

OBJECT ORIENTED MODELING AND SIMULATION

Models with Standard Resources

Dr. José M. Garrido
Department of Computer Science

Spring 2017

College of Computing and Software Engineering
Kennesaw State University

© 2016 J. M. Garrido

1 Introduction

Resource management involves the manipulation of resources by the various processes in a model. There are two important types of resources in OOSimL:

1. *Standard resources* that are accessed in a mutually exclusive manner by processes. These resource objects represent a finite pool of resource units and are created with the *Res* class.
2. *Detachable resources* that are *produced* and *consumed* by different processes. These resource objects represent infinite pools of resource units and are created with the *Bin* class.

Resource management of standard resources involves resource allocation to processes, and resource de-allocation from processes. The models that include resource management also deal with process interactions that are present when processes compete for resources.

The type of synchronization involved with this process interaction is . Resources can be held by only one process at a time; the other processes requesting the same resources must wait.

As mentioned in previous chapters, resources are passive components of a model; in the software implementation of resource pools are passive objects.

2 Resource Manipulation

A system can have several resource types defined, also called resource pools. Each resource pool can contain one or more resource units or resource items; these resource units can be requested by several processes in the system. A process that needs a number of resource units from a resource pool follows the following steps:

1. Request a specific number of resource units from resource pool.
2. If there are sufficient resource units available, acquire these; otherwise wait in a hidden queue, which belongs to the resource pool.
3. Use the acquired resource units for a finite interval.
4. Release all or some of the acquired resource units.

Resource units of a specific resource pool can only be acquired by a single process at a time so they are considered mutual exclusive resources. These resource units are acquired and released by the same process.

For every resource pool, there are a specified total number of resource units. When the number of resource units in the resource pool is less than the number requested by a process, the request cannot be granted immediately. In this case, the object is placed in a hidden priority queue and suspended. This waiting process will be reactivated when another process holding resources units (of the same resource pool) releases them.

When the number of resource units is small, there will be a relatively large group of processes waiting to acquire these resource units.

A process can hold some resource units of a resource pool and be requesting units of a different resource pool. The process will usually be competing with other process for these resources.

These resources are acquired in a mutual exclusive manner. Only one process at a time can acquire a particular resource and the type of process synchronization involved is known as mutual exclusion.

2.1 Software Implementation of Resources

A resource pool is represented by an object of the resource class *Res*. To manipulate resources, a simulation model needs to define the following sequence of steps:

1. Declare references to resource objects, each representing a resource type or resource pool. These are objects of class *Res*.
2. Create the resource objects, each with a finite number of total resource units, this number represents the maximum number of units of the resource pool.
3. In the main body of the process definitions:
 - (a) Acquire a specific number of resource units of a resource pool.
 - (b) Use the resource units for a finite time interval.
 - (c) Release all or part of the resource units acquired.

When a process acquires a number of resource units of a resource pool, the number of available resource units is reduced accordingly. For example, if the initial total number of chairs in a model is 15, then after process *P* acquires 10 chairs, there will be only 5 chairs left (available). The resource pool is defined with 15 chairs. If another process, *Q*, requests 7 chairs, it has to wait until process *P* releases one or more chairs.

After an object releases a number of resource units, a waiting object is reactivated and will attempt again to acquire the number of resource units initially requested. If there are not sufficient resource units available, it will be suspended again and put back into the resource priority queue.

Some internal checks are carried out by the simulation executive. If a process attempts to acquire a number of resource units greater than the maximum defined for that resource pool, an error message is issued and control will return to the process. In a similar manner, when a process attempts to release a number of resource units greater than the number that the process holds, a message is issued and control is returned to the process.

2.1.1 Creating Standard Resources

Creating a standard resource pool involves creating an object of class *Res*. This object represents a finite resource pool with a specified number of resource units.

The constructor for creating a resource pool of this class requires the name of the resource pool and the pool size. The pool size defines the total number of available resource units in the resource pool.

