SYSC 5104 – Methods of Discrete Event Modelling

Assignment 1

O-Train DEVS in Cadmium

Author:
Zachary Dunnigan

Student Number:
100892725

Date Submitted:
October 31, 2019

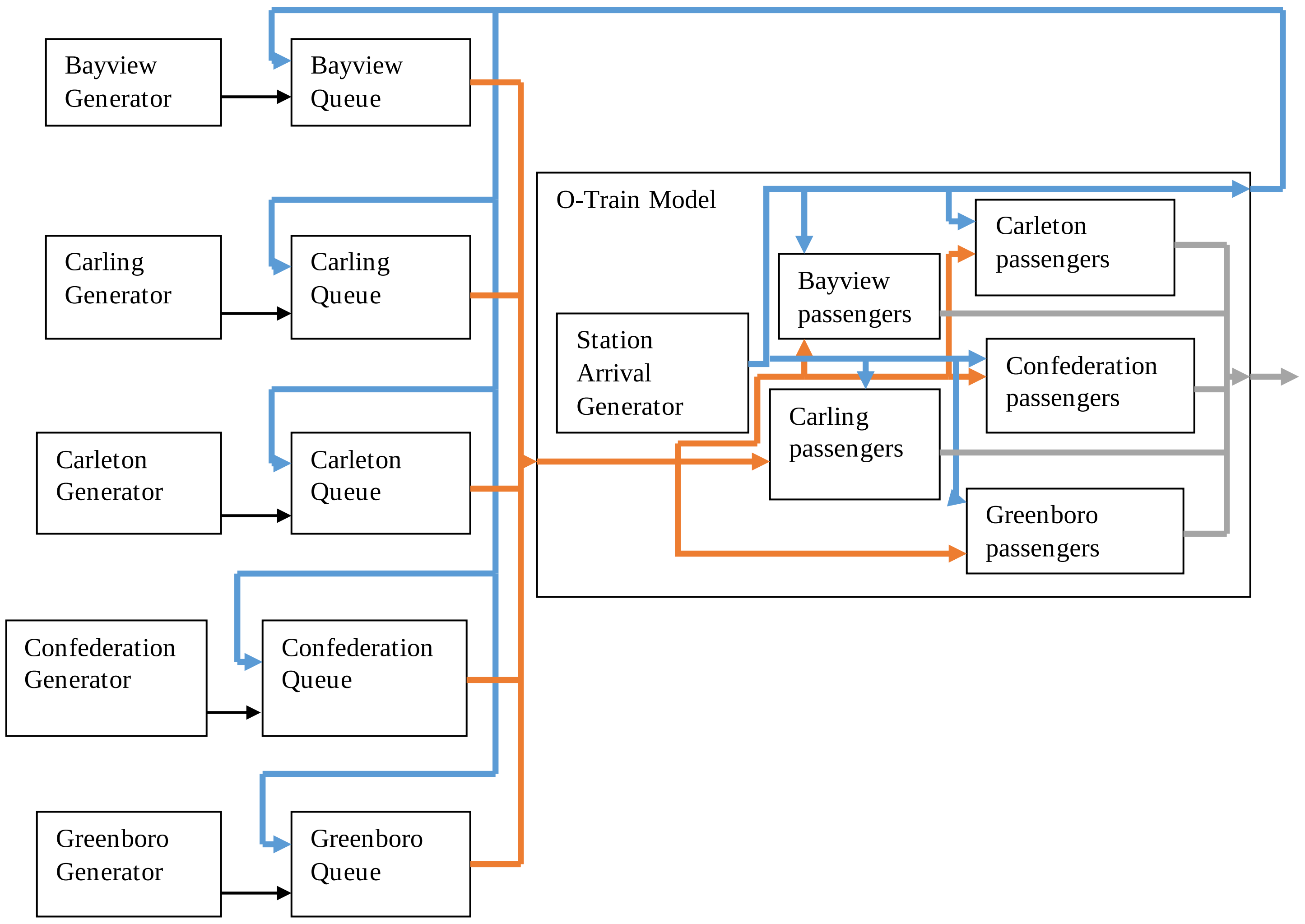
Table of Contents

1.0 Model Choice & Description

The CD++ model selected for this assignment is the O-Train Trip Time Analysis model (or otrain for short). This model has been converted to a cadmium model, improved upon, and tested. The basic structure of the model is identical to its original implementation.

Each station along the O-Train track is represented as a queue with an associated passenger generator. The passenger generators output unique passenger IDs that identify the passenger number, their origin station, and their destination station. The frequency of passenger generation is controlled using a varying time interval determined by a statistical distribution. Once a passenger is generated, they enter a queue at their station. Passenger queues simply hold the passenger IDs in a list while waiting for the train to arrive. The O-Train itself is a coupled model composed of a station arrival generator and a series of destination queues. The station arrival generator outputs a message when the train arrives at the next station. The timing of arrivals is pre-determined by the train's schedule; this has been approximated to roughly 3 and a half minute lapses between stations. The passenger queues within the O-Train correspond to destinations. Each passenger that boards the train enters a queue associated with their destination. Once the destinations station is reached, the associated on-train queue empties and the passengers leave the train.

This model is useful for analyzing the total passenger trip time defined from the time a passenger is generated at a station to arriving at their destination. Using this model, the average trip time of a rider can be extracted. The diagram below illustrates the conceptual model for the train system with input /output relationships shown using arrows.



The top model contains 4 types of atomic models and 1 coupled model. There are also 2 different types of data structures corresponding to passenger ID's and station numbers. Atomic models, coupled models, and data structures used by this simulation are listed below.

Atomic models:

- Station Arrival Generator
- In-Train Queue
- Station Passenger Generator
- Station Queue

Coupled models:

- O-train
- Top model

Data Structures

- Station Number → 2 integers, one for direction of train (north/south), and one for station the train is arriving at
- Passenger ID → integer holding a unique passenger ID

The station number data structure has two integers. The first is a binary integer that represents the direction of travel (0 for South, 1 for North). The second number is an integer that corresponds to one of the 5 possible station IDs: Bayview, Carling, Carleton, Confederation, and Greenboro. Since there is only one possible direction for the end stations (Bayview and Greenboro), there are 8 possible unique station numbers shown in the table below. For some of the atomic models, the direction is not an important parameter and only the station ID is considered.

Name	Direction	Station ID
Bayview	0	1
CarlingS	0	2
CarletonS	0	3
ConfedS	0	4
Greenboro	1	5
ConfedN	1	4
CarletonN	1	3
CarlingN	1	2

Passenger ID's are a concatenation of 3 integers to form a single integer with information about the passenger number, their origin station, and their destination station. The structure of a passenger ID is shown below with 2 examples.

PassengerID = xyz
x is the passenger number, unique to each passenger generator
y is the passenger's origin station ID
z is the passenger's destination station ID
numbers are concatenated by x*100 + y*10 + z

examples: PassengerID = 241 → passenger #2 generated at Confederation Station, going to Bayview
PassengerID = 1512 → passenger #15 generated at Bayview Station, going to Carling

2.0 DEVS Formalism for Atomic & Coupled Models

This section of the report is divided into several sub-sections. First, the atomic model formalisms will be described (2.1 to 2.4) followed by the coupled model formalisms (2.5 and 2.6). In the next Section 3.0, the experimental framework is discussed including tests for each of the atomic models. A coupled model test of the O-Train is performed before combining it with the top model for final simulations and testing.

2.1 Station Arrival Generator Atomic Model

This model is a deterministic output generator and is nearly identical to the CD++ atomic model in behaviour. It consistently loops through outputting the station the train has arrived at. Because the train runs on a single line north and south, the order of station arrival outputs is always the same and repeats in a cycle. There are no inputs for this model. The model has been marginally improved from its original CD++ implementation. The original model began with the train in transit and time needed to pass before the first train arrival. In the updated model, the train initializes immediately at the default station (Bayview) and then begins its first inter-station trip. This means in the case that passengers are already present when the simulation begins, they made board immediately. Round trip time for the train to complete one lap of the track is 30 minutes.

