

OBJECT ORIENTED MODELING AND SIMULATION

Models with Detachable Resources

Dr. José M. Garrido
Department of Computer Science

January 2017

College of Computing and Software Engineering
Kennesaw State University

© 2016 J. M. Garrido

1 Introduction

Detachable resources behave in a different manner compared to the standard resources. Objects of *detachable resources* are *infinite containers* created with an initial number of resource units. The resource container has no upper limit in the number of resource units in it. With these resources another type of process synchronization is involved, the producer-consumer resource *synchronization*.

2 Infinite Containers

A process can add more resource units and another process can remove resource units. A process will not normally return the resource units taken from the container. Some processes behave as producers of resource items, other process behave as consumers.

A producer generates resource units that are placed into the resource container and the consumer takes resource units from the container.

The consumer process is suspended if there are no resource units available in the resource container. The producer process increases the number of available resource units and the consumer process decreases the number of resource units in the resource container.

This handling of detachable resources involves a type of *synchronization* between the producer process and the consumer process. The simplest type of synchronization is in the form of *cooperation* between the consumer and producer processes.

3 Producer and Consumer Processes

A producer process generates resource items and this activity takes a finite interval of time. The process then places or *gives* the resource items into a resource container. Figure 1 shows a simplified activity diagram that illustrates a producer process placing k resource items into a container. The process will never have to wait because the resource object is an infinite container.

A consumer process *takes* a number of resource items from a resource container and then spends some finite interval of time using these items. Figure 2 shows a consumer process that takes k resource items from a resource container.

If the number of available resource items in the resource container is not sufficient for the consumer's request, the consumer process is suspended (and placed in a hidden queue that belongs to the resource container object). The consumer process

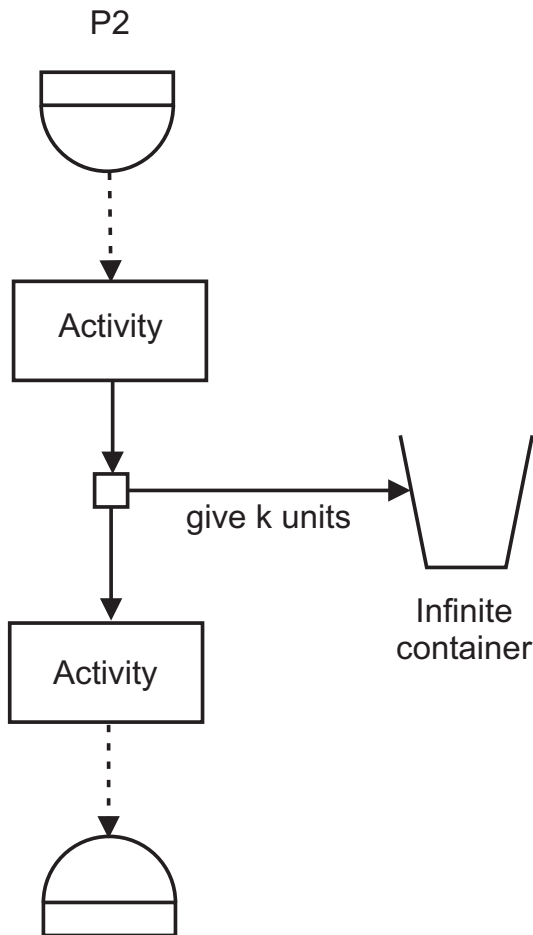


Figure 1: A producer process and a detachable resource container

will attempt again to take the specified number of resources when it is reactivated, until it finally gets the requested resource items.

4 Implementing Detachable Resources

The following steps are carried out to implement a detachable resource container in a simulation model.

1. Declare the reference to the resource container object of class *Bin*.
2. Create the resource container with a title and an initial number of resource