The following lines of code declare and create a standard resource pool, *chairs*, with a title “Customer Chairs” and a size of 30 resource units.

```
define chairs of class Res    // declare ref resource pool
...
// create resource pool
create chairs of class Res using "Customer Chairs", 30
```

2.1.2 Available Resource Units

The **assign available** statement gets the number of available resource units in the resource pool and assigns this number to the specified variable. The general structure of this statement follows.

assign available units of \langle *ref_variable* \rangle **to** \langle *var_name* \rangle

The following lines of code get the current number of available resource units in the resource pool *chairs*, which was defined above, and assigns this value to variable *num_res*.

```
define num_res of type integer // number of resource units
...
assign available units of chairs to num_res
```

2.1.3 Acquire Resource Units

The **acquire** statement allows the requesting process to acquire a specified number of units from the specified resource pool. If the resource pool does not have all the requested resource units available, the requesting process is suspended.

Thus, the statement allows the process to acquire a specified number of resource units from a resource pool only if there are sufficient resource units available. The general form of the statement is:

```
acquire < int_variable > units from < ref_variable >  
request < int_variable > units from < ref_variable >
```

In the following lines of code, the process attempts to acquire 10 resource units from the resource pool *chairs* (defined above).

```
acquire 10 units from chairs  
// execute instructions when resources are acquired
```

2.1.4 Release Units of a Resource

This statement releases the specified number of resource units from the process, and returns these to the resource pool. This deallocates the specified number of resource units that the process currently holds. The statement may cause reactivation of any suspended processes that are waiting for resources. One or more of the reactivated processes can then acquire the requested resources units now available, if there are sufficient resource units.

```
release < int_variable > units of < ref_variable >
```

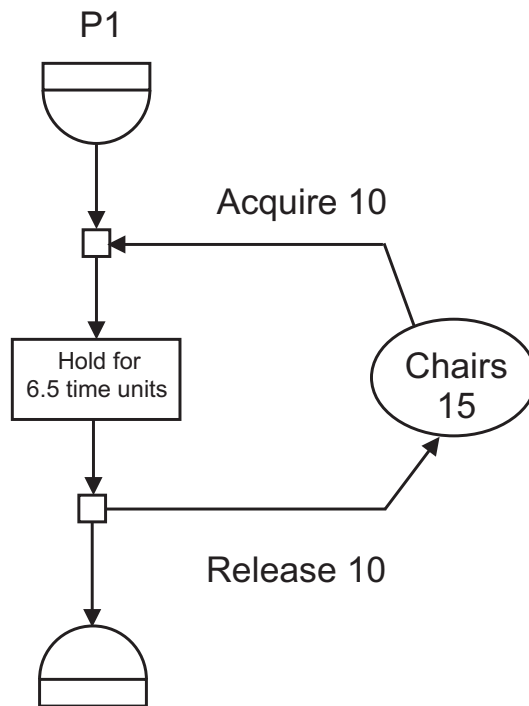
In the following line of code, a process releases 10 resource units from the resource pool *chairs* (defined above).

```
release 10 units of chairs
```

2.2 Using Resources

As mentioned previously, a process will normally acquire a certain number of resource units from a resource pool, use these resource units for a finite time interval, then release the resource units.

The following lines of code show the allocation and de-allocation of resources units by a process, *P1*. This process, *P1*, acquires 10 resource units of *chairs*; it

Figure 1: Activity diagram using *chairs* resource pool

holds the resource units for 6.5 time units; after this interval, it releases the 10 resource units of *chairs*.

```

. . .
acquire 10 units of chairs
hold self for 6.5          // use the resource units
release 10 units of chairs
. . .

```

The process attempts to acquire 10 resource units of *chairs*. If the request for resources of *chairs* is not successful, then the process is automatically suspended and placed in a hidden queue that belongs to object *chairs*. When the process is reactivated, it will again try to acquire the specified number of resource units.

