How to speed up your co-simulation using OpenDSS COM interface

Authors: Davis Montenegro, Roger Dugan
October 27, 2015

Many times users have mentioned performance issues when co-simulating using OpenDSS. The interface proposed in OpenDSS to perform co-simulations is the COM interface, which can be accessed from almost every programming language; however, not necessarily in the faster way.

Because of this, many users blame the COM interface for their performance issues. The aim of this document is to clarify why users could experience a low performance in terms of co-simulation speed and to give a guide about how to improve it.

Early Binding and Late Binding

Early and Late bindings are two computer programming mechanisms for accessing COM servers/ActiveX controls considering the features of the interface objects and classes. Late binding is focused on giving flexibility in case the objects/classes contained within the interface are polymorphic. For doing this, the late binding mechanism uses a Virtual table (vtable) that includes the memory address of each allocated class and its features. Every time the external program accesses an object/class using the COM interface it must go to the vtable first (if something changes it will be updated into the vtable), then look for the object to obtain the memory address and its features, which adds significant overhead on each iteration.

In contrast, early binding considers that every object is static in time and will be the same during the connection with the external software. For doing this, the vtable is eliminated and the index to each object/class is made by specifying the DispID memory address directly during the compilation phase. This eliminates the overhead generated by accessing vtables and the access to the COM interface objects is drastically faster [1].

To get connected to the COM interface using early binding, first check to see if your programming language supports early binding connection. Normally, all programming languages support late bindings but not necessarily early binding. However, in the next sections, this article presents an alternative for programming languages that do not support early binding.

If your programming software language supports early binding the activation of this connection mechanism consists of adding a couple of code lines when establishing the connection with the COM server. To evaluate the performance of the early binding vs. late

binding connections to OpenDSS, Roger Dugan proposes the following algorithm with the IEEE 8500-node Test Feeder to measure the computation time:

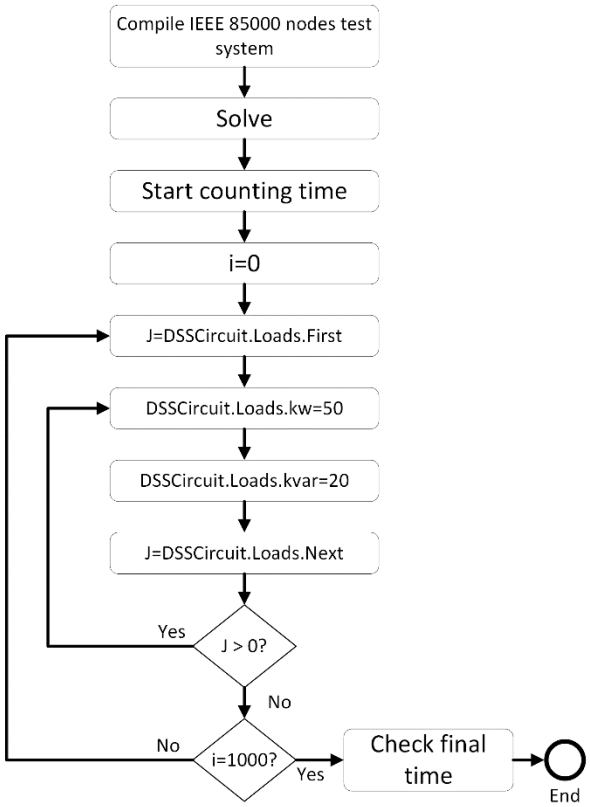

Figure 1. Proposed algorithm to evaluate the co-simulation performance

The algorithm was coded into a compiled Delphi program that achieves an early binding connection to the COM interface. The algorithm shown in Figure 1 takes 1056 ms (average) in a standard computer with an Intel core i7-4510U processor (up to 3.1 GHz) and windows 10 Operating system. This value is going to be the reference to evaluate the performance when using early bindings and late bindings in different programming languages during this document.

