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

THE EXTEND SIMULATION ENVIRONMENT

David Krahl

Imagine That, Inc.
6830 Via Del Oro, Suite 230
San Jose, CA 95119, U.S.A.

ABSTRACT

The Extend simulation environment provides an integrated structure for building simulation models and developing new simulation tools. This environment supports simulation modelers on a wide range of levels. Model builders can use Extend's pre-built modeling components to quickly build and analyze systems without programming. Simulation tool developers can use Extend's built-in, compiled language, ModL, to develop new reusable modeling components. All of this is done within a single, self-contained software program that does not require external interfaces, compilers, or code generators.

1 INTRODUCTION

Over the last decade, there has been a convergence in the simulation industry. Simulation languages have become easier to use, often adding a user interface layer similar to that traditionally found in simulators. Simulators have added functionality to the point where their power and flexibility rival that of traditional languages. Because of this, it has become difficult for an individual to determine the advantages of one product over the others based strictly on a feature comparison.

In this confusing marketplace, Extend stands out as a product whose basic design provides a combination of unparalleled ease of use, power, and extensibility (Krahl 1999). 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 to enhance productivity and ease of use; again no coding is necessary.
- A development environment for building sets of custom reusable and integrated components. This programming environment allows the modeler to create their own simulator.

2 SIMULATION PHILOSOPHY

Extend is the first simulation program to successfully bring simulation to the desktop. Originally released in 1988, Extend brought capabilities to analysts that were previously available only on mainframe computers. The vision which lead to the first graphical user-interface based simulation environment continues today.

Imagine That! believes that working with its customers is the best way to guarantee success. Technical support to those who have purchased the full version of Extend is generous in its scope. This includes all aspects of the simulation process including installation, questions about specific features, troubleshooting model problems, programming in the ModL language, and even if they are just seeking guidance in how best to build their model.

In the process of developing and enhancing Extend, Imagine That! has scored a number of "firsts" in the simulation industry. Table 1 illustrates a few of the pioneering features in Extend.

Table 1: Firsts for the Extend Simulation Software

Year	Innovation
1988	First template-based (library) simulation system
1988	First open source modeling components
1988	First simulation software designed for a GUI
1992	First hierarchical modeling environment
1992	First message based discrete event architecture
1995	First Windows/Macintosh simulation system
1998	First DDE scriptable simulation environment
2001	First open source optimizer
2001	First drag and drop ActiveX support
2001	First integrated support for Proof Animation
2001	First integrated network communication support

In fact, Extend continues to be unique in its use of open source and an integrated development environment. This allows model builders to create their own components based on the standard components from Imagine That!

Table 2: The Extend Product Family

Extend Product	**Description**	**Typical use**
Extend	Drag and drop simulation for continuous models	Continuous modeling of scientific and engineering systems
Extend+BPR	Business process reengineering package	Modeling business processes
Extend+Manufacturing	Advanced discrete event modeling components	Manufacturing, healthcare, and communications
Extend+Industry	Adds an integrated database and high speed systems modeling to Extend+Manufacturing	High speed processes and complex systems where it is useful to separate the model data from the structure
Extend Suite	Proof Animation and Stat::Fit as well as the BPR and Manufacturing modules	Organizations which need to model complex processes and build high quality animations
Industry Suite	Extend Suite and the added benefit of the Industry module	The ultimate in simulation support, combining continuous, discrete event, rate, embedded database, and sophisticated animation in one package

Imagine That! is also notable for what it does not do. We do not provide consulting services, as that would distract us from our primary mission of developing high quality software. Our sales processes are simple and straightforward, performed directly from our San Jose, CA office. This allows us to price Extend more competitively.

3 EXTEND PRODUCTS

The Extend product family is designed to meet the needs of the entire enterprise. Table 2 illustrates the range of Extend based products sold directly by Imagine That! In addition to these, third-party developers have created their own vertical market modules in diverse areas such as chemical processing, supply chain, pulp and paper manufacturing, and others. All products based on Extend include:

- Drag and drop modeling using the Extend built-in modeling components.
- A full suite of interprocess communication tools for communicating with other application such as Microsoft Excel.
- Hierarchical modeling capabilities.
- Evolutionary optimization.
- Animation.
- The innovative ModL language for development of vertical market simulation solutions.

