Chapter

64 *The Performance Library*

This chapter presents an overview of a method for performance simulation projects using the Performance Library. The Performance Library is available as an optional product.

The method is in no way complete or described in all details. The intention is not to describe the method itself in depth, but rather to present an application area for the SDL suite. For more details about performance simulations in general, please refer to the literature in that field.

A Performance Simulation Project

To develop a performance simulation and use it to obtain estimates about a particular system involves a number of steps. The SDL suite may be helpful in many, but not all, of these steps.

The major activities in a performance simulation project are:

1. Collect information about the behavior of the system to be simulated and define the purpose of the simulation, that is, what estimates should be the result of the project?

2. Create a performance model for the system. This is often expressed as a queuing network. As the behavior of a system is usually too complex to be simulated in all details, simplifications have to be made.

   This modeling phase is the most critical phase in the complete project. A good performance model leads to relevant results, while a bad model leads to nonsense. The key question is what simplifications and abstractions you may make and still obtain relevant results from simulations of the model.

3. Describe the performance model in detail in SDL and simulate it, using the library *Simulation*, until the SDL model behaves satisfactorily.

4. Execute the generated simulation a number of times to collect statistical data from the model.

   You will normally need 10,000 to 50,000 samples (of for example a queue length or a waiting time) to obtain any significance in the data. This means that program executions will be fairly long. There are two ways to execute the performance simulation:

   – Use the library *Simulation* and start by performing the monitor commands “Set-Trace 0" and “Go”, or

   – Use the library *PerformanceSimulation*. It is specially designed to execute performance simulations and does not include the monitor system. It will execute the performance simulation in the order **10 to 15** times faster than executing it using the library *Simulation*.

5. During the execution of the performance simulation, the best way to handle data measured in the system is to write the data on file. You may then analyze the data files using packages or programs for statistical analysis to obtain, for example, mean value, variance and confidence intervals for these estimates. (The SDL suite does not provide means for statistical analysis.)

6. To validate these estimates it is common practice to compare the simulation results with results from mathematical methods applied on the system (simplified versions of the system). Queuing theory and the theory for queuing networks are the most relevant mathematical methods for such activities.

Some of these steps will discussed in more detail in the following subsections. For other aspects please see literature about performance simulations.

The Performance Model

Let us first state that a description of the functional behavior of a system and a performance model of the same system are (usually) not the same, even if both models can be expressed in SDL. The two models describe two different aspects of the same system. This is why there is a modeling phase in the development of the performance simulation.

Queuing Models

Most performance models can be viewed as queuing models or queuing networks, where a queuing network is an interconnected system of queuing models. A typical model for a queue would look like:

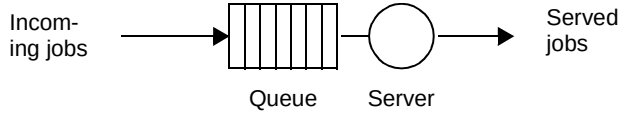

Figure 554: A queue model

The model contains jobs that need some service from the server. The jobs may have to wait in a queue for the service. The basic properties of this model are:

- The service time for a job. This can be modeled as a property of the job or of the server, depending on the system to be simulated.
- The queue discipline, that is, the order in which jobs are inserted into the queue and removed from the queue to get service. Typical disciplines are, to mention a few:
  - FCFS (First Come - First Served)
  - Priority order according to the priority of the jobs
  - Priority order with pre-emption.
- The inter-arrival time for jobs, or more generally: when new jobs are entered into the queue. The creation of new jobs is usually modeled in job-generators described as separate objects.

The queue model is a general model that may be used as an abstraction in many situations, for example to model programs (jobs) that are to be executed by the CPU in a computer (server). The queue is then the scheduled list of jobs that are in a “ready to execute” state. The queuing discipline is usually complex, involving for example priorities, pre-emption and cyclic execution of jobs.

Another quite different example would be a port, where ships (jobs) are coming for loading or unloading (the service). To perform loading or unloading the ship needs a crane (the server).

The systems that we want to simulate are usually not simple enough to be modeled by just one queue. However, models using interconnected queues-servers, connected in such a way that jobs leaving one server are inserted into the queue of another, have the power to describe many interesting real systems. An example of a simple queuing network model is given in Figure 555.

