Proceedings of the 1998 Winter Simulation Conference
D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.

MODELING WITH EXTEND™

Jim Rivera

Imagine That, Inc.
6830 Via Del Oro, Suite 230
San Jose, CA 95119, USA.

ABSTRACT

This document presents an overview of the Extend modeling environment. Extend is a general-purpose, graphically-oriented, discrete event, and continuous simulation application with an integrated authoring environment and development system. Extend's features will be demonstrated by examining a simple model of a single server, single queue system to which detail and enhancements will be added.

1 INTRODUCTION

For many years there has been a perceived dichotomy in simulation software between simulation languages and simulators. The languages were viewed as more powerful and general purpose, while simulators focused on ease-of-use and were generally industry specific. Extend bridges these two types of programs in one easy-to-use yet flexible software program. It exists as:

- A stand-alone simulation tool which can be used to create complex discrete event and continuous models without programming
- A simulation authoring package where model interfaces can be easily created, without programming, to enhance productivity and ease of use
- A development environment for building customized models of unique types of systems. The programming environment allows the modeler to create a simulator for a specific industry.

2 EXTEND'S MODELING ENVIRONMENT

Before looking into how Extend can be used to build models, it is helpful to understand the Extend modeling environment.

Extend models are constructed with library-based iconic blocks. Each block describes a step in a process or a calculation. The dialogs associated with each block allow you to define the behavior of the block as well as report on block results. Blocks reside in libraries. Each library represents a grouping of blocks with similar characteristics such as Discrete Event, Plotters, Electronics, or Business Process Reengineering. Blocks are placed on the model worksheet by dragging them from the library window onto the worksheet. The flow is then established between the blocks.

There are two types of logical flows between the Extend blocks. The first type of flow is that of "items," which represent the objects that move through the system. Items can have attributes and priorities associated with them. Examples of items include parts, patients, or a packet of information. The second type of logical flow is "values," which will change over time during the simulation run. Values represent a single number. Examples of values include the number of items in queue, the result of a random sample, and the level of fluid in a tank.

Each block has connectors that are the interface points of the block. Figure 1 shows the connector symbols for the value and item connectors.

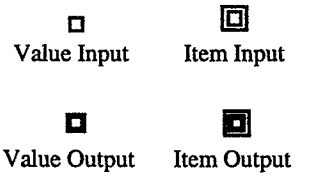

Figure 1: Value and Item Connectors

Connections are lines used to specify the logical flow from one connector to another. Double lines represent item connections and single lines represent value connections.

3 CAR WASH EXAMPLE

In the following example, we will consider a single server, single queue system. For the purpose of illustration, we will model a car wash with one wash bay and one waiting line. The model for this car wash is shown in Figure 2.

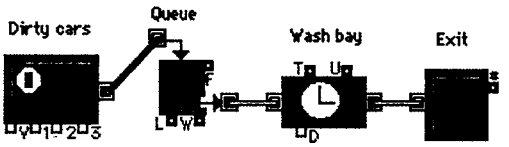

Figure 2: A Single Server Single Queue Model

The block on the far left is a Generator block and periodically creates items (in this case dirty cars). Following this is a Queue, FIFO block that holds the cars until requested by the next block. The Activity Delay has a limited capacity of one processing unit and delays the car for a fixed amount of time. This block represents the wash bay. The last block in the model is an Exit block that removes the cars from the system.

3.1 Random Processing Time

Suppose that the processing time for the wash bay is best represented by a specific random distribution. This can be modeled by connecting the output of an Input Random Number block to the delay connector (labeled "D") on the Activity Delay block as in Figure 3. Every time a car enters the wash bay, a new processing time is requested from the Input Random Number block. For each request, the Input Random Number block generates a new processing time from the specific random distribution defined in the block's dialog.

