Proceedings of the 1999 Winter Simulation Conference
P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds.

SIMPHONY: AN ENVIRONMENT FOR BUILDING SPECIAL PURPOSE CONSTRUCTION SIMULATION TOOLS

Dany Hajjar
Simaan AbouRizk

Department of Civil Engineering
220 Civil Electrical Building
University of Alberta
Edmonton CANADA T6G 2G7

ABSTRACT

Special Purpose Simulation (SPS) is a proven principle that can lead to the effective transfer of simulation knowledge to the construction industry. Three separate industry experiments have led to the identification of a set of requirements that construction SPS tools should adhere to in order to be successful. This set of requirements was then used in the implementation of a computer system called Simphony. The system greatly simplifies the SPS tool development process and standardizes the simulation, modeling, analysis and integration features of such tools. The result is a complete environment that tailors to the needs of both novice and advanced simulation tool developers and users.

1 INTRODUCTION

Computer simulation is a proven technique for the planning of construction projects. Its effective use within the industry is best done through specialization and customization of the modeling, analysis and reporting components of the simulation systems. This philosophy is the basis for the "special purpose simulation" (SPS) methodology (AbouRizk and Hajjar 1998). SPS was followed in the successful development and implementation of several custom simulation tools in an industry setting.

Although each tool was successfully implemented, it was observed that the relatively large initial investment required for their development would hinder the application of the SPS-based approach to other construction operations. Further, several limitations of these tools were identified and used to obtain a comprehensive list of requirements that such tools must possess. This list was used as the basis for the design and implementation of a complete construction simulation tool development and utilization environment called Simphony.

This paper will begin with an overview of the individual tools developed. The next section will then introduce the Simphony environment including its overall system structure, features, and related computer programs. This is followed by a sample session to demonstrate the tool development process.

In order to validate and test the flexibility and effectiveness of Simphony, a tool that allows for the construction of models using the CYCLONE methodology (Halpin 1977) was created. The Simphony-based development effort was then compared with the effort applied in the development of a standalone CYCLONE simulator.

2 SPS TOOLS

Three standalone construction simulation tools were developed with the objective of transferring simulation knowledge to the industry. The first tool, called AP2-Earth (Hajjar and AbouRizk 1996), allows for the analysis of large earth moving projects. The second tool, called CRUISER (Hajjar and AbouRizk 1998), can be used for modeling aggregate production plants. And the third tool was CSD (Hajjar, AbouRizk, and Xu 1998) which allows for the optimization of construction site dewatering operations.

The success and limitation of these tools helped in the identification of a broad set of requirements that simulation models and tools must posses in order to be successfully applied in the construction industry:

1. The user interface should support graphical representation and manipulation of the model structure. Integrity violations should be trapped as soon as possible and reported to the user through the modeling interface in a helpful fashion. Graphical modeling support should be the primary means of model

definition and manipulation. However, advanced users should still be accommodated and allowed to bypass the graphical system.

2. The modeling process should be done in a manner that is natural and relevant to the specific target domain of the simulation tool. Users should not be exposed to the abstract underlying constructs, which require expertise with fundamental simulation concepts.
3. Construction methods vary in complexity and as a result the simulation tools must be able to accommodate different types of simulation processors.
4. Results generated by the simulation tools should be of immediate relevance to the target user. Specific post simulation analysis should be performed when required with results presented in a familiar and natural manner.
5. Simulation tools must be able to integrate with existing systems in order to reduce the data entry requirements as well as to generate information for use by existing systems. The generated information from each tool should follow a standard structure in order to simplify its analysis process by external systems.
6. Users should be able to combine models based on several tools in an effective manner in order to allow for the modeling of complete projects involving multiple construction methods.
7. Tools should support and even encourage the reusability of exiting simulation models.
8. Tool developers should be able to create new tools in a relatively short time with minimal effort. The tool development process should be standardized in order to provide inherent support for all the previously mentioned requirements.

A simulation tool development and utilization environment called Simphony was implemented based on this set of requirements.

3 SIMPHONY OVERVIEW

Simphony is a Microsoft Windows based computer system developed with the objective of providing a standard, consistent and intelligent environment for both the development as well as the utilization of construction SPS tools.

Tool developers can use Simphony to implement highly flexible simulation tools that support graphical, hierarchical, modular and integrated modeling with great ease.

Tool users have access to a single program that allows them to build simulation models in an intuitive and user-friendly manner. Results can be viewed as part of the graphical user interface or exported for use by external systems such as estimating and scheduling programs.

