2 Simulation Languages

Simulation languages are versatile, general purpose classes of simulation software that can be used to create a multitude of modeling applications. In a sense, these languages are comparable to FORTRAN, C#, Visual Basic.net or Java but also include specific features to facilitate the modeling process. Some examples of modern simulation languages are GPSS/H, GPSS/PC, SLX, and SIMSCRIPT III. Other simulation languages such as SIMAN have been integrated into broader development frameworks. In the case of SIMAN, this framework is ARENA. Simulation languages exist for discrete, continuous and agent-based modeling paradigms. The remainder of this book will focus on the discrete event family of languages.

2.1 Simulation Language Features

Specialized features usually differentiate simulation languages from general programming languages. These features are intended to free the analyst from recreating software tools and procedures used by virtually all modeling applications. Not only would the development of these features be time consuming and difficult, but without them, the consistency of a model could vary and additional debugging, validation and verification would be required. Most simulation languages provide the features shown in Table 2.1.

1) Simulation clock or a mechanism for advancing simulated time.
2) Methods to schedule the occurrence of events.
3) Tools to collect and analyze statistics concerning the usage of various resources and entities.
4) Methods for representing constrained resources and the entities using these resources.
5) Tools for reporting results.
6) Debugging and error detection facilities.
7) Random number generators and related sets of tools.
8) General frameworks for model creation.

Table 2.1 Simulation Language Features

2.1.1 Comparison to Traditional Languages

Although many models are written using simulation languages, some analysts still prefer to rely on traditional programming languages for model development. In other cases, extensions to a language are developed to add capabilities to a traditional language. For instance, Repast Simphony is a free and open source, agent-based modeling toolkit that adds features to Java in order to simplify model creation and use. This blended approach provides the advantages of both the traditional language and the simulation modeling extensions. The motivations behind using a general purpose language include:

Programmer familiarity: Developers already know the general purpose programming language. They may not have the time or inclination to learn a simulation language.

Flexibility: Programming languages inherently are flexible, giving the analyst freedom to create the model using his or her preferred methodology.

Cost: Programming language software is usually more accessible and far less expensive than specific simulation software. This may not always be true since several leading simulation languages can be downloaded for no cost. However, other leading simulation language packages can be very expensive.

Hardware Concern: General purpose software may be available on any hardware platform while some simulation languages may require special machines and memory configurations.

Lack of Analyst Knowledge: The analyst may not understand simulation languages and may lack knowledge on the advantages of using a simulation language package.

Training: Available classes in the use of traditional languages are more likely to be available than specialty simulation training.

Although traditional languages do offer some advantages, most of these are outweighed by features standard to many simulation languages. In a typical modeling application, the programmer or analyst will find the initial investment in a simulation language more than pays off. A simulation language will provide a savings in coding, debugging, analysis of results, and in making changes.

2.1.2 Simulation Languages

A variety of simulation languages exist and are used by businesses, researchers, manufacturing and service companies, and consultants. The next sections briefly discuss two common simulation languages: GPSS and SIMSCRIPT.

GPSS: General Purpose Simulation System (GPSS) was originally developed by Geoffrey Gordon of IBM and released in October of 1961. Following IBM's release of GPSS to the public domain, it became a multivendor simulation language and has been in continuous use since.

In general, GPSS enjoys widespread popularity due to its sensible world view and overall power. Its basic functions can be easily learned while powerful features make it ideal for modeling complex systems. In general, GPSS is used to simulate queuing systems that consist of customer entities interacting and completing in a system of constrained resources. The resources are structured as networks of blocks that entities (also called transactions) enter and use to perform various tasks in certain amounts of simulated time. As entities move through these networks, which have been organized to represent a real world system, statistics are collected and used to determine if the system contains bottlenecks, is over or under utilization, or exhibits other characteristics. Output data is made available for analysis at the end of a production run.

