A TOP-DOWN APPROACH TO MODEL INTEROPERATION PROVISION IN COTS SIMULATION PACKAGES

Michael D. Ryde
Simon J. E. Taylor
Centre for Applied Simulation Modelling
Department of Information Systems and Computing
Brunel University, Uxbridge, UB8 3PH. UNITED KINGDOM

ABSTRACT

This paper examines current methods for model interoperation when using COTS (Commercial Off-The Shelf) simulation packages. The viewpoint taken for this work is from that of the simulation engineer. By applying distributed simulation theory an attempt is made to suggest how an example COTS simulation package could be modified to provide the necessary functions and interoperability required. Further, by studying current methods employed, which enable COTS simulation packages to interoperate, this paper will discuss the tools currently used, examine their appropriateness and suggest further areas of research.

INTRODUCTION

The interoperation of simulation models through distributed simulation has provided many areas for academic research. Much of this research has been focused on the technological challenges faced by software engineers who have strived to determine the most efficient and accurate way of enabling simulation models to communicate. The technological problems have included (amongst others) Web and Network based simulations, see Miller et al. 2001, model and object reuse, see Pidd 2002, model synchronization, see Fujimoto 1990 and 1999 and the technical implications of using distributed simulation in a specific application area, for two examples from many, see Zeigler et al. 1999, Carothers et al. 1994.

A major contribution made by the distributed simulation field of research is the High Level Architecture (HLA). The standard (IEEE 1516) provides a framework for distributed simulation. Each model, or federate, interacts with each other (interoperates) to accomplish the simulation exercise and the combined set of interoperating federates is referred to as a federation. The HLA gives standards for data representation (needed so that the communicating federates can "talk" the same language – the format of data exchanged between models) and middleware (to allow communicating parties to "talk" – this is the federate interface specification, the implementation of which is called a run time infrastructure, RTI).

Distributed simulation enabled by the HLA has been used extensively in military systems (see previous Winter Simulation Conferences and SISO's Simulation Interoperability Workshops for many examples). There have been relatively few examples of this in industry. This is not for the lack of opportunity. See Strassburger 2001 for an in-depth discussion on how the HLA could be used outside of the defence arena. Another use of the HLA outside the defence arena was put forward as part of the Intelligent Manufacturing Systems (IMS) mission project. See McLean and Riddick 2000. Interestingly, an observation made during this research was that the current RTIs (developed by different sources) did not interoperate with each other thus all models in a distributed simulation would need to use the same RTI.

Another RTI based development includes GRIDS, which provides a generic run-time infrastructure for the execution of distributed simulations. GRIDS provides basic simulation services to connect simulation models (federates) cooperating to perform a distributed simulation (federation), and extensible simulation services to provide performance enhancement, time-management, mobile entities, as required. Sudra et al. 2000.

This paper however, is primarily concerned with Commercial-Off-The-Shelf simulation applications and takes a top-down approach to the same issues mentioned above. We attempt to show how the nature of a COTS Simulation package can be changed through the provision of interoperational features. This approach is taken with the firm belief that simulationalists will use the tools at their disposal and currently, in many COTS simulation applications, model interoperation functionality either does not exist or requires software development knowledge to use. Arguably, this could be given as a significant reason for the low up-take and use of distributed simulation in commercial sectors.

This paper is divided into 5 sections. Following the introduction in section 1, Section 2 is concerned with the current definition of the COTS simulation package and highlights the typical attributes found in these packages. Section 3 examines how interoperation may be achieved using current packages and suggests potential pitfalls. Section 4 suggests possible enhancements that could be adopted by the software development houses to enable model interoperation. Finally, section 5 concludes and suggests further areas for research in this field.

Proceedings 15th European Simulation Symposium
Alexander Verbraeck, Vlatka Hlupic (Eds.)
(c) SCS European Council / SCS Europe BVBA, 2003
ISBN 3-936150-28-1 (book) / 3-936150-29-X (CD)

COTS SIMULATION PACKAGES

A typical COTS simulation package, for the purposes of this paper, is considered an application in which simulation models can be constructed, saved and reused. The model would normally be constructed from objects, some of which would be standardized between models. Further, it is also expected that the package would have some form of representation for entities (items of work) that would be used within the model. Typically, these packages would include definitions for entity distributions and methods by which various objects within the model could be linked or ordered. COTS simulation packages can be, and often are, used by various sized organizations but are easily accessible to even the smallest of businesses because of their low cost. Thus the diversity of model that the packages are expected to deal with is fairly broad.

