Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

AN OBJECT-ORIENTED PHONE CENTER MODEL USING SIMPLE++

Gerald A. Levasseur

AESOP Corporation
233 South Wacker Drive
Sears Tower, Suite 9604
Chicago, Illinois, 60606, U.S.A.

ABSTRACT

A demonstration model and Application Object Template (object library) was created to show how SIMPLE++ simulation software can be applied to service industry telephone call handling centers. In addition, this example was designed to show modeling techniques that can be used to take advantage of some key object-orientation concepts to quickly create highly flexible simulation models. Finally, some features of the SIMPLE++ simulation package are illustrated.

1 INTRODUCTION

The object-oriented simulation package SIMPLE++ was first commercially introduced in Germany in February of 1992, and AESOP GmbH was tasked to further develop and market the software and related products and services. In North America, AESOP Corporation is responsible for SIMPLE++ licensing and related services. Since its creation, SIMPLE++ has been applied in a variety of industries including automotive manufacturing, rail transportation, electronics assembly and fabrication, pharmaceuticals, hospitals, chemical plants, semi-conductor facilities, material handling, and warehousing. Application functions include scheduling, production planning and control, design support, and capital equipment justification. Users include corporations, governments, universities and research institutions worldwide.

Geuder presented some of the most important object-oriented features of SIMPLE++ in two recent papers (Geuder 1995a, Geuder 1995b). The two most notable concepts that will be used in this example are inheritance and hierarchy. Inheritance is the ability of a child (a specific instance of a class) or a sub-class to maintain the properties or functionality of a parent (class). Hierarchy is the structure where multiple layers of inheritance are possible.

1.1 Problem Description

The Phone Center Demonstration Model presented here illustrates how SIMPLE++ can be used to distribute calls to telephone customer service sites, such as those shown in Figure 1. Such sites are widely used in financial service, travel, customer support, catalog and mail order businesses to answer customer calls. In this model, different types of calls arrive at a central distribution site, with each type of call following its own arrival distribution.

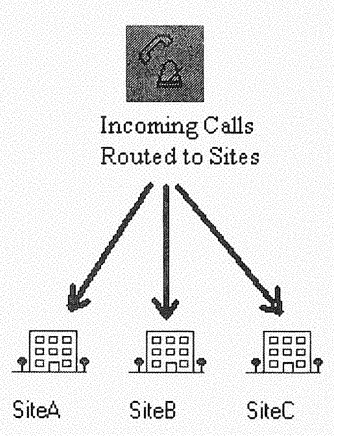

Figure 1: Incoming Calls are Distributed to Sites

For each type of call, there is one type of agent able to answer it. Upon a call's arrival, a decision is made to distribute the call to a specific phone site where the agents are located. There can be any number of sites, as long as there is at least one. One of three primary criteria may be selected to determine which site a call should be routed to:

1. Proportion of Agents at Site. The number of agents at a given site that can handle an incoming type of call is divided by the total number of agents at all sites that can

handle the call. This proportion corresponds to the probability the call will be routed to a given site.

2. *Shortest Queue Length.* The site with the shortest queue length for a given type of call is selected. If there are no queues, then a secondary criterion may be selected.

3. *Number of Free Agents.* The site with the greatest number of free agents that can handle a given type of call is selected. If there are no free agents, a secondary criterion may be selected.

Secondary criterion may be selected in the event the first criterion conditions do not exist. This criterion can be any of the three primary criteria; however, if the secondary and primary criteria selected are the same, the secondary criterion will be ignored. If the primary criterion was Proportion of Agents at Sites, no secondary criterion is required. If the primary criterion was Shortest Queue Length and there are no queues, the secondary criterion will be used. If the primary criterion was Number of Free Agents and there are no free agents at any site, the secondary criterion will be used.

Once a call is distributed to a site, the call is given to the appropriate agent. If all agents are busy, the call will be placed in a queue until an agent becomes free. Calls exit the queue based on the First-In-First-Out (FIFO) rule. When a call is completed, appropriate statistics are collected and the caller exits the system.

