
SYSC 5807:Methodological Aspects of Simulation and Modeling

Assignment No.1

By:

Umar Farooq

ufarooq@sce.carleton.ca

Student No. 258695

Systems and Computer Engineering

Carleton University

October 20th, 2003.

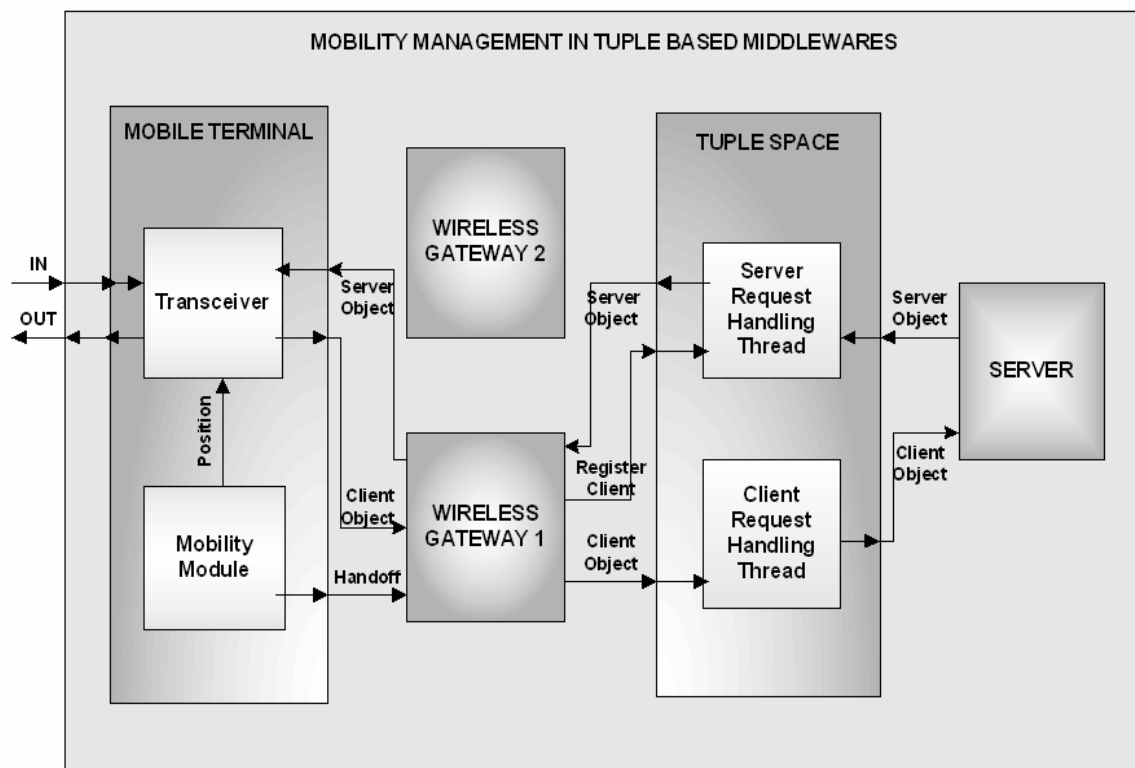
System to be analyzed with DEVS: MOBILITY MANAGEMENT IN TUPLE BASED MIDDLEWARE SYSTEMS

PART 1

1. Brief Description

The system chosen to be analyzed with DEVS involves managing user mobility in mobile wireless tuple based middleware systems. It consists of a mobile user that accesses the tuple space through different access points (wireless gateways) at different points in time due to its mobility. The application running at the user's mobile device writes a client object in the tuple space (through the wireless gateway it is currently connected to) to be serviced by the server. The space notifies the server about the object. The server after servicing the object puts it back to the space. The wireless gateways register the clients they are currently serving with the space. Hence, whenever a new object for the client is written into the space, the space locates in its list the gateway which is currently serving this client and notifies it. That gateway after acquiring the object from the space delivers it to the mobile client.

2. Sketch of the Model Structure



The above figure shows the sketch of the model structure. It consists of 3 levels and involves 10 unique atomic models. Note that all the inputs and output ports present in *wireless gateway 1* are also present in *wireless gateway 2*. However, they are not shown in the figure for clarity purposes. The user communicates to one of the gateways at one time depending on its current location. The details of each of the component are given in the next section.

3. Description of the Components

1. Mobile Terminal

This component represents the user's mobile device. The application running on the device provides inputs to this component in the form of client objects to be serviced and the component outputs the serviced objects to the application. It consists of transceiver and mobility module.

1a) Transceiver

The transceiver is the component that sends the client objects from the application to the wireless gateway to which the user is currently connected to. Similarly, it transfers the server objects from the gateway to the application. It maintains the reference to the gateway to which the user is currently connected as one of its state variables (possible values: gateway_1, gateway_2).

1b) Mobility Module

This component simulates user mobility and keeps track of the user's location and hence the gateway to which the user is currently connected to. It maintains as one of its state variables the reference to the current gateway the user is connected to. After a certain period of time (another state variable), it goes an internal transition and switches the user from one gateway to the other. At this moment it informs the transceiver of the change of gateway and also sends a handoff request to the new gateway.

2. Wireless Gateway 1 (and 2)

The wireless gateway transfers the client objects from the mobile terminal to the tuple space and the server objects from the space to the mobile terminal. If it receives a handoff request from the client, it registers itself with the space as the active gateway for the objects for this client.

3. Tuple Space

The tuple space represents the shared repository of objects. The mobile client and server communicate with each other by writing and reading objects from the space. The space is multithreaded and consists of client request handling thread and server request handling thread.

3a) Client Request Handling Thread

The client request handling thread services requests from the client (wireless gateway). It *delivers* the client objects from the wireless gateway (1 or 2) to the server.

3b) Server Request Handling Thread

This component *delivers* the server objects from the server to the wireless gateway to which the user is currently connected to. It maintains as one of its state variables a reference to the gateway to which the user is currently connected to. Whenever the mobile terminal switches from one gateway to the other, the new gateway sends a registration request to this thread to deliver all the incoming server objects for this client to it. At this moment, this thread updates its state variable containing the reference to the active gateway for the user.

4) Server

The server acquires client objects from the space to service and after servicing them writes them back to the space. It has several state variable associated with it: object_id (the id of the object it is currently servicing), object_queue (the queue of the objects it still needs to service), phase (busy or idle) and sigma (the time function).

