**CARLETON UNIVERSITY**

DEPARTMENT OF SYSTEMS AND COMPUTER ENGINEERING



**METHODOLOGIES FOR DISCRETE EVENT MODELLING AND SIMULATION**

**Modeling Discrete-Event Systems Using DEVS (2019 Fall)**

**ASSIGNMENT1: Cadmium implementation of “Offline File Transfer Simulator”**

SUBMITTED BY:

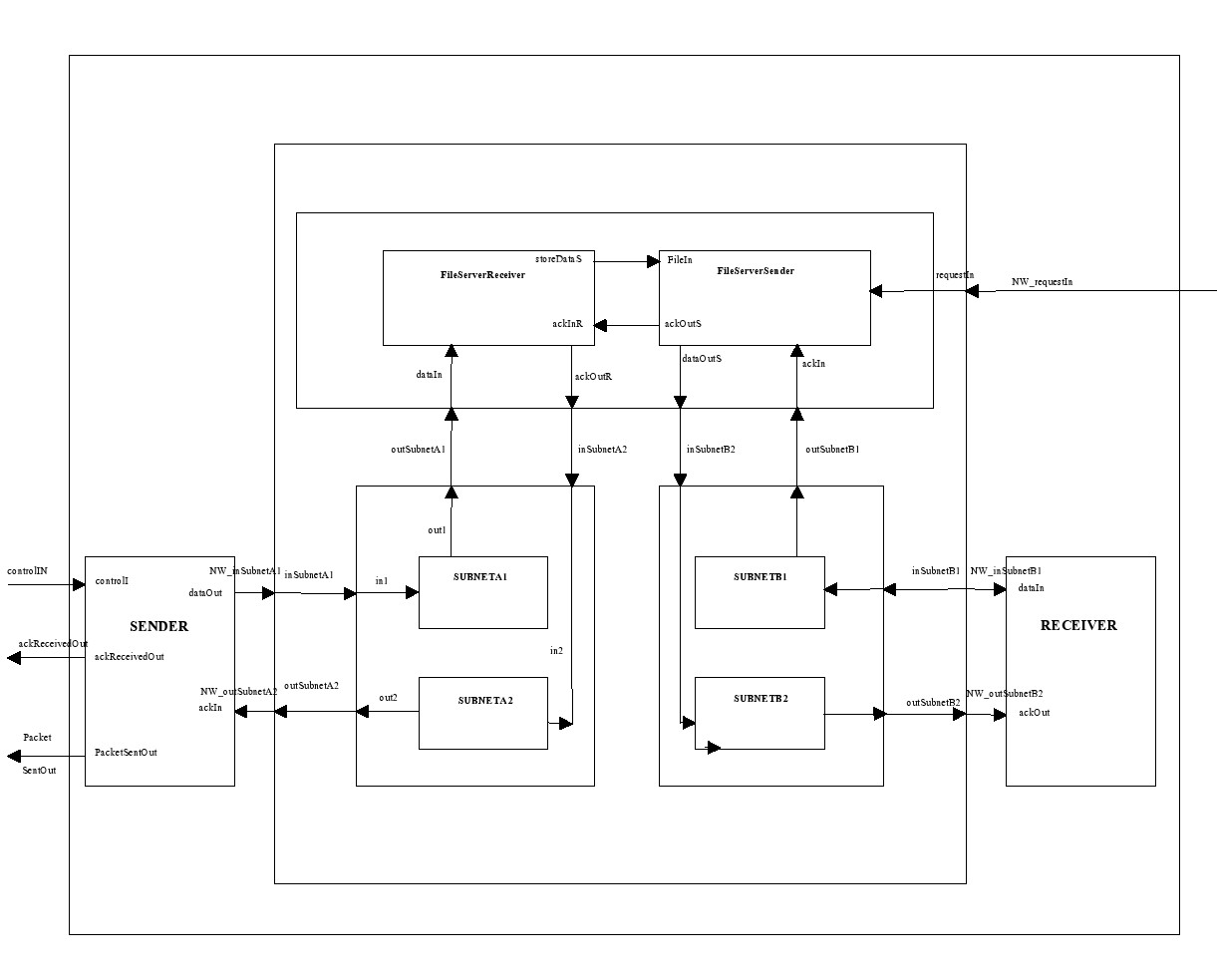
Amitav Shaw 101152226

Arshpreet Singh 101138246

**PART I**

We undertake the model: “Offline File Transfer Simulator” submitted in Fall-2012 by Weiwei Li and Yuhan Zeng from the University of Ottawa and port the model from CD++ to Cadmium. As we have inferred, this model is an asynchronous file transfer method where receiver can be offline and download the file from the server once it is online. Our understanding of the model is presented henceforth.

