A PROPOSAL OF A DISTRIBUTED COMPONENT ENVIRONMENT FOR THE INTEGRATION OF SIMULATION MODELS

Alfredo Anglani, Antonio Grieco, Massimo Pacella*
Dipartimento Ingegneria dell'Innovazione
Università degli Studi di Lecce
Via per Monteroni, 73100 Lecce, Italy

Lucio Colizzi
Divisione Ingegneria Informatica
Consorzio CETMA – Centro di Progettazione Design & tecnologie dei Materiali, Cittadella della Ricerca
Strada Statale 7 km 7 + 300 per Mesagne, 72100 Brindisi, Italy

KEYWORDS

Discrete event simulation, distributed component computing, CORBA, ARENA.

ABSTRACT

The simulation of complex systems, by means of computerised models, is shifting to a new paradigm: the DCC (Distributed Component Computing). The applications of this simulation paradigm are client/server running – programs that use collaborating distributed components. These components may be located on different platforms with different operating systems, and they may be developed in heterogeneous simulation languages. In this paper, a framework for the integration of two diverse discrete event simulation languages is discussed. The integration is achieved by implementing a suite of abstractions and of simulation services, which are based on the distributed component platform technology CORBA (Common Object Request Broker Architecture). Software components can be used to assemble simulators from a variety of heterogeneous services and models.

INTRODUCTION

Nowadays, simulation applications are becoming very complex software models. This is mainly due to the increasing complexity of the analysed systems that, as those in the manufacturing production field, are characterised by several groups of co–ordinated interacting elements (Kellert *et al.* 1997). The growing complexity of simulation projects involves the increasing of both the required simulation devices, and of the heterogeneity levels among them. To address this complexity, future simulation applications must be substantially redesigned from a software implementation perspective. Several works are studying the application of component – oriented paradigms to develop simulation models in the manufacturing field (McArthur *et al.* 2002). In particular, the work done within the Object Management Group (OMG) has led to the definition of the Computer Integrated Manufacturing (CIM) framework, which is a distributed component – oriented architecture for the integration of simulators in the manufacturing environment.

A promising approach to re – design simulation applications is to move toward the Distributed Component Computing (DCC) paradigm in which monolithic software systems are being replaced by a collection of different components (Sheremetov and Smirnov 1999). A valuable feature of the DCC paradigm is that it is heterogeneous. Ideally, heterogeneity permits to use the best combination of hardware and software elements. In the simulation field, that implies the option to use the more suitable simulation tool in order to model each specific part of the analysed system. The DCC paradigm allows the developer to implement complex simulation models by simply connecting a set of elements that provide a variety of simulation services (McArthur *et al.*).

Large efforts have been made in the last decade in order to combine simulation models. For example, the High Level Architecture (HLA – IEEE standard 1516) defines a framework that makes interaction possible for various simulation components. The aims of HLA are mainly to get an interoperability of the simulators and to reuse components over a large number of applications. The HLA framework is able to establish the technical foundation for the combination of sub – models on the same planning level using the same simulation method but probably different simulation tools (Wilcox *et al.* 2000). Nevertheless, the HLA does not solve the problem that arises if different paradigms, levels of detail and points of view occur when simulation models are exchanged and coupled. Examples include models which combine object – oriented components and transaction – oriented modules, or the coupling of continuous process simulation with discrete event simulation.

The objective of this work is to present a framework, based on the standard CORBA, that allows two heterogeneous discrete event simulation tools (ARENA and DEOS) to interact themselves. ARENA by System Modeling Corporation (Pegden *et al.* 1995) is a graphical transaction – oriented language for discrete event simulation; it is one of the most commonly used simulation languages at this moment both in academic and in industrial fields. DEOS has been implemented at the University of Lecce (Caricato *et al.* 2000) in order to supply a C++ class library able to provide a substantial support for the development of object – oriented discrete event simulation models. The presented framework consists of heterogeneous components that are interoperable, reusable and operate on a common platform.

The remainder of the paper is organised as follows. In section two, the tools, which have been exploited for the integration of two simulation languages, are briefly described. In section three, the integration software framework is reported. In the fourth section, the implementation of plug – in interfaces for ARENA software is discussed, while in section five, the plug – in software for the DEOS environment is presented. In section six, a simple application example is presented. Finally, conclusions and future development issues are both briefly discussed.

