Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.

SIMULATION INTEROPERABILITY WITH THE MICRO SAINT SIMULATION SOFTWARE AND COM SERVICES

Daniel W. Schunk
Wendy K. Bloechle

Micro Analysis and Design, Inc.
4900 Pearl East Circle
Boulder, CO 80301, U.S.A.

ABSTRACT

In today's high tech world the need for interoperability among programs has never been more necessary. If a user were able to utilize different programs' strengths in unison, then the ability for programs to work together would greatly expand current software's ability to analyze. In response to this request, COM Services was added to the most recent release of Micro Saint. This paper will feature an example of how to apply interoperability to the Micro Saint simulation software as well as present some examples of how to further utilize COM Services.

1 INTRODUCTION

Communication has become key to the successful operation of business, military and health care systems. In particular, companies see the value of having interoperability between various software programs. However, in many cases, the information cannot be transferred from one software program to another.

Recognizing this emerging need for inter-model communication, COM Services was developed as a new feature for Micro Saint. One of the essential needs that has been identified in the simulation community is the ability for different models, developed by different organizations, possibly using different modeling tools to be able to communicate. In general, there are two types of communication that might be sought:

1. *Dynamic data exchange during simulation runs* whereby, one simulation relies upon another simulation federate to provide data during the simulation. This concept is illustrated in Figure 1 where different military simulation objects reflecting different systems (e.g., tanks, and airplanes) interact in order to form a larger simulation of the battlefield.
2. *Sharing data between simulation runs* through a central data repository. For example, Figure 2 illustrates how, during system requirements definition and development, some models' outputs may serve as other models' inputs.

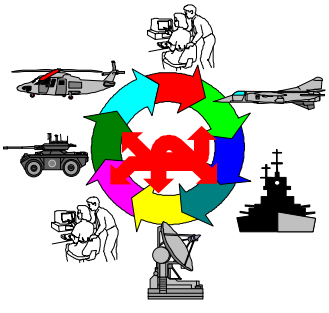

Figure 1: An Example of Dynamic Interaction of Simulation Components

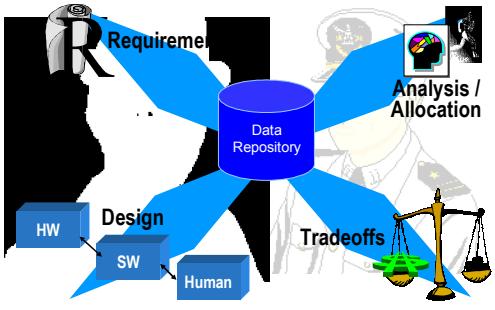

Figure 2: An Example of Simulation Model Data Exchange Needs

Micro Saint has recognized the need for communication in the simulation market and has taken the steps to ensure interoperability using COM Services. COM Services was developed using the Component Object Model (COM) architecture, COM allows applications and systems to be built from components supplied by different software vendors. COM is programming language independent and al-

lows more than one application to send information to another application. These capabilities are part of the reason it was chosen for use with Micro Saint. Additionally, COM is the most widely used object model for developing distributed and concurrent systems.

Using Visual Basic, Visual C++, Borland C++ or some other programming language as the middleware between Micro Saint and another application is the key to setting up interoperability. This sharing of information makes solutions more accurate and also saves needed time in any project. More accurate results mean better models being built and better data collection. This could indirectly yield higher profits, or more efficient system designs. For example, Figure 3 presents the middleware concept for the integration of Micro Saint into HLA-compliant environments.

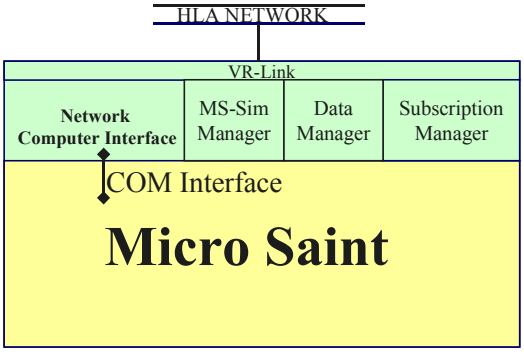

Figure 3: The Middleware Concept for HLA-compliant Simulations using Micro Saint COM Services

Along with the interaction between software applications, COM also allows Micro Saint and the user to interact in real time. Changes can be made to the simulation model as the model is running, such as users can changing variable values while the model is running.