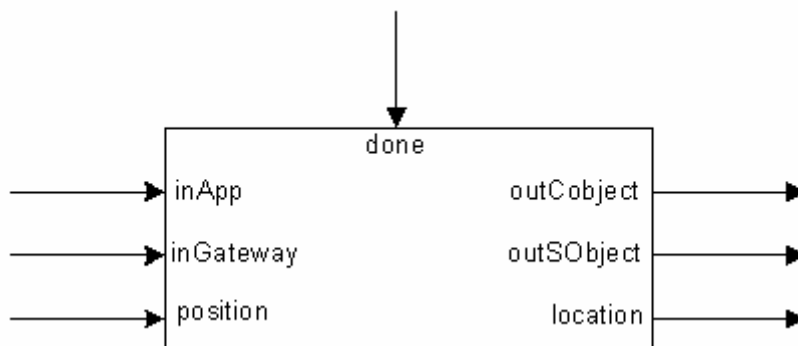
PART 2

1. Organize the model as atomic/coupled models; define the structure and coupling scheme.

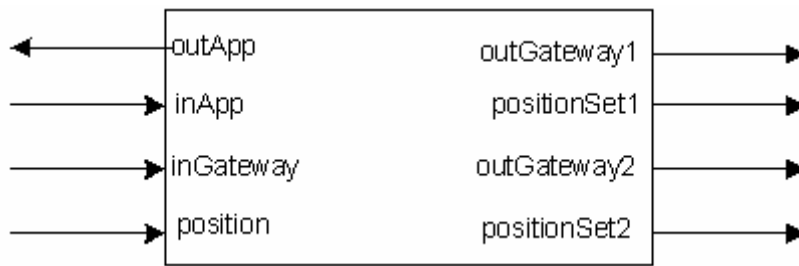
1) Mobile Terminal

During the redefinition of the model it was found necessary to have an atomic model that can queue different events coming for the transceiver. Hence a new model called TransceiverQueue was defined. The Mobile Terminal component thus consists of 3 atomic models given as follows:

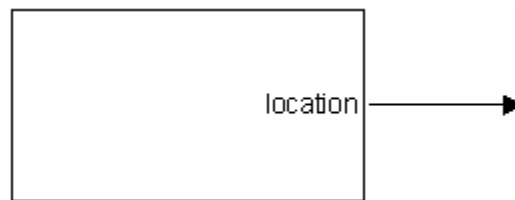
1a) Atomic Model: TransceiverQueue



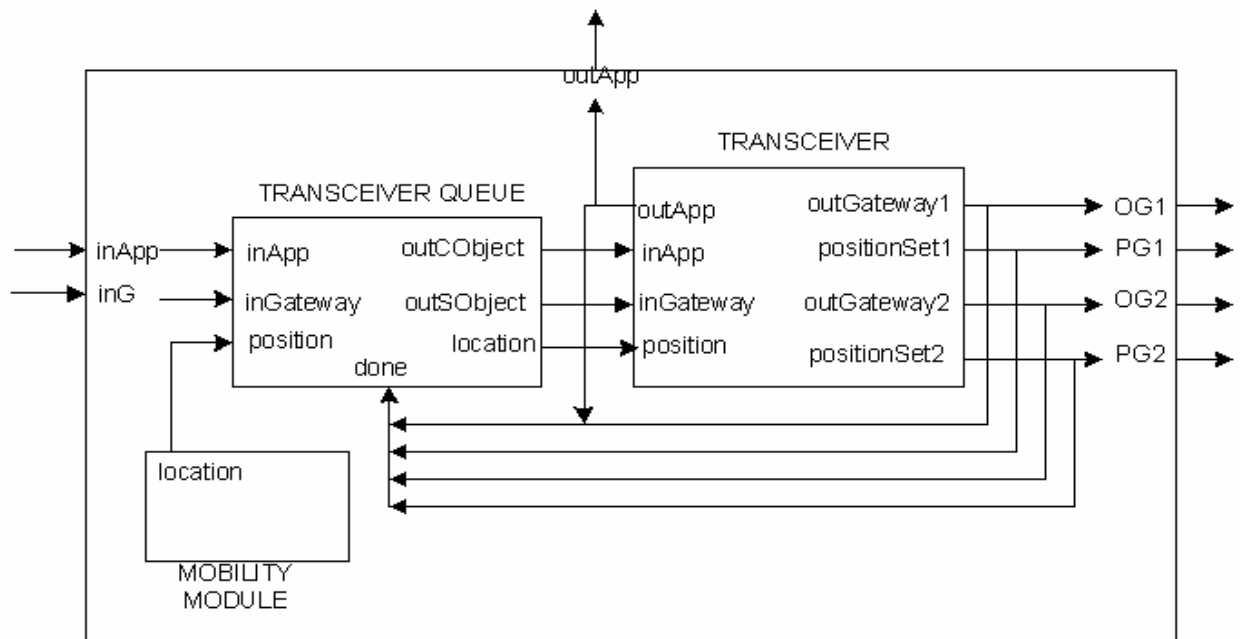
1b) Atomic Model: Transceiver



1c) Atomic Model: MobilityModule



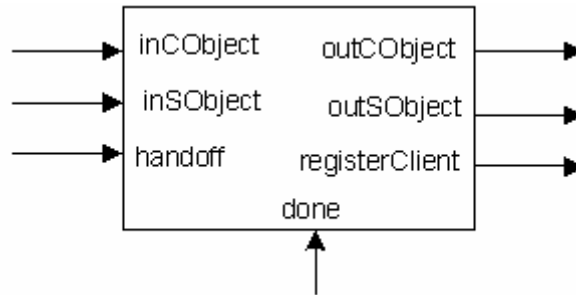
1d) Coupled Model: Mobile Terminal



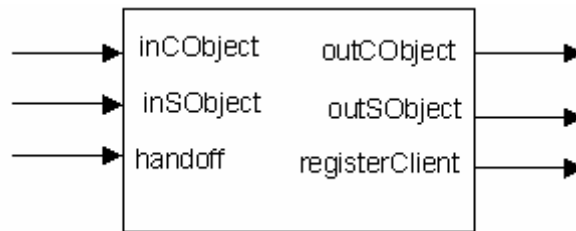
2) Wireless Gateway

During the redefinition of the model it was found necessary to have an atomic model that can queue different events coming for the gateway. Hence a new model called GatewayQueue was defined. The Gateway component thus consists of 2 atomic models given as follows:

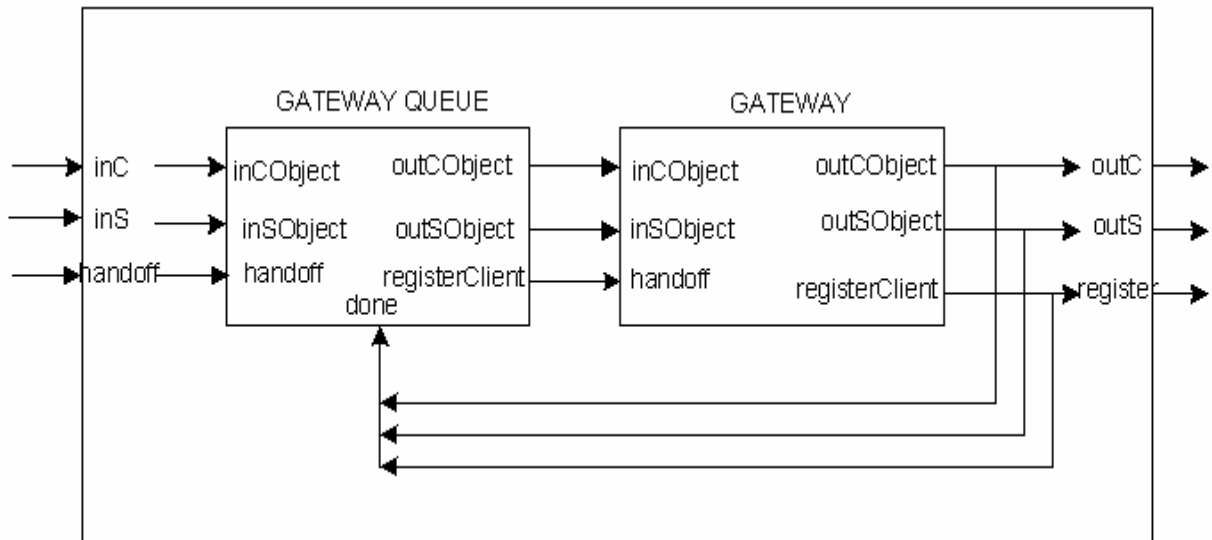
2a) Atomic Model: GatewayQueue



2b) Atomic Model: Gateway



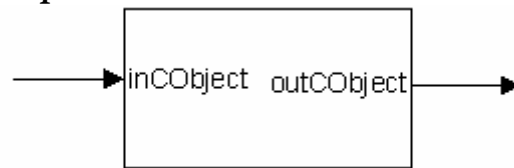
2c) Coupled Model: Wireless Gateway



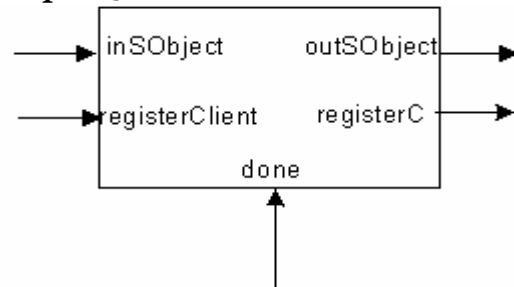
3) Tuple Space

During the redefinition of the model, it was found necessary to have queues for each of the client request handling thread and server request handling thread. The client request handling thread uses the queue model explained in the class and hence not given here. The server request handling thread uses a queue called tuplequeue to queue the different requests. The atomic models for tuple space are given as follows:

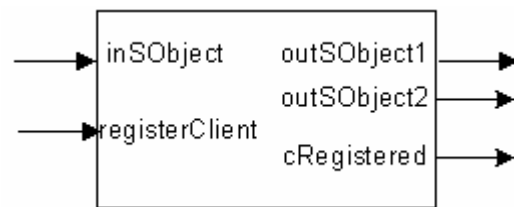
3a) Atomic Model: TupleClient



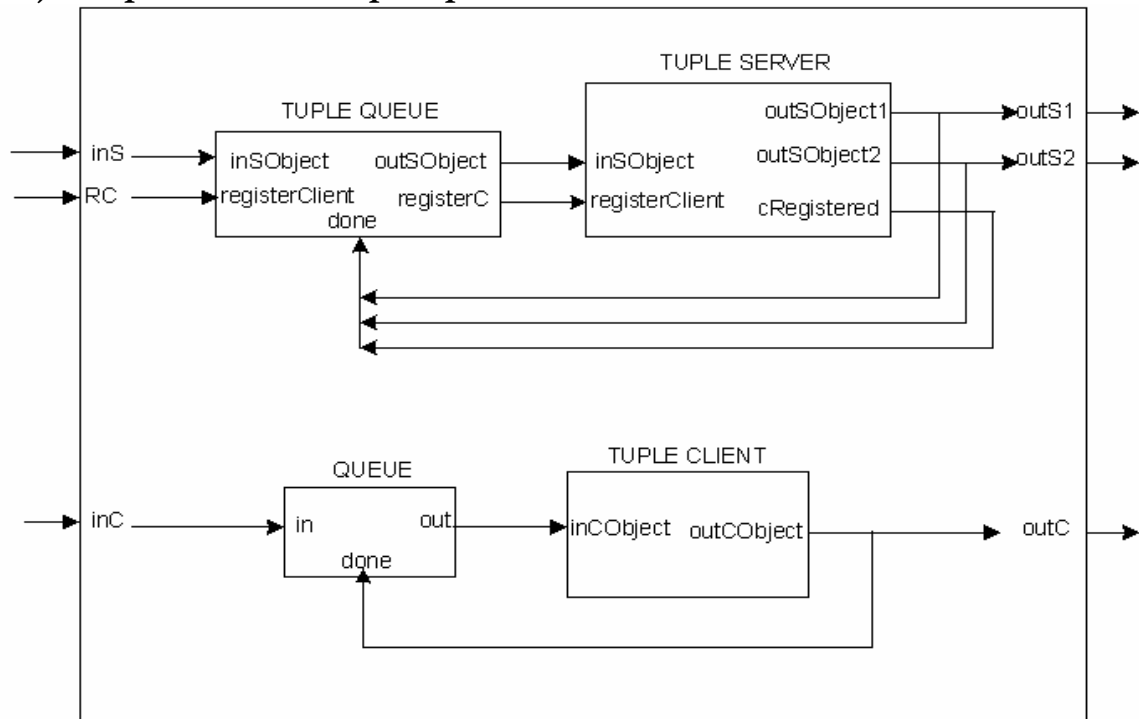
3b) Atomic Model: TupleQueue



3c) Atomic Model: TupleServer



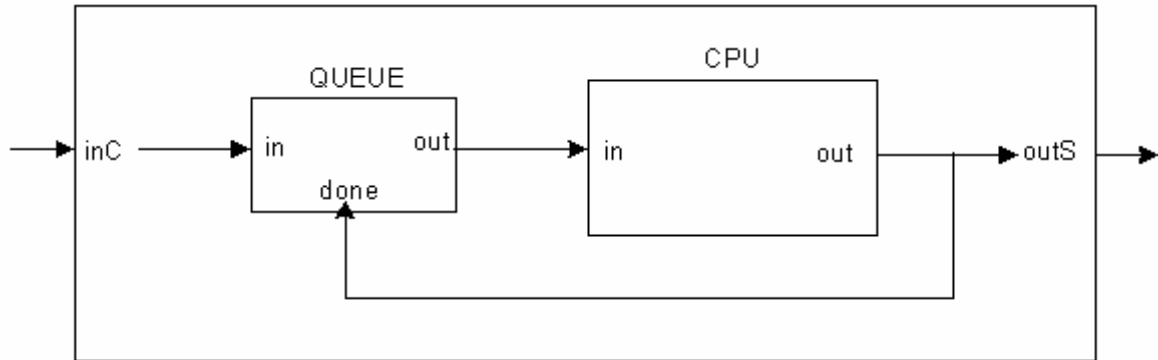
3d) Coupled Model: Tuple Space



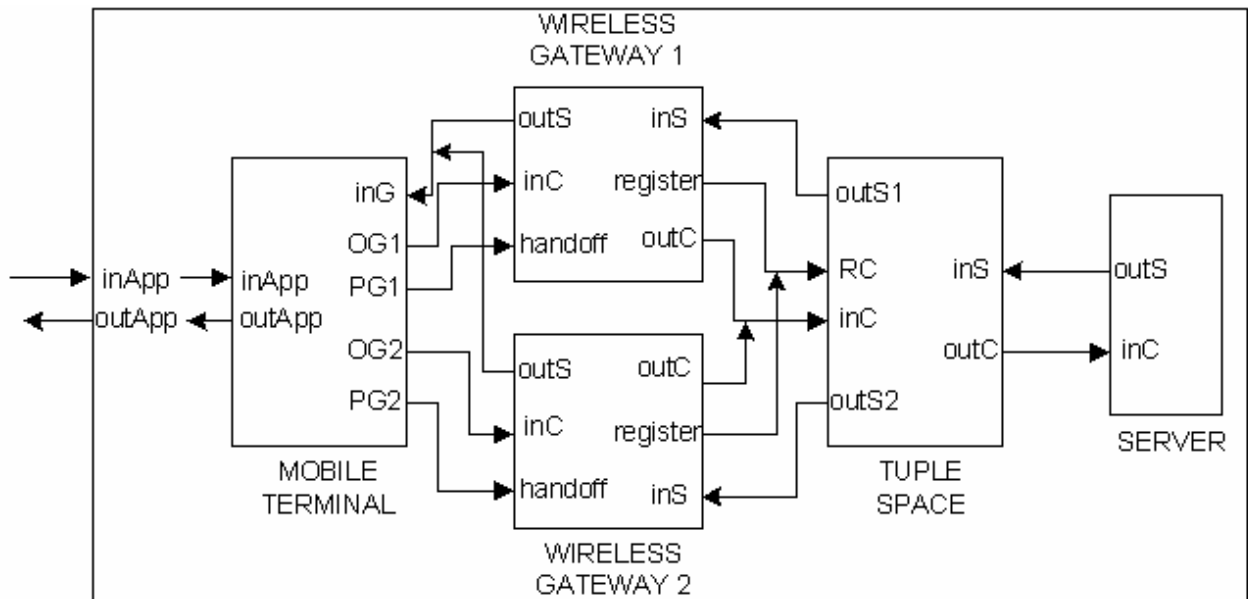
4) Sever

The Sever consists of a Queue and CPU model explained in the class and hence are not presented here. The coupled server model is given as follows:

4a) Coupled Model: Server



5) Coupled Model: Mobility Management (Top)



2. Write a formal specification for each of the coupled models.

1) Mobile Terminal

Let MOB, TRANS and QUEUE be the instance of Mobility Module, Transceiver and Transceiver Queue respectively.

MOBILETERMINAL = <X,Y,{MOB,QUEUE,TRANS}, EIC, EOC, IC, SELECT>

X = {inApp, inG}

Y = {outApp, outG1, outG2, pG1, pG2}

EIC = {(MOBILETERMINAL.inApp, QUEUE.inApp),
(MOBILETERMINAL.inG, QUEUE.inGateway)}

EOC = {(TRANS.outApp, MOBILETERMINAL.outApp),
(TRANS.outGateway1, MOBILETERMINAL.outG1), (TRANS.outGateway2,
MOBILETERMINAL.outG2), (TRANS.positionSet1, MOBILETERMINAL.pG1),
(TRANS.positionSet1, MOBILETERMINAL.pG1)}

IC = { (QUEUE.outCObject, TRANS.inApp), (QUEUE.outSObject,
TRANS.inGateway), (QUEUE.location, TRANS.position), (MOB.location,
QUEUE.position), (TRANS.outApp, QUEUE.done), (TRANS.outGateway1,
QUEUE.done), (TRANS.outGateway2, QUEUE.done), (TRANS.positionSet1,
QUEUE.done), (TRANS.positionSet2, QUEUE.done) }

SELECT = ({MOB, QUEUE, TRANS}) = MOB
({QUEUE, TRANS}) = QUEUE

2) Wireless Gateway

Let GW and GWQ be the instance of Gateway and Gateway Queue respectively.

WIRELESSGATEWAY = <X,Y,{GW,GWQ}, EIC, EOC, IC, SELECT>

X = {inS, inC, handoff}

Y = {outS, outC, register}

EIC = {(WIRELESSGATEWAY.inC, GWQ.inCObject),
(WIRELESSGATEWAY.inS, GWQ.inSObject), {(WIRELESSGATEWAY.handoff,
GWQ.handoff)}

EOC = { (GW.outCObject, WIRELESSGATEWAY.outC), (GW.outSObject,
WIRELESSGATEWAY.outS), (GW.registerClient, WIRELESSGATEWAY.register)}

IC = {(GWQ.outCObject, GW.inCObject), (GWQ.outSObject, GW.inSObject),
(GWQ.registerClient, GW.handoff), (GW.outCObject, GWQ.done), (GW.outSObject,
GWQ.done), (GW.registerClient, GWQ.done)}

SELECT = ({GWQ, GW}) = GWQ

3) Tuple Space

Let TQUEUE, TCLIENT, TSERVER and TSQUEUE be the instance of Queue, Tuple Client, Tuple Server and Tuple Queue respectively.

TUPLESPACE = <X,Y,{TQUEUE, TCLIENT, TSERVER, TSQUEUE}, EIC, EOC, IC, SELECT>

X = {inS, inC, RC}

Y = {outS1, outS2, outC}

EIC = {(TUPLESPACE.inC, TQUEUE.in), (TUPLESPACE.RC, TSQUEUE.registerClient), (TUPLESPACE.inS, TSQUEUE.inSObject)}

EOC = {(TCLIENT.outCObject, TUPLESPACE.outC), (TSERVER.outSObject1, TUPLESPACE.outS1), (TSERVER.outSObject2, TUPLESPACE.outS2)}

IC = {(TQUEUE.out, TCLIENT.inCObject), (TCLIENT.outCObject, TQUEUE.done), (TSQUEUE.registerC, TSERVER.registerClient), (TSQUEUE.outSObject, TSERVER.inSObject), (TSERVER.outSObject1, TSQUEUE.done), (TSERVER.outSObject2, TSQUEUE.done), (TSERVER.cRegistered, TSQUEUE.done)}

SELECT = ({TQUEUE, TCLIENT}) = TQUEUE
({TSQUEUE, TSERVER}) = TSQUEUE

4) Server

Let SERV and SERVERQUEUE be the instance of CPU and Queue respectively.

SERVER = <X,Y,{SERV, SERVERQUEUE}, EIC, EOC, IC, SELECT>

X = {inC}

Y = {outS}

EIC = {(SERVER.inC, SERVERQUEUE.in)}

EOC = {(SERV.out, SERVER.outS)}

IC = {(SERVERQUEUE.out, SERV.in), (SERV.out, SERVERQUEUE.done)}

SELECT = ({SERVERQUEUE, SERV}) = SERVERQUEUE

5) Top (Mobility Management)

Let MOBTERM, TS and SERVER be the instance of Mobile Terminal, Tuple Space and Server respectively while WG1 and WG2 are the instances of Wireless Gateway.

TOP = <X, Y, {MOBTERM, TS, SERVER, WG1, WG2}, EIC, EOC, IC, SELECT>

X = {inApp}

Y = {outApp}

EIC = { (TOP.inApp, MOBTERM.inApp) }

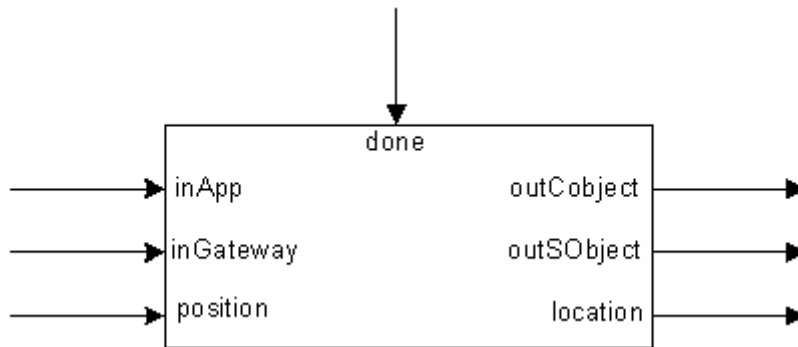
EOC = { (MOBTERM.outApp, TOP.outApp) }

IC = { (MOBTERM.outG1, WG1.inC), (MOBTERM.outG2, WG2.inC), (MOBTERM.pG1, WG1.handoff), (MOBTERM.pG2, WG2.handoff), (WG1.outS, MOBTERM.inG), (WG2.outS, MOBTERM.inG), (WG1.outC, TS.inC), (WG2.outC, TS.inC), (WG1.register, TS.RC), (WG2.register, TS.RC), (TS.outS1, WG1.inS), (TS.outS2, WG2.inS), (TS.outC, SERVER.inC), (SERVER.outS, TS.inS) }

