- Accepted abstract: European Simulation Conference, June 1991 Copenhagen Denmark

SIC – A SYSTEM FOR STOCHASTIC SIMULATION IN C++

Carmelita Görg, Bernd T. Kluth[1], Udo Salewski
Institute for Teleprocessing, Aachen University of Technology
Kopernikusstr. 16, D-5100 Aachen, Germany

Introduction

Object oriented programming provides an ideal method for describing the structure of a model as a simulation program. Previous examples of such systems, e.g. SIMULA or SMALLTALK, were either not sufficiently widespread or not efficient enough. Simulations of high speed communication systems, however, tend to require high event-counts, thus making efficiency a crucial requirement. C++, on the other hand, combines efficient execution with the object oriented programming paradigm. The language, therefore, is well suited to be used as the basis of a simulation system. Furthermore, establishing an appropriate modelling concept on top of C++ can offer additional convenience to the user. The system described in this paper is called SIC[2]. Based on a portable simulation kernel it offers additional support by a modelling toolkit and a flexible facility for graphical model specification and representation of results. The simulation itself is supported by options for animation, debugging, and functional parallelization. The benefits of this modelling concept established on top of C++ are demonstrated based upon examples of queueing models and communication systems.

Simulation system SIC

The simulation system SIC [1] is written entirely in C++. A portable kernel enabling process oriented simulation is at its core. On top of that, SIC contains a set of building blocks for specifying queueing network models in a well structured way. For portability reasons, process oriented simulation is implemented in SIC without using the task library supplied with the AT&T C++ version. SIC is implemented on various UNIX machines, the graphics editor with the program generation system as well as the animation system run on a workstation under the *X window system*. The system is also being used for a student simulation lab on personal computers with MS-DOS[3].

SIC process and program structure

Processes are defined in SIC as C++ classes. In this way, only the *structure* of a process is defined, thus allowing several process incarnations to be created from one template. As it is possible to parameterize a process in SIC, each of these process incarnations may be set up individually, e.g. concerning its stochastic characteristics and the use of model-resources. In this manner, sets of similar, though slightly differing processes may be maintained efficiently in SIC.

The execution of the model's processes is controlled by a scheduler class which maintains process activations. A process definition in SIC consists of three distinct and clearly distinguishable parts, which naturally match the modeler's view: *persistent data:* as members in the class definition, *initialization:* via the parameterized constructor, *algorithmic behavior:* in a specific member function.

Resulting from this process structure, the first part of a typical SIC program consists of a collection of *process definitions.* It is not till the subsequent main program that process incarnations are *created* and parameterized. In the context of queueing models, both the model's topology and selection of parameters focus here, allowing for easy maintenance by the user.

A modelling toolkit

The structure defined offers an elegant way to establish a library with components of simulation models. Their use reduces the effort of writing a simulation program considerably, while simultaneously advancing its clearness.

Based on the SIC program structure, a toolkit of modelling components for queueing network models was established. It contains definitions of several predefined process types. Besides that, it contains components for modelling the interaction of processes in a simulation program such as queues and semaphores. Employing these blocks in his own simulation program enables a user to quickly develop well-structured models. A generalization of queues in the class concept additionally allows for model components and thus program parts to be reused in different models.

As another example, a library with modules for the simulation of computer networks using a token protocol is available. With this library a simulation program can easily be written for the *token ring* and for the *FDDI* access protocols.

Graphical program generator

Based on the SIC modelling toolkit described, the creation of simulation models may further be facilitated by program generators. A user-configurable system for the graphical generation of queueing network simulation programs was developed for SIC. Based upon a structured graphics editor written in C++ and running under the *X window system*, it serves to generate a SIC program from a schematic drawing (see Fig. 1) that the user enters by selecting predefined model components from a menu. New model components can be added to the menu. After parameters have been added, a simulation program may be generated, translated and run without leaving the editor.

A module for animated simulation

To give the user better control and understanding of his simulation, an animation module based on *Motif* and the *X window system* was developed, that allows the control of a running simulation while showing its actual state on a graphic display.

The user can interact with the running program, request intermediate results from the statistical evaluation, ask for some system specific lists, e.g. a list of queues, or just look at the dynamic behavior of the simulated system.

The graphic display shows the individual components of the

[1] the author is now with Mannesmann Mobilfunk, Düsseldorf

[2] SIC: *Simulation in C++* or *Right this way, that's it* (Latin).