4 THE EXTEND MODELING ENVIRONMENT

Before looking into how Extend can be used to build models, it is helpful to understand the Extend modeling environment (Imagine That, Inc. 2001)

Extend models are constructed with library-based iconic blocks. Each block describes a calculation or a step in a process. Block dialogs are the mechanism for entering model data and reporting block results. Blocks reside in libraries. Each library represents a grouping of blocks with similar characteristics such as Discrete Event, Plotter, 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. Figure 2 illustrates the overall structure of an Extend model.

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.

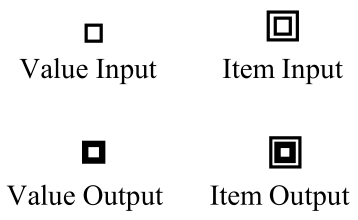
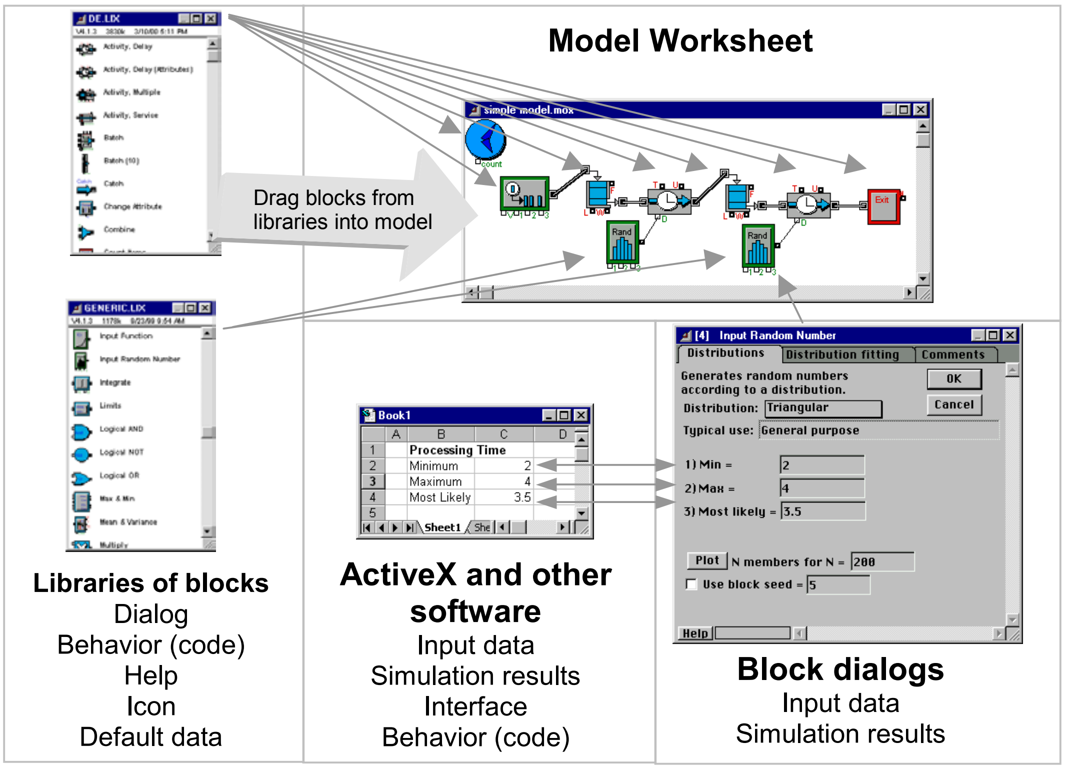


Figure 1: Value and Item Connectors



	A	B	C	D
1		Processing Time		
2		Minimum	2	
3		Maximum	4	
4		Most Likely	3.5	
5				

Generates random numbers according to a distribution.
Distribution: Triangular
Typical use: General purpose
1) Min = 2
2) Max = 4
3) Most likely = 3.5
N members for N = 200
Use block seed = 5

Figure 2: Extend Modeling Structure

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. The concept of value connections in addition to item connections is unique to Extend. Other contemporary simulation applications require that a function be written whenever a simulation input is based on a value from another point in the model. In Extend, this type of logic is performed without programming of any type. More importantly, the logic of the model is visible to anyone examining the model structure.

Figure 2 illustrates the relationship between the libraries, blocks, worksheet and any external programs (such as an ActiveX object, Excel, or a DLL) which may be linked to Extend. It also shows the visual nature of an Extend model. Note that the Input Random Number blocks can be clearly recognized as providing the delay (D connector) for the activities.

5 SINGLE SERVER, SINGLE QUEUE EXAMPLE

The following example is of a single server, single queue system. For the purpose of illustration, the model of a car wash will be used. This car wash will include one wash bay and one waiting line. The model for this car wash is shown in Figure 3.

