

PROCESS SYNCHRONOUS COMMUNICATION WITH OOSIML

Appying the OOSimL Simulation Language

José M. Garrido
Department of Computer Science

Spring 2019

College of Computing and Software Engineering
Kennesaw State University

© 2018 J. M. Garrido

1 Introduction

Synchronous cooperation results among two or more processes when executing a *joint activity* during a finite time interval called the *cooperation interval*. For the processes to carry out the joint activity, the simultaneous participation of the processes is required during the time interval of cooperation. At the end of this joint activity, the processes will continue independently with their individual activities.

This report discusses the principles of synchronous cooperation among processes, and presents a case study with a simulation model that implements synchronous communication among a set of sender and receive processes.

2 Joint Process Activities

In order for two processes to take part in a joint activity, one of the processes takes a more active role in the interaction and designated as the *master*; the other process is designated as the *slave* during the cooperation interval. The master is the dominant process and the slave behaves subordinated to the master process during the joint activity.

When the interaction starts, the slave process is suspended until the end of the cooperation interval. The master process then reactivates the slave process, and the two processes will continue their own independent activities. When the slave process is not available, the master process has to wait and is suspended until a slave process becomes available.

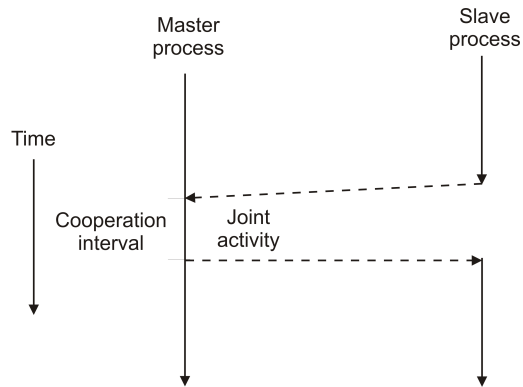


Figure 1: Interaction of master and slave processes

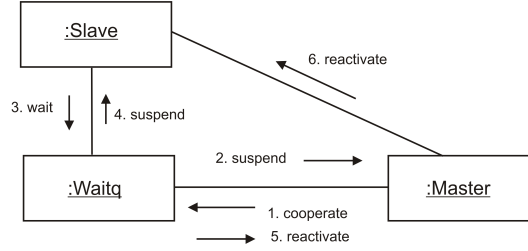


Figure 2: Cooperation of master and slave processes with a cooperation object

Figure 1 illustrates the master and slave processes cooperating in a joint activity. The figure shows the partial timelines of the two processes. The slave process becomes subordinated to the master process during the interval of cooperation.

3 Synchronization Mechanism

The synchronization mechanism needed for the process cooperation discussed is provided by objects of class *Waitq*, known as cooperation objects. These objects support the cooperation of multiple slave processes and multiple master processes. Every cooperation object has two hidden queues: the slave queue and the master queue.

At the beginning of the interaction, the slave processes are suspended and placed in the slave queue until the end of the cooperation interval. A master process then reactivates a slave process. At the end of the cooperation interval, the processes will continue their own independent activities. When one of or more of the slave processes are not available, a master process has to wait suspended and placed in the master queue.

A slave process that requests cooperation with a master process, executes the **wait** statement. A master process that requests cooperation with a slave process, executes the **cooperate** statement. Figure 2 is a UML diagram that shows the process cooperation between a master process and a slave process using a cooperation object of class *Waitq*.

