Modeling a Pharmaceutical Plant Using the Graphical Interface of CD++

Saman Jafartayari [1]

***Abstract*—In this paper, a simple model of a pharmaceutical plant is simulated using a new extension of CD++ toolkit. With this feature, a model is written using state machines. The graphical interface of CD++ toolkit simplifies the process of modeling and simulation and as a result cost and time is saved. Moreover, modification and debugging of such model would be easier and more straightforward. In addition, since this tool permits the user to combine only-graphical notations with standard representation of C++, both the capabilities of CD++ have been employed to simulate the simplified model of the pharmaceutical plant in a more complex way.**

***Index Terms*—CD++, State Machines, DEVS, DEVS graphs, Discrete event simulation**

I. Introduction

GRAPHICAL notations are always used to present a model in a more clear and accurate way. This can help the modeler to see if the model's behavior is as desired or not. Besides, it can give him/her a good outlook of the model and find out where the shortcomings might be originated from. One of the graphical notations used widely in computer science is the state diagrams. They show the different states of a model and the transition conditions that can make the model move from one state to another. In DEVS-graphs, not only the states and transition conditions are shown, the internal and external transitions, time base, inputs, outputs and the output functions are defined as well.

In this paper, a simulation of a simplified model of a pharmaceutical factory which itself is a part of a supply chain of a specific tablet (pill) is presented. Since the graphical interface of the CD++ permits the users to combine only-graphical notations with standard representation of C++, both of these methods are used in defining the model and simulating it. Therefore, to keep the complexity of the model, the main coupled model includes both graphical and standard sub-models.

II. Background

A. Supply Chain Management

Supply Chain Management (SCM) is the key to having a competitive edge in the current global market. It involves efficiently managing the flow of materials, cash, information and services. In order to evaluate various advantages and disadvantages of different SCM models, simulation before implementation is key [1].

B. DEVS graphs

The formalism used for describing the graphical DEVS models is somehow the same as the DEVS formalism used previously for describing the behavior of a discrete event system in which we had inputs, states, outputs, internal and external functions, output function and time advance functions. The atomic DEVS is defined formally by[3]:

$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$

A coupled DEVS model is described by inputs, outputs, set of models (atomic or coupled), index of influences and the function that converts the output of one model to an input port of another. Therefore, a coupled DEVS model is defined formally by [3]:

$CM = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\} >$

The formalism used for defining a DEVS-graph is an extended version of the one used for transitional DEVS models. This formalism for atomic graphs includes the definition of inputs, states based on internal and external transition functions. It is describe by:

$GGAD = < X_M, S, Y_M, \delta_{int}, , \delta_{ext}, \lambda, D >$
$X_M = \{(p,v) | p \in IPorts, v \in X_p\}$ set of input ports;
$Y_M = \{(p,v) | p \in OPorts, v \in Y_p\}$ set of output ports;
$S = B \times P(V)$ states of the model,
$B = \{ b | b \in Bubbles \}$ set of model states.
$V = \{ (v,n) | v \in Variables, n \in R_0 \}$ intermediate state variables of the model and their values [3].

1. S. Jafartayari . E-mail : sjafa059@uottawa.ca

III. CONCEPTUAL MODEL

The model used for this simulation is a part of a whole supply chain of a pharmaceutical product. The whole supply chain covers the system that contribute to the provision of raw material, production of the tablets, distributing the final products to retailers and delivering them in the hands of customers.

One chunk of this chain is the mother factory which acts the main role in this process. This factory includes an administration that controls all the transactions inside the factory and acts as a communicator between different sections. The warehouse holds the inventory of raw materials and final products in a way that whenever new batches of raw materials are delivered or a batch of final product is produced, they are sent by the administrator to the warehouse. This batch of final products are produced at the pharmaceutical manufacturing plant(PMP) and with the order that is placed by the administrator to replenish the capacity of the warehouse. Another role of the administrator is to send the demanded amount of products to retailers as soon as enough stock is available in the warehouse. The PMP itself has some subsection through them the raw material is transformed to final packed product.

DEVS model block diagram of the initial factory model is shown in Figure 1. For simplicity purposes, the PMP model which was initially considered and coded using C++ standard representation as a coupled model is defined using graphical interface of CD++ and is considered as a coupled model consisting of different atomic model representing different section of the plant. The factory's warehouse's C++ model is replaced with a graphical DEVS. It resulted the same outputs and same characteristics from the whole factory model. This modified version of block diagram is shown in Figure 2.