THE INTEGRATION TOOLS

The need for interaction among software components led to the specification of middle – ware models. The Object Management Group's Common Object Request Broker Architecture (CORBA) and the Microsoft's Distributed Component Object Model (DCOM) are two platforms that enable software objects to work

* Corresponding author, e-mail: massimo.pacella@unile.it, fax: +39 0832 320 279

together. In this section, the tools, which have been used for the integration of the ARENA and DEOS simulation environments, are briefly presented.

Common Object Request Broker Architecture (CORBA)

The Common Object Request Broker Architecture (CORBA) is an Object Management Group (OMG) specification that defines the framework required to develop distributed object – oriented software systems. The idea is to provide the users of an object – oriented programming model, for distributed computing, that is as close as possible to the programming with normal local objects.
In CORBA, the applications are divided into the client part, and the server part. The client part provides the user interface, and it has an interface toward the server part fitting to the Interface Definition Language (IDL) specification. The server part can be a single object, or a group of objects, which can be positioned in any location, and are retrieved by means of the ORB. The ORB is the software layer that sets up the client/server relationship among objects. The ORB captures the call, it finds an object that can implement a request and passes it the parameters. Finally, it invokes the server method and returns the results to the client. The ORB provides interoperability among applications on different platforms in heterogeneous distributed environments. CORBA offers a number of services that provides language – linking systems for different object – oriented programming languages.

Interface Definition Language (IDL)

Each CORBA object has a defined interface, specified in the Interface Definition Language (IDL). An interface specifies the operations that the object supports, and thus, it describes the requests that can be made to that object. It is kept independent from the implementation of the object. To use a software module that has been transformed into a CORBA object, the user is only required to see its interface. IDL compiler software compiles the IDL interface. IDL compilers are available for several programming languages such as C++ and Java. The use of IDL to define object interfaces allows these interfaces to be used from a variety of programming languages and computing platforms.

ORBacus

Practically, a software implementation of the CORBA specification is referred to as an ORB (Object Request Broker). In the implementation presented in this work, the ORBacus software has been chosen. Some of the highlights of ORBacus are: 1) full CORBA IDL supporting, and 2) complete CORBA IDL to C++ mapping. Because CORBA does not require implementation, a well – designed ORB does not require that components and technologies already in use must be abandoned. Instead, the CORBA specification allows ORBs to incorporate and integrate existing protocols and applications, such as Microsoft DCOM, rather to replace them.

DCOM and ActiveX Automation

ActiveX is the Microsoft's marketing name for technologies that enable interoperability using COM (Component Object Model). The target of COM is to allow two or more applications or objects to easily co – operate with one another. The mechanism of interaction between COM objects and client applications are generally the same, no matter where they have been deployed. COM objects can be *in process*, which means that the interaction is fast and efficient. Applications can also interact with COM objects in another process on the same machine (*cross process*). Finally, *Distributed COM* (DCOM) allows a client to interact with a COM object across the network. DCOM objects are delivered as compiled objects rather than a source code. This implies that the end user can use these components without having to understand how the object is implemented.

Simulator Configuration & Control Tool (SCCT)

The Simulator Configuration & Control Tool (SCCT) is a software environment, developed at the University of Lecce (ESPRIT Asia project – EP n. 28661, Baresi and Coen-Porisini 2000). It is based on the CORBA platform, and it is conceived to develop distributed simulation systems (fig. 1). The SCCT is used to generate the simulation architecture, setting simulation parameters, defining simulation probes and expected measures, monitoring and controlling on – going simulation.

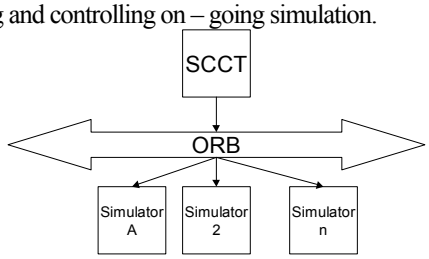

Fig. 1 – The SCCT role on the CORBA framework

The SCCT assigns the simulators to the system architecture components; moreover, it defines both data flows (local parameters, filters, batches) and control flows between them. Practically, the SCCT can be considered as a controller of the integrated simulation. The simulators operate as black boxes accessible only by some services defined by means of IDL (fig. 2).