Figure 3 shows the simulation activity diagram of two processes, P1 and P2, cooperating in a joint activity. Process P1 is the slave and process P2 is the master process. The two processes start with their own individual activities (*Act.1a* and *Act.2a*) then they initiate the cooperation by using the facilities provided by the cooperating object of class *Waitq*. Process P1 gets suspended by the cooperating object. After executing the joint activity, process P2 reactivates process P1. From

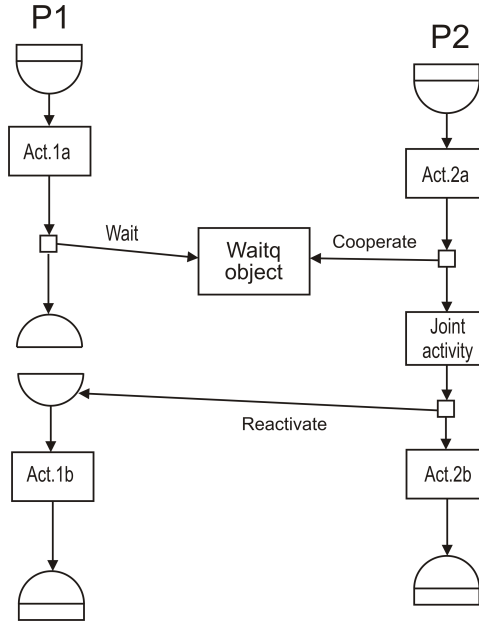


Figure 3: Activity diagram with cooperation of master and slave processes

this point on, both processes continue with their independent activities, *Act.1b* and *Act.2b*.

3.1 Creating Cooperation Objects in OOSimL

Cooperation objects are created with class *Waitq*. The objects are created and initialized with a title and an optional maximum number of priorities to use. Cooperation objects have two internal priority queues, one for the *master* processes and one for the *slave* processes.

The following lines of code declare a cooperation object, *coopt_cust*, and create the object with title “Channel A”, and with the default number of priorities.

```

define coopt_cust of class Waitq
...
create coopt_cust of class Waitq using "Channel A"

```

3.2 Wait to Cooperate

The **wait** statement places current process in the slave queue of the cooperation object and suspends it to wait for a master process. This allows a slave process

to cooperate with a master process, when there is a master process available. The general form of this statement follows.

wait for master in $\langle \text{ref_variable} \rangle$

The following line of code allows a slave process to wait to cooperate with a master process using the cooperation object *coopt_cust* (defined above).

```
wait for master in coopt_cust    // wait for master process
...
```

3.3 Cooperate with Slave Process

The **cooperate** statement allows a master process to attempt cooperation with slave processes. A slave process is retrieved from the slave queue of the cooperation object, if this queue is not empty. If there is no slave process available in the slave queue, the master process that executes this statement is suspended and placed in the master queue. The general form of the statement follows.

cooperate with slave $\langle \text{ref_variable} \rangle$
of class $\langle \text{cl_name} \rangle$ **in** $\langle \text{ref_variable} \rangle$

The following lines of code allow a master process to attempt cooperation with a slave process using the *coopt_cust* synchronization object. When the cooperation becomes possible, a slave process is removed from the slave queue, the master process continues executing normally. Otherwise, the process is suspended to wait for a slave process.

```
define custobj of class Customer
...
cooperate with slave custobj of class Customer in coopt_cust
// execute when a slave is found
```

3.4 Length of Slave Queue

The **assign length** statement assigns the length (number of processes) in the slave queue to a specified variable. This value represents the number of slave processes waiting on the cooperation (synchronization) object. The general form of the statement follows.

assign length of slave queue $\langle \text{ref_variable} \rangle$ **to** $\langle \text{var_name} \rangle$

The following lines of code gets the number of slave processes waiting in the *coopt_cust* object defined above and stores the value in variable *num_slaves*.

```
define num_slaves of type integer
...
assign length of slave queue coopt_cust to num_slaves
```

3.5 Length of Master Queue

The **assign length** statement gets the length of the master queue (i.e., the number of master processes waiting on the cooperation object), and assigns it to the specified variable. The general form of the statement follows.

```
assign length of master queue < ref_variable >
to < var_name >
```

The following lines of code get the number of master processes waiting in master queue of the cooperation object *coopt_cust* defined above, and store this integer value into variable *num_master*.

```
define num_master of type integer
...
assign length of master queue coopt_cust to num_master
```

4 Cooperation with Several Slaves

A master process can cooperate in a joint activity with several slave objects. The master process creates a separate queue to place the slave processes. For example, a master process that needs to cooperate with N slave processes defines and creates a queue to place each reference to slave process.