Offline file transfer supports asynchronous file transfer between the users. Let us consider that the sender has to share a file with a receiver but the receiver is offline. Had it been the synchronous file transfer protocol, sender would have had to wait until the receiver comes online so an alternate methodology of offline file transfer is put forth. The main idea behind this methodology is that the sender will send the files to a temporary file server and the receiver can download the files from the temporary file server once it comes online. To increase the utilization of file server, the files will be deleted after successful delivery. This step encourages capacity for upcoming file transfers.

The simulator consists of 3 components: Sender, Network and Receiver.

The Network can be divided into three parts: file server exchange and two access Networks i.e. Access Network A and Access Network B. File server exchange is further comprised of two modules: file server sender and file server receiver.

In addition, there are 2 subnets each of the access Networks: subnetA1 and subnet A2 under Access Network A whereas subnetB1 and subnetB2 under Access Network B. Each subnet has a sending and receiving component.

In this simulator, the ABR module presented in simulation sample is reused. The advantage of using ABR module is; the assurance of reliable file transfer between the users and the file server exchange. The sender sends a file to the receiving module of the file server exchange through subnetA1 and awaits acknowledgement. If the sender doesn’t receive an acknowledgement within a prescribed time, it resends the file unless it receives the acknowledgement before going passive.

On the other side, receiver receives the file from the sending module of the file server exchange through subnetB2. The protocol used by the sending module is the same as used by the receiving module mentioned above to ensure reliable file transfer.

File server exchange has two behaviors according to its modules. The file server receiver module receives the file from sender through subnetA1 and transfers it to the file server sender. Then, file server sender writes the file into the file system. The file server sender module receives a requestIn signal which means receiver is activated and starts to send the files stored in the file system to receiver through subnetB2. The files will be deleted after successful delivery.

**PART II → Description of the atomic and coupled DEVS model**

Offline File Transfer simulator has two inputs and two outputs as depicted in Figure 1. The input controlIn is for the number of files (packets) to be transferred from sender to the FileSeverExchange through Access Network A whereas the input requestIn implies that the receiver is ready to download the files. The outputs indicate an expected acknowledgement that has been received (ackReceived) and total number of files that have been sent (fileSent).

Offline File Transfer simulator coupled model has three components: sender, Network and receiver. Network coupled model can be further divided into three coupled model which are FileServerExchange, Access Network A, Access Network B. Sender sends files to FileServerExchange through Access Network A and receiver receives the files through Access Network B.

Receiver may request for the files from FileServerExchange coupled model via Access Network B either during or after the sender sends the file. FileServerExchange sends the files to the receiver and deletes the files one by one once receiver has successfully received the files. The FileServerExchange coupled model can also be further decomposed into two components; FileServerReceiver and FileServerSender. The files from sender are received by the FileServerReceiver and then send the files to FileServerSender. The FileServerSender will write the files into File System (which is emulated by a queue).

As soon as a request is received from the receiver, FileServerSender sends the files and delete them after successful delivery. The read and write thread in the CD++ implementation have been integrated in the FileServerSender and FileServerReceiver respectively. In addition, the access Networks defined earlier namely Access Network A and Access Network B have same structure and are further decomposed into two subnets each. The subnets have the same behaviors and use the same atomic model called subnet.

Offline File Transfer simulator has five atomic models: sender, receiver, subnet, FileServerSender and FileServerReceiver. Three of the atomic models namely sender, receiver and subnet are reused models written by Christina Ruiz. Some modifications have been done to these models for this particular implementation. The same information was posted into the header of the files that were modified.

In the Cadmium implementation of the model, we had to figure out the links between the coupled models and inputs and outputs from the atomic as well as the coupled model. Functionality of each atomic model is described in the .HPP files. The important functions and features according to DEVS specification are programmed in the atomic models: State, internal function, external function, time advance and output function.

**ATOMIC MODEL**:

The formal specification for the atomic models is given by the following expression:

Atomic DEVS = <S, X, Y, δint, δext, λ, ta >All the atomic models have to implement the internal function, external function and output function. The entire message passing is done using Message\_t data type which has a packet and the bit information.

Below we provide the basic functionality of each atomic model:

**Sender**

S= {passive, active};

X= {controlIn, ackIn};

Y= {packetSentOut, ackReceivedOut} // The ack received from network and number of packets sent are sent out through these out ports.

δint (active)= idle; // The internal transition takes care of alternating the bits in the next packet. For our implementation we send the data and go passive till ack doesn’t come from receiver. Here we implemented an asynchronous model in which sender will receive an ack only when there’s a download request from the receiver.

if(false==state.sending)

state.next\_internal = std::numeric\_limits<TIME>::infinity();

δext(in, passive)= active; // transitions into state for sending the output for controlIn number of times. It checks for ack each time and compares it with the bit field of the packet just sent. If they are equal it sends the next packet otherwise it repeats the same packet.