1.2 Concepts Demonstrated

In addition to showing the potential application of SIMPLE++ in the service industry, this model demonstrates the following SIMPLE++ features and modeling techniques:

Hierarchy and Inheritance. This model shows how hierarchy and inheritance can be used to simplify modeling and contribute to modeling flexibility.

SimTALK. SimTALK is the built-in SIMPLE++ simulation language. This language is accessed by means of methods (objects that contain SimTALK coding).

Anonymous Identifiers. In SIMPLE++, anonymous identifiers are used to indirectly refer to an object without having to know in advance what object it is. This is especially useful because identical methods can be used in different models or objects (AESOP GmbH 1995). These anonymous identifiers can be used to greatly enhance the flexibility of a SIMPLE++ model, especially when used in conjunction with its object-orientation features.

Modeling Flexibility. This demonstration model will show how constructors and destructors, when used in conjunction with anonymous identifiers and other SIMPLE++ features, give the modeler a powerful tool in creating flexible models. SIMPLE++ constructors activate specified methods when a given object is inserted into a model, while destructors activate specified methods when a given object is removed from a model.

Running Multiple Models Simultaneously. Because all SIMPLE++ models are encapsulated objects, several models can be run at the same time by placing them inside another object. This demonstration model will show an example of how to run two models at once. A typical use may be to directly compare two dispatching criteria to see the effects side-by-side.

Other SIMPLE++ Features. This paper will also show some other SIMPLE++ features: Dialogue Boxes, Tables, and Plotters. Dialogue boxes are used so modelers can create a way for the end users to easily make choices while hiding model complexity. SIMPLE++ tables can hold different types of data, including tables within tables and even objects. Finally, plotters are used to graphically view performance statistics over time.

2 THE PHONE CENTER MODEL

Objects shown in the top five rows of the Application Object Template in Figure 2 are basic objects. These objects come with every SIMPLE++ package. The user may use these objects directly in a model, or may create an Application Object Template from them. In this example, the Application Object Template consists of the objects created for this demonstration model, and is found in the bottom four rows of the template.

To start the model, the user double-clicks on the Demo object in the Application Object Template. This will cause the CriteriaDialog box to open. The dialogue box feature in SIMPLE++ is used to hide model complexity when desired, so the ultimate end user does not need to understand the inner workings of a model to operate it. In this example, the user may select the previously described primary and secondary criteria for dispatching calls to sites.

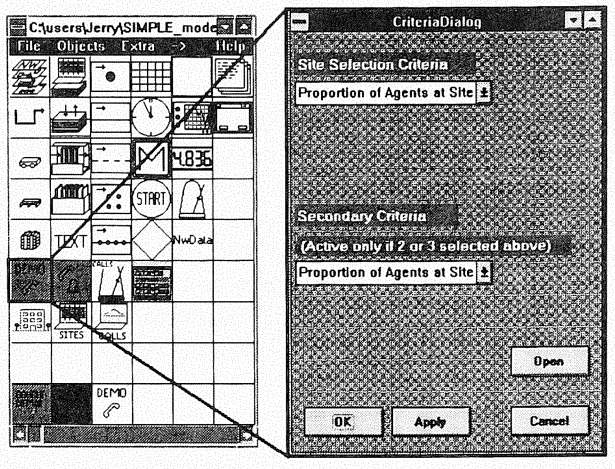

Figure 2: The Application Object Template for the phone center demonstration model (left), and a dialogue box used to select decision criteria (right).

After the criteria are selected, the user should select the Open button to open the Demo frame. A frame is simply an empty object that is used to hold other objects. In this example, the Demo frame holds an entire model. Once Open is selected, the frame shown in Figure 3 will appear, showing the contents of the frame Demo and revealing the structure of the model. The default behavior of double clicking on a frame is to open it directly; however, in this demonstration example the modeler added the command to open the dialogue box before opening the frame.

