Manufacturing Cell Control System

Cadmium Simulator

Alvi Jawad
ST. ID: 101148341
Email: alvi.jawad@carleton.ca

SYSC 5104
Assignment 1
October 25, 2020

Table of Contents

Part I: Conceptual Design

Manufacturing Cell Control System

A Manufacturing Cell Control System (MCCS) is part of a distributed production network that prepares batches of materials for production. The three components of an MCCS, namely the Control Agent, the Storage Agent, and the Handling Agent, coordinate their activities using message-passing communication to deliver the required materials for production. The system responds to external production requests coming to the Control Agent by activating and communicating with the other agents and turns idle once all requested materials have been successfully prepared for processing.

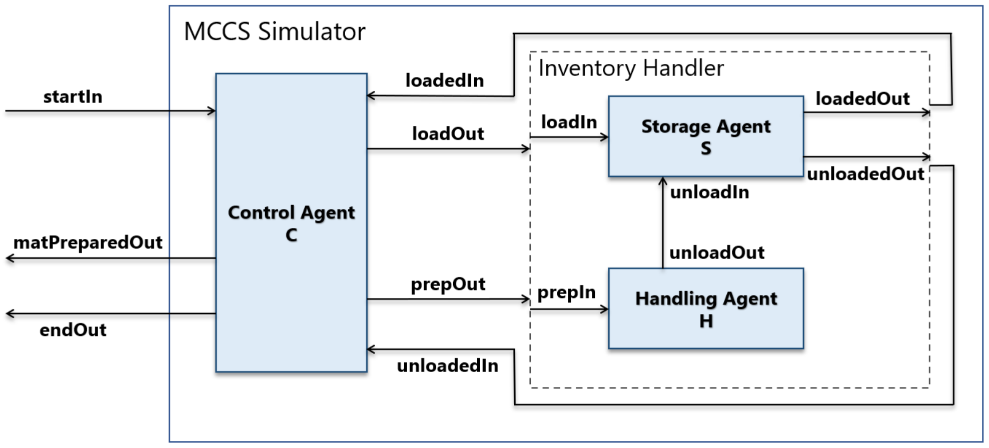


Figure 1: Structure of the MCCS Simulator

Figure 1 represents a brief sketch of the MCCS simulator with its three atomic models - Control Agent C, Storage Agent S, and Handling Agent H - and a structural (coupled) model Inventory Handler that encompasses S and H. The Control Agent acts as the supervisory system that coordinates actions between the other agents and is also the only agent capable of receiving and sending signals to external entities.

At the beginning of the production phase, Control Agent C receives batches of production requests (***start***) from *startIn*. C immediately activates the Storage Agent S by sending a ***load*** signal. Upon receiving the signal, S checks the inventory, loads the material if storage is empty, and sends an acknowledgement (***loaded***) to C when material storage is full. This indicates that materials are now ready to be transported. C then sends a ***prep*** message to the Handling Agent H requesting the preparation of materials for processing. The system now enters the handling phase where H, which is analogous to a robotic arm, starts moving the material to the designated place (e.g., a conveyor belt) for processing. After a certain period, all materials are moved from the storage, and H transitions to its idle behavior after sending an ***unload*** message to S. Upon receipt, S sends an acknowledgement (***unloaded***) that the inventory is now empty to C and changes to its idle behavior, waiting for further ***load*** instructions from C.

After receiving the ***unloaded*** signal, Control Agent C sends a message ***matPrepared*** to external entities. This may activate other systems in the production process and/or a Processing Agent (outside of scope) that now starts the processing phase. Concurrently, C also checks if more materials are needed to be delivered. If this was the last requested material, C also sends an ***end*** message through *endOut*, and the entire MCCS becomes idle. Otherwise, C reactivates the Storage Agent and the previous process continues until all requested materials have been successfully delivered, and there are no more incoming requests.

Part II: Formal Specification

In this section, we provide a formal specification for all of our atomic and coupled models using the Discrete Event System Specification (DEVS) formalism. The models are specified in ascending hierarchical order.

The Handling Agent H - Atomic

We start with a brief discussion and the specification of the simplest model in our system, the Handling Agent H. *Figure 2* presents its two internal states, *passive (P)* and *active (A)*, and the behavioral changes to those states upon the receipt of an external message. Prior to receiving any input, H remains in its *passive* state. After receiving the ***prep*** message at the start of the Preparation phase from Control Agent C, H transitions to its active *state* and begins moving the material. This process takes some time, and in our simulations, we assume the moving time to be 5 seconds for each unit of material. After the moving is complete, H sends an ***unload*** signal to the Storage Agent S to inform the successful transportation of material.

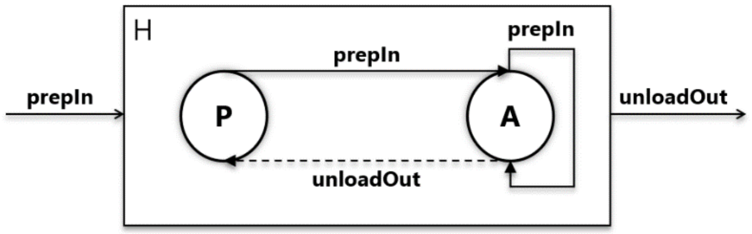


Figure 2: Handling Agent H

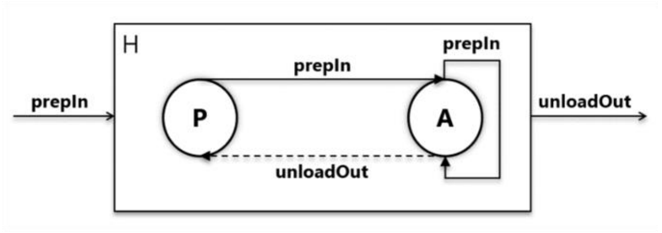
The state variables for H are initialized as follows:

```
phase = empty;                //H in passive mode before receiving any input
sending  = false;             //not sending any output
moving_time = 5;              //time required to move each unit of material
index = 0;                    //used only in the simulator to track incoming requests
```

DEVS Specification: H

Handling Agent, H = <S, X, Y, δ_{ext}, δ_{int}, λ, ta>

S = {phase, sending, moving_time}
X = {prepIn}
Y = {unloadOut}



```
δext (phase, e, prepIn){
        case phase
                passive:
                        phase = active;
                        sending = true;
                active:
                        ;        //input should not be here, ignore input
}

δint (phase){
        case phase
                passive:
                        ;        //do nothing
                active:
                        phase = passive;
                        sending = false;
}

λ(sending){
        send message to port unloadOut
}

ta(sending){
        if (sending){
                next_internal = moving_time;
        } else {
                next_internal =  INFINITY;
        }
}
```

The Storage Agent S - Atomic

The Storage Agent S, illustrated in *Figure 3*, acts as the material inventory for the MCCS. It has two states, *empty (E)* and *full (F)*, representing whether the inventory storage is loaded with materials or not. S receives ***load*** messages from the Control Agent C, asking for the loading of material. After a loading time, which we assume to be 2 seconds, S informs C of its *full* state by sending a ***loaded*** signal in response. S also receives ***unloaded*** messages from the Handling Agent H. In this case, S recognizes that the inventory has been emptied and moves to its *empty* state immediately. Additionally, S sends an ***unloaded*** message to C, allowing C to start the preparation phase.