When the process is able to acquire 10 resource units of *chairs*, it holds the resource units for 6.5 time units; this represents usage of the resource units for a finite interval. When this interval elapses, the process releases 10 resource units of *chairs* and the process continues with its other activities. Figure 1 illustrates the

use of the resource pool, *chairs*.

3 Model of a Warehouse System

A warehouse receives two types of trucks, small and big, that arrive periodically with goods. Each truck needs an unloading bay to dock and to unload the goods. Small trucks need two workers to unload, big trucks need three workers. The warehouse has a limited number of unloading bays and a limited number of workers. Trucks may need to wait if there are no available unloading bays and/or not sufficient available workers. Each type of truck has a mean arrival rate and a mean unloading time interval. The simulation model of the Warehouse system consists of the following components:

- Process definitions
 - Small trucks
 - Big trucks
 - Arrivals of small trucks
 - Arrivals of big trucks
 - The main process
- Resource definitions
 - Unloading bays
 - Workers

The inter-arrival intervals for the trucks follow an exponential distribution and the unloading intervals follow a uniform distribution. The truck objects acquire one resource unit of the unloading bay resource, then acquire the necessary resource units of workers. After a truck object unloads, it releases the resources units, departs, and terminates.

Figure 2 shows activity diagram that describes the behavior of a big truck. The following sequence of activities is performed by big trucks:

1. After arriving, acquire one unloading bay if available, otherwise wait.
2. Dock into bay, this takes a constant time interval.
3. Acquire 3 workers if available, otherwise wait.
4. Unload, takes a random time interval (from a uniform distribution).

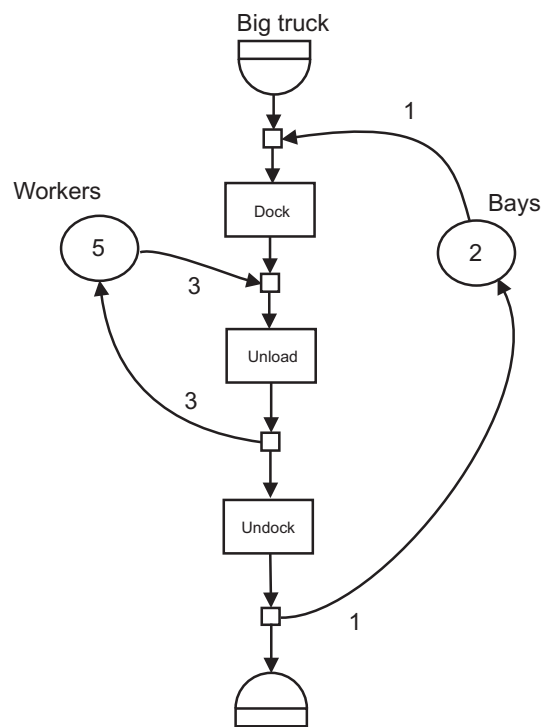


Figure 2: Activity diagram of big truck

5. Release workers.
6. Undock, takes a constant time interval.
7. Release unloading bay.
8. Depart and terminate.

Small trucks perform a similar sequence of activities. The following sequence of activities defines the behavior of small trucks.

1. After arriving, acquire one unloading bay if available, otherwise wait
2. Acquire 2 workers if available, otherwise wait
3. Unload, takes a random time interval (from a uniform distribution)
4. Release workers
5. Release unloading bay
6. Depart and terminate

Listing 1 shows the trace listing of the output from a simulation run of the model. The trace in the output shows the relevant events, such as:

- Processes (trucks) start to request resources
- Resources are allocated to the processes
- Other processes start to wait for available resources
- Processes terminate after they have completed all of their activities

Listing 1 Trace of a simulation run of the Warehouse model.

```

OOSimL model: Busy Warehouse System
Simulation date: 11/13/2016 time: 4:09
----- TRACE -----
Time: 0 Warehouse model holds for 625.65
Time: 0 Big T Arrivals holds for 18.629
Time: 0 Small T Arrivals holds for 17.151
Time: 17.151 Small T Arrivals holds for 1.461
Time: 17.151 Struck1 requesting 1 units of Bays
Time: 17.151 Struck1 acquired 1 units of Bays