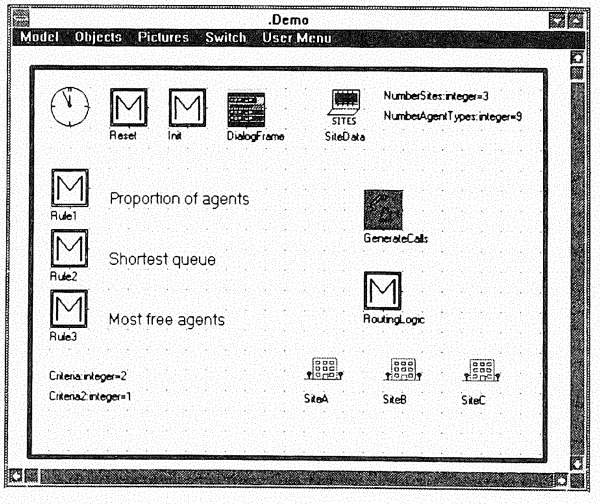

Figure 3: The Demonstration Model.

2.1 Inside the Demo Object

The EventController, represented by a clock icon in Figure 3, is an instance of the basic object that is used to control the model's simulated time. Before running the model, the user should open the Event Controller and run the Reset and Init options. This will call all methods in the model named Reset and Init, respectively, which are written by the modeler. In this example, they clear all data inserted into tables by previous runs while leaving user entered data intact. The Init methods in this example will calculate the total number of agents and the proportion of agents at each site, and will write this information to the SiteData table and other appropriate tables. Reset and Init are also excellent ways to set the initial conditions of a simulation run.

The SiteData table shown in Figure 3 is opened in Figure 4. This table contains the phone sites in the model, as well as tables-in-a-table that show the overall numbers of agents at all sites. The objects in the SiteName column are automatically entered into the table whenever a new site is inserted into the model. The user names the embedded table (called Number here) and enters the total number of agents at this site in the Total column. The Reset and Init methods fill in the TotalAllSites and Proportion columns, which contains the total numbers of agents at all sites and the proportion of the total at this site, respectively.

NumberSites and NumberAgentTypes shown in Figure 3 are NwData objects. NwData objects contain network data that can be used anywhere in the frame. In this example, the user does not need to do anything with these objects -- they are set automatically by Init. When Init methods are activated via the Event Controller, they check the dimension of the SiteData table. All variables in the model that refer to the number of sites are based on this dimension. Thus, any loops or counters referring to the number of sites are automatically re-adjusted, even if a site is added or deleted during a simulation run.

Criteria and Criteria2 are also NwData objects. These show the primary and secondary criteria selected in the dialogue box. The user does not need to do anything with these objects, but may change the criteria directly from here.

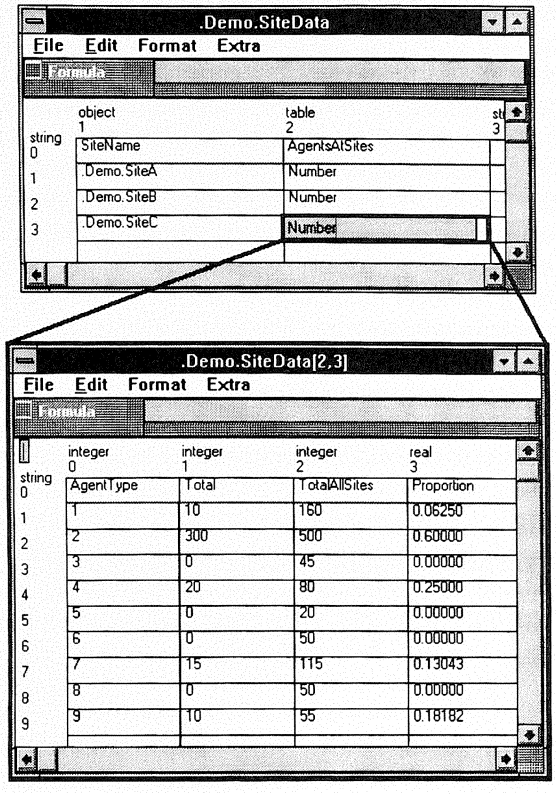

Figure 4: SiteData tables. Note they can contain a variety of types of data, including objects and nested tables.