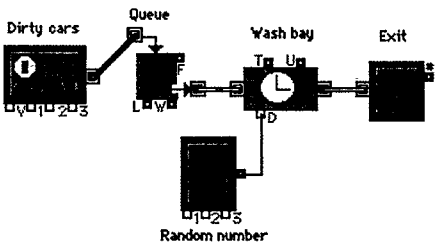

Figure 3: A Model with Random Process Times

3.2 Graphical Output

To graphically display model metrics, you can add a Discrete Event Plotter. In this example (Figure 4), the Plotter will graph the contents of the Queue, or the number of dirty cars waiting in line, over time. To accomplish this, the Discrete Event Plotter value input connector is connected to the Queue's length (labeled "L") value output connector as follows:

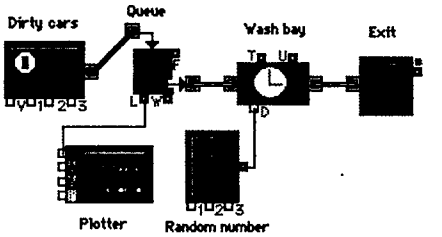

Figure 4: Discrete Event Plotter Added to Model

3.3 Attributes

Assume that our car wash offers two types of washes, basic and deluxe, and that the processing time is dependent upon the type of wash requested. To differentiate between the two different types of wash requests, we can add attributes to the dirty cars. Using a Set Attribute Block we can add an attribute called "type" to each car and randomly set the value of this attribute to 0 (basic) or 1 (deluxe) using another Input Random Number Block as shown in Figure 5. As the dirty cars leave the queue and enter the wash bay, we can read the "type" attribute using a Get Attribute block, and convert this number to a value representing the mean processing time for washes of that type using a Conversion Table block. The mean value can then be fed into the Input Random Number Block that is already connected to the delay connector of the Activity Delay (Figure 5).

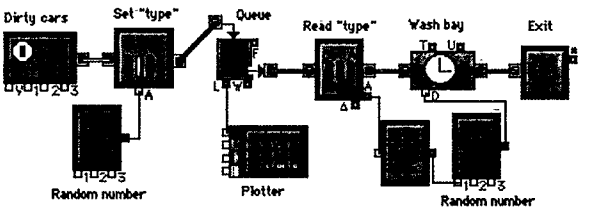

Figure 5: Setting "Type" Attribute

3.4 Resources

When the dirty cars are ready to be washed, they are driven through the wash by one of the car wash attendants. We can model the attendants as resources by adding a Resource Pool block. Within this block we can specify how many attendants are on shift. We must also replace the Queue, FIFO block with a Queue, Resource Pool block.

Within the Queue, Resource Pool Block we can specify the type and number of resources required before the item may be released to the next block. Therefore, dirty cars will enter the Queue, Resource Pool block and wait until an attendant is available. If an attendant is available and the wash can accept another car, the number of attendants in the Resource Pool block is decremented by one and the car is allowed to proceed into the wash bay. Upon exiting the wash bay, the attendant is no longer needed and may be release back to the Resource Pool with the Release Resource block as shown in Figure 6.

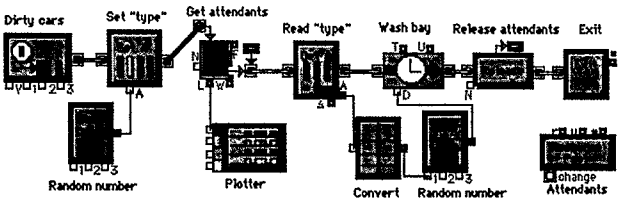

Figure 6: Modeling Resources

3.5 Activity Based Costing

Now that we have a basic model of our car wash, we may decide that we would like to calculate the average cost of washing the cars. We know that each attendant is paid $8.50 an hour. We also know that each car will use $1.25 in soap and that the electricity and water used by the wash bay cost $1.50 per minute. We can define the cost of the attendant within the Resource Pool block and the cost of the soap, water and electricity into the Activity Delay block (Figure 7). As we run the model, the accumulated cost of each vehicle is automatically calculated and stored in an attribute. The Cost By Item Block can be added to read the cost attribute, sort the items by an attribute, such as the "type" attribute, and report on the throughput, total cost and average cost by type of wash requested. The Cost Stats Block can also be added to report the total cost generated in each of the blocks, for example the total cost generated by the attendants (Resource Pool) or the wash bay (Activity Delay).

