Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

OBJECT ORIENTED MODELING WITH SIMPLE++

Dietmar F. Geuder

AESOP GmbH
Mittlerer Pfad 9
70499 Stuttgart, GERMANY

ABSTRACT

Object orientation has become a major competitive factor in information technology. Reasons for this are the control of complexity, increased quality and possible reuse of modules. The discrete event simulation system SIMPLE++ being presented in this paper brings these advantages to the simulation user.

SIMPLE++ is a general purpose system for the object oriented, graphical, and integrated modeling, simulation, and animation of systems and business processes. Models are built with objects from a class library containing predefined building blocks. User defined classes can be created in an integrated way, allowing the modeling environment to be tailored to different application areas. The class structure, inheritance and unlimited number of hierarchical levels lead to a significant increase of productivity in building, changing, and maintaining models.

The concept and benefits of object oriented modeling will be illustrated in the following. In addition to the modeling environment, the support for validation, simulation, and evaluation of results is covered. Also, the integration of simulation models into existing IT solutions as well as application specific object libraries are explained.

1 INTRODUCTION

During the last forty years simulation systems evolved from basic programming languages to graphical and application oriented modeling and simulation environments (Schmid 1994). With the original pure simulation languages, modeling requires a comparatively high level of programming skills and is therefore mainly performed by specialists. Yet these languages provide for a substantial flexibility to model the real world to as much detail as necessary and allow a vast variety of application areas to be covered.

Modern graphical simulation tools being targeted for specific application areas usually offer a standard set of building blocks or templates. With these a model can easily be built graphically on the computer screen. The model is then adjusted to the real system by setting parameters and selecting standard operation rules. This makes simulation systems more user friendly and requires less programming expertise. Thereby models can be built by practitioners with only limited experience, making simulation more widely accepted in industry.

The modeling power of these conventional tools, however, reaches its limit with the high complexity of modern processes in industry. As technical processes in all types of application areas become increasingly complex and interconnected, methods to manage this complexity must be provided by the modeling environment. Also, users need to be able to define and integrate specific rules and control strategies existing in the real system. Adjusting parameters and selecting rules often cannot provide this flexibility.

For handling complexity and for increasing efficiency in software development the concept of object orientation proved to be a powerful paradigm. Many complex systems can only be built and maintained if they are structured according to the principles of object orientation. An additional effect of the object oriented approach is the increase in quality and the reuse of components. This is even more evident with modeling environments of simulation systems.

For this reason SIMPLE++ was designed to provide a fully object oriented modeling environment. Models are built by graphically inserting instances from a set of classes of a library. In the same way new more complex user-defined classes are built and automatically become part of the modeling environment. An unlimited number of hierarchical levels is supported for modeling and inheritance makes the modification of models very efficient.

The properties and benefits of this object oriented approach and further unique features of SIMPLE++ will be illustrated in the following sections.

2 OBJECT ORIENTED MODELING

Object orientation has become an important criterion for software development in recent years and is widely propagated by major IT companies. Instead of a 'flat' functional approach to describing and implementing systems, the functionality is encapsulated in a hierarchy of distinct classes with fixed interfaces. This approach applies in the same way for object oriented simulation systems. SIMPLE++ being fully object oriented is not only implemented with this technique in C++ but also brings the full power of this methodology to the user.

2.1 Concepts of Object Orientation

The key concept of object orientation is the separation of complex systems into distinct independent parts. These parts or objects have their own data and functionality and communicate with other objects by defined interfaces. By this encapsulation the complexity of simulation models becomes manageable.

Specific types of objects are represented by a *class* while the objects themselves as used in a system or model are *instances* of this class. This is much like the concept of templates from which as many copies as necessary can be placed in a model. The important difference, however, is that the instances (children) of a class will always be in an *inheritance* relation to their parent class, i.e. they will inherit the properties and functionality from the class. Any changes made to a class will thereby automatically be propagated throughout the entire model and be applied to all instances. The copies of templates used with conventional systems do not reflect changes made to the template (see Figure 1).