2.2 Call Generation

The GenerateCalls object shown in Figure 3 is opened in Figure 5. This object contains nine generators, each which send calls to the RoutingLogic method. A generator is a basic object that calls a given method according to a specified distribution. Each of the generators in this example has its own distribution function that simulates each of the nine incoming call types. The user may change these distributions in the GenerateCalls dialogue box. There are nine built in distributions, or the user may define a formula, read a value from a table, or simply set a constant. In addition, any parameter for a distribution can be a variable. In the generator shown in Figure 5, the method RoutingLogic is activated according to a uniform distribution between 5 and 23 seconds. The anonymous identifier *location* refers to the frame the generator's current frame is in. If all generators follow identical distributions, the user may activate inheritance to use the parent's (located in the Application Object Template) distribution. Alternatively, the inheritance can be switched off to set individual values.

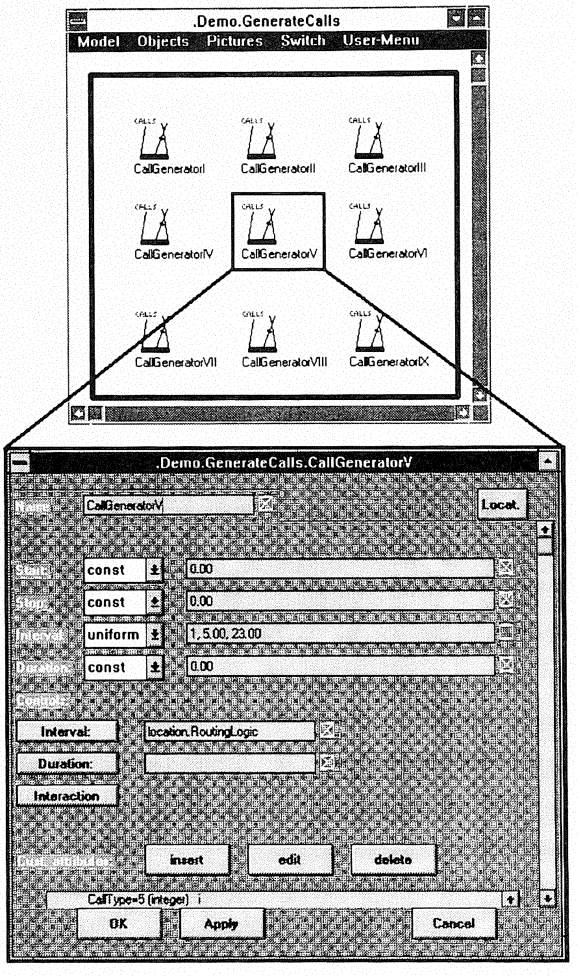

Figure 5: GenerateCalls and a Generator's Dialogue Box.

Because of SIMPLE++ object-orientation, one could easily modify this object and not worry about how it would effect the rest of the model. For example, the modeler could easily replace the GenerateCalls object and its nine generators with another object that contained only a single generator, a method and some tables that accomplished the same end result. That would be useful if the modeler wished to generalize the number of agents at some future date.

2.3 Routing Logic Methods

The RoutingLogic object shown in Figure 3 is a method. This method sends each in-coming call to other methods that perform Rule1, Rule2, or Rule3 depending on the primary criteria selected. Figure 6 shows this method's coding written in SimTALK. While the scope of this

paper does not allow a detailed discussion of SimTALK, the reader can see how it may be activated. A full description of SimTALK is contained in the SIMPLE++ reference manual (AESOP GmbH, 1995).