The block on the far left is a Generator block that 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 wash bay is represented by the Activity Delay block with a limited capacity of one processing unit. The delay for the activity is specified by an Input Random Number block (connected to the “D” or delay connector). Each time a car arrives to the activity, a new value is sampled from the Input Random Number block. The last block in the model is an Exit block that removes the cars from the system.

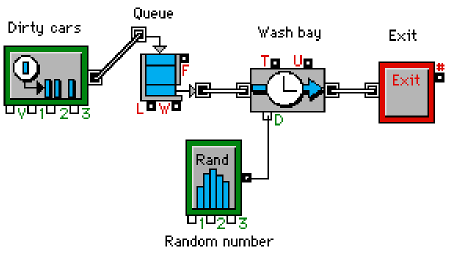


Figure 3: A Single Server Single Queue Model

5.1 Graphical Output

A Discrete Event Plotter graphically displays model metrics (values). In this example (Figure 4), the Plotter will graph the contents of the Queue (the number of dirty cars waiting in line) over time. Here the length connector (L) on the Queue FIFO is connected to an input on the Plotter. Figure 6 illustrates a sample plot from this model.

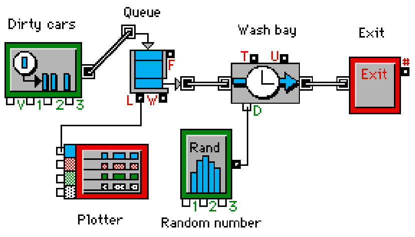
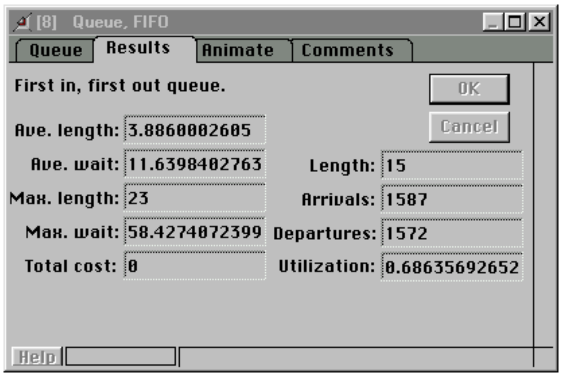


Figure 4: Discrete Event Plotter Added to Model

5.2 Model Results

During and after the simulation run, the results of the simulation are reported within the blocks, displayed on plotters, sent to reports, and exported to other applications. Double-clicking on each block reveals the information collected from the simulation run. For example, double-clicking on the Queue, FIFO block opens a dialog showing the following information about the state of the block:


[8] Queue, FIFO

Queue | Results | Animate | Comments

First in, first out queue.

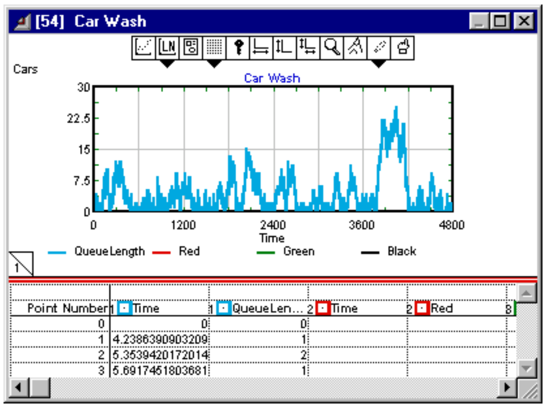
OK | Cancel

Ave. length:	3.8860002605		
Ave. wait:	11.6398402763	Length:	15
Max. length:	23	Arrivals:	1587
Max. wait:	58.4274072399	Departures:	1572
Total cost:	0	Utilization:	0.68635692652

Help

Figure 5: Dialog of Queue FIFO

The Plotter block shows the number of items stored in the Queue, FIFO over time in both graphical and tabular format:



Point Number	1 Time	1 QueueLen...	2 Time	2 Red	8
0	0	0			
1	4.2386390903209	1			
2	5.3539420172014	2			
3	5.6917451803681	1			