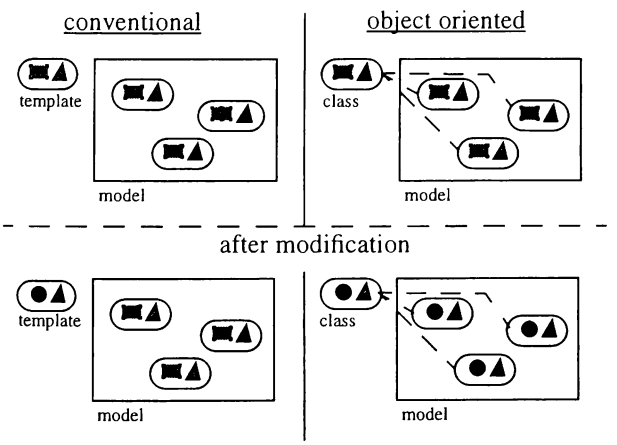

Figure 1: Changes to Classes are Automatically Propagated by Inheritance

The automatic propagation of changes is especially important for making modifications and for maintenance of simulation models. The productivity and quality is considerably increased by inheritance compared to traditional systems without this feature.

The same principle holds for *subclasses* being derived from a class. *Variants* of an object type can easily be created by building subclasses of a more general class which contains properties and functionality common to the variants. The properties which distinguish the different variants are then modified or added to the corresponding subclasses (see Figure 2).

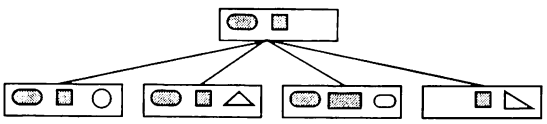

Figure 2: A Class Hierarchy of Derived Classes is Used for Variants with Common Properties

Again any modification to the common properties need only be performed once to the top level class as the changes are inherited automatically by all subclasses/ variants. With an unlimited number of levels an entire hierarchy of derived subclasses can be established in this way.

This feature proves to be a very powerful approach to create models for the "what if" scenarios simulation models are commonly used for. Variations of a model can efficiently be built and maintained. The common parts of the system are constructed and changes to it are made centrally in the top level class while the subclasses contain the parts specific to the different systems. An assembly and a storage area may, for example, be identical for variants of a manufacturing system while different transportation systems are to be evaluated. Should the storage area need to be modified at any later time, this has to be done only once, reducing the amount of modeling work and the probability of errors.

For simulation models an object focus is very appropriate as the systems to be modelled usually contain physical objects. In a manufacturing environment, for example, different types (classes) of machines may be used and a few of each type (instances) will actually be located in a production line. These objects are usually grouped into hierarchical levels representing entities or objects on their own. Thus an object oriented approach comes very natural for a modeler trying to implement a real life system. It is not by chance that the first language introducing the concept of object orientation was the simulation language Simula.

2.2 Modeling

SIMPLE++ as an object oriented system brings the object oriented approach and its benefits to the simulation user. Due to their development history most common simulation systems are either conventional (Banks 1994) or are merely object based and/or incorporate limitations in the use of hierarchy (Pegden 1992). SIMPLE++ in contrast was designed and developed to incorporate the full features of object orientation internally and for its modeling and simulation environment.

The basis for modeling any system is a library of classes containing all necessary elements and tools to build a model. The default class library of SIMPLE++ shown below contains the following categories of generic objects: processors, movable units, data storage, interfaces, controls, and user interface elements. Being a general purpose simulation tool, the classes are not domain restricted but can be used to represent any type of material or information flow.