The following code shows N slave processes that cooperate in a joint activity with the master process. The slave processes are placed in queue, *s_queue*, during the cooperation. This queue is owned by the master process. The duration of the cooperation interval is stored in the variable *coop_int*. At the end of the cooperation interval, the master process removes each slave process and reactivates it. The master process then performs its own activities. The passive object *waitq_obj* of class *Waitq*, is defined in the main class of the model.

```
constants
  define N = 25 of type integer
variables
  define numslaves of type integer    // number of slaves
```

```

object references
  define s_queue of class Pqueue    // priority queue
  define slave_ref of class Slave
  . . .
  set numslaves = 0
  create s_queue = of class Pqueue using "Slave queue"
  . . .
  while numslaves < N do
    cooperate with slave slave_ref of class Slave
      in waitq_obj
    insert slave_ref into s_queue    // enqueue
      into slave queue
    increment numslaves
  endwhile
  //
  // carry out joint activity
  hold self for coop_int           // cooperation interval
  // now release slave processes
  for j = 0 to N-1 do
    remove slave_ref of class Slave from s_queue
    reactivate slave_ref now
  endfor
  . . .

```

5 Synchronous Process Communication

A synchronous communication of two processes occurs when a sender process and a receiver process are engaged in a joint activity of *data transfer*. Because this is synchronous communication, both processes need to participate simultaneously during the interval of cooperation.

A sender process attempts to send a message and is suspended if the receiver process is not willing to receive the message at that time. In the same manner, a receiver process is suspended if it attempts to receive a message but the sender is not available to cooperate at the same time. A *channel* is the means of communication between the sender and the receiver processes.

Figure 4 shows a UML communication diagram that illustrates direct communication between a sender process and a receiver process. As mentioned previously, the synchronization occurs when executing a specific communication task or activity in the sender simultaneously with a communication activity in the receiver. The channel is used as a carrier of the message in transfer from the sender process to the receiver process.

After the cooperation interval, the receiver process (master) reactivates the

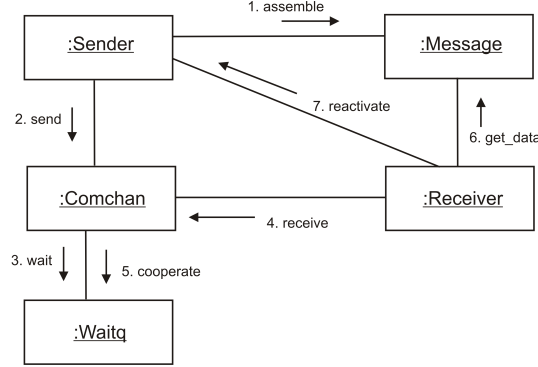


Figure 4: UML diagram with synchronous communication of two processes

sender process (slave), and from this time on, each process continues with its own independent activities.

6 Process Cooperation Model

The simulation model explained is an implementation of the objects shown in Figure 4 that shows the interaction among five objects, of which only two are active objects, the sender and the receiver processes. The cooperation object of class *Waitq* implements the mechanism that controls the interaction of the sender and receiver processes. This cooperation object is used by the channel, implemented as an object of class *Comchan*. The data sent by the sender process and received by the receiver process is assembled into an object of class *Message*.

The simulation model consists of five pairs of sender and receiver processes and is implemented with five classes written with OOSimL. There is a channel for each pair of sender and receiver. The source files are: `Scomm.osl`, `Sender.osl`, `Receiver.osl`, `Comchan.osl`, and `Message.osl`. These source files are archived in the file `scomm.jar`, which also includes the output listings of a sample simulation run.

Listing 1 shows the listing of the results summary of a simulation run with this model. The output listing includes performance metrics such as: the average wait times for the sender and receiver processes and the total number of messages successfully sent and received, and the channel utilization.