2 COM SERVICE CAPABILITIES AND EXAMPLE

COM Services allows communication between Micro Saint and other software applications possible. Included with COM Services are command line capabilities that will allow users to start, stop, and continue the model. In addition, model control allows the user to pause, halt and abort the model through parsed expressions.

Data exchange allows users to pass variable values into and out of Micro Saint. Control of the event queue allows users to insert scenario events into the event queue at specified times in the future. In addition, users can receive event queue information from Micro Saint while the model is running. Lastly, COM Services allows Micro Saint to send messages to the user when a model has ended or if errors have occurred.

3 USING COM SERVICES TO ANALYZE AN EMERGENCY ROOM

3.1 Problem Explanation

In this example a hospital's emergency room is being analyzed. The problem consists of how will dynamically allocating the number of doctors on staff affect the number of patients waiting for treatment and doctor utilization.

3.2 Model Explanation

The model being used for this analysis is a discrete event simulation model developed using the Micro Saint simulation software. The model is based upon the hospital's emergency room in which patients arrive, and based on triage, the patient will either immediately receive treatment or will go to registration. From treatment the patient may have to undergo tests and then will either be discharged or admitted to the hospital. Micro Saint has the option of setting variables to be "external". External variables are the variables that Micro Saint will send to Visual Basic when a change occurs in that variable's value.

3.3 Visual Basic Program Development

3.3.1 Develop New ActiveX .exe Project

In order to use COM Services, when building a new Visual Basic project, the project must be an ActiveX .exe. ActiveX controls allows Visual Basic programs to interact with other outside programs. The key element of an ActiveX program is the *class module*. The class module allows Micro Saint to call commands in the Visual Basic Program. The user should also name the program "TestCOM".

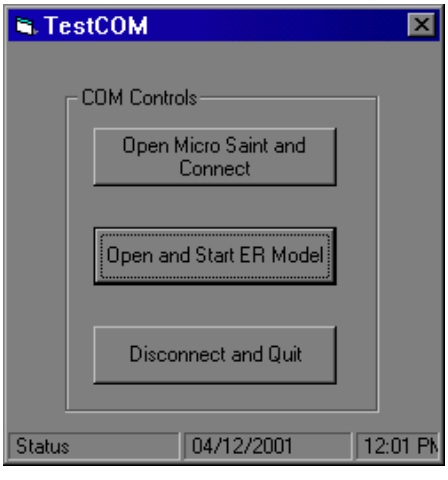

Figure 4: Form frmMain

3.3.2 Interface Design

The interface for this Visual Basic program will consist of three forms. The first form performs the basic COM commands consisting of connecting to Micro Saint, loading the emergency room model into Micro Saint and starting the model. It will consist of three buttons which will perform the previously mentioned functions (See Figure 4). The second form will inform the user of some model statistics (i.e. how many doctors are on staff, doctor utilization, patients waiting, etc.) as well as ask the user whether to increment the number of doctors on staff, decrement the number of doctors on staff, or continue the model with no change (See Figure 5). The final form will be a report (See Figure 6) in which Visual Basic will show the user what effect their changes had on the model.

