# Developing CORBA-based distributed control and building performance environments by run-time coupling

A. Yahiaoui, Center for Building & Systems TNO-TU/e, P.O. Box 513, 5600 MB Eindhoven, Netherlands
J.L.M. Hensen, Center for Building & Systems TNO-TU/e, P.O. Box 513, 5600 MB Eindhoven, Netherlands
L.L. Soethout, TNO Building and Construction Research, P.O. Box 49, 2600 AA DELFT, Netherlands
(A.Yahiaoui@tue.bwk.nl; J.L.M.Hensen@tue.bwk.nl; L.Soethout@bouw.tno.nl)

## Summary

Communication software and distributed applications for control and building performance simulation software must be reliable, efficient, flexible, and reusable. This paper reports on progress of a project, which aims to achieve better integrated building and systems control modeling in building performance simulation by run-time coupling of distributed computer programs. These requirements motivate the use of the Common Object Request Broker Architecture (CORBA), which offers sufficient advantage than communication within simple abstraction. However, set up highly available applications with CORBA is hard. Neither control modeling software nor building performance environments have simple interface with CORBA objects. Therefore, this paper describes an architectural solution to distributed control and building performance software tools with CORBA objects. Then, it explains how much the developement of CORBA based distributed building control simulation applications is difficult. The paper finishes by giving some recommendations.

## 1 Introduction

With the increasing importance of integrated building simulation environments into the concurrent building design, the demand on model complexity and interdisciplinary led to a growing number of difficulties between the various environments. A particular alternative may reduce some complexities by integrating an existing tool (codes) as a part of the building simulation environment. Although, such integration consists in developing a framework that is in charge of calling the different simulation codes in a specific order and to let them exchange their simulation results. An object-oriented approach is well suited for the development of a framework requiring that each of those simulation codes to be an object. Indeed, the use of distributed objects allows a transparent communication.   However, distributed object technologies have to be adapted in the context of high-performance computing by tacking into account that a distributed objects can be used to implement parallel-programming paradigms (e.g. two running applications share or exchange the same data).

In this paper, we relate our experience that was aimed at run-time coupling of control and building performance simulation software to perform a distributed simulation. The run-time coupling is base on the use of CORBA (Common Object Request Broker Architecture) as communication architecture.

The current situation is that there exists domain independent software that is very advanced in control modeling and simulation features, but doesn't have any building performance simulation concepts (e.g. Matlab / Simulink). On the other hand, domain specific software for building performance simulation (ESP-r, TRNSYS, etc) is usually relatively basic in terms of control modeling and simulation capabilities. Marrying the two approaches by run-time coupling building simulation environments and control modeling software would potentially enable integrated performance assessment by predicting the overall effect of innovative control strategies for integrated building systems.

Our approach is built around the notion of run-time coupling between building simulation environments and control modelling software that would potentially enable integrated performance assessment by predicting the overall effect of innovative control strategies for integrated building systems. The consistency specification would depend on the requirements of the physical process being simulated and the accuracy desired. The important requirement would be an exchange data when one of the processes is called at every another process time-step.

In this paper, we have related our experience that aimed in run-time coupling control software with building performance environments using CORBA as communication architecture. The coupling is based on a schematic approach to interface distributed simulation with CORBA objects.

This paper is organized as follow: section 2 describes run-time approach. Section 3 gives a short overview of CORBA. In section 4, we sketch the architectural solution of IDL to ESP-r and Matlab compiler. Section 5 illustrates an example of the encapsulation FORTRAN code to CORBA object. Section 6 discusses some recommendations. Finally, we sum up some remarks in section 6.

## 2 Run-time coupling Approach

The coupling of tools requires an underlying communication mechanism to allow the transfer of both data and control between codes. The data transfer correspond to the sending of the results calculated by one of the process and receiving by another process whereas the transfer of control is the execution of one particular function to be performed remotely in another process. Most of attempts to couple codes together rely on the use of shared file system, which is not efficient to exchange data at an adequate level of abstraction (Hughes and Hughes 2003, Ranganathan et al. 1996).

While, the aim of the current work is to establish better control modelling in building performance simulation by integrating distributed computer programs. Indeed, communication can be viewed at different levels of abstraction. At higher level, it appears as an exchange of information between processes. These processes are either contained in a single parallel or distributed application, or may otherwise belong to different application that needs to communicate. At low level, communication appears as the mere traffic of bits from one address space to another. Distributed applications require support for expressing communication at a high level of abstraction. In (Yahiaoui et al. 2003), various options for inter-application data transfer facilities are described and compared. Moreover, it might have noted that a return value between processes requires the use of IPC (Inter-Process Communication) techniques.