Many COTS simulation packages have a Visual Interactive Modeling (VIM) interface, use event lists and have defined entities. In addition, these packages are accessible to many organizations due to their costing structures and as with many 'volume' packages are available on the Microsoft Windows platform. VIM interfaces provide a method of control to functions available to the user. Also, these types of interface often support a 'drag and drop' style of interaction making simulation model building a rapid process.

A brief review carried out during March 2003 revealed the following (although not exhaustive) list of COTS simulation applications:

1. ARENA (Rockwell Software)
2. AUTOMOD (Brooks Automation AutoSimulations Division)
3. Awe Sim (Frontstep, Inc.)
4. EXTEND (Imagine That, Inc.)
5. GPSS for Windows (Minuteman Software)
6. GPSS/H/Proof Animation/SLX (Wolverine Software Corporation)
7. iGraphx Process 2000 (Micrografx, Inc.)
8. microGPSS/webGPSS (Ingolf Stahl)
9. ProModel (Production Modelling Corporation)
10. QUEST (DELMIA Corporation)
11. SIGMA (Custom Simulation)
12. SIMPROCESS/SIMSCRIPT II.5 (CACI Products Company)
13. SIMUL8 (SIMUL8 Corporation)
14. Taylor Enterprise Dynamics (F & H Simulations)
15. Visual Simulation Environment (Orca Computer, Inc.)
16. WITNESS (Lanner Group, Inc.)

THE INTEROPERATION OF COTS SIMULATION PACKAGES

Currently, there are no known products that have the ability to support and natively allow, multiple models to interoperate without at-least the use of some basic middleware component. See figure 1. However, there are methods used to emulate the interoperation of models.

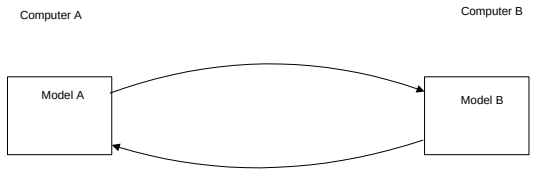

Figure 1: Native Model Interoperation

Usually simulation models require, as a minimum, input in the form of a distribution of entities. The entity distribution for a model could be taken from existing models by executing a number of experimental runs to determine the required spread and frequency. This information could then be passed directly into a model via a spreadsheet, see figure 2. Many COTS simulation packages provide functionality to write out to and read variables from a spreadsheet package in order to provide a way of passing information between models. In many cases this provides little more than the passing of information sequentially from one model to another.

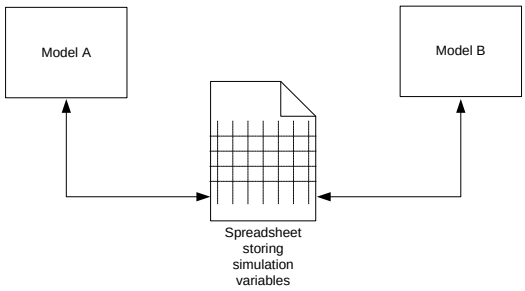

Figure 2: Simple Model Interoperation using a Spreadsheet package

To apply the same method to many models passing information (entities) to one another, one must consider the synchronization if causality issues are to be avoided. It is likely that if multiple models were running and passing information to each other, then these models could be running at different speeds; i.e. the simulation clocks could be different. Thus in figure 3, Model A, when receiving an event from Model B and Model C, would need to determine which event to process first. Using a spreadsheet package to facilitate the passing of entities may provide some limited mechanism for reading/writing time-stamped information, event list information and even synchronization logic (time-management). However it is suggested that a spreadsheet, using basic functions would be grossly

inadequate and such a mechanism would require some further middleware logic (program instructions) to give the required functionality. It can then be argued that the spreadsheet package is no longer acting as a simple data passing mechanism, more as a fully-fledged time-management component. Is a spreadsheet package really the best tool for the job in this case?