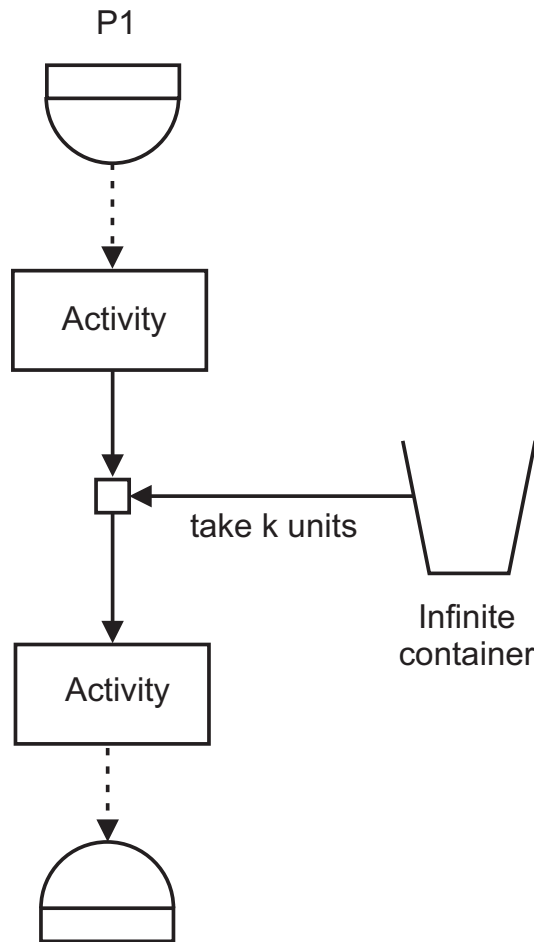


Figure 2: Activity diagram for a consumer process

items.

3. A producer process:

- (a) Spends a finite interval to produce a number of resource items.
- (b) Places, or gives, a number of resource items into the resource container.

4. A consumer process:

- (a) Removes, or takes, a number of resource items from the resource container.
- (b) Spends a finite interval of time to use (consume) the resource items.

4.1 OOSimL Implementation

4.1.1 Create Detachable Resource Objects

Class *Bin* is used to declare references and create objects of a detachable resource container. A title and the number of initial number resource units in the resource container are required.

The following lines of code declare and create a detachable resource container, *cust_cont*, with title “Parts” and an initial size of 10 resource units.

```
define cust_cont of class Bin // declare resource container
...
// create the resource container (bin container)
create cust_cont of class Bin using "Parts", 10
```

4.1.2 Available Units in a Resource Container

The **assign available** statement gets the number of available resource units in a detachable resource object, and assigns this number to the specified integer variable. The general form of this statement follows.

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

The following lines of code gets the current number of available resource units in the resource container *cust_cont*, which was defined above, and assigns this number to the integer variable *num_units*.

```
define num_units of type integer
...
assign available units of cust_cont to num_units
```

4.1.3 Take Units from a Resource Container

A process can remove a specified number of units from the detachable resource object using the **take** statement. If there are sufficient units in the resource container, the process takes these units and continues normally. When the operation completes, the number of resource units in the resource container is decreased by the number of resource items taken by the process.

If there are not sufficient resource units available, the process is suspended and waits for available resource units. The general form of the **take** statement follows.

take \langle int_variable \rangle **units from** \langle ref_variable \rangle

The following lines of code allow a process to take 10 resource units from the *cust_cont* container (defined above).

```
take 10 units from cust_cont
// execute after resource units are taken
```

4.1.4 Give Resource Units to a Resource Container

A process uses the **give** statement to insert (or give) a specified number of resource units to a detachable container object. This places a specified number of resource units into the container. The number of units in the container is then updated. The general form of the statement follows.

```
give < int_variable > units to < ref_variable >
```

In the following lines of code, a process places 8 resource units, which is stored in integer variable *num_units*, into the *cust_cont* resource container (defined above).

```
define num_units of type integer
...
set num_units = 8
...
give num_units units to cust_cont
```

4.2 Producer Process

In the following statements, the behavior of a basic producer process is implemented. This is shown in Figure 1. The resource container was declared and created with name *steel_bolts*.

```
hold self for produce_items // interval to produce items
give k units to steel_bolts // deposit k items into container
```

In the code above, the producer process spends *produce_items* time units to produce the resource items. After this time interval has elapsed, the producer process gives two items to the resource container.

4.3 Consumer Process

The following lines of code implement the basic behavior of a consumer process, as shown in Figure 2.

```
// attempt to remove items from resource container
take k units from steel_bolts
hold self for consume_items // time interval to consume items
. . .
```

If there are not sufficient items in the resource container, the consumer process is suspended and placed in the hidden queue of the resource object. If there are sufficient items available in the resource container, the consumer continues normally and spends a time interval consuming the items. The number of items in the container is reduced accordingly.

A suspended consumer process is reactivated when it is at the head of the resource queue and another process has given items to the resource container. When this happens, the consumer process again tests if it can take two items from the resource container.

5 A Machine Parts-Replacement System

There are several machines in a shop, each having a part that fails periodically. The part that fails needs to be removed from the machine, and a replacement part installed, if available. This removal and replacement of parts is carried out by a repairman. The parts that fail are to be repaired by the repairman, who also has other jobs to carry out when not repairing parts.