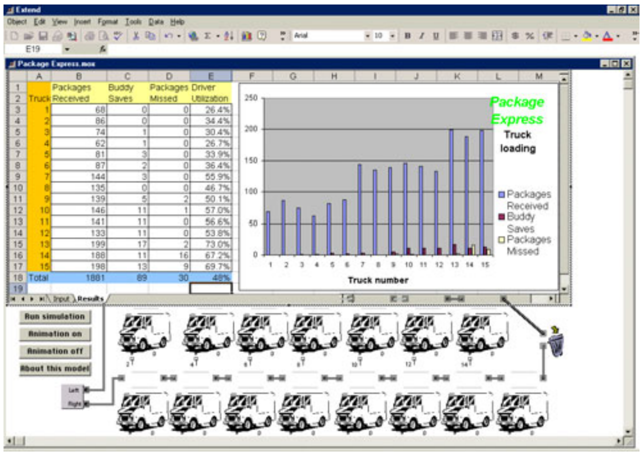
Figure 6: Plot of Queue Length

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 even used to control some aspect of the outside world through external device drivers.

5.3 Communication with Other Applications

The term interprocess communication (IPC) describes the act of two applications communicating and sharing data with one another. This feature allows the integration of external data and applications into and out of Extend models. Automatic communication between Extend and other applications can take one of five forms:

- "Paste-Link" where the information is automatically updated between Extend and Excel. Setting up this type of communication only requires copying the value in one application (Extend or Excel) and selecting paste-link in the other application. This produces a "live" link that updates whenever the value in the host application changes. These updates even occur when the model is running and can be used to display data or graphs in Excel that "animate" while the simulation model is running.
- Blocks that utilize the IPC functions to communicate directly with other applications. The IPC library in Extend allows models to send data to, get data from, and execute macros within other applications, including Excel spreadsheets. These blocks can respond to simulation events and traverse the spreadsheet.
- ODBC (Open DataBase Connectivity). Extend can access database information through ODBC. As with all of Extend's interprocess communication features, this is available both on a block level (accessible with no programming required) and on an API level within Extend's ModL programming environment.
- Embedded ActiveX or OLE (Object Linking and Embedding) objects. These retain their native user interface, but reside with the Extend model worksheet or blocks. All of the features and the interface of the embedded application are directly available within Extend. Figure 7 shows an ActiveX Excel spreadsheet embedded in an Extend model. Extend's paste-link functionality was used to connect the cells in the spreadsheet to the dialogs in the Extend blocks.
- DLL (Dynamic-Link Library). A separate application in the form of a DLL can be called from the ModL code. This library can be written in any one of a number of popular programming languages including FORTRAN, C, C++, or Pascal.



	Packages	Buddy	Packages	Driver
Truck	Received	Saves	Missed	Utilization
1	68	0	0	26.4%
2	86	0	0	34.4%
3	74	1	0	30.4%
4	62	1	0	26.7%
5	81	3	0	33.9%
6	87	2	0	36.4%
7	144	3	0	55.9%
8	135	0	0	46.7%
9	139	5	2	50.1%
10	146	11	1	57.0%
11	141	11	0	56.6%
12	133	11	0	53.8%
13	199	17	2	73.0%
14	188	11	16	67.2%
15	198	13	9	69.7%
Total	1881	89	30	48%

Figure 7: ActiveX Excel Spreadsheet Embedded in Extend

5.4 Integrated Database

The Extend+Industry package contains an integrated relational database. This database provides a complete data management system for model input and output. The database is built directly into the model to house product data, process information, and experiment with scenarios.

By separating your data from the model, the database enables fast scenario implementation, flexible analysis and improved project management.

- Configure tables for experiments and reports.
- Use database-aware blocks to build powerful model constructs.
- Assign strings to items using database-aware attributes.
- Leverage dates, times and other data formats such as currency.

The Industry database is relational and parent-child relationships can be used to better organize the information in the model. For example, each entry in a table of part types can reference its own unique routing table. This is an extremely powerful feature for organizing the information used in complex simulation models.

5.5 Data Analysis

Extend offers a number of methods for analyzing both input and output data. These range from internal analysis features to built-in interfaces with other applications.

An interface to distribution-fitting programs is provided to 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. For example: to determine how sensitive the car wash is to changes in the inter-arrival time of dirty cars, sensitivity analysis can be performed 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, the change in the parameter value from one run to the next is defined. Simulation parameters such as the number of runs and simulation end time can be specified in the Simulation Setup dialog under the Run menu. By cycling through different inter-arrival times for the dirty cars and comparing the results from the different runs, an understanding of how sensitive the car wash is to the arrival rate of dirty cars can be obtained.