Code Snippet:

for(const auto &x : get\_messages<typename Sender\_defs::controlIn>(mbs)){

if(state.model\_active == false){

state.totalPacketNum = x;

if (state.totalPacketNum > 0){

state.packetNum = 1;

state.ack = false;

state.sending = true;

state.alt\_bit = 0; //set initial alt\_bit

state.model\_active = true;

state.next\_internal = preparationTime;

}else{

if(state.next\_internal != std::numeric\_limits<TIME>::infinity()){

state.next\_internal = state.next\_internal - e;

}

}

}

for(const auto &x : get\_messages<typename Sender\_defs::ackIn>(mbs)){

if(state.model\_active == true) {

if (state.alt\_bit == x.bit) {

state.ack = true;

state.sending = false;

state.next\_internal = TIME("00:00:00");

}else{

if(state.next\_internal != std::numeric\_limits<TIME>::infinity()){

state.next\_internal = state.next\_internal - e;

}

}

}

λ (active) // Outputs controlIn number of packets with alternating bit for each subsequent packets.

Code Snippet:

if (state.sending){

out.packet = state.packetNum;

out.bit = state.alt\_bit;

get\_messages<typename Sender\_defs::dataOut>(bags).push\_back(out);

get\_messages<typename Sender\_defs::packetSentOut>(bags).push\_back(state.packetNum);

}else{

if(state.ack){get\_messages<typename Sender\_defs::ackReceivedOut>(bags).push\_back(state.alt\_bit);

}

}

ta (idle)= INFINITY;

ta (active) = 10ms;

The sender sends the controlIn number of packets to the Network. When there is an input in the controlIn port the external function is called which loops through controlIn number of times. In this function it sets the state variables such that there is an output. Then, the internal function is called and it sets the alternate bit each time it is invoked.

**Subnet**

Subnet works like pipe in this model where it takes input at one end and sends out at the other end. In this model there are two access Networks using two subnets in each for sending and receiving the packets.

S= {passive, active};

X= {in};

Y= {out};

δint (active)= idle;

δext(in,passive)= active; // state transition for transmitting the data received at in port. It throttles the data inflow and just allows one packet at a time.

Code Snippet:

if(bag\_port\_in.size()>1) assert(false && "One message at a time");

state.index ++;

state.packet = bag\_port\_in[0];

state.transmitting = true;

λ (active) //In ABP implementation data is sent with 95% probability. But, for this implementation we assume no data loss at Network.

Code Snippet:

typename make\_message\_bags<output\_ports>::type bags;

vector<Message\_t> bag\_port\_out;

get\_messages<typename Subnet\_defs::out>(bags).push\_back(state.packet);

return bags;

ta (passive)= INFINITY;

ta (active)= 3 secs;

**File Server Sender**

FileServerSender maintains a database which has been implemented using a queue data structure. It receives packet from FileServerReceiver and stores it into the queue. This is the same packet that the Sender sent it to FileServerReceiver through Network A. All the packets are queued until a request doesn’t come in. If there is a request for data download through requestIn port, the FileServerSender takes out the front element from the queue and sends it to the receiver through Network B. The receiver after receiving the file sends an ack to the FileServerSender through the Network B.

State Variables:

* state.transmitFile
* state.index
* state.ack

S= {passive, active};

X= {requestIn, dataIn, ackIn};

Y= {ackReceivedOut, dataOut};

δint (transmitFile, ack) // These are boolean values which are set to false in the internal function. On the basis of these states the time\_advance decides the next state. Code snippet given below:

state.transmitFile = false;

state.request = 0;

state.ack = false;

δext(state.transmitFile)= true; //Whenever there is data at input the FileServerSender segregates the inputs and based on the different inputs it either transitions to the state to tranfer the data to Receiver or ack to FileServerReceiver

Code snippet for external function:

if(!bag\_port\_FileIn.empty())

tmpFile.push(bag\_port\_FileIn[0]);

if (!bag\_port\_ackIn.empty()) {

state.ack=true;

state.packet\_ack = bag\_port\_ackIn[0];

}

if (!bag\_port\_requestIn.empty()){

if(!tmpFile.empty()){

state.request = bag\_port\_requestIn[0];

if(state.request){

state.packet = tmpFile.front();

tmpFile.pop();

state.transmitFile = true;

}

}

}

λ (active) //Sends data on the output ports according to the state variables set in the external function.

Code snippet:

if(state.transmitFile){

get\_messages<typename FileServerSender\_defs::dataOutS>(bags).push\_back(state.packet);

}else if(state.ack){

get\_messages<typename FileServerSender\_defs::ackOutS>(bags).push\_back(state.packet\_ack);

}

ta (passive)= INFINITY;

ta(active) = 3 secs.