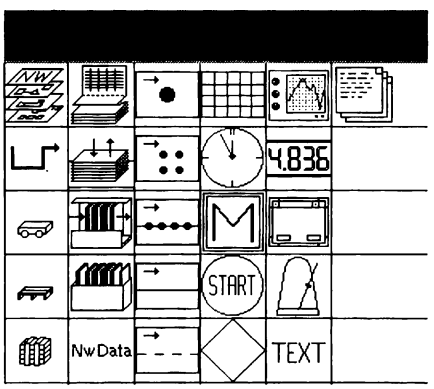

Figure 3: Class Library with Generic Objects for Modeling

Models are being built with these classes by graphically inserting instances of the required objects into an empty frame and interconnecting them as necessary. Every model being built that way is again contained in the library as a new user-defined class. It can then be used to create even more complex models on the next higher hierarchical level. In this way sets of domain specific application objects can easily be built making the modeling environment extremely tailorable to any field of application. Examples of existing application objects include manufacturing, assembly, chemical processes, transportation and distribution, and business process reengineering.

Figure 4 gives an example of a user defined object representing a processing element with an input buffer, single processor and output buffer. It may, for instance, be used as a machine for a manufacturing model or as a department for a business process model. Two serial processors for the buffers and one single processor are employed to construct the new building block. The flow of material or information is determined by the directed connections between these objects.

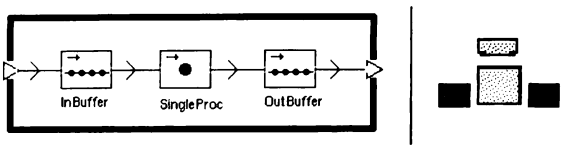

Figure 4: Example of a User-defined Object and the Icon Created for it

Using the built-in picture editor, a new icon together with the corresponding animation structure may be created for that application object. It can then be used in the same way as any other basic or user defined class in the library. A next higher level representing, for example, an entire production line may be modelled using this new class. The following figure shows an example for such a line containing basic conveyor elements and the above designed buffered processor.

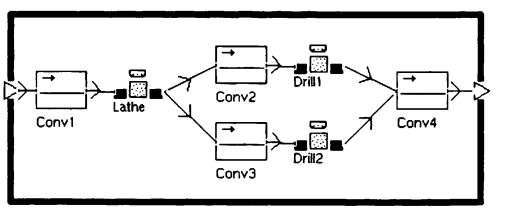

Figure 5: A Simple Production Line Containing Basic and User-defined Objects

2.3 Hierarchy

The above example already showed the application of hierarchy in a model. The production line above will again be used as part of a next higher level while it itself consists of lower level hierarchical elements. In SIMPLE++ the number of levels is unlimited and models can grow as detailed as necessary by adding further layers.

Stepwise refinement of models is supported as further hierarchical levels can be added at any time during the development of a simulation model. By this, *prototyping* can easily be done. Fairly coarse models of a real life system may be created first and more detail is added incrementally wherever necessary and appropriate. Vice versa models can be aggregated to more complex structures. With each model being represented as a class in the

library, it can also be exported and imported into another class library to combine the work of several users working in parallel on a large system.

A *top-down* and a *bottom-up* approach and any combination thereof are possible as either incremental refinement or aggregation of lower hierarchical levels is inherently supported. As the hierarchy is implemented in a fully integrated manner, the different layers can be "exploded" at any time. That is, during modeling or during a simulation run lower levels may be opened and the behavior of the model may be observed on different levels of detail.

In conjunction with the user definable dialogue boxes *different views* of the system can easily be generated. For presentations to top management, for instance, only the top level may be relevant together with the results from simulation experiments. For regular users, e.g. engineers, the behavior of specific lower level parts may be of interest while model developers will need to have full access and control. With hierarchy and dialogue boxes the information to be presented can be defined and irrelevant information and complexity be hidden.