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

5.6 Optimization

Until now, optimization of a simulation model has been a process of trial and error. Extend's Evolutionary Optimizer employs powerful "enhanced evolutionary" algorithms to determine the best possible model configuration. Using a drag and drop interface, performance metrics and parameters that can be varied are entered into the Optimizer block. These parameters are used in an equation that defines the objective function. When the model is run, the Optimizer block generates alternatives and locates the statistically best configuration. Unlike external optimizers, Extend's optimization is well integrated into the program. For example, when the optimization process is complete, model parameters are automatically set to the optimal configuration. In addition, because the optimizer has been implemented in a block, the source code is available for examination and modification.

6 CUSTOMIZING EXTEND

The above discussion illustrates the highly graphical and interactive nature of Extend. However, Extend can also take the shape of the modeled system. Interfaces, components, and graphics can be created 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. This means that 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 facilitate customization.

6.1 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, create their own in an external drawing package, or import them.

For example, adding animation to show cars traveling from block to block in the car wash model is done by selecting the appropriate icon in the Animate tab of the Generator block. From here, the picture representing all of the items created by the Generator is defined. Each block that the items pass through has the capability of changing the item's animation icon. Every item exiting the Generator block can be represented with a picture of a dirty car. As the items pass through the wash bay, the Activity Delay block changes each item's animation picture to a clean car, 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, and pixel maps.

An interface also exists between Wolverine Software's animation package, Proof Animation™ and Extend. Activities, Resources, Generators, and Exit blocks each have specific functionality to send information to the Proof animation during simulation execution (Wolverine Software Corporation 1995). Additional animation features in Proof can be accessed in Extend through the Proof library of blocks. This allows Extend modelers to easily utilize the industry's most sophisticated animation package.

6.2 Hierarchical Modeling

In the past, there have been at least two definitions of hierarchy in simulation modeling. The first definition, coined by Imagine That, Inc in 1992 describes the grouping or aggregation of system components (blocks) into a single object. The second definition, first referred to later in 1992, (Pegden and Davis 1992) describes a modeling system in which new primitive modeling constructs can be created from existing primitives provided by the simulation system. It appears that, as other simulation software packages have matured and added features, the first definition has become the standard (Bapat and Swets 2000).

Extend provides unlimited layers of hierarchy, created using a simple menu command. Hierarchy allows models to be subdivided into logical components or sub-models, represented by a single descriptive icon. Double-clicking on the hierarchical block opens a new window displaying the sub-model. This greatly simplifies the representation of a model and allows the user to hide and show model details as appropriate for the target audience.

Even a medium-sized call center model can become difficult to maintain if all of the modeling components must be at the same level. Extend's hierarchy allows the modeler to decompose the model into smaller, more manageable segments. Additionally, new model segments can be added by dropping in a new hierarchical block. Figure 8 illustrates the use of hierarchy to organize a model where each icon encapsulates a separate model segment.

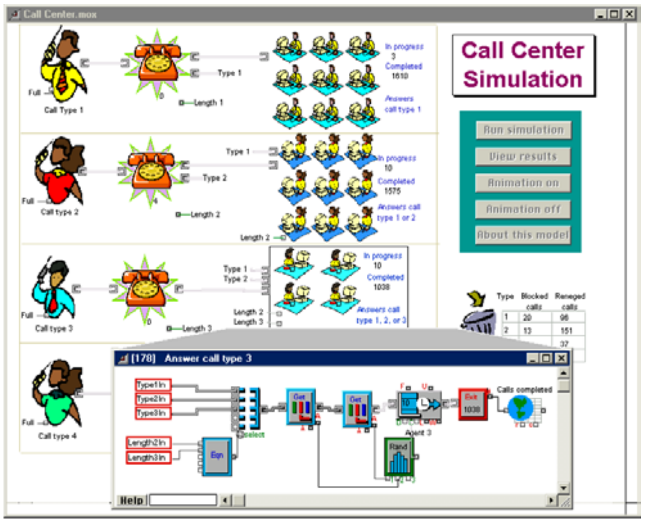


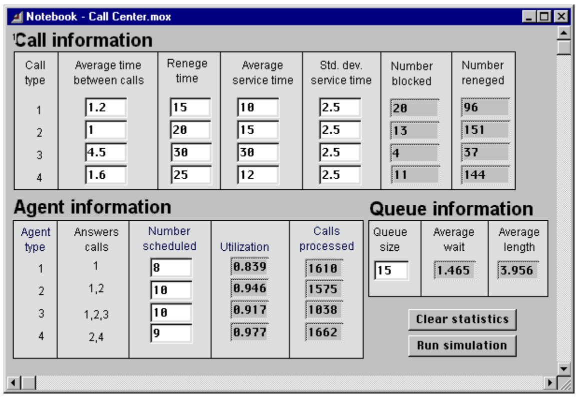
Figure 8: Call Center Model with Hierarchical Blocks