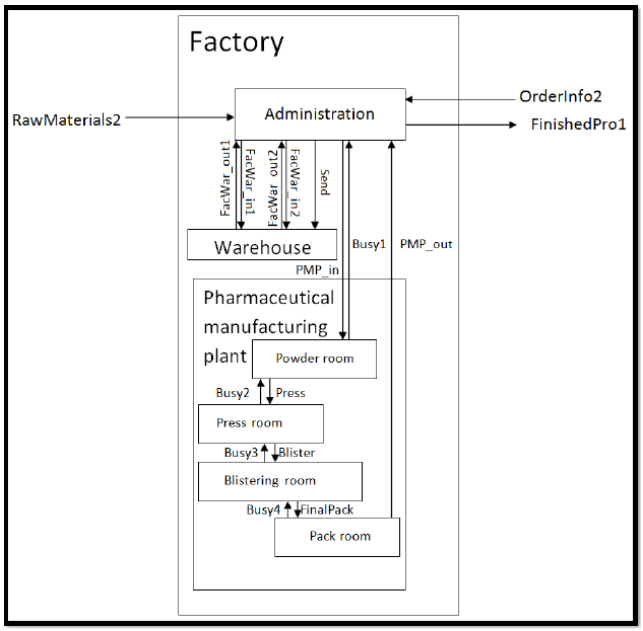

Figure 1: Initial DEVS model block diagram of factory

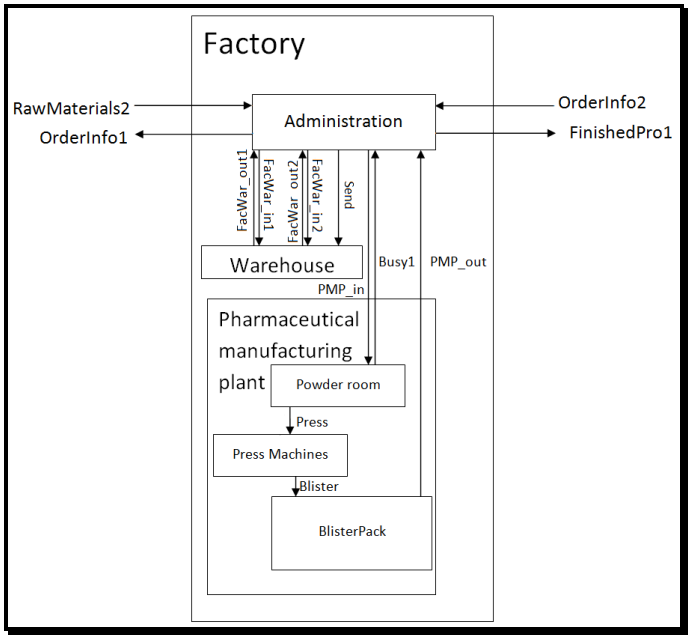

Figure 2: Modified DEVS model block diagram

A more detailed description of all the models is followed:

A. *Administration*

The administrative duties such as: receiving good from the supplier, attaining the orders from the distributor, placing orders with the supplier and shipping finished products to the distributor; are modeled by the administrator atomic model. It is comprised of seven input ports: RawMaterials2, OrderInfo2, FacWar_out1, FacWar_out2, Busy1, PMP_out and Send. It should be noted that the administrator always tries to maintain a full capacity of finished products.

i. RawMaterials2 is the port through which the supplier transports raw materials to the factory. In order for the supplies to be transferred from the supplier to the factory there exists variable lead times (due to factors such as transportation delay). However, for the purpose of simplicity, we will model this as a constant time delay of 1 day.

ii. OrderInfo2 is a one-way port that originates from the distributor and terminates at the factory. It is used to place orders to replenish the distributor's inventory. Realistically the delay of information flow can vary from case to case. To avoid complexity we set this delay to a constant value of 12 hours.

iii. FacWar_out1 is the port through which the warehouse sends its raw materials to the administration.

iv. FacWar_out2 is the port through which the warehouse sends its finished products to the administration.

v. Busy1 is the port that is Boolean; it is set to true if the powder atomic model (coupled in the factory's pharmaceutical manufacturing plant) is busy and false otherwise.

vi. PMP_out is the port through which the factory's pharmaceutical manufacturing plant (in particular the packing model) communicates with administration.

vii. Send is the port through which the factory's administrator communicates to the warehouse how many raw materials or finished products it needs.

In addition to input ports, the Administrator also encompasses four output ports: FinishedPro1, FacWar_in1, FacWar_in2 and PMP_in.

i. FinishedPro1 port is utilized by the factory to ship its finished products to the distributor. The shipment delay for this modeled as a constant value of 1 day.

ii. FacWar_in1 is the port through which the administration transports raw materials to the factory's warehouse.

iii. FacWar_in2 is the port through which the administration transports finished products to the factory's warehouse.

iv. PMP_in is the port through which the administration communicates with the factory's pharmaceutical manufacturing plant (in particular the powder room model).

B. Factory Warehouse

The role of the factory's warehouse is also to act as a storage facility for the factory's raw materials and finished products. It has a maximum carrying capacity of 80 batches for raw materials (represented as positive integers ranging from 111 to 180) and 20 batches for finished products (represented as positive integers ranging from 181 to 200). In the event of an overflow of either raw materials and/or finished products, the redundant batches will just be discarded without notification. It receives raw materials and finished products from the administrator through the ports FacWar_in1 and FacWar_in2, respectively. In addition, it sends raw materials and finished products as the administrator requires it through the ports FacWar_out1 and FacWar_out2, respectively. Lastly, the number or products (both raw materials and finished products) that need to be sent to the administrator is communicated through the port Send.

C. Pharmaceutical Manufacturing Plant (PMP)

The role of the pharmaceutical manufacturing plant is to manufacture pills (i.e. convert raw materials into a pill that will then be stored in the warehouse and then sent to the distributor). Since the pharmaceutical manufacturing plant has no internal storage, it is important to have a negative feedback from the plant. The negative feedback is the Boolean port: Busy1.

IV. Model Specifications

The factory is modeled as a two-level DEVS model with three components that are described as atomic models. Initially this same model had been design in three levels, two coupled models with six atomic components. The DEVS formal specification for each model is outlined below starting from atomic models and concluding with the factory itself. This also proves to be the appropriate order for implementation and testing.

A. Atomic models

Administrator

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

Where

InPorts = {"OrderInfo2", "RawMaterials2", "FacWar_out1", "FacWar_out2"}, where XOrderInfo2 = {1,2,3,...,18,19,20}, XRawMaterials2 = {1,2,3,...,58,59,60}, XFacWarout1 = {111, 112, 113, ..., 178, 179, 180}, XFacWarout2 = {181,182,183, ... , 197, 198, 200}

X = {(p,v)|p∈InPorts, v∈Xp} is the set of input ports and values

OutPorts = {" FinishedPro1","FacWar_in1", "FacWar_in2", "Send"}, where YFinishedPro1 = {1,2,3,...,18,19,20}, YFacWar_in1 = {111, 112, 113, ..., 178, 179, 180}, YFacWar_in2 = {181,182,183, ... , 197, 198, 200}, YSend = {100, 102, 103, ..., 198, 199, 200}

Y = {(p,v)|p∈OutPorts, v∈Yp } is the set of output ports and values

S = {phase, sigma, currentship, current_finished, warehouseFinishRequest, warehouseFinishQuantity, current_raw, rawRecieved, warehouseRaw, rawMaterial, readyToPMP, finishedProduct, readyToShip, powderStat, sent, inProgress, storefinishedProduct, shipToWarehouse, requestRaw, warehouseRawQuantity}