The Delphi Test Routine

The actual Delphi code for the loop in Figure 1 is:

```
procedure TForm1.Button2Click(Sender: TObject);
Var
     i:Integer;
     iLine : Integer;
     DSSLoads : ILoads;

begin
      SW := TStopWatch.Create();
      DSSLoads := DSSCircuit.Loads ;
      SW.Start;
      With DSSLoads  Do
        for i := 1 to 1000 do  Begin
          iLine := First;
          while iline > 0 do  Begin
               kW := 50.0;
               kvar := 20.0;
               iLine := Next;
          End;
      End;

      SW.Stop ;
      Edit1.Text := Format('%d ms for %d Loads 1000 times',
                           [SW.ElapsedMilliseconds, DSSLoads.count]);
end;
```

The early-binding connection to the OpenDSS COM interface is made using the following code to make a connection to the IDSS interface. Then some local variables are used to achieve a little better performance by maintaining a static connection to other interfaces that are created by the call to “coDSS.Create”.

Definition of Variables

```
Var
    DSSObject: IDSS;  // DSS Interface
    DSSText:   IText;
    DSSCircuit: ICircuit;
    DSSSolution:ISolution;
    DSSProgressfrm:IDSSProgress;
```

Connection to the Interface

```
        TRY

             DSSObject := coDSS.Create; // Creates early binding
             DSSObject.Start(0);
         EXCEPT
             On E:Exception Do Begin
               MessageDlg('Did not Start.',  mtConfirmation, [mbYes, mbNo],
0, mbYes);
               Exit;
             End;

         END;
```

VBA Early and Late Binding

VBA (Visual Basic) is a programming language that supports both early binding in a simple way as well as the traditional initialization of the COM interface using late binding.

Late Binding

Public DSSObj as Object
….
Set DSSObj = CreateObject("OpenDSSEngine.DSS")

Early Binding

Public DSSObj as OpenDSSEngine.DSS
…
Set DSSObj = New OpenDSSEngine.DSS

The measured computing times using each technique are shown in Figure 2.

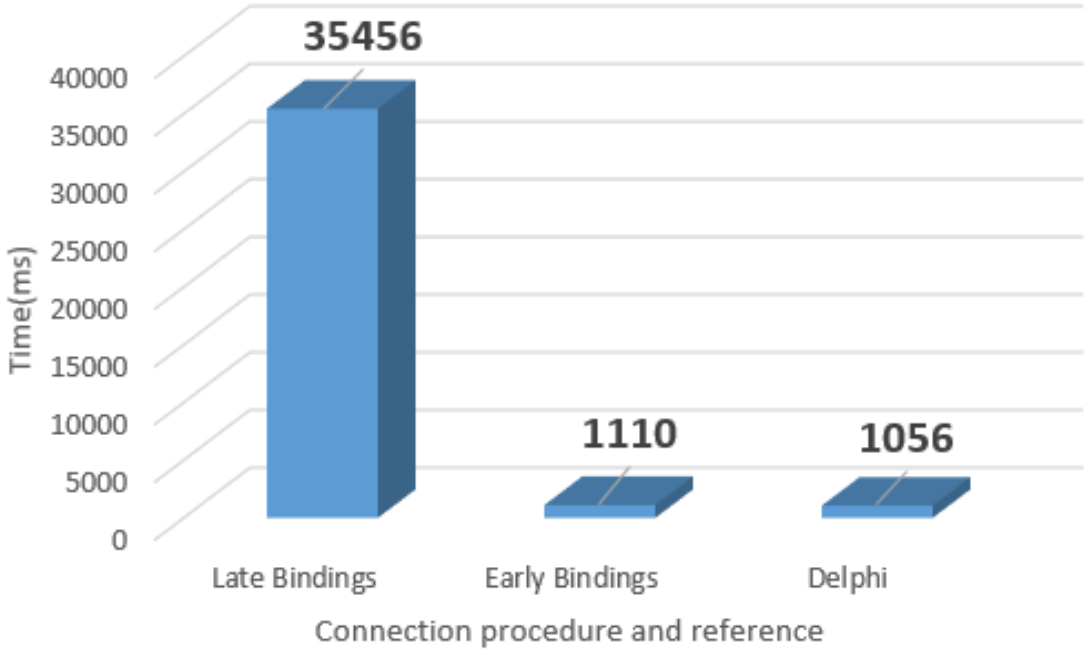

Figure 2. Computing times in *ms* for the algorithm in Figure 1 using VBA late binding and early binding in comparison with the reference (Delphi)

Early Binding in Python

In contrast to VBA and other languages that have early binding built into the compiler, python includes the m*akepy* procedure to generate a static table and emulate early binding connection.