Presently, several vendors offer versions of GPSS. Included are:

> Wolverine Software which produces GPSS/H, a powerful, state-of-the-art version of GPSS engineered to allow creation of large, complex models (http://www.wolverinesoftware.com).
>
> Minuteman Software which produces a user friendly GPSS simulation environment called GPSS World that features special model development tools (http://minutemansoftware.com).
>
> ngolf Ståhl and Beliber AB which produce WebGPSS, a stream-lined version of GPSS, with a focus simulation and modeling concept education (http://www.webgpss.com).

SIMSCRIPT III: This language is a direct descendant of the original SIMSCRIPT language produced at Rand Corporation in the 1960s. SIMSCRIPT III has constructs that allow a modeler to approach a problem from either a process or an event oriented world view. SIMSCRIPT III offers unique features which add to its appeal. Among these are:

- Object-Oriented Programming
- Modularity
- SIMSCRIPT III Development Studio (SimStudio)
- Object-Oriented Simscript III graphics
- Data Base Connectivity SDBC

In general, SIMSCRIPT III is a free form language with English-like syntax. This syntax allows the code in the system to become self-documenting. Model components can be programmed clearly enough to provide an excellent representation of the organization and logic of the system being simulated. SIMSCRIPT III is maintained and distributed at http:// www.simscript.com by CACI Products Company.

2.1.3 How Simulation Languages Work

Most discrete event simulation languages model a system by updating the simulation clock to the time that the next event is scheduled to occur. Events and their scheduled times of occurrence are maintained automatically on one of two ordered lists: the current events chain or the future events chain. The current events chain keeps a list of all events that will (or may) occur at the present clock time. The future events chain is a record of all events that can occur at some point in the future. A simulation clock moves to the next event on the future events chain and changes the system state of the model based on that event's characteristics. Figure 2.1 illustrates.

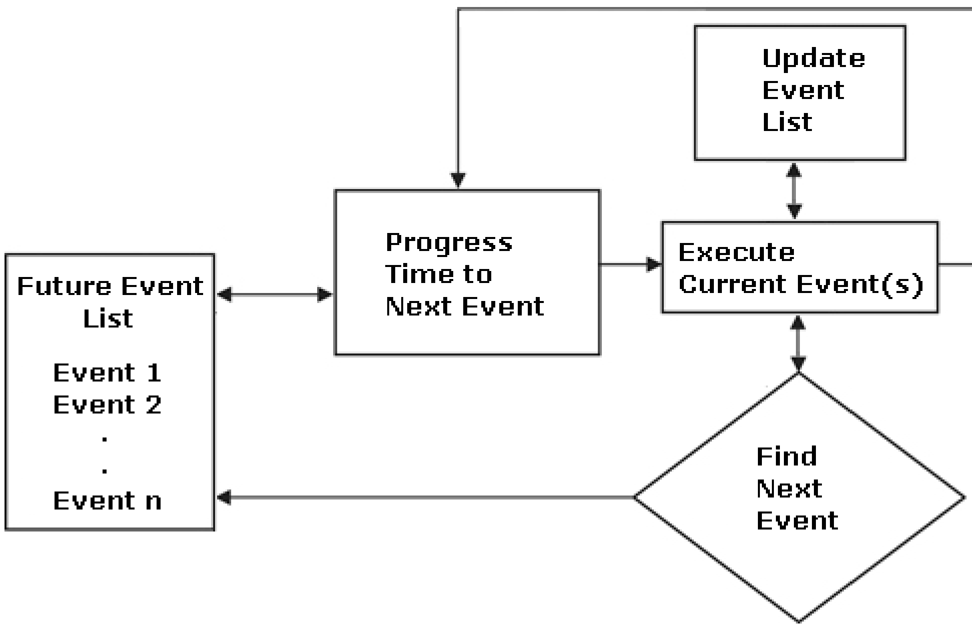


Figure 2.1 How Simulation Languages Work

Transactions are dynamic entities that traverse through the networks of blocks in a simulation. Their meaning is defined by the simulation analyst during the process of building the model. They are usually created to represent an active object such an automobile in a traffic simulation, an AGV in a manufacturing simulation, or a customer in a barber shop simulation.