In this model, there are two process definitions, *Machine* and *Repairman*; these are implemented as OOSimL classes. Several machine objects and one repairman object are created.

The model also includes two different resource containers, which are implemented as objects of class *Bin*. These resource containers are:

- The parts that fail, *fault_parts*;
- The replacement parts, *rep_parts*. These are the parts that have been repaired.

The machine processes are producers of damaged parts; they give items to the *fault_parts* resource container. The machine processes are also consumers of replacement parts; they take items from the *rep_parts* resource container. Figure 3 shows a simplified activity diagram of the machine process with the two resource containers.

The repairman process is a consumer of damaged parts; it takes items from the *fault_parts* resource container. The repairman process is also a producer of replacement parts; it gives items to the *rep_parts* resource container.

Listing 1 shows the listing of the results summary of a simulation run with this model. The output listing includes the resource container usage and performance

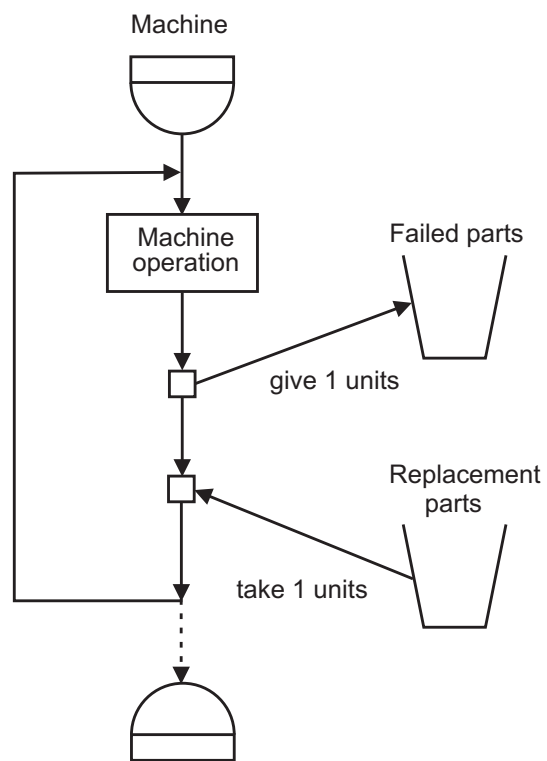


Figure 3: Activity diagram of the machine process

metrics; these are average machine up period, the average down period, the machine utilization, and the repairman utilization.

Listing 1 Summary results of a simulation run of the model.

```

OOSimL model: Machine Parts Replacement System
Simulation date: date: 9/28/2016 time: 11:34
Mean oper. interval: 162.5 mean repair interval: 13.3
-----STATISTICS REPORT-----
-----
Replacement parts
Number of completed operations: 51
Max items: 15
Avg num available items: 13.850104111136883

-----
-----
Fault parts
Number of completed operations: 51
Max items: 3
Avg num available items: 0.7108392461027906

-----
-----
Random generator: repair interval
Distribution: Normal
Number of obs: 51
Seed: 25
Mean: 13.3
Standard deviation: 0.5
-----
Random generator: Operation interval Machine 3
Distribution: Normal
Number of obs: 18
Seed: 19
Mean: 162.5
Standard deviation: 6.5
-----
Random generator: Operation interval Machine 2
Distribution: Normal
Number of obs: 18
Seed: 13
Mean: 162.5
Standard deviation: 6.5
-----

```

```

Random generator: Operation interval Machine 1
Distribution: Normal
Number of obs: 18
Seed: 7
Mean: 162.5
Standard deviation: 6.5

```

```

-----
Random generator: background task
Distribution: Uniform
Number of obs: 67
Seed: 0
Lower bound: 28.45
Upper bound: 62.3
-----

```

```

End Simulation of Machine Parts Replacement System
date: 9/28/2016 time: 11:34

```

```

Total machines serviced: 3
Average down period: 15.299999999998931
Average up period: 2782.15476170466
Average machine utilization: 0.7273607220142901

```

Listing 2 shows the trace output of the simulation run. The listing shows the various time events, mainly when each process starts to wait for a requested part and when it gets the part.