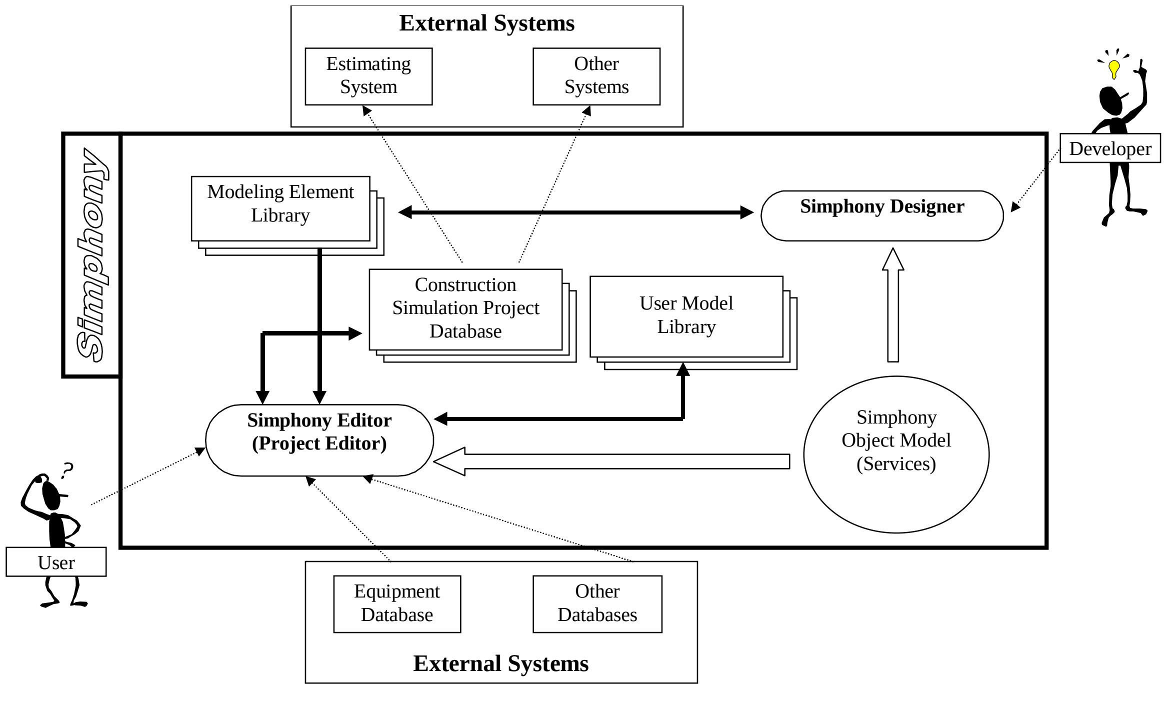


Figure 1: Simphony Environment

3.1 System Architecture

A high level depiction of the overall environment is presented in Figure 1. Developers utilize the *Simphony Editor* program to create SPS tools called "SPS Templates". SPS templates are a collection of modeling elements targeted for a single domain. Developed templates are stored in the *Modeling Element Library*. Users utilize the *Simphony Editor* to create simulation models based on the existing SPS templates in the *Modeling Element Library*. Constructed simulation models and their simulated results are stored in and retrieved from the *Construction Simulation Project Database*. This is a relational database management system that other systems can access to extract the desired information. The *User Model Library* is another such database that is utilized by users to store and retrieve reusable simulation models.

3.2 Features

Simphony provides a highly flexible yet user friendly environment for the simulation modeling process including support for:

1. Modular and hierarchical modeling for the representation of complex and large construction projects,
2. Both general purpose modeling constructs (eg. process interaction and CYCLONE) as well as specialized templates for specific construction methods (eg. earth-moving and aggregate production),
3. Extension of specialized SPS tools through the construction of models based on several templates (eg. a model based on an earth-moving template as well as a CYCLONE template),
4. Generation of custom output results in the form of tables and graphs,
5. Automated generation of externally accessible project planning data in a standard format,
6. Script based modeling for accommodation of advanced users wishing to bypass the graphical user interface, and
7. Storage and retrieval of commonly used simulation model structures in the *User Model Library*.

The tool development component of Simphony consists of a comprehensive and complete tool definition, compilation and testing platform. It is tailored for the development of construction SPS tools in that it provides all standard and commonly used SPS structures and routines in the form of libraries (called services) which are easily accessible by the developers. A list of the available services and their function is provided in Table 1. Simphony also includes support for "User Services". These represent supplementary services which can be implemented as ActiveX libraries by expert developers and easily integrated into the Simphony development environment. Example applications may include CAD and neural network integration support services.