It has long been suggested that the distribution and interoperation of simulation models can be achieved through the use of a 'Spreadsheet' some evidence of this can be found in Clarke 1993. This we term as the *'Spreadsheet Approach'*, which, it is postulated, is inappropriate for all but the simplest interoperations.

As suggested earlier, it can be seen that using this method for distributed simulation cannot work without some layer of intermediate code to deal with the time-management functionality. It could therefore be assumed that programming skills would also be required by the simulation engineer in order to create this middleware.

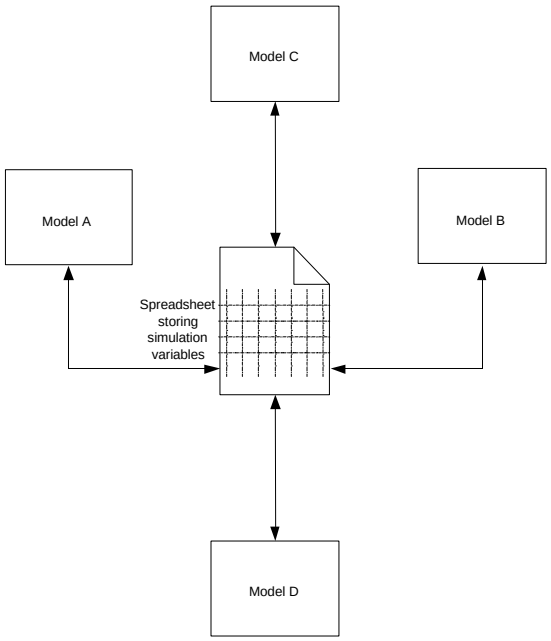

Figure 3: Complex Model Interoperation

The following is an attempt to build an initial set of requirements for the interoperation of simulation models using COTS packages:

- To be able to link objects in different models and use their output as entity distributions or actual 'parcels' of data where required. This provides the core functionality for the linking of models by connecting defined simulation objects, which generate simulation events and output entities.
- The facility to pass entity data between objects in disparate models. This will define the mechanisms to pass actual simulation data between the models and would involve the standardisation of entities throughout the entire distributed simulation model.
- The provision of access to control the starting and stopping of a model externally. This is essential so that models can be started and terminated in a synchronized manner.
- The implementation of time-management algorithms for model synchronization. Possibly the provisions of different synchronization algorithms could be provided so that the appropriate type of synchronization could be used for particular simulation problems.
- The ability to interrogate the event list in order to examine the next event before it is executed. This is required to facilitate the implementation of time management functionality, specifically look-ahead as used in the Conservative synchronization algorithms.
- Separate control for re-running C-Phase of operation as specified in the three-phase simulation methodology, see Brooks and Robinson 2001. Again, this would be required specifically for synchronisation algorithms.

SUGGESTED ENHANCEMENTS TO A COTS SIMULATION PACKAGE

It is believed that due to the way many COTS Simulation packages are designed adding interoperability could be relatively straightforward. For the purposes of this paper we restrict ourselves to one package, SIMUL8 (SIMUL8 Corporation). This package has a VIM interface and uses event lists with defined entities. SIMUL8 is an accessible package for many organizations due to its costing structure and is available on the Microsoft Windows platform. The VIM provides a high level of control to many of the technical features and functions available to the simulation engineer and the package is believed to be an appropriate candidate for our suggested enhancements. An attempt has been made to suggest new or modified functions and even a possible user interface, using SIMUL8 as an example.

We have also decided for the purposes of this case study not to address heterogeneous COTS simulation package interoperability.

Functions

Table 1 gives examples of functions that could be made available in COTS simulation packages such as SIMUL8. The authors of this paper have no knowledge of the internal mechanisms or software design that SIMUL8 uses and so these functions serve merely as general software design suggestions.

At the current time the main body of work has focused on run control and entity exchange. The functions suggested would allow a model to use external objects and variables and also enable the model to share it's

own objects and variables. Further, the distribution of an 'input' could be defined as an external function, providing an alternate method of distribution. A final function is provided to enable a selected model to become 'the master' for ease of control and synchronization of the 'Global Model' or 'Federation'.