**File Server Receiver**

State Variables:

* state.transmitFile
* state.index
* state.ack

S= {passive, active};

X= {dataIn, ackInR};

Y= {ackOutR, storeDataS};

δint (transmitFile, ack) // These are boolean values which are set to false in the internal function. On the basis of these states the time\_advance decides the next state.

state.transmitFile = false;

state.ack = false;

δext(in,passive)= active; // In the external function the states are transitioned to the transfer either data or ack.

Code snippet:

if (!bag\_port\_dataIn.empty()) {

state.packet = bag\_port\_dataIn[0];

state.transmitFile = true;

} else if(!bag\_port\_ackInR.empty()){

state.ackInR = bag\_port\_ackInR[0];

state.ack = true;

}

ta (passive)= INFINITY;

ta(active) = 3 secs.

**Receiver**

State Variables:

S= {passive, receiving};

X= {in};

Y= {out};

δint (sending) = false

δext(in,passive)= active; //The receiver just waits for the data and sends an ack when it receives one. It extracts the bit field of the message it received from network and sends sets the state for transfer after ta.

message\_port\_in = get\_messages<typename Receiver\_defs::in>(mbs);

state.ackNum = message\_port\_in[0].bit;

state.sending = true;

**Coupled Model**

The formal specification for the coupled models is as follows:

CM=<X, Y, D, {}, I, Z, select>

In our model we have the following coupled modes:

**Network:**

X= {NW\_inSubnetA1, NW\_inSubnetB1, requestIn};

Y= {NW\_outSubnetA2, NW\_outSubnetB2};

D= {AccessNetworkA, AccessNetworkB, FileExchangeServer};

I (AccessNetworkA) =FileExchangeServer;

I (FileExchangeServer) = AccessNetworkB;

I (AccessNetworkB) = self;

Z (AccessNetworkA) =FileExchangeServer;

Z (FileExchangeServer) = AccessNetworkB;

Z (AccessNetworkB) = self;

SELECT:

({AccessNetworkA, AccessNetworkB, FileExchangeServer}) = AccessNetworkA;

({AccessNetworkB, FileExchangeServer}) = FileExchangeServer;

**AccessNetworkA**

X= {inSubnetA1, inSubnetA2};

Y= {outSubnetA1, outSubnetA1};

D= {subnetA1, subnetA2};

I (inSubnetA1) =self;

I (inSubnetA2) =self;

Z (outSubnetA1) =self;

Z (outSubnetA2) =self;

SELECT:

({subnetA1, subnetA2})= subnetA1

**AccessNetworkB**

X= {inSubnetB1, inSubnetB2};

Y= {outSubnetB1, outSubnetB1};

D= {subnetB1, subnetB2};

I (inSubnetB1) =self;

I (inSubnetB2) =self;

Z (outSubnetB1) =self;

Z (outSubnetB2) =self;

**FileExchangeServer:**

X= {FSReceiverDataIn, FSSenderAckIn, requestIn};

Y= {FSReceiverAckOut, FSSenderDataOut};

D= {fileserverreceiver, fileserversender };

I (fileserverreceiver) = fileserversender;

I (fileserversender) = (fileserversender)

**OfflineFileTransfer simulator**

X= {controlIn, requestIn};

Y= {packetSent, ackReceived};

D= {Sender, Network, Receiver};

I (Sender) = Network;

I (Network) =Receiver;

I (Receiver) = Network;

I (Network) =Sender;

Z (Sender) = Network;

Z (Network) =receiver;

Z (receiver) = Network;

Z (Network) =sender;

SELECT:

({Sender,Network,Receiver})=Sender;

({ Network,Receiver })=Network;

## PART III → Test Strategies

**Testing the atomic models (Blackbox Testing)**

Each atomic model can be tested individually to verify their functionality. This way we can integrate the models into the coupled models without much debug effort thereafter. Each atomic model should pass the black box test and the outputs should be according to the functionality of the atomic model for a given set of inputs.We have created sample inputs for each atomic model to check if the outputs are according to the expectation. For this model we have created input reader model instances to feed the input to the input ports of the atomic models.

**Receiver**

The receiver receives data of type Message\_t in the IN port and sends out the acknowledgment with the same bit value as the packet received

**Input:**

00:00:10 1 1

00:00:30 2 0

00:00:45 3 1

00:00:52 3 1

00:01:25 4 0

00:01:35 4 0

00:01:55 5 1

**The expected result**

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {}] generated by model input\_reader

00:00:10:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {1 1}] generated by model input\_reader

00:00:20:000

[Receiver\_defs::out: {0 1}] generated by model receiver1

00:00:30:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {2 0}] generated by model input\_reader

00:00:40:000

[Receiver\_defs::out: {0 0}] generated by model receiver1

00:00:45:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {3 1}] generated by model input\_reader

00:00:52:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {3 1}] generated by model input\_reader

00:01:02:000

[Receiver\_defs::out: {0 1}] generated by model receiver1

00:01:25:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {4 0}] generated by model input\_reader

00:01:35:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {4 0}] generated by model input\_reader

[Receiver\_defs::out: {0 0}] generated by model receiver1

00:01:45:000

[Receiver\_defs::out: {0 0}] generated by model receiver1

00:01:55:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {5 1}] generated by model input\_reader

00:02:05:000

[Receiver\_defs::out: {0 1}] generated by model receiver1

### FileServerSender

FileServerSender handles most of the logic of the file transfer. It keeps on queuing the packets received and sends the packet to its out port only when there is a request to it.