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

OOSimL model: Synchronous communication with channels

Simulation date: date: 11/9/2018 time: 11:02

-----STATISTICS REPORT-----

Wait queue: synChannel4

Number of observations master q: 59
 Number of obs master queue zero wait: 44
 Max number of objects in master queue: 1
 Average number of objects in master queue: 0
 Average time in master queue: 16.14204921623553
 Master queue utilization: 0.09458231962638006
 Master queue usage: 0.09458231962638006
 Number of observations slave q: 59
 Number of obs slave queue zero wait: 15
 Max number of objects in slave queue: 1
 Average number of objects in slave queue: 0
 Average time in slave queue: 24.8603104133925
 Slave queue utilization: 0.5729524665586553
 Slave queue usage: 0.5729524665586553

Wait queue: synChannel3

Number of observations master q: 44
 Number of obs master queue zero wait: 33
 Max number of objects in master queue: 1
 Average number of objects in master queue: 0
 Average time in master queue: 14.959279939894998
 Master queue utilization: 0.06427815599173632
 Master queue usage: 0.06427815599173632
 Number of observations slave q: 44
 Number of obs slave queue zero wait: 11
 Max number of objects in slave queue: 1
 Average number of objects in slave queue: 0
 Average time in slave queue: 39.1784255397218
 Slave queue utilization: 0.6733791889639684
 Slave queue usage: 0.6733791889639684

Wait queue: synChannel2

Number of observations master q: 50
 Number of obs master queue zero wait: 35
 Max number of objects in master queue: 1
 Average number of objects in master queue: 0
 Average time in master queue: 13.991424961906587
 Master queue utilization: 0.08198100563617142
 Master queue usage: 0.08198100563617142
 Number of observations slave q: 50
 Number of obs slave queue zero wait: 15

Max number of objects in slave queue: 1
 Average number of objects in slave queue: 0
 Average time in slave queue: 31.532534115764385
 Slave queue utilization: 0.6158698069485231
 Slave queue usage: 0.6158698069485231

 Wait queue: synChannel1
 Number of observations master q: 60
 Number of obs master queue zero wait: 36
 Max number of objects in master queue: 1
 Average number of objects in master queue: 0
 Average time in master queue: 13.581375576776347
 Master queue utilization: 0.12732539603227827
 Master queue usage: 0.12732539603227827
 Number of observations slave q: 60
 Number of obs slave queue zero wait: 24
 Max number of objects in slave queue: 1
 Average number of objects in slave queue: 0
 Average time in slave queue: 25.551580670991093
 Slave queue utilization: 0.5988651719763538
 Slave queue usage: 0.5988651719763538

 Wait queue: synChannel0
 Number of observations master q: 46
 Number of obs master queue zero wait: 35
 Max number of objects in master queue: 1
 Average number of objects in master queue: 0
 Average time in master queue: 12.631481116210853
 Master queue utilization: 0.05427589542121851
 Master queue usage: 0.05427589542121851
 Number of observations slave q: 46
 Number of obs slave queue zero wait: 11
 Max number of objects in slave queue: 1
 Average number of objects in slave queue: 0
 Average time in slave queue: 39.66874996934699
 Slave queue utilization: 0.7127978510117037
 Slave queue usage: 0.7127978510117037

 Random generator: Consume interval4
 Distribution: Negative exponential
 Number of obs: 59
 Seed: 61
 Mean: 42.75

 Random generator: Produce data interval4

Distribution: Negative exponential
Number of obs: 60
Seed: 5
Mean: 13.25

Random generator: Consume interval3
Distribution: Negative exponential
Number of obs: 44
Seed: 49
Mean: 42.75

Random generator: Produce data interval3
Distribution: Negative exponential
Number of obs: 45
Seed: 4
Mean: 13.25

Random generator: Consume interval2
Distribution: Negative exponential
Number of obs: 50
Seed: 37
Mean: 42.75

Random generator: Produce data interval2
Distribution: Negative exponential
Number of obs: 51
Seed: 3
Mean: 13.25