```

Time: 17.151 Struck1 requesting 2 units of Workers
Time: 17.151 Struck1 acquired 2 units of Workers
Time: 17.151 Struck1 holds for 8.264
Time: 18.612 Small T Arrivals holds for 31.645
Time: 18.612 Struck2 requesting 1 units of Bays
Time: 18.612 Struck2 acquired 1 units of Bays
Time: 18.612 Struck2 requesting 2 units of Workers
Time: 18.612 Struck2 acquired 2 units of Workers
Time: 18.612 Struck2 holds for 6.821
Time: 18.629 Big T Arrivals holds for 19.638
Time: 18.629 Btruck1 requesting 1 units of Bays
Time: 18.629 Btruck1 deactivated
Time: 25.416 Struck1 releasing 2 units of Workers
Time: 25.416 Struck1 releasing 1 units of Bays
Time: 25.416 Struck1 terminating
Time: 25.416 Btruck1 acquired 1 units of Bays
Time: 25.416 Btruck1 holds for 1.45
Time: 25.433 Struck2 releasing 2 units of Workers
Time: 25.433 Struck2 releasing 1 units of Bays
Time: 25.433 Struck2 terminating
Time: 26.866 Btruck1 requesting 3 units of Workers
Time: 26.866 Btruck1 acquired 3 units of Workers
Time: 26.866 Btruck1 holds for 12.879
Time: 38.268 Big T Arrivals holds for 6.08
Time: 38.268 Btruck2 requesting 1 units of Bays
Time: 38.268 Btruck2 acquired 1 units of Bays
Time: 38.268 Btruck2 holds for 1.45
Time: 39.718 Btruck2 requesting 3 units of Workers
Time: 39.718 Btruck2 deactivated
Time: 39.745 Btruck1 releasing 3 units of Workers
Time: 39.745 Btruck1 holds for 1.15
Time: 39.745 Btruck2 acquired 3 units of Workers
Time: 39.745 Btruck2 holds for 11.168
Time: 40.895 Btruck1 releasing 1 units of Bays
Time: 40.895 Btruck1 terminating
Time: 44.348 Big T Arrivals holds for 32.315
Time: 44.348 Btruck3 requesting 1 units of Bays
Time: 44.348 Btruck3 acquired 1 units of Bays
Time: 44.348 Btruck3 holds for 1.45
Time: 45.798 Btruck3 requesting 3 units of Workers
Time: 45.798 Btruck3 deactivated
Time: 50.257 Small T Arrivals holds for 2.214
Time: 50.257 Struck3 requesting 1 units of Bays
Time: 50.257 Struck3 deactivated
Time: 50.912 Btruck2 releasing 3 units of Workers

```

Time: 50.912 Btruck2 holds for 1.15
Time: 50.912 Btruck3 acquired 3
. . .
Time: 612.885 Big T Arrivals holds for 20.089
Time: 612.885 Btruck43 requesting 1 units of Bays
Time: 612.885 Btruck43 acquired 1 units of Bays
Time: 612.885 Btruck43 holds for 1.45
Time: 614.335 Btruck43 requesting 3 units of Workers
Time: 614.335 Btruck43 deactivated
Time: 618.338 Btruck42 releasing 3 units of Workers
Time: 618.338 Btruck42 holds for 1.15
Time: 618.338 Btruck43 acquired 3 units of Workers
Time: 618.338 Btruck43 holds for 9.807
Time: 619.488 Btruck42 releasing 1 units of Bays
Time: 619.488 Btruck42 terminating
Time: 625.65 Warehouse model holds for 14.283
Time: 628.145 Btruck43 releasing 3 units of Workers
Time: 628.145 Btruck43 holds for 1.15
Time: 629.295 Btruck43 releasing 1 units of Bays
Time: 629.295 Btruck43 terminating
Time: 632.974 Big T Arrivals holds for 5.943
Time: 632.974 Btruck44 requesting 1 units of Bays
Time: 632.974 Btruck44 acquired 1 units of Bays
Time: 632.974 Btruck44 holds for 1.45
Time: 634.424 Btruck44 requesting 3 units of Workers
Time: 634.424 Btruck44 acquired 3 units of Workers
Time: 634.424 Btruck44 holds for 10.602
Time: 638.918 Big T Arrivals holds for 4.21
Time: 638.918 Btruck45 requesting 1 units of Bays
Time: 638.918 Btruck45 acquired 1 units of Bays
Time: 638.918 Btruck45 holds for 1.45
Time: 639.933 Small T Arrivals holds for 3.179
-----

End Simulation of Busy Warehouse System
date: 11/13/2016 time: 4:10