We have taken input\_reader atomic model to infuse the inputs into the input ports of the FileServerSender. Based on the requestIn we see the out port transmitting the same data as the data that comes from FileServerReceiver. The file doesn’t go out to output port till the requestIn doesn’t come in. Here to mimic the request, the value of RequestIn should be a positive integer and zero in case of no request.

We take three input streams to the three in ports of this model to test the output. For the case where we get positive RequestIn values we see that the data is sent to the receiver:

**Test Scenario 1**

**RequestIn:**

00:00:10 5

00:01:30 1

00:02:00 1

00:02:30 5

00:03:00 1

00:03:30 1

00:04:00 5

00:40:30 1

**DataIn:**

00:00:10 1 0

00:00:20 5 1

00:00:30 6 0

00:00:40 9 1

**AckIn:**

00:00:10 10 0

00:00:20 50 1

00:00:30 60 0

00:00:40 90 1

**Output:**

00:00:00:000

00:00:00:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {}] generated by model input\_reader\_ReqIn

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {}] generated by model input\_reader\_dataIn

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {}] generated by model input\_reader\_ackIn

00:00:10:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {5}] generated by model input\_reader\_ReqIn

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {1 0}] generated by model input\_reader\_dataIn

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {10 0}] generated by model input\_reader\_ackIn

00:00:13:000

[FileServerSender\_defs::ackOutS: {}, FileServerSender\_defs::dataOutS: {1 0}] generated by model FileServerSender1

00:00:20:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {5 1}] generated by model input\_reader\_dataIn

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {50 1}] generated by model input\_reader\_ackIn

00:00:23:000

[FileServerSender\_defs::ackOutS: {50 1}, FileServerSender\_defs::dataOutS: {}] generated by model FileServerSender1

00:00:30:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {1}] generated by model input\_reader\_ReqIn

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {6 0}] generated by model input\_reader\_dataIn

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {60 0}] generated by model input\_reader\_ackIn

00:00:33:000

[FileServerSender\_defs::ackOutS: {}, FileServerSender\_defs::dataOutS: {5 1}] generated by model FileServerSender1

00:00:40:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {9 1}] generated by model input\_reader\_dataIn

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {90 1}] generated by model input\_reader\_ackIn

00:00:43:000

[FileServerSender\_defs::ackOutS: {90 1}, FileServerSender\_defs::dataOutS: {}] generated by model FileServerSender1

00:00:50:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {1}] generated by model input\_reader\_ReqIn

00:00:53:000

[FileServerSender\_defs::ackOutS: {}, FileServerSender\_defs::dataOutS: {6 0}] generated by model FileServerSender1

00:01:30:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {5}] generated by model input\_reader\_ReqIn

00:01:33:000

[FileServerSender\_defs::ackOutS: {}, FileServerSender\_defs::dataOutS: {9 1}] generated by model FileServerSender1

**TEST SCENARIO 2**

For testing the case when requestIn is not present we mimic the scenario by putting all zero values for requestIn and observe that no packet is sent to receiver, that is to say no files are downloaded.

**RequestIn:**

00:00:10 0

00:01:30 0

00:02:00 0

00:02:30 0

00:03:00 0

00:03:30 0

00:04:00 0

00:40:30 0

**DataIn**

00:00:10 1 0

00:00:20 5 1

00:00:30 6 0

00:00:40 9 1

**AckIn**

00:00:10 10 0

00:00:20 50 1

00:00:30 60 0

00:00:40 90 1

**Output //** We see that “transmittingToReceiver” doesn’t transmit any data that comes in through input\_reader\_dataIn port because there’s no request for download (RequestIn = 0)

00:00:00:000

State for model input\_reader\_ReqIn is next time: 00:00:00:000

State for model input\_reader\_dataIn is next time: 00:00:00:000

State for model input\_reader\_ackIn is next time: 00:00:00:000

State for model FileServerSender1 is index: 0 & transmittingToReceiver: 0 0 & ackFromReceiver: 0 0 & requestIn: 0

00:00:00:000

State for model input\_reader\_ReqIn is next time: 00:00:10:000

State for model input\_reader\_dataIn is next time: 00:00:10:000

State for model input\_reader\_ackIn is next time: 00:00:10:000

State for model FileServerSender1 is index: 0 & transmittingToReceiver: 0 0 & ackFromReceiver: 0 0 & requestIn: 0