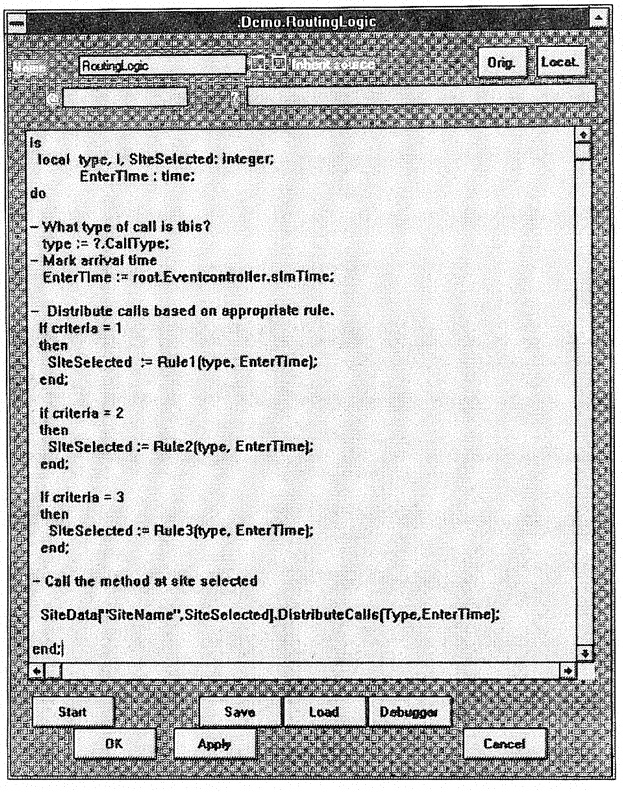

Figure 6: A Sample Method and SimTALK

The question mark used in the coding is an anonymous identifier that refers to the object that called this method, in this case a generator. Thus, this method can be called from anywhere in the model without even knowing what object called it. Also note the last line of coding before the end statement directly calls the site method DistributeCalls based on the object found in the SiteData table. By using tables to hold the object, any object with a method called DistributeCalls can be referenced, even if the object was inserted into the model after the simulation was started.

The rule methods select the sites according to the criteria previously described. If a secondary criterion is required, they will call the appropriate rule method. After the site has been selected, they will return an integer to RoutingLogic indicating which site has been selected. RoutingLogic then calls the DistributeCalls method that is located within the selected site's frame, giving it the type of call and the time it entered the system.

2.4 Inside the Site Object

An example of the Site frame is shown in Figure 7. When a call comes in from the RoutingLogic method in the Demo frame, it will go to the selected site's DistributeCalls method. Note that the site is directly referenced by the object in the table, and not by a string or integer variable. The method DistributeCalls decides if the call will go to a free agent immediately, or to a waiting list if no agents are free. If an agent is free, it decrements the number of free agents for this type of call in the Agents table, then it calculates the time the call will take. It will next calculate statistics and record them in the appropriate tables. Finally, after the call is complete, it will call the method Tally.

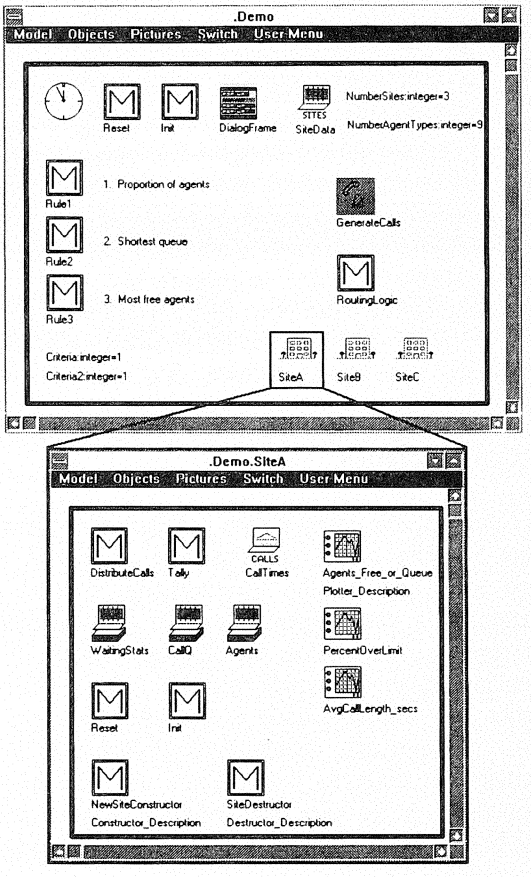

Figure 7: The Site Frame Contains the Objects Located in Each Site.

If no agents are free, DistributeCalls places this call in the CallQ table, which contains queues for each type of waiting call. It also updates the current count of waiting calls.

The method Tally is called when an agent becomes free. It adds one agent to the counter in the table Agents

for this type of call. If there is a queue for this type of agent, it will take the next waiting call out of the queue and then call the method DistributeCalls. This time DistributeCalls will be able to send the call to an agent because an agent is free.