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

State Variables:

```
state = active;               /* always active while simulation running */
sigma = 0;                    /* begin immediately */
Bayview = true;               /* assumed initial destination station */
CarlingS = CarliingN = CarletonS = CarletonN = ConfedS = ConfedN = Greenboro = false;
initializing = true;           /* allows O-Train to arrive at default station immediately when sim begins */
```

Formal Specification:

X = ϕ;
Y = { (StationNumberOut, *N*); };
S = { state ∈ {active, passive},
    sigma ∈ {NDTime},
    Bayview ∈ {true, false},
    CarlingS ∈ {true, false},
    CarlingN ∈ {true, false},
    CarletonS ∈ {true, false},
    CarletonN ∈ {true, false},
    ConfedN ∈ {true, false},
    ConfedS ∈ {true, false},
    Greenboro ∈ {true, false},
    initializing ∈ {true, false} }

```
δint(s){
case state
       active:
              if (Bayview){
                     Bayview = false;
                     CarlingS = true;
              else if (CarlingS)
                     CarletonS = false;
                     CarletonS = true;
              else if (CarletonS)
                     CarletonS = false;
                     ConfedS = true;
              else if (ConfedS)
                     ConfedS = false;
                     Greenboro = true;
              else if (Greenboro)
                     Greenboro = false;
                     ConfedN = true;
```

```
                else if (ConfedN)
                        ConfedN = false;
                        CarletonN = true;
                else if (CarletonN)
                        CarletonN = false;
                        CarlingN = true;
                else if (CarlingN)
                        CarlingN = false;
                        Bayview = true;
                        }
        passive:
        /* Never happens, generator is always active while simulation running and ta = ∞ if passive */
}
```

```
λ(s){
        if (Bayview){
                send 01 to port StationNumberOut;
        else if (CarlingS)
                send 02 to port StationNumberOut;
        else if (CarletonS)
                send 03 to port StationNumberOut;
        else if (ConfedS)
                send 04 to port StationNumberOut;
        else if (Greenboro)
                send 15 to port StationNumberOut;
        else if (ConfedN)
                send 14 to port StationNumberOut;
        else if (CarletonN)
                send 13 to port StationNumberOut;
        else if (CarlingN)
                send 12 to port StationNumberOut;
        }
}
```

```
ta(s){
        if (initializing == true) {           // This case is the improved code update
                sigma = TIME("00:00:00:000"); // If the train has just initialized, output current station now
        else if (state == active){
                sigma = TIME("00:03:45:000"); // Inter-station travel time is approximately 3 mins 45 sec
        else if (state == passive)
                sigma =  ∞;
        }
}
```

2.2 In-Train Queue Atomic Model

This model has near identical behavior to the original CD++ atomic model but has been improved to handle a case not accounted for in the original implementation. The cadmium atomic model has been modified such that if a passenger has a destination station that is the same as their start station, the passenger can enter the train, be added to the back of current passenger queue which is emptying, and will proceed to immediately exit the train. In other words, the passenger will get on the train and get right back off. This should never happen due to the improved implementation of the Station Passenger Generator and the Station Queue, but if it does, the model will behave appropriately.

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

State Variables:

```
sigma = ∞;                             /* wait for input */
state = passive;                       /* initially passive, waits for input */
FillQueue = false;                     /* initially passive, queue cannot be filled */
EmtptyQueue = false;                   /* initially passive, queue cannot be emptied */
StationDest = var;                     /* var is the destination station variable passed by coupled model */
PassengerQueue = Empty(*PassengerID); /* contains list of Passenger IDs in queue on train */
```

Formal Specification:

X = { (StationNumberIn, N), (PassengerIn, N) };
Y = { (PassengerOut, N) };
S = { state ∈ {active, passive},

sigma ∈ {NDTime},
FillQueue ∈ {true, false},
EmptyQueue ∈ {true, false},
StationDest ∈ {1, 2, 3, 4, 5},
PassengerQueue ∈ {PassengerID ∈ N} }

```
δ_ext(s,e,x){
case state
        passive:
                if (PassengerIn)              /* Passenger is boarding */
                        if (value(PassengerIn)%10 == StationDest){
                                state = active;
                                FillQueue = true;
                                Add PassengerIn to PassengerQueue
                        }
                }

                if (StationNumberIn)   {      /* Train arrives at station, queue must empty */
                        if (value(StationNumberIn)%10 == StationDest &&  size(PassengerQueue) > 0){
                                state = active;
                                EmptyQueue = true;
                        }
                }

        active:
        /* Should never have passenger being added to queue when queue is emptying but just in case */
        /* In this case a person gets on the train and gets back off immediately */
                if (value(PassengerIn)%10 == StationDest){
                        Add PassengerIn to PassengerQueue
                        Allow PassengerQueue to continue emptying with new passenger added to back of queue
                        sigma = sigma - e;
                }
}
```

```
δ_int(s){
case state
        active:
                if (EmptyQueue){
                        Delete first Passenger in PassengerQueue
                        if (PassengerQueue.size() > 0){
                                state = active
                                EmptyQueue = true;
                        else
                                state = passive;
                                FillQueue = false;
                                EmptyQueue = false;
                        }
                }

                if (FillQueue){
                        state = passive;
                        FillQueue = false;
                        EmptyQueue = false;
                }

        passive:
        /* Never happens, if passive sigma = ∞ */
}
```

```
λ(s){
        if (EmptyQueue && PassengerQueue.size() > 0){
                send first Passeneger in PassengerQueue to port PassengerOut;
        }
}
```

```
ta(s){
        if (state == active){
                if (FillQueue){
                        sigma = TIME("00:00:00:000");
                else if (EmptyQueue)
                        sigma = TIME("00:00:00:025");
                }
        else if (state == passive)
                sigma =  ∞;
        }
}
```

2.3 Station Passenger Generator Atomic Model

This model behaves the same as the CD++ atomic model; passenger IDs are generated over a certain wait time varying by a normal distribution with mean of 5 minutes and standard deviation of 5 minutes. Passenger destinations are generated as integers between 1 and 5 using a uniform distribution. The passenger generator is assigned to a station using a station ID number via a variable passed during instantiating in the top model.

Small improvements to this model include enforcing that wait times between generated passengers is never negative or greater than 10 minutes. Additionally, if a station destination is generated which is identical to the start station number, the destination is regenerated immediately until a valid destination is created. This improves on the previous functionality in CD++ of the passenger generator outputting nothing if destination and start station happen to be the same. A 500 millisecond delay has been added when the generator initializes as a safeguard. This ensures that a passenger is not outputted to a station queue at the same time the train arrives. Simultaneous inputs to the station queue would cause issues during the simulation. Train arrivals always occur at times with no fractions of a second present. The initial 500 millisecond delay for passenger generation ensures that passengers are always generated at a time either before or after the train arrives.

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

State Variables:

```
state = active;               /* should never be passive */
WaitTime = 0                  /* how often a passenger is generated depends on distribution */
sigma = 0      ;              /* generates a passenger every time WaitTime elapses */
StationStartNumber = var;     /* varies by station, var is passed to atomic instance from coupled model */
Station = 0;                  /* a number that will be used to determine the destination station */
PassengerNumber = 0;          /* keeps track of total passengers generated at a given station */
PassengerOutput = 0;          /* unique passenger identifier to be outputted */
```

Formal Specification:

X = ϕ;
Y = { (PassengerInitial, *N*) };
S = { state ∈ {active, passive},
 sigma ∈ {NDTime},
 WaitTime ∈ *Double*,
 StationStartNumber ∈ {1, 2, 3, 4, 5},
 Station ∈ *N*,
 PassengerNumber ∈ *N*,
 PassengerOutput ∈ *N* }

```
δint(s){
case state
        active:
                /* Increment Passenger Number */
                PassengerNumber++;

                /* Get distributions for next wait time and destination station */
                WaitTime = getNormDistribution;       /* normal dist, mean = 5, std dev = 5 * /
                Station = getUniformDistribution;     /* uniform dist, integer between 1 and 5 */
                while (WaitTime<=0 || WaitTime>=10){
                        WaitTime = getNormDistribution;
                }                                            // These while loops are improvements to original
                while (Station ==  StationStartNumber){
                        Station = getUniformDistribution;
                }

                if (Station == 1){     /* Destination: Bayview */
                        PassengerOutput = PassengerNumber*100+StationStartNumber*10+1;
                else if (Station == 2) /* Destination: Carling */
                        PassengerOutput = PassengerNumber*100+StationStartNumber*10+2;
                else if (Station == 3) /* Destination: Carleton */
                        PassengerOutput = PassengerNumber*100+StationStartNumber*10+3;
                else if (Station == 4) /* Destination: Confed */
                        PassengerOutput = PassengerNumber*100+StationStartNumber*10+4;
                else                    /* Destination: Greenboro */
                PassengerOutput = PassengerNumber*100+StationStartNumber*10+5;
                }

        passive:
        /* Never happens, generator is always active while simulation running */
}
```

```
λ(s){
        if (PassengerOutput%10 == StationStartNumber || PassengerNumber == 0){
                do nothing
                /* passenger is already at their destination, no need to output to queue */
                /* or the generator is initializing and there is no passenger to output */
        else
                send PassengerOutput to port PassengerInitial
        }
}
```

```
ta (s){
        if (PassengerNumber == 0){
                /* to avoid a simultaneous input with StationArrivalGenerator which outputs immediately, */
                /* the StationPassengerGenerator has a 0.5 second initialization delay */
                sigma = TIME(“00:00:00:500”)
        else if (state == active){
                sigma = TIME(“00:WaitTime:00:000”)
        else if (state == passive)
                / * Never happens, generator is always active until ST expires * /
        }
}
```

2.4 Station Queue Atomic Model

This atomic model is very similar to the in-train queue atomic model, however, it receives passenger IDs from the station passenger generators. Station queues have an input for passenger IDs from the passenger generators and an input for a station number from the station arrival generator. The model has a single output for passenger IDs to board the train once it arrives. A station queue is assigned a station ID and only accepts passenger ID’s with a matching origin station ID. For example, this means that passengers from a generator at Bayview station cannot enter the queue at Confederation station. Destination stations ate irrelevant to station queues except in the case of identical origin and destination stations. In such cases the passenger is already at their destination.

A major difference between the station queue and in-train queue is that this model needs to be able to handle the case of a valid passenger input being received while the queue is emptying; the original CD++ model ignored this. The atomic model implementation has been improved so that in such a scenario: emptying the queue (train boarding) is paused, the passenger is added to the end of the queue, and then the emptying process proceeds.

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

State Variables:

```
sigma = ∞;                              /* waits for input */
state = passive;                        /* initially passive, waiting for input */
FillQueue = false;                      /* initially passive, queue cannot be filled */
EmtptyQueue = false;                    /* initially passive, queue cannot be emptied */
StationDest = var;                      /* var is the station location variable passed by coupled model */
StationQueue = Empty(*PassengerID);     /* contains list of Passenger IDs in queue at station */
```

Formal Specification:

X = { (StationNumberIn, *N*), (PassengerIn, *N*) };
Y = { (PassengerOut, *N*) };
S = { state ∈ {active, passive},
 sigma ∈ {NDTime},
 FillQueue ∈ {true, false},
 EmptyQueue ∈ {true, false},

StationDest ∈ {1, 2, 3, 4, 5},
StationQueue ∈ {PassengerID ∈ N} }

```
δext(s,e,x){
case state
        passive:
                if (PassengerIn && !EmptyQueue){      /* Passenger arrives at station*/
                        if (value(PassengerIn)%10 != StationDest
                        && (value(PassengerIn)%10 - value(PassengerIn)%100)/10 == StationDest){
                                state = active;
                                FillQueue = true;
                                Add PassengerIn to StationQueue
                        }
                }

                if (StationNumberIn)   {       /* Train arrives at station, queue must empty */
                        if (value(StationNumberIn)%10 == StationDest &&  size(StationQueue) > 0){
                                state = active;
                                EmptyQueue = true;
                        }
                }

        active:
        /* If this state occurs, a passenger input to the queue occurred while the queue was emptying*/
                if (value(PassengerIn)%10 != StationDest
                && (value(PassengerIn)%10 - value(PassengerIn)%100)/10 == StationDest){
                        Add PassengerIn to StationQueue
                        Allow StationQueue to continue to empty with new passenger added to back of queue
                        sigma = sigma – e;
                }
}
```

```
δint(s){
case state
        active:
                if (EmptyQueue){
                        Delete first Passenger in StationQueue
                        FillQueue = false;
                        if (StationQueue.size() > 0){
                                state = active
                                EmptyQueue = true;
                        else
                                state = passive;
                                EmptyQueue = false;
                        }
                }

                if (FillQueue){
                        state = passive;
                        FillQueue = false;
                        EmptyQueue = false;
                }

        passive:
        /* Never happens, if passive sigma = ∞ */
}
```

```
λ(s){
        if (EmptyQueue && StationQueue.size() > 0){
                send first Passeneger in StationQueue to port PassengerOut;
        }
}
```

```
ta(s){
        if (state == active){
                if (FillQueue){
                        sigma = TIME(“00:00:00:000”);
                else if (EmptyQueue)
                        sigma = TIME(“00:00:00:025”);
                }
        else if (state == passive)
                sigma =  ∞;
        }
}
```

2.5 O-Train Coupled Model

O-Train coupled model is the Cadmium equivalent of the CD++ model, no major changes. Formal specification shown below. Connections follow the same flow chart shown in the conceptual model.

OTrain= < X, Y, D, {M_i}, IC, EIC, EOC, select >

Formal Specification:

X = { (PassengerIn, *N*) };
Y = { (PassengerOut, *N*), (StationNumber, *N*) };
D = { StationArrivalGeneraor,
InTrainQueue_Bayview,
InTrainQueue_Carling,
InTrainQueue_Carleton,
InTrainQueue_Confed
InTrainQueue_Greenboro };
M_i = {M_{SAG}, M_{ITQ_Bay}, M_{ITQ_Carli}, M_{ITQ_Carlt}, M_{ITQ_Con}, M_{ITQ_Gre}};
IC = { StationNumberOut@StationArrivalGenerator → StationNumberIn@InTrainQueue_Bayview,
StationNumberOut@StationArrivalGenerator → StationNumberIn@InTrainQueue_Carling,
StationNumberOut@StationArrivalGenerator → StationNumberIn@InTrainQueue_Carleton,
StationNumberOut@StationArrivalGenerator → StationNumberIn@InTrainQueue_Confed,
StationNumberOut@StationArrivalGenerator → StationNumberIn@InTrainQueue_Greenboro };
EIC = { PassengerIn → PassengerIn@InTrainQueue_Bayview,
PassengerIn → PassengerIn@InTrainQueue_Carling,
PassengerIn → PassengerIn@InTrainQueue_Carleton,
PassengerIn → PassengerIn@InTrainQueue_Confed,
PassengerIn → PassengerIn@InTrainQueue_Greenboro };
EOC = { StationNumberOut@StationArrivalGenerator → StationNumber,
PassengerOut@InTrainQueue_Bayview → PassengerOut,
PassengerOut@InTrainQueue_Carling → PassengerOut,
PassengerOut@ InTrainQueue_Carleton → PassengerOut
PassengerOut@ InTrainQueue_Confed → PassengerOut
PassengerOut@ InTrainQueue_Greenboro → PassengerOut };

2.6 Top Coupled Model

Top coupled model is the Cadmium equivalent of the CD++ model, no major changes. Formal specification shown below. Connections follow the same flow chart shown in the conceptual model.