00:00:10:000

State for model input\_reader\_ReqIn is next time: 00:01:20:000

State for model input\_reader\_dataIn is next time: 00:00:10:000

State for model input\_reader\_ackIn is next time: 00:00:10:000

State for model FileServerSender1 is index: 1 & transmittingToReceiver: 0 0 & ackFromReceiver: 10 0 & requestIn: 0

00:00:13:000

State for model input\_reader\_ReqIn is next time: 00:01:20:000

State for model input\_reader\_dataIn is next time: 00:00:10:000

State for model input\_reader\_ackIn is next time: 00:00:10:000

State for model FileServerSender1 is index: 1 & transmittingToReceiver: 0 0 & ackFromReceiver: 10 0 & requestIn: 0

00:00:20:000

State for model input\_reader\_ReqIn is next time: 00:01:20:000

State for model input\_reader\_dataIn is next time: 00:00:10:000

State for model input\_reader\_ackIn is next time: 00:00:10:000

State for model FileServerSender1 is index: 2 & transmittingToReceiver: 0 0 & ackFromReceiver: 50 1 & requestIn: 0

00:00:23:000

State for model input\_reader\_ReqIn is next time: 00:01:20:000

State for model input\_reader\_dataIn is next time: 00:00:10:000

State for model input\_reader\_ackIn is next time: 00:00:10:000

State for model FileServerSender1 is index: 2 & transmittingToReceiver: 0 0 & ackFromReceiver: 50 1 & requestIn: 0

00:00:30:000

State for model input\_reader\_ReqIn is next time: 00:01:20:000

State for model input\_reader\_dataIn is next time: 00:00:10:000

State for model input\_reader\_ackIn is next time: 00:00:10:000

State for model FileServerSender1 is index: 3 & transmittingToReceiver: 0 0 & ackFromReceiver: 60 0 & requestIn: 0

00:00:33:000

State for model input\_reader\_ReqIn is next time: 00:01:20:000

State for model input\_reader\_dataIn is next time: 00:00:10:000

State for model input\_reader\_ackIn is next time: 00:00:10:000

State for model FileServerSender1 is index: 3 & transmittingToReceiver: 0 0 & ackFromReceiver: 60 0 & requestIn: 0

00:00:40:000

State for model input\_reader\_ReqIn is next time: 00:01:20:000

**Subnet**

Subnets behave as a pipe in the implementation. For this model we have considered only reliable Network. In future we need to test the scenario where there is packet drop in the subnet. As this is an asynchronous file transfer method, we need kept it simple to just get the functionality working. The test results are as follows

**Input:**

00:00:10 1 1

00:00:20 2 0

00:00:30 3 1

00:00:40 4 0

00:00:50 5 1

00:01:00 6 0

00:01:10 7 1

00:01:20 8 0

00:01:30 9 1

**Output:**

00:00:00:000

00:00:00:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {}] generated by model input\_reader

00:00:10:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {1 1}] generated by model input\_reader

00:00:13:000

[Subnet\_defs::out: {1 1}] generated by model subnet1

00:00:20:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {2 0}] generated by model input\_reader

00:00:23:000

[Subnet\_defs::out: {2 0}] generated by model subnet1

00:00:30:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {3 1}] generated by model input\_reader

00:00:33:000

[Subnet\_defs::out: {3 1}] generated by model subnet1

00:00:40:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {4 0}] generated by model input\_reader

00:00:43:000

[Subnet\_defs::out: {4 0}] generated by model subnet1

00:00:50:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {5 1}] generated by model input\_reader

00:00:53:000

[Subnet\_defs::out: {5 1}] generated by model subnet1

00:01:00:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {6 0}] generated by model input\_reader

00:01:03:000

[Subnet\_defs::out: {6 0}] generated by model subnet1

00:01:10:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {7 1}] generated by model input\_reader

00:01:13:000

[Subnet\_defs::out: {7 1}] generated by model subnet1

00:01:20:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {8 0}] generated by model input\_reader

00:01:23:000

[Subnet\_defs::out: {8 0}] generated by model subnet1

00:01:30:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {9 1}] generated by model input\_reader

00:01:33:000

[Subnet\_defs::out: {9 1}] generated by model subnet1

00:01:40:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {10 0}] generated by model input\_reader

### FileServerReceiver

File receiver receives the file from the Network and simply transfers it to the FileServerSender. It also receives the ack from FileServerReceiver which in turn is the ack received from the receiver.