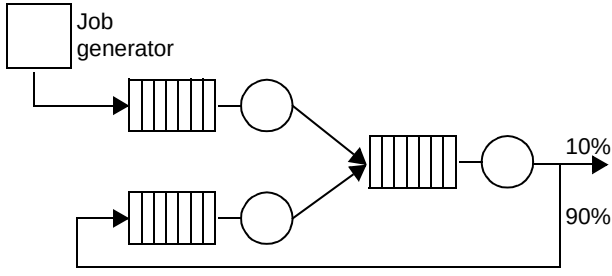

Figure 555. Simple queuing network

Each node in the network consists of one or several servers. The network also contains places where jobs are entered into the system and places where jobs are leaving the system.

Measurements

In a queue model or a queuing network model it is of interest to estimate for example:

- For jobs: total time in system, total waiting time
- For servers: load
- For queues: average and maximum queue length.

Such estimates can be obtained in two ways, using mathematical theories like queuing theory, or by measurements in simulations. With the mathematical theories, rather complicated models can be analyzed, usually more complicated than a nonspecialist thinks is possible. You should investigate this possibility before taking the decision to implement a simulation program.

Implementation of the Model

Mapping of Queue Models to SDL

A queuing network model may easily be described in SDL. Appropriate mapping rules are, for example:

- A server is implemented as a process and the queue as a variable in the server process.
- Jobs are described as data objects, that is, as passive entities that can be inserted into queues.
- Job generators are implemented as processes.
- Signals, with jobs as parameters, are used to send the jobs from job generators to servers and from servers to other servers.

Abstract Data Types for Queues and Random Numbers

In the implementation of the model there will be extensive use of queues and jobs and of queue manipulations, for example inserting a job into a queue. The chapter 63, *The ADT Library* provides an abstract data type specially designed for this purpose. See "Abstract Data Types for List Processing" on page 3165.

The ADT library also contains an abstract data type that can be used to generate random numbers. This data type can, for example, be used to draw job lengths and inter-arrival times according to a given distribution. See "Abstract Data Type for Random Numbers" on page 3156.

Random numbers are, in most situations in a performance simulation, used to model time intervals (for example service time required for a job) or to model a number of something (the number of jobs to be generated by a job generator at a certain time). The abstract data type for random numbers therefore contains the possibility to generate random numbers according to distributions returning non-negative values, for example:

- Negative exponential distribution
- Erlang distribution
- Hyperexponential distribution
- Uniform distribution
- Poisson distribution.

Implementation of Job Generators and Servers

To give a better understanding of job generators and servers, two process graphs are presented in Figure 556 and Figure 557, showing simple but typical processes.

A job generator sets a timer, and when the timer time expires, it generates a new job, sends it into the queuing system and sets the timer again. The time value in the set statements is usually Now + some random time interval. Note that the data types `Queue` and `ObjectInstance` are defined in the abstract data types for queue handling that are included in the ADT library (see "Abstract Data Types for List Processing" on page 3165 in chapter 63, *The ADT Library*).