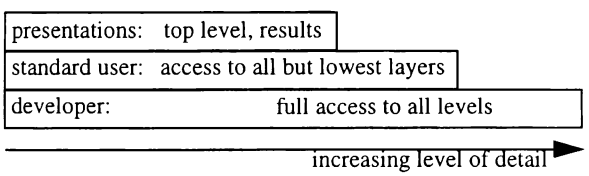

Figure 6: Different Views of the Model are Supported by Hierarchy and User-defined Dialogues

2.4 Inheritance

Inheritance as a major characteristic of object orientation is a key factor for efficient modeling. In most models a number of components are used repeatedly. Examples for these are single conveyor elements in an entire transportation system or aisles of a high bay warehouse. In an object oriented model these repeatedly used units will be instances (children) of a single class (parent) and will inherit its structure and properties.

In contrast to conventional copies of a template, all these instances can now be changed automatically by modifying the class. Without the inheritance relation changes need to be made repeatedly to all copies which is a tedious and error prone exercise. Inheritance on the other hand provides for easy and efficient modeling and maintenance and leads to a major increase in productivity (see also Figure 1).

SIMPLE++ as a fully object oriented system incorporates this inheritance for instances of a class as well as for subclasses being derived from a top level class. If necessary, the inheritance relation can be overridden by local changes to an object's properties. If necessary at a later time the inheritance relation can be switched on again, causing the corresponding properties to be changed back to the parent's values.

2.5 Flexibility

Standard domain restricted simulation environments usually offer a set of strategies and rules that can be selected for a specific behavior of the model (Lilegdon 1994). Thereby common operation rules can easily be included in a model. However, this comes at the expense of the flexibility to model the behavior or the real system as closely as it is possible with simulation programming languages.

SIMPLE++ offers the easy to use graphical modeling environment of domain restricted simulators, yet it also incorporates a complete object oriented *programming language* called SimTALK. Most of the modeling will usually be graphical, relying on the default behavior of the basic objects. However, specific rules for material or information flow or decision rules for actions to be taken can be implemented as closely as necessary.

Together with the list classes for complex data structures, e.g. queues and tables, and the available interfaces SimTALK eliminates the need to link externally written program code to the model. As full access to any model component is possible by program statements, the model may even be changed by a program. In this way a simulation model may not only be created interactively but also by program code. Using the interfaces discussed below, simulation models can thereby even be created by external programs and SIMPLE++ can be completely integrated into another application.

The flexibility available by the generic objects and the integrated programming language can be used to tailor the modeling environment exactly to the relevant field of application. The application objects being created this way can then be exported and imported to *reuse* them in future projects.

3 MODEL VERIFICATION AND VALIDATION

A very important step in every simulation project is the verification and validation of the model, i.e. the proof that the model is operating in its intended manner and that it correctly represents the real system (Gogg 1992). This task has to a great extent to be performed interactively by the user and thus contributes considerably to the overall cost of a simulation study. It must therefore be supported by the system as far as possible. Again, object orientation of the modeling environment makes this process very

efficient and controllable. Apart from the inherent transparency of the model gained by the class structure, validation in SIMPLE++ is actively supported by an *integrated modeling, animation and simulation environment* together with a method debugger and an event debugger.

3.1 Animation

A major feature of modern simulation systems is an *animation* component by which the behavior of the model can directly be observed. In SIMPLE++ this animation is integrated with the model. Animation structures are defined for the icons or pictorial representations of an object and are automatically updated during the simulation run. The animation thereby gives a first method to check the intended behavior of the model.

3.2 Integration

With conventional systems a cycle of modeling, compiling, running the model, and looking at results often is the standard procedure. This is not the case for a truly *integrated* system. Instead, the model can be run directly and modeling can even continue while the simulation is running. Also any model element or attribute is fully accessible at any time during the simulation run. In conjunction with the animation, the accurate behavior of a model can be ensured and may even be influenced directly. In Figure 7 the difference between the conventional and integrated systems is depicted.