Table 1: List of Simphony Services and their Function

Service	Description
Simulation	Provides support for discrete-event simulation including event scheduling, next-event polling, implicit queuing, queues, and stacks. More advanced method are also available to support event cancellation and resource preemption.
Random Number Generation	Exposes a set of routines that return random numbers based on several stochastic distributions including normal, exponential and beta.
Tracing	Provides a general purpose mechanism for relaying information to the user which could include simulation trace results, error messages, and integrity errors. Simphony also uses this service internally to inform the user in case a run-time error is detected in the developer's code.
Statistical Analysis	Provides support for the collection and analysis of observations. Standard statistical results include average, standard deviation, minimum and maximum.
Planning	Allows for the generation of project plans. Information generated by the modeling elements is transferred directly to a relational database for processing by other applications.
Database Access	Allows developers to access external ODBC databases to retrieve or store information. Examples of such information includes standard equipment databases and accounting systems.
Graphical User Interface	Exposes a variety of functions that allow for the graphical representation of modeling elements including line and circle drawing, bitmap graphics, and text output. Other functions are also available for manipulating chart and grid type display objects.

The development process itself is simplified and reduced to the task of declaring a set of modeling elements and implementing their definition using a commonly known macro language.

3.3 SPS Template Development and the Simphony Designer

Creating new SPS templates involves the design and implementation of the modeling elements that will be used to create simulation models for a given domain. The design component is not a trivial task. It involves a complete understanding of the targeted construction domain itself and the fundamental principles of SPS modeling. Once the design is complete, implementing the template involves the creation of the required modeling elements through the *Simphony Designer*. Creating new modeling elements involves the customization of the behaviors of a supplied generic base modeling element. This, in turn, involves writing code in the form of event handlers in response to the various events. Simphony generates these events in response to user or system actions.

The generic base modeling element includes intelligence in the form of default implementations for most events. This allows developers to concentrate on the uniqueness of each element.

A simulation model becomes a collection of *instances* of modeling elements. Figure 2 illustrates this definition. Modeling element instances of the same type (i.e. based on the same modeling element) would share the same code and be differentiated solely by the value of their properties.

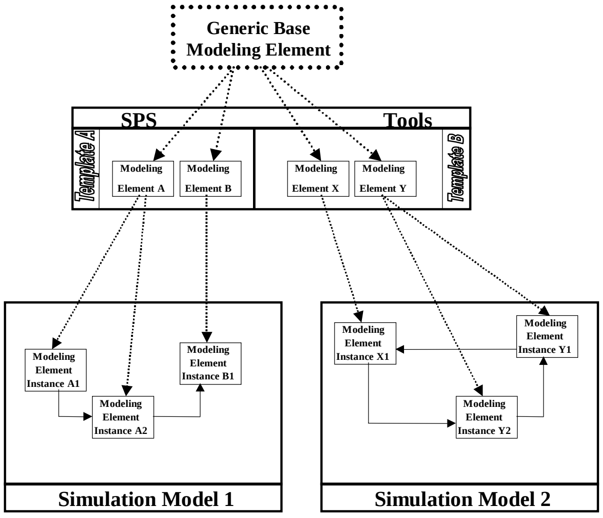

Generic Base Modeling Element
SPS
Tools
Template A
Template B
Modeling Element A
Modeling Element B
Modeling Element X
Modeling Element Y
Modeling Element Instance A1
Modeling Element Instance A2
Modeling Element Instance B1
Modeling Element Instance X1
Modeling Element Instance Y1
Modeling Element Instance Y2
Simulation Model 1
Simulation Model 2