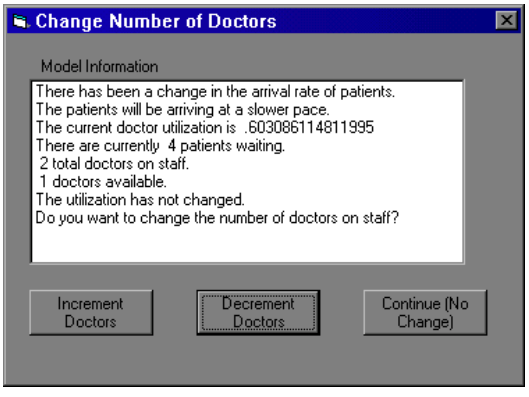

Figure 5: Form frmQuestion

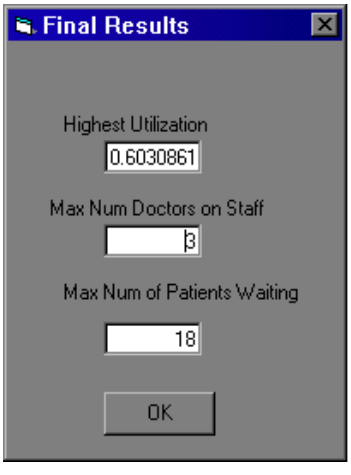

Figure 6: Form frmReport

These three forms will start to use the COM commands. These commands and their definitions are:

- **saint Connect(char *comProgramName)** – Connects to Micro Saint (This is the COM Program name. Note: In the test programs it is ReceiveCOMMessages)
- **saint InsertIntoQueue(char *expr, short firstOrLastAtTime, float timeOffset, float eventTime = 0.0)** – Inserts an event into the event queue (The first part is the expression you are sending to Micro Saint, the second part is deciding whether you want it to go at the beginning of the event queue or at the end, how much time (if any) to offset it by, and the time you want this event to happen. Note: If there is an offset time then it will NOT happen at the time you put in.)

To develop this part of the model the user must place all the necessary parts of the interface on the form as well as give the forms the appropriate names (frmMain, frmQuestion). The next part of this example is to input coding into the forms. For the form frmMain, the user needs to insert code so that this form can connect to Micro Saint, load up the appropriate simulation model and start it, and disconnect from Micro Saint. This can be done through a series of commands that Micro Saint understands through COM. The coding for the aforementioned actions are as follows:

1. Connect to Micro Saint
```
Set saint = CreateOb-
ject("saint.Application.1")
saint.Connect("TestCOM.ReceiveCOMMessages"
AppActivate "TestCOM"
```
2. Load up the appropriate simulation model and execute it
```
saint.OpenModel (whole)
saint.ExecuteModel (whole)
```
3. Disconnect from Micro Saint
```
saint.Disconnect
saint.Quit
```

For the form frmQuestion, the program will have to be able to send commands to Micro Saint that will either increment or decrement the current and total number of documents on staff as well as updating variables. The coding for these tasks are as follows:

1. Incrementing number of doctors on staff
```
Call saint.InsertIntoQueue("doctors += 1;
maxdocs += 1;", 1, 0, 10)
```
2. Decrementing number of doctors on staff
```
Call saint.InsertIntoQueue("doctors -= 1;
maxdocs -= 1;", 1, 0, curtime)
```
3. Update variable
```
oldutil = docutil
newval = 0
Call saint.InsertIntoQueue("docutil :=
0;", 1, 0, curtime)
frmQuestion.Hide
```

3.3.3 Class Module Design

The class module (ReceiveCOMMessages) is what enables Micro Saint to send information over to the Visual Basic program. There are two main functions in the class module that are important to the user. These functions and their definitions are ReceiveIntVariable and ReceiveFloatVariable. When Micro Saint calls these functions, it passes to Visual Basic the name of the variable and the variable's new value. Using this information, Visual Basic will record (internally) the variable's value and prompt the user to make a choice if necessary.

The Code for the ReceiveIntVariable function is as follows:

```
Public Function ReceiveIntVari-
    able(ReceiveVariable As String, number As
    Long) As Long
If ReceiveVariable = "doctors" Then
    curdoc = number
ElseIf ReceiveVariable = "maxdocs" Then
    maxdoc = number
ElseIf ReceiveVariable = "day" Then
    curday = number
ElseIf ReceiveVariable = "numpatients" Then
    curpatients = number
End If
End Function
```

Essentially this function looks at what variable is being passed and assigns it to the appropriate Visual Basic variable.