SELECT	=	{MOBTERM, TS, SERVER, WG}}	=	WG (1 or 2 depending on which one is active)
		{TS, SERVER, MOBTERM}	=	MOBTERM
		{TS, SERVER}	=	TS

3. Write a formal specification for each of the atomic models.

1) Atomic Model: TransceiverQueue



TRANSCIVERQUEUE = <S, X, Y, δ_{int} , δ_{ext} , λ , ta>

X = {inApp, inGateway, position, done}

Y = {outSObject, outSObject, location}

$S = \{\text{phase, job-queue}\}$
 where
 $\text{phase} = \{\text{idle, busy}\}$
 $\text{job-queue} =$ List of jobs in the queue where each job element consists of two fields object-id and port-id (the id of the port from which job entered the queue).

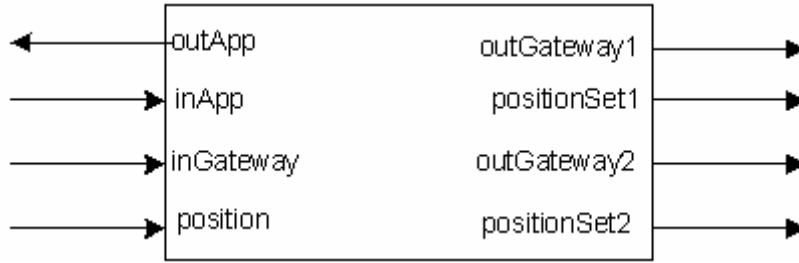
$\text{ta(Busy)} =$ preparation time
 $\text{ta(Idle)} =$ infinite

$\delta_{\text{int}}(s)\{$
 case phase
 busy:
 if (job-queue empty)
 passivate
 else
 get next job from the queue in preparation time
 passive:
 //never happens
 }

$\delta_{\text{ext}}(s, e, x)\{$
 case port
 inApp:
 add the job to the queue with object-id equal to the id of the job and port-id equal to 1.
 inGateway:
 add the job to the queue with object-id equal to the id of the job and port-id equal to 2.
 position:
 add the job to the queue with object-id equal to the id of the job and port-id equal to 3.
 done:
 pop the first element in the queue.
 }

$\lambda(s)\{$
 case port-id of the front element in the job
 1:
 send the job to the port outCObject
 2:
 send the job to the port outSObject
 3:
 send the job to the port location
 }

2) Atomic Model: Transceiver



TRANSCIVER = $\langle S, X, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta \rangle$

$X = \{\text{inApp, inGateway, position}\}$

$Y = \{\text{outApp, outGateway1, outGateway2, positionSet1, positionSet2}\}$

$S = \{\text{phase, object-id, inport-id, gatewayID}\}$

where

phase = {idle, busy}

object-id = The id of the object currently being served by the transceiver

inport-id = The id of the input port from which the current job is received.

gatewayID = The id of the gateway to which the terminal is currently connected.

$ta(\text{ Busy }) = \text{send time}$

$ta(\text{ Idle }) = \text{infinite}$

```

 $\delta_{\text{int}}(s) \{$ 
  case phase
    busy:
      passivate after send time
    passive:
      //never happens
 $\}$ 

```

```

 $\delta_{\text{ext}}(s, e, x) \{$ 
  case port
    inApp:
      Set port-id equal to 1.
    inGateway:
      Set port-id equal to 2.
    position:
      Set port-id equal to 3 and gatewayID equal to the object-id of the job.
 $\}$ 

```

```

λ (s){
    case inport-id
        1:
            if (gatewayID equal to 1)
                send the job to the port outGateway1
            else
                send the job to the port outGateway2
        2:
            send the job to the port outApp
        3:
            if (gatewayID equal to 1)
                send the job to the port positionSet1
            else
                send the job to the port positionSet2
}

```

3) Atomic Model: MobilityModule



MOBILITYMODULE = <S, X, Y, δ_{int} , δ_{ext} , λ , ta>

X = \emptyset

Y = {location }

S = {gatewayID}

where

gatewayID = Id of the current gateway to which the mobile terminal is connected.

ta = handoff time

$\delta_{int}(s)$ {
Do nothing for time interval equal to handoff time
}

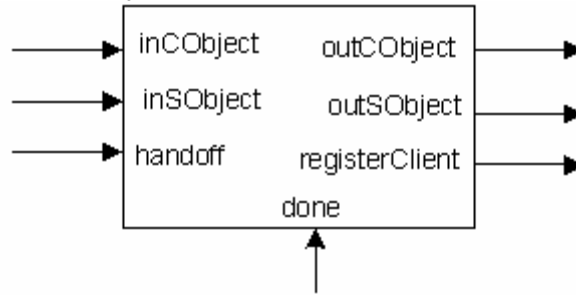
$\delta_{ext}(s, e, x)$ Unavailable

```

λ (s){
    send the current gatewayID to the port location.
    case gatewayID-id
        1:
            gatewayID equal to 2
        2:
            gatewayID equal to 1
    }

```

4) Atomic Model: GatewayQueue



GATEWAYQUEUE = $\langle S, X, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$

$X = \{inCObject, inSObject, handoff, done\}$

$Y = \{outCObject, outSObject, registerClient\}$

$S = \{phase, job\text{-}queue\}$

where

phase = {idle, busy}

job-queue = List of jobs in the queue where each job element consists of two fields object-id and port-id (the id of the port from which job entered the queue).

$ta(Busy) = \text{preparation time}$

$ta(Idle) = \text{infinite}$

```

δint(s){
    case phase
        busy:
            if (job-queue empty)
                passivate
            else
                get next job from the queue in preparation time
        passive:
            //never happens
    }

```

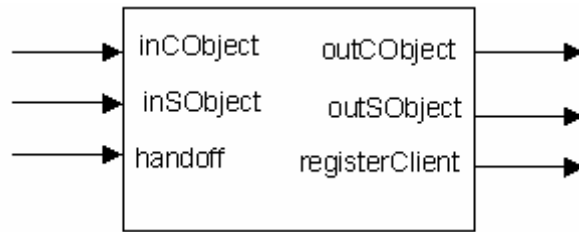
```

 $\delta_{\text{ext}}(s, e, x) \{$ 
    case port
        inCObject:
            add the job to the queue with object-id equal to the id of the job
            and port-id equal to 1.
        inSObject:
            add the job to the queue with object-id equal to the id of the job
            and port-id equal to 2.
        handoff:
            add the job to the queue with object-id equal to the id of the job
            and port-id equal to 3.
        done:
            pop the first element in the queue.
    }

 $\lambda(s) \{$ 
    case port-id of the front element in the job
        1:
            send the job to the port outCObject
        2:
            send the job to the port outSObject
        3:
            send the job to the port registerClient
    }

```

5) Atomic Model: Gateway



$\text{GATEWAY} = \langle S, X, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, \text{ta} \rangle$