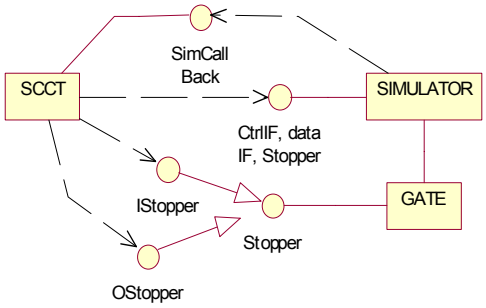

Fig. 2 – The SCCT/simulator module interaction

Each simulator component included in SCCT must support two kinds of interfaces.

- The Control Interface (CtrlIF, fig. 3): through which SCCT manages the execution flow (simulator start, stop and pause).
- The Data Exchange Interface: it is composed by The DataIF, Stopper and StopperFactory interfaces (fig. 2). By means of such interfaces, the SCCT passes the data produced by one simulator to another one.

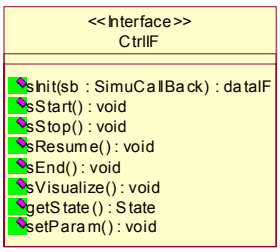

Fig. 3 – The CtrlIF interface object

The interaction between SCCT and the simulators is obtained by means of objects that communicate through the IDL interfaces on the ORB. The SCCT manages both the control functions and the exchange of data between the simulators. This is made by means of simulator – gates that model input and output points of the simulator component. A single interface is presented by SCCT to the simulator: the SimuCallBack.

ARENA AND DEOS INTEGRATION

The goal of this work is to develop a *plug – in* system, i.e. an appropriate interface that allows ARENA and DEOS programs to interact by means of the CORBA/SCCT framework. The integration of ARENA and DEOS by means of the CORBA/SCCT framework requires to solve two problems (Grieco *et al.* 2001).

1. *The interaction of the SCCT framework and simulation components.* The SCCT can be considered as a controller of the integrated simulation. The running control functions required by the SCCT (e.g. *start*, *interrupt*, and *pause*) are usually available in the commercial simulation environments (such as ARENA). However, it is necessary to implement a mechanism that allows the SCCT framework to use them run – time (i.e. during a simulation run).
2. *The synchronisation of simulator components.* At the end of a simulation run, the SCCT, by means of the ORB, may transfer the resulting output to a specific group of different simulation objects. In this way, a simulator can be informed that some data are produced, but it does not known *when* they have been produced. The "timing" information is fundamental in a distributed discrete event simulation..

At the current state of the research, integration can be obtained only for sequentially – coupled modules. That means that each single module of the overall simulation model (assuming that it is coupled with *n* input and *m* output modules) cannot have input produced by one of its own output modules (i.e. no recursion is yet possible). The timing information, which is used for the synchronisation of simulators, is passed from a specific module to the following ones by means of shared files. Such files are processed by special objects (namely the *Gate_IN* and *Gate_OUT*) both implemented in the ARENA and DEOS environments. The *Gate_OUT* object collects the entities, and the temporal information, that pass from the current simulation module to a different one. The *Gate_IN* transforms the input timing information, into the starting list of events for the actual simulation module.

The implementation of the ARENA and DEOS plug – ins are based on the *Windows Automation Technology* (i.e. the DCOM model). The DCOM is a standard model that establishes the rules of interaction between different software. This is made possible by using a particular set of Application Program Interfaces (API) collected in specific libraries. Various programming languages support the Windows Automation Technology providing users by mechanism in order to create both the *Automation Controller Model* and the *Automation Object Model*.

In the reference case study the Automation Controller Model (the client) coincides with the procedure recalled by the SCCT, the Automation Object Model (the server) is one (or more) object that represents either the ARENA or DEOS simulator.

