The Simulation System ***Lancelot***

Prof. Dr. W. Kowalk, Universität Oldenburg
Ammerländer Heerstraße 121, 26121 Oldenburg, Germany

Abstract

We present a new simulation system, ***Lancelot****. This system has been designed to sustain graphical, animated simulation in a process oriented manner. It can be used for discrete, continuos or hybrid simulation, which allows to model time discrete queuing systems as well as time continuos physical models. Besides these examples many other applications are supported; for example,* ***Lancelot*** *assists engineers also in development, implementation and analysis of embedded control systems.*

1 Introduction

Detailed modeling of realistic aspects of technical or other dynamic systems can be supported efficiently by advanced simulation techniques. To assist computational engineers it is required to supply them with efficient modeling paradigms, besides modeling tools like graphical representation of the models and statistical analysis techniques. Also if cooperation of engineers working in special application areas is to be supported, different modeling techniques should be provided in the same modeling environment.

Lancelot has been developed to fulfill these requirements. It provides the programmer with several modeling paradigms, is executed under a graphical interface in X, uses C-syntax as programming language, and sustains efficient management of complex model libraries.

2 The programming logic of C-Sim

The programming language ***Lancelot*** is based on a C-based simulation language, called C-Sim (this name has been used by several other C- or C++-based simulation systems, which will not be considered here).

The logic of C-Sim can be described to be

1. process-oriented (similar to Simula 67),
2. event-oriented (similar to SimScript),
3. asynchronous or synchronous message handling

Since these concepts are very well known, we will elaborate on them further only with respect to their extensions in C-Sim.

2.1 Processes in C-Sim

The notion of a process is central to the programming logic of C-Sim, since a process specifies the dynamic behavior of either a system or a constituent of a system. Since programmers can model the properties of a process completely independently of other processes, a process is a well structured object, which can be easily redefined, altered or reused in other systems. Processes enclose a unique name space, so that no conflicts occur with other processes or complex objects; of course, properties of other processes or classes can be inherited by processes.

During execution, processes can be created, halted (passivated) or activated and internal values of processes in C-Sim can be inspected during execution. Exchange of messages between processes can also be animated, using ***Lancelot***'s interface.

Since processes are well encapsulated modules, they have to communicate with each other via special concepts. The first one uses a global name space, which is common to all processes. The second one uses special communication methods, being explained in sections 2.3 and 2.4.

2.2 Phases of a process

A process is structured into a local declaration part and a set of phases. While the declarations describe the local environment of the process, the dynamic behavior is determined by the phases.

Each phase consists of a set of instructions in C code and one or several final statements, stating the next phase and the time. An initial part of a phase may state some conditions, under which the phase is to be executed. Thus in general a phase can be executed only in a particular circumstance, which can depend on

- system time,
- arrival of a message,
- logical condition,
- any combination of these conditions.

The statements of a phase usually change the local state of a process; they can however also change a global variable, create and start a new process, or send a message to another process. Thus, C-Sim is as flexible as a general programming language, and can moreover easily model dynamic behavior of a system.

2.3 Communication in C-Sim

To communicate, processes exchange messages, which can be called jobs, messages or signals. A process sends a message to another process by naming the message and the corresponding process.

The receiving process stores each message in a local „`SendBuffer`", the size of which can be selected according to the communication mechanism.

- If the size is set to infinity, then all sending processes can send arbitrarily many messages, which are served according to the performance of the receiving process.
- If the size of the „`SendBuffer`" of the receiving process is set to a finite number bigger than 0, then sending processes can send only a finite number of messages. If the buffer is filled, the sender has to wait until the buffer is emptied by the receiver.
- Finally, if the size of the „`SendBuffer`" of the receiving process is set to 0, then sending processes have to synchronize with receiving processes.

This method can be used to model both, asynchronous, synchronous, and even generalized synchronous communication by limited buffer space; if the buffer size is n, we call this n-synchronous communication.

2.4 Communication Statements

Exchange of information is specified by statements, which can send and receive messages.