Top = < X, Y, D, {M_i}, IC, EIC, EOC, select >

X = ϕ;
Y = { (PassengerArrived, *N*) };
D = { OTrain,
StationPassengerGenerator_Bayview,
StationPassengerGenerator_Carling,
StationPassengerGenerator_Carleton,
StationPassengerGenerator_Confed
StationPassengerGenerator_Greenboro
StationQueue_Bayview,
StationQueue_Carling,
StationQueue_Carleton,
StationQueue_Confed,

StationQueue_Greenboro, };
M_i = {M_{OT}, M_{SPG_Bay}, M_{SPG_Carli}, M_{SPG_Carlt}, M_{SPG_Con}, M_{SPG_Gre}, M_{SQ_Bay}, M_{SQ_Carli}, M_{SQ_Carlt}, M_{SQ_Con}, M_{SQ_Gre} };
IC = { PassengerInitial@StationPassengerGenerator_Bayview → PassengerIn@StaionQueue_Bayview,
PassengerInitial@StationPassengerGenerator_Carling → PassengerIn@StaionQueue_Carling,
PassengerInitial@StationPassengerGenerator_Carleton → PassengerIn@StaionQueue_Carleton,
PassengerInitial@StationPassengerGenerator_Confed → PassengerIn@StaionQueue_Confed,
PassengerInitial@StationPassengerGenerator_Greenboro → PassengerIn@StaionQueue_Greenboro,
PassengerOut@StaionQueue_Bayview → PassengerIn@OTrain,
PassengerOut@StaionQueue_Carling → PassengerIn@OTrain,
PassengerOut@StaionQueue_Carleton → PassengerIn@OTrain,
PassengerOut@StaionQueue_Confed → PassengerIn@OTrain,
PassengerOut@StaionQueue_Greenboro → PassengerIn@OTrain,
StationNumber@OTrain → StationNumberIn@StationQueue_Bayview,
StationNumber@OTrain → StationNumberIn@StationQueue_Carling,
StationNumber@OTrain → StationNumberIn@StationQueue_Carleton,
StationNumber@OTrain → StationNumberIn@StationQueue_Confed,
StationNumber@OTrain → StationNumberIn@StationQueue_Greenboro };
EIC = ϕ;
EOC = { PassengerOut@OTrain → PassengerArrived};

The next section will explain the experimental framework of each atomic / coupled model. The tests to be carried out and specific cases of interest will be discussed. The results of the tests are shown and model functionality is validated for all atomic, coupled, and top models.

3.0 Experimental Framework and Testing

The original documentation only tested atomic models and the top model. This report has some intermittent testing that expands on the original tests performed in CD++ documentation. Both generator atomic models have no inputs and testing is performed by observing the outputs over time and confirming that all are valid. The queues are more complex due to multiple inputs. Due to this, queue testing is performed in 3 phases to look for specific cases that may cause issues and to confirm the operation of the models. The coupled O-train model is also tested in 3 phases before proceeding to the final top-level simulation

Experiments to be tested:

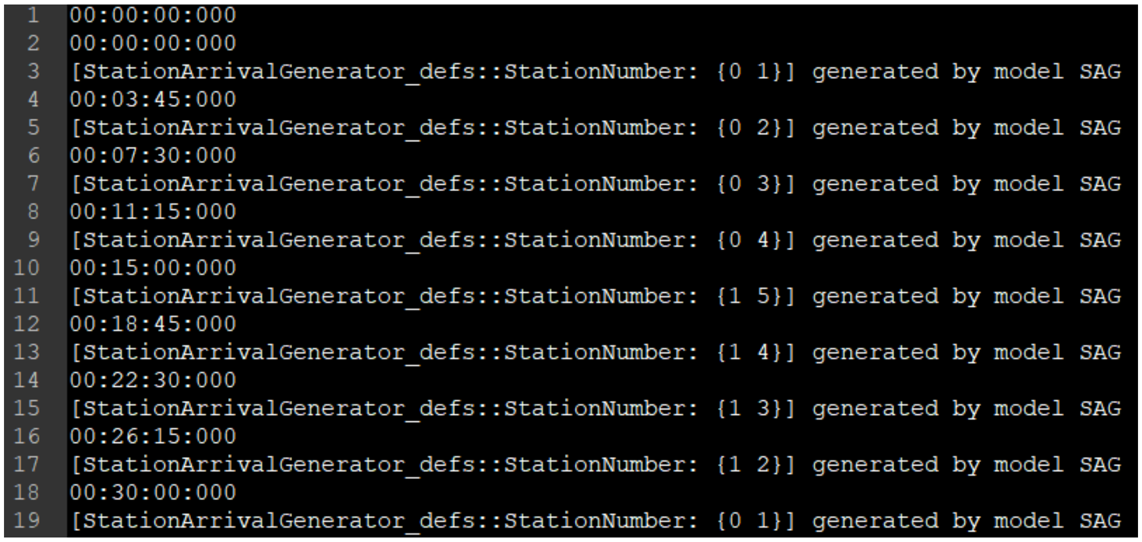
- Each atomic model is tested individually:
  - Station Arrival Generator (no inputs needed)
  - Station Passenger Generator (no inputs needed)
  - Station Queue (input files for PassengerIn and StationNumberIn)
  - In-Train Queue (input files for PassengerIn and StationNumberIn)
- One coupled model test:
  - O Train Coupled model (input file for PassengerIn)

3.1 Station Arrival Generator Atomic Test

The station arrival generator outputs the station the O-Train is arriving at in a deterministic order based on the train's schedule. According to the original CD++ model, the train takes 15 minutes to travel one length of the track from Bayview to Greenboro with 5 stations in between. This means there are 4 intervals in total between each station with an approximate gap averaging 3 minutes and 45 seconds. Due to the generator having no inputs, the testing is simply to let the generator run for a fixed amount of time. The order and timing of the station outputs must match the O-Trains schedule.

In the previous CD++ model, the train initializes from Bayview station travelling south but does not actually output that it was at Bayview station for time t = 0. In the improved model the train should output the station code for Bayview at t = 0 and arrive at Greenboro at exactly t = 15 minutes. All station arrivals in between should be announced in the correct order with 3 minute 45 seconds gaps in between.

The model was run for a total time of 1 hour to observe these outputs. The Cadmium generated outputs of the Station Arrival Generator model (or SAG for short) are shown below.



```
00:00:00:000
00:00:00:000
[StationArrivalGenerator_defs::StationNumber: {0 1}] generated by model SAG
00:03:45:000
[StationArrivalGenerator_defs::StationNumber: {0 2}] generated by model SAG
00:07:30:000
[StationArrivalGenerator_defs::StationNumber: {0 3}] generated by model SAG
00:11:15:000
[StationArrivalGenerator_defs::StationNumber: {0 4}] generated by model SAG
00:15:00:000
[StationArrivalGenerator_defs::StationNumber: {1 5}] generated by model SAG
00:18:45:000
[StationArrivalGenerator_defs::StationNumber: {1 4}] generated by model SAG
00:22:30:000
[StationArrivalGenerator_defs::StationNumber: {1 3}] generated by model SAG
00:26:15:000
[StationArrivalGenerator_defs::StationNumber: {1 2}] generated by model SAG
00:30:00:000
[StationArrivalGenerator_defs::StationNumber: {0 1}] generated by model SAG
```

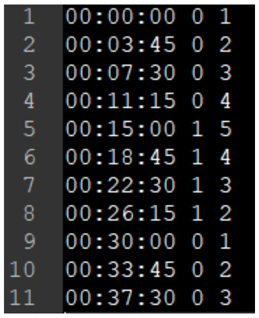
Figure 1: Station arrival generator test outputs for first 30 minutes.

In Figure 1 we can observe the outputs of the atomic model generated by Cadmium. The model is behaving as expected with the deterministic outputs of the train's arrival schedule. At time t = 0 an output is generated showing that the train has arrived at Bayview station and is heading South. Recall that the first integer refers to the train's direction (0 for South and 1 for North). As expected, at time t = 15 minutes the train arrives as Greenboro station and outputs the integers { 1 5 } indicating it is now heading North. The O-Train completes 1 lap of the linear track at time t = 30 minutes when it arrives back at Bayview station. All outputs and timing during the trip are as expected.

3.2 In-Train Queue Atomic Test

The in-train queue models are more complex than the station generator as they have 2 inputs as well as an output. Inputs include the station number where the train is arriving and passenger IDs to represent people boarding the train. Each in-train queue is specific to a unique destination station; the only passengers that should be added to a given queue are those with destination stations that match the destination station for the queue. For example, passengers heading to Carling station should not be allowed to enter the Greenboro in-train queue. When a station arrival input indicates that that the train has arrived at the station corresponding to that queue; passengers in the queue exit the train. This is represented by them being removed from the queue and their passenger IDs outputted. Passengers deboard the train with an initial 25 millisecond delay in order to allow passengers waiting at the current station to board immediately. There is an additional 25 millisecond delay between each deboarding passenger.

Testing will involve an input file to represent the station arrival generator that mimics the deterministic output behavior of that model. It provides the arriving station number at a specific time based on the trains schedule. The input file is identical to the outputs from test in the previous section (refer to Figure 2 below).