ARENA

ARENA exploits Windows Automation Technology. Therefore, the ARENA Automation Object Model, which is a list of application objects that can be controlled by external applications, has been used in order to integrate the ARENA environment to the CORBA/SCCT framework. The Automation Object Model is registered when the application is installed. By using this model, the implementation of the ARENA plug – in systems can be divided in two parts: 1) the *Control Management* and 2) the *Data Exchange Management*.

The *Control Management* consists in the implementation of the methods reported in the SCCT CtrlIF interface. Two particular objects of the ARENA Automation Object Model, has been used in order to implement these methods:

- ARENA "Application" Object (fig. 4, fig. 5): it presents a set of methods that allow an object to access the ARENA simulation environment. In our application, it is invoked by the method Init of the SCCT CtrlIF interface in order to recall the set up functions of the ARENA environment (as GetApplication, Refresh and Activate).

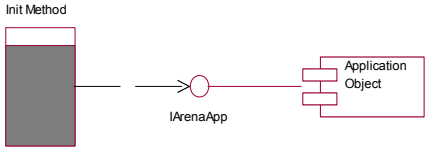

Fig. 4 – ARENA Application Object

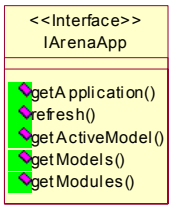

Fig. 5 – The ARENA Application object interface

- ARENA "Model" Object (fig. 6, fig 7): it allows an application object to control the simulation running, as well as to open, to create and to close an ARENA simulator model.

All methods have been implement as independent threads. In this way, SCCT and the simulator modules can operate in a parallel manner.

In SCCT, data can be exchange through the Stoppers. The methods that are used in this case are "Send" and "Retrieve". With the former, the SCCT sends input data to a simulator, with the latter it captures the output produced by a simulator. At the current state of work, the *Data Exchange Management* is implemented by means of files. Two kinds of files may be use as input: 1) TXT files that contain temporal information for a specific simulation model, and 2) DOE files that specify the simulation model.

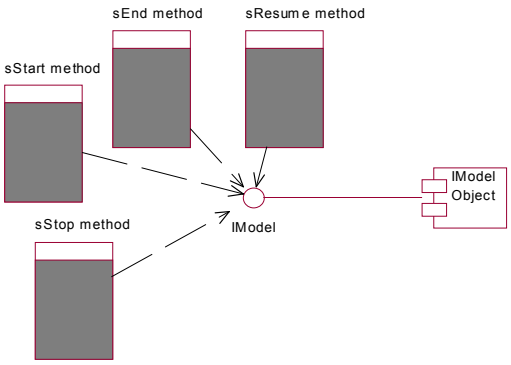

Fig. 6 – ARENA Model Object

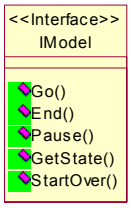

Fig. 7 – The ARENA Model object interface

DEOS

The implementation of the DEOS plug – in system has involved a preliminary phase. In this phase, the appropriate DEOS Automation Object Model (called DEOS Object, fig. 8, fig. 9) has been implemented and embedded in the distributed environment. The DEOS Object has a unique interface, called IDeos, which methods can be referenced to by the applications that manage the object. The development of the DEOS plug – in can be subdivided in two phases: 1) *Control Management* and 2) *Data Exchange Management*.

The DEOS Object is controlled by the SCCT interface method CtrlIF in order to interact both with the DEOS running environment (to run or to stop a simulation) and with a DEOS simulation model (to send the input parameters). The management of the simulation session is obtained by means of the methods Start, Resume, Stop and End.

By means of the Send method, the DEOS plug – in is able to manage two kinds of files: 1) TXT files, which contain the temporal information for a given simulation model, and 2) DSF files, which have the DEOS simulation model.