Since we envisage that the different software modules may run simultaneous on a heterogeneous network (e.g. Unix, Windows) and we, potentially, also would like to be able to run-time couple with e.g. a building energy management system, a test-rig for a controller (hard-ware testing in the loop), or even a building emulator, the inter-process communication needs to be platform independent.

In our case, the run-time coupling should support asynchronous events that occur when the time step of one process is different from another process; and synchronous events that occur when the two processes are synchronizing with the same defined time in execution. The exchange can be either unidirectional (data send in one-way) or bi-directional (the data alter from the first process to the second and from the second require passing back to the first, using client/server technique). The data structure can be transferred in the form of data files or data types, having a common format so that processes running in two different operating systems can translate them as well as the underlying communication protocols.

We have advocated an approach, in (Yahiaoui et al. 2003), like others (Keahey and Gannon 1998), that consists in implementing communication paradigm in coherent way. This approach is based on the use of CORBA-based integration framework for run-time coupling building simulation environment with control modeling software, as shown in figure 1.

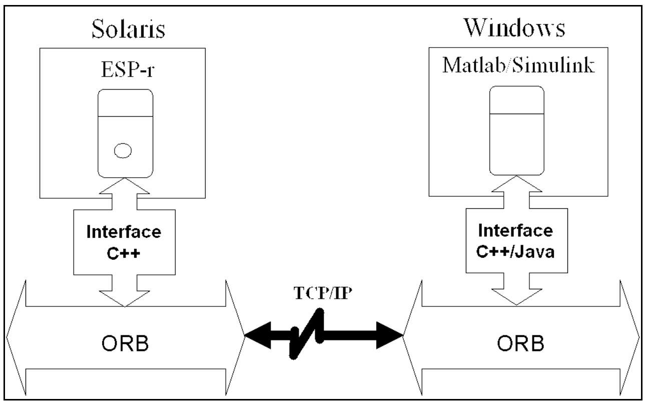

Figure 1 Implementation of distributed control and building Performance simulation software through ORB.

## 3 Overview of CORBA

CORBA is the acronym for the Common Object Request Broker Architecture, which defined by the Object Management Group (OMG) and specifies how to define interfaces, and how to pass requests between remote systems (Omg 2003). CORBA is an infrastructure for communicating distributed objects and heterogeneous systems (e.g. Windows and Unix). The CORBA was designed to allow intelligent software components to discover each other and interoperate on an object bus called Object Request Broker (ORB). The ORB is a server application that functions like a network switch. Any client object can make a request to a local or remote server object through an ORB. The server also responds through it. Being language-independent, CORBA introduces an Interface Definition Language (IDL) to provide an interface inheritance so that services can be extended easily.

The CORBA specification has several components: ORB Core, Interface Definition Language (IDL), Dynamic Invocation/Skeleton Interface (DII/DSI), Internet Inter ORB Protocol (IIOR), General Inter-ORB Protocol (GIOP), Object Adapters (OA) and Interface Repository. It is necessary to ensure the inter-ORB interoperability of ORB specifications supplied by different vendors (Omg 2003).

The core communication of the CORBA architecture is the ORB core. The CORBA is a distributed client/server platform with an object Oriented spin, in the IONA's words (Iona 2001). When a client invokes a metnod of a remote object, communication between the client and the server is performed through the ORB (see figure 1). The client is connected to the ORB through a stub whereas the server uses skeleton. Stubs and Skeletons are generated a number of 4 files after compiling the IDL file. Although, stubs take the requests of the client and marshal data (format typed data into packet-level representation) to the server. And skeletons on the server side demarshal or unmarshal (deformat the packet-level representation back into typed data) and pass the request to the server so that it can serviced. CORBA uses an ORB as the middleware layer that etablishes a client/server relationships between the components. This layer allows distributed objects to communicate over network to each other without knowing their location and the programming language and operating system are used (Baker 1997).

Referring to figure 2, it indicates that object request semantics are handled via CORBA IDL. For transfer and message syntax, CORBA specifies its GIOP which specifies a set of message formats and common data representations for communications between ORBs. The GIOP was specifically included in the specification to address ORB-to-ORB interactions. It is designed to work directly over any connection-oriented transport protocol. To facilitate interoperability between different ORB implementations, the IIOP specifies how GIOP messages are exchanged over a TCP/IP network.