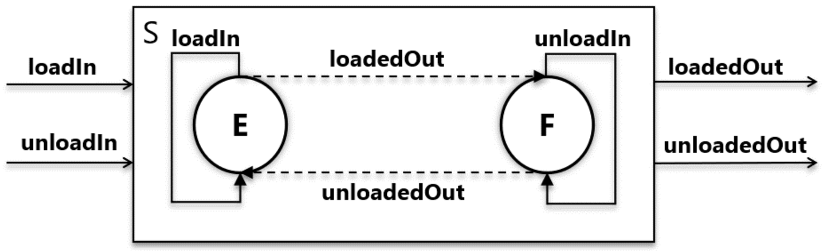


Figure 3: Storage Agent S

The state variables for S are initialized as follows:

```
phase = empty;                  //The inventory is empty before receiving any input
sending  = false;               //not sending any output
loading_time  = 2;              //time taken to load one unit of material
load_request_index = 0;         //used only in the simulator to track the no. of load
                                //requests from C
unload_request_index  = 0;      //used only in the simulator to track the no. of unload
                                //requests from H
```

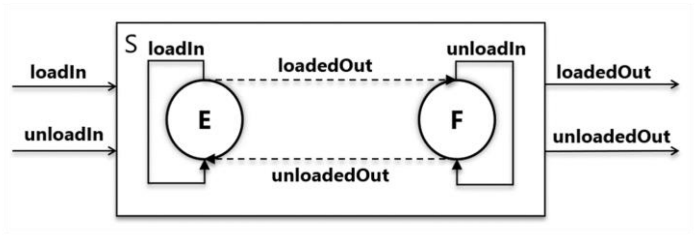
Type: Message

Both the Handling Agent H and Storage Agent S receive input messages in the form of <material_unit_x> <ready_status>. Upon receiving the first batch request, the Control Agent C creates the first message "*material unit 1 (not ready)*". This message propagates through S to H in the same form. After H completes moving the material it changes the state of the message to "*material unit 1 (ready)*". This modified message now propagates via S to C to let C know about the ready status of the prepared material and send the ***matPrepared*** signal accordingly. If for any reason the material is not ready, C does not create a new material request and does not send the ***matPrepared*** signal. Rather, C sends the previous request "*material unit 1 (not ready)*" again for a proper delivery.

DEVS Specification: S

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

S = {phase, sending, loading_time}
X = {loadIn, unloadIn}
Y = {loadedOut, unloadedOut}



```
δext (phase, e, loadIn, unloadIn){
    case phase
        empty:
            if (input is from loadIn){
                sending = true;
            } else {
                ;       //input should not be here, ignore input
            }
        full:
            if (input is from unloadIn){
                sending = true;
            } else {
                ;       //input should not be here, ignore input
            }
}

δint (phase){
    sending = false;
    if (empty){
        phase = full;
    } else {
        phase = empty;
    }
}

λ(phase, sending){
    if (empty & sending){
        send message to port loadedOut
    } else if (full & sending){
        send message to port unloadedOut
    } else {
        ;           //no output
    }
}
```

```
ta(phase, sending){
        if (empty & sending){
                next_internal = loading_time;
        } else if (full & sending){
                next_internal =  0;             //immediately trigger λ and δint
        } else{
                next_internal =  INFINITY;
        }
}
```

The Control Agent C - Atomic

The Control Agent C acts as the supervisory system for the entire MCCS. This is the only agent that can communicate with the external entities to receive batches of production requests as well as send messages signifying the successful preparation of material to other external agents. The control agent has three internal states, presented in *Figure 4*. The states *idle (Id)*, *init (In)*, and *prep (Pr)* symbolizes the Idle, Initialization, and Preparation stages of the MCCS system. In other words, these states represent the system state of the MCCS.

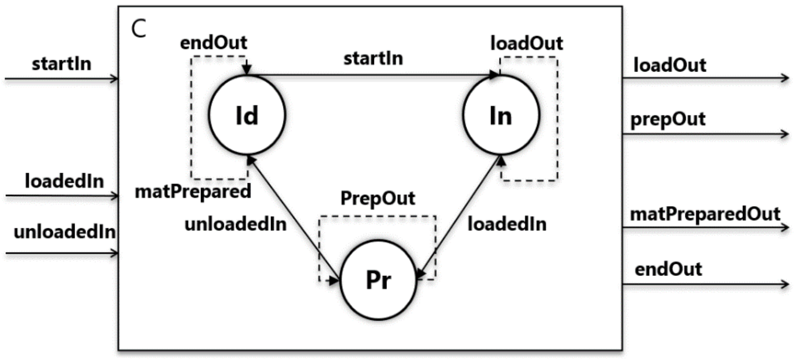


Figure 4: Control Agent C

In the Idle stage, Control Agent C awaits a production request to trigger a transition. Upon receiving one or more (batch) requests from an external entity (***start***), C begins the Initialization stage and transitions to its *Init* phase. In addition, C creates a new material delivery request "*material unit 1 (not ready)*" and sends this as a ***load*** message to Storage Agent S. C keeps waiting until a response from S arrives in the form of a ***loaded*** message. C now knows that the material storage is full and according to the specification, transitions to the *prep* phase, sending out a ***prep*** request to Handling Agent H.

In this Preparation stage, Control Agent C waits until the receipt of an ***unloaded*** message. Upon receiving the message C checks the *ready_status* of the message to see whether the material was properly moved or not. Upon verification, C sends out a **matPrepared** signal to external entities. Furthermore, if this was the last requested unit, C also sends out an ***end*** message and moves to its *idle* state. This puts the MCCS in an Idle system state until further external requests are received. However, if there are still materials left to be prepared, C instead moves to the *init* stage, creates a new material request, and starts the Initialization-Preparation cycle all over again.

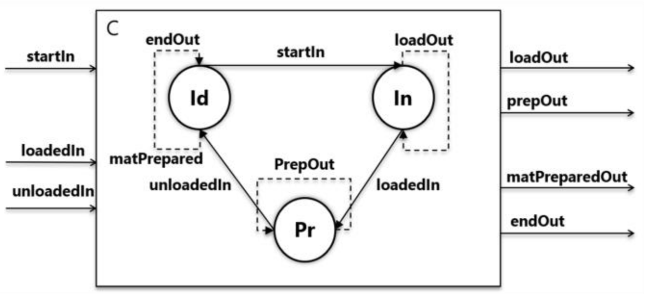
The state variables for C are initialized as follows:

```
phase = idle;               //MCCS system state is idle initially
sending  = false;           //not sending any output
total_mats  = 0;            //the total number of materials to be prepared
num_prepared = 0;           //no. of materials prepared up until now
fin  = false;               //whether the current material request is ready or not
```