3.6 Interprocess Communication

The term interprocess communication (IPC) describes the act of two applications communicating and sharing data with one another. This feature allows you to integrate external data and applications into your Extend models. For example, suppose we would like to use data from an Excel spreadsheet as inputs to our car wash model. After copying data from the spreadsheet, you can select a parameter in Extend and choose Paste Link from the Edit menu. The data will be copied into the parameter's field and a dynamic link between that parameter and the specific cells from the Excel spreadsheet will be created. If the values change in the spreadsheet, the parameters will automatically be updated in Extend. A link of this type can be created for both input parameters and output data. Blocks and functions are also provided for additional interprocess communication capabilities such as running macros, using spreadsheets as lookup tables, or even controlling Extend from another application.

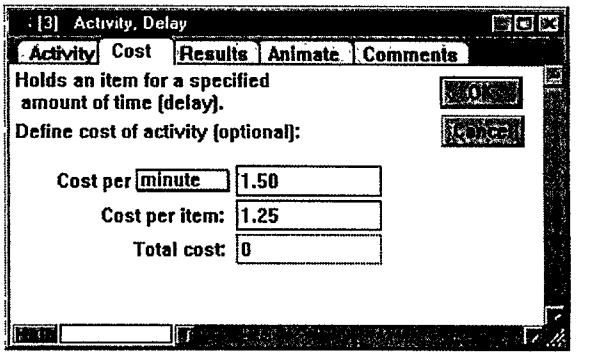

Figure 7: Cost Tab of Activity Delay Block

3.7 Model Results

Once the simulation run has completed, the results of the simulation are reported within the blocks. Double clicking on each block reveals the information collected from the simulation run. For example, double clicking on the Queue, Resource Pool block opens a dialog showing the information illustrated in Figure 8 about the state of the Queue, Resource Pool block:

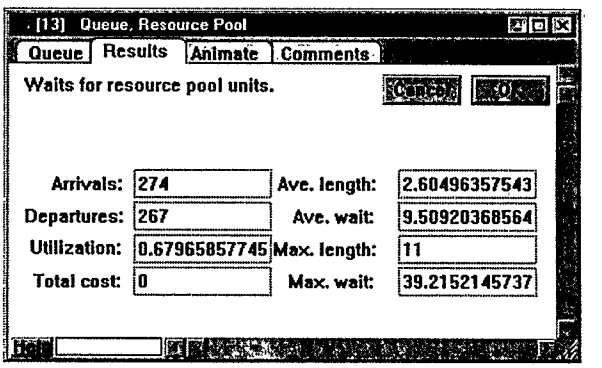

Figure 8: Dialog of Queue FIFO

The Plotter block shows the number of items stored in the Queue, Resource Pool over time in both graphical and tabular format as illustrated in Figure 9.

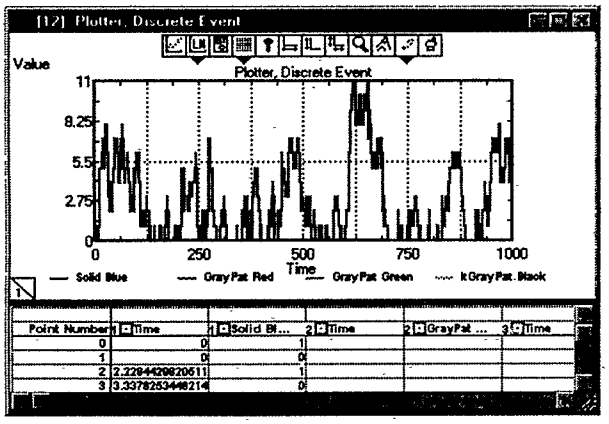

Figure 9: Plot of Queue Length

Simulation results may be stored in a table, plotted, cloned to a different area of the worksheet, exported to another program such as a spreadsheet or database, displayed in an animation, or used to control some aspect of the outside world via external device drivers.

3.8 Data Analysis

Extend offers a number of tools for analyzing both input and output data. An interface is provided to distribution-fitting programs that aid users in selecting the appropriate statistical distributions based on empirical data collected in the field.