```

Listing 2 shows the listing with the summary statistics of the simulation run. The listing displays resource types used, the random number generators used, the total number of trucks serviced of each type, and the average wait period, for each type.

Listing 2 Summary statistics of a simulation run of the Warehouse model.

00SimL model: Busy Warehouse System

```

Simulation date: date: 11/13/2016 time: 4:10
-----STATISTICS REPORT-----
-----
Res Bays
Bays usage: 0.8585299019794168
  avg num Bays items used: 1.8061836569706509

% of time of Bays resource usage: 0.9506562620761964
Avg waiting time for Bays: 2.5500048924851084
Avg. num waiting processes: 2.5683087642958395
% of time spent waiting for Bays resources: 0.7695569252035672
-----
Res Workers
Workers usage: 0.6476054685731456
  avg num Workers items used: 3.5306934454374033

% of time of Workers resource usage: 0.9171080392295521
Avg waiting time for Workers: 7.638598375234128
Avg. num waiting processes: 1.0
% of time spent waiting for Workers resources: 0.3103504064311
-----
Random generator: Small truck Inter-arr
Distribution: Negative exponential
Number of obs: 38
Seed: 15
Mean: 13.1
-----
Random generator: Big truck Inter-arr
Distribution: Negative exponential
Number of obs: 46
Seed: 7
Mean: 14.2
-----
Random generator: Small truck service
Distribution: Uniform
Number of obs: 37
Seed: 0
Lower bound: 5.3
Upper bound: 9.35
-----
Random generator: Big truck service
Distribution: Uniform
Number of obs: 45
Seed: 0
Lower bound: 8.5

```

Upper bound: 14.5

End Simulation of Busy Warehouse System
date: 11/13/2016 time: 4:10

Total small trucks serviced: 36
Average period small truck spends in warehouse: 24.759
Small truck average wait period: 17.113
Total number of big trucks serviced: 43
Big truck average wait period: 21.189
Avg per btruck in warehouse: 34.101

Listing 3 shows the source code for class *Btruck*. The complete model implementation of the Warehouse model is stored in the archive file, *wareh.jar*. This archive includes the following source files: *Wareh.osl*, *Sarrivals.osl*, *Barrivals.osl*, *Struck.osl*, and *Btruck.osl*.