Random generator: Consume interval1
Distribution: Negative exponential
Number of obs: 60
Seed: 25
Mean: 42.75

Random generator: Produce data interval1
Distribution: Negative exponential
Number of obs: 61
Seed: 2
Mean: 13.25

Random generator: Consume interval0
Distribution: Negative exponential
Number of obs: 46
Seed: 13

Mean: 42.75

Random generator: Produce data interval0
Distribution: Negative exponential
Number of obs: 47
Seed: 1
Mean: 13.25

Random generator: Comm int Channel4
Distribution: Normal
Number of obs: 59
Seed: 55
Mean: 3.45
Standard deviation: 0.75

Random generator: Comm int Channel3
Distribution: Normal
Number of obs: 44
Seed: 43
Mean: 3.45
Standard deviation: 0.75

Random generator: Comm int Channel2
Distribution: Normal
Number of obs: 50
Seed: 31
Mean: 3.45
Standard deviation: 0.75

Random generator: Comm int Channel1
Distribution: Normal
Number of obs: 60
Seed: 19
Mean: 3.45
Standard deviation: 0.75

Random generator: Comm int Channel0
Distribution: Normal
Number of obs: 46
Seed: 7
Mean: 3.45
Standard deviation: 0.75

End Simulation of Synchronous communication

```

Sender0 wait interval: 1824.7625
Receiver0 wait interval: 0138.9463
Sender1 wait interval: 1533.0948
Receiver1 wait interval: 0325.9530
Sender2 wait interval: 1576.6267
Receiver2 wait interval: 0209.8714
Sender3 wait interval: 1723.8507
Receiver3 wait interval: 0164.5521
Sender4 wait interval: 1466.7583
Receiver4 wait interval: 0242.1307
Average sender wait: 1625.0186
Average receiver wait: 0216.2907

```

Listing 2 shows a partial listing of the trace of a simulation run with this model.

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

```

OOSimL model: Synchronous communication with channels
Simulation date: 11/9/2018 time: 11:02
----- TRACE -----
Time: 0 Input Parameters:
Time: 0 Simulation period: 2560.0
Time: 0 Receiver mean consume interval: 42.75
Time: 0 Sender mean produce interval: 13.25
Time: 0 Number of rec/sender processes: 5
Time: 0 Synchronous Communication holds for 2560
Time: 0 Message No. 0 Computer Science sender
    Sender0 rec Receiver0
Time: 0 Sender0 holds for 17.392
Time: 0 Receiver0 attempting comm via Channel0
Time: 0 Receiver0 coopt with synChannel0
Time: 0 Message No. 1 Information Systems sender
    Sender1 rec Receiver1
Time: 0 Sender1 holds for 17.405
Time: 0 Receiver1 attempting comm via Channel1
Time: 0 Receiver1 coopt with synChannel1
Time: 0 Message No. 2 Kennesaw State University sender
    Sender2 rec Receiver2
Time: 0 Sender2 holds for 17.401
Time: 0 Receiver2 attempting comm via Channel2
Time: 0 Receiver2 coopt with synChannel2
Time: 0 Message No. 3 1000 Chastain Road sender
    Sender3 rec Receiver3

