language (Tcl/Tk).

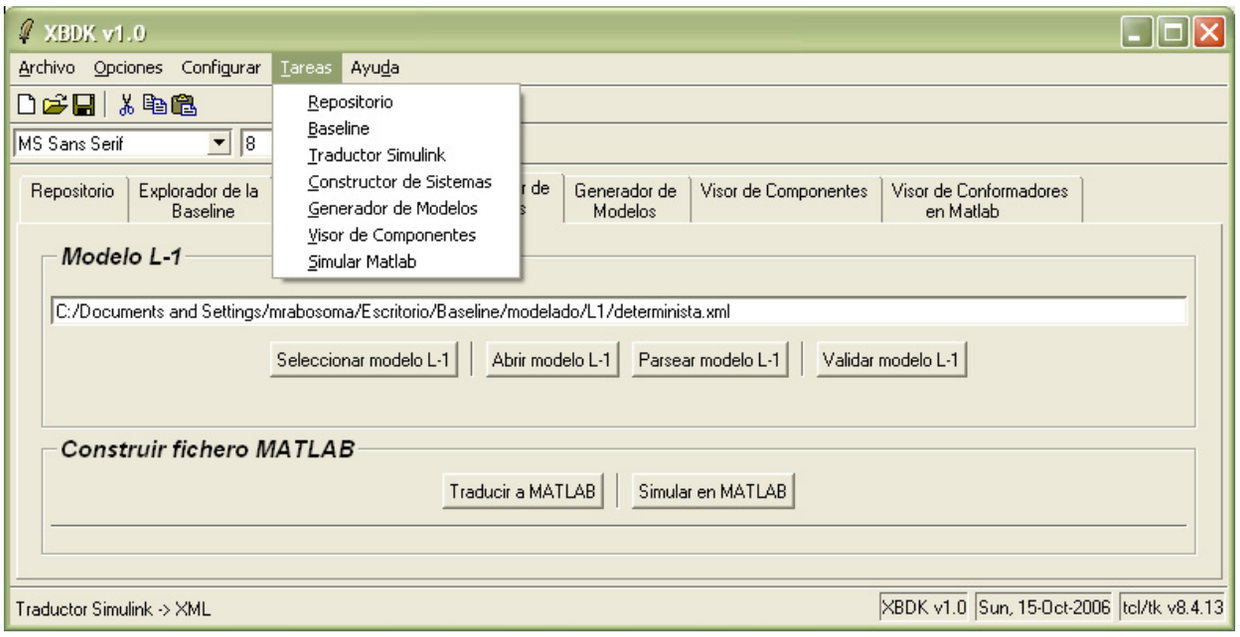


Figure 9. XBDK main menu.

Three COM servers are used for implementing different tasks:

- Matlab COM server is used for running Matlab programs for simulation tasks.
- Altova XML COM server is used for XML parsing and validation.
- Microsoft Excel COM server is used for data management.

XBDK uses a data model that gives a high-level abstraction management of a variety of signal processing systems, specifically for Beamformers (Raboso et al. 2003). Beamformers are array signal processing systems that process the receiving or transmitting signals of an array of sensors (antennas) for obtaining interesting radiation properties as high directivity, low sidelobe levels or nulls for some directions of arrivals. Beamformers developed with XBDK are described using XML language, providing a natural human description of such systems. The abstraction model architecture is shown in figure 10. The specific tools to manage the information of each level are located at the right side of each layer.