Figure 2: Structure of Simulation Models in Simphony

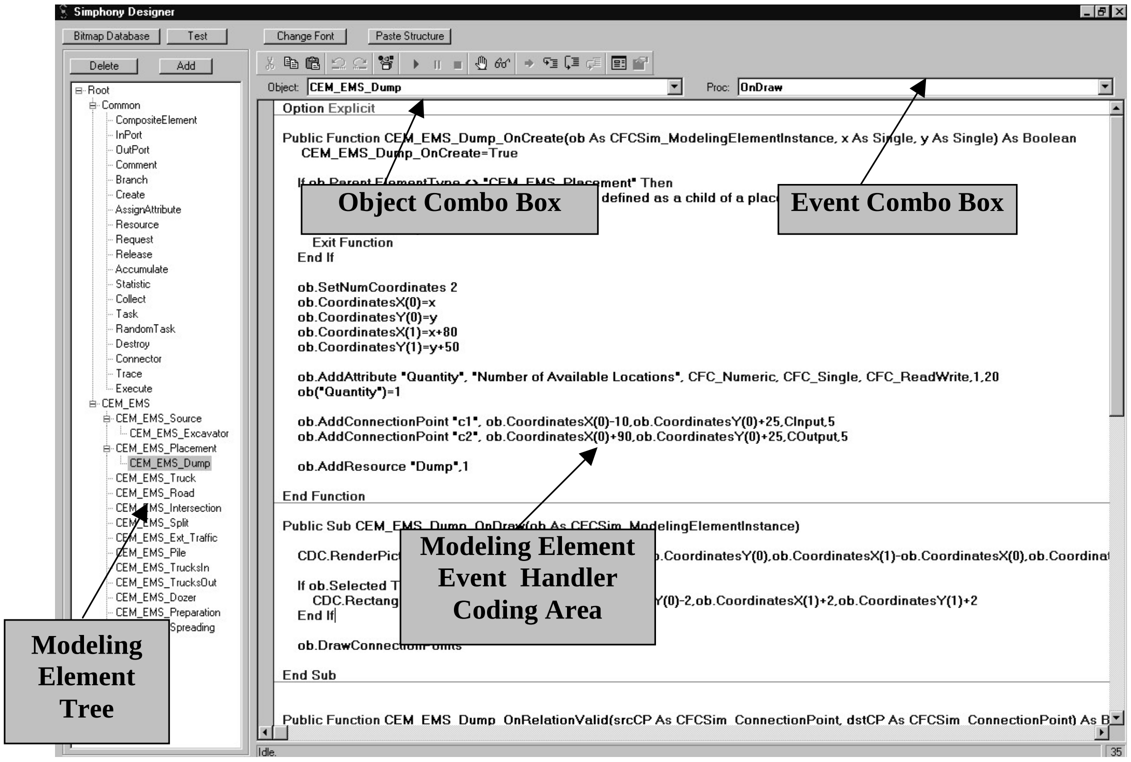
The Simphony Designer is similar to a typical integrated development interface in that it combines the processes of code entry, validation and compilation into a single program. Access to the Simphony services is also made available through this program. Developers define the element behaviors using a language based on Visual Basic for Applications (VBA). VBA was developed by Microsoft Corporation for use as macro-based extension language for its series of office productivity tools including Access, Excel and Word. The decision to use VBA was driven by the fact that numerous major developers now use it as the standard macro support language within their applications. This includes such products as AutoCAD and Primavera. The main form for the program is shown in Figure 3.

The modeling element tree displays a list of all currently defined modeling elements in the modeling element library. A tree view is used because the structure of the library is hierarchical. Clicking on a given modeling element from the tree displays its event handlers in the "Event handler Coding Area" and allows developers to change their definition or insert new event handlers.

3.4 Construction Simulation using the Simphony Editor

Simphony Editor is the computer system that allows users to create and execute simulation models based on the elements available in the modeling element library. The main form, shown in Figure 4, is based on a multiple document interface (MDI) standard. This allows multiple windows to be open at the same time in order to represent different views of the model.

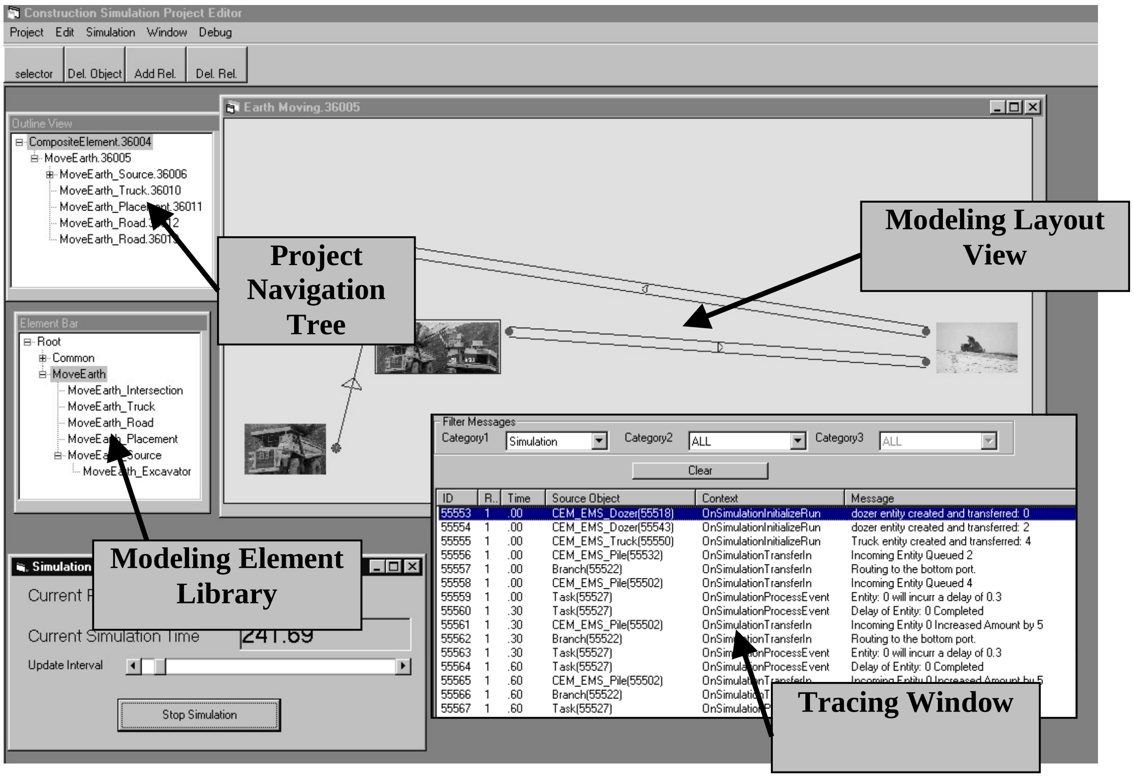
The modeling element library displays all available elements that can be used to construct new models. The model layout window displays the current set of defined elements at a certain level in the project hierarchy. Displayed elements can be selected, deleted, edited and linked. Viewing the contents of one of the displayed elements will open another layout window. The project navigation tree displays the structure of the simulation project. Double-clicking on an item on the tree will bring up a model layout window for the element represented by the tree entry. The trace window is used to display and filter generated messages. The tool bar displayed at the top of the main form is used to delete selected elements, add and delete relationships, and change the zoom setting for the active model layout window. Double-clicking on a given element instance displays the modeling element attribute dialog box which can be used to manipulate the parameters of individual modeling elements as well as view the statistical analysis results.