$X = \{\text{inCObject}, \text{inSObject}, \text{handoff}\}$

$Y = \{\text{outCObject}, \text{outSObject}, \text{registerClient}\}$

$S = \{\text{phase}, \text{object-id}, \text{inport-id}\}$

where

phase = {idle, busy}

object-id = The id of the object currently being served by the gateway

inport-id = The id of the input port from which the current job is received.

ta(Busy) = send time
 ta(Idle) = infinite

```

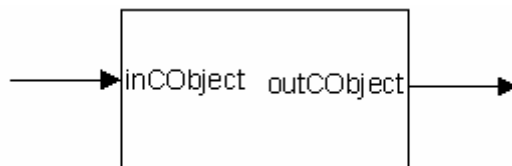
 $\delta_{\text{int}}(s)\{$ 
  case phase
    busy:
      passivate after send time
    passive:
      //never happens
 $\}$ 

 $\delta_{\text{ext}}(s, e, x)\{$ 
  case port
    inCObject:
      Set port-id equal to 1.
    inSObject:
      Set port-id equal to 2.
    handoff:
      Set port-id equal to 3.
 $\}$ 

 $\lambda(s)\{$ 
  case inport-id
    1:
      send the job to the port outCObject
    2:
      send the job to the port outSObject
    3:
      send the job to the port registerClient
 $\}$ 

```

6) Atomic Model: TupleClient



$\text{TUPLECLIENT} = \langle S, X, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, \text{ta} \rangle$

X = {inCObject}

Y = {outCObject}

S = {phase, object-id}

where

phase = {idle, busy}

object-id = The id of the object currently being served by the gateway

ta(Busy) = write time

ta(Idle) = infinite

```

 $\delta_{int}(s)\{$ 
    case phase
    busy:
        passivate after write time
    passive:
        //never happens
 $\}$ 

```

```

 $\delta_{ext}(s, e, x)\{$ 
    receive the object
 $\}$ 

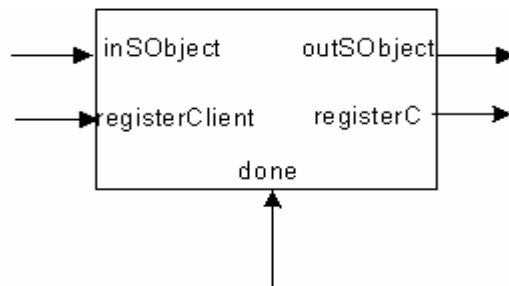
```

```

 $\lambda(s)\{$ 
    send the job to the port outCObject
 $\}$ 

```

8) Atomic Model: TupleQueue



TUPLEQUEUE = <S, X, Y, δ_{int} , δ_{ext} , λ , ta>

X = {inSObject, registerClient, done}

Y = {outSObject, registerC}

$S = \{\text{phase, job-queue}\}$
 where
 $\text{phase} = \{\text{idle, busy}\}$
 $\text{job-queue} =$ List of jobs in the queue where each job element consists of two fields object-id and port-id (the id of the port from which job entered the queue).

$\text{ta(Busy)} =$ preparation time
 $\text{ta(Idle)} =$ infinite

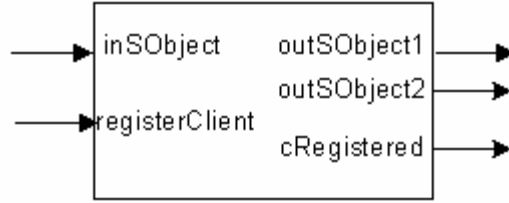
$\delta_{\text{int}}(s)\{$
 case phase
 busy:
 if (job-queue empty)
 passivate
 else
 get next job from the queue in preparation time
 passive:
 //never happens
 }

$\delta_{\text{ext}}(s, e, x)\{$
 case port
 inSObject:
 add the job to the queue with object-id equal to the id of the job and port-id equal to 1.
 registerClient:
 add the job to the queue with object-id equal to the id of the job and port-id equal to 2.
 done:
 pop the first element in the queue.
 }

$\lambda(s)\{$
 case port-id of the front element in the job
 1:
 send the job to the port outSObject
 2:
 send the job to the port registerC
 }

9) Atomic Model: TupleServer

$\text{TUPLESERVER} = \langle S, X, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, \text{ta} \rangle$



X = {inSObject, registerClient}

Y = {outSObject1, outSObject2, cRegistered}

S = {phase, object-id, inport-id, gatewayID}

where

phase = {idle, busy}

object-id = The id of the object currently being served by the tuple server request handling thread.

inport-id = The id of the input port from which the current job is received.

gatewayID = The id of the gateway to which the mobile terminal whose object is being served is currently connected to.

ta(Busy) = transfer time

ta(Idle) = infinite

```

 $\delta_{\text{int}}(s)\{$ 
  case phase
    busy:
      passivate after transfer time
    passive:
      //never happens
 $\}$ 
  
```

```

 $\delta_{\text{ext}}(s, e, x)\{$ 
  case port
    inSObject:
      Set port-id equal to 1.
    registerClient:
      Set port-id equal to 2. Set gatewayID equal to the object-id of the job.
  }
  
```

```

 $\lambda(s)\{$ 
  case inport-id
    1:
      if (gatewayID equal to 1)
        send the job to the port outSObject1
  }
  
```

```

else
    send the job to the port outSObject2
2:
    send the job to the port cRegistered
}

```

4. Propose a testing strategy for each one of the models.

1) Atomic Model: TransceiverQueue

This atomic model can be tested by giving inputs through different ports of the model and checking whether the model generates the output at the correct output port and in the expected amount of time. Since this queue has not been connected to the Transceiver yet, the 'done' event would have to be fed manually. The details of the event file and test output of the tests conducted on this model are given in Part 3 of this assignment (not given here to avoid repetition).

2) Atomic Model: Transceiver

This atomic model can be tested by giving inputs through different ports of the model and checking whether the model generates the output at the correct output port and in the expected amount of time. It is important to check that the model outputs the client object to the gateway the user is currently connected to and it correctly updates the gateway ID when it receives an event from the mobility module. The details of the event file and test output of the tests conducted on this model are given in Part 3 of this assignment (not given here to avoid repetition).

3) Atomic Model: MobilityModule

This atomic model can be tested by checking the output of the model. The model should generate alternate gateway ID at its location port after every handoff interval. The test outputs of this model are given in Part 3 of this assignment (not given here to avoid repetition).

4) Coupled Model: Mobile Terminal

This coupled model can be tested by giving inputs through different ports of the model and checking whether the model generates the output at the correct output port and in the expected amount of time. It is important to check that the model outputs the client object and the 'position set request' to the gateway the user is currently connected to. The details of the event file and test output of the tests conducted on this model are given in Part 3 of this assignment (not given here to avoid repetition).

5) Atomic Model: GatewayQueue

This atomic model can be tested by giving inputs through different ports of the model and checking whether the model generates the output at the correct output port and in the expected amount of time. Since this queue has not been connected to the Gateway yet, the 'done' event would have to be fed manually. The details of the event file and test

output of the tests conducted on this model are given in Part 3 of this assignment (not given here to avoid repetition).

6) Atomic Model: Gateway

This atomic model can be tested by giving inputs through different ports of the model and checking whether the model generates the output at the correct output port and in the expected amount of time. The details of the event file and test output of the tests conducted on this model are given in Part 3 of this assignment (not given here to avoid repetition).