A „`Send`" statement determines the message and the receiving process, where a message is just a structure in C of a special type. A „`Send`" statement can have a time restriction, which is only used for synchronous communication. If time expires, the sending process proceeds and revokes its message. Thus n-synchronous communication is extended to „n-synchronous, time limited".

A „`Wait`" statement waits for the arrival of a message. Again, this statement can have several different parameters. A message variable accepts the message that is sent. A time parameter limits the time this statements waits for the next message. If this time expires, the process proceeds with an empty message variable.

This concept is highly efficient and helps to simplify many simulation models. Particularly, protocol simulation can be considerably simplified by this concept, since for example the timer concept to restrict the waiting time for acknowledgments can easily be implemented.

2.5 Multiple communication

We have seen that communication can easily be controlled by some simple concepts. In C-Sim the number of processes taking part in communication are not limited. Thus any process can receive arbitrarily many messages from any other process. Also, C-Sim provides a general buffer concept, where several processes can read from a common buffer. Thus, any process can write to or read from this buffer, which generalizes the corresponding concepts again. C-Sim allows very general communication patterns, which are not restricted by any artificial constraints.

2.6 General Conditions

Another concept of C-Sim to schedule a process is much more general, however, also less efficient. The `WaitFor`-statement activates a process, if a (general logical) condition becomes true.

All processes waiting for such a condition to become true have to be executed before any other event occurs. If no process becomes ready to execute, i.e. all such conditions are wrong, the next regularly scheduled event can be performed. However, now the state of the system might have been changed, so that again all processes waiting for such an event have to test the condition.

This shows, that the conditions of the `WaitFor`-statements are always to be executed, which makes this method less efficient. However, for some special applications it might be useful to sustain this concept, so it is provided in C-Sim.

2.7 Scheduling in C-Sim

C-Sim schedules a process with the „`Next`" statement, which determines the time when as well as the phase which is to be executed next. The simpler „`Repeat`" statement executes the current phase again.

In some applications, more complex scheduling is required. For example in protocol simulation the programmer must be able to decide arbitrarily whether the processes running in different layers are executed on the same or on different processors. This is usually not sustained by common simulation systems, and is usually cumbersome to implement. In C-Sim a simple, but efficient mechanism can be used to model this type of scheduling.

To achieve this, the „`Next`" statement is extended by a process name. This process has access to a list of detached processes and their execution times. It delays the corresponding process according to an adequate strategy, like FIFO or processor sharing, and by the „`Deschedule`" statement the corresponding process proceeds in the next phase. This very simple mechanism can be used to model many types of schedulers, where the scheduled objects are no longer passive jobs or messages, but active processes.

This opens the possibility to model in a transaction like style, where active objects seize the service of passive servers. Here, of course, both objects are active, which permits much more flexibility.

3 Graphical interface of *Lancelot*

The process-oriented simulation system ***Lancelot*** is based on the C-based simulation language C-Sim, which is syntactically similar to C. To simplify implementation and application of C-Sim programs, a graphical interface was developed, ***Lancelot***, which accomplished some further requirements. Some of them are listed below.

1. Processes are to be structured, supplying local environments for each process.
2. Global environments are used to model relationships between processes.
3. A hierarchical concept is introduced to structure several processes into one complex object.
4. Effective model management from the programmers interface sustains reusability of models.
5. Model checking is sustained by automatically animated state transitions.
6. Simulation results are presented under the same graphical interface as the simulation programs.
7. Additional features for presentation of continuos models are provided.

3.1 Architecture of *Lancelot*

Lancelot provides a flexible interface, using X standard libraries on a Linux operating system. ***Lancelot*** uses several Linux processes; one controls the interface, another one the type manager, and a third one the compiler, which maps the structured processes of ***Lancelot*** to the C-Sim pre-compiler.

Communication takes place via TCP connections, which are reliable and efficient enough. This concept also allows to run the processes on different computers, which aids to distribute the systems into an open environment.