Object: CEM_EMS_Dump
Proc: OnDraw

```
Option Explicit

Public Function CEM_EMS_Dump_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single) As Boolean
    CEM_EMS_Dump_OnCreate=True

    If ob.ParentElementType <> "CEM_EMS_Placement" Then
                                        defined as a child of a plac

       Exit Function
    End If

    ob.SetNumCoordinates 2
    ob.CoordinatesX(0)=x
    ob.CoordinatesY(0)=y
    ob.CoordinatesX(1)=x+80
    ob.CoordinatesY(1)=y+50

    ob.AddAttribute "Quantity", "Number of Available Locations", CFC_Numeric, CFC_Single, CFC_ReadWrite,1,20
    ob("Quantity")=1

    ob.AddConnectionPoint "c1", ob.CoordinatesX(0)-10,ob.CoordinatesY(0)+25,CInput,5
    ob.AddConnectionPoint "c2", ob.CoordinatesX(0)+90,ob.CoordinatesY(0)+25,COutput,5

    ob.AddResource "Dump",1

End Function

Public Sub CEM_EMS_Dump_OnDraw(ob As CFCSim_ModelingElementInstance)

    CDC.RenderPic                       .CoordinatesY(0),ob.CoordinatesX(1)-ob.CoordinatesX(0),ob.Coordinat

    If ob.Selected T
       CDC.Rectang                      Y(0)-2,ob.CoordinatesX(1)+2,ob.CoordinatesY(1)+2
    End If

    ob.DrawConnec

End Sub

Public Function CEM_EMS_Dump_OnRelationValid(srcCP As CFCSim_ConnectionPoint, dstCP As CFCSim_ConnectionPoint) As B
```

Figure 3: Main Form of Simphony Designer



ID	R..	Time	Source Object	Context	Message
55553	1	.00	CEM_EMS_Dozer(55518)	OnSimulationInitializeRun	dozer entity created and transferred: 0
55554	1	.00	CEM_EMS_Dozer(55543)	OnSimulationInitializeRun	dozer entity created and transferred: 2
55555	1	.00	CEM_EMS_Truck(55550)	OnSimulationInitializeRun	Truck entity created and transferred: 4
55556	1	.00	CEM_EMS_Pile(55532)	OnSimulationTransferIn	Incoming Entity Queued 2
55557	1	.00	Branch(55522)	OnSimulationTransferIn	Routing to the bottom port.
55558	1	.00	CEM_EMS_Pile(55502)	OnSimulationTransferIn	Incoming Entity Queued 4
55559	1	.00	Task(55527)	OnSimulationProcessEvent	Entity: 0 will incurr a delay of 0.3
55560	1	.30	Task(55527)	OnSimulationProcessEvent	Delay of Entity: 0 Completed
55561	1	.30	CEM_EMS_Pile(55502)	OnSimulationTransferIn	Incoming Entity 0 Increased Amount by 5
55562	1	.30	Branch(55522)	OnSimulationTransferIn	Routing to the bottom port.
55563	1	.30	Task(55527)	OnSimulationProcessEvent	Entity: 0 will incurr a delay of 0.3
55564	1	.60	Task(55527)	OnSimulationProcessEvent	Delay of Entity: 0 Completed
55565	1	.60	CEM_EMS_Pile(55502)	OnSimulationTransferIn	Incoming Entity 0 Increased Amount by 5
55566	1	.60	Branch(55522)	OnSimulationT	
55567	1	.60	Task(55527)	OnSimulationP	

Figure 4: Simphony Editor Main Form

4 SAMPLE TOOL DEVELOPMENT

An example will now be provided to illustrate how Simphony can be used to create a simple modeling element. The new element will allow a plant owner to model a simple concrete batch plant operation. Incoming trucks arrive at the plant site and wait for their turn. When the mixer is available, the next truck in the queue loads a certain quantity of concrete and then proceeds to its destination.