[3] The system can be obtained for UNIX or MS-DOS at the address given above (e-mail: cg@dfv.rwth-aachen.de).

model and their actual states. State descriptions depend on the type of a component. A queue, for instance, is characterized by its name, type, maximum capacity, actual count and average count. The count as a dynamic value is represented as a level meter. Fig. 2 shows a snapshot of the animation window. A possible result of such a simulation run is shown in Fig. 3 in form of a distribution function obtained by the new *LRE-algorithm* [2] for the evaluation of correlated random sequences.

Parallelization

Besides the case and security of modelling, the *efficiency* of simulations is still an issue. Parallel processing offers a way to speed up simulations. The SIC system was extended to enable simulations to be executed as parallel programs. Parallelized according to the *functional parallelization approach*, the same simulation program source can either be translated to a sequential simulation program or a parallel one. Using the object oriented features of C++, this is accomplished mainly by changing the implementation of some internal classes inside SIC and by use of a preprocessor, which serves to create the additional processes needed.

[1] B. Kluth. *Multiprocessor architectures with functional parallelization for stochastic simulation.* Ph. D. Thesis, Aachen University of Technology, 1990.

[2] F. Schreiber. *Effective control of simulation runs by a new evaluation algorithm for correlated random sequences.* AEÜ, Vol. 42, pp. 347–354, 1988.