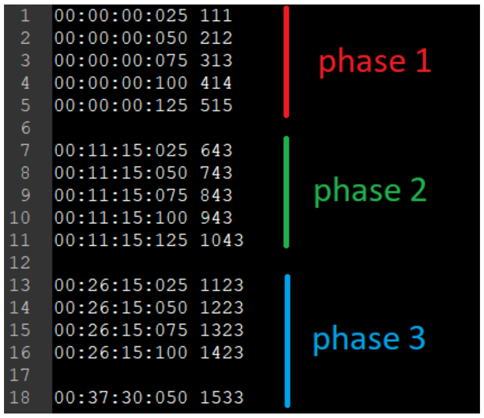


```
00:00:00 0 1
00:03:45 0 2
00:07:30 0 3
00:11:15 0 4
00:15:00 1 5
00:18:45 1 4
00:22:30 1 3
00:26:15 1 2
00:30:00 0 1
00:33:45 0 2
00:37:30 0 3
```

Figure 2: In-Train Queue test – input data for station number input.

The in-train queue will be tested with destination station { 3 } corresponding to Carleton station. The only passengers who should be able to enter this queue are those with a destination station that matches the queue. Testing will be performed in 3 phases with distinct groups of passengers attempting to board. The 1st phase will have the train arrive at Bayview at t = 0, and a group of 5 passengers will be sent as inputs to the Carleton station in-train queue. Only one of the passengers has the correct destination station of Carleton and therefore 4 of the passengers should be ignored. Only the single valid passenger should be added to the queue. This passenger should then deboard the train when it first arrives at Carleton station at t = 7 minutes 30 seconds. The second phase features a group of 5 passengers waiting at Confederation station at t = 11 minutes 15 seconds. In this case, all 5 of the passengers have a destination of Carleton, therefore, all 5 should board the train. All the added passengers in the queue should deboard the train the next time it arrives at Carleton station (t = 22 minutes 30 seconds).

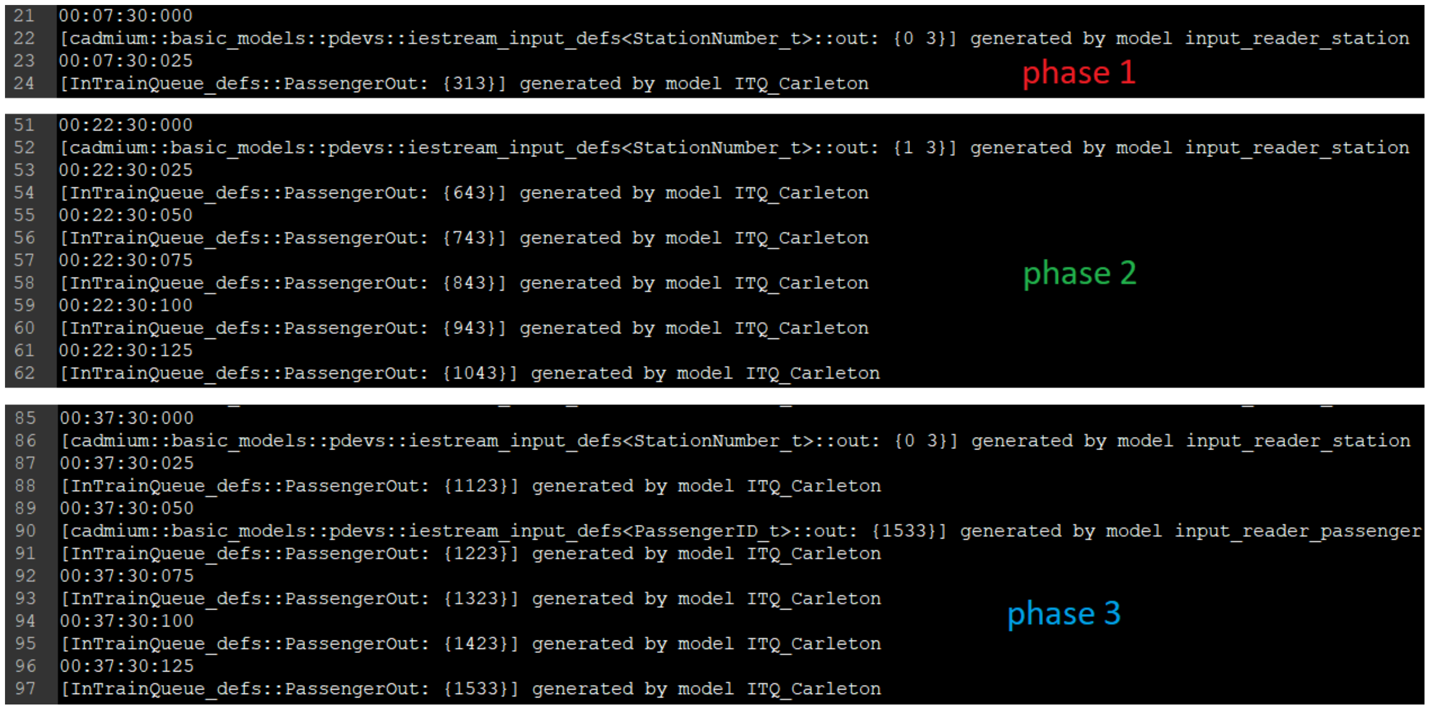
One improvement to the original model includes the case that a passenger attempts to board the train with a destination station that matches the current station the train is at. Although this should never happen due to improvements made to the station passenger generator, the simulator should be able to handle the case anyway. In the third phase of testing, a group of passengers are added to the queue with Carleton station as their destination at t = 26 minutes 15 seconds. When the train next arrives at Carleton station at t = 37:30, a passenger with identical origin and destination stations (Carleton) will attempt to board the train as the passengers in the queue exit. For this test to be successful, the passenger waiting at Carleton station should be added to the back of the queue, and then the entire queue should be emptied. The passenger ID inputs representing all 3 test phases are shown below in Figure 3.



```
00:00:00:025 111
00:00:00:050 212
00:00:00:075 313
00:00:00:100 414
00:00:00:125 515

00:11:15:025 643
00:11:15:050 743
00:11:15:075 843
00:11:15:100 943
00:11:15:125 1043

00:26:15:025 1123
00:26:15:050 1223
00:26:15:075 1323
00:26:15:100 1423

00:37:30:050 1533
```

Figure 3: In-Train Queue test – input data for passenger IDs.

The test was run for 40 minutes and the output behavior was as expected. In phase 1 of the test, the only passenger to be added to the Carleton in-train queue and outputted at Carleton station was passenger ID { 313 }. In phase 2 of the test, all valid passengers were added to the queue, and all of them successfully deboarded at the next Carleton arrival. Phase 3 was also a success: the passenger waiting at Carleton was added to the queue already containing passengers from an earlier pickup, and then all passengers deboarded. Refer to Figure 4 below for the relevant outputs (note that ITQ is short for in-train queue).



```
00:07:30:000
[cadmium::basic_models::pdevs::iestream_input_defs<StationNumber_t>::out: {0 3}] generated by model input_reader_station
00:07:30:025
[InTrainQueue_defs::PassengerOut: {313}] generated by model ITQ_Carleton
```

```
00:22:30:000
[cadmium::basic_models::pdevs::iestream_input_defs<StationNumber_t>::out: {1 3}] generated by model input_reader_station
00:22:30:025
[InTrainQueue_defs::PassengerOut: {643}] generated by model ITQ_Carleton
00:22:30:050
[InTrainQueue_defs::PassengerOut: {743}] generated by model ITQ_Carleton
00:22:30:075
[InTrainQueue_defs::PassengerOut: {843}] generated by model ITQ_Carleton
00:22:30:100
[InTrainQueue_defs::PassengerOut: {943}] generated by model ITQ_Carleton
00:22:30:125
[InTrainQueue_defs::PassengerOut: {1043}] generated by model ITQ_Carleton
```

```
00:37:30:000
[cadmium::basic_models::pdevs::iestream_input_defs<StationNumber_t>::out: {0 3}] generated by model input_reader_station
00:37:30:025
[InTrainQueue_defs::PassengerOut: {1123}] generated by model ITQ_Carleton
00:37:30:050
[cadmium::basic_models::pdevs::iestream_input_defs<PassengerID_t>::out: {1533}] generated by model input_reader_passenger
[InTrainQueue_defs::PassengerOut: {1223}] generated by model ITQ_Carleton
00:37:30:075
[InTrainQueue_defs::PassengerOut: {1323}] generated by model ITQ_Carleton
00:37:30:100
[InTrainQueue_defs::PassengerOut: {1423}] generated by model ITQ_Carleton
00:37:30:125
[InTrainQueue_defs::PassengerOut: {1533}] generated by model ITQ_Carleton
```