= {"passive", "active"} × R0+ × {0,1,2,...,18,19,20} × {0,1,2,...,18,19,20} × {true, false} × {181,182,183,...,198,199,200} × {0,1,2,...,78,79,80} × {0,1,2,...,78,79,80} × {true, false} × {0,1,2,3,4} × {true, false} × {0,1,2,...,18,19,20} × {true, false} × {true, false} × {true, false} × {0,1,2,...,18,19,20} × {101,102,103,...,178,179,180} × {true, false} × {true, false} × {104}

δext(phase, sigma, currentship, current_finished, warehouseFinishRequest, warehouseFinishQuantity, current_raw, rawRecieved, warehouseRaw, rawMaterial, readyToPMP, finishedProduct, readyToShip, powderStat, sent, inProgress, storefinishedProduct, shipToWarehouse, e){

```
if(msg.port() == OrderInfo2){
if(msg.value() <=0){
Display error message! Invalid order!
}else if((msg.value() >= 1) && (msg.value() <= 20)){
if (currentship+msg.value()>current_finished){
```

Display error message! Can't ask for more than available!
}else{
currentship= currentship+msg.value();
warehouseFinishRequest=true;
warehouseFinishQuantity=currentship+180;
current_finished=current_finished-currentship;
}
} else {
Display error message! No order can be greater than 20 units, distributor should never request more.
}
}
if (msg.port()== RawMaterials2){
if (msg.value()<=0){
readyToPMP=true;
}
}
if (msg.port()== FacWar_out2){
if (msg.value()>20){
Display Error Message because total capacity for warehouse is only 20 so this should never happen
}else if(msg.value () <=20 && msg.value()>0){
finishedProduct=msg.value();
readyToShip=true;
}else{
Display Error Message because simulation should never reach this
}
}
if(msg.port()==Busy1){
if(msg.value()==1){
powderStat=true;
}else if (msg.value()==0){
powderStat=false;
sent=false;
}else{
cout<<"Error: Port Busy1 should never receive an input other than 0 or 1!"<<endl;
}
}
if(msg.port()==PMP_out){
if (msg.value()!=1){
Display Error Message because PMP_out should only output a value of 1
}else if(msg.value()==1){
int temp=msg.value();
if (current_finished>=20){
Display Error Message because Simulation should never get here!
}
inProgress=inProgress-1;
storefinishedProduct=temp+180;
current_finished=current_finished+1;
shipToWarehouse=true;
}
}
}
δint(phase, sigma, current_finished, inProgress, current_raw, powderStat, sent, requestRaw, warehouseRawQuantity){
if(active){
if((20-current_finished-inProgress)>0 && (20-current_finished-inProgress)<=20 && current_raw>=4 && !powderStat && !sent){
requestRaw=true;
warehouseRawQuantity=4+100;
current_raw=current_raw-4;
}else{
passivate();
}
} else {
//this will never happen
if(passive){
Display Error Message!
}
}
}
λ (phase, sigma, warehouseFinishRequest, requestRaw, sent, readyToPMP, powderStat, inProgress, readyToShip, shipToWarehouse){
if(warehouseFinishRequest){
warehouseFinishRequest=false;
send output "warehouseFinishQuantity" to send
}
if(requestRaw){
requestRaw=false;
sent=true;
send output "warehouseFinishQuantity" to send
}
if(warehouseRaw){
warehouseRaw=false;
send output "current_raw" to FacWar_in1
}
if(readyToPMP && !powderStat && ((inProgress+current_finished)<20)){
readyToPMP=false;
powderStat=true;
inProgress=inProgress+1;
send output "rawMaterial" to PMP_in
}
if(readyToShip){
readyToShip=false;
send output "finishedProduct" to FinishedPro1
}
if(shipToWarehouse){
shipToWarehouse=false;
send output "storefinishedProduct" to FacWar_in2
}
}
ta("passive") = ∞;
ta("active") = (0,0,1,0);

Testing Strategy for Administrator

1. Verify the effects of overflow – if the shipment will incur overflow, the shipment must not be accepted and an error message should be displayed.
2. Verify the effects non-positive - error message should be displayed.
3. Verify the current_finished is always at max capacity, if it is not request raw materials from administrator and send to PMP_in.
4. Verify the raw_current is decreased by finished_units when there is an input to FacWar_out1.
5. Verify the finished_current is decreased by finished_units when there is an input to FacWar_out2.
6. Verify the raw_current is increased by raw_units when there is an output from FacWar_out1.
7. Verify the finished_current is increased by finished_units when there is an input from FacWar_out2.

Warehouse

Warhouse = $< X_M, S, Y_M, \delta_{int}, , \delta_{ext}, \lambda, D >$

Where

InPorts = {"FacWar_in1", "FacWar_in2", "Send"}, where XFacWar_in1 = {111, 112, 113, ..., 178, 179, 180}, XFacWar_in2 = {181,182,183, ... , 197, 198, 200}, YSend = {100, 102, 103, ..., 198, 199, 200}

X_M = {(p,v)|p∈InPorts, v∈Xp} is the set of input ports and values

OutPorts = {"FacWar_out1", "FacWar_out2"}, where YFacWarout1 = {111, 112, 113, ..., 178, 179, 180}, YFacWarout2 = {181,182,183, ... , 197, 198, 200}

Y_M = {(p,v)|p∈OutPorts, v∈Yp } is the set of Output ports and values .

S = B x P(V) states of the model,
B = { wait, SendRaw, SendFinished, RawReceived, FinishedReceived, *InvalidInput*} set of model states.
V = { currentRaw, currentFinished, RawToBeSent, FinishedToBeSent, RawCapacity, FinishedCapacity}.

The warehouse model was designed using the graphical interface of CD++ toolkit. The design is depicted in Figure 3.

