# METHODOLOGIES FOR DISCRETE-EVENT MODELLING AND SIMULATION

(SYSC-5104)

## Assignment 1

Product Packing And Shipping

## Sukhvir Kaur Sahota

## Student ID: 101089844

sukhvirsahota@cmail.carleton.ca

**Part I: Description of the Conceptual Model**

**System under Analysis**: Product packing and shipping

**Conceptual Model Description**: In this assignment I programed the handing of ordering product by a Vendor. This model is based on online shopping where customer made a product request for Delivery. This model is composed of 5 atomic models and 2 coupled model.

The top model is the PackingAndShipping. It can be decomposed into 7 sub models (Atomic and Coupled):

1. RequestHandler model
2. Checkstock model
3. Packing model
4. Shipping model
5. Delivery model
6. Order
7. Vendor

PackingAndShipping

Checkstock DONE

IN

IN

OUT

Shipping

Packing

Order

ReqIN

OUT

Checkstock

OUT

RequestHandler

IN

Vendor

CloseOrder

IN

OUT

Delivery

The Components and functions are as follows:

* **RequestHandler**: The major function of this is to get order request and put it in a queue. This model is a first level of service. Stock Handler receives the request id and put it in the queue (which works on a first come, first serve technique) in order to Check the availability in stock.
* **Checkstock**: Here the stock checking process of a required item is done. It needs a random processing time and when stock checking is completed, it sends a DONE signal to the RequestHandler for next Product Id.
* **Packing**: Its main function is to pack the order and send it for shipping. The processing time is random for packing.
* **Shipping**: In this state product is shipped for delivery and processing time is also random for that.
* **Delivery**: Its major function is to safely deliver the product to the customer which turns to close the order.

**Part II:**

**FORMAL SPECIFICATION**

**The formal specification of all atomic models**

1) RequestHandler model

Checkstock OUT

ReqIn

RequestHandler

MODEL

Checkstock DONE

RequestHandler=<S, X, Y, δint, δext, λ, ta>

X ={ReqIn, CheckstockDONE }

Y = {CheckstockOUT}

δint ( Active) = Passive

δint ( Passive ) =Active

δext (IN, Passive) =Active

δext (DONE, Passive) = Active

ta( Active) ={RequestHandling time }

ta( Passive ) = infinite

λ (Active) = pid

**Pseudo code**

Sigma= infinite, Phase={ Active, Passive}, PreparationTime=10;

δext (s,e,x)

{

case port

in : Case phase

save the product Id in the Request Handling queue

done : Case phase

get the first product Id in the queue

}

δint (s)

{

Case Phase

Serving: passive

passive: // Should not happen.

}

λ (s)

{

send the product Id (pid) to the out port

}

**2) Checkstock model**

Checkstock

MODEL

CheckstockIN

Packing OUT

Checkstock DONE

Checstock=<S, X, Y, δint, δext, λ, ta>

X ={CheckstockIN}

Y = {PackingOUT, CheckstockDONE }

δint ( Active) = Passive

δint ( Passive ) = Active

δext (IN, Passive ) =Active

ta( Active) ={Checkstock time }

ta( Passive ) = infinite

λ (Active) = pid

**Pseudo code**

δext (s,e,x)

{

case port

in : case phase

passive: get the product id,

hold in Serving during processing-time

Serving: discard the input value

}

δint (s)

{

case

Serving: passive

passive:

}

λ (s)

{

send the pid to the PackingOUT port

send the pid to the ChecstockDONE port

}

3) Packing model

ShippingOUT

PackingIN

Packing

MODEL

Packing =<S, X, Y, δint, δext, λ, ta>

X = {PackingIN}

Y = {ShippingOUT}

δint ( Active ) =Passive

δint ( Passive) = Active

δExt (IN, Passive ) =Serving

ta(Active ) ={Packing time }

ta(Passive) =infinite

λ (Active) = pid

**Pseudo code**

δext (s,e,x)

{

case port

in : case phase

passive: get the pid,

hold in Serving during processing-time

Serving: discard the input

}

δint (s)