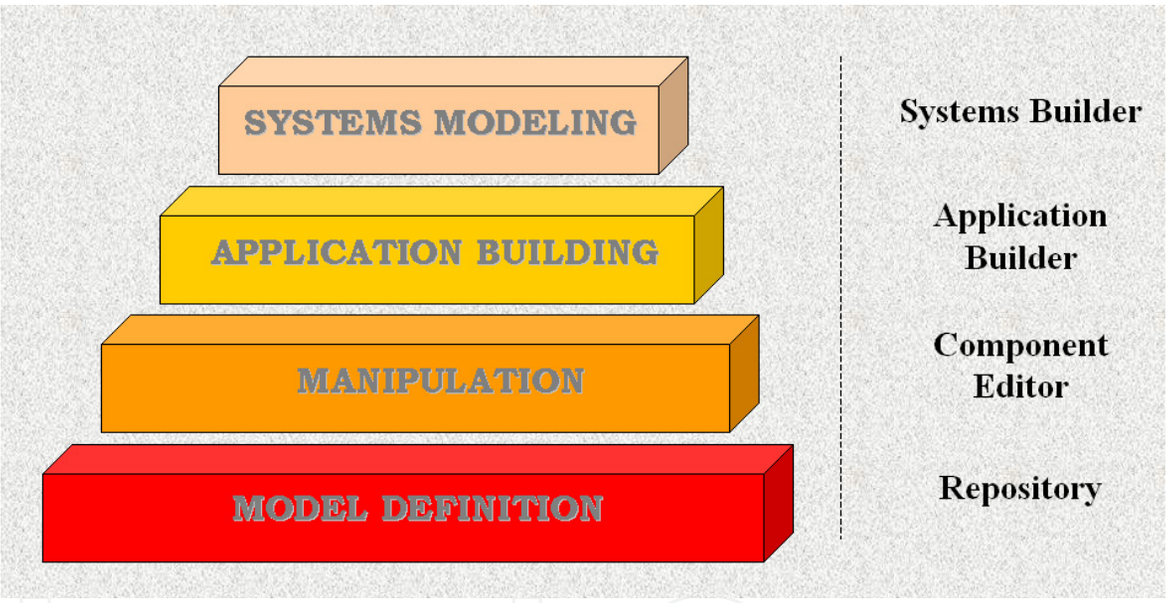


Figure 10. Beamforming XML data representation.

Beamformers are designed and later written in XML using two abstraction levels corresponding to the upper two levels of the pyramid. For XML data manipulation, files must be well-formed and validated. As XBDK is also responsible of these tasks, this tool integrates several routines that use another COM server designed by Altova (AltovaXML).

The next figure and source code below shows a .Net example performing validation and well-forming tasks. Note that the Altova COM server must be started before the methods are called. As a result, a window with the server is automatically launched. This window can be push into background until the application is closed. As the quit() method is not implemented in the interface, close operation must be done manually with CTRL+C key combination or using the task manager.

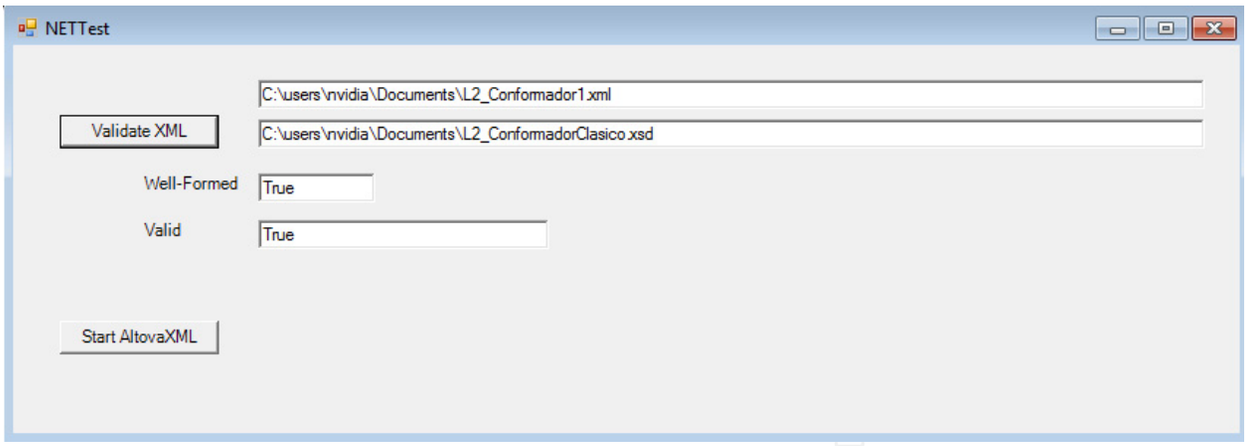


Figure 11. Beamforming XML data representation.