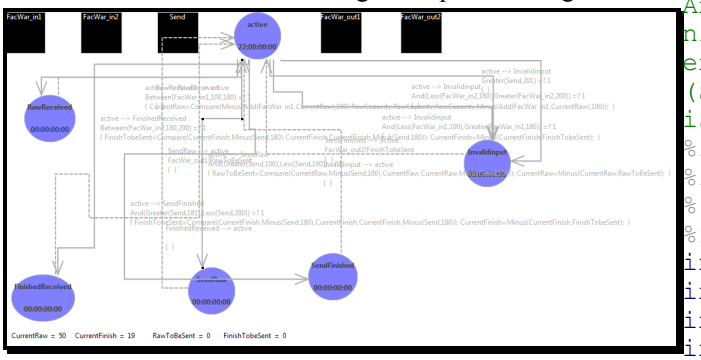

Figure 3: FactoryWarehouse model designed using DEVS-Graphs

The code associated with this graphical notation in a CDD file is as followed:

```
[Top]
%input ports declaration
in:FacWar_in1 FacWar_in2 Send
%output ports declaration
out:FacWar_out1 FacWar_out2
%variables declaration
var:CurrentRaw CurrentFinish RawToBeSent
FinishTobeSent RawCapacity FinishCapacity
%set of states
state:active RawReceived FinishedReceived
SendRaw SendFinished Invalidinput
%initial state definition
initial:active
%external transition functions (ex. In the
%first line we have a transition between
%two states of active and RawReceived with
%some conditions in which the value of the
%input ports are tested.
ext:active RawReceived
Between(FacWar_in1,100,180)?1{CurrentRaw=C
ompare(Minus(Add(FacWar_in1,CurrentRaw),10
0),RawCapacity,RawCapacity,RawCapacity,Min
us(Add(FacWar_in1,CurrentRaw),100));}
ext:active Invalidinput
And(Less(FacWar_in1,100),Greater(FacWar_in
1,180))?1
ext:active Invalidinput
And(Less(FacWar_in2,180),Greater(FacWar_in
2,200))?1
ext:active Invalidinput
Greater(Send,201)?1
int:Invalidinput active
ext:active FinishedReceived
Between(FacWar_in2,180,200)?1{FinishTobeSe
nt=Compare(CurrentFinish,Minus(Send,180),C
urrentFinish,CurrentFinish,Minus(Send,180)
);CurrentFinish=Minus(CurrentFinish,Finish
TobeSent);}
ext:active SendRaw
And(Greater(Send,100),Less(Send,180))?1{Ra
wToBeSent=Compare(CurrentRaw,Minus(Send,10
0),CurrentRaw,CurrentRaw,Minus(Send,100));
CurrentRaw=Minus(CurrentRaw,RawToBeSent);}
ext:active SendFinished
And(Greater(Send,181),Less(Send,200))?1{Fi
nishTobeSent=Compare(CurrentFinish,Minus(S
end,180),CurrentFinish,CurrentFinish,Minus
(Send,180));CurrentFinish=Minus(CurrentFin
ish,FinishTobeSent);}
%internal transition functions
%lifetime of the bubbles are defined
%( Ex. For "wait" state the holdin
%function has the time value of infinity)
int:RawReceived active
int:FinishedReceived active
int:SendRaw active FacWar_out1!RawToBeSent
int:SendFinished active
FacWar_out2!FinishTobeSent
```

```
%initialization of the variables
active:22:00:00:00
RawReceived:00:00:00:00
FinishedReceived:00:00:00:00
SendRaw:00:00:00:00
SendFinished:00:00:00:00
Invalidinput:00:00:00:00
CurrentRaw:50
CurrentFinish:19
RawToBeSent:0
FinishTobeSent:0
RawCapacity:80
FinishCapacity:20
```

Testing Strategy for the Warehouse

1. Verify the effects of overflow – all surplus units should be discarded without warning.
2. Verify the effects non-positive – they should not be accepted.
3. Verify the raw_current_units is decreased by raw_units when there is an output to FacWar_out1.
4. Verify the finished_current_units is decreased by finished_units when there is an output to FacWar_out2.
5. Verify the raw_current_units is increased by raw_units when there is an input from FacWar_out1.
6. Verify the finished_current_units is increased by finished_units when there is an input from FacWar_out2.

Powder Room

PowderRoom = <X, Y, S,δext,δint,λ, ta>
Where
InPorts = {"PMP_in"}, where XPMP_in = {4}
X M = {(p,v)|p∈InPorts, v∈Xp} is the set of input ports and values
OutPorts = {"Busy1", "Press"}, where YBusy1 = {true,false}, YPress = {1}
Y M = {(p,v)|p∈OutPorts, v∈Yp } is the set of Output ports and values
S = B x P(V) states of the model,

B = {wait, ready, Powdering, InvalidInput, Keep}
set of model states.
V = {Powder. PowderStat}

The Powder Room model was designed using the graphical interface of CD++ toolkit. The design is depicted in Figure 4.

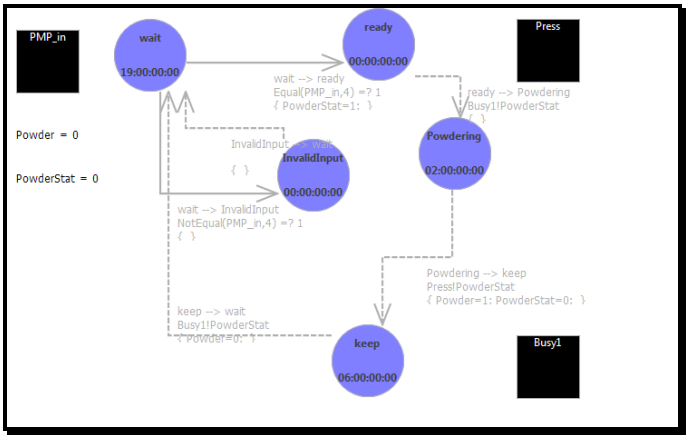

Figure 4: PowderRoom model designed using DEVS-Graphs

The code associated with this graphical notation in a CDD file is as followed:

```
[Top]

in:PMP_in
out:Press Busy1
var:Powder PowderStat
state:wait Powdering keep ready
InvalidInput
initial:wait
ext:wait ready
Equal(PMP_in,4)?1{PowderStat=1;}
ext:wait InvalidInput NotEqual(PMP_in,4)?1
int:InvalidInput wait
int:ready Powdering Busy1!PowderStat
int:Powdering keep Press!PowderStat
{Powder=1;PowderStat=0;}
int:keep wait Busy1!PowderStat {Powder=0;}
wait:19:00:00:00
Powdering:02:00:00:00
keep:06:00:00:00
ready:00:00:00:00
InvalidInput:00:00:00:00
Powder:0
PowderStat:0
```

Testing Strategy for Powder Room

1. Verify that the input received through PMP_in port is checked to have the value of 4.
2. Verify that the values received through the port Busy2 is either a zero or one.
3. Verify that the message sent through the port Busy1 is either 0 or 1.
4. Validate that the port Busy1 sends a value of 1 when the powder room starts the process of converting raw materials to powder.
5. Verify that the port Busy1 sends a value of 0 the moment the powder room is free.
6. Ensure that processing time of the powder room is 2 hours.