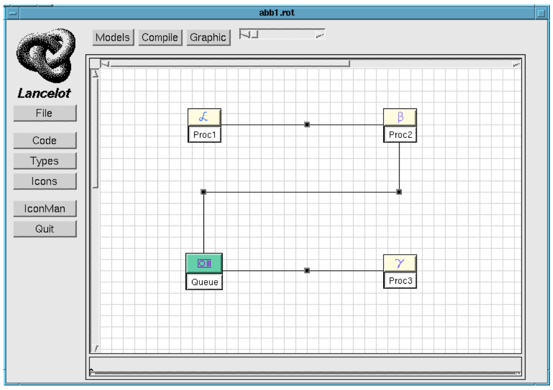
The compiler provides a complete check of all names used in the processes and global environments. This allows to model in a hierarchical manner, which is one of the most advanced features of ***Lancelot*** and will be explained in detail below.

Structuring of models is very easy in ***Lancelot***, since all simulation processes enclose their sequence of events, which are called „phases“ in C-Sim, to denote its temporal character. Also, the local name space of a process is directly protected from other processes. Thus the compiler can prevent different processes to interfere with each other harmfully.

3.2 *Lancelot*'s interface

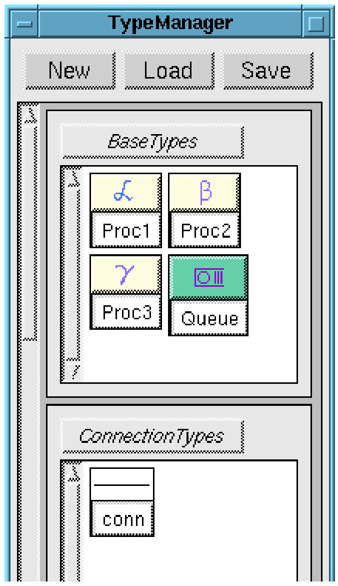
If a simulation program is to be implemented or executed, ***Lancelot*** opens a window, which presents all features, required for modeling in ***Lancelot*** (see Picture 3-1). Then an existing simulation model can be loaded and executed by a pull down menu. Picture 3-1 exhibits some features of ***Lancelot***'s interface.

Several pull-down menus have been integrated in the interface to simplify programming. Also, some pop-up menus are displayed when the programmer clicks a mouse bottom in the model area. Thus, complex functionality is integrated using a very well structured pictogram techniques.



Picture 3-1: The interface of ***Lancelot***

If a new process is to be implemented, only some basic clicks and pulls are to be done. However, ***Lancelot*** sustains a model data base, so that a collection of models can be managed in an additional window, which is accessible from beginning of execution of ***Lancelot*** automatically. Picture 3-2 shows this window, the **TypeManager**. This diagram displays, that the Type Manager encloses several processes, as well as some connection objects.

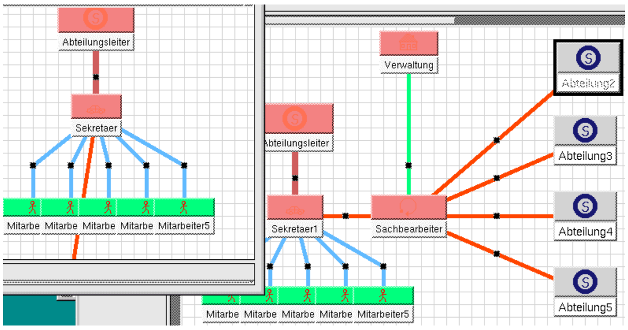


Picture 3-2: Management of simulation objects

3.3 Management of objects in *Lancelot*

Besides objects and connection types the TypeManager can include complex objects, which represent the most advanced feature of ***Lancelot*** for structuring complex models. Any collection of processes can become a complex object; complex objects can even enclose other complex objects, so that a hierarchy of processes can be created and managed in a very efficient way (see Picture 3-3). This concept shows the real power of ***Lancelot***, since each model can become an instance of an earlier defined model as often as required; such instances can easily be modified according to their need. Thus complex objects can become a very smart concept for complex models to be developed, implemented, displayed and executed.

Lancelot sustains graphical presentation of models by providing basic objects, which are used to model processes, connection objects for specifying relationships between objects, complex objects for structuring of complex models, icon objects for particular representation of objects, and type objects for the definition of common properties of processes, jobs, queues, or local environments.