The Simphony Designer program is first used to create a new element called "Batch_Plant". Next, the simulation behavior of the new element is defined to model the described scenario.

4.1 Step 1: Building a Working Model

Simulation model entities representing the trucks will be needed. So will a resource to represent the mixer location. The discrete simulation events will be: *TruckArrive*, *RequestMixer*, and *ReleaseMixer*. The mixer resource is declared by implementing a handler for the *OnCreate* event as follows:

```
Public Function Batch_Plant_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single) As Boolean
    Batch_Plant_OnCreate = True
    ob.OnCreate x,y,True

    ob.AddResource "Mixer",1
End Function
```

The first line inside the handler sets the return value to True to instruct Simphony to create the element. If for some reason the element should not be created, the return value can be set to False. The second line calls the default implementation of the event handler. The third line declares the mixer resource with an initial quantity of one. Next, the needed events are declared as part of the developer's implementation of the *OnSimulationInitialize* event handler:

```
Public Sub Batch_Plant_OnSimulationInitialize(ob As CFCSim_ModelingElementInstance)
    ob.AddEvent "TruckArrive", True
    ob.AddEvent "RequestMixer"
    ob.AddEvent "ReleaseMixer"
End Sub
```

A starting truck will need to be initialized at the beginning of the simulation. At each truck's arrival, the arrival of the following truck is scheduled. In order to initialize the first truck in the model, the *OnSimulationInitializeRun* event, which is triggered at the beginning of each simulation run, is implemented as follows:

```
Public Sub Batch_Plant_OnSimulationInitializeRun (ob As CFCSim_ModelingElementInstance, RunNum As Integer)
  Dim truck As CFCSim_Entity
  Set truck = ob.AddEntity

  ob.ScheduleEvent truck, "TruckArrive",0
End Sub
```

The above code first declares a variable of type CFCSim_*Entity*. The second line calls the *AddEntity* method to obtain a reference to a new entity, which is then assigned to the declared variable. On the third line, the *ScheduleEvent* method is used to schedule the first event for the starting entity. The *ScheduleEvent* method expects the entity as a first parameter. The second parameter is the event name to be scheduled and the third parameter is the duration from the current simulation time at which the event should occur.

The main simulation processing code is defined in *OnSimulationProcessEvent* event handler. For the batch_Plant, this is done as follows:

```
Public Sub Batch_Plant_OnSimulationProcessEvent (ob As CFCSim_ModelingElementInstance, MyEvent As String, Entity As CFCSim_Entity)
  Dim NewTruck As CFCSim_Entity

  Select Case MyEvent
  Case "TruckArrive"
    ' schedule the arrival of the next truck
    Set NewTruck = ob.AddEntity
    ob.ScheduleEvent NewTruck,"TruckArrive",20.0

    ' schedule the next event for the current truck
    ob.ScheduleEvent entity,"RequestMixer",0.0

  Case "RequestMixer"
    If ob.RequestResource("Mixer",entity) Then
      ob.ScheduleEvent entity,"ReleaseMixer",15.0
    End If
  Case "ReleaseMixer"
    ob.ReleaseResource "Mixer",entity

    ob.DeleteEntity entity
  End Select
End Sub
```

The parameters of the event provide the event name that must be processed as well as a reference to whichever entity originally scheduled the event. For the "TruckArrive" event, the above code first creates another truck entity representing the next truck that will arrive then schedules its arrival time at 20 minutes. This means that currently, the assumption is made that the inter-arrival time of the trucks is 20 minutes. Next the code schedules the "RequestMixer" event for the current truck.

When the current event is the "RequestMixer", the above code first calls the *RequestResource* method to obtain the mixer resource. If the mixer is available, then the method call will return a True value. Otherwise a False value will be returned and both the current event and the requesting entity are saved and automatically added to the resource queue. When the mixer becomes available, the saved event will be automatically rescheduled for the saved entity. When the mixer is obtained, the "ReleaseMixer" event is scheduled to occur in 15 minutes. The assumption here is that loading time will be 15 minutes.

When the "ReleaseMixer" event is to be processed, the above code first calls the *ReleaseResource* method to release the mixer and then destroys the current truck entity as it is no longer needed.

At this point a working tool that models the described situation has been defined. It can be tested using the Simphony Editor program and statistics can be examined for the mixer resource.