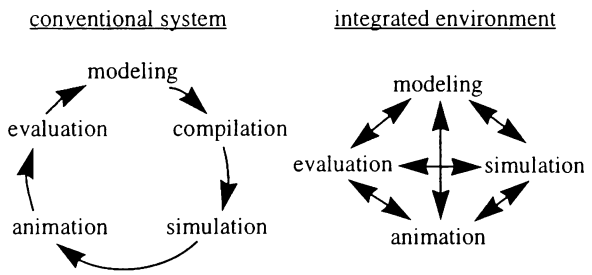

Figure 7: No Development Cycle is Necessary for an Integrated Simulation System

3.3 Incremental Testing

Taking advantage of the object orientation and hierarchy, each object can also be tested stand alone. Using a bottom up approach, the single components and hierarchy levels from which the model is built are already tested as soon as they are constructed. This incremental test and modeling makes the complexity of the model's functionality manageable. With the ever increasing complexity of today's systems the modular and hierarchical testing is a necessary prerequisite for a proper validation of a model.

3.4 Event and Method Debugger

For a proof of a model's correct behavior a closer look at single events of a discrete event simulation system may be necessary. While most systems allow a step by step execution of a model, finding an error by stepping through the events can prove to be a tedious task. Support of this process by an integrated *event debugger* is extremely helpful in this context.

The event debugger in SIMPLE++ allows one to view the list of scheduled events with all corresponding information. *Breakpoints* may be set for any of these events causing the simulation run to stop prior to its execution. A very powerful feature is the possibility to specify templates for events by which the simulation can be stopped whenever any event for which the template fits is scheduled to take place. The templates are freely configurable with possible examples being 'any disruption' or 'a disruption for a specific workcentre' or the 'movement of a specific type of object within a given time frame'.

Furthermore a *method debugger* is built into SIMPLE++ by which the strategies and rules implemented in SimTALK can be controlled step by step. Being completely integrated, full access to any component of the model is guaranteed at any time while the event or method debugger is active.

4 SIMULATION AND EVALUATION

Like most commercial simulators SIMPLE++ is a *discrete event* simulation system. This means that the behavior of individual entities representing objects or groups of objects are considered. Events are scheduled for any point in time something 'happens' with one of these entities. Thus time in a discrete event system does not proceed linearly but in irregular intervals (see Pidd 1994 for further details). In SIMPLE++ the events are scheduled, controlled and executed by a so called Eventcontroller. It allows one to reset, initialize, and start the simulation run as well as stop it at any time and proceed step by step.

One part of the evaluation of a simulation run obviously is the animation by which the overall behavior and performance of the real world system being modelled can be inspected. The built in bar graph, numerical display or the plotter may also be applied to visualize values or sequences of values which are important to the user.

Usually more important, however, are the *statistical values* gained from a number of simulation runs. With SIMPLE++, a large number of statistical data are automatically collected by default. Such data is collected

for individual stationary objects and movable units, or even for entire classes of objects. Taking advantage of object orientation, the collection of statistical values can also be controlled by the classes and gathering of statistical data can be reset, enabled or disabled for all instances by inheritance.

The statistical data being collected may be processed further as they can be accessed via attributes of the corresponding objects. Thus the utilization of processors, for instance, may be used in control rules determining the flow of material. Also, they can be used to calculate any other type of values being relevant for evaluation. Statistics can also be written to an internal table or to an external file, from which the data can, for instance, easily be imported into spreadsheets or word processors to allow creation of reports or presentations.

5 INTERFACES AND ADDITIONAL FEATURES

Simulation models in many cases are not used stand alone, but need to be integrated with existing IT solutions. This may either be a true seamless integration or it may be a connection for data exchange. Standard interfaces are therefore a viable component of a simulation system.

A set of different interfaces is available in SIMPLE++ for connection to other programs. The most common way to exchange data is by *ASCII data transfer.* Also, a file interface is available to write data to and read it from a file.

Often data needs to be imported from external data bases. For this reason an *SQL-interface* is provided which allows direct access to all major relational data bases to either retrieve data or write data into the data base. As an example, order and resource allocation data may be read from an MRP system into a simulation model while schedules generated by the simulation run are exported. This can be done automatically using the available interfaces and thus reduces the amount of work necessary to provide required data.