Picture 3-3: Model hierarchy in a complex object

3.4 Creating simulation models with Lancelot

Now, we demonstrate how simulation programs can be constructed by ***Lancelot***. Since it is very simple to do this, we will give only some short comments on the steps required for this.

A first step is always to start the program ***Lancelot***. The empty root formula (see Picture 3-1, with no icon inside the formula) can now be filled with basic objects by simply pressing the left mouse button and selecting „basic object". A window opens, in which a process according to C-Sim syntax has to be implemented. Closing this window results in a process, ready to be compiled and executed. Picture 3-1 shows such icons, named `Proc1`, `Proc2`, and `Proc3`. Also queue objects can be created in a similar easy way.

If two such processes have been established in the root formula, they can be connected via a connection object. Again, clicking the left mouse button and selecting „connection" yields a new state, where now a sequence of points can be selected via a left mouse button click, starting with the icon of the first process, followed by some intermediate points, and finishing with the aim object, which can be a process or a queue, or even a complex object. Thus a figure similar to Picture 3-1 can be created quickly and safely, where almost no syntactical overhead is required.

Alternatively it is possible to load a predefined model, which is done by the drop-down menu „file". Of course, all models already preserved by the programmer can be accessed by this menu.

Now the simulation model can be compiled, using the drop-down menu „compile". The compiler reports all errors, where the error can be selected from the compiler's error report list, so that all typing errors or similar faults can easily be eliminated. When no syntactical errors are found anymore, simulation can be started via the drop down menu „models". The next section shows, how ***Lancelot*** presents its output to the programmer, and how this can be influenced by the programmer.

4 Output analysis with *Lancelot*

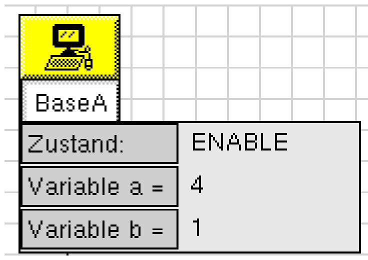
One of the main features of ***Lancelot*** is its output analysis. In many simulation systems, this has to be sustained by manually inserting function calls, which are only difficult to remove, if the simulation program is to run efficiently. Control of this has been automated by ***Lancelot*** by several methods.

If a program runs under control of ***Lancelot***, the current execution can be observed with several methods. The first one is the coloring of state changes, highlighting the process in execution, and animation of message transfer.

The second visualization method is to output sequences of data, called time series, to observe average or variation of quantities like sojourn times, number of jobs in queues, etc. The third method to visualize output results is to sample data for distributions, plotting them in diagrams, where by some parameters the plotting rate as well as the type of the displayed distribution can be controlled interactively. Since these methods are central features to sustain efficient and structured modeling, they will be explained in more details in the next sections.

4.1 Display of states and transitions

During simulation execution, ***Lancelot*** assists the programmer to observe the performed program by some special features.



Picture 4-1: Display of process variables

If a process is executed, it can step through several phases. It is useful to know, which phase a process is just performing. This can be displayed by a special color the process' icon is changed to. Thus the programmer can proceed the sequence of phases the process executes.

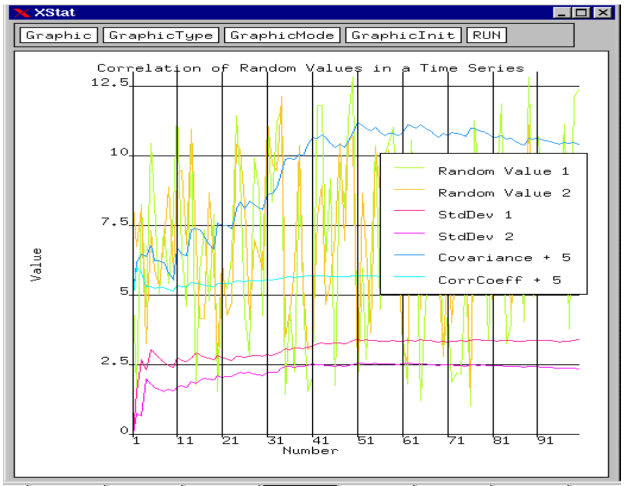
Since processes are executed quasi-parallel, exactly one process is active at any time. ***Lancelot*** highlights this process by a special color, selected in the icon formula. Thus, the order in which the processes of a model are activated can be observed by the programmer as well.