In addition, sensitivity analysis can be performed to determine how sensitive a system is to changes in specific input parameters. Suppose we would like to determine how sensitive our car wash is to changes in the inter-arrival time of dirty cars. To accomplish this, we can perform sensitivity analysis on the inter-arrival mean parameter of the Generator Block. By selecting the inter-arrival time dialog item and choosing Sensitize Parameter from the Edit menu, we can define how the parameter should change from run to run. Simulation parameters such as the number of runs and simulation end time can be specified in the Simulation Setup dialog item under the Run menu. By cycling through different inter-arrival times for the dirty cars and comparing the results from the different runs, we can get an understanding of how sensitive our car wash is to the arrival rate of dirty cars.

The Statistics library helps users to collect and analyze output data. Blocks from the Statistics library automatically gather data from the appropriate blocks and calculate confidence intervals.

4 CUSTOMIZING EXTEND

The above discussion illustrates the highly graphical and interactive nature of Extend. However, Extend can also take the shape of the model application. Interfaces, components, and graphics can be used which tailor the model to a specific application area.

The most visible aspect of a custom model is the user interface. By modifying an existing interface or creating a new one, the simulation modeler is able to create a model which can be exercised by someone more familiar with the system than with the simulation tool. Models can be built that fit naturally into the conceptual framework of the person using the model. The following sections will describe some of the tools provided in Extend that allow you to customize your model.

4.1 Animation

Animation is a powerful presentation and debugging tool that can greatly increase model clarity. In Extend, animation icons moving from block to block represent the flow of items through the system. Users can choose from a number of icons provided with Extend or create their own in an external drawing package.

For example, we may want to see cars traveling from block to block in our car wash model. By selecting the appropriate icon in the Animate tab of the Generator block, we can define how all of the items created by the Generator will be represented. In addition, any block that the items pass through has the capability of changing the item's animation icon. For example, we may choose to represent every item exiting the Generator block with a picture of a dirty car. As the items pass through the wash bay, we can have the Activity Delay block change each item's animation picture to a clean car, thus providing visual cues of how the items are changing as they progress through the model.

In addition, custom animation can be added to display pictures and text, level indicators, pixel maps, and QuickTime movies.

4.2 Hierarchical Modeling

Hierarchy allows models to be subdivided into logical components or sub-models. A single descriptive icon can represent each sub-model. Double clicking on the hierarchical block will open a new window displaying the sub-model. This can greatly simplify the representation of a model and allow the user to hide and show model details as appropriate for the target audience.

Let us consider our car wash model (Figure 6). As we have added detail to the model, the number of blocks has increased. As a result, the representation of the model has become slightly encumbered with model details.

Suppose we would like to represent the model by the system's most basic elements:

- the arrival of dirty cars
- the queue of dirty cars waiting for availability of the wash bay
- the wash bay
- the departure of clean cars

By selecting a group of blocks and choosing Make Selection Hierarchical from the Model menu, you can encapsulate a section of the model within a hierarchical block. This block can be saved to a library and re-used in other models. The icon for the hierarchical block can be modified by using the built-in icon editor or by importing an existing picture. The number of hierarchical layers allowed in Extend is unlimited. Figure 10 shows the car wash model with hierarchical blocks representing some basic elements of our car wash. While the representation of the model is more intuitive and simple than Figure 6, the detail of the model can still be accessed by double clicking on any of the hierarchical blocks to display the underlying sub-model.

Figure 10: Car Wash Model with Hierarchical Blocks

4.3 Dialog Cloning and the Notebook

As noted in the previous section, input and output parameters associated with the model can be found in the dialogs of the appropriate blocks. While this provides an intuitive association between system metrics and the constructs used to model them, it can make searching for specific data more difficult when working with large models containing many layers of hierarchy. An effective way of dealing with this is to use the notebook and cloning feature. With the notebook, you can create a single custom interface to your model consolidating critical parameters and results to a central location.

The notebook is a separate window associated with each model. Initially the notebook is a blank worksheet to which text, pictures, and clones can be added. Clones are direct links to dialog parameters and can be create by selecting the Cloning Tool from the tool bar and dragging a dialog parameter from a block dialog to the notebook or model worksheet. Once a clone is created, any changes to the clone are immediately reflected in the block and visa versa. Therefore it is no longer necessary to access the block's dialog to change an input parameter or view updated results. Creative use of the notebook can result in a simple yet effective interface for a large, complex model. Figure 11 shows the notebook for the car wash model as an illustration of how the notebook can be used to consolidate important parameters into one location.