DEVS Specification: C

Control Agent, C = <S, X, Y, δ_{ext}, δ_{int}, λ, ta>

S = {phase, sending, total_mats, num_prepared, fin}
X = {loadIn, unloadIn}
Y = {loadedOut, unloadedOut}



```
δ_ext (phase, e, startIn, loadedIn, unloadedIn){
        case phase
                idle:
                        if (input is from startIn){
                                phase = init;
                                total_mats = startIn;
                                sending = true;
                        } else {
                                ;        //input should not be here, ignore input
                        }
                init:
                        if (input is from startIn){
                                total_mats += startIn;      //increase the total required number
                        } else if (input is from loadedIn){
                                phase = prep;
                                sending = true;
                        } else {
                                ;        //input should not be here, ignore input
                        }
```

```
                    prep:
                              if (input is from startIn){
                                        total_mats += startIn;          //increase the total required number
                              } else if (input is from unloadedIn){
                                        If (material is in ready state){
                                                  fin = true;           //one material preparation finished
                                                  num_prepared++;
                                        }
                                        sending = true;
                                        if (num_prepared == total_mats){
                                                  phase = idle;         //all materials prepared
                                        } else {
                                                  Phase = init;
                                        }
                              } else {
                                        ;         //input should not be here, ignore input
                    }
}

δint (){
          sending = false;
          fin = false;
}

λ(phase, sending, fin){
          if (init & sending){
                    if (fin){
                              send num_prepared to port matPreparedOut
                    }
                    send message to port loadOut
          } else if (prep & sending){
                    send message to port prepOut
          } else if (idle & sending){
                    send a message indicating request completion to port endOut
          } else {
                    ;                   //no output
          }
}

ta(sending){
          if (sending){
                    next_internal = 0;                 //immediately trigger λ and δint
          } else {
                    next_internal =  INFINITY;
          }
}
```

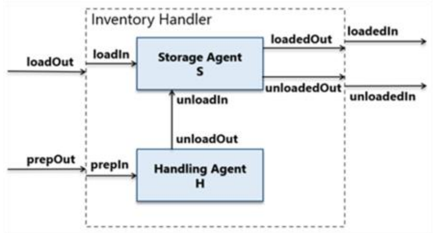
The Inventory Handler IH – Coupled

The first coupled model in the MCCS is the Inventory Handler (IH). The inventory handler encompasses the Storage Agent S and Handling Agent H atomic models and acts as an abstract entity that receives ***load*** and ***prep*** signals from the Control Agent C and replies with subsequent ***loaded*** and ***unloaded*** responses.

DEVS Specification: IH

Inventory Handler, IH = <X, Y, D, M, EIC, EOC, IC, SELECT>

X = {loadIn, prepIn}
Y = {loadedOut, unloadedOut}
D = {storage1, handling1}
M (storage1) = S
M (handling1) = H



EIC = {((self, “loadOut”), (storage1, “loadIn”)), ((self, “prepOut”), (handling1, “prepIn”)}
EOC = {((storage1, “loadedOut”), (self, “loadedIn”)), ((storage1, “unloadedOut”), (self, “unloadedIn”))}
IC = {((handling1, “unloadOut”), (storage1, “unloadIn”))}
SELECT: //not used

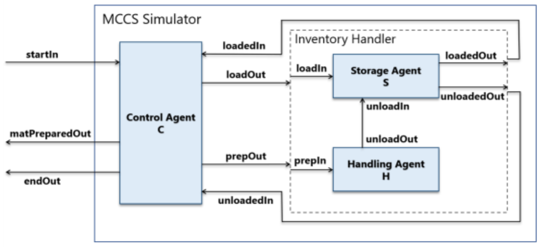
The MCCS – Coupled

The final coupled model is the MCCS itself. The MCCS encapsulates the Control Agent C, Storage Agent S, and Handling Agent H, and acts as an abstract entity that receives material requests (***start***) from external entities, and sends messages that indicate a prepared material (***matPrepared***) or the end of service (***end***) to other external entities.

DEVS Specification: MCCS

Manufacturing Cell Control System, MCCS = <X, Y, D, M, EIC, EOC, IC, SELECT>

X = {startIn}
Y = {matPreparedOut, endOut}
D = {control1, IH1}
M(control1) = C
M (IH1) = IH



EIC = {((self, “startIn”),(control1, “startIn”))}
EOC = {((control1, ”matPreparedOut”), (self, “matPreparedOut”)),
((control1, ”endOut”), (self, “endOut”))}
IC = {((control1, “loadOut”), (IH1, “loadIn”)), ((IH1, “loadedOut”), (control1, “loadedIn”))
((control1, ”prepOut”), (IH1, “prepIn”)), ((IH1, ”unloadedOut”), (control1, “unloadedIn”))}
SELECT: //not used

Experimentation Strategy

The models built using the specifications provided above will be tested using the "black box" testing method. We will abstract away from the inner workings of the models and test and observe the reaction of the model when faced with different input values. The goal here is to confirm whether different combinations of inputs, in all possible cases, provide the correct and expected set of outputs according to our system specification.

Test Cases – Atomic Models [H, S, C]

The MCCS deals with the preparation of each unit of material at a time. All of the atomic models perform very specific functions, and these functionalities will be thoroughly tested for each of the atomic models. In this section, several test cases will be briefly outlined. These test cases will be followed by detailed experiments in the next section.

Handling Agent H

The Handling Agent H receives preparation requests from the Control Agent C to move a specific unit of material. H, as an automated system, has a predefined time (five seconds) that it takes to move each unit of material. After the moving is complete, H sends a message to notify the Storage Agent that the moving is complete.

We will check whether the Handling Agent H produces correct outputs after moving the material (a five-second delay) given a correct set of inputs and whether it rejects invalid requests and/or inputs.

Storage Agent S

The Storage Agent S receives loading requests from the Control Agent C and loads the material inventory after a predefined time (two seconds), sending a notification that the storage is now full and the material can be moved. S also receives unload requests from the Handling Agent H that indicates that H has already moved the current requested material. S uses this signal to switch phases and relays the message to C.

We will check whether the Storage Agent S produces correct outputs loading the material (a two-second delay) given a correct set of inputs and whether it rejects invalid requests and/or inputs.

Control Agent C

The Control Agent C receives external requests to prepare materials for production in batches. It notifies the Storage Agent S and the Handling agent H to start the loading and moving of material, respectively. After any of these are done, C receives both acknowledgements from S. After any unit of material has been prepared, C notifies external entities that may now start processing that unit. Finally, C sends out a signal indicating the system idle state when all requested materials have been delivered for production.

We will check whether the Handling Agent H successfully produces all requested materials, whether it produces the correct outputs given a correct set of inputs, and whether it rejects invalid requests and/or inputs.

Test Cases – Coupled Models [IH, MCCS]

Having checked all the atomic models for invalid requests and/or incorrect inputs, for the coupled models, we will only focus on whether a correct set of inputs produce a correct and expected set of outputs or not.

Inventory Handler IH

The Inventory Handler H acts as an abstract entity that receives loading and moving tasks from Control Agent C and produces acknowledgements when the corresponding tasks are complete. Includes:

- Atomic model – Storage Agent S
- Atomic model – Handling Agent H

We will check whether the Inventory Handler H produces correct outputs both after loading the storage (a two-second delay) and after moving the material (a five-second delay) given a correct set of inputs.

Manufacturing Cell Control System MCCS

The abstract top model of our system that communicates only with the outside world. It receives material preparation requests in batches and notifies the external entities after each material is produced and when the entire system becomes idle after servicing all material requests.