Listing 2 Trace output of a simulation run of the model.

```

OOSimL model: Machine Parts Replacement System
Simulation date: 9/28/2016 time: 11:34
----- TRACE -----
Time: 0 Mean oper. interval: 162.5 mean repair interval: 13.3
Time: 0 Machine Shop holds for 3825
Time: 0 Machine 1 to oper for 167.9938394270824
Time: 0 Machine 1 holds for 167.994
Time: 0 Machine 2 to oper for 173.4387407155666
Time: 0 Machine 2 holds for 173.439
Time: 0 Machine 3 to oper for 171.1538128332344
Time: 0 Machine 3 holds for 171.154
Time: 0 Repairman to perform background task
Time: 0 Repairman holds for 53.208
Time: 53.208 Repairman to perform background task
Time: 53.208 Repairman holds for 41.747

```

Time: 94.955 Repairman to perform background task
Time: 94.955 Repairman holds for 55.054
Time: 150.009 Repairman to perform background task
Time: 150.009 Repairman holds for 50.081
Time: 167.994 Machine 1 failed
Time: 167.994 Machine 1 holds for 0.45
Time: 168.444 Machine 1 gives 1 of Fault parts
Time: 168.444 Replacement parts available: 15
Time: 168.444 Machine 1 requesting 1 of Replacement parts
Time: 168.444 Machine 1 took 1 of Replacement parts
Time: 168.444 Machine 1 holds for 0.45
Time: 168.894 Machine 1 took replacement
Time: 168.894 Machine 1 to oper for 168.43369516189736
Time: 168.894 Machine 1 holds for 168.434
Time: 171.154 Machine 3 failed
Time: 171.154 Machine 3 holds for 0.45
Time: 171.604 Machine 3 gives 1 of Fault parts
Time: 171.604 Replacement parts available: 14
Time: 171.604 Machine 3 requesting 1 of Replacement parts
Time: 171.604 Machine 3 took 1 of Replacement parts
Time: 171.604 Machine 3 holds for 0.45
Time: 172.054 Machine 3 took replacement
Time: 172.054 Machine 3 to oper for 157.22493327498918
Time: 172.054 Machine 3 holds for 157.225
Time: 173.439 Machine 2 failed
Time: 173.439 Machine 2 holds for 0.45
Time: 173.889 Machine 2 gives 1 of Fault parts
Time: 173.889 Replacement parts available: 13
Time: 173.889 Machine 2 requesting 1 of Replacement parts
Time: 173.889 Machine 2 took 1 of Replacement parts
Time: 173.889 Machine 2 holds for 0.45
Time: 174.339 Machine 2 took replacement
Time: 174.339 Machine 2 to oper for 159.86357967388378
Time: 174.339 Machine 2 holds for 159.864
Time: 200.09 Repairman num of faulty parts 3
Time: 200.09 Repairman requesting 1 of Fault parts
Time: 200.09 Repairman took 1 of Fault parts
Time: 200.09 Repairman to repair part
Time: 200.09 Repairman to repair task for 13.40529040458385
Time: 200.09 Repairman holds for 13.405
Time: 213.496 Repairman done repairing
Time: 213.496 Repairman gives 1 of Replacement parts
Time: 213.496 Repairman num of faulty parts 2
Time: 213.496 Repairman requesting 1 of Fault parts
. . .

Time: 2843.673 Repairman took 1 of Fault parts
Time: 2843.673 Repairman to repair part
Time: 2843.673 Repairman to repair task for 13.620773138276915
Time: 2843.673 Repairman holds for 13.621
Time: 2857.294 Repairman done repairing
Time: 2857.294 Repairman gives 1 of Replacement parts
Time: 2857.294 Repairman to perform background task
Time: 2857.294 Repairman holds for 43.982
Time: 2901.276 Repairman to perform background task
Time: 2901.276 Repairman holds for 58.623
Time: 2959.899 Repairman to perform background task
Time: 2959.899 Repairman holds for 52.357
Time: 3012.256 Repairman to perform background task
Time: 3012.256 Repairman holds for 36.656
Time: 3048.912 Repairman to perform background task
Time: 3048.912 Repairman holds for 41.792
Time: 3090.705 Repairman to perform background task
Time: 3090.705 Repairman holds for 41.374
Time: 3132.079 Repairman to perform background task
Time: 3132.079 Repairman holds for 28.584
Time: 3160.662 Repairman to perform background task
Time: 3160.662 Repairman holds for 57.878
Time: 3218.54 Repairman to perform background task
Time: 3218.54 Repairman holds for 57.038
Time: 3275.578 Repairman to perform background task
Time: 3275.578 Repairman holds for 32.777
Time: 3308.355 Repairman to perform background task
Time: 3308.355 Repairman holds for 47.549
Time: 3355.905 Repairman to perform background task
Time: 3355.905 Repairman holds for 42.427
Time: 3398.332 Repairman to perform background task
Time: 3398.332 Repairman holds for 54.187
Time: 3452.519 Repairman to perform background task
Time: 3452.519 Repairman holds for 47.558
Time: 3500.077 Repairman to perform background task
Time: 3500.077 Repairman holds for 49.724
Time: 3549.801 Repairman to perform background task
Time: 3549.801 Repairman holds for 55.71
Time: 3605.511 Repairman to perform background task
Time: 3605.511 Repairman holds for 44.614
Time: 3650.125 Repairman to perform background task
Time: 3650.125 Repairman holds for 28.934
Time: 3679.059 Repairman to perform background task
Time: 3679.059 Repairman holds for 48.797
Time: 3727.856 Repairman to perform background task

```

Time: 3727.856 Repairman holds for 59.229
Time: 3787.085 Repairman to perform background task
Time: 3787.085 Repairman terminating
End Simulation of Machine Parts Replacement System
date: 9/28/2008 time: 11:34