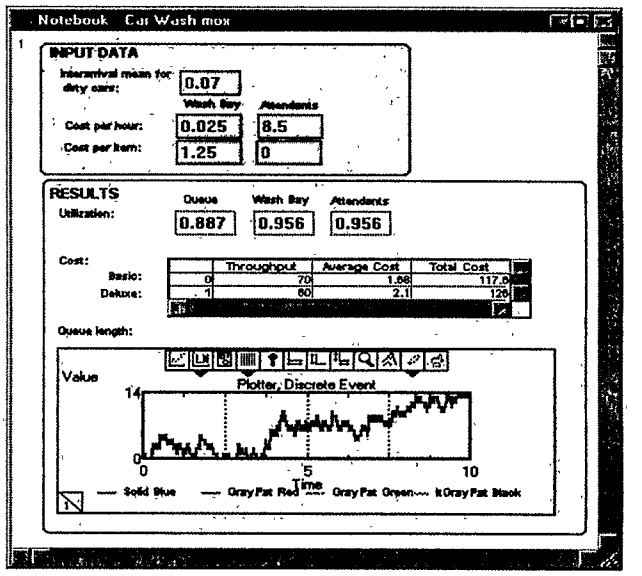

Figure 11: Notebook for Car Wash Model

4.4 Block Development

The block development environment is one of Extend's most powerful features. While the majority of Extend's users find the pre-built constructs sufficient for their needs, the block development environment provides a way for users to expand the modeling capabilities to perform unusual or highly specialized tasks.

Extend's open architecture allows you to access the structure of any block that is shipped with Extend. By opening the structure, you may edit the icon, dialog, help text, and programming code of the block. You can modify the interface and functionality of any block or create a new block from scratch.

ModL is the powerful and flexible language used to define the behavior of the block. This language provides high-level functions and features while having a familiar look and feel for users with experience programming in C. In addition, external XCMDs and DLLs can be called from within ModL giving you the option of programming in any language.

This level of extensibility has prompted many users to develop libraries of custom blocks for specific industries. Users and third-party developers have created libraries for modeling many systems including neural networks, control systems, bulk manufacturing systems, chemical processes, silicon wafer fabrication, pulp and paper mills, and radio and microwave communication systems.

4.5 Scripting

Since Extend was created from the ground up as a graphical simulation tool, much of the process of defining a model was originally dependent on user interaction. For example, the user places blocks on the model worksheet, connects blocks together by drawing a connection between the two, defines the block's behavior by double-clicking the block to open its dialog and filling the appropriate parameters, etc. Scripting is a feature that allows models to be created and/or modified through a suite of ModL functions. With this functionality, users can create their own objects that can automatically build and modify models. With scripting, users can develop their own model building "wizards" or self-modifying models. Without having to rely on general-purpose "wizards" provided by the software vendor, users can develop "wizards" specific to their needs and can have complete control over the level of detail and accuracy resultant from automated model building.

Coupled with Extend's ability to communicate with other applications using interprocess communication (IPC), scripting provides an easy way to allow other applications to control every aspect of Extend including building the model, importing/exporting data, and running the simulation.

5 SUMMARY

As demonstrated above, the ease-of-use of a graphical simulator and the power and flexibility of a simulation language are not mutually exclusive. By providing an intuitive interface along with an extensive authoring and development environment, Extend has succeeded in combining the best of both simulation worlds.

REFERENCES

Imagine That, Inc. 1997. *Extend Software Manual.* San Jose, CA.

Rivera, Jim. 1997. Modeling with Extend, *1997 Winter Simulation Conference Proceedings,* ed. S. Andradóttir, K. Healy, D. Withers, B. Nelson, 674-679. IEEE, Piscataway, NJ.

AUTHOR BIOGRAPHY

JIM RIVERA is a senior software developer with Imagine That, Inc. whose responsibilities range from the development of the Extend program and its libraries to manual writing and technical support. Mr. Rivera received a BS in 1993 in Electrical Engineering from Michigan State University. Prior to joining Imagine That, Inc., Mr. Rivera worked as design and analysis engineer with the Nuclear Power Department at Wisconsin Electric and as a design engineer with the Powertrain Division of General Motors.