```

```

Time: 0 Sender3 holds for 17.379
Time: 0 Receiver3 attempting comm via Channel3
Time: 0 Receiver3 coopt with synChannel3
Time: 0 Message No. 4 Kennesaw, GA sender
    Sender4 rec Receiver4
Time: 0 Sender4 holds for 17.374
Time: 0 Receiver4 attempting comm via Channel4
Time: 0 Receiver4 coopt with synChannel4
Time: 17.374 Sender4 attempting to communicate via
    chan Channel4
Time: 17.374 Sender4 wait on synChannel4
Time: 17.374 Receiver4 holds for 4.745
Time: 17.379 Sender3 attempting to communicate via
    chan Channel3
Time: 17.379 Sender3 wait on synChannel3
Time: 17.379 Receiver3 holds for 3.79
Time: 17.392 Sender0 attempting to communicate via
    chan Channel0
Time: 17.392 Sender0 wait on synChannel0
Time: 17.392 Receiver0 holds for 4.084
Time: 17.401 Sender2 attempting to communicate via
    chan Channel2
Time: 17.401 Sender2 wait on synChannel2
Time: 17.401 Receiver2 holds for 4.567
Time: 17.405 Sender1 attempting to communicate via
    chan Channel1
Time: 17.405 Sender1 wait on synChannel1
Time: 17.405 Receiver1 holds for 4.449
Time: 21.169 Sender3 to reactivate
Time: 21.169 Receiver3 received message 3 via
    channel Channel3
Time: 21.169 Receiver3 received 1000 Chastain Road from
    sender: Sender3
Time: 21.169 Receiver3 holds for 55.886
Time: 21.169 Sender3 sent message # 3
Time: 21.169 Message No. 5 1000 Chastain Road sender
    Sender3 rec Receiver3
. . .
Time: 2539.37 Receiver0 attempting comm via Channel0
Time: 2539.37 Receiver0 coopt with synChannel0
Time: 2542.24 Sender1 to reactivate
Time: 2542.24 Receiver1 received message 257 via
    channel Channel1
Time: 2542.24 Receiver1 received Information Systems from
    sender: Sender1

```

```

Time: 2542.24 Receiver1 holds for 109.153
Time: 2542.24 Sender1 sent message # 257
Time: 2542.24 Message No. 262 Information Systems sender
      Sender1 rec Receiver1
Time: 2542.24 Sender1 holds for 0.385
Time: 2542.625 Sender1 attempting to communicate via
      chan Channel1
Time: 2542.625 Sender1 wait on synChannel1
Time: 2549.441 Sender0 attempting to communicate via
      chan Channel0
Time: 2549.441 Sender0 wait on synChannel0
Time: 2549.441 Receiver0 holds for 3.201
Time: 2552.643 Sender0 to reactivate
Time: 2552.643 Receiver0 received message 260 via
      channel Channel0
Time: 2552.643 Receiver0 received Computer Science from
      sender: Sender0
Time: 2552.643 Receiver0 holds for 13.07
Time: 2552.643 Sender0 sent message # 260
Time: 2552.643 Message No. 263 Computer Science sender
      Sender0 rec Receiver0
Time: 2552.643 Sender0 holds for 1.691
Time: 2554.334 Sender0 attempting to communicate via
      chan Channel0
Time: 2554.334 Sender0 wait on synChannel0
Time: 2554.677 Sender3 attempting to communicate via
      chan Channel3
Time: 2554.677 Sender3 wait on synChannel3
Time: 2560 Synchronous Communication holds
      for 115.127
Time: 2565.712 Receiver0 deactivated
Time: 2604.223 Receiver3 deactivated
Time: 2651.393 Receiver1 deactivated
Time: 2654.566 Receiver4 deactivated
Time: 2675.127 Receiver2 deactivated
-----
End Simulation of Synchronous communication with channels
date: 10/7/2008 time: 14:36

Time: 2675.127 Sender0 terminating
Time: 2675.127 Receiver0 terminating
Time: 2675.127 Sender1 terminating
Time: 2675.127 Receiver1 terminating
Time: 2675.127 Sender2 terminating

```

Time: 2675.127 Receiver2 terminating
 Time: 2675.127 Sender3 terminating
 Time: 2675.127 Receiver3 terminating
 Time: 2675.127 Sender4 terminating
 Time: 2675.127 Receiver4 terminating

Listing 3 shows the implementation of class *Sender* of the simulation model.