Listing 3. Code implementation of class *Btruck* of the Warehouse model.

```
use all psimjava
description
    A model of a busy warehouse system
    Small and big trucks constantly arrive to a warehouse to
    unload goods. Each truck needs an unloading bay to dock
    and start unloading. Big trucks need three workers and a
    constant time interval for docking and undocking. Trucks
    may need to wait for an available unloading bay and/or
    wait for available workers.
    Both types of trucks have different mean arrival rates
    and different mean unloading periods.
    The warehouse has 2 unloading bays and 5 workers.

    OOSimL model, J. Garrido, updated September 2015.

    class:    Btruck
    File:     Btruck.osl
    This class defines the behavior of big trucks

*/
public class Btruck as process is
    private
    constants
        define DOCK_INTERV = 1.45 of type double
```

```

    define UNDOCK_INTERV = 1.15 of type double
variables
    define truckNum of type integer    // Truck number
    define arrivalTime of type double  // arrival time
    define service_dur of type double  // unloading interv
    define bname of type string
//
public
function initializer parameters bname of type string,
                        unload_dur of type double is
begin
    call super using bname
    set truckNum = Wareh.get_bnumArr()
    assign simulation clock to arrivalTime
    set service_dur = unload_dur      // unloading interval
    set bname = bname
endfun initializer
//
function get_arrivalT return type double
is begin
    return arrivalTime
endfun get_arrivalT
//
function get_serv_dur return type double is
begin
    return service_dur
endfun get_serv_dur
//
function Main_body is
    variables
        define start_wait of type double // starts wait
        define wait_p of type double     // wait interval
        define bsojourn of type double   // total interval
        define simclock of type double
    begin
        // setb the start wait time for this truck
        assign simulation clock to start_wait
        //
        display bname, " request unloading bay at ", start_wait
        // tracewrite bname, " request unloading bay "
        // process will wait for an unload bay
        //
        acquire 1 units from Wareh.bays      // 1 unloading bay
        //
        assign simulation clock to simclock

```

```

display bname, " acquired unloading bay at: ", simclock
//
// docking
hold self for DOCK_INTERV
//
display bname, " requesting 3 workers at ", simclock
//
// process will wait until 2 workers become available
//
acquire 3 units from Wareh.workers    // 3 workers
//
assign simulation clock to simclock
display bname, " acquired workers at: ", simclock
//
// wait period for this truck
set wait_p = get_clock() - start_wait
// accumulate truck wait interval
call Wareh.accumBWait using wait_p    // accumulate wait
//
// now the unloading service of the truck
hold self for service_dur    // unloading interval
//
// service completed, release workers
//
release 3 units of Wareh.workers
//
// undock
hold self for UNDOCK_INTERV
//
// ok, now release the bay
release 1 units of Wareh.bays
//
// increment num big trucks serviced
call Wareh.incr_bigTserv

// total time in warehouse: bSojourn
set bsojourn = get_clock() - arrivalTime
// accumulate sojourn time
call Wareh.accum_Bsojourn using bsojourn
//
assign simulation clock to simclock
display bname, " terminates at ", simclock
terminate self    //terminates itself
endfun Main_body
endclass Btruck

```

4 Allocating Resources with Priorities

In the model of the Warehouse system, the processes compete for mutual exclusive access to resources, all with the same default priority (priority 0). In larger models, processes may have different priorities.

A process that is assigned a lower priority than the default, needs to use a priority higher than zero.

Assigning a higher priority to a process affects the behavior of the simulation model because processes are placed by priority in the hidden resource queue when they need to wait for resources. Priority with value zero corresponds to the highest priority. Processes with higher priority will be given preference in the allocation of resources.

5 Deadlock

When competing with each other for several resources, a set of processes may reach a state of deadlock. Processes that have already acquired one or more resources and are requesting other resources, may have to wait indefinitely.

Every process in the set waits until another process releases the resources that it needs. The processes are actually waiting for each other.

The simulation language and libraries do not automatically avoid, prevent, or detect deadlock. The model designer has to apply one of many techniques for preventing or avoiding deadlock that are explained in specialized literature.

6 Summary

Standard resource manipulation involves basic resource synchronization with mutual exclusive resources. A resource type in a simulation model is defined as a resource pool with an initial number of available resource units of that resource type. Every resource pool is created as an object of the *Res* library class. The resource units can only be held by one process at a time. The process needs to acquire the resources, use the resources for some finite interval of time, and then release all or part of the number of resources held. When a process is waiting for available resources, it is suspended and placed (by priority) in a hidden queue owned by the resource pool.

Priorities in processes are important in allocating resources. Another important concept is the notion of deadlock. This may occur if the order of the allocation of resources of different types to set of processes is not carefully selected. Deadlock is

a condition of indefinite waiting for resources that other processes hold but will not release because they are also waiting for other resources.