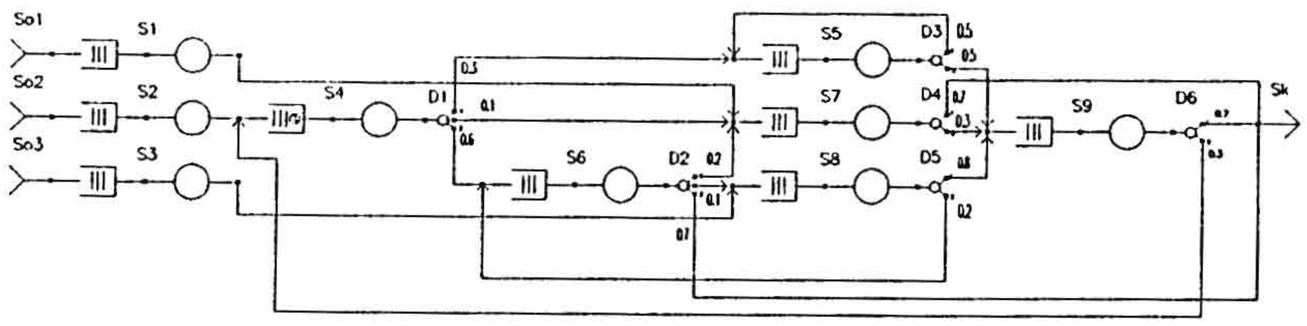

Figure 1 : A queueing network model

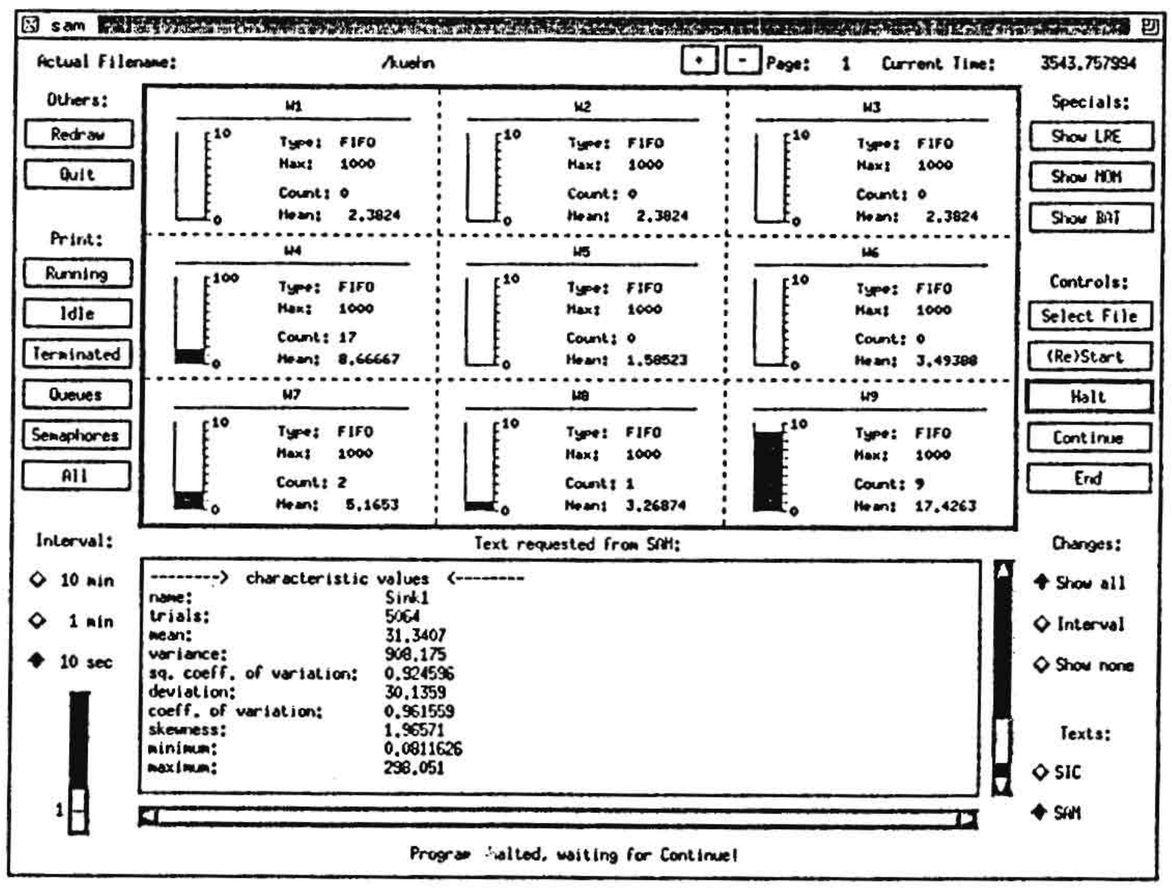

Figure 2 : Snapshot of an animation window

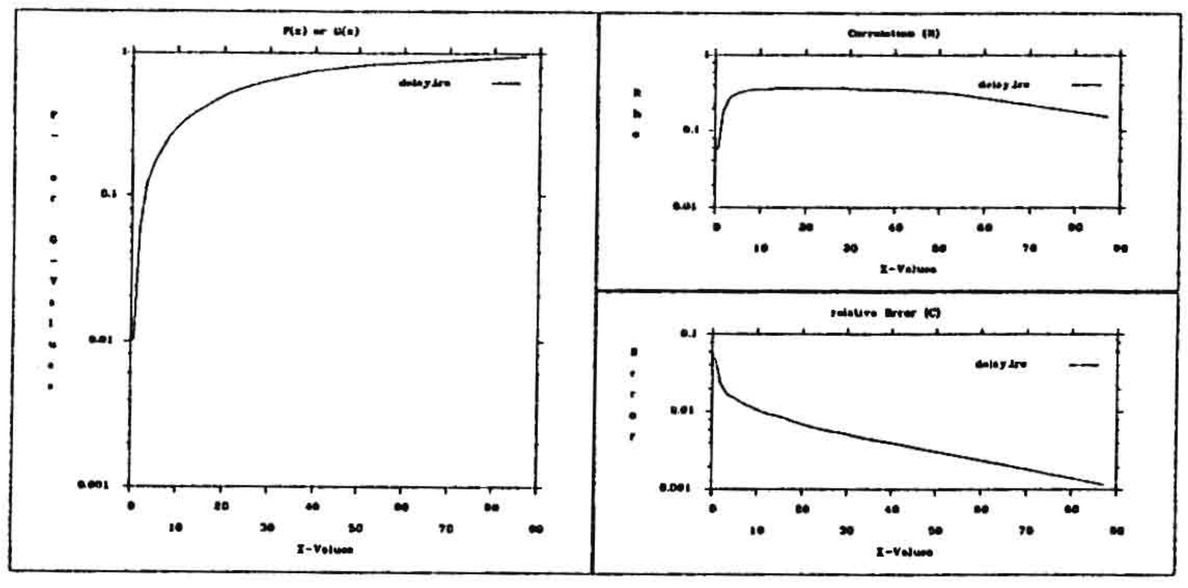

Figure 3 : Distribution function of the total delay time obtained by the LRE-algorithm.