7) Coupled Model: Wireless Gateway

This coupled model can be tested by giving inputs through different ports of the model and checking whether the model generates the output at the correct output port and in the expected amount of time. The details of the event file and test output of the tests conducted on this model are given in Part 3 of this assignment (not given here to avoid repetition).

8) Atomic Model: TupleClient

This atomic model can be tested by giving input through inCObject port of the model and checking whether the model generates the output at the outCObject port in the expected amount of time. The details of the event file and test output of the tests conducted on this model are given in Part 3 of this assignment (not given here to avoid repetition).

9) Atomic Model: TupleQueue

This atomic model can be tested by giving inputs through different ports of the model and checking whether the model generates the output at the correct output port and in the expected amount of time. Since this queue has not been connected to the Tuple Server yet, the 'done' event would have to be fed manually. The details of the event file and test output of the tests conducted on this model are given in Part 3 of this assignment (not given here to avoid repetition).

10) Atomic Model: TupleServer

This atomic model can be tested by giving inputs through different ports of the model and checking whether the model generates the output at the correct output port and in the expected amount of time. It is important to check that the model outputs the server object to the gateway the user is currently connected to and it correctly updates the gateway ID when it receives a handoff request from a wireless gateway. The details of the event file and test output of the tests conducted on this model are given in Part 3 of this assignment (not given here to avoid repetition).

11) Coupled Model: Tuple Space

This coupled model can be tested by giving inputs through different ports of the model and checking whether the model generates the output at the correct output port and in the expected amount of time. It is important to check that the model outputs the server object to the gateway the user is currently connected to. The details of the event file and test

output of the tests conducted on this model are given in Part 3 of this assignment (not given here to avoid repetition).

12) Coupled Model: Server

This coupled model can be tested by giving input through inC port of the model and checking whether the model generates the output at the outS port in the expected amount of time. The details of the event file and test output of the tests conducted on this model are given in Part 3 of this assignment (not given here to avoid repetition).

13) Coupled Model: Mobility Management (TOP)

This coupled model can be tested by giving input through inApp port of the model and checking whether the model generates the output at the outApp port in the expected amount of time. To verify that the object travels through each of the component in the expected amount of time and follows the expected path through gateways, we can connect testing output ports to the model at different points in the model. This would provide a detailed transition map of the input objects during their ‘journey’ through the model. The details of the event file and test output of the tests conducted on this model are given in Part 3 of this assignment (not given here to avoid repetition).

PART 3

1. Execution Results/Testing

1) Atomic Model: TransceiverQueue

The following events file was input to the TransceiverQueue atomic model with the preparation time of the TransceiverQueue equal to 10ms.

```
00:00:10:00 inApp 1
00:00:18:00 inGateway 2
00:00:30:00 done 1
00:00:45:00 position 3
00:00:50:00 done 2
00:00:52:00 done 3
```

The output is as follows:

```
00:00:10:010 outcobject 1
00:00:30:010 outsobject 2
00:00:50:010 location 3
```

As the output results show that the model outputs the inputs at the correct port and in the expected amount of time. The model is found to be working correctly.

2) Atomic Model: Transceiver

The following events file was input to the Transceiver atomic model with the send time of the Transceiver equal to 1 second and GatewayID equal to 1 initially. Note that since the Transceiver is to be connected to TransceiverQueue in the coupled model, it cannot receive any more events until it is done serving the current event. Hence for the tests the events are placed far enough to ensure that no events are missed.

```
00:00:10:00 inApp 1
00:00:30:00 position 2
00:00:40:00 inApp 3
00:00:50:00 inGateway 4
```

The output is as follows:

```
00:00:11:000 outgateway1 1
00:00:30:000 positionset2 2
00:00:41:000 outgateway2 3
00:00:51:000 outapp 4
```

As the output results show that the model outputs the inputs at the correct port and in the expected amount of time. For example, before the event 'positon 2' the user was connect to gateway1 and hence inApp event at 10 was outputted to outgateway1 but after the 'position 2' event that changes the active gateway to gateway2 the inApp event is outputted to gateway2. The model is thus found to be working correctly.

3) Atomic Model: MobilityModule

This model does not take any input. The output with handoff time equal to 30 seconds and the total simulation time of 3 minutes is shown as follows:

```
00:00:00:000 location 1
00:00:30:000 location 2
00:01:00:000 location 1
00:01:30:000 location 2
00:02:00:000 location 1
00:02:30:000 location 2
00:03:00:000 location 1
```

The results clearly show that the mobility module alternates the connected gateway after every 30 seconds (handoff time). The model is thus found to be working correctly.

4) Coupled Model: Mobile Terminal

The coupled model Mobile Terminal was tested by using the following input events file. The total simulation time was set as 5 minutes.

```
00:00:10:00 inApp 1
00:00:25:00 inG 2
00:00:40:00 inApp 3
00:00:50:00 inG 4
00:01:10:00 inApp 5
00:01:40:00 inApp 6
```


The output is as follows:

```
00:00:00:010 pg1 1
00:00:11:010 outg1 1
00:00:26:010 outapp 2
00:00:30:010 pg2 2
00:00:41:010 outg2 3
00:00:51:010 outapp 4
00:01:00:010 pg1 1
00:01:11:010 outg1 5
00:01:30:010 pg2 2
00:01:41:010 outg2 6
00:02:00:010 pg1 1
00:02:30:010 pg2 2
00:03:00:010 pg1 1
00:03:30:010 pg2 2
00:04:00:010 pg1 1
00:04:30:010 pg2 2
```

The output shows that the model delivers the input objects to the correct gateway. For example during the first 30 seconds of the simulation when the mobile terminal was connected to gateway1, it outputted the inApp 1 at 10 seconds to outg1. Similarly in the next 30 seconds when the mobile terminal was connected to the gateway2, it outputted the handoff request at pg2 and inApp 3 at 40 seconds to outg2. All objects coming from the gateway are correctly outputted to outApp. The model is thus found to be working correctly.

5) Atomic Model: GatewayQueue

The following events file was input to the Gatewayqueue atomic model with the preparation time of the Gatewayqueue equal to 10ms.

```
00:00:10:00 inCObject 1
00:00:18:00 inSObject 2
00:00:30:00 done 1
00:00:45:00 handoff 3
00:00:50:00 done 2
00:00:52:00 done 3
```

The output is as follows:

```
00:00:10:010 outcobject 1
00:00:30:010 outsobject 2
00:00:50:010 registerclient 3
```

As the output results show that the model outputs the inputs at the correct port and in the expected amount of time. The model is found to be working correctly.

6) Atomic Model: Gateway

The following events file was input to the Gateway atomic model with the send time of the Gateway equal to 100ms. Note that since the Gateway is to be connected to

GatewayQueue in the coupled model, it cannot receive any more events until it is done serving the current event. Hence for the tests the events are placed far enough to ensure that no events are missed.

```
00:00:10:00 inCObject 1
00:00:18:00 inSObject 2
00:00:45:00 handoff 3
```

The output is as follows:

```
00:00:10:100 outcobject 1
00:00:18:100 outsubject 2
00:00:45:100 registerclient 3
```

As the output results show that the model outputs the inputs at the correct port and in the expected amount of time. The model is found to be working correctly.