- Atomic model – Control Agent C
- Coupled model – Inventory Handler IH

We will check whether the MCCS produces correct outputs in all possible cases given any correct set of inputs.

Part III: Simulation and Experimentation

In this section, we explain individual test cases for all of our atomic and coupled models. Additionally, we perform simulations and present output logs to ensure the proper and expected behavior of our models according to our specifications.

TEST: Handling Agent H – Atomic

We start with our simple Handling Agent H (introduced on page 3) model and its behavior evaluation. H as a single input and a single output, both of type *message* (see page 5) in the form of “*material unit x (ready_status)*”. Here, *x* represents the current index of the material requested by Control Agent C and the preparation of which is in progress. *ready_status,* on the other hand, represents the moved/not moved status of the current material. For example, When the second ***prep*** request arrives from C, the message should be in the form “*material unit 2 (not ready)*”.

After H completes the transportation, the outgoing ***unload*** message to Storage Agent S should read "*material unit 2 (ready)*". H should take exactly five seconds to finish the moving of one unit of material. Finally, while H is already working on a material, any requests arriving within those five seconds (invalid requests) should be discarded.

We prepare an input file named *handling_input_test.txt* (located in *project_folder/input_data/)* that contains the necessary inputs to test the above conditions. Inputs are in the form of <time> <material_unit> <ready_status>. So, the input **"**`00:00:10 1 0`**"** means that message arrived 10 seconds after the simulation start, this is the first unit, and the material status is not ready. In short, this should produce the message "*material unit 1 (not ready)*". The input file contains:

```
00:00:10 1 0
00:00:15 2 0
00:00:22 3 0
00:00:30 4 0
```

According to our specification, all four units of material should be successfully prepared after five seconds. If there are invalid inputs, e.g., a new request arrives while a previous material preparation is still in progress, or material already arrives prepared (*ready*), the program should terminate with an assert statement.

After compilations, running **HANDLING_TEST.exe** (located in *project_folder/bin*) should create the log file for output messages (*handling_test_output_messages.txt*) as well as output states (*handling_test_output_state.txt*) in the *project_folder/simulation_results* folder. The output messages from our simulation are displayed below.

Output Messages:

```
00:00:10:000
[Material unit 1 (not ready) generated by model input_reader
00:00:15:000
[{Material unit 2 (not ready)} generated by model input_reader
[unloadOut: {Material unit 1 (ready)}] generated by model handling1
00:00:20:000
[unloadOut: {Material unit 2 (ready)}] generated by model handling1
00:00:22:000
[{Material unit 3 (not ready)}] generated by model input_reader
00:00:27:000
[unloadOut: [{Material unit 3 (ready)}] generated by model handling1
00:00:30:000
[{Material unit 4 (not ready)}] generated by model input_reader
00:00:35:000
[UnloadOut: {Material unit 4 (ready)}] generated by model handling1
```

The simulation results are simplified, and the simulation messages are bolded for easier visualization. The complete results can be seen in their respective files mentioned above after running the simulations. The simulation returns expected results, as each ***prep*** request is replied with an ***unload*** request exactly five seconds after the material moving is complete.

The output states for the first two units of material are also shown below.

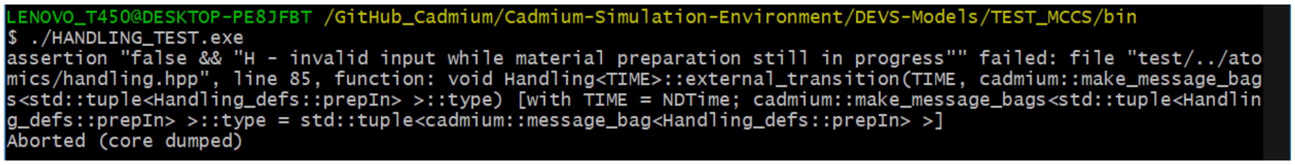
Output States:

```
00:00:00:000
State for model input_reader is next time: 00:00:00:000
State for model handling1 is :
        phase: passive    sending: 0    index: 0
00:00:00:000
State for model input_reader is next time: 00:00:10:000
State for model handling1 is :
        phase: passive    sending: 0    index: 0
00:00:10:000
State for model input_reader is next time: 00:00:05:000
State for model handling1 is :
        phase: active     sending: 1    index: 1
00:00:15:000
State for model input_reader is next time: 00:00:07:000
State for model handling1 is :
        phase: active     sending: 1    index: 2
00:00:20:000
State for model input_reader is next time: 00:00:07:000
State for model handling1 is :
        phase: passive    sending: 0    index: 2
```

We now repeat our simulation steps with invalid inputs (bolded). First, we try to send new material requests while another material loading is already in progress.

```
00:00:10 1 0
00:00:12 2 0
```

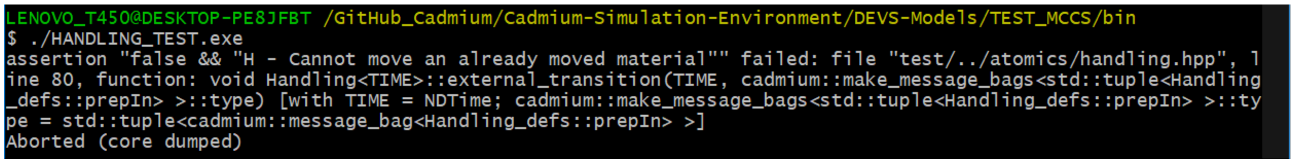
The program terminates with an assert statement "*H - invalid input while material preparation still in progress*" indicating an invalid request.


```
LENOVO_T450@DESKTOP-PE8JFBT /GitHub_Cadmium/Cadmium-Simulation-Environment/DEVS-Models/TEST_MCCS/bin
$ ./HANDLING_TEST.exe
assertion "false && "H - invalid input while material preparation still in progress"" failed: file "test/../ato
mics/handling.hpp", line 85, function: void Handling<TIME>::external_transition(TIME, cadmium::make_message_bag
s<std::tuple<Handling_defs::prepIn> >::type) [with TIME = NDTime; cadmium::make_message_bags<std::tuple<Handlin
g_defs::prepIn> >::type = std::tuple<cadmium::message_bag<Handling_defs::prepIn> >]
Aborted (core dumped)
```

Next, we also try to pass material requests that have already been prepared (bolded).

```
00:00:10 1 0
00:00:15 2 1
```

The result is another assert statement "*H - Cannot move an already moved material*" signifying another unexpected input.


```
LENOVO_T450@DESKTOP-PE8JFBT /GitHub_Cadmium/Cadmium-Simulation-Environment/DEVS-Models/TEST_MCCS/bin
$ ./HANDLING_TEST.exe
assertion "false && "H - Cannot move an already moved material"" failed: file "test/../atomics/handling.hpp", l
ine 80, function: void Handling<TIME>::external_transition(TIME, cadmium::make_message_bags<std::tuple<Handling
_defs::prepIn> >::type) [with TIME = NDTime; cadmium::make_message_bags<std::tuple<Handling_defs::prepIn> >::ty
pe = std::tuple<cadmium::message_bag<Handling_defs::prepIn> >]
Aborted (core dumped)
```

TEST: Storage Agent – Atomic

For our second test case, we now move the Storage Agent S. S has two inputs and two outputs, all of which are of type *message* (see page 5). The input message ***load***, as well as the output message ***loaded***, should be in the form "material unit x (not ready)" as that unit has not been moved yet. The second input from the Handling Agent H comes after the material has been moved, and thus, the input message ***unload*** as well as the subsequent output message ***unloaded*** should be in the form "material unit x (ready)". In terms of time, ***loaded*** messages should be sent after 2 seconds of receiving a ***load*** message, whereas ***unloaded*** messages should be sent immediately after receiving the corresponding **unload** messages.