The following code corresponds to the example below:

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim resultado As String

        AltovaXMLNET.XMLValidator.InputXMLFileName = TextBox1.Text
        AltovaXMLNET.XMLValidator.SchemaFileName = TextBox4.Text
        resultado = AltovaXMLNET.XMLValidator.IsWellFormed()
        TextBox2.Text = resultado
        If (resultado) Then
            'resultado = AltovaXMLNET.XMLValidator.IsValid()
            resultado                                              =
AltovaXMLNET.XMLValidator.IsValidWithExternalSchemaOrDTD()
            TextBox3.Text = resultado
        Else
            TextBox3.Text = "The document is not well-formed"
        End If
End Sub

Private Sub Button2_Click(sender As System.Object, e As
System.EventArgs) Handles Button2.Click
        AltovaXMLNET = New Altova.AltovaXML.Application
End Sub
```

Table 7. Example using AltovaXML COM server.

5. Conclusions

COM technology is a useful tool for integrating software from different vendors. This is specifically interesting for engineering applications, which have to integrate a great variety of software functionality, from specialized and low-level task, to intuitive GUIs.

Fortunately, software component industry has provided solutions to properly combine software components to get a complete solution without implementing the software from scratch. This can be made using reusability techniques following the standards defined by the Component Base Software Engineering (CBSE).

Engineers working with Matlab and other software can take advantage of CBSE using COM and .Net technologies from Microsoft. Furthermore, integrating different objects from different applications accelerates software development and reduces costs. Today, COTS (Commercial Off-The-Shelf) components are ready to be integrated into the engineering applications and vendors exploit these advantages developing components while assuring quality and standard conforming.

After reading this chapter, I expect that the interested reader can take into account component technology on their future projects, and gain effectiveness on the overall software development process.

Author details

Mariano Raboso and Myriam Codes
Universidad Pontificia de Salamanca (Facultad de Informática), Spain

María I. Jiménez, Lara del Val, Alberto Izquierdo and Juan J. Villacorta
Universidad de Valladolid, Spain

Acknowledgement

This research has been supported by projects: 10MLA-IN-S08EI-1 (Pontifical University of Salamanca), and PON323B11-2 (Junta de Castilla y León). I also want to thank the work made by professors Maribel and Domingo, who carefully made the technical review of the chapter.

6. References

Gunderloy, M. (2001). Calling COM Components from .NET clients, In: *MSDN Library*, 23.03.2012, Available from: http://msdn2.microsoft.com/en-us/library/ms973800.aspx

Huang, C. (2006). Tcom, In: *Access and implement Windows COM objects with Tcl*, 23.03.2012, Available from: http://www.vex.net/~cthuang/tcom/

Raboso, M.; Izquierdo, A. & Villacorta J.J. (2003). Beamforming Systems Modeling using Component Reusability with XML Language, Proceedings of the International Signal Processing Conference, ISPC 2003, Dallas, Texas, USA, March 31-April 3, 2003.

Raboso, M. (2007). *Beamforming Systems Modeling Using XML Language, based on Software Component Reuse,* ProQuest Information and Learning, ISBN 978-0-549-26134-6, USA

Raboso M. ; Izquierdo A. ; Villacorta J. ; del Val L. & Jiménez, M. (2009). Integración de componentes COM de MATLAB/SIMULINK en el entorno CASE XBDK para el modelado de sistemas de conformación de haz. *Ingeniare, Revista chilena de ingeniería*, Vol.17, No.1, (January 2009), pp. 122-135, ISSN 0718-3305

Tcl community (2009). Tcl and Tk manual pages, In: *Tcl/Tk Documentation*, 23.03.2012, Available from: http://www.tcl.tk/doc/

The Mathworks (2012). MATLAB COM Automation Server Support, In: *Matlab R2012a Documentation*, 23.03.2012, Available from:
http://www.mathworks.es/help/techdoc/matlab_external/brd0v3w.html