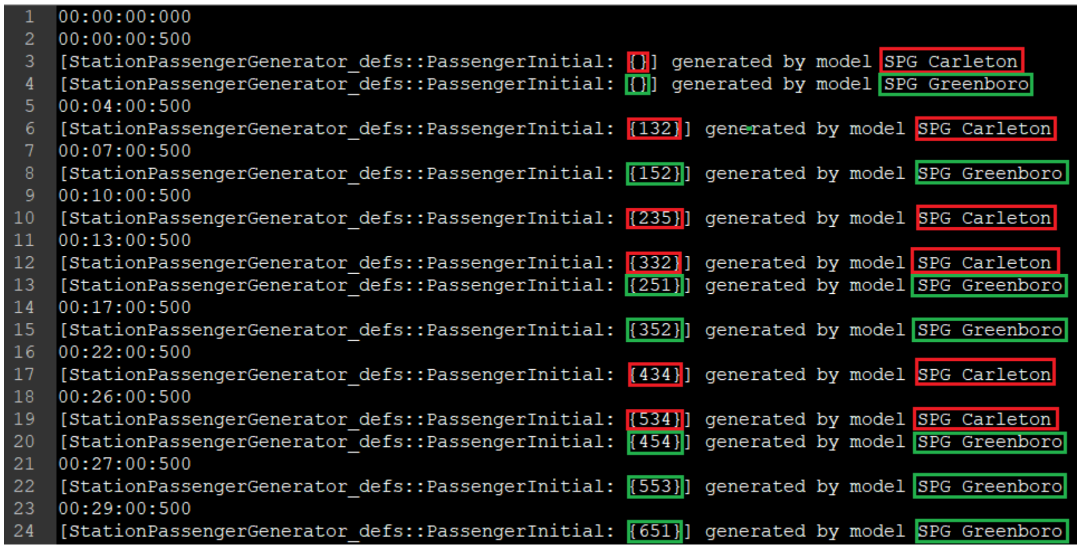
Figure 4 – Relevant in-train queue test outputs.

3.3 Station Passenger Generator Atomic Test

The station passenger generator atomic model is responsible for generating passenger IDs that include a unique passenger number, the station of origin, and the destination station of the passenger. Each station consists of one passenger generator and one station queue. The passenger generator outputs a passenger after a certain wait time which is determined by a normal distribution with a mean of 5 minutes and standard deviation of 5 minutes. The destination stations are generated using a uniform distribution with a range of 1 to 5 for each of the five possible destinations. There is an initial 500 millisecond delay before the generator begins operating to ensure that simultaneous inputs do not occur for the station queue the generator is connected to. Station passenger generators should never output a passenger with identical origin and destination stations.

This atomic model has no inputs, so testing is trivial. Two passenger generators are instantiated and arbitrarily assigned to Carleton and Greenboro stations by passing the station IDs { 3 } and { 5 } respectively to the atomic models from the top model. The simulation is run for 30 minutes and the outputs from each generator are observed and checked for validity. The passenger number within the ID should increment by 1 for every output. The origin station should always correspond to the generator's assigned station; in this case { 3 } or { 5 }. The destination stations should be random integers between 1 and 5 but should never match the origin station of a generator. Wait times should vary between passenger ID outputs and should never be less than 1 minute or greater than 10 minutes.

The two instantiated generators are named SPG_Carleton and SPG_Greenboro which are short for station passenger generator belonging to Carleton or Greenboro. Over a 30 minute period, we expect each generator to produce approximately 6 passengers due to the 5 minute normal distribution for wait times but this may vary due to randomness. Outputs of the test are shown in Figure 5 below. For visibility, the Carleton passenger generator outputs are highlighted in red and the Greenboro passenger generator outputs are highlighted in green.



```
00:00:00:000
00:00:00:500
[StationPassengerGenerator_defs::PassengerInitial: {}] generated by model SPG Carleton
[StationPassengerGenerator_defs::PassengerInitial: {}] generated by model SPG Greenboro
00:04:00:500
[StationPassengerGenerator_defs::PassengerInitial: {132}] generated by model SPG Carleton
00:07:00:500
[StationPassengerGenerator_defs::PassengerInitial: {152}] generated by model SPG Greenboro
00:10:00:500
[StationPassengerGenerator_defs::PassengerInitial: {235}] generated by model SPG Carleton
00:13:00:500
[StationPassengerGenerator_defs::PassengerInitial: {332}] generated by model SPG Carleton
[StationPassengerGenerator_defs::PassengerInitial: {251}] generated by model SPG Greenboro
00:17:00:500
[StationPassengerGenerator_defs::PassengerInitial: {352}] generated by model SPG Greenboro
00:22:00:500
[StationPassengerGenerator_defs::PassengerInitial: {434}] generated by model SPG Carleton
00:26:00:500
[StationPassengerGenerator_defs::PassengerInitial: {534}] generated by model SPG Carleton
[StationPassengerGenerator_defs::PassengerInitial: {454}] generated by model SPG Greenboro
00:27:00:500
[StationPassengerGenerator_defs::PassengerInitial: {553}] generated by model SPG Greenboro
00:29:00:500
[StationPassengerGenerator_defs::PassengerInitial: {651}] generated by model SPG Greenboro
```

Figure 5 - Station passenger generator test outputs for 2 generators.

Looking at Figure 5 we can see the colour-coded outputs are as expected. The Carleton passenger generator (red) output a total of 5 unique passenger IDs while the Greenboro passenger generator (green) output a total of 6 unique passenger IDs within a 30 minute simulation. This agrees with the expected value of approximately 1 passenger per station every 5 minutes. There were no cases of passenger ID's with identical origin and destination stations. For each new passenger ID the generators output, the passenger number represented by digits in the 100's place incremented by 1. We can also note that there is a 500 millisecond initialization delay and passengers are always output at some fraction of a second to avoid simultaneous inputs to other atomic models. The shortest wait time observed was 2 minutes and the longest wait time observed was 9 minutes which are within the minimum and maximum wait times (1 and 10 mins respectively). Simulating this model for extended durations (several hours) produces results that meet all the expected behavior for the model.

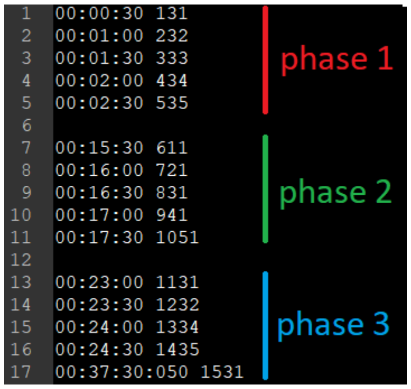
3.4 Station Queue Atomic Test

The station queue atomic model is similar to the in-train queue model. The main difference is that the queue is fixed at a station and receives its passenger ID inputs from a station generator. The other input it receives is from the station arrival generator in order to know when the train has arrived. When this occurs, passengers are transferred from the station queue to the appropriate in-train-queues. Because the station queue receives inputs from the generator, an additional case must be considered and tested. It is possible that when the train arrives at a station and the queue begins emptying, a new passenger is inputted from the passenger generator. In such a case the queue emptying should be paused, the new passenger added to the back of the queue, and queue emptying resumes so all passengers can board the train.

Testing will involve an input file for the station arrival generator that mimics the deterministic output behavior of that model. It outputs the corresponding station number based on the trains schedule in the same manner shown from the in-train queue test. Refer to Figure 2 from that section for the relevant station number inputs and timing.