7. Verify that the port Press sends a value of 1 once the powder room has converted the raw materials to powder and that the press room is free.

Press Room

PressRoom = <X, Y, S,δext,δint,λ, ta>
Where
InPorts = {"Press"}, where X_{Press} = {1}
X_M = {(p,v)|p∈InPorts, v∈Xp} is the set of input ports and values
OutPorts = {"Blister"}, where $Y_{Blister}$ = {1}Y_M = {(p,v)|p∈OutPorts, v∈Yp } is the set of Output ports and values
S = B x P(V) states of the model,
B = {wait, ready, Pressing, InvalidInput} et of model states.
V = {Press. PressStat}

The Press Room model was designed using the graphical interface of CD++ toolkit. The design is depicted in Figure5.

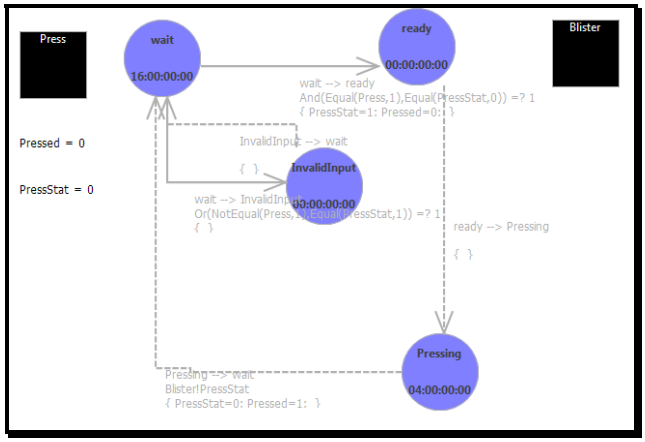

Figure 5: PressRoom model designed using DEVS-Graphs

The code associated with this graphical notation in a CDD file is as followed:

```
[Top]
in:Press
out:Blister
var:Pressed PressStat
state:wait Pressing ready InvalidInput
initial:wait
ext:wait ready
And(Equal(Press,1),Equal(PressStat,0))?1{P
ressStat=1;Pressed=0;}
ext:wait InvalidInput
Or(NotEqual(Press,1),Equal(PressStat,1))?1
int:InvalidInput wait
int:ready Pressing
int:Pressing wait Blister!PressStat
{PressStat=0;Pressed=1;}
wait:16:00:00:00
Pressing:04:00:00:00
ready:00:00:00:00
InvalidInput:00:00:00:00
Pressed:0
PressStat:0
```

Testing Strategy for Press Room

1. Verify that the input received through Press port is checked to have the value of 1.
2. Ensure that processing time of the press room is 4 hours.
3. Verify that the port Blister sends a value of 1 once the press room has converted the powder to pill/tablet and that the blister room is free.

Blister-Pack
BlisterPack = <X, Y, S,δext,δint,λ, ta>
Where
InPorts = {"Blister"}, where X_{Press} = {1}
X_M = {(p,v)|p∈InPorts, v∈Xp} is the set of input ports and values
OutPorts = {"PMP_out"}, where Y_{PMP_out} = {1}
Y_M = {(p,v)|p∈OutPorts, v∈Yp } is the set of Output ports and values
S = B x P(V) states of the model,

B = {wait, ready, Blistering, InvalidInput}
set of model states.
V = {Blisered. BlisterStat}

The BlisterPack model was designed using the graphical interface of CD++ toolkit. The design is depicted in Figure 6.

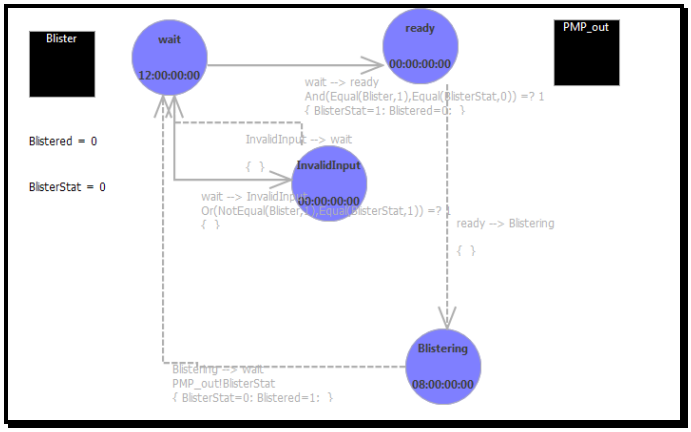

Figure 6: BlisterPack model desinged using DEVS-Graphs

The code associated with this graphical notation in a CDD file is as followed:

```
[Top]
in:Blister
out:PMP_out
var:Blistered BlisterStat
state:wait Blistering ready InvalidInput
initial:wait
```

```
ext:wait ready
And(Equal(Blister,1),Equal(BlisterStat,0))
?1{BlisterStat=1;Blistered=0;}
ext:wait InvalidInput
Or(NotEqual(Blister,1),Equal(BlisterStat,1
))?1
int:InvalidInput wait
int:ready Blistering
int:Blistering wait PMP_out!BlisterStat
{BlisterStat=0;Blistered=1;}
wait:12:00:00:00
Blistering:08:00:00:00
ready:00:00:00:00
InvalidInput:00:00:00:00
Blistered:0
BlisterStat:0
```

Testing Strategy for Blister-Pack

1. Verify that the input received through Blister port is checked to have the value of 1.
2. Ensure that processing time of the pack room is 8 hours.
3. Verify that the port PMP_out sends a value of 1 once the blister pack room has done blistering and packaging the tablets.