We prepare two input files:

1. *storage_input_test_loadIn.txt (*for ***load*** messages*)*
2. *storage_input_test_unloadIn.txt (*for ***unload*** messages*)*

Both files are located in *project_folder/input_data/* that contain inputs in the same form as the Handling Agent test case (see page 12). We keep tests simple this time and only check for the expected behavior of the Storage Agent S. The input file contents are as follows:

load messages	***unload*** messages
00:00:05 1 0	00:00:12 1 1
00:00:13 2 0	00:00:20 2 1
00:00:30 3 0	00:00:37 3 1

The time difference between ***load*** messages and ***unload*** messages should be seven seconds in order to account for the loading time (two seconds) and moving time (five seconds). The request for material unit 2 arrives immediately after the first unit is prepared. This represents a continuous production for batch requests. The request for material unit 3 arrives slightly late. Regardless of the arrival time, the Storage Agent S should produce ***loaded*** messages after a two -second delay and ***unloaded*** messages immediately, as mentioned before.

After compilations, running **STORAGE_TEST.exe** (located in *project_folder/bin*) should create the log file for output messages (*storage_test_output_messages.txt*) as well as output states (*storage_test_output_state.txt*) in the *project_folder/simulation_results* folder. The output messages from our simulation are displayed below.

Output Messages:

```
00:00:05:000
[{Material unit 1 (not ready)}] generated by model input_reader_load
00:00:07:000
[{Material unit 1 (ready)}] generated by model input_reader_unload
[loadedOut: {Material unit 1 (not ready)}, unloadedOut: {}] generated by model Storage1
00:00:07:000
[loadedOut: {}, unloadedOut: {Material unit 1 (ready)}] generated by model Storage1
```

```
00:00:12:000
[{Material unit 1 (ready)}] generated by model input_reader_unload
00:00:12:000
[loadedOut: {}, unloadedOut: {Material unit 1 (ready)}] generated by model Storage1
00:00:13:000
[{Material unit 2 (not ready)}] generated by model input_reader_load
00:00:15:000
[loadedOut: {Material unit 2 (not ready)}, unloadedOut: {}] generated by model Storage1
00:00:20:000
[{Material unit 2 (ready)}] generated by model input_reader_unload
00:00:20:000
[loadedOut: {}, unloadedOut: {Material unit 2 (ready)}] generated by model Storage1
00:00:30:000
[{Material unit 3 (not ready)}] generated by model input_reader_load
00:00:32:000
[loadedOut: {Material unit 3 (not ready)}, unloadedOut: {}] generated by model Storage1
00:00:37:000
[{Material unit 3 (ready)}] generated by model input_reader_unload
00:00:37:000
[loadedOut: {}, unloadedOut: {Material unit 3 (ready)}] generated by model Storage1
```

The outputs are exactly as we expected. We can see that each ***load*** request generated is serviced exactly two seconds after with a subsequent ***loaded*** message. Similarly, each ***unload*** request is immediately relayed to the Control Agent C as an ***unloaded*** message.

The output states for the first unit of material is also shown below.

Output States:

```
00:00:00:000
State for model input_reader_load is next time: 00:00:00:000
State for model input_reader_unload is next time: 00:00:00:000
State for model Storage1 is :
        phase: empty   sending: 0   load requests received: 0   unload requests received: 0
00:00:00:000
State for model input_reader_load is next time: 00:00:05:000
State for model input_reader_unload is next time: 00:00:12:000
State for model Storage1 is :
        phase: empty   sending: 0   load requests received: 0   unload requests received: 0
00:00:05:000
State for model input_reader_load is next time: 00:00:08:000
State for model input_reader_unload is next time: 00:00:12:000
State for model Storage1 is :
        phase: empty   sending: 1   load requests received: 1   unload requests received: 0
00:00:07:000
State for model input_reader_load is next time: 00:00:08:000
State for model input_reader_unload is next time: 00:00:12:000
State for model Storage1 is :
        phase: full    sending: 0   load requests received: 1   unload requests received: 0
00:00:12:000
State for model input_reader_load is next time: 00:00:08:000
State for model input_reader_unload is next time: 00:00:08:000
State for model Storage1 is :
        phase: full    sending: 1   load requests received: 1   unload requests received: 1
00:00:12:000
State for model input_reader_load is next time: 00:00:08:000
State for model input_reader_unload is next time: 00:00:08:000
State for model Storage1 is :
        phase: empty    sending: 0   load requests received: 1    unload requests received: 1
```

Similar to before, inputs at invalid times end the simulation with assert statements such as "*S - Invalid load request while material storage already full*" or "*S - Invalid unload request while material storage already empty*". In addition, incorrect requests (e.g., asking to load an already moved material) also end with assert statements such as "*S - Cannot load an already moved material*" or "S - unload request without moving the material". We should also be careful when creating input files as multiple inputs at the same time will also trigger an assert statement saying "*S - Only one message is allowed per time unit*". We would like to avoid that situation since that will never happen in the MCCS case.

TEST: Inventory Handler IH – Coupled

Before testing our last atomic model, we first test the Inventory Handler IH as it has a behavior that involves both of the atomic models tested until now. IH has two inputs and two outputs, all of which are of type *message* (see page 5). IH takes ***load*** and ***prep*** messages from the Control Agent C as input and responds with ***loaded*** and ***unloaded*** messages after two and five seconds, respectively.

We prepare two input files:

1. *InventoryHandler_input_test_loadIn.txt (*for ***load*** messages*)*
2. *InventoryHandler_input_test_prepIn.txt (*for ***prep*** messages*)*

Both files are located in *project_folder/input_data/* that contain inputs in the same form as the Handling Agent test case (see page 12). Since we have already verified the atomic model behaviors, we will only focus on simple inputs and outputs this time. The input file contents are as follows:

load messages	***prep*** messages
00:00:05 1 0	00:00:07 1 0
00:00:13 2 0	00:00:15 2 0
00:00:30 3 0	00:00:32 3 0

The time difference between the two inputs for each material should be two seconds (necessary to load the material storage). The ***load*** input for material unit 2 is scheduled to arrive at the same time when material unit 1 has been prepared, whereas the third load request has a significant delay. Nevertheless, we expect each ***load*** and ***prep*** request to be answered with a corresponding ***loaded*** and ***unloaded*** request exactly after two and five seconds, respectively.

After compilations, running **IH_TEST.exe** (located in *project_folder/bin*) should create the log file for output messages (*InventoryHandler_test_output_messages.txt*) as well as output states

(*InventoryHandler_test_output_state.txt*) in the *project_folder/simulation_results* folder. The output messages from our simulation are displayed below.