The functions in Table 1 serve merely as example functions, which could exist within an API (Application Program Interface), but are not intended to represent a complete list. However, they do serve to highlight some important mechanisms, which are required to provide external control and entity exchange within the COTS simulation package.

Table 1: Run control and entity exchange functions

Function	Description
Handle ExternObj(Object)	Externalisation of objects for external access. Returns handle to object.
Handle Extern(Variable)	Externalisation of variables for external access. Returns handle to variable.
SetMaster(Boolean)	Set Master Model - Allows a specific model to be set as a master to stop and start the entire simulation.
Entity GetExternDist(Model, FromObject, ToObject)	Get external distribution - Modify existing routine to interrogate objects within separate SIMUL8 models for distribution patterns. Returns Entity.
Boolean LinkExternal(Model A, Object, Model B, Object)	Links object in model A to an external object in model B. Returns True if successful.

SIMUL8 supports the notion of Plugins, which enable specific software modules to be integrated in to the package. A possible use for this could be for time-management algorithms. This could allow different synchronization protocols to be used when models have been distributed. The Plugins could include Conservative (lookahead, lookback and null message protocols) and Optimistic (Time Warp) algorithms. The integration detail is expected to be more complex for these software components; however, the mechanism could provide a neat and elegant solution to the problem.

SIMUL8 Application Programming Interfaces

Although strictly not relevant to the simulation engineers (due to the requirement of software development skill), the application programming interfaces (API's) provide the first steps towards interoperability. Once the necessary native functions have been introduced to the application, it is not unreasonable to expect separate organisations and even users with software development experience, to develop standardized middleware to be used by general simulation engineers (to allow model interoperability). Currently SIMUL8 supports API's at a number of different levels, i.e. OLE Automation, COM and through the ActiveX interface. There are also some direct linking facilities, using the user interface, which can enable the user to link to Microsoft Excel or Visual Basic (although these probably use the facilities provided in the API).

SIMUL8 Interface Suggestions

Modifications to the SIMUL8 interface will be required to enable the Simulation Engineer to design interoperating models. Below are suggested interface enhancements to provide access to the interoperability functionality, primarily focusing on model selection, object linking and setting the master control.

Selecting External Models

The current object linking box in SIMUL8 version 9 provides a mechanism to link various objects within the same simulation, see figure 4. Figure 5 suggests a modification to this dialog box to allow links to be made to external objects by first selecting the model in which the object resides.

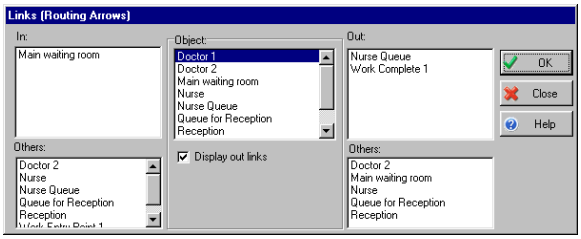

Figure 4: Current Object Linking Dialog

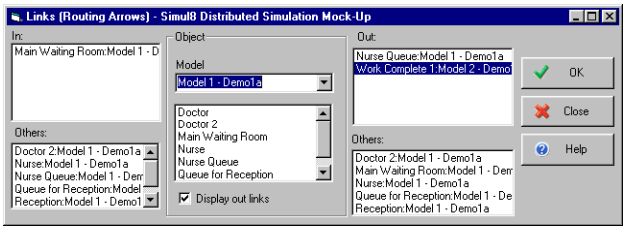

Figure 5: Modified Object Linking Dialog

Linking external objects

Once a model has been selected, external objects could then be used for specific distributions. Alternatively, an externalised variable from the model such as a published 'results' variable could be used to provide the input. Figure 6 shows an example of the dialog boxes to enable external distribution selection.

The main purpose of creating external distributions is to replace the commonly used stochastic distributions and provide 'real' input in the form of entity occurrences (as opposed to a statistically derived distribution). The input captured from interoperating models could then be used to define, after a number of experimentations, a distribution, which could be used within a single model. Further implementation could be considered to integrate the process with the 'optimisers', which are often provided in COTS simulation packages. This could provide a mechanism by which experiments could be automated from which a set of distributions could be derived from interoperating models.