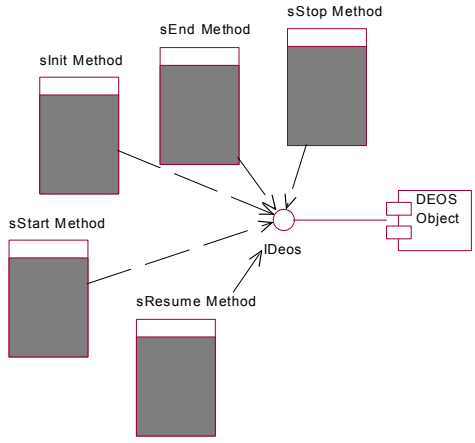

Fig. 8 – The DEOS Object

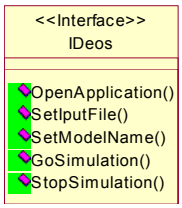

Fig. 9 – The DEOS interface object

AN APPLICATION

In this section, a simple example of the ARENA and DEOS integration is presented. In the following case study, a machining centre (MC) and a buffer queue compose the reference environment. The purpose of the MC is to transform raw parts in final products.

Let us assume that the simulator has been divided into two different components. The first one models the arrival succession of raw pieces, while the second one simulates the buffering and the machining processes. In particular, it has been decided to model the first component by means of the DEOS language (fig. 10). It consists of a block that generates the entities at interval times that follow a specific statistical distribution. On the other hand, the second module, which simulates the sequence of the buffering and of the machining operations, has been implemented in the ARENA environment (fig. 12).

The two simulation modules may be located on different platforms and they communicate to the CORBA/SCCT framework by means of the "Gate IN" and "Gate OUT" modules. From the user viewpoint, the "Gate OUT" is a particular module of the environment that allows linking an external simulation component even if it has been implemented in a different language on a different machine. As well as the "Gate IN" module represents the entry point for the output produced by a different simulation component. Both the "Gate IN" and the "Gate OUT" module has been implemented in the ARENA and in DEOS language as interface modules between the simulation environments and the CORBA/SCCT integration framework.

Fig. 10 – The DEOS simulator

Once the simulation components have been fully implemented in the specific language and located on the specific platforms, the user models the overall simulation schema by means of the graphical SCCT interface (fig. 11). In the referenced case study, this implies to implement a simulation schema composed by two sequential simulator components (DEOS and ARENA fig 11), two input points and a single input output. Each input point (Input1 and Input2 in fig. 11) is used in the integration environments in order to control the overall simulation run. The ARENA simulator component produces the output of the simulation.