{

case phase

Serving: passive

passive:

}

out(s)

{

send the product with pid to the out port

}

4) Shipping model

ShippingIN

DeliveryOUT

Shipping

MODEL

Shipping =<S, X, Y, δint, δext, λ, ta>

X = {ShippingIN}

Y = {DeliveryOUT}

δint ( Active ) =Passive

δint ( Passive) = Active

δExt (IN, Passive ) =Active

ta(Active ) ={Shipping time }

ta(Passive) =infinite

λ (Active) = pid

**Pseudo code**

δext (s,e,x)

{

case port

in : case phase

passive: get the product with,

hold in Serving during processing-time

Serving: discard the input

}

δint (s)

{

case phase

Serving: passive

passive:

}

out(s)

{

send the product with pid to the out port

}

5) Delivery model

CloseOrderOUT

DeliveryIN

Delivery

MODEL

Packing =<S, X, Y, δint, δext, λ, ta>

X = {DeliveryIN}

Y = {CloseOrderOUT}

δint ( Active ) =Passive

δint ( Passive) = Active

δExt (IN, Passive ) =Active

ta(Active ) ={Packing time }

ta(Passive) =infinite

λ (Active) = pid

**Pseudo code**

δext (s,e,x)

{

case port

in : case phase

passive: get the product with pid for delivery,

hold in Serving during processing-time

Serving: discard the input

}

δint (s)

{

case phase

Serving: passive

passive:

}

out(s)

{

send the product with pid to the out port

}

**The formal specification of all couple models**

1) Order model

Order

IN

OUT

Checkstock

RequestHandler

ReqIN

Checkstock DONE

OUT

Order=<X, Y, {RequestHandler, Checkstock}, {Ii}, {Zij}, SELECT>

X ={ ReqIN};

Y = { PackingOUT};

Mi ={ RequestHandler, Checkstock }

I(RequestHandler) = Checkstock;

I(Checkstock) ={ RequestHandler, self}

Z(RequestHandler) = Checkstock;

Z(Checkstock) = RequestHandler;

Z(Checkstock) = self;

SELECT : ({RequestHandler, Checkstock }) = RequestHandler

2) Vendor model

OUT

OUT

shipping

Vendor

Packing

IN

IN

VENDOR =<X, Y, {Packing, Shipping}, {Ii}, {Zij}, SELECT>

X ={ PackingIN};

Y = {Delivery OUT};

Mi ={ Packing, Shipping }

I(Packing) = Shipping;

I(Shipping) ={ Packing, self}

Z(Packing) = Shipping;

Z(Shipping) = Packing;

Z (Shipping) = self;

SELECT : ({Packing, Shipping }) = Packing

PackingAndShipping Model

ReqIN

CloseOrder

PackingAnd Shipping

PackingAndShipping =<X, Y, {Stock, Vendor, Delivery}, {Ii}, {Zij}, SELECT>

X ={ ReqIN};

Y = { CloseOrder};

Mi ={ Stock, Vendor, Delivery }

I(Order) ={ Vendor, self}

I(Vendor) ={ Order,Delivery}

I(Delivery) ={ Vendor}

Z(SOrder) = Vendor;

Z(Order) = self;

Z(Vendor) = stock;

Z(delivery)=self;

SELECT : ({Order,Vendor,Delivery}) = Order

({Order,Vendor}) = Order

({Vendor,Delivery}) = Vendor;

**Part III:**

All atomic and couple models are tested separately and with linking by adding different combinations of inputs in event files(.ev).

**1. RequestHandler model testing (atomic)**

In this atomic initial tate is idel before getting an order request. The preparation time for this model is 00:00:10:000. For testing this model, Event file is *RequestHandler.ev*

00:00:05:000 ReqIn 1

00:01:13:000 done 1

00:02:10:000 ReqIn 35

00:03:00:000 ReqIn 36

So the CD++ output file is:

00:00:15:000 out 1

00:02:20:000 out 35

As it is clear that until a done confirmation is not coming from next model the Request Handler model did not send the next input.