The connection mechanism for late binding is as follows (provided by Adam Birchfield):

```
import win32com.client
dssObj = win32com.client.Dispatch("OpenDSSEngine.DSS")
```

To use early binding the declaration code is amended as follows:

```
import win32com.client
from win32com.client import makepy
import sys
sys.argv = ["makepy", "OpenDSSEngine.DSS"]
makepy.main()
dssObj = win32com.client.Dispatch("OpenDSSEngine.DSS")
```

The measured computing times for each technique are shown in Figure 3. The *makepy* approach makes a significant improvement and is nearly equal to the VBA result.

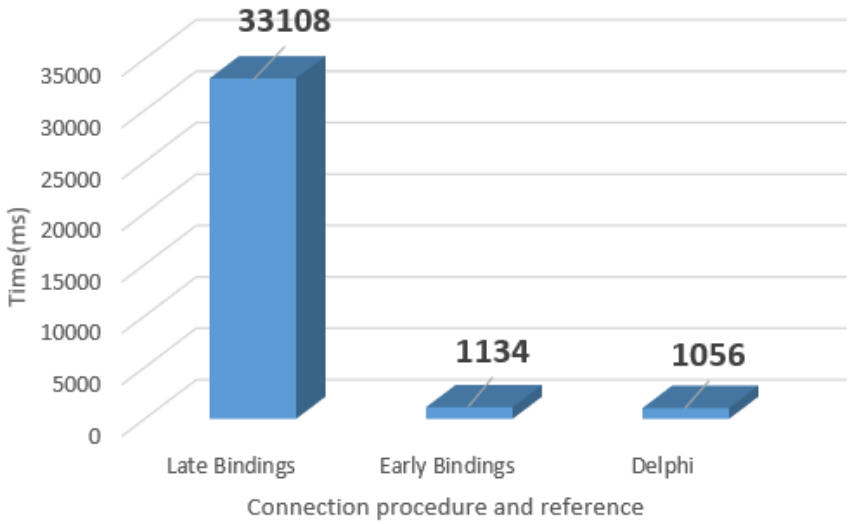

Figure 3. Computing times in *ms* for algorithm in Figure 1 using Python late binding and early binding in comparison with the reference (Delphi)

NI LabVIEW (Direct-connection Shared Library)

As mentioned above, not all programming languages support early binding for connecting to COM servers/ActiveX Controls. However, to cover this issue we have developed the Direct-connection Shared Library (DCSL), which is a standard DLL with *stdcall* functions that reproduces the same objects and functionality as the COM interface. This is not as flexible as the COM interface but it offers the same performance as achieved by early binding to the COM interface. The speed improvement achieved with LabVIEW is quite significant as shown in Figure 4. The Shared Library is connected to the code using the "Call Library Function Node".

http://forums.ni.com/t5/LabVIEW/Automation-open-and-early-bindings/td-p/3201691

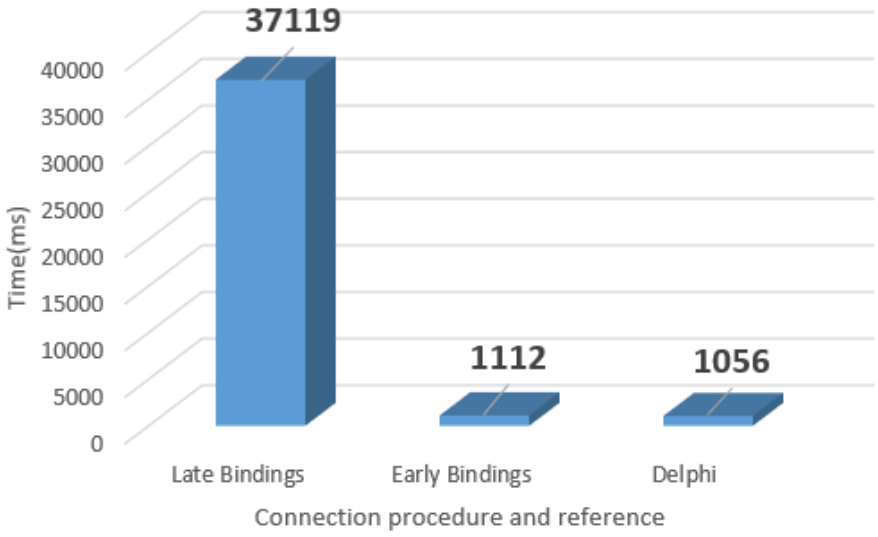

Figure 4. Computing times for algorithm in Figure 1 using NI LabVIEW late bindings and early binding via the Direct Connection Shared Library in comparison with the reference (Delphi)