**Inputs:**

**DataIn:**

00:00:10 10 0

00:00:20 50 1

00:00:30 60 0

00:00:40 90 1

**ackIn:**

00:00:10 0 0

00:00:20 0 1

00:00:30 0 0

00:00:40 0 1

**Output:**

00:00:00:000

00:00:00:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {}] generated by model input\_reader

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {}] generated by model input\_reader

00:00:10:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {10 0}] generated by model input\_reader

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {0 0}] generated by model input\_reader

00:00:13:000

[FileServerReceiver\_defs::ackOutR: {}, FileServerReceiver\_defs::storeDataS: {10 0}] generated by model FileServerReceiver1

00:00:20:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {50 1}] generated by model input\_reader

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {0 1}] generated by model input\_reader

00:00:23:000

[FileServerReceiver\_defs::ackOutR: {}, FileServerReceiver\_defs::storeDataS: {50 1}] generated by model FileServerReceiver1

00:00:30:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {60 0}] generated by model input\_reader

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {0 0}] generated by model input\_reader

00:00:33:000

[FileServerReceiver\_defs::ackOutR: {}, FileServerReceiver\_defs::storeDataS: {60 0}] generated by model FileServerReceiver1

00:00:40:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {90 1}] generated by model input\_reader

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {0 1}] generated by model input\_reader

00:00:43:000

[FileServerReceiver\_defs::ackOutR: {}, FileServerReceiver\_defs::storeDataS: {90 1}] generated by model FileServerReceiver1

### Sender

The sender takes one input controlIn and gives two outputs: ackReceivedOut and packetSent Out. It generates controlIn number of packets and sends it over the Network. It also waits for the ack from the Network. Here we input two input streams to check the functionality of Sender model. The first input of ControlIn is “-1” and second is “0”. The expected result is that no packet should be generated in these cases. In the results below we see that for controlIn as -1 or 0 there are no packets generated. However for subsequent positive integer inputs (5) there are packets sent out. The results are in line with expectation.

**TEST SCENARIO 1 and 2.**

**Inputs:**

**ControlIn:**

00:00:00 -1 – Negative test case.

00:00:05 0 - Negative test case.

00:00:15 5

00:02:50 3

**Sender\_input\_ack**

00:00:10 0

00:00:30 1

00:01:30 0

00:01:55 1

00:02:20 1

00:02:45 0

00:02:55 1

**Output: //** The output in red highlight indicates no packet generated because of negative request number.

00:00:00:000

00:00:00:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {}] generated by model input\_reader\_con

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {}] generated by model input\_reader\_ack

00:00:00:000

**[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {-1}] generated by model input\_reader\_con**

**00:00:05:000**

**[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {0}] generated by model input\_reader\_con**

00:00:10:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<Message\_t>::out: {0 0}] generated by model input\_reader\_ack

00:00:15:000

**[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {5}] generated by model input\_reader\_con**

**00:00:18:000**

**[Sender\_defs::packetSentOut: {1}, Sender\_defs::ackReceivedOut: {}, Sender\_defs::dataOut: {1 0}] generated by model sender1**

**00:00:24:000**

**[Sender\_defs::packetSentOut: {}, Sender\_defs::ackReceivedOut: {}, Sender\_defs::dataOut: {}] generated by model sender1**

**00:00:27:000**

**[Sender\_defs::packetSentOut: {1}, Sender\_defs::ackReceivedOut: {}, Sender\_defs::dataOut: {1 0}] generated by model sender1**

**00:00:33:000**

**[Sender\_defs::packetSentOut: {}, Sender\_defs::ackReceivedOut: {}, Sender\_defs::dataOut: {}] generated by model sender1**

00:00:36:000

[Sender\_defs::packetSentOut: {1}, Sender\_defs::ackReceivedOut: {}, Sender\_defs::dataOut: {1 0}] generated by model sender1

00:00:42:000

[Sender\_defs::packetSentOut: {}, Sender\_defs::ackReceivedOut: {}, Sender\_defs::dataOut: {}] generated by model sender1

00:00:45:000

[Sender\_defs::packetSentOut: {1}, Sender\_defs::ackReceivedOut: {}, Sender\_defs::dataOut: {1 0}] generated by model sender1

00:00:51:000

[Sender\_defs::packetSentOut: {}, Sender\_defs::ackReceivedOut: {}, Sender\_defs::dataOut: {}] generated by model sender1

00:00:54:000

**INTEGRATION TESTING:**

**Testing the Main Model**:

Here we have arranged to allow inputting readers to input to the controlIn and requestIn ports of this model. To run the main program we need to input two input files at the command line.

./OFFLINE\_FILE\_TRANSFER ../input\_data/controlIn.txt ../input\_data/requestIn.txt

Here the OFFLINE\_FILE\_TRANSFER is the executable and controlIn.txt and requestIn.txt are the two input files. The server downloads the packet only when there’s a non-zero requestIn value inputted. To interpret the output logs below i**nput\_reader** inputs into the controlIn and **input\_reader1** inputs into requestIn. **FileServerSender1** starts transmitting as soon as RequestIn comes. In one cycle we see that the data packet reaches the receiver and ack sent back.