By selecting a group of blocks and choosing Make Selection Hierarchical from the Model menu, a section of the model can be encapsulated within a hierarchical block. Extend's hierarchy fully encapsulates the enclosed block and does not require the renaming of variables and connections. All of the connection names within the hierarchical block are local to that block. This allows multiple instances of identical hierarchical blocks in the same model (Pidd and Castro 1998). The hierarchical blocks can be copied within a model or saved to a library to be used again 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. Figure 8 shows the Call Center model with hierarchical blocks encapsulating the major operations of the model. While the representation of the model is more intuitive and simple than a non-hierarchical model, all of the detail of the model can still be accessed by double-clicking on any of the hierarchical blocks to display the underlying sub-model.

6.3 Dialog Cloning and the Notebook

As noted earlier, 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 cumbersome. This is especially true when working with large models containing many layers of hierarchy. An effective way of dealing with this is to use the Extend notebook and cloning feature. With the notebook, a single custom interface can be created that consolidates critical parameters, results, and model control 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 are created by selecting the Cloning Tool from the tool bar and using it to drag 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 vice-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. As an illustration of how the notebook can be used to consolidate important parameters into one location, Figure 9 shows the notebook for the Call Center model.


Notebook - Call Center.mox

Call information

Call type	Average time between calls	Renege time	Average service time	Std. dev. service time	Number blocked	Number reneged
1	1.2	15	10	2.5	20	96
2	1	20	15	2.5	13	151
3	4.5	30	30	2.5	4	37
4	1.6	25	12	2.5	11	144

Agent information

Agent type	Answers calls	Number scheduled	Utilization	Calls processed
1	1	8	0.839	1610
2	1,2	10	0.946	1575
3	1,2,3	10	0.917	1038
4	2,4	9	0.977	1662

Queue information

Queue size	Average wait	Average length
15	1.465	3.956

Figure 9: Notebook for Call Center Model

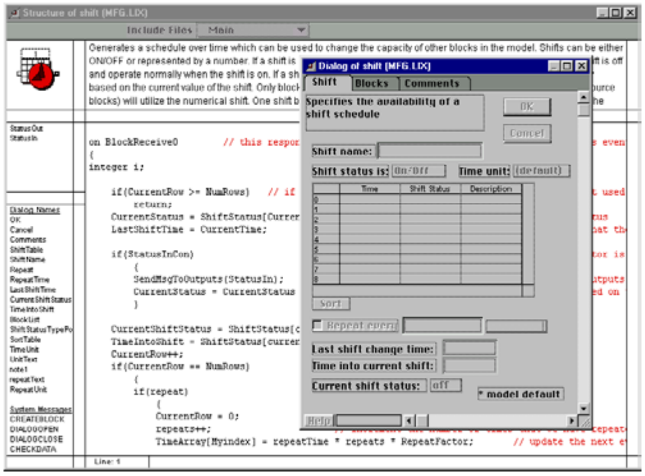
6.4 Block Development

The block development environment is one of Extend's most powerful features. While the majority of Extend users find the pre-built constructs sufficient for their needs, the block development environment provides a way for users to expand their modeling capabilities to perform unusual or highly specialized tasks. It typically takes only minutes for someone with any programming experience to learn the basics of building modeling components in Extend.

Extend's open source architecture allows access to the structure of most blocks that are shipped with Extend. By opening the structure, the icon, dialog, help text, and programming code of the block can be edited. The interface and functionality of any block can be modified or a new block created from scratch.

ModL is the powerful and flexible language used to define the behavior of each 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 the option of programming in any language which supports this feature (such as C or Pascal).

The ModL development environment with its interface for editing the dialog, help, icons, connectors, and code, is illustrated in Figure 10. Other tools include block performance profiling, "include" files, and an interactive debugger.