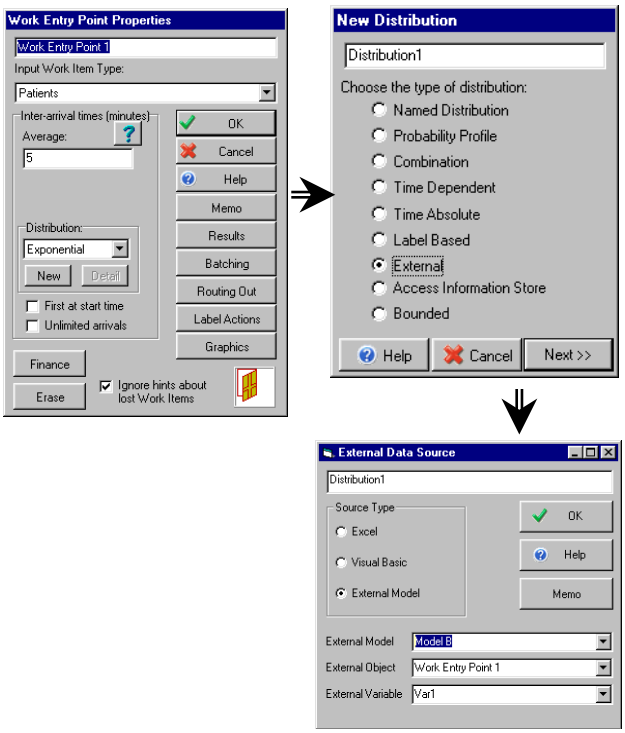

Figure 6: Modified External Distribution Dialog

Setting Model to be the Master

The modified user interface shown in figure 7 reveals an additional menu option to set the current model to be the master controller for all linked models. This functionality could provide 'central' control for all interoperating models, such as synchronized start and stop.

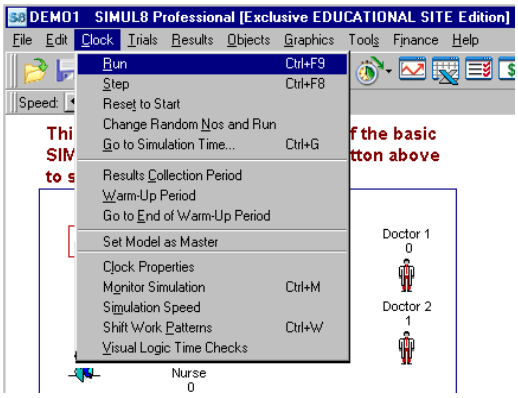

Figure 7: Modified Clock Menu

CONCLUSIONS AND SUGGESTED AREAS FOR FURTHER RESEARCH

COTS simulation packages tend to be designed for use on a single disparate simulation model. This paper has pointed to some areas, which could be addressed to provide further integration of interoperability (i.e. distributed simulation) functionality.

The two key areas discussed are functionality and interface design. Of-course many packages include functionality for the reading and writing of variable data to a specified application, such as a spreadsheet package. Some packages like SIMUL8 also include the facility to pass data directly into a program developed by the simulation engineer Functionality is being addressed already, in packages like SIMUL8, where an API is currently in development to facilitate model interoperation. However, the level of integration required to enable any reasonable amount of distributed theory integration, such as synchronisation, still requires much work on the behalf of the software companies.

Although some development is underway on the functionality of individual COTS simulation packages, further research is required to determine how model interoperation can be standardised between heterogeneous packages. Even once this standardisation work has taken place a body of research, in parallel, could well be required to investigate the tools and methodologies required by the simulation engineer to aid the development of large models within a team. This has been eluded to in more recent years, see Hibino et al. 2002. Concurrent development of a simulation model would require a tool set and methodologies similar to that used by software engineers. i.e. source code control (or model control) and version control. Further, the paradigm could be extended to include specific development tools for the simulation modeller, for example, determining the best partition points within a simulation – this could be calculated through experimentation, possibly an extension to the simulation optimising tools currently available. It also believed that

the paradigm could be extended to include specific methodologies and practices for use in large model development, in much the same way that project management and systems management methodologies are used in large IT developments (such as PRINCE or SSADM). Extensions to existing software development tools could also be investigated, such as UML (the Unified Modelling Language), to include a standardised set of development stages and model definition.