The testing is once again performed in 3 phases. Phase 1 consists of 5 passengers attempting to enter the Carleton station queue but one of those passengers has an invalid ID due to identical origin and destination stations (they are already at their destination). The 4 valid passengers should be accepted into the queue to wait for the train's arrival and only the invalid passenger should be rejected. Phase 2 consists of 5 passengers attempting to enter the Carleton station queue but only one of them have a valid origin station (passenger ID { 831 }). The remaining passengers have origin stations that do not correspond to Carleton; therefore, the station queue should reject them. In this test, only passenger ID { 831 } should enter the queue to await the arrival of the train.

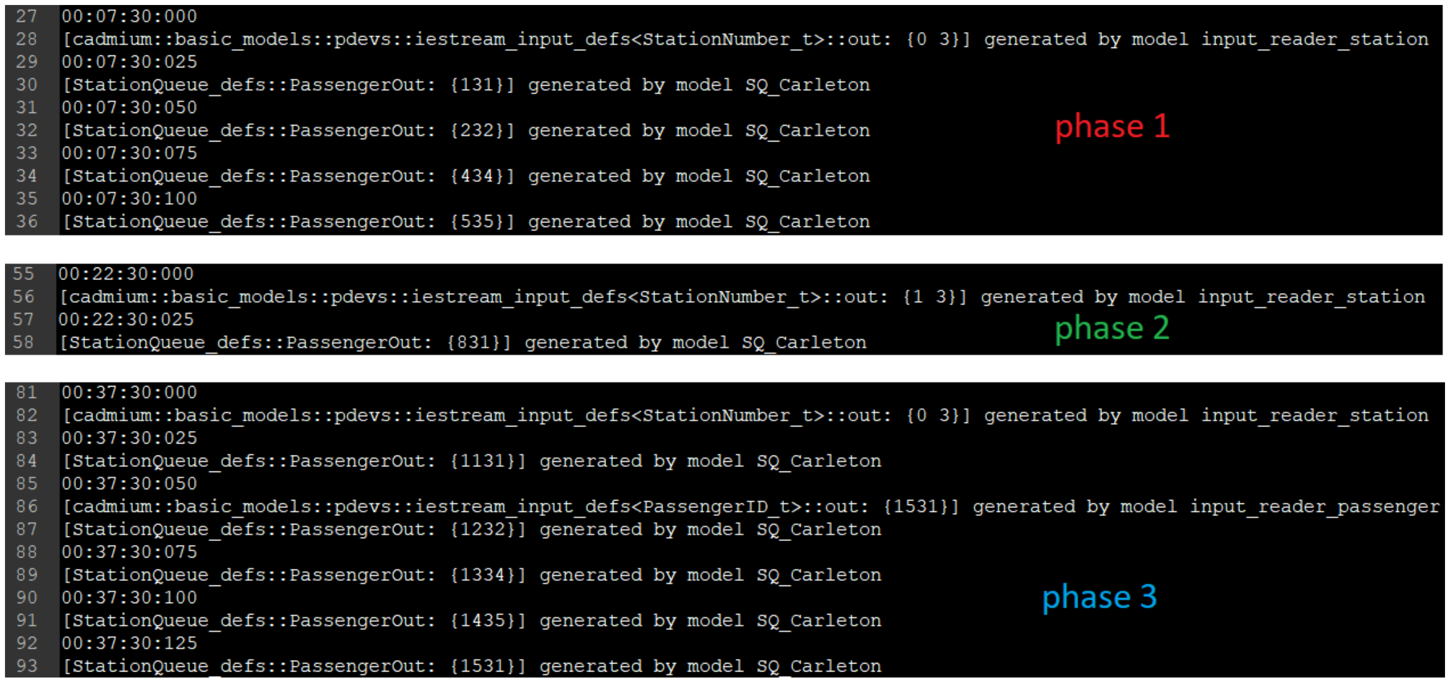
One improvement to the original model included handling the case of a passenger being generated and inputted to the station queue after passenger boarding had already commenced. Phase 3 of the test adds 4 valid passengers to the Carleton station queue well ahead of the train's arrival. After the train arrives and boarding commences, an additional passenger is simulated as being generated. Boarding the train should pause, the new passenger should be added to the back of the queue (if otherwise valid) and train boarding resumes. The passenger ID inputs for this test are shown in Figure 6 below.



```
00:00:30 131
00:01:00 232
00:01:30 333
00:02:00 434
00:02:30 535

00:15:30 611
00:16:00 721
00:16:30 831
00:17:00 941
00:17:30 1051

00:23:00 1131
00:23:30 1232
00:24:00 1334
00:24:30 1435
00:37:30:050 1531
```

Figure 6: Station Queue test – input data for passenger IDs.

The test was run for 40 minutes and the output behavior was as expected. In phase 1 of the test, the only passenger to be rejected to the Carleton in-train queue was passenger ID { 333 }. All remaining passengers from phase 1 entered the queue and boarded the train when it arrived at t = 7 minutes 30 seconds. In phase 2 of the test, only a single passenger possessed the correct origin station ID corresponding to Carleton. Consequently, only passenger ID { 831 } was added to the queue and boarded the train at its next arrival (t = 22 minutes 30 seconds). Phase 3 was also a success: the passenger generated at Carleton was added to the end of the Carleton station queue while it was already emptying (boarding the train). Refer to Figure 7 below for the relevant outputs (note that SQ is short for station queue).



```
00:07:30:000
[cadmium::basic_models::pdevs::iestream_input_defs<StationNumber_t>::out: {0 3}] generated by model input_reader_station
00:07:30:025
[StationQueue_defs::PassengerOut: {131}] generated by model SQ_Carleton
00:07:30:050
[StationQueue_defs::PassengerOut: {232}] generated by model SQ_Carleton
00:07:30:075
[StationQueue_defs::PassengerOut: {434}] generated by model SQ_Carleton
00:07:30:100
[StationQueue_defs::PassengerOut: {535}] generated by model SQ_Carleton
```

```
00:22:30:000
[cadmium::basic_models::pdevs::iestream_input_defs<StationNumber_t>::out: {1 3}] generated by model input_reader_station
00:22:30:025
[StationQueue_defs::PassengerOut: {831}] generated by model SQ_Carleton
```

```
00:37:30:000
[cadmium::basic_models::pdevs::iestream_input_defs<StationNumber_t>::out: {0 3}] generated by model input_reader_station
00:37:30:025
[StationQueue_defs::PassengerOut: {1131}] generated by model SQ_Carleton
00:37:30:050
[cadmium::basic_models::pdevs::iestream_input_defs<PassengerID_t>::out: {1531}] generated by model input_reader_passenger
[StationQueue_defs::PassengerOut: {1232}] generated by model SQ_Carleton
00:37:30:075
[StationQueue_defs::PassengerOut: {1334}] generated by model SQ_Carleton
00:37:30:100
[StationQueue_defs::PassengerOut: {1435}] generated by model SQ_Carleton
00:37:30:125
[StationQueue_defs::PassengerOut: {1531}] generated by model SQ_Carleton
```

Figure 7 – Relevant station queue test outputs.

3.5 O-Train Coupled Test

With atomic models confirmed to be functional, the single coupled model from the original CD++ specification needs to be assembled and tested with passenger inputs. The original CD+ documentation did not perform any coupled model testing, so this test was added to verify the assembled train model operated as expected.

The coupled O-Train model has a single passenger arrival generator and 5 in-train queues corresponding to each station. Passenger inputs are the only input to the train and are copied to all in-train queues, but only passengers who belong to a given queue will be accepted. Outputs of the train are passenger IDs which correspond to passengers deboarding the train to arrive at their destination. Additionally, the deterministic output of the passenger arrival generator is an output of the coupled model. The station arrival outputs from this generator will later be connected to station queues to announce the arrival of the train.

Three phase testing was once again implemented for the O-Train coupled model. The 1st phase of testing features a group of 4 passengers immediately attempting to board from Bayview station. All 4 passengers are valid and have different destination stations. They should each be added to their appropriate in-train queue and only deboard the train when they arrive at their destination one-by-one. Phase 2 of testing is another group of 4 valid passengers all attempting to board at Confederation station. They all have the same destination of Carleton station and should exit the train together when it arrives there next. A successful test should show all 4 passengers being accepted to the train and deboarding together.