B. Coupled models

Pharmaceutical Manufacturing Plant (PMP)

PMP = < X, Y, D, {Md | d∈D}, EIC, EOC, IC, Select>
Where
InPorts = {"PMP_in"}, where XPMP_in = {1, 2, 3, 4}
X = {(" PMP_in" , v) | v∈X_{PMP_in}}
OutPorts = {"PMP_out", "Busy1"}, where YPMP_out = {1}, YBusy1 = {-1, 1}
Y = {(p,v)|p∈OutPorts, v∈Yp }
D = {PowderRoom, PressRoom, BlisterPack}
$\mathbf{M}_{PMP}$ = {**M**PowderRoom, **M**PressRoom, **M**BlisterPack}
EIC = {(PMP, "PMP_in"), (PowderRoom, "PMP_in")}
EOC = {((PowderRoom, "Busy1"), (PMP, "Busy1")),
((PackRoom, "PMP_out"), (PMP, "PMP_out"))}
IC = {((PowderRoom, "Press"), (PressRoom, "Press")),
((PressRoom, "Blister"), (BlisterPack, "Blister"))
Select: ({BlisterPack, PressRoom}) = PackRoom
({BlisterPack, PowderRoom}) = BlisterPack
({PressRoom, PowderRoom}) = PressRoom

Testing Strategy for PMP

1. Verify that when a value of 1 is passed to PMP through the port PMP_in there is an output through the port PMP_out after the pre-defined time of 20 hours.
2. Verify that when a value of 1 is passed to PMP through the port PMP_in there is an output through the port BUSY1 simultaneously.

Factory = < X, Y, D, {Md | d∈D}, EIC, EOC, IC, Select>
InPorts = {"RawMaterials2", "OrderInfo2"}, where XRawMaterials2 = {1, 2, 3, … , 78, 79, 80}, XOrderInfo2 = {1, 2, 3, … , 18, 19, 20}
X = {(p, v) | v∈XPMP_in}
OutPorts = { "FinishedPro1"}, where YFinishedPro1 = {1}
Y = {(p,v)|p∈OutPorts, v∈Yp }
D = {FactoryWarehouse, FactoryAdministrator, PMP}
MFactory = {MFactoryWarehouse, MFactoryAdministrator, MPMP}
EIC = {(Factory, "RawMaterials2"), (FactoryAdministrator, "RawMaterials2")
((Factory, "OrderInfo2"), (FactoryAdministrator, "OrderInfo2"))}
EOC = {((FactoryAdministrator, "FinishedPro1"), (Factory, "FinishedPro1"))}
IC = {((FactoryAdministrator, "Send"), (FactoryWarehouse, "Send")),
((FactoryAdministrator, "FacWar_in1"), (FactoryWarehouse, "FacWar_in1")),
((FactoryAdministrator, "FacWar_in2"), (FactoryWarehouse, "FacWar_in2")),
((FactoryWarehouse, "FacWar_out1"), (FactoryAdministrator, "FacWar_out1")),
((FactoryWarehouse, "FacWar_out2"), (FactoryAdministrator, "FacWar_out2")),
((PMP, "PMP_out"), (FactoryAdministrator, "PMP_out"))
((PMP, "Busy1"), (FactoryAdministrator, "Busy1"))
((FactoryAdministrator, "PMP_in"), (PMP, "PMP_in"))}
Select: ({PMP, FactoryAdministrator }) = PMP
({PMP, FactoryWarehouse }) = PMP
({FactoryAdministrator, FactoryWarehouse }) = FactoryAdministrator

Testing Strategy for Factory

1. Each atomic model (administrator, warehouse, powder room, press room and blister pack) was tested and verified individually according to the test strategies defined above.
2. The coupled model (PMP) was also tested and verified separately.
3. The following test conditions were used to test and verify the complete factory model:
1. Send an input of value 10 through the port OrderInfo2 at 00:00:01:01 and send an input of 20 through the port RawMaterials2 at 00:00:02:01
2. Send an input of value 22 through the port OrderInfo2 at 00:00:01:01 and send an input of 19 through the port OrderInfo2 at 00:00:02:01
3. Send an input of value 50 through the port RawMaterials2 at 00:00:01:01 and send an input of 20 through the port RawMaterials2 at 00:00:02:01

Note: All the above test conditions the number of finished products in the warehouse initially was set to 19 and the number of raw materials in the warehouse was set to 50. In addition, initially the pharmaceutical plant (PMP) was not processing anything in any of its components.

V. Testing, Simulation Analysis and Animation

Both the atomic and coupled models were all implemented and tested using the toolkit CD++. The atomic models (administrator, warehouse, powder room, press room, blister pack) were implemented and tested independently before there were incorporated into the coupled models. The PMP coupled model was also implemented and tested separately before it was integrated into the factory coupled model. Although numerous tests were performed to ensure all models were behaved as desired, only a selected three test cases are outlined below.

Note that to better understand the test cases; the block diagram was modified as seen in Figure 7. This not only ensures that the outputs of the factory coupled model are outputted, but also shows some processing of the second level models (administrator, warehouse and PMP). In addition for all the test conditions outlined below, the number of finished products in the warehouse initially was set to 19 and the number of raw materials in the warehouse was set to 50. Furthermore, initially the pharmaceutical plant (PMP) was not processing anything in any of its components.

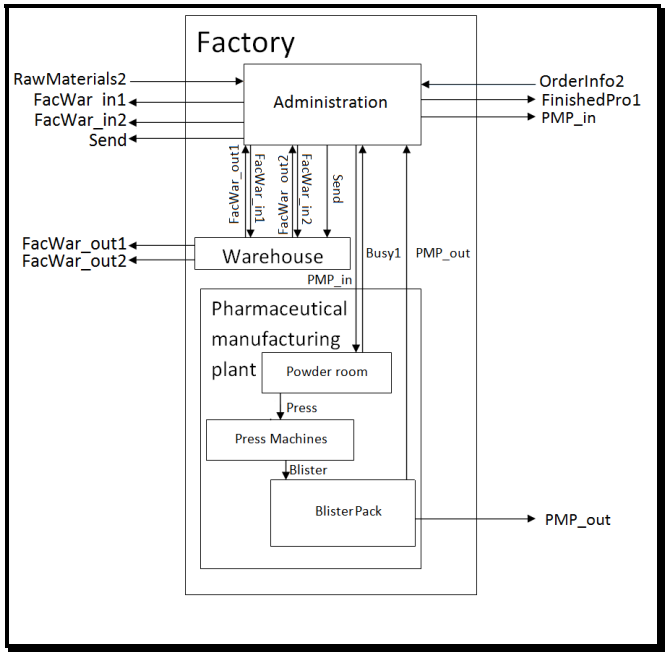

Figure 7: Modified block diagram of Factory model for testing purposes

A. Test Case 1: Valid Inputs

Table 1: Input (event file) and output (out file) displayed in time sequence for case 1 - valid inputs