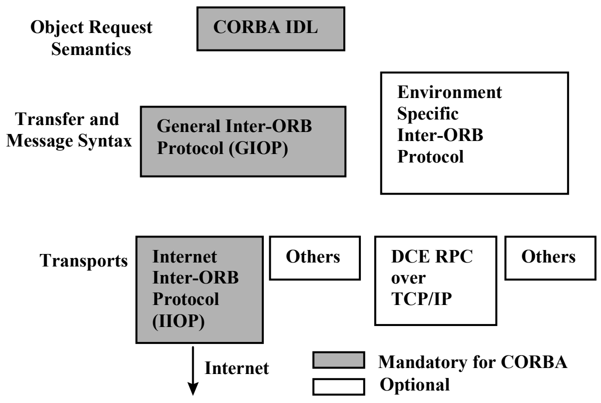
Object Request Semantics
CORBA IDL
Transfer and Message Syntax
General Inter-ORB Protocol (GIOP)
Environment Specific Inter-ORB Protocol
Transports
Internet Inter-ORB Protocol (IIOP)
Others
DCE RPC over TCP/IP
Others
Internet
Mandatory for CORBA
Optional

Figure 2 CORBA Internet Inter-ORB Protocol Architecture

## 4 Distributed Simulation

The main idea to integrate the ESP-r and Matlab tools in a combined simulation environment is to identify the specific components that should be coupled to ensure a distributed simulation. The CORBA standard (Omg 2003) and its IDL provides an easy and efficient way to define interfaces for distributed components written in programming languages (like C, C++, JAVA, …) for which so-called mappings are defined. Since such mappings are available, applications can be coupled via CORBA.

Conversely ESP-r is written in FORTRAN for which no IDL mapping is defined whereas for Matlab/Simulink, the COM (Common object model) standard mechanism is supported. In this case, the DCOM (Distributed Component Object Model) is used as well. The usual way of interfacing such tools into CORBA objects is to adapt a schematic approach of IDL to ESP-r and Matlab compiler, as shown in figure 3. In that case, ESP-r is used as a client/master and Matlab as a server/slave in order that Matlab can be launched at every ESP-r time-step as a separate process. With such an approach, a run-time coupling should support loosely coupled processes which can run concurrently and exchange data through ORB.

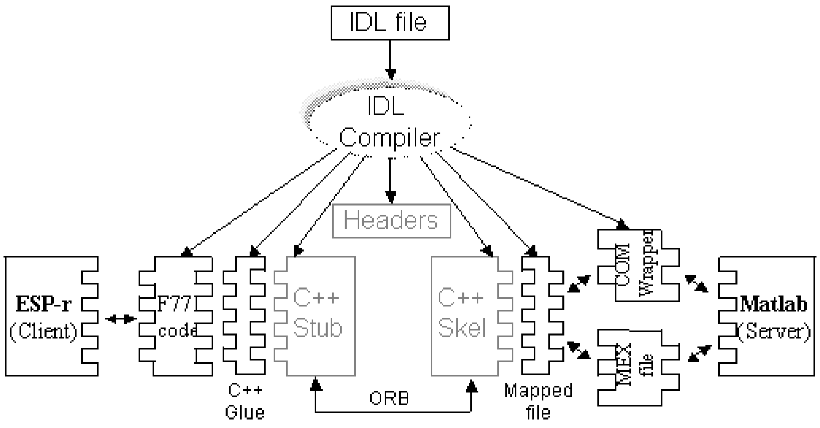

Figure 3 Architectural solution of IDL to ESP-r and Matlab compiler

A prototype IDL file has been created that can be configuired to create stubs and skeletons in both of ESP-r and Matlab. For ESP-r side, a C++ glue code is required to interoperate programs between languages. A (memory) mapped file is used for Matlab side to interact with ORB. C++

Stub and Skeleton codes can be generated by CORBA IDL compiler, such as ORBacus (Iona 2004.), for distributed applications.

The benefit to use ORB is to reduce the communication overhead. But, using this approach the overhead can be important. Much of the perceived overhead in this approach is inherent in a distributed environment and can mediate data representations. However, there is no solution to avoid undesirable overhead for FORTRAN part because of the intermidiate C++ glue.

In Matlab side, COM object can be developped to wrap up the CORBA IDL interfaces and structures necessary to access data services. A MEX file is a DLL that exports a standard entry point that MATLAB uses when a function call is invoked. This type of interface might introduce serious synchronisation issues because the implementation of CORBA-COM gateway, or bridge (Geraghty et al. 1999), is not an easy solution. Sometimes, it can also become the bottleneck for large data transfers, which isn't suitable for distributed simulations that use large amount of data per transfer. (MathWorks 2004).