Figure 7 also shows an excellent example of hierarchy. Each site in the Demo object is a child of the site object in the Application Object Template. When the modeler wishes to make a change to the sites, only the object in the template needs to be modified -- all of the children in the model will automatically inherit those changes. In addition, issues such as repeated names are not a problem. For example, all sites have tables in them called SiteData, but each is unique because each is in a different child of the site object. The CallTimes table in SiteA is distinguished from the CallTimes table in SiteB by virtue of its location. Likewise, a modeler adding a new site does not need to remember to add the table because each site will inherit the table from the parent. Thus, the behavior of each child can be easily modified and tested.

The time a call will take once it is answered by an agent is assumed to follow a type of truncated normal distribution. The parameters this distribution will use is set by the user in the CallTimes table shown in Figure 8. The mean and standard deviation is set by the user, and the method DistributeCalls will set any generated values below zero equal to zero. For this demonstration model, the normal distribution was selected because most people are familiar with it, even though it is unlikely to be the distribution selected in an actual application. To make all distributions the same for each site, the user may set them in the Application Object Template CallTimes table and turn on the Inherit Contents option in each site's table in the Demo model. To set the distributions individually for each site, the user may change them in the Demo model with the Inherit Contents option off. This allows the greatest flexibility and ease of data input.

2.4.1 Plotters in the Site Object

This model contains three plotters in each site. They are:

Agents_Free_or_Queue: The number of free agents, or if there are no free agents, the queue length (represented as negative numbers).

AvgCallLength_secs: The average length of a call. This is defined as the time the call arrives until the call has been completed, including any time in spent queues.

PercentOverLimit: The percent of calls that take over a user-defined time limit. A sample plot is shown in Figure 9.

Here, the hierarchical structure assures that each plotter will plot the variables associated with the site it is in, even if this site did not exist when the run was started. No changes to the model are required for the plotters when adding a new site.

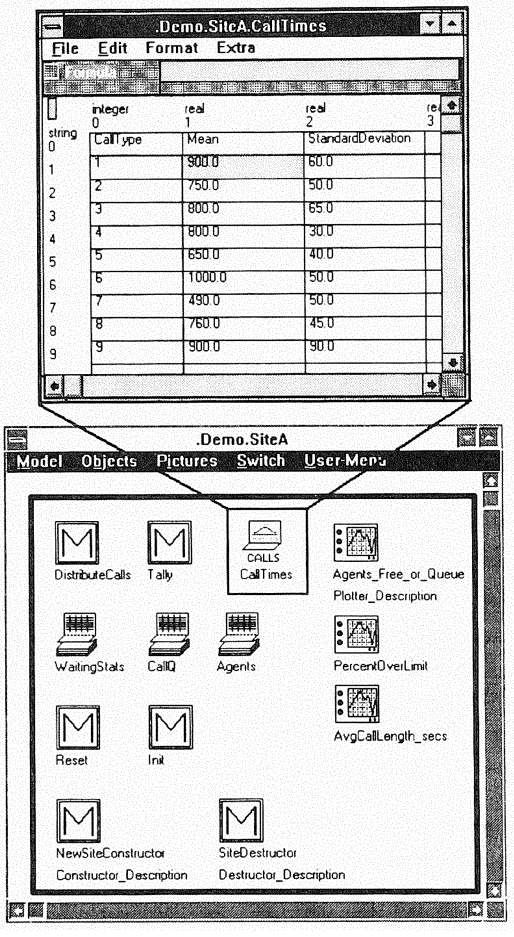

Figure 8: The CallQ Table Keeps Track of Waiting Calls

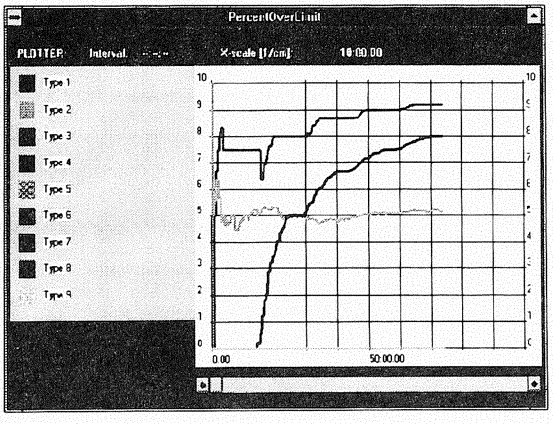

Figure 9: A Sample Plotter Output

2.4.2 Tables in the Site Object

In addition to the CallTimes table previously described, there are three other tables used in each site object. The user does not need to interact with the tables in the sites, but may open them to view data.