The next function is the ReceiveFloatVariable function, it operates in a similar fashion as the ReceiveIntVariable except that if the variable is the variable representing the arrival rate, then Visual Basic will prompt the user on the doctor staffing.

```
Public Function
ReceiveFloatVariable(ReceiveVariable As String,
number As Double) As Long
Dim change As String
If ReceiveVariable = "meanrate" Then
    meanrt = number
        If oldrate > 0 Then
            If oldutil > 0 Then
                utilchange = oldutil - docutil
            End If
            If utilchange < 0 Then
                change = "The utilization has in-
creased from the last decision"
            ElseIf utilchange > 0 Then
                change = "The utilization has de-
creased from the last decision"
            Else
                change = "The utilization has not
changed."
            End If
            Call saint.PauseModel
            frmQuestion.Quest.Clear
            If oldrate < meanrt Then
                frmQuestion.Quest.AddItem "There
has been a change in the arrival rate of pa-
tients."
                frmQuestion.Quest.AddItem "The
patients will be arriving at a slower pace."
                frmQuestion.Quest.AddItem "The
current doctor utilization is " + Str(docutil)
                frmQuestion.Quest.AddItem "There
are currently " + Str(curpatients) + " patients
waiting."
                frmQuestion.Quest.AddItem
Str(maxdoc) + " total doctors on staff. "
                frmQuestion.Quest.AddItem
Str(curdoc) + " doctors available. "
                frmQuestion.Quest.AddItem change
                frmQuestion.Quest.AddItem "Do you
want to change the number of doctors on staff?"
            Else
                frmQuestion.Quest.AddItem "There
has been a change in the arrival rate of pa-
tients."
                frmQuestion.Quest.AddItem "The
patients will be arriving at a faster pace."
                frmQuestion.Quest.AddItem "The
current doctor utilization is " + Str(docutil)
                frmQuestion.Quest.AddItem "There
are currently " + Str(curpatients) + " patients
waiting."
                frmQuestion.Quest.AddItem
Str(maxdoc) + " total doctors on staff. "
                frmQuestion.Quest.AddItem
Str(curdoc) + " doctors available. "
                frmQuestion.Quest.AddItem change
                frmQuestion.Quest.AddItem "Do you
want to change the number of doctors on staff?"
            End If
            frmQuestion.time = curtime
            frmQuestion.Show 1
            Call saint.ExecuteModel(App.path +
"\er.mod")
            oldrate = meanrt
        Else
            oldrate = meanrt
        End If
ElseIf ReceiveVariable = "doctorut" Then
    docutil = number
ElseIf ReceiveVariable = "clock" Then
    curtime = number
End If
End Function
```

This function will show the user the form frmQuestion if it is necessary as well as perform some simple statistical calculation in order to help the user make a decision on staffing.

3.4 Finished Example

With this finished example, the user can expand the analysis to include all aspects of the emergency room including nurses and other staff members. This is a way for the user to run through certain analyses and play "what if" without having to commit resources.

4 OTHER EXAMPLE OF COM SERVICES APPLICATION

COM Services' are not just limited to the previous example. COM Services versatility allow it to be applied to many different types of analyses in many different types of industry. The following are two more ways that COM could be used effectively.

4.1 Course of Action Training Tool (COATT)

Good pilot judgment and decision making are critical to safe flight operations. A recent study of helicopter accidents, as demonstrated in Reference 1, investigated the NTSB database from 1990-1996 and analyzed 1165 accidents. Pilot decision making was found to be a factor in 10% of all accidents and 15% of all fatal accidents in this database. This is also borne out in incident data. The Aviation Safety Reporting System (ASRS) performed an analysis, of 833 helicopter incidents that occurred from 1989 through 1996. This study found that 362 of the incidents or 43% had pilot judgment as a major issue. However, judgment and decision making are not a formal part of the initial rotary wing training, nor recurrent training.

The emergency medical service (EMS) industry is especially susceptible to this problem. The current project was undertaken to develop a judgement/ decision making trainer for EMS pilots that will help them to evaluate the changing conditions and recognize the available courses of action and their utility.

The technical approach for providing mission training for helicopter pilots on the dynamic conditions and alternative courses of action that occur is a combination of computer-based simulation, full motion video, still photography, and audio. These techniques will be accessed, driven, and displayed by a Windows graphical user interface (GUI). This combination of technologies provides helicopter pilots with a low cost simulator named the Course of Action Training Tool (COATT) for investigating and learning about the potential impacts of the decisions they can make during a mission scenario. The beta test version of the first mission scenario has been completed.