Listing 3 Source code of class *Sender*.

```

use all java.io
use all psimjava
description
    Synchronous communication (direct comm)
    Revised, September 2018
    J. Garrido
    Process synchronous communication with OOSimL
    Sender process directly communicates with a receiver process
    to transfer a message through a channel.
    This model uses an array of sender and array of receiver
    processes. A waitq object for the rendezvous represents a
    channel for each pair of processes.

File: Sender.osl
*/
class Sender as process is
    private
    variables
        define mean_prod of type double    // interval produce mess
        define idnum of type integer        // sender id number
        define wait_int of type double      // sender wait interval
        define sender_wait of type double   // total sender wait
        define startw of type double        // time of start waiting
        define simclock of type double      // current sim time
        define sname of type string         // name of sender process
    object references
        define smsg of class Message        // message to send
        define produc of class NegExp       // period to produce mess
        define rec_proc of class Receiver   // receiver object
        define lchan of class Comchan       // ref to channel object

    public
    function initializer parameters sname of type string,
        sendnum of type integer, mean_prod of type double

```



```

is
begin
    call super using sname
    set idnum = sendnum           // sender id number
    //
    // random number generator for
    // interval to produce data
    create produc of class NegExp using
        "Produce data interval"
        concat idnum, mean_prod, idnum+1
    set sender_wait = 0.0
    //

    display sname, " created"
endfun initializer
//
function get_wait return type double is
begin
    return sender_wait
endfun get_wait
//
// assemble a message with contents
function assemble parameters contents of type string,
    rec of class Receiver
is
begin
    create smsg of class Message using contents
    call smsg.set_sender using self    // sender
    call smsg.set_rec using rec        // receiver
    //
    display "Message No. ", smsg.get_messnum(), " ",
        smsg.get_data(), " sender ", get_name(), " rec ",
        rec.get_name()
    tracewrite "Message No. ", smsg.get_messnum(), " ",
        smsg.get_data(), " sender ", get_name(), " rec ",
        rec.get_name()
endfun assemble
//
//
function Main_body is
variables
    define produce_per of type double // int produce data

object references
    define lchan of class Comchan    // ref to channel

```

```

begin
  set sname = call get_name
  set lchan = Scomm.channel[idnum]
  assign simulation clock to simclock
  while simclock < Scomm.simperiod do
    // display sname, " to generate msg at ", simclock
    //
    //
    // complete setting up the channel
    set rec_proc = Scomm.rec_obj[idnum] // receiver
    // call lchan.set_sender using self
    // call lchan.set_rec using rec_proc
    //
    // assemble data for receiver process rec_proc
    call assemble using Scomm.data[idnum], rec_proc
    //
    // generate random interval to produce item
    assign random value from produc to produce_per
    //
    hold self for produce_per // interval produce msg
    //
    assign simulation clock to simclock
    set startw = simclock // start to wait
    //
    display sname, " attempting to communicate via chan ",
      lchan.get_chname(), " at ", simclock
    tracewrite sname, " attempting to comm via chan ",
      lchan.get_chname()
    //
    call send using lchan, msg // wait to send message
    //
    assign simulation clock to simclock
    display sname, " sent message # ", msg.get_messnum(),
      " at ", simclock
    tracewrite sname, " sent message # ",
      msg.get_messnum()
    //
    set sender_wait = sender_wait + wait_int
  endwhile
  // display sname, " terminating at ", simclock
  suspend self // terminate in main class
endfun Main_body
//
function send parameters mchan of class Comchan,
  msg of class Message is

```

```
variables
    // communication interval
    define comm_int of type double
begin
    //
    // display sname, " to comm via ", synchan.get_name()
    // display sname, " sending msg with data: ",
    //         lmsg.get_data()
    //
    set startw = simclock
    //
    // send message through the channel
    // communicate with receiver
    call mchan.send using lmsg
    //
    // get communication interval for channel
    set comm_int = call mchan.get_commint
    // display sname, " comm interval: ", comm_int,
    //         " chan: ", mchan.get_chname()
    //
    //
    // display sname, " communicated chan: ",
    //         synchan.get_name()
    //
    assign simulation clock to simclock
    // wait interval
    set wait_int = (simclock - startw) - comm_int
    //
endfun send
endclass Sender
```