When a process sends a job to another process, using the `Send`-statement, this activity is animated by a moving icon, following the connection between these two processes. Thus the programmer can track explicitly all paths a job is sent through the model. This helps to detect errors like mistaken process names or incorrect activation rates.

Further features can be implemented to control the state of a process. If some quantities or the state of a process are to be exhibited, a display function called „buffer“ can be used. The defined values are displayed only, if the process' icon is clicked by the right mouse button, see Picture 4-1.

4.2 Output of time sequences

If some state variables are changing permanently, it is more useful to plot a graph over time of them than to display their actual value. ***Lancelot*** provides a time series feature that is comfortable to use and to adopt to different applications.



Picture 4-2: Plotting time sequences in ***Lancelot***

Picture 4-2 shows a possible output of a time series. The programmer can pursue these values, observing immediately utilization and other performance data of his simulation program. Besides line diagrams, also histograms, sequences of dots, linked dots, numbered histograms, pie diagrams etc. can be depicted.

4.3 Statistical analyzes of simulation results

Besides on-the-fly analyzes of performance information, analysts are interested in final results of a simulation run. For this several statistical function are provided by ***Lancelot***, which include

- moments, variance, squared coefficient of variation, and confidence intervals of a distribution,
- station concept,
- distributions of quantities.

Additional functionality for data analysis can be implemented by the programmer or included into the system in further releases of ***Lancelot***.

5 Conclusions

Lancelot has been developed to model real time systems of a wide range of applications, from time discrete queuing models to time continuos systems like movement of planets and embedded systems, where both, the real world model as well as the control system can be implemented within the same interface, simplifying modeling and evaluating of real world systems. ***Lancelot***'s simple graphical interface helps to model reliably and implement quickly complex systems, demonstrate their behavior and analyze them efficiently.

6 Acknowledgments

I am grateful acknowledge to Torsten Sterzenbach and Gerd Block, who have implemented the most of the interface (Sterzenbach) and the C-Sim compiler (Block).

7 Literature

D. Grunwald	„A user's guide to AWESIME; An Object Oriented Parallel Programming and Simulation System." Technical Report CU-CS-552-91. University of Colorado at Boulder (1991).
G. Agha, C. Hewitt	„Concurrent Programming using Actor." Object Oriented Concurrent computer system series, pp. 37-53 (1987).
G. Block, T. Sterzenbach	„Das Simulationssystem ***Lancelot***" Internal report of the University of Oldenburg (1996).
M. Little, D. McCue	„Construction and Use of a Simulation Package in C++." Technical Report 437, University of Newcastle upon Tyne, (1993).
	C++Sim
O.Rose	SIMPlusPlus Part I-Sim: „An Object Oriented Language for Process-Oriented Discrete-Event Simulation." Institute of Telematics Universtität of Karlsruhe (1992).
P. Musse, G. Siegel	The PROSIT Sequential Simulator: A Test-Bed for Object Oriented Discrete Event Simulation." ESS'95 European Simulation Symposium 1995, pp. 297-301 (1995).
W. P. Kowalk	„Das Simulationssystem C-Sim" Internal report of the University of Oldenburg (1992).
W. P. Kowalk	„Simulation in C-Sim. Principles of modeling, simulation techniques, and simulation tools" Internal report of the University of Oldenburg (1993).
W. P. Kowalk	„High-Precision simulation technique" Internal report of the University of Oldenburg (1996).
W. P. Kowalk	„Distributed Development of Embedded Systems." Concurrent Engineering Europe 97 pp. 162- (1997).
Y. Merkuryev	„Object Oriented Model Base for Discrete Event Simulation." ESS'95 European Simulation Symposium 1995, pp. 310-314 (1995).