4.1.1 Scenario Development

To identify a helicopter scenario that included a rich set of dynamically changing conditions, 17 EMS pilots were interviewed using a semi-structured questionnaire containing 30 questions, most of which were open-ended. The questions encouraged pilots to describe situations where they had to make difficult decisions and the circumstances surrounding those decisions.

A content analysis of the interviews revealed two problematic areas for decision making — the weather and the conditions at sites where they have to land. They cited weather-related decisions as among the most important pre-flight, in-flight, and stressful decisions they make; judging safe landing zone conditions were important in-flight decisions and were also rated as hard to make.

Based on these findings, a weather-related scene pick-up mission was developed as the training scenario. In this scenario, the trainee has to pick up a patient at an accident scene and transport him to a designated hospital. The scenario contains potentially changing weather, radio and crew communication, possibly equipment malfunctions, and varying terrain.

4.1.2 The Simulation Model

After defining mission scenarios, network flow diagrams were developed to depict all of the alternative processes that could occur in the execution of the mission. Using the network diagrams as blueprints, a discrete event simulation model was developed using the Micro Saint simulation tool. These simulations include rule-based and probabilistic logic for the stochastic portions of the simulations such as weather conditions. However, the alternative course of action (COA) choices that are made by the pilot are not programmed into the simulation. These decisions are made by pilots as they interact directly with the simulation. Each node in a network diagram represents either an action that is made by the pilot or an indication of the current environmental conditions. Figure 7 illustrates an example of a network diagram.

The user interface presents pilots with combinations of full-motion video clips, still photography, audio, and text to depict the activities that are occurring in the simulation model and the options that the pilot has as the simulation progresses. At points throughout the mission, pilots are presented with an update of their situation. At these points, they are given different options, such as:

- Continuing the mission
- Aborting the mission and returning to base
- Taking an alternate route
- Landing immediately

Feedback is provided by COATT to offer pilots a review and critique of the decisions they made during the mission and different courses of action they might have chosen to improve mission safety. This feedback was developed to be consistent with the training that pilots receive and with what is known about the types of scenarios and accidents used in COATT.

4.1.3 Use of COM to Make Simulation Interactive

Communication between the underlying Micro Saint simulation model and the GUI application is accomplished using COM Services. Micro Saint COM Services allows pilots to send commands and data from software that is external to Micro Saint. The COATT application uses Visual Basic as the external COM compliant software for the GUI development.

When a user starts COATT, he or she selects a training scenario or a previously run mission to replay. The controlling GUI application then launches the Micro Saint simulation model. As a node in the model executes, Micro Saint sends a message to the GUI application to play a designated video clip. When the video clip completes, the controlling application sends a message to Micro Saint to execute the next node. Due to the stochastic nature of the model, different nodes execute based on the probabilities associated with each one. As each node executes it sends a message to the GUI application to play a different video clip or audio segment.

When the simulation encounters a pilot course of action decision point, as shown in node 6 of Figure 7, the simulation model sends a message to the controlling application to display a pilot decision window. This window presents the user with a text message describing the current situation and provides choices for the course of action he or she can take.

When the user makes a decision, the controlling application sends a message to the Micro Saint model to execute the appropriate node, which in turn sends a message back to the GUI to play the next video or audio segment. This process of displaying video and audio segments that represent what is happening in the underlying simulation model and periodically asking the user to make a COA decision continues until the simulation model completes. The use of COM functionality to develop this interactive discrete event simulation combined with multi-media presentation has been demonstrated in this project to be technically feasible and relatively low cost when compared to other types of man-in-the-loop simulators.

4.2 Combat Automation Requirements Testbed (CART)

CART was developed under the Air Force Research Laboratory (AFRL) research development and acquisition of human/system design tools through the Human Effectiveness Directorate. The CART simulation environment consists of a High Level Architecture compliant federation of simulations (or federates). Within this simulation environment there is the CART Human Performance Modeling (HPM) Environment (CHE). The CHE modeling and simulation software is an expansion of the Army Research Laboratory (ARL) Human Research and Engineering Directorate (HRED) which developed the Improved Performance Research Integration Tool (IMPRINT).