The WaitingStats table shown in Figure 10 is used by the model to collect data for the plotters. The columns NumberOfCalls, TotTime, Avg, OverLimit, and PercentOverLimit are filled in as the model runs. The Limit column is filled in before the model is run. If the user does not wish to use the values set in the Application Object Template's parent model, they can switch off the inheritance and set their own values at individual sites. Alternatively, they can change the parent's values to suit their needs.

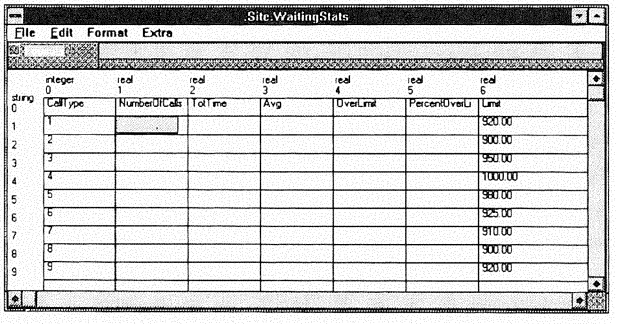

Figure 10: The WaitingStats Table

The CallQ table in Figure 11 keeps track of the queues for each type of call. In this example, the third type of agent has a queue of five calls waiting, each with its arrival time recorded in a queue file. The first type of agent has no queue, so the queue file is empty.

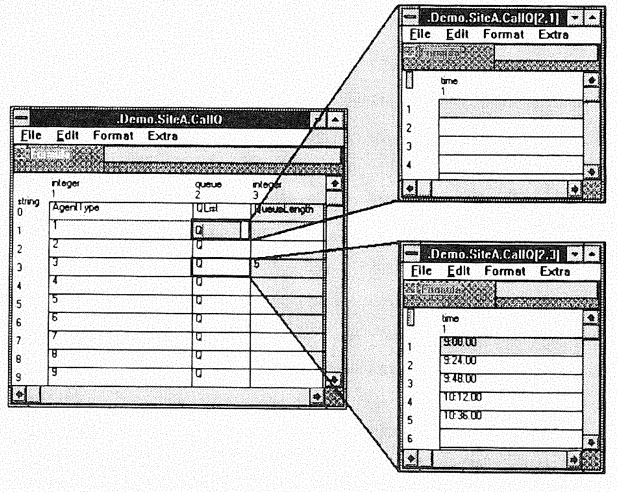

Table 11: The CallQ Table

Finally, the Agents table shown in Figure 12 gives the total number of agents and the number available. The total is set by the Init function when the model is started, and the number available is updated continuously as the model runs.

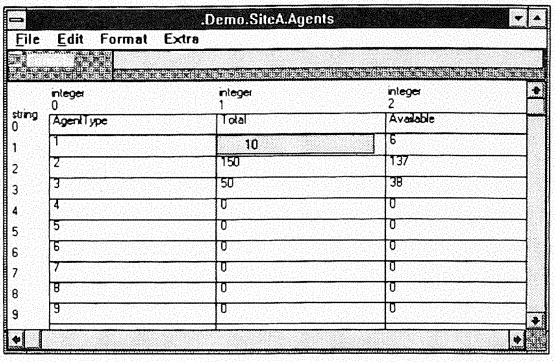

Figure 12: The Agents Table

3 MODIFYING THE PHONE CENTER MODEL

The Demo model in the Application Object Template can be easily modified or incorporated into other models. For example, sites can be easily added or deleted, or two instances of this model can be placed into a larger model to run both at once.

3.1 Adding and Deleting Sites.

The number of sites is controlled by the user, and the model is changed using the methods NewSiteConstructor or SiteDestructor shown in Figure 7. To add a site, the user just needs to add a child of the parent in the Application Object Template. This is done simply by clicking on the parent, then clicking in the Demo frame to add the child. In this demonstration model, whenever new site is added to a model as in Figure 13, a constructor modifies the model to automatically adjust for this new site. It does this by adding the new site to the SiteData table. The only action the user needs to take is to add the number of agents at the new site into the nested AgentsAtSites table. The operating system's output window will prompt the user of the required changes. Other than that, the entire process is fully automatic and no model modifications are required.