### 4.1 IDL to ESP-r binding

Ideally, the IDL to ESP-r bindings should be F77 bindings as ESP-r is a legacy FORTRAN codes. However, a direct mapping from IDL to FORTRAN is not available. One solution could be to map IDL to C++ and then convert the C++ code to FORTRAN; this is an area where more work is necessary because of a lack of a formal conversion from F77 to C++ codes and precision issues. Another option that is more promising to use the defined IDL-to-C++ mappings and wrap the FORTRAN applications in C++ codes.

The integrated C++ wrapper approach avoids the overhead mentioned above. With this alternative solution, a C++ wrapper incorporating the CORBA implementation is linked to conserve and decompose the codes into functions modules and wrap each module into a distributed object, as described and called f2CORBA conversion tool in (Follan et al. 2001). The objective to use this current approach is keep modification of ESP-r source codes to the minimum. Moreover this f2CORBA converter isn't available as it was developed for NASA Glenn Research Center (Sang et al. 2000, Sang et al. 2001, Follan et al. 2001). Therefore, the only possibility to use such a converter is to redevelop the approach, which is not evident in terms of time and profitability of research efficiency.

There is a simple example of the different steps that shows a mapping structure from IDL to FORTRAN code in Section 5.

### 4.2 IDL to Matlab binding

The IDL to Matlab bindings are somehow simpler than F77 bindings since a COM standard is supported on the Windows platform, which is necessary to wrap the data from the application into a form that COM can marshal to MATLAB. MATLAB then unwraps the data from the COM data type and converts it into the MATLAB data type. This marshalling and transferring of data is necessary for isolating the two applications, but becomes a bottleneck when working with large data sets. A more detail description of Matlab COM integration can be found in (MathWorks 2004). Some bridges for inter-working between the COM and CORBA are available, like (OrbixCOMet 2000) and (COM2CORBA 2002). But the bridge is necessary to be adapted to the schematic approach described above. A client accesses COM interface when a call is performed, the proxy object maps all in-parameters from CORBA to COM, invokes the COM function, maps all out-parameters from COM to CORBA, and returns to the client.

## 5 Example

This section gives an example that encapsulates a very simple arithmetic operation written in FORTRAN code to CORBA IDL. This example is used to bind from IDL to ESP-r code. The present development process from IDL to implementation consists of writing the IDL file, implementing a client and a server, and developing a server implementation file. Three example programs are given below.

Here is a sample of the IDL specification of a very simple interface:

```
// Implementation of the mult operation
interface Mult {
  long step(in long y1, in long y2);
};
```

Let's assume that in FORTRAN program, a simple function is as follow:

```
integer  function multiple (x1, x2)
integer x1, x2
multiple = x1*x2
end
```

The function *multiple* should be declared as a *multiple* class implementation, in the server implementation file, which itself call the existing FORTRAN function *multiple*. The form of this part file might depend on data type calling form the C++ to FORTRAN and some special points when passing parameters between those languages. A simple example to illustrate this part file can be in the following structure:

```
…
extern "C" long multiple_ (long&, long&);
long mult_srv::step(long y1, long y2, CORBA::Environment &IT_env)
{
long mult;
mult = multiple_(y1, y2);
return mult;
} …
```

The rest of the application (client and server codes) can be implemented using the basic nature of the CORBA specification used. Many studies in developing CORBA applications can be found in (Omg 2004). This topic is beyond the scope of this paper.

By using the above encapsulation approach, it is possible to build several applications, but becomes very complicated when working with legacy scientific codes. Therefore this approach isn't suitable for the applications underlying mathematical equations, as a critical point for their simulation efficiency is to avoid any data copying.

## 6 Discussion

It was stated above that the use of CORBA-based integration of control modelling software and building performance environments by run-time coupling is efficient and flexible as data movement between applications do not have to be hand coded. Although, CORBA facilitates interconnection between heterogeneous systems since we envisage that ESP-r runs under Unix and Matlab may run in Windows. Moreover, its communication part is responsive to exchange data between ESP-r and Matlab via IDL links and its data exchanged between these environments is performed at high-level of abstraction. For those reasons, the motivation to use CORBA for the current work was highly supported. But, the adoption of CORBA for distributed control and building performance simulation by run-time coupling raises difficulties mainly when dealing with the interfacing of ESP-r and Matlab tools. The usual approach, as described above, is not efficient in terms of the communication overhead that can mediate data structures because of using C++ glue code. However, the CORBA-based distributed ESP-r and Matlab simulation could be much useful if the approach of f2CORBA converter (Follan et al. 2001) can be used. Then the problem is not the implementation of IDL ESP-r interface, but rather the performance of the implementation of CORBA-COM gateway that matches the interoperability between the COM and CORBA with the Matlab interface.