7) Coupled Model: Wireless Gateway

The coupled model Wireless Gateway was tested by using the following input events file.

```
00:00:10:000 inC 1
00:00:25:000 inS 2
00:00:40:000 handoff 2
00:00:41:000 inC 3
00:00:41:030 inS 4
00:01:40:000 inC 5
```

The output is as follows:

```
00:00:10:110 outc 1
00:00:25:110 outs 2
00:00:40:110 register 2
00:00:41:110 outc 3
00:00:41:220 outs 4
00:01:40:110 outc 5
```

As the output results show that the model outputs the inputs at the correct port and in the expected amount of time. The model is found to be working correctly.

8) Atomic Model: TupleClient

The following events file was input to the TupleClient atomic model with the write time of the TupleClient equal to 2 s. Note that since the TupleClient is to be connected to Queue in the coupled model, it cannot receive any more events until it is done serving the current event. Hence for the tests the events are placed far enough to ensure that no events are missed.

```
00:00:10:00 inCObject 1
00:00:18:00 inCObject 2
00:00:45:00 inCObject 3
```

The output is as follows:

```
00:00:12:000 outcobject 1
00:00:20:000 outcobject 2
00:00:47:000 outcobject 3
```

As the output results show that the model outputs the inputs at the output port in the expected amount of time. The model is found to be working correctly.

9) Atomic Model: TupleQueue

The following events file was input to the TupleQueue atomic model with the preparation time of the TupleQueue equal to 10ms.

```
00:00:10:00 inSObject 1
00:00:18:00 inSObject 2
00:00:30:00 done 1
00:00:45:00 registerClient 3
00:00:50:00 done 2
00:00:52:00 done 3
```

The output is as follows:

```
00:00:10:010 outsubject 1
00:00:30:010 outsubject 2
00:00:50:010 registerc 3
```

As the output results show that the model outputs the inputs at the correct port and in the expected amount of time. The model is found to be working correctly.

10) Atomic Model: TupleServer

The following events file was input to the TupleServer atomic model with the transfer time of the TupleServer equal to 100 ms and GatewayID equal to 1 initially. Note that since the TupleServer is to be connected to TupleQueue in the coupled model, it cannot receive any more events until it is done serving the current event. Hence for the tests the events are placed far enough to ensure that no events are missed.

```
00:00:10:00 inSObject 1
00:00:30:00 registerClient 2
00:00:40:00 inSObject 3
```

The output is as follows:

```
00:00:10:100 outsubject1 1
00:00:30:000 cregistered 2
00:00:40:100 outsubject2 3
```

As the output results show that the model outputs the inputs at the correct port and in the expected amount of time. For example, before the event 'registerClient 2' the active gateway was gateway1 and hence inSObject event at 10 was outputted to outSObject1 but after the 'registerClient 2' event that changes the active gateway to gateway2 the

inSObject event is outputted to outSObject2. The model is thus found to be working correctly.

11) Coupled Model: Tuple Space

The coupled model Tuple Space was tested by using the following input events file.

```
00:00:10:000 inC 1
00:00:25:000 inS 2
00:00:40:000 RC 2
00:00:45:000 inS 3
00:00:50:030 inS 4
00:01:40:000 inC 5
```

The output is as follows:

```
00:00:12:010 outC 1
00:00:25:110 outs1 2
00:00:45:110 outs2 3
00:00:50:140 outs2 4
00:01:42:010 outC 5
```

As the output results show that the model outputs the inputs at the correct port and in the expected amount of time. For example, in the beginning when the active gateway was gateway1, the model outputs the inS 2 object to outs1. But after receiving an event RC at 40 seconds, gateway2 becomes the active gateway. The model thus forwards the next inS objects inS 3 and inS 4 to outS2. All client objects are delivered successfully to outC. The model is thus found to be working correctly.

12) Coupled Model: Server

The coupled model Server was tested by using the following input events file. The queue preparation time was set as 10 ms while the CPU processing time was set as a normally distributed with a mean of 10 seconds and standard deviation of 2.

```
00:00:10:000 inC 1
00:00:15:000 inC 2
00:00:16:000 inC 3
00:00:40:000 inC 4
00:01:40:000 inC 5
```

The output is as follows:

```
00:00:19:985 outs 1
00:00:27:587 outs 2
00:00:35:512 outs 3
00:00:50:080 outs 4
00:01:48:375 outs 5
```

As the output results show that the model outputs the inputs at the correct port and in the expected amount of time. The model is found to be working correctly.

2. Integration Testing

All the coupled models in the system were combined to do the integration testing. Few additional output ports were added to the top model to trace the path of the objects inside the model during the simulation. The maximum time of the simulation was set as 5 minutes. The following input event files was used.

```
00:00:10:00 inApp 1
00:00:20:00 inApp 2
00:00:40:00 inApp 3
00:01:10:00 inApp 4
```

The output is as follows:

```
00:00:11:120 clientobjectoutofgateway1 1
00:00:13:130 clientobjectoutoftuple 1
00:00:21:120 clientobjectoutofgateway1 2
00:00:23:115 objectoutofserver 1
00:00:23:130 clientobjectoutoftuple 2
00:00:23:335 serverobjectoutofgateway1 1
00:00:24:345 outapp 1
00:00:30:732 objectoutofserver 2
00:00:30:952 serverobjectoutofgateway2 2
00:00:31:962 outapp 2
00:00:41:120 clientobjectoutofgateway2 3
00:00:43:130 clientobjectoutoftuple 3
00:00:51:055 objectoutofserver 3
00:00:51:275 serverobjectoutofgateway2 3
00:00:52:285 outapp 3
00:01:11:120 clientobjectoutofgateway1 4
00:01:13:130 clientobjectoutoftuple 4
00:01:23:210 objectoutofserver 4
00:01:23:430 serverobjectoutofgateway1 4
00:01:24:440 outapp 4
```

The extra output ports helps in locating the object and verifying the model. The results show that the model is giving the exact behavior as expected. For example, during the first 30 seconds, the first two input objects are outputted to gateway1 as the user is connected to gateway1 during that instant of time while during the next 30 seconds, the object 3 is outputted to gateway 2 as the mobile terminal is connected to gateway2. Similarly in the next 30 seconds, the object 4 was outputted to the gateway1.

Note that after the client objects are served by the server and are in tuple space. The space sends the object to the gateway to which the user is connected. For example, in the first 30 seconds, the first object was sent to the mobile terminal through gateway1 while in the next 30 seconds the object 2 and object 3 were sent to the mobile terminal through gateway2. Similarly, in the next 30 seconds, the object 4 was sent through gateway 1.

The model is thus giving expected behavior and the trace of the object through the model is in accordance with the prediction. A number of tests were performed to double check the behavior and the similar results were obtained.

The model is thus found to be working correctly.

3. Reaction of the Model to Different Inputs Than Those Defined in the Specifications.

The top model (mobility management) has only one input port called inApp. This port takes the id of the object to be serviced by the server. The id could be any integral value. Even if a floating number is entered it is cast to integer by the model and thus similar results are obtained as shown in the above section.

However, it is worth mentioning here that when the individual models are tested, there are only 2 possible values of the gateway ID (1 or 2). If during the individual testing of the model a value other than 1 or 2 is specified, the model is unable to locate the gateway with that ID. In an integrated model, however, since the IDs of the gateway are generated by the MobilityModule, it is always either 1 or 2 and hence no problems occur.

It should also be mentioned here that all queue models have some preparation time. If events are sent to the queue model spaced less than the preparation time, some of the events are lost. However, the preparation time is kept quite small to decrease the probability of any such occurrence.