Multiple transactions can exist in a model at any given time. They move through the block network as far as the current system state allows and stop either for a predetermined delay, a logical delay, or because a desired resource is currently unavailable. Parallel processes can be simulated since multiple transactions can be present in a model. Each transaction has associated with it a list of attributes which can be altered according to logic contained in the model. These parameters can be used to define unique characteristics of a particular transaction, thereby affecting its movement through the block networks.

More permanent equipment entities can be used to model constrained resources. The system state of the model at any given time is defined by the status and attributes of both permanent and temporary entities in the model.

2.2 Simulators and Integrated Simulation Environments

Simulators and integrated simulation environments have emerged to provide the analyst with additional capabilities that further automate and remove drudgery from the modeling process. Models that used to take days or weeks to develop with a simulation language can now be modeled in minutes with pre-developed representations of commonly modeled real-world items requiring no more than customization of certain parameters. New software, specialized to the application area, allows the analyst to model, execute, analyze, and animate systems in their domain (such as manufacturing, healthcare, logistics, communication, et cetera). Advanced versions of simulation software support the following:

- Environments specific to the domain area so the analyst can quickly enter the geometry, system characteristics and resource constraints for a model.
- Expert system technology uses input parameters to generate details automatically.
- Windows, help systems, and pop-up menus provide guidance and automate the modeling process.
- Facilities for comparing changes and statistical reporting tools are provided.
- Built in system specific templates make analysts more productive and eliminate programming time.
- Analysts have tools to validate, verify and test designs.
- Facilities are provided to automate and support exploration of "what if" questions being tested.
- 3-D animation graphics are automatically created from analyst inputs.
- Results can be communicated in real time using animation and statistics

2.2.1 Simulators

A simulator is defined as a user friendly software package that will develop a model for a particular application. These models generally are created by a person who is not a simulation analyst or programmer but still wishes to analyze a system. For example, the person who is an automation expert (rather than a computer programming expert) is able to model a conveyor system with pull-down menus and user friendly interface tools. MedModel (health care simulator), PRISM (police force simulator) and PX-Sim (pharmaceutical simulator) are all examples of simulator software packages. Simulators are often characterized by "buzz phrases" such as those shown in Table 2.2:

Simulator Buzz Phrases
No programming
Graphical model building interface
Model any system in just a few hours
Any manager can use it
Anyone on the shop floor can use it

Table 2.2 Simulator Buzz Phrases

2.2.2 Advantages of Simulators

Simulators offer several advantages over simulation languages. Included among these are:

Ease of Use – Simulators are designed specifically for the non-programmer. They typically have user-friendly features such as pull down command menus and special tools to simplify model construction. Many simulators are marketed with more emphasis on their user friendliness than on their underlying system modeling capability. They are meant for use by the domain expert.

Quick Model Development – Many simulators are set up to provide a fast method of constructing a model. This development speed is gained because the underlying system model has already been created and the user of the simulator is only changing parameters through a user interface. This interface may be set up to allow model construction through the piecing together of graphic icons, with a series of questions, or with series of prompts and user screens.

Base System Simulation Already Complete – At the center of all simulators is a model which has already been constructed. The user of the simulator, in theory, does not need to concern themselves with fully understanding and analyzing the system to be modeled. As long as several key attributes of the system can be identified, the simulator will do the work. This saves time, energy, and frustration on the part of the model builder.

Framework for Analysis of a Particular Type of System – Simulators have been used numerous times for the same task and therefore are constructed to provide a logical method of analyzing a particular type of system. This structure or experimental framework has evolved from repeated use and, in many cases, is superior to what a first time simulation analyst could devise. This framework is another time saving feature.

Although simulators may sound too good to be true, they offer a world of knowledge in return for little effort. When used in the proper context, they can be a useful tool. But when misused, they can provide misleading results. Some of the pitfalls of simulators are in the following list.