**2. Checkstock model testing**

. For testing this model, Event file is *Checkstock.ev*

00:00:05:00 in 15

00:01:13:00 in 2

So the CD++ output file is:

00:00:08:963 out 15

00:00:08:963 done 15

00:01:13:388 out 2

00:01:13:388 done 2

As it is noticeable that after stock checking this model simulatneously send two outputs at the same time

• out is sent for packing

• done is for requesting next product

**3. Packing model testing**

For testing this model, Event file is *Packing.ev*

00:00:10:00 in 1

00:00:30:00 in 3

00:00:50:00 in 25

So the CD++ output file is:

00:00:11:987 out 1

00:00:30:796 out 3

00:00:50:957 out 25

The output is generated after random processing time.

**4. Shipping model testing**

For testing this model, Event file is *Shipping.ev*

00:00:10:00 in 1

00:00:30:00 in 3

00:00:50:00 in 25

So the CD++ output file is:

00:00:10:00 in 1

00:00:30:00 in 32

00:00:50:00 in 25

The output is generated after random processing time.

As it is noticeable that after stock checking this model simulatneously send two outputs at the same time

* out is sent for packing
* done is for requesting next product

**5. Delivery model testing**

For testing this model, Event file is *Delivery.ev* with the preparation time of 00:00:10:000

00:00:30:000 in 25

00:00:60:000 in 75

So the CD++ output file is:

00:00:40:000 closeorder 25

00:01:10:000 closeorder 75

**3. Add model Checkfile and test it together with Checkin model**

[top]

components : RequestHandler@RequestHandler Checkstock@Checkstock

out : out

in : ReqIn

Link : ReqIn ReqIn@RequestHandler

Link : out@RequestHandler in@Checkstock

Link : out@Checkstock out

Link : done@Checkstock done@RequestHandler

[RequestHandler]

preparation : 00:00:10:000

[Checkstock]

distribution : Normal

mean : 4

deviation : 3

Input file *Link1.ev*

00:00:05:00 ReqIn 1

00:01:13:00 ReqIn 2

00:02:30:00 ReqIn 3

CD++ output

00:00:18:963 out 1

00:01:23:388 out 2

00:02:40:873 out 3

**Whole System testing**

[top]

components : Delivery@DeliveryType Order Vendor

out : closeOrder

in : ReqIn

Link : ReqIn ReqIn@Order

Link : out@Order in@Vendor

Link : out@Vendor in@Delivery

Link : closeOrder@Delivery closeOrder

[Delivery]

preparation : 00:00:10:000

[Order]

components : RequestHandler@RequestHandler Checkstock@Checkstock

out : out

in : ReqIn

Link : ReqIn ReqIn@RequestHandler

Link : out@Checkstock out

Link : out@RequestHandler in@Checkstock

Link : done@Checkstock done@RequestHandler

[RequestHandler]

preparation : 00:00:10:000

[Checkstock]

distribution : Normal

mean : 4

deviation : 3

[Vendor]

components : Packing@Packing Shipping@Shipping

out : out

in : in

Link : in in@Packing

Link : out@Shipping out

Link : out@Packing in@Shipping

[Packing]

distribution : Normal

mean : 2

deviation : 1

[Shipping]

distribution : Normal

mean : 2

deviation : 1

The all above mention atomic and couple models describes the principle of first come first serve basis. The order which is requested first is served first and other requests have to wait until the serving of first order is completed.

Test 1

I have conducted a test for this with name *packingAndshipping.ev*

00:00:20:000 ReqIn 1

00:00:35:000 ReqIn 2

00:01:10:000 ReqIn 3

CD++ output

00:00:45:716 closeorder 1

00:01:02:448 closeorder 2

00:01:41:650 closeorder 3

Test 2

I have conducted another test which proves that the order which come first will serve first without giving priority to any other.

*packingAndshipping.ev*

00:00:30:000 ReqIn 1

00:00:10:000 ReqIn 2

00:03:00:000 ReqIn 3

CD++ output

00:00:55:716 closeorder 2