4.2 Step 2 – Adding Randomness

The second step will demonstrate how randomness can be incorporated into the simulation code through the random number sampling service. The assumption is now that, instead of trucks arriving every 20 minutes, the inter-arrival time will be exponentially distributed with a mean of 20 minutes. Similarly, the loading time will be normally distributed with a mean of 15 minutes and a standard deviation of 2 minutes. To do this, the implementation of the *OnSimulationProcessEvent* is modified as follows:

Before:

```
…
ob.ScheduleEvent NewTruck,"TruckArrive",20.0
…
ob.ScheduleEvent entity,"ReleaseMixer",15.0
…
```

After:

```
…
ob.ScheduleEvent
NewTruck,"TruckArrive",Sampler.expntl(20.0)
…
ob.ScheduleEvent
entity,"ReleaseMixer",Sampler.normal(15.0,2)
…
```

Instead of supplying direct deterministic values, the above code calls the sampling service to obtain a random number.

4.3 Step 3 – Incorporation of Different Truck Sizes

The next step is to demonstrate how trucks of different sizes can be incorporated into the model. An assumption is made that, on average, half the incoming trucks will be large and half will be small. For large trucks, the loading time is Normal(25,2); for small trucks it is Normal(15,3). To do this, an entity attribute will be used to determine the size of the truck. For the initial entity created at the beginning of the simulation run, it is assumed to be large:

```
Public Sub Batch_Plant_OnSimulationInitializeRun (ob As
CFCSim_ModelingElementInstance, RunNum As Integer)
  Dim truck As CFCSim_Entity
  Set truck = ob.AddEntity

  truck("Size")="Large"

  ob.ScheduleEvent truck, "TruckArrive",0
End Sub
```

The main change highlighted above initializes a new entity attribute called "Size" and assigns it the value "Large". Next, a similar change is made to the *OnSimulationProcessEvent* event handler as follows:

```
    Dim x as single
    …
    Select Case MyEvent
    Case "TruckArrive"
     ' schedule the arrival of the next truck
     Set NewTruck = ob.AddEntity
     x = Sampler.uniform(0,1)
     If x < 0.5 Then
      NewTruck("Size")="Small"
     Else
     NewTruck("Size")="Large"
    End If

    ob.ScheduleEvent
NewTruck,"TruckArrive",Sampler.expntl(20.0)
  …
```

The above code first declares a single precision floating point variable (Single). When the "TruckArrive" event is handled, the above code samples a random uniform number between 0 and 1. If this number is less than 0.5, it sets the truck size to "Small" otherwise, it sets it to "Large". This has the effect of generating an equal number of small and large trucks.

The other change required will be to the processing code for the "RequestMixer" simulation event. Changes are as follows:

```
  Case "RequestMixer"
     If ob.RequestResource("Mixer",entity) Then
       If entity("Size")="Large" Then
         ob.ScheduleEvent
entity,"ReleaseMixer",Sampler.normal(25,2)
       Else
         ob.ScheduleEvent
entity,"ReleaseMixer",Sampler.normal(15,3)
       End If
     End If
```

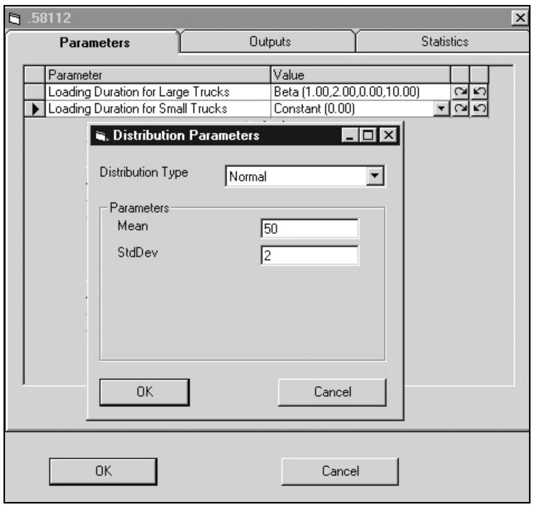
The above code first checks the value of the entity's "Size" attribute and, depending on its value, the appropriate parameters of the distribution are used.

4.4 Step 4 – Using Parameter Attributes

Thus far, the element definition includes numerous assumptions about the batch plant operation. This means that if the owner of the batch-plant company would like to experiment with different truck inter-arrival times or loading duration, they must either posses enough information to modify the code, which is highly unlikely, or contact the developer to make the necessary changes. This is obviously not practical; what is needed instead is a parameterized modeling element. This means that attributes for the modeling element must be defined, which the plant owner can change in the Simphony Editor. In order to provide the plant owner with control over the truck loading duration, two attributes are added to the modeling element as follows:

```
Public Function Batch_Plant_OnCreate(ob As
CFCSim_ModelingElementInstance, x As Single, y As
Single) As Boolean
  Batch_Plant_OnCreate=True
  ob.OnCreate x,y,True
  ob.AddAttribute "LoadLarge","Loading Duration for
Large Trucks",CFC_Distribution, CFC_Single,
CFC_ReadWrite
  ob.AddAttribute "LoadSmall","Loading Duration for
Small Trucks",CFC_Distribution, CFC_Single,
CFC_ReadWrite
  ob.AddResource "Mixer",1
End Function
```

The changes highlighted above use the *AddAttribute* method to declare two attributes of type "CFC_Distribution". This means that users will be presented with a special dialog box, as shown in Figure 5, where they can choose the distribution type and set its parameter values.