A typical cycle of data transfer when request is positive is as follows:

Sender → inSubnetA1 → FileServerReceiver → FileServerSender → Receiver → outSubnetB1 → FileServerSender → FileServerReceiver → inSubnetA2 → sender

**TEST SCENARIO 1.**

**ControlIn:**

00:00:10 5

00:01:30 3

**RequestIn:**

00:00:10 5

00:01:30 1

00:02:00 1

00:02:30 5

.

**Output Snippets:**

**State for model FileServerSender1 is index: 3 & transmittingToReceiver: 1 0 & ackFromReceiver: 0 0 & requestIn: 1**

**State for model subnetA1 is index: 1 & transmitting: 1 0**

**.**

**.**

State for model FileServerSender1 is index: 4 & transmittingToReceiver: 0 0 & ackFromReceiver: 0 0 & requestIn: 0

State for model subnetA1 is index: 1 & transmitting: 1 0

State for model subnetA2 is index: 0 & transmitting: 0 0

State for model subnetB1 is index: 1 & transmitting: 0 0

State for model subnetB2 is index: 1 & transmitting: 1 0

00:01:55:000

State for model input\_reader is next time: inf

State for model input\_reader1 is next time: 00:00:30:000

State for model sender1 is packetNum: 1 & totalPacketNum: 5

State for model receiver1 is ackNum: 0

**State for model FileServerReceiver1 is index: 2 & transmitting: 1 0 & ackToSender: 0 0**

**State for model FileServerSender1 is index: 4 & transmittingToReceiver: 0 0 & ackFromReceiver: 0 0 & requestIn: 0**

State for model subnetA1 is index: 1 & transmitting: 1 0

State for model subnetA2 is index: 1 & transmitting: 0 0

State for model subnetB1 is index: 1 & transmitting: 0 0

State for model subnetB2 is index: 1 & transmitting: 1 0

00:01:58:000

State for model input\_reader is next time: inf

State for model input\_reader1 is next time: 00:00:30:000

State for model sender1 is packetNum: 1 & totalPacketNum: 5

State for model receiver1 is ackNum: 0

State for model FileServerReceiver1 is index: 2 & transmitting: 1 0 & ackToSender: 0 0

State for model FileServerSender1 is index: 4 & transmittingToReceiver: 0 0 & ackFromReceiver: 0 0 & requestIn: 0

State for model subnetA1 is index: 1 & transmitting: 1 0

State for model subnetA2 is index: 1 & transmitting: 0 0

State for model subnetB1 is index: 1 & transmitting: 0 0

State for model subnetB2 is index: 1 & transmitting: 1 0

**TEST SCENARIO**

Now, to test if there’s any data sent to the receiver when there is no requestIn to the File Server Exchange, an input set was created where the incoming request is zero.

Note: In this implementation the requestIn has to be positive integer to allow for download. Zero value means no request has been placed. So we input the values for controlIn to create packet and send to the receiver. At the same time we input zero values for the RequestIn. This mimics the condition when no request has been placed. The result for this scenario is as follows:

**ControlIn:**

00:00:10 5

00:01:30 3

**RequestIn:** ( Here we use “NoRequest.txt” to input for this scenario)

00:00:10 0

00:01:30 0

00:02:00 0

00:02:30 0

**Output:** (Here we see that no data is transferred on FileServerSender because there is no requestIn to the server for download. There is no FileServerSender :: dataOutS logs in the below output.)

00:00:00:000

00:00:00:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {}] generated by model input\_reader

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {}] generated by model input\_reader1

00:00:10:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {5}] generated by model input\_reader

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {0}] generated by model input\_reader1

00:00:20:000

[Sender\_defs::packetSentOut: {}, Sender\_defs::ackReceivedOut: {}, Sender\_defs::dataOut: {}] generated by model sender1

00:00:30:000

[Sender\_defs::packetSentOut: {1}, Sender\_defs::ackReceivedOut: {}, Sender\_defs::dataOut: {1 0}] generated by model sender1

00:00:33:000

[Subnet\_defs::out: {1 0}] generated by model subnetA1

00:00:36:000

[FileServerReceiver\_defs::ackOutR: {}, FileServerReceiver\_defs::storeDataS: {1 0}] generated by model FileServerReceiver1

00:01:30:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {3}] generated by model input\_reader

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {0}] generated by model input\_reader1

00:02:00:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {0}] generated by model input\_reader1

00:02:30:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {0}] generated by model input\_reader1

00:03:00:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {0}] generated by model input\_reader1

00:03:30:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {0}] generated by model input\_reader1

00:04:00:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {0}] generated by model input\_reader1

00:40:30:000

[cadmium::basic\_models::pdevs::iestream\_input\_defs<int>::out: {0}] generated by model input\_reader1

**CONCLUSION**

The Cadmium implementation of the Offline File Transfer simulator handles most of the features of its CD++ counterpart model. Here we have tried to understand the DEVS modeling and simulation through this implementation.