The input and output was run for 99:00:00:00. The simulation shows that for 'proper' inputs (i.e. the order placed for finished products from the distributor is between 0 and 20, and is less than the current number of products that is held in the factory's warehouse. In addition the number of raw materials received from the supplier is less than or equal to the available amount of space in the factory's warehouse for raw materials, and the input is between the values of 0 and 80) there were no error messages displayed, and the simulation ran till the specified time.

This simulation also shows that factory is trying to replenish its finished products stock from the beginning (00:00:00:000) since the amount of finished products stored in its warehouse was not at full capacity. In addition the input and output times for PMP were verified to be correct. Lastly, the time at which various values were outputted from the administrator and warehouse through the ports facwar_in1, facwar_in2, facwar_out1, facwar_out2 all occurred at desired times. Thus, it is verified that for correct inputs, the entire factory model behaves as specified.

Table 1: Input (event file) and output (out file) displayed in time sequence for case 1 - valid inputs

Inputs	Outputs
00:00:01:01 OrderInfo2 10	00:00:00:000 send 104
	00:00:00:000 facwar_out1 4
	01:00:00:050 pmp_in 4
	00:00:01:090 send 190
	00:00:01:090 facwar_out2 10
	00:00:01:137 out 10
00:00:02:01 RawMaterials2 20	00:00:02:062 facwar_in1 120
	09:00:00:081 send 104
	09:00:00:081 facwar_out1 4
	10:00:00:131 pmp_in 4
	15:00:00:089 facwar_in2 181
	18:00:00:162 send 104
	18:00:00:162 facwar_out1 4
	19:00:00:212 pmp_in 4
	24:00:00:170 facwar_in2 181
	27:00:00:243 send 104
	27:00:00:243 facwar_out1 4
	28:00:00:293 pmp_in 4
	33:00:00:251 facwar_in2 181
	36:00:00:324 send 104
	36:00:00:324 facwar_out1 4
	37:00:00:374 pmp_in 4
	42:00:00:332 facwar_in2 181
	45:00:00:405 send 104
	45:00:00:405 facwar_out1 4
	46:00:00:455 pmp_in 4
	51:00:00:413 facwar_in2 181
	54:00:00:486 send 104
	54:00:00:486 facwar_out1 4
	55:00:00:536 pmp_in 4
	60:00:00:494 facwar_in2 181
	63:00:00:567 send 104
	63:00:00:567 facwar_out1 4
	64:00:00:617 pmp_in 4
	69:00:00:575 facwar_in2 181
	72:00:00:648 send 104
	72:00:00:648 facwar_out1 4
	73:00:00:698 pmp_in 4
	78:00:00:656 facwar_in2 181
	81:00:00:729 send 104
	81:00:00:729 facwar_out1 4
	82:00:00:779 pmp_in 4
	87:00:00:737 facwar_in2 181
	90:00:00:810 send 104
	90:00:00:810 facwar_out1 4
	91:00:00:860 pmp_in 4
	96:00:00:818 facwar_in2 181

B. Test Case 2: Invalid and Valid values for OrderInfo2

Table 2: Input (event file) and output (out file) displayed in time sequence for case 2

Inputs	Outputs
	00:00:00:000 send 104
	00:00:00:000 facwar_out1 4
00:00:01:01 OrderInfo2 22	
	01:00:00:050 pmp_in 4
00:00:02:01 OrderInfo2 19	
	00:00:02:090 send 199
	00:00:02:090 facwar_out2 19
	00:00:02:137 out 19
	09:00:00:081 send 104
	09:00:00:081 facwar_out1 4
	10:00:00:131 pmp_in 4
	15:00:00:089 facwar_in2 181
	18:00:00:162 send 104
	18:00:00:162 facwar_out1 4
	19:00:00:212 pmp_in 4
	24:00:00:170 facwar_in2 181
	27:00:00:243 send 104
	27:00:00:243 facwar_out1 4
	28:00:00:293 pmp_in 4
	33:00:00:251 facwar_in2 181
	36:00:00:324 send 104
	36:00:00:324 facwar_out1 4
	37:00:00:374 pmp_in 4
	42:00:00:332 facwar_in2 181
	45:00:00:405 send 104
	45:00:00:405 facwar_out1 4
	46:00:00:455 pmp_in 4
	51:00:00:413 facwar_in2 181
	54:00:00:486 send 104
	54:00:00:486 facwar_out1 4
	55:00:00:536 pmp_in 4
	60:00:00:494 facwar_in2 181
	63:00:00:567 send 104
	63:00:00:567 facwar_out1 4
	64:00:00:617 pmp_in 4
	69:00:00:575 facwar_in2 181
	72:00:00:648 send 104
	72:00:00:648 facwar_out1 4
	73:00:00:698 pmp_in 4
	78:00:00:656 facwar_in2 181
	81:00:00:729 send 104
	81:00:00:729 facwar_out1 4
	82:00:00:779 pmp_in 4
	87:00:00:737 facwar_in2 181
	90:00:00:810 send 104
	90:00:00:810 facwar_out1 4
	91:00:00:860 pmp_in 4
	96:00:00:818 facwar_in2 181

The above input and output was run for 99:00:00:000. This simulation shows how the factory model handles error. Note that the first input was an improper input – i.e. the order placed by the distributor was more than 20 finished products. This is invalid because the model specifications dictate that no order exceeding 20 can be placed. However, the second order shows that for a proper input the simulation runs for specified amount of time.