Generates a schedule over time which can be used to change the capacity of other blocks in the model. Shifts can be either ON/OFF or represented by a number. If a shift is
and operate normally when the shift is on. If a sh
based on the current value of the shift. Only block
blocks) will utilize the numerical shift. One shift b

Specifies the availability of a shift schedule

Shift name:
Shift status is: On/Off
Time unit: (default)

	Time	Shift Status	Description
0			
1			
2			
3			
4			
5			
6			
7			
8			

Repeat every
Last shift change time:
Time into current shift:
Current shift status: off
* model default

```
StatusOut
StatusIn

Dialog Names
OK
Cancel
Comments
ShiftTable
ShiftName
Repeat
RepeatTime
LastShiftTime
CurrentShiftStatus
TimeIntoShift
BlockList
ShiftStatusTypePo
SortTable
TimeUnit
UnitText
note1
repeatText
RepeatUnit

System Messages
CREATEBLOCK
DIALOGOPEN
DIALOGCLOSE
CHECKDATA
```

```
on BlockReceive0        // this respor
{
integer i;

    if(CurrentRow >= NumRows)    // if
        return;
    CurrentStatus = ShiftStatus[Currer
    LastShiftTime = CurrentTime;

    if(StatusInCon)
        {
        SendMsgToOutputs(StatusIn);
        CurrentStatus = CurrentStatus
        }

    CurrentShiftStatus = ShiftStatus[c
    TimeIntoShift = ShiftStatus[currer
    CurrentRow++;
    if(CurrentRow == NumRows)
        {
        if(repeat)
            {
            CurrentRow = 0;
            repeats++;
            TimeArray[Myindex] = repeatTime * repeats * RepeatFactor;       // update the next e
```

Figure 10: ModL Block Development Environment

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 high-speed production systems, chemical processes, silicon wafer fabrication, pulp and paper mills, environmental processes, and radio and microwave communication systems. Some blocks coded by customers can be found on the company web site (http://www.imaginethatinc.com).

6.5 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 them, defines the block's behavior by double-clicking the block to open its dialog and entering 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 objects that 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 resulting 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.

7 DISCRETE EVENT ARCHITECTURE

Extend utilizes a message-based architecture which allows for more natural model building than is possible in other simulation tools. Messages are used to pass information to connected blocks about the state and actions of the block sending the messages. For example, as soon as a queue receives an item it will send a "wants" message to the downstream blocks to see if any of them can accept an item. The messaging system is applied to the item as well as the value connectors. Because of this, complex models and logic can be built without resorting to "dummy" resources or "synthetic" workstations

In using a modern, message-based system, Extend allows the modeler to focus on the modeling task rather than the simulation tool.

- Complex model segments can be built from simple, elemental blocks. These segments can then be saved in a library for use in other models. This type of model construction eliminates the need for "kitchen sink" modeling components in which every possible permutation must be programmed by the developer (making the interface unnecessarily complex) or requiring programming to enhance the capabilities of the modeling component.
- Easier rescheduling of events. Because blocks, not items (entities), are entered into the event calendar, changing an event time is a simple assignment. In other simulation tools, the event calendar must be searched for a specific item before the change can be made.
- Events do not have to be item based. Blocks can post themselves on the event calendar even if they do not handle items. This reduces the overhead in the model because items do not have to be generated or processed when an event occurs.
- Blocking can be done through decisions. Extend automatically determines which path an item takes before it arrives at the decision point. The alternative to this would be adding "dummy" resources to prevent the item from moving forward if space was not available.
- Queues can be separated from activities. Any number of blocks that do not hold items (passing blocks) can be positioned between a queue and the next activity.
- Conditions do not need to be "time checked". Messages are sent to connected blocks whenever a condition changes and the condition is evaluated immediately.
- Model logic is represented graphically and is visible as part of the model structure.
- User-defined "sub-executives" can be added to react to specific simulation events. For example, a block could be connected to the Executive that checked a condition at the end of every simulation event.

8 WHAT MAKES EXTEND UNIQUE

Extend provides features and capabilities not found in other simulation software. This allows the modeler to concentrate on the modeling process and quickly produce a model that is easy to manipulate and communicate to others. These features include:

- An integrated development environment for building modeling components that fit naturally into the user interface.
- Graphical logic that makes the model easier to understand and communicate.
- An unparalleled level of interactivity. Model parameters can be changed and results viewed during simulation execution. This is done through the graphical user interface; there is no need to enter a debugging mode or enter cryptic debugging commands.
- Superior hierarchy. Extend's hierarchy allows for animation and reuse, and can be any number of levels deep. This gives modelers an excellent tool for organizing large models and reusing model segments in other models.
- An innovative discrete event architecture which makes model building more intuitive.

9 APPLICATIONS

Since Extend is a general purpose simulation program, it has been applied in a wide range of areas. The two application examples included here are medical laboratory automation and supply chain management.