```

Listing 3 shows the implementation of class *Machine*. The source files with implementations for all classes are stored in archive file `preplace.jar`. This archive includes files: `Preplace.osl`, `Machine.osl`, and `Repairman.osl`.

Listing 3 Source code with implementation of class *Machine*.

```

use all psimjava
description
  A model of machine parts replacement system.
  Several machines in a shop each have a part that fails
  periodically. The part that fails needs to be removed from
  the machine, and a replacement part, if available, installed.
  The parts that fail are to be repaired by a repairman, who
  also have other jobs to carry out when not repairing parts.

  OOSimL model, J. Garrido, updated September 2015.
  This class defines behavior of Machine objects
*/
class Machine as process is
private
variables
  define machineNum of type integer    // machine number
  define mean_oper of type double      // operation interval
  define st_dev of type double         // standard deviation
  define simclock of type double       // current sim time
  define removal_per = 0.45 of type double
  define replace_per = 0.45 of type double
  define start_wait of type double     // start to wait
  define wait_p of type double         // wait interval
  define total_oper of type double
  define mname of type string          // name of machine
object references
  define oper_rand of class Normal     // random generator
public
  function initializer parameters pname of type string,
    oper_dur of type double, std of type double
  is
  begin

```

```

call super using pname
set mean_oper = oper_dur
set st_dev = std
create oper_rand of class Normal
    using "Operation interval " concat pname,
    mean_oper, st_dev
endfun initializer
//
function Main_body is
variables
    define operation_per of type double // operation interval
    define opercmp of type double      // completion time

    define rep_flag = true of type boolean
begin
    set mname = get_name()
    set total_oper = 0.75 * Preplace.simPeriod
    assign simulation clock to simclock
    while simclock < total_oper and rep_flag do
        // display mname," starting at ", simclock
        // tracewrite mname," starting"
        assign random value from oper_rand to operation_per
        display mname, " to oper for ", operation_per,
            " at ", simclock
        tracewrite mname, " to oper for ", operation_per
        //
        // completion of operation
        set opercmp = simclock + operation_per
        if ( opercmp < total_oper)
        then
            hold self for operation_per // operation interval
            // part failed
            assign simulation clock to simclock
            display mname, " failed at ", simclock
            tracewrite mname, " failed"
            //
            // remove faulty part and replace
            call remov_rep
            //
            // now compute this wait period
            set wait_p = simclock - start_wait
            add wait_p to Preplace.adown // accumulate wait
            // accumulate operation intervals
            add operation_per to Preplace.aserv
        else

```

```

        set rep_flag = false
    endif
endwhile
display mname, " terminating at: ", simclock
terminate self
endfun Main_body
//
function remov_rep is
variables
    define removcmp of type double    // removal completion
    define availrep of type integer  // available rep parts
begin
    // now remove and replace part
    //
    assign simulation clock to simclock
    set start_wait = simclock
    set removcmp = simclock + removal_per
    if removcmp < total_oper
    then
        hold self for removal_per          // removal of part
        assign simulation clock to simclock
        display mname, " requesting replacement at ",
simclock
        //
        // place in container the damaged part to repair
        give 1 units to Preplace.fault_parts    //
        // process will try until replacement available
        assign available units of Preplace.rep_parts
        to availrep
        display "Rep parts available: ", availrep
        tracewrite "Replacement parts available: ", availrep
        //
        take 1 units from Preplace.rep_parts
        //
        hold self for replace_per
        assign simulation clock to simclock
        display mname, " took replacement at: ", simclock
        tracewrite mname, " took replacement"
    endif
endfun remov_rep
endclass Machine

```