This simulation also shows that factory is trying to replenish its finished products stock from the beginning (00:00:00:000) since the amount of finished products stored in its warehouse was not at full capacity. In addition the input and output times for PMP were verified to be correct. Lastly, the time at which various values were outputted from the administrator and warehouse through the ports facwar_in1, facwar_in2, facwar_out1, facwar_out2 all occurred at desired times. Thus, it is verified that for correct inputs, the entire factory model behaves as specified. In the case 2, we receive the following error in CD++ console view:

"Error: Factory Administrator asked to send more than 20 finished products at a time!"

This shows that the factory model does handle error as specified by above.

C. Test Case 3: Invalid and Valid values for RawMaterials2

Inputs	Outputs
	00:00:00:000 send 104
	00:00:00:000 facwar_out1 4
00:00:01:01 RawMaterials2 50	
00:00:02:01 RawMaterials2 20	
	01:00:00:050 pmp_in 4
	00:00:02:062 facwar_in1 120
	15:00:00:089 facwar_in2 181

The above input and output was run for 99:00:00:000. This simulation shows how the factory model handles error. Note that the first input was an improper input – i.e. the amount of raw materials received by the factory from the supplier exceeded the factory's warehouse capacity. This is invalid because the model specifications dictate that the factory will not accept a shipment of raw materials unless it has the capacity to store the entire shipment. However, the second order shows that for a proper input the simulation runs for specified amount of time.

This simulation also shows that factory is trying to replenish it's finished products stock from the beginning (00:00:00:000) since the amount of finished products stored in it's warehouse was not at full capacity. In addition the input and output times for PMP were verified to be correct. Lastly, the time at which various values were outputted from the administrator and warehouse through the ports facwar_in1, facwar_in2, facwar_out1, facwar_out2 all occurred at desired times. Thus, it is verified that for correct inputs, the entire factory model behaves as specified. In addition unlike the previous two test cases, the factory administrator only sent one batch of products to be manufactured. This is because no orders from the distributor were placed and the current number of finished products held in the warehouse is 19 units.

D. Animation

Animation of coupled and atomic models is another capability of CD++ toolkit. By animating the models, a simple graphical notation is extracted from the log files .Therefore we can see the way data are transferred through the ports and delayed on different states of the model. This tool shows a state-base representation of the model. One snapshot of simulating the warehouse of the factory model is brought in Figure 8 .As seen; the picture shows the possible states of the model in

which there are no values inputted the model through different port.

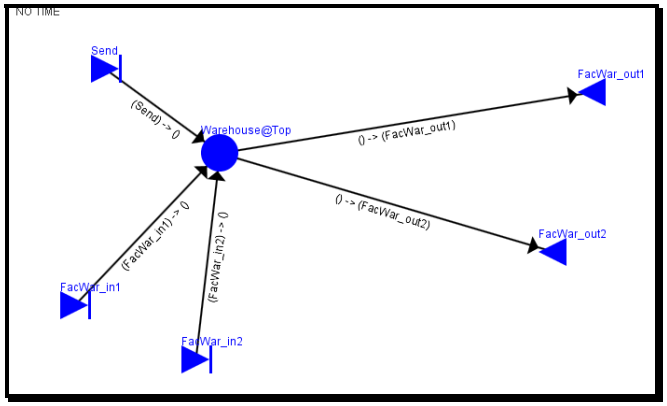

Figure 8: A snapshot of Animation of Warehouse model using CD++ toolkit

A snapshot of the animation of PMP model is depicted in Figure 9.

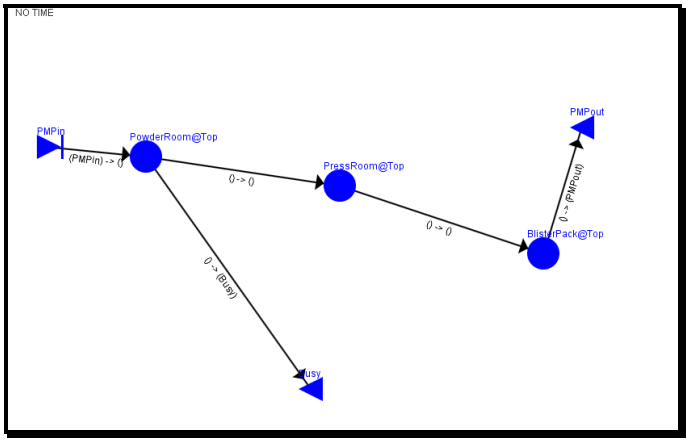

Figure 9: A snapshot of Animation of PMP model using CD++ toolkit

VI. Conclusion

A simulation of a pharmaceutical plant which is a part of a supply chain of a pharmaceutical product is performed using a combination of C++ and DEVS-graph models in CD++ toolkit. This was done following an initial modeling and simulation using pure C++ standard models in CD++ toolkit. It is asserted that using the graphical methods is much simpler and straightforward. One of the advantages of employing graphical interface of CD++ toolkit is that the model is easily designed, managed and modified. In addition, in a typical model, less variable are required to be declared due to the tools auto generation of the states' specifications. Employing this tool however may sacrifice some complexities of the model. That is because the graphical interface lacks some detailed definition of internal and external transition functions. Therefore, the best option is to use a combination of both kind of models to benefit from both simplicity in modeling and complexity of design.

Acknowledgment

S. Jafartayari thanks professor G. Wainer for his support and Cheryl de Souza for her help on the initial programming of this project in C++.

References

[1] James A. Tompkins, "The Challenge of Warehousing," in The Warehouse Management Handbook, James A. Tompkins and Jerry D. Smith, Eds. Raleigh, North Carolina, United States of America: Tompkins Press, 1998, ch. 1, p. 2.

[2] G. Christen, A. Dobniewski, G. A. Wainer, "Modeling State-Based DEVS Models in CD++," Proceedings of MGA, Advanced Simulation Technologies Conference (ASTC04), Arlington, VA. U.S.A - 2004

[3] Gabriel A. Wainer, "Introduction to the DEVS Modeling and Simulation Formalism," in Discrete-event modeling and simulation: a practitioner's approach. Boca Raton, United States of America: CRC Press, 2009, ch. 2, pp. 35-54.