9.1 Medical Laboratory Equipment

The Medical Laboratory model shown in Figure 11 simulates an entire lab analysis operation. While the model looks like just a picture of an analyzer, in reality the "picture" is an Extend hierarchical block. Double-clicking on the Immulite 2000 icon reveals a lower-level layer composed of three additional hierarchical blocks joined together much like a jigsaw puzzle ("Taking a Closer Look"). The Pre-analytical, Analytical, and Post-Analytical blocks each simulate their portion of the lab's operation. The user can change aspects of the model to more closely resemble their actual operations, and explore opportunities for improving testing processes.

Opening (double-clicking) a hierarchical block reveals its underlying model. For example, the Pre-analytical section models specimen receipt, including STAT and routine accessions, accession number, specimen receipt time, and the number of tests to be performed. The Analytical hierarchical gathers information about the number of accessions and tests performed by the analyzer and determines turn-around-times and utilization for all instruments. The Post-Analytical block exports pertinent data to a Microsoft Excel spreadsheet for analysis and reporting. Communication between the model and the spreadsheet is handled automatically by Extend's IPC (InterProcess Communication) library. This application was created by Solution Consulting Service.

Figure 11: Medical Laboratory Model

9.2 Supply Chain Management

The US Marine Corps is undergoing a revolution in the way they conduct combat operations. There are many ideas regarding how the tactical supply chain must change to provide the necessary logistical support. The challenge has been described as "warehouses that move", referring to the changing location of supply ships and depots (Hamber 1999).

The TLoaDS model (Figure 12) has been developed to explore the ability of existing, evolutionary, or revolutionary methods and equipment for this mission.

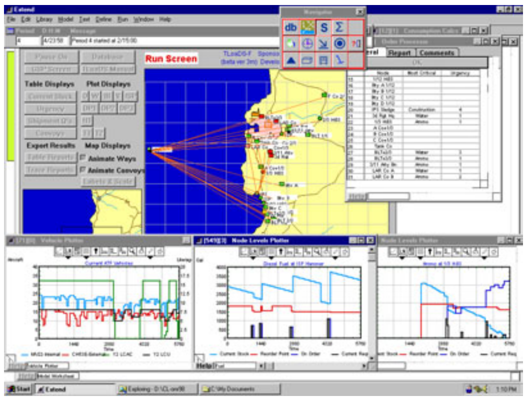


Figure 12: Supply Chain Model

10 SUMMARY

As demonstrated above, Extend's design provides a superior simulation environment. By incorporating an intuitive interface, an extensive authoring and development environment, and a more advanced simulation technology, Extend has succeeded in defining its position as the leader in simulation software.

REFERENCES

Bapat, V. and Swets, N. 2000. The Arena Product Family: Enterprise Modeling Solutions. In *Proceedings of the 2000 Winter Simulation Conference Proceedings,* ed. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 163-169. IEEE, Piscataway, NJ.

Hamber, R. 1999. CloaDS & TloaDS. *1999 Simulation Solutions Conference.* Institute of Industrial Engineers, Norcross, GA.

Imagine That, Inc. 1992. *Extend Software Manual*. San Jose, CA.

Imagine That, Inc. 2001. *Extend User's Guide*. San Jose, CA.

Krahl, D. 1999. Modeling with Extend. In *Proceedings of the 1999 Winter Simulation Conference Proceedings,* ed. P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, 188-195. IEEE, Piscataway, NJ.

Pegden, C. D. and Davis, D. C. 1992. Arena: A SIMAN/Cinema-Based Hierarchical Modeling System. In *Proceedings of the 1992 Winter Simulation Conference,* ed. J.J. Swain, D. Goldsman, R. C. Crain, J. R. Wilson, 390-399. IEEE, Piscataway, NJ.

Pidd, M and Castro, RS. 1998. Hierarchical Modeling in Discrete Simulation. In *Proceedings of the 1998 Winter Simulation Conference,* ed. D. J Medeiros, E. F. Johnson, J. S. Carson, M. S. Manivannan, 383-389. IEEE, Piscataway, NJ.

Wolverine Software Corporation. 1995. *Using Proof Animation.* Annandale, VA.

AUTHOR BIOGRAPHY

DAVID KRAHL, Lead Engineer with Imagine That, Inc., is responsible for Extend block architecture, development, and support. He received a MS in Project and Systems Management in 1996 from Golden Gate University and a BS in Industrial Engineering from the Rochester Institute of Technology in 1986. Mr. Krahl has worked extensively with a range of simulation programs and is actively involved in the simulation community. His email and web addresses are `<davek@imaginethatinc.com>` and `<www.imaginethatinc.com>`.