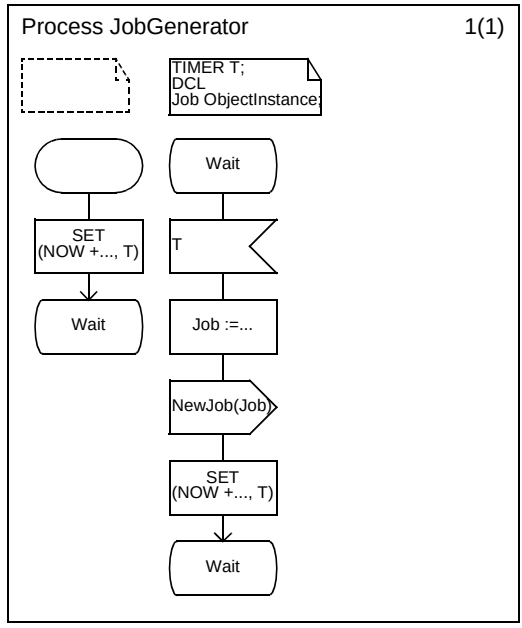

Figure 556: A job generator

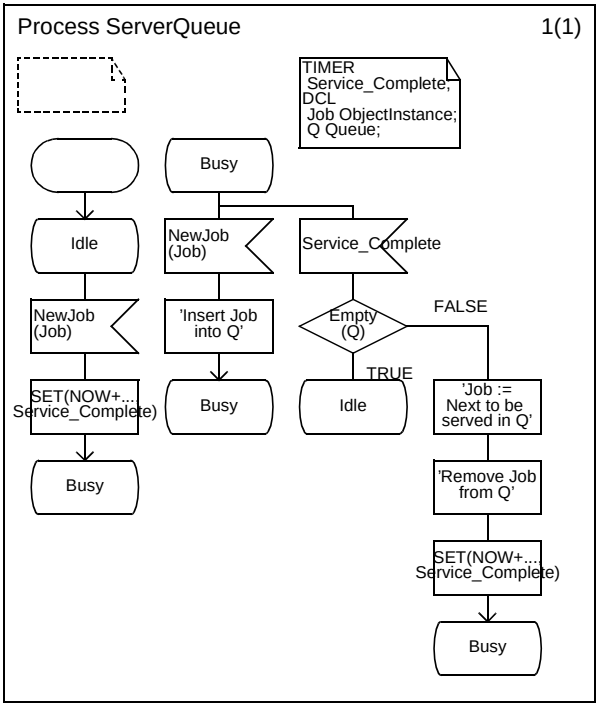

Figure 557: A server with a queue

The most important observation concerning the server process, is that the act of giving service is modeled by letting the server process wait in the state Busy. It is thus only the time needed for the service that is modeled. Otherwise the process is rather straight forward. The variable Job is used to refer to the job that is currently given service and the variable Q is used to store other jobs waiting for service. The queuing discipline will be implemented in the details in the tasks “Next to be served in Q”, “Insert Job in Q” and “Remove Job from Q”.

I/O and Performance Simulations

A performance simulation uses I/O mainly for two things:

1. Writing measurements on file
2. Prompting for parameters to the simulation.

A specially designed abstract data type for the purpose of simplifying I/O is included in the ADT library (see "Abstract Data Type for File Manipulations and I/O" on page 3143 in chapter 63, *The ADT Library*). With this data type it is possible to open files and to perform read and write operations in SDL.

It is, of course, interesting to parameterize a performance simulation on, for example, seed values for random generators, mean values for inter-arrival times and service times.

These values should be read at simulation start-up time. In SDL the concept of external synonyms can be used for such a purpose. As synonyms can only be defined in processes, not in instances, it is difficult to handle cases when the process instances should have different values for a certain simulation parameter. In such cases the abstract data type for I/O operations can be useful.

The second category of I/O mentioned above is printing of measurement data. There are basically two different types of measurements that have to be handled in a performance simulation.

1. Data concerning jobs such as waiting times.
   It is usually best to store this type of data in the job itself, until the job leaves the system, when the appropriate values are printed on a file.
2. Queue lengths and other related data.
   Such data is easiest to handle by introducing measurement processes that with regular time intervals print the queue lengths on file.

The best way to obtain the queue lengths is, in most situations, to let the measurement process view (or import) the appropriate queue variables. Otherwise a signal interface must be implemented only for measurement purposes.

The reason for printing all measurement data on file is that it is difficult to compute the relevant statistical entities at simulation time. If only the

mean values are of interest, these are simple to compute, but if variances and confidence intervals are to be computed it is better to let a professional statistical tool perform the job.

Note that data series produced from a simulation contains dependent data. If, for example, a job has been subject to a long waiting time, then the probability is high that the next job will also have a long waiting time.

Exit from a Simulation

The appropriate way to terminate the execution of a performance simulation is to call the function SDL_Halt(). It is best performed by introducing the call in a #CODE directive in a task symbol (see *"Including C Code in Task – Directive #CODE" on page 2656 in chapter 57, The Cadvanced/Cbasic SDL to C Compiler*). SDL_Halt is a function in the run-time library with the following prototype:

```
void SDL_Halt (void);
```

Execution with the Library Performance Simulation

As mentioned earlier, you may use the library *PerformanceSimulation* to speed up the execution of a performance simulation. This library has the same properties as the library *Simulation*, except that it does not contain the monitor system, which may increase the execution speed by a factor of 10 to 15.

Use the library *Simulation* to debug and verify the simulation model. Then select the library *PerformanceSimulation* in the make dialog in the Organizer (see "Make" on page 119 in chapter 2, *The Organizer*) and press the button *Make*. The thereby generated performance simulation may be executed from the OS as an ordinary program.