IMPRINT provides the means for estimating manpower, personnel, and training requirements and constraints for new weapon systems early in the acquisition process. Although there were many subtle enhancements and additions made to IMPRINT for CART, the two major enhancements were as follows:

1. The addition of goal orientation modeling capability.
2. The addition of an adaptive simulation interoperability environment to allow CART models to communicate with other simulations through an HLA RTI.

4.2.1 HLA "Middleware" Scheme

CART simulation models act as a federate within a High Level Architecture (HLA) compliant federation. In order for this interaction to occur, certain "middleware" was introduced into the CART HPM Environment. The data sent across the federation, object attributes and interactions, was mapped to the Real-time Platform Reference Federation Object Model (RPR FOM).

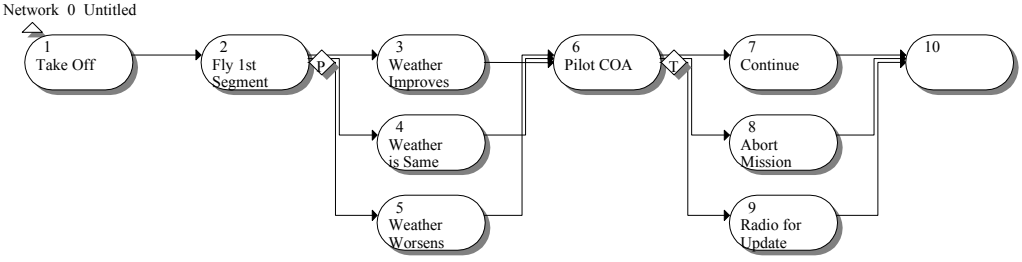

Figure 7: Network Diagram

The components of the CART "middleware" are shown in Figure 8. User-defined external variables are the interface between CART models and other HLA federates prior to and during federation run-time. During federation run-time, COM Services are used to communicate between the "middleware" and the Micro Saint Run-Time Engine (MSRTE) CART models. Mäk Technologies' VR-Link product provides an interface to the RTI. The "middleware" interface allows the CART user to map external variables to RPR FOM interaction parameters prior to run-time. Run-Time "middleware" (RT MW) Code allows the RTI with VR-Link to interact with COM during run-time.

In addition to the architecture described above, the two keys to creating CART's adaptive simulation interoperability environment were the external variable concept and the use of the RPR FOM. The Simulation Management (SIMAN) Interactions in the RPR FOM provided more flexibility for the CART user.

The CART GUI publishes an external variable list. By designating some variables as "external" in the CART model, a user sets up the model to interact with other members of a CART federation. Prior to federation run-time, these external variables are mapped to HLA RPR FOM interaction parameters. They are also mapped to actions to be taken (whether the CART model wants to "receive" or "send" the data represented) when the variables are encountered during federation run-time. During federation run-time, when these variables are used in a CART model or associated RPR FOM data is sent in through the "middleware" by another simulation, their values update accordingly and a simulation action (such as suspending a task or starting a goal network) may be triggered.

5 SUMMARY

The future growth of the simulation market will hinge largely on our ability to build models that can communicate. We must be able to build models that we know others can take advantage of, not ones that have a limited shelf life of only solving one problem. As different models can share data, the power of simulation during engineering and design will be greatly increased. COM is a start to the basic architecture that will support that strategy.

AUTHOR BIOGRAPHIES

DANIEL W. SCHUNK is an industrial engineer for Micro Analysis and Design, Inc. He has a Bachelor of Science in Industrial Engineering from Purdue University. His email address is dschunk@maad.com.

WENDY K. BLOECHLE is the Director of Sales and Marketing for Micro Analysis and Design, Inc. She has a Bachelor's of Science in Industrial Engineering from the University of Illinois and a Master of Business Administration from the University of Colorado. Her email is wbloechle@maad.com.