Output Messages:

```
00:00:05:000
[{Material unit 1 (not ready)}] generated by model input_reader_load
00:00:07:000
[{Material unit 1 (not ready)}] generated by model input_reader_prep
[loadedOut: {Material unit 1 (not ready)}, unloadedOut: {}] generated by model storage1
00:00:12:000
[unloadOut: {Material unit 1 (ready)}] generated by model handling1
00:00:12:000
[loadedOut: {}, unloadedOut: {Material unit 1 (ready)}] generated by model storage1
00:00:13:000
[{Material unit 2 (not ready)}] generated by model input_reader_load
00:00:15:000
[{Material unit 2 (not ready)}] generated by model input_reader_prep
[loadedOut: {Material unit 2 (not ready)}, unloadedOut: {}] generated by model storage1
00:00:20:000
[unloadOut: {Material unit 2 (ready)}] generated by model handling1
00:00:20:000
[loadedOut: {}, unloadedOut: {Material unit 2 (ready)}] generated by model storage1
00:00:30:000
[{Material unit 3 (not ready)}] generated by model input_reader_load
00:00:32:000
[{Material unit 3 (not ready)}] generated by model input_reader_prep
[loadedOut: {Material unit 3 (not ready)},
unloadedOut: {}] generated by model storage1
00:00:37:000
[unloadOut: {Material unit 3 (ready)}] generated by model handling1
00:00:37:000
[loadedOut: {}, unloadedOut: {Material unit 3 (ready)}] generated by model storage1
```

As expected, we can see that each ***load*** message is responded exactly two seconds after with a ***loaded*** message. Additionally, each ***prep*** request is generated at the same time as the previous loaded response to mimic the behavior of Control Agent C, which is responded with a modified ***unloaded*** message (material is now ready) after five seconds. All cases return expected results.

The output states for the first unit of material is also shown below.

Output States:

```
00:00:00:000
State for model input_reader_load is next time: 00:00:00:000
State for model input_reader_prep is next time: 00:00:00:000
State for model storage1 is :
        phase: empty    sending: 0   load requests received: 0   unload requests received: 0
State for model handling1 is :
        phase: passive  sending: 0     index: 0
00:00:00:000
State for model input_reader_load is next time: 00:00:05:000
State for model input_reader_prep is next time: 00:00:07:000
State for model storage1 is :
        phase: empty    sending: 0   load requests received: 0   unload requests received: 0
State for model handling1 is :
        phase: passive  sending: 0     index: 0
00:00:05:000
State for model input_reader_load is next time: 00:00:08:000
State for model input_reader_prep is next time: 00:00:07:000
State for model storage1 is :
        phase: empty    sending: 1   load requests received: 1   unload requests received: 0
State for model handling1 is :
        phase: passive  sending: 0     index: 0
00:00:07:000
State for model input_reader_load is next time: 00:00:08:000
State for model input_reader_prep is next time: 00:00:08:000
State for model storage1 is :
        phase: full     sending: 0   load requests received: 1   unload requests received: 0
State for model handling1 is :
        phase: active   sending: 1     index: 1
00:00:12:000
State for model input_reader_load is next time: 00:00:08:000
State for model input_reader_prep is next time: 00:00:08:000
State for model storage1 is :
        phase: full     sending: 1   load requests received: 1   unload requests received: 1
State for model handling1 is :
        phase: passive  sending: 0     index: 1
00:00:12:000
State for model input_reader_load is next time: 00:00:08:000
State for model input_reader_prep is next time: 00:00:08:000
State for model storage1 is :
        phase: empty    sending: 0   load requests received: 1   unload requests received: 1
State for model handling1 is :
        phase: passive  sending: 0     index: 1
```

TEST: Control Agent C - Atomic

Now, we move toward verifying the correctness of our final atomic model, the Control Agent C. The behavior of C is rather complex, as it contains three inputs and four outputs of different types. The ***load*** and ***prep*** messages are requests to the Storage Agent S and Handling Agent H, respectively. The ***loaded*** and ***unloaded*** messages are both returned messages from S to notify C about its *full* and *empty* states, respectively. These four messages are of type message (see page 5), and their contents have been discussed extensively in the previous test cases. Only the unloaded message should contain the material state in the ready form as it is sent only after the material has been moved, and the inventory is empty.

The three other inputs and outputs are of type *int*. The Control Agent C uses these three messages to get as well as respond to external entities about the current system state and the state of production progress. C receives external ***start*** inputs in the form of batch requests. So the input **"**`00:00:05 3`**"** should be read as "a production request to produce 3 units of material", received five seconds after the simulation start. Inputs can arrive at any time; if inputs arrive while C is in its *idle* state, C starts the material preparation by creating a new ***load*** request for the Storage Agent S and moves to the *init* state. Any further inputs during loading (in the *init* state) and moving (in the *prep* state) are added to the total list of materials to be prepared.

When in the *prep* state, and C receives an ***unloaded*** message, one of two things can happen. This message notifies C that a material was just prepared. C now has to check whether this was the last of all the materials that were requested. If that is the case, C now moves to its *idle* state and sends out an ***end*** signal with an integer value of *1* that notifies external entities that the MCCS system has finished preparing all materials and is now idle. Otherwise, there are still materials left to be produced, and C reacts by moving to its init state, and creating another ***load*** request immediately. In both cases, a ***matPrepared*** message is sent to one or more external entities with the *integer value* of the material unit (e.g., *5* for material unit 5), indicating that a particular unit has been prepared and its processing can begin at once.

We prepare three input files:

1. *control_input_test_startIn.txt (*for ***start*** messages*)*
2. *control_input_test_loadedIn.txt (*for ***loaded*** messages*)*
3. *control_input_test_unloadedIn.txt (*for ***unloaded*** messages*)*

We test a simple case where we have only two start messages and they do not overlap. All three files are located in *project_folder/input_data/* with varying inputs. The contents of the input files are as follows:

start messages	***loaded*** messages	***unloaded*** messages
`00:00:05 2`	`00:00:07 1 0`	`00:00:12 1 1`
`00:00:30 1`	`00:00:14 2 0`	`00:00:19 2 1`
	`00:00:32 3 0`	`00:00:37 3 1`

The Control Agent C produces a new material request either at each start request (e.g., at five seconds or 30 seconds) or immediately after receiving an unloaded message (e.g., at time 12 seconds) when there are still materials left to be prepared in the batch request. Each unit of material should take seven seconds to be prepared to account for the time needed to load and move that particular unit. Keeping that in mind, we should forge ***loaded*** messages to appear two seconds after each material request and ***unloaded*** messages after five seconds of each loaded message.

After compilations, running **CONTROL_TEST.exe** (located in *project_folder/bin*) should create the log file for output messages (*Control_test_output_messages.txt*) as well as output states (*Control_test_output_state.txt*) in the *project_folder/simulation_results* folder. The output messages from our simulation are displayed below.