AUTHOR BIOGRAPHIES

MICHAEL D. RYDE is a Ph.D. student at the Department of Information Systems and Computing, Brunel University in the United Kingdom. He also received his M.Sc. at Brunel University in 2000 and is a member of the university's Centre for Applied Simulation Modelling (CASM).

SIMON J.E. TAYLOR is the Chair of the Simulation Study Group of the UK Operational Research Society and the collaborative simulation-modelling forum, the GROUPSIM Network (www.groupsim.com). He is a Senior Lecturer in the Department of Information Systems and Computing and is a member of the Centre for Applied Simulation Modelling, both at Brunel University, UK. With Dr Gary Tan of the School of Computing, National University of Singapore he is joint leader of the UK (EPSRC)/Singapore (DSTR)-funded BRUNUSIM distributed simulation research programme. He has an undergraduate degree in Industrial Studies (Sheffield Hallam), a M.Sc. in Computing Studies (Sheffield Hallam) and a Ph.D. in Parallel and Distributed Simulation (Leeds Metropolitan). His main research interest is collaborative simulation modelling. He is also a member of the London-based Purple Theatre Company.

REFERENCES

Boer, C.A. and Verbraeck, A. 2002. Connecting High level Distributed Simulation Architectures: An Approach for a FAMAS-HLA Bridge. In *Proceedings of the 14th European Simulation Symposium*. . Society for Computer Simulation Publishing House, Erlangen, Germany. 398–405.

Brooks, R.J. and Robinson, S. 2001. *Simulation*. Palgrave, Hampshire, UK 32-35.

Carothers, C.D., Fujimoto R. M., Lin, Y., and England P. 1994. Distributed simulation of large-scale PCS networks. MASCOTS 1994.

Clarke, R. 1993. Module interconnection frameworks for a real-time spreadsheet. Computer Abstracts International Database, reference: 39_1890.

Fujimoto R.M. 1990. Discreet event simulation; Communications of the ACM. Vol. 33, No 10.

Fujimoto R.M. 1999. Parallel and distributed simulation; In *Proceedings of the 1999 Winter Simulation Conferenc*e, P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds: 122-131.

McLean, C. and Riddick, F. 2000. The IMS Mission Architecture for Distributed Manufacturing Simulation. In *Proceedings of the 2000 Winter Simulation Conferenc*e, J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds. Association for Computing Machinery Press, New York, NY. 1539-1548.

Miller J, Fishwick P.A., Taylor S.J.E., Benjamin, B. and Szymanski, B. 2001. Research and Commercial Opportunities in Web-Based Simulation. Simulation: Practice and Theory. 9(1-2), pp. 55-72.

Pidd M., 2002. Simulation Software and Model Reuse: A Polemic. *Proceedings of the 2002 Winter Simulation Conference*, E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds. Association for Computing Machinery Press, New York, NY. 772-775.

Strassburger, S. 2001. Distributed Simulation Based on the High Level Architecture in Civilian Application Domains. Society for Computer Simulation Publishing House, Erlangen, Germany.

Sudra, R., Taylor, S. J. E. and Tharumasegaram, J. 2000. Distributed Supply Chain Simulation in GRIDS. In *Proceedings of the 2000 Winter Simulation Conferenc*e, J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds. Association for Computing Machinery Press, New York, NY. 356-361.

Taylor, S. J. E., Bruzzone, A., Fujimoto, R., Boon Ping Gan, Straßburger, S. and Paul, R.J. 2002. Distributed Simulation and Industry: Potentials and Pitfalls. *Proceedings of the 2002 Winter Simulation Conference*, E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds. Association for Computing Machinery Press, New York, NY. 688-694.

Zeigler B.P., Kim, D. and Buckley, S. J. 1999. Distributed supply chain simulation in a DEVS/CORBA execution environment; In *Proceedings of the 1999 Winter Simulation Conferenc*e, P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds: 1333-1340.