Oversimplification – Simulators tend to oversimplify system representation. In order to sell a simulator, a vendor must be able to apply his product to many comparable systems. This means a generic model forms the basis for the software and that it may have simplifying assumptions built in. One Automatic Guided Vehicle vendor evaluated several simulator packages only to discover the base model would not permit use of their advanced design concepts. Instead of displaying their edge over the competition, the simulator made their system like everyone else's.

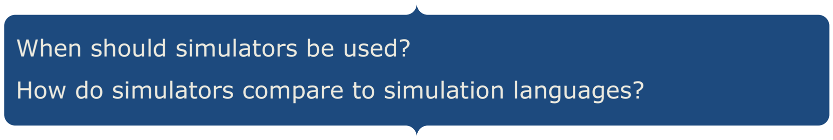
Inflexible System Representation – Simulators can be somewhat inflexible. The base model has already been built and with it are some general, unchangeable, underlying assumptions. A comment was once heard from an engineer concerning the use of a simulator to aid in system design. "I'm not going to throw out my concept because the simulator can't be changed to show how it will work," he said.

Encourages Jumping the Gun – Due to the nature of simulators and the ease with which models can be created, the modeler is often tempted to skip intermediate steps in the model creation process and immediately begin performing a quick analysis. This temptation is sometimes hard to overcome and can lead to the construction of invalid models.

Visual Component May Create a False Sense of Credibility – Most simulators, especially personal computer based ones, rely heavily on the use of graphics for model creation and output displays. This visual component may give the model users a "warm fuzzy feeling" but it shouldn't be unconditionally accepted as valid. All models should be analyzed using proper validation and verification techniques.

2.2.3 Simulators vs. Simulation Languages

The following questions come to mind when trying to decide whether to use a simulator or a more robust simulation language for modeling project (see Figure 2.2).



When should simulators be used?
How do simulators compare to simulation languages?

Figure 2.2 Simulator Questions

The following observations can be made:

1. Commercially available simulators should not be used for leading edge or specialty systems. In most cases, simulators are created for broad and general applications. If a modeler wants a simulation that exploits all specific attributes of his/her system to gain a competitive edge, a simulation language may provide a better fit for the project.
2. Simulators can make excellent "concept modeling tools." One of the steps in the simulation life cycle is the creation of a rough-cut or concept model. Simulators can be used to build quick models for feasibility studies.
3. Simulators can be used as estimating tools. The company that wishes to purchase a conveyor, for example, can use a simulator to provide a general estimate of what will be needed. A simulator will provide a good idea without actually getting too specific. This is particularly true when a product vendor has not yet been selected.
4. Use of a simulator does not automatically eliminate all work. Simulators reduce the time involved in model construction. The other steps in the simulation process still require time and expertise.

2.2.4 Integrated Environments

Most simulation languages and simulators are now part of broad, integrated environments intended to facilitate all aspects of the modeling process. Generally, integrated simulation environments include software tools for designing, writing, verifying, running, and analyzing models. Software environments started to appear in the simulation marketplace in the mid-1980s.

TESS (The Extended Simulation System) was one of the first integrated simulation environments to be offered. TESS originally ran in conjunction with several simulation languages and provided databases, graphics, and other integrated support features. GPSS World, produced by Minuteman software, is an example of a simulation language which has been incorporated as part of an integrated environment approach to modeling. This GPSS software package includes facilities for creation, debugging and verifying, running, analyzing, and animating of the model. Arena is another integrated modeling environment specifically configured to support manufacturing, production, and logistics.

Integrated environments allow the simulation analyst to easily move from one step in the simulation process to the next without much of the unproductive housekeeping work required by non-integrated simulation tools.

2.3 Hardware Requirements for Simulation

Computer hardware technology has advanced tremendously in the past decade. The computing power once found only in a room sized mainframe can now be carried in a small pocket sized computer. Execution speeds and memory capacity have increased while device sizes have decreased. All of these factors have helped foster an environment well suited to modeling applications.