Output Messages:
```
00:00:05:000
[{2}] generated by model input_reader_start
00:00:05:000
[loadOut: {Material unit 1 (not ready)}, prepOut: {}, matPreparedOut: {}, endOut: {}] generated by
model control1
```

```
00:00:07:000
[{Material unit 1 (not ready)}] generated by model input_reader_loaded
00:00:07:000
[loadOut: {}, prepOut: {Material unit 1 (not ready)}, matPreparedOut: {}, endOut: {}] generated by
model control1
00:00:12:000
[{Material unit 1 (ready)}] generated by model input_reader_unloaded
00:00:12:000
[loadOut: {Material unit 2 (not ready)}, prepOut: {}, matPreparedOut: {1}, endOut: {}] generated by
model control1
00:00:14:000
[Material unit 2 (not ready)}] generated by model input_reader_loaded
00:00:14:000
[loadOut: {}, prepOut: {Material unit 2 (not ready)}, matPreparedOut: {}, endOut: {}] generated by
model control1
00:00:19:000
[cadmium::basic_models::pdevs::iestream_input_defs<Message_t>::out:  {Material  unit  2  (ready)}]
generated by model input_reader_unloaded
00:00:19:000
[loadOut: {}, prepOut: {}, matPreparedOut: {2}, endOut: {1}] generated by model control1
00:00:30:000
[{1}] generated by model input_reader_start
00:00:30:000
[loadOut: {Material unit 3 (not ready)}, prepOut: {}, matPreparedOut: {}, endOut: {}] generated by
model control1
00:00:32:000
[{Material unit 3 (not ready)}] generated by model input_reader_loaded
00:00:32:000
[loadOut: {}, prepOut: {Material unit 3 (not ready)}, matPreparedOut: {}, endOut: {}] generated by
model control1
00:00:37:000
[{Material unit 3 (ready) }] generated by model input_reader_unloaded
00:00:37:000
[loadOut: {}, prepOut: {}, matPreparedOut: {3}, endOut: {1}] generated by model control1
```

The output messages in this case will require slightly more explanation. The Control Agent C receives the ***start*** request to prepare *two* units of material five seconds after the start of the simulation. C immediately generates and sends out a ***load*** message to Storage Agent S, which responds two seconds after (at seven seconds) with a ***loaded*** message. C immediately activates the moving by notifying Handling Agent H with a prep message at seven seconds. H takes five seconds to move the material and replies with an ***unloaded*** message through S 12 seconds after the simulation start.

At twelve seconds, Control Agent C does a couple of things. First, since the first unit of material has been prepared, it sends a ***matPrepared*** signal with the unit number, in this case, *1*, to notify external entities that the processing of material unit 1 can now begin. At the same time, C also checks if there are more materials left to be prepared. Since it received two material requests at the same time, it now moves to its init state, generates, and sends out another load request for material unit 2. On the other hand, at times 19 and 37 seconds, we can see that previous start requests have been met with proper preparation of materials and C outputs an additional ***end*** signal (*1* represents the end of all requests), along with the corresponding material unit number that was prepared.

The output messages in this case are exactly as we expected. We will test for overlapping ***start*** requests in the MCCS simulator case, although that has been verified to work here as well.

The output states for the first unit of material is also shown below.

Output States:

```
00:00:00:000
State for model input_reader_start is next time: 00:00:00:000
State for model input_reader_loaded is next time: 00:00:00:000
State for model input_reader_unloaded is next time: 00:00:00:000
State for model control1 is :
        phase: idle   sending: 0   fin: 0
        total requests: 0   current prepared materials: 0
00:00:00:000
State for model input_reader_start is next time: 00:00:05:000
State for model input_reader_loaded is next time: 00:00:07:000
State for model input_reader_unloaded is next time: 00:00:12:000
State for model control1 is :
        phase: idle   sending: 0   fin: 0
        total requests: 0   current prepared materials: 0
00:00:05:000
State for model input_reader_start is next time: 00:00:25:000
State for model input_reader_loaded is next time: 00:00:07:000
State for model input_reader_unloaded is next time: 00:00:12:000
State for model control1 is :
        phase: init   sending: 1   fin: 0
        total requests: 2   current prepared materials: 0
00:00:05:000
State for model input_reader_start is next time: 00:00:25:000
State for model input_reader_loaded is next time: 00:00:07:000
State for model input_reader_unloaded is next time: 00:00:12:000
State for model control1 is :
        phase: init   sending: 0   fin: 0
        total requests: 2   current prepared materials: 0
00:00:07:000
State for model input_reader_start is next time: 00:00:25:000
State for model input_reader_loaded is next time: 00:00:07:000
State for model input_reader_unloaded is next time: 00:00:12:000
State for model control1 is :
        phase: prep   sending: 1   fin: 0
        total requests: 2   current prepared materials: 0
00:00:07:000
State for model input_reader_start is next time: 00:00:25:000
State for model input_reader_loaded is next time: 00:00:07:000
State for model input_reader_unloaded is next time: 00:00:12:000
State for model control1 is :
        phase: prep   sending: 0   fin: 0
        total requests: 2   current prepared materials: 0
00:00:12:000
State for model input_reader_start is next time: 00:00:25:000
State for model input_reader_loaded is next time: 00:00:07:000
State for model input_reader_unloaded is next time: 00:00:07:000
State for model control1 is :
        phase: init   sending: 1   fin: 1
        total requests: 2   current prepared materials: 1
00:00:12:000
State for model input_reader_start is next time: 00:00:25:000
State for model input_reader_loaded is next time: 00:00:07:000
State for model input_reader_unloaded is next time: 00:00:07:000
State for model control1 is :
        phase: init   sending: 0   fin: 0
        total requests: 2   current prepared materials: 1
```

Similar to our previous atomic test cases, invalid inputs such as ***loaded*** message in the *prep* state or ***unloaded*** messages in the *init* state terminate the program with assert statements such as "*C - loaded messages only allowed in init mode*" and "*C - unloaded messages only allowed in prep mode*", respectively. Additionally, Incorrect message formats also generate similar assert statements such as "*C - invalid input from S, material storage cannot be empty while material has not moved yet*" or "*C - invalid input from S, material storage cannot be full while material has already been moved*".

All of the test cases mentioned above are accompanied by corresponding input files in the *project_folder/input_data/* folder and the complete outputs of the tests can be seen in the *project_folder/simulation_results* folder after running the simulations. Since the previous results will be overwritten when new simulations are performed, we move all simulations done before submitting the project to a new folder named “old_results” inside the same directory (*project_folder/simulation_results).*

TEST: MCCS Simulator – Coupled/Top

Finally, we simulate and observe the behavior of the MCCS simulator coupled model for verification. The MCCS, as an abstract entity, only receives ***start*** requests from external entities and responds with ***matPrepared*** and ***end*** messages. All of these messages are of type *int*, and the conditions for generating each output are exactly the same as described in the Control Agent C test case (see page 19).