MATLAB (using Direct connection Shared Library)

In the case of MATLAB it is also necessary to use the DCSL DLL proposed as the alternative to accelerate the co-simulation speed. However, MATLAB requires the *.h* file of the DLL to be connected. The DCSL is not provided with the DLL header file but this is not a problem; you can create a prototype file (*.m*) to declare the functions you need, which are described in the DLL's user manual. To see the prototype file format check the file *DEngineProto.m* provided with the example about how to connect MATLAB to the DCSL.

http://www.mathworks.com/matlabcentral/answers/95331-why-are-none-of-the-properties-or-methods-available-from-my-activex-server-com-object

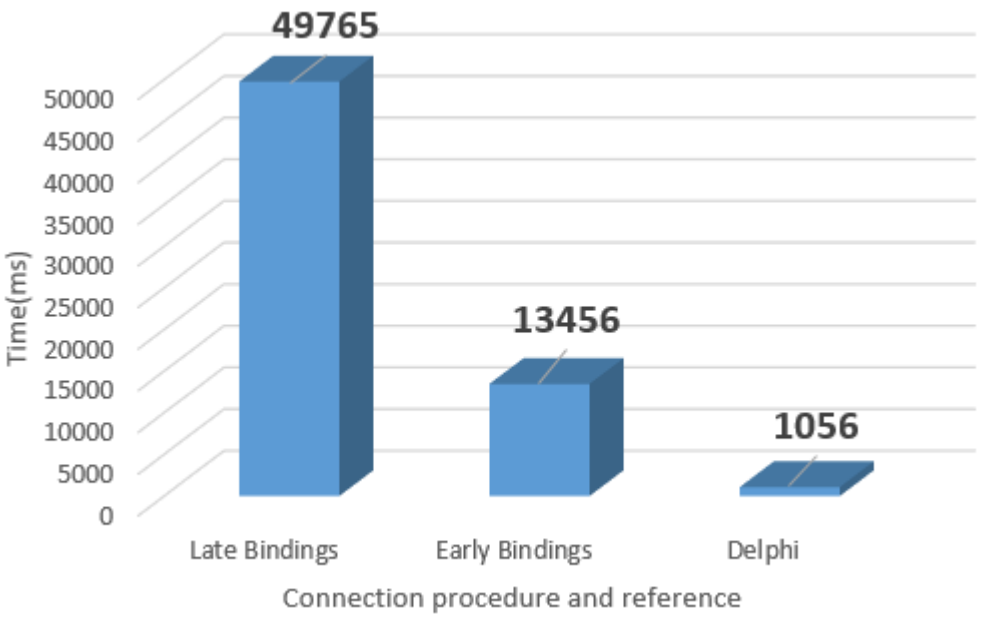

Figure 5. Computing times in *ms* for algorithm in Figure 1 using MATLAB Late binding and early binding via the Direct Connection Shared Library in comparison with the reference (Delphi)

Even if the improvement achieved in MATLAB is not as good as in the other programming languages, there is an important gain in time that is certainly worth investigating.

Conclusions

To accelerate the co-simulation speed between OpenDSS and external programming languages take into account the following features of your programming environment:

- First check if your programming environment supports early binding. Early binding is a connection mechanism that statically connects to objects in the COM interface at compile time, which allows faster access to objects within a COM interface.
- The default mechanism in all programming languages for accessing COM objects is late binding.

- Not all programming languages support early-binding connections. In this case, use the Direct Connection Shared Library.

References

[1] D. Rogerson, *Inside COM*: Microsoft Press, 1997. (https://support.microsoft.com/en-us/kb/245115)