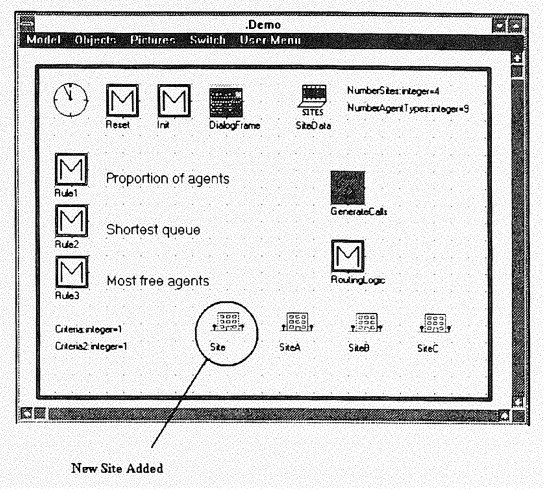

Figure 13: Constructors Used to Add a Site

If a site is removed from the model, a destructor automatically adjusts the model so that the model is fully operational without any user changes. Because each site is an object, no change to the model coding is required. The user simply needs to delete the site object from the model, and the computer will automatically re-calculate all of the data in the tables. If, however, deleting a site results in no agents of a given type being left in the model, the computer will prompt the user to add more agents of that type.

3.2 Running Multiple Models Simultaneously

The Application object Double Demo contains two Demo models, as shown in Figure 14. These can be run at the same time, each with its own number of sites, site selection criteria, number of agents, or other changes. This allows direct comparison of two strategies or options. In this example, hierarchy was used to treat two complete models as objects within a larger model.

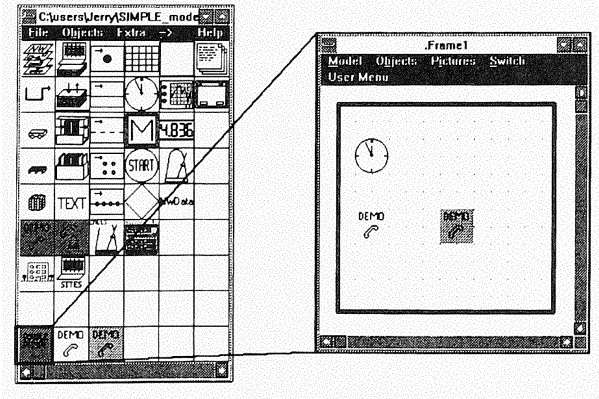

Figure 14: Two Models Can be Run Simultaneously

4 CONCLUSIONS

Object orientation, especially when combined with other SIMPLE++ features, can be used to create highly flexible models. In this paper, a service industry example was presented, but the concepts may be applied to any discrete-event simulation model.

REFERENCES

AESOP GmbH. 1995. *SIMPLE++ reference manual version 3.1*. Stuttgart, Germany.

Geuder, D. 1995a. Modular application objects: closing the gap between flexibility and ease of use. In *Proceedings of the 1995 European Simulation Conference*, ed. F. Breitenecker, I. Husinsky, 37-40. Eurosim Congress '95, Vienna, Austria.

Geuder, D. 1995b. Object Oriented Modeling with SIMPLE++. In *Proceedings of the 1995 Winter Simulation Conference*, ed. C. Alexopoulos, K. Kang, W. Lilegdon, D. Goldsman, 534-540. Arlington, Virginia

AUTHOR BIOGRAPHY

GERALD A. LEVASSEUR recently joined AESOP Corporation as their Chief Consultant. He received a BIE degree from Cleveland State University in 1985, an MSE from the University of Washington in 1990, and a Ph.D. degree in Industrial Engineering from the University of Washington in 1994. Jerry taught at the University of Tennessee at Chattanooga for two years, and has industrial engineering experience as a civilian employee of the Navy. He is a member of **SCS**, **IIE**, **ASQC**, and **APICS**.