While large scale scientific simulation work still relies on supercomputers, most manufacturing, service sector, and specialty simulators run on microcomputer platforms. Other large scale models may be run over the Internet with host computers providing required processing power. But in general, most simulation software vendors produce packages focused on Windows or Linux-based operating systems.

Microcomputers offer the ability to share simulation libraries over networks, produce high resolution graphics, and run large simulation programs. In addition, the availability of other software such as spreadsheets, statistical analysis packages, and standard programming languages gives the simulation analyst a wide variety of other tools for manipulating input data, analyzing output data, and performing other tasks in the modeling process.

2.4 Animation

Using computer graphics to dynamically display simulation entities and related activities is defined as animation. In some cases, the term simulation visualization is used for the same thing. The widespread availability of microcomputers and inexpensive, high quality graphics coupled with expectations of users to see animated systems has steadily increased the demand for a visual component in simulation.

2.4.1 Using Animation in The Proper Context

Animation is used for several reasons. Among these are:

Produces User Friendly Output – For the non-technical person, an animation will be much easier to comprehend than a statistical printout. By watching simulated entities move about on the screen, the observer can become familiar with the system's operation and with the logic embedded by the analyst.

Validation – When the simulation analyst is working with a system expert, animation provides a method of communicating information concerning the construction of the model. The expert and analyst both can visually examine the model as a means of validating its operation.

Tool for Debugging – The simulation analyst can use an animation as a debugging aid. Subtle inconsistencies may be visually detected easier than through statistics. The analyst is able to verify that the actual model matches his or her conceptual design. Animation will make errors and problems obvious, thereby eliminating the temptation to say, "It's such a rare problem that it's not worth fixing." No matter how rare or inconsequential the problem may be, if an observer can see it happen during an animation, it has to be dealt with or the face validity of the model will be destroyed.

Visual Impact/Sales Tool – One of the most popular reasons for using animation is the visual impact or sense of reality that it brings to the modeling process. It is much easier to sell a system concept when it appears to work when viewed on the screen of a computer than it is to sell a system represented by a stack of paper covered with numerical data. Most vendors of industrial equipment rely heavily on animation as a tool for use by its sales force when trying to "sell" a concept to a customer. Seeing is believing and seeing an animation is no exception.

2.4.2 Misuse of Animation

Like all tools, animation can be misused. The analyst must remember to use it to supplement a simulation study rather than drive it. The following pitfalls are common when animation is used.

Creation of Overconfidence – A model is still just a model and is only as accurate as the data and assumptions it uses. Animation tends to make model users and creators forget some of the underlying assumptions and techniques that were incorporated during the construction stage. It is easy to think that if it looks realistic on the screen then the system must be modeled properly. This is not necessarily always the case.

Unwise Time Spent on Animation – Rather than spending time building the actual model, undue time is spent trying to build an attractive animation. The animation should be used as a supplement to the simulation not as its focal point.

A Substitute for Good Statistical Work – Rather than spending time to perform a proper analysis on the results of a simulation study, the decision maker bases his or her opinions on a short period of observed animation. The animated portion of the simulation observed might not be representative of the system's true behavior. It is very important not to jump the gun and draw conclusions only on the basis of an animation.

2.4.3 Animation Attributes

Animation software may be structured in different ways depending on the application. For instance, most personal computer animators are 3-dimensional, exhibiting graphics with height, width and depth. Some animation packages offer pixel-based graphics and others, like Proof Animation from Wolverine Software, offer CAD-like 3-dimensional vector-based animation.

Another attribute of animation software is that it allows a model to run real time or in the post processor mode. The term real time is used indicate that the animation runs on the computer screen while the simulation program is executing. The disadvantages for this type of animator are a loss of speed, possible inaccurate statistical representation of the time period being modeled, and the need to run the simulation when the animation is viewed. Additionally, some animation packages allow model parameters to be changed on the fly. This can result in the corruption of output statistics.