Phase 3 is similar to Phase 2: a group of 4 passengers board the train at Carling station heading south to Carleton. All passengers have the same destination station and will deboard one-by-one upon arrival. However, during deboarding, a passenger at Carleton station attempts to board the train with a destination that is also Carleton. For the test to be successful the passenger should board the train as passengers are exiting, get in the back of the in-train queue, and then deboard the train immediately (get on and get back off). The passenger inputs for the coupled O-Train test is shown below in Figure 8.

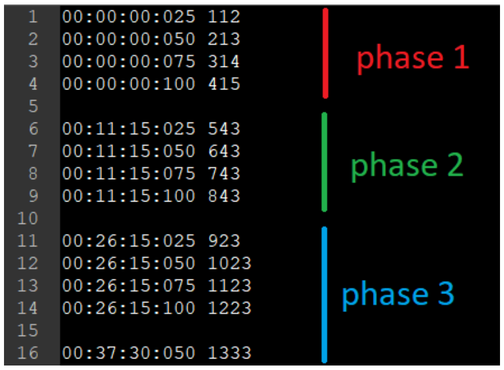

```
00:00:00:025 112
00:00:00:050 213
00:00:00:075 314
00:00:00:100 415

00:11:15:025 543
00:11:15:050 643
00:11:15:075 743
00:11:15:100 843

00:26:15:025 923
00:26:15:050 1023
00:26:15:075 1123
00:26:15:100 1223

00:37:30:050 1333
```

Figure 8: Coupled O-Train test – input data for passenger IDs.

All three phases of testing were passed with the expected outputs. In phase 1, each of the passengers deboarded at the correct times and correct stations. In phase 2, multiple passengers were able to successfully deboard the train one-by-one at Carleton station. In phase 3 of testing, a passenger boarded the train while passengers were exiting. The boarding passenger had the same destination as the deboarding passengers and immediately followed them in exiting the train. The output of this test is too long to show in a single screenshot, therefore, please refer to the “COUPLED_TEST_OTrain_outputs.txt” file in the *simulation results* folder to confirm the outputs.

4.0 Top Model Simulation and Trip Analysis

With the atomic and coupled models fully tested, the top-level model can be simulated. Trip time analysis will then be performed to determine how long the average passenger waits at a station and spends on the train. The simulation was run for 1 hour and the behaviour of the model was as expected upon manual inspection. No cases of incorrect boarding/deboarding were found. All generators behaved as intended.

To analyze the trip times, the following parameters have been recorded for a sample of the total passengers: time of generation, time when boarding the train, and time when arriving at their destination. The average wait time at a station along with the average time spent on the train can then be computed. Data for the first 25 passengers is logged in Table 1 below.

Table 1: Generation time, boarding time, and arrival time for first 25 O-Train Passengers.

Passenger ID	Time Generated	Time Boarded Train	Time Arrived at Destination
132	00:03:00:500	00:07:30:025	00:26:15:025
113	00:04:00:500	00:30:00:025	00:37:30:050
235	00:04:00:500	00:07:30:050	00:15:00:025
151	00:04:00:500	00:15:00:025	00:30:00:025
335	00:05:00:500	00:07:30:075	00:15:00:050
125	00:06:00:500	00:26:15:025	00:45:00:100
251	00:06:00:500	00:15:00:050	00:30:00:050
354	00:08:00:500	00:15:00:075	00:18:45:025
142	00:09:00:500	00:11:15:025	00:26:15:050
435	00:10:00:500	00:22:30:025	00:45:00:050
225	00:11:00:500	00:26:15:050	00:45:00:125
534	00:11:00:500	00:22:30:050	00:41:15:025
214	00:13:00:500	00:30:00:050	00:41:15:050
454	00:13:00:500	00:15:00:100	00:18:45:050
553	00:14:00:500	00:15:00:125	00:22:30:025
245	00:16:00:500	00:18:45:025	00:45:00:025
313	00:17:00:500	00:30:00:075	00:37:30:075
651	00:17:00:500	00:45:00:025	01:00:00:050
635	00:18:00:500	00:22:30:075	00:45:00:075
323	00:19:00:500	00:26:15:075	00:37:30:025
731	00:20:00:500	00:22:30:100	00:30:00:075
751	00:20:00:500	00:45:00:050	01:00:00:075
445	00:22:00:500	00:41:15:050	00:45:00:175
412	00:24:00:500	00:30:00:100	00:33:45:025
421	00:24:00:500	00:26:15:100	00:30:00:100

From Table 1, the time spent waiting at the station and time spent on the train are now computed for each passenger. The wait times are show below in Table 2. Note that these times have been rounded to the nearest whole second.

Table 2: Station wait time and time on train for first 25 O-Train Passengers.

Passenger ID	Station Wait Time	Time on Train
132	00:04:30	00:18:45
113	00:26:00	00:07:30
235	00:03:30	00:07:30
151	00:11:00	00:15:00
335	00:02:30	00:07:30
125	00:20:15	00:18:45
251	00:09:00	00:15:00
354	00:07:00	00:03:45
142	00:02:15	00:15:00
435	00:12:30	00:22:30
225	00:15:15	00:18:45
534	00:11:30	00:18:45
214	00:17:00	00:11:15
454	00:02:00	00:03:45
553	00:01:00	00:07:30
245	00:02:45	00:26:15
313	00:13:00	00:07:30

651	00:28:00	00:15:00
635	00:04:30	00:22:30
323	00:07:15	00:11:15
731	00:02:30	00:07:30
751	00:25:00	00:15:00
445	00:19:15	00:03:45
412	00:06:00	00:03:45
421	00:02:15	00:03:45
Average	**00:10:14**	**00:12:18**

The data from the first 25 passengers in the simulations indicates that the average time a passenger spent waiting at the station was 10 minutes 14 seconds and the average time spent travelling on the O-Train was 12 minutes 18 seconds. These combine for an average total trip time of 22 minutes 32 seconds. One important factor to note is that the top model simulation only has a single train on the tracks. Realistically, the O-Train track has multiple trains at any given time. This severely reduces the wait time of a passenger at a given station but does not necessarily reduce the time spent on the train. For example, if there were two trains running on the track in opposite directions, in theory the wait time at stations should be reduced by roughly half. This would mean a total trip time closer to 17 minutes 25 seconds.

One important observation can be made from Table 2 regarding the total time each passenger spent on the train. Each passenger's time spent on the O-Train were multiples of the 3 minute 45 second interval between each station. This confirms that passengers were only boarding / deboarding during the scheduled station arrivals and that the model was functioning properly. The wait times at stations are more random due to the stochastic nature of when passengers are generated. Additionally, they may be lucky and be generated shortly before an arrival or they may be unlucky and be generated just after the train's departure. This accounts for the high variance in initial station wait times which is made even worse by the fact there is only one train on the track. If a passenger is generated at Bayview or Greenboro immediately after the train leaves, they need to wait almost an entire 30 minutes for it to complete the round trip back to them. This is exactly what happened to passenger 651 in Tables 1 and 2.

5.0 Discussion

The original CD++ O-Train model was successfully implemented using Cadmium. The overall behaviour of the top model is nearly identical to the original with some small improvements to each of the atomic models. The station arrival generator was improved so that when the simulation begins, there is an immediate arrival at the default station before the train moves to the next station. The passenger generators were improved so that they would never produce invalid passengers with identical origin / destination stations. They were also improved by enforcing limits on the maximum and minimum wait times between generated passengers. Both the in-train queues and station queues were improved so that they could handle the cases of an input being provided to the queue during the emptying process. All models were tested successfully and the top-level simulation produced the expected behaviour.

One improvement that could have been implemented would be to make each station its own coupled model containing a passenger generator and station queue. This would simplify the connections and instantiations in the top-level model. Another improvement that could have been implemented would be to use 2 trains instead of one. While this would have made the trip times more accurate but it also would have complicated the model. Specifically, the model would have to account for both trains simultaneously arriving at Carleton station. Special cases would need to be coded to put certain passengers on the northbound train and other passengers on the southbound train.

In the trip time analysis, the original CD++ documentation did not distinguish between the time spent waiting at a station versus the travel time on the train. As we saw in this simulation, there is an important distinction between the two. Time spent waiting at the station was almost equal to the time spent on the train when only one train was running on the tracks. Future simulations with more trains may show reductions for station wait times.