In different way of thinking, it might be conceivable to develop another approach prototype, if the f2CORBA converter is used, that consists to generate the IDL interface for every tool used in distributed simulation. This approach would be based on the use of two CORBA servers that ensures Matlab and ESP-r tools running concurrently and exchanging data through ORB protocol. The CORBA client, here, constitutes a distributed control software and building performance environments that uses their IDL interfaces with the intention to coordinate the exchanged data between them. Nevertheless, CORBA is difficult because distributed object-oriented programming is inherently complex.

## 7 Conclusions

Communication at high-level network programming interfaces, like CORBA, has many advantages over communication at low-level of abstraction, which is non-typesafe programming intefaces (Fatoohi 1996), such as sockets. For that purpose, the use of CORBA for distributing simulation tasks can assemble the simulation environment in confortable way and increase the computoing performance as weel. But, CORBA, in particular, is difficult to grasp due to its concept of middleware.

In this paper, we have presented an approach to interface ESP-r and Matlab applicatios with CORBA baded on IDL techniques. Issues and strategies regrading the encapsulation method of Fortran codes into CORBA objects and the interfacing Matlab function with CORBA IDL have been discussed. We have also developped a method which encapsule a program written in Fortran code with CORBA stuff. Although we have potentialy considered the use of CORBA-based development integration of control software and building performance environments by run-time coupling. Using such an approach, it requires much time and extra materials to develop it in perferble way. At this time, we think that it is promissing to use applications within simple abstractions such as pipes, sockets and shared memory segments to exchange data between address spaces.

## 8 References

Fatoohi, R. (1996). Performance Evaluation of Communication Software Systems for Distributed Computing. Report NAS-96-006, eds. NASA Ames Research Center, USA.

Follen, G., Kim, C., Lopez, I., Sang, J., and Townsend, S. (2001). *A CORBA-based Distributed Component Environment for Wrapping and Coupling Legacy Scientific Codes*. In proceedings of "Cluster Computing", IEEE, Vol. 5, Issue 3, 277-285.

COM2CORBA (2002). ICL DAIS COM2CORBA. <http://filemakeradvisor.com/Solutions.nsf/0/e8ede0ebcf95a813882565fd0075b9d2?OpenDocument> (ADVISOR MEDIA, Inc.).

Geraghty, R., Joyce, S., Moriarty , T. and Noone, G. (1999). COM-CORBA Interoperability. Prentice Hall PTR, USA.

Hughes, C. and Hughes, T. (2003). Parallel and Distributed Programming using C++. Addison-Wesley, USA.

IONA (2001). CORBA/C++ Programming with ORBacus. IONA Technologies, printed in USA.

IONA (2004). ORBacus CORBA Soouce Code <http://www.orbacus.com/support/new_site/index.jsp> (IONA Technologies, Inc.).

Keahey, K. and Gannon d. (1998). *Developing and evaluating abstractions for distributed supercomputing*. In proceedings of "Cluster Computing"1(1): 69-79

MathWorks (2004). MathWorks's website. < http://www.mathworks.com> (The MathWorks, Inc.).

Omg (2003). The Common Object Request Broker: Architecture and Specification. <http://www.corba.org> (Object Management Group, Inc.).

OrbixCOMet (2000). CORBA-to-COM Mapping. <http://www.iona.com/support/docs/orbix2000/2.0/comet/html/IDL_COM.html - 2.7KB - IONA> (IONA technologies, Inc.)

Ranganathan, M., Acharya, A., Edjlali, G., Sussman, A. and Saltz, J. (1996); *Flexible and efficient coupling of data parallel programs*. In Proceedings of "Parallel Object-Oriented Methods and Applications (POOMA96)".

Sang, J., Follen, G., Kim, C., Lopez, I. and Townsend, S. (2001). *Migrating Legacy Scientific Applications towards CORBA-based Client/Server Architectures*. In proceedings of "Software -- Practice & Experience", Vol. 31(14), 1313—1330.

Sang, J., Kim, C., and Lopez, I. (2000). *Developing CORBA-based Distributed Scientific Applications from Legacy Fortran Programs*. In Proceedings of "the HPCC Computational Aerosciences (CAS) Workshop".

Yahiaoui. A., Hensen J.L.M. and Soethout L.L. (2003). *Integration of control and building performance simulation software by run-time coupling*. In Proceedings of "IBPSA Conference and Exhibition 2003",Vol. 3, pp. 1435-1441, Eindhoven, NL.