The post-processor mode means the animation is displayed after the simulation run is complete. The simulation creates a data file that is used as an input to the animation program. A post processor offers many advantages such as the ability to fast forward, reverse, speed up or slow down a viewing. The animation can be easily transported requiring only the animator software and data file. Since the animation is run from a data file, multiple copies can be running at one time without needing additional copies of the simulation program.

2.4.4 Example Animation Software Packages

Animations are available in a wide array of simulation environments, simulators and simulation languages. Most simulation projects are accompanied with an expectation of producing an animation. Animations can be found for simulation applications ranging from flight simulators to modes coded in Second Life's Linden Script to medical simulations. To support these efforts, a variety of specialty tools exist.

On one end of the spectrum are the general graphic display objects found in programming languages such as Visual C++ or Java and the programming features to move these objects across a computer screen. On the other end of the spectrum are specialty animations developed to accompany specific models.

Many animations are created using graphics tools provided simulation software vendors. Among these are the tools found in Arena 3DPlayer and Wolverine Software's Proof Animation.

Arena 3DPlayer is a post-processor that enables a simulation analyst to create and view three dimensional animations of models created in Arena. First, a system layout is developed and when an Arena model is run, a simulation history file is created. This file feeds information into the 3DPlayer and enables the movement of dynamic, user-defined entities through the model layout. 3DPlayer provides a user-friendly interface which employs pull down menus to allow non-programmers to quickly develop realistic animations.

Proof from Wolverine Software Corporation is a low priced, high performance animation program. Originally designed to work with GPSS/H, Proof's power and flexibility appeal to analysts using other simulation software products. A family of Proof Animation products currently exists including:

- Proof 3D
- Proof Professional with unlimited memory for large, complex animations
- Personal Proof which has size and memory limitations
- Student Proof Animation intended for learning and classroom settings
- Run-time Proof intended for displaying Proof animations but lacking development
- Proof Demo Maker and Demo Viewer which allow animations to be distributed
- Proof for Extend which is a custom version of Proof for the Extend simulation product from Imagine That, Inc.

2.5 Bibliography

Arena Simulation Information, http://www.arenasimulation.com/, Retrieved June 26 (2009).

D.T. Brunner (Chair) "Easy to Use Simulation 'Packages: What Can You Really Model?" In Proc. 1988 Winter Simulation Conference (San Diego, Calif. Dec 12–14), SCS, San Diego, Calif., pp. 887–891 (1988).

D.T. Brunner and James O. Henriksen. "A General Purpose Animator," In Proc. 1989 Winter Simulation Conference, SCS, San Diego, Calif., pp. 155–163, (1989).

Imagine That and ExtendSim Information, http://www.extendsim.com/, Retrieved June 26 (2009).

R.W. McHaney, "Discrete-Event Simulators Optimize Industrial Operations," Personal Engineering & Instrumentation News, (Aug), pp. 57–62 (1989).

Minuteman Software Information, http://minutemansoftware.com, Retrieved June 23 (2009).

T.J. Schriber, Introduction to Simulation using GPSS/H, John Wiley & Sons, New York (1990).

T.J. Schriber and D.T. Brunner, "Inside discrete-event simulation software: How it works and why it matters," In Proc. 2008 Winter Simulation Conference, SCS, San Diego, Calif., pp. 182–192 (2008).

Simscript III Information, CACI Products Company, http://www.simscript.com, Retrieved June 23 (2009).

Ingolf Ståhl and Beliber AB, WebGPSS Information, http://www.webgpss.com, Retrieved June 21 (2009).

C. Standridge, "A Tutorial on TESS : The Extended Simulation System," In Proc. 1985 Winter Simulation Conference. SCS, San Diego, Calif., pp. 73–79 (1985).

Wolverine Software Corporation Information, http://www.wolverinesoftware.com, Retrieved June 23 (2009).