We prepare only one input file:

1. *MCCS_input_test_startIn.txt (*for ***start*** messages*)*

We test a case where we have three ***start*** messages but this time the latter two cases overlap as these messages arrive before the previous ***start*** request could be completed. The input file is located in *project_folder/input_data/*. The contents of the input files are as follows:

```
00:00:05 2
00:00:10 1
00:00:20 3
```

After compilations, running **CONTROL_TEST.exe input_file_path** (located in *project_folder/bin*) should create the log file for output messages (*MCCS_main_test_output_messages.txt*) as well as output states (*MCCS_main_test_output_state.txt*) in the *project_folder/simulation_results* folder. As an example, using the command (without the quotes)

"**./MCCS.exe ../input_data/MCCS_input_test_startIn.txt**"

should generate the following output messages:

Output Messages:

```
00:00:05:000
[{2}] generated by model input_reader_main_start
00:00:05:000
[loadOut: {Material unit 1 (not ready)}, prepOut: {}, matPreparedOut: {}, endOut: {}] generated by
model control1
00:00:07:000
[loadedOut: {Material unit 1 (not ready)}, unloadedOut: {}] generated by model storage1
00:00:07:000
[loadOut: {}, prepOut: {Material unit 1 (not ready)}, matPreparedOut: {}, endOut: {}] generated by
model control1
00:00:10:000
[{1}] generated by model input_reader_main_start
00:00:12:000
[unloadOut: {Material unit 1 (ready)}] generated by model handling1
00:00:12:000
[loadedOut: {}, unloadedOut: {Material unit 1 (ready)}] generated by model storage1
00:00:12:000
[loadOut: {Material unit 2 (not ready)}, prepOut: {}, matPreparedOut: {1}, endOut: {}] generated by
model control1
00:00:19:000
[loadOut: {Material unit 3 (not ready)}, prepOut: {}, matPreparedOut: {2}, endOut: {}] generated by
model control1
00:00:20:000
[{3}] generated by model input_reader_main_start
00:00:26:000
[loadOut: {Material unit 4 (not ready)}, prepOut: {}, matPreparedOut: {3}, endOut: {}] generated by
model control1
00:00:33:000
[loadOut: {Material unit 5 (not ready)}, prepOut: {}, matPreparedOut: {4}, endOut: {}] generated by
model control1
00:00:40:000
[loadOut: {Material unit 6 (not ready)}, prepOut: {}, matPreparedOut: {5}, endOut: {}] generated by
model control1
00:00:47:000
[loadOut: {}, prepOut: {}, matPreparedOut: {6}, endOut: {1}] generated by model control1
```

The first ***start*** message with a request to produce two units of material arrives five seconds after the simulation start. After all the internal transitions within the MCCS coupled model, the resulting material is expected to be prepared after seven seconds (time taken to load and move the material). However, before we reach twelve seconds when that can happen, another ***start*** request arrives from outside entities at ten seconds. Despite this new input, we can see that the preparation of the material unit 1 continues until 12 seconds when the Control Agent simultaneously outputs *1* through ***matPrepared*** and creates a new ***load*** request for material unit 2.

After the preparation of the first unit of material, we abridge the output messages by removing all the internal output messages to provide a better visualization experience. We can now only see the inputs to and outputs from the MCCS directly. Another ***start*** message requesting the preparation of three additional units arrives at 20 seconds. The total number of materials requested to be prepared now becomes six. All of these materials are prepared in a steady manner by the MCCS, and we can see the resulting material unit numbers through ***matPrepared*** after every seven seconds at 19, 26, 33, 40, and 47 seconds respectively. Material unit 6 is the final material to be prepared, and as there are no more ***start*** requests to respond to, the MCCS sends out an ***end*** signal, and the entire MCCS system becomes idle 47 seconds after the beginning of the simulation.

The output states until the first unit of material preparation is shown below.

Output States:

```
00:00:00:000
State for model control1 is :
        phase: idle   sending: 0   fin: 0
        total requests: 0   current prepared materials: 0
State for model storage1 is :
        phase: empty   sending: 0   load requests received: 0   unload requests received: 0
State for model handling1 is :
        phase: passive   sending: 0   index: 0
State for model input_reader_main_start is next time: 00:00:05:000
00:00:05:000
State for model control1 is :
        phase: init   sending: 1   fin: 0
        total requests: 2   current prepared materials: 0
State for model storage1 is :
        phase: empty   sending: 0   load requests received: 0   unload requests received: 0
State for model handling1 is :
        phase: passive   sending: 0   index: 0
State for model input_reader_main_start is next time: 00:00:05:000
00:00:05:000
State for model control1 is :
        phase: init   sending: 0   fin: 0
        total requests: 2   current prepared materials: 0
State for model storage1 is :
        phase: empty   sending: 1   load requests received: 1   unload requests received: 0
State for model handling1 is :
        phase: passive   sending: 0   index: 0
State for model input_reader_main_start is next time: 00:00:05:000
00:00:07:000
State for model control1 is :
        phase: prep   sending: 1   fin: 0
        total requests: 2   current prepared materials: 0
State for model storage1 is :
        phase: full   sending: 0   load requests received: 1   unload requests received: 0
```

```
State for model handling1 is :
        phase: passive   sending: 0   index: 0
State for model input_reader_main_start is next time: 00:00:05:000
00:00:07:000
State for model control1 is :
        phase: prep   sending: 0   fin: 0
        total requests: 2   current prepared materials: 0
State for model storage1 is :
        phase: full   sending: 0   load requests received: 1   unload requests received: 0
State for model handling1 is :
        phase: active   sending: 1   index: 1
State for model input_reader_main_start is next time: 00:00:05:000
00:00:10:000
State for model control1 is :
        phase: prep   sending: 0   fin: 0
        total requests: 3   current prepared materials: 0
State for model storage1 is :
        phase: full   sending: 0   load requests received: 1   unload requests received: 0
State for model handling1 is :
        phase: active   sending: 1   index: 1
State for model input_reader_main_start is next time: 00:00:10:000
00:00:12:000
State for model control1 is :
        phase: prep   sending: 0   fin: 0
        total requests: 3   current prepared materials: 0
State for model storage1 is :
        phase: full   sending: 1   load requests received: 1   unload requests received: 1
State for model handling1 is :
        phase: passive   sending: 0   index: 1
State for model input_reader_main_start is next time: 00:00:10:000
00:00:12:000
State for model control1 is :
        phase: init   sending: 1   fin: 1
        total requests: 3   current prepared materials: 1
State for model storage1 is :
        phase: empty   sending: 0   load requests received: 1   unload requests received: 1
State for model handling1 is :
        phase: passive   sending: 0   index: 1
State for model input_reader_main_start is next time: 00:00:10:000
00:00:12:000
State for model control1 is :
        phase: init   sending: 0   fin: 0
        total requests: 3   current prepared materials: 1
State for model storage1 is :
        phase: empty   sending: 1   load requests received: 2   unload requests received: 1
State for model handling1 is :
        phase: passive   sending: 0   index: 1
State for model input_reader_main_start is next time: 00:00:10:000
```

The MCCS simulator was also tested with a varied set of inputs, and in every case, the simulator produced results that matched our system specifications. Other test cases can be checked by either changing the contents of the input file *../input_data/MCCS_input_test_startIn.txt* in the same format, or creating a new input file to be used as an argument while running the **MCCS.exe** executable.

Conclusion

To conclude, the MCCS simulator was specified using the DEVS formalism, and the simulator was created in Cadmium for a manufacturing system that prepares batches of material requests for further processing. All the atomic models were tested with the ideal as well as anomalous sets of inputs followed by simple coupled model experimentations to verify the correct and expected behavior of the model. The simulations produced expected results in all cases, and as such, no changes were made to the original specification.