Apart from the fairly simple data exchange a simulation model may be integrated into another application more directly using the connectivity of modern multitasking systems. The *remote procedure calls* of UNIX and the *dynamic data exchange* (DDE) capabilities of Windows NT are supported and allow one to create a seamless environment of modular applications supplementing each other. A SIMPLE++ simulation model may, for example, provide a dynamic component for static layout programs or business reengineering programs or may serve as a monitoring component for a process control application.

6 APPLICATION AREAS

Being a general purpose simulation system SIMPLE++ is employed for a wide variety of applications. The generic objects together with the hierarchy and object orientation allow to adapt the modeling environment by graphically creating application specific objects. Due to the built-in programming language these may be tailored exactly to the required functionality. These application objects are contained as classes in the class library and may be used like the basic objects shipped with SIMPLE++.

Class libraries from different application areas are already available and a selection of them are shipped with SIMPLE++. Examples are AGV systems, personnel pool, standard strategies, warehousing, shop floor control, and process industry. Objects of the different domains can be imported into one library and may be used jointly in simulation models. In addition, these classes are open for the user and can therefore still be adapted or enhanced to fulfill project specific requirements (Geuder 1995).

Discrete event simulation is so far mostly used as an off-line planning and optimizing tool. It is mainly utilized to determine the performance of planned systems prior to their construction or to evaluate changes to existing systems. With growing computing power, models that are created for these type of off-line planning are increasingly used for on-line tasks controlling daily operations. Thanks to its openness and flexibility SIMPLE++ also supports this type of applications and is used for e.g. scheduling and monitoring tasks to a growing extent.

7 CONCLUSION

Simulation has proved to be a powerful tool for evaluating and optimizing the performance of complex real world systems. The advances in computer hardware and the modern user interfaces for graphical modeling help to increase the popularity and widespread use of simulation. The limiting factor, however, often is the complexity of today's systems and the missing flexibility of domain restricted simulators.

Object orientation is an approach which provides the necessary means to control the complexity of simulation systems. SIMPLE++ as a fully object oriented discrete event simulation system brings the benefits of object orientation to the simulation user. Being fully object oriented, it incorporates inheritance, modularity and hierarchy and thereby leads to higher productivity, reusability and improved maintenance. The necessary flexibility is achieved by an integrated programming language which allows to exactly specify decision rules or strategies.

REFERENCES

AESOP GmbH. 1995. *SIMPLE++ reference manual version 3.0.* Stuttgart, Germany.

Banks, J. 1994. Software for simulation. In *Proceedings of the 1994 Winter Simulation Conference*, ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, 26-33. Orlando, Florida.

Geuder, D. 1995. Modular application objects: closing the gap between flexibility and ease of use. In *Proceedings of the 1995 European Simulation Conference,* Vienna, Austria (accepted).

Gogg, T. J. and Mott, J. 1992. *Improve quality and productivity with simulation.* JMI Consulting Group.

Lilegdon, W. R. 1994. Manufacturing decision making with factor. In *Proceedings of the 1994 Winter Simulation Conference*, ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, 420-426. Orlando, Florida.

Pidd, M. 1994. An introduction to computer simulation. In *Proceedings of the 1994 Winter Simulation Conference*, ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, 7-14. Orlando, Florida.

Pegden, C. D. and Davis, D. A. 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 and J. R. Wilson, 390-399. Arlington, Virginia.

Schmid, B. 1994. Simulationssysteme der 5. Generation. In *Fortschritte in der Simulationstechnik vol. 6*, Berlin.

AUTHOR BIOGRAPHY

DIETMAR F. GEUDER is currently a product manager in the sales and marketing division of AESOP GmbH. After a year of graduate studies of artificial intelligence at the University of Colorado at Boulder he earned his diploma in computer science at the University of Erlangen with a minor in industrial engineering. Since joining AESOP he has worked in development, consulting, and training and did project work in the area of transportation systems and shop floor control.