Parameter	Value
Loading Duration for Large Trucks	Beta (1.00,2.00,0.00,10.00)
Loading Duration for Small Trucks	Constant (0.00)

Distribution Type: Normal

Parameters
Mean: 50
StdDev: 2

Figure 5: Sample Element Property Dialog Box

Once the attributes are declared, their values will need to be used as part of the simulation processing. To do this, the *OnSimulationProcessEvent* event handler is changed as follows:

```
    If entity("Size")="Large" Then
      ob.ScheduleEvent
entity,"ReleaseMixer",ob("LoadLarge")
    Else
      ob.ScheduleEvent
entity,"ReleaseMixer",ob("LoadSmall")
    End If
```

Instead of specifying a value directly, the above code simply references the declared attribute, which returns a random value based on the user's specifications.

5 CASE STUDY

Simphony was introduced to five students in a graduate class. Each student had the benefit of one computer applications course, a basic Visual Basic education, and simulation training in the course. After attending four one-hour lectures and four two-hour lab sessions, students were asked to develop a Simphony CYCLONE template as part of an assignment. CYCLONE was selected for the assignment because it consists of a relatively small and easily understood number of modeling elements. Further, a standalone CYCLONE simulation tool was developed by a third-year computing science student for instructional purposes. The programmer's development time was approximately 230 hours. This did not include time spent becoming familiar with the CYCLONE method. The template development time of the students, which averaged approximately 40 hours, is detailed in Table 2.

Table 2: Summary of Students' CYCLONE Template Development

Student	Hours	Grade	Normalized Hours
A	35	60%	58.3
B	36	100%	36
C	15	90%	16.7
D	26	60%	43.3
E	32	65%	49.2

Provided hours were first normalized by dividing them by the assigned percentage grade. This was done as a means of extrapolating the number of hours it would have taken to provide a fully functional tool. Some students did not implement certain features such as priority queue allocations and counter statistics.

6 CONCLUSIONS

This paper presented a system called Simphony which simplifies and standardizes both the development and the utilization of construction special purpose simulation (SPS) tools. The modeling component allows for the construction of hierarchical and modular simulation models using graphical or script-based interfaces. Users have access to numerous templates; some general purpose and others for modeling specific construction methods. Novice developers can also extend the available templates by creating new modeling elements for specific use.

Experiments involving graduate students proved that novice developers are able to easily develop new simulation templates. Further, the time required for students to develop a Simphony based CYCLONE template was shorter than the time it took a programmer to develop a standalone version by a factor of six.

ACKNOWLEDGMENTS

This work was funded by a number of construction companies in Alberta and the Natural Science and Engineering Research Council of Canada under grant number IRC-195558/96.

REFERENCES

AbouRizk, S., Hajjar, D. (1998). A Framework for Applying Simulation in the Construction Industry, *Canadian Journal of Civil Engineering*, Volume 25, Number 3, June 1998.

Hajjar, D., AbouRizk, S., (1996) Building a Special Purpose Simulation Tool for Earth Moving Operations, *Proceedings of the 1996 Winter Simulation Conference*, pp. 1313-1320

Hajjar, D., AbouRizk, S. (1998). Modeling and Analysis of Aggregate Production Operations, *Journal of Construction Engineering and Management*, ASCE, Volume 124, Number 5, September 1998.

Hajjar, D., AbouRizk, S., Xu, J. (1998). Optimizing Construction Site Dewatering Operations using CSD, *Canadian Journal of Civil Engineering*, CSCE, Volume 25, Number 3, June 1998.

Halpin, D. W. (1977). CYCLONE: Method for Modeling of Job Site Processes, *Journal of the Construction Division*, ASCE, 103(3),489-499.

AUTHOR BIOGRAPHIES

DANY HAJJAR is a Research Engineer in the Department of Civil Engineering at the University of Alberta. He received his Bachelor of Science with Specialization in Computing Science from the University of Alberta in 1995. He received his Construction Engineering and Management Ph.D. degree from the University of Alberta in 1999. His research interests are focused on applying software engineering and simulation techniques to the management and control of construction projects.

SIMAAN ABOURIZK is a Professor in the Department of Civil Engineering at the University of Alberta. He received his B.S.C.E. and M.S.C.E. in Civil Engineering from Georgia Institute of Technology in 1984 and 1985, respectively. He received his Ph.D. degree from Purdue University in 1990. His research interests focus on the application of computer methods and simulation techniques to the management of construction projects.