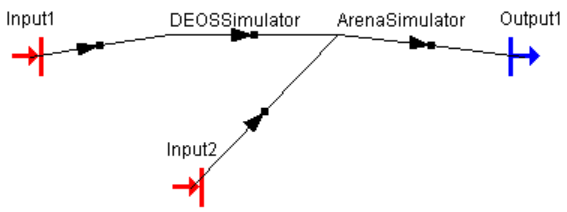

Fig. 11 – SCCT overall simulation schema interface

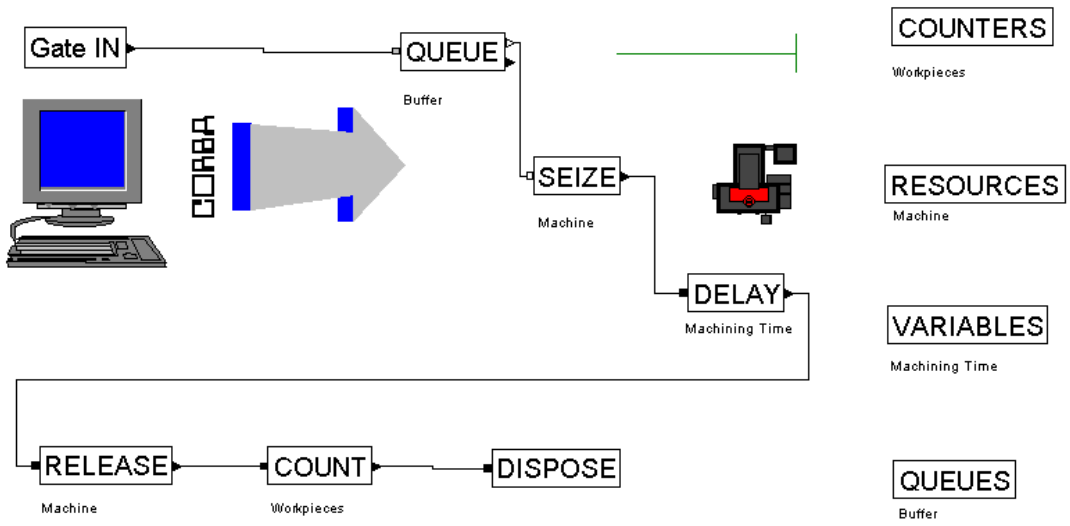

Fig. 12 – The ARENA simulator

CONCLUSIONS AND FUTURE ISSUES

CORBA has the potential to address the problem associated with the need for interoperability among the huge number of simulation software products available today. In this paper, an integrated approach for discrete event simulation, which integrates two heterogeneous simulation languages (ARENA and DEOS) by means of CORBA, has been presented. The integration is achieved by constructing a suite of abstractions and simulation services built on a common kernel. The abstractions and services are encapsulated into combinable software components.

The most important benefit of the proposed framework, at the current state of the research, is that it allows the user to integrate different models that are developed by two heterogeneous language: the transaction – oriented ARENA environment to the object – oriented DEOS language. From end user viewpoint, each module, which encapsulates a specific simulation sub – model, can be integrated with each other no matter where it is located or how it has been implemented. Moreover, as indicated by the example, a distributed simulation model can be implemented in a simple and intuitive manner.

A future development consists in the extension of the presented framework in order to execute co – simulation. By co – simulation, we mean the possibility of having simulators exchanging data during the actual simulation no matter how they are coupled. This requires that all simulators share a notion of a "global clock" so that the temporal information that is exchanged within them can be considered consistent with each single simulator clock.

ACKNOWLEDGEMENT

The work described in the paper has been partially funded by the Ministry of Instruction, University and Research of Italy (MIUR), project PRIN2001 – prot. MM09164148_004 'Models for capacity planning in advanced manufacturing systems'.

REFERENCES

Grieco A., M. Pacella and A. Anglani "Integration of heterogeneous discrete event simulation tools by means of CORBA", *Proceedings of the annual conference of the Italian Society for Computer Simulation*, Naples (Italy) December 6-7, 2001, 61-69.

Baresi L. and A. Coen-Porisini "An Approach for Designing and Enacting Distributed Simulation Environments" *Proceedings of the 16th IFIP Word Computer Congress – Proceedings of Conference on Software: Theory and Practice, ICS 2000*, Beijing (China), August 25-28, 2000, 637-645.

Caricato P., A. Grieco, F. Nucci, A. Zacchino and A. Anglani "An open-source visual environment for discrete event simulation: DEOS", *Proceedings of the annual conference of the Italian Society for Computer Simulation*, Naples (Italy) December 6-7, 2001, 47-54.

CIM Framework Architecture Guide 1.0, http://www.sematech.org/ public/ docubase/ abstracts/ 3379aeng.htm (September '02).

High Level Architecture, https://www.dmso.mil/ public/ transition/ hla/, (September 2002).

Kellert P., N. Tchernev and C. Force, "Object Oriented Methodology for FMS modelling and Simulation", *International Journal on Computer Integrated Manufacturing*, vol. 2, (1997) no. 6, 405-434.

McArthur K., H. Saiedian and M. Zand, "An evaluation of the impact of component-based architectures on software reusability". *Information and Software Technology*, vol. 44, (2002), 351-359.

Microsoft COM White Papers Internet Web Page, http://www.microsoft.com/ com/ wpaper/ default.asp, (September '02).

Object Management Architecture (OMA) Guide http:// www.omg.org/ technology/ documents/ formal/ object_management_architecture.htm (September '02).

ORBacus home page, http://www.iona.com/ products/ orbacus_home.htm, IONA Technologies, (September '02).

Pegden C.D., R.E. Shannon and R. P. Sadowski, *Introduction to Simulation Using SIMAN*, (McGraw-Hill, 1995).

Sheremetov L. B., A. V. Smirnov, "Component integration framework for manufacturing systems re-engineering: agent and object approach", *Robotics and Autonomous Systems*, vol. 27 (1999), 77-89.

Wilcox P. A., A. G. Burger and P. Hoare, "Advanced distributed simulation: a review of developments and their implication for data collection and analysis". *Simulation Practice and Theory*, vol. 8, (2000), 201-231.