1/11/2016

Ifeoluwa Oyelowo - 100918125

Border Crossing Model (CDBoost)

SYSC 5104 – Assignment 1

Table of Contents

[1.0 Introduction 2](#_Toc465624439)

[1.1 Border Crossing Model 2](#_Toc465624440)

[2.0 Border Crossing Model implemented in CDBoost 4](#_Toc465624441)

[2.1 Formal Specifications: Atomic Models 6](#_Toc465624442)

[2.2 Formal Specifications: Coupled Models 7](#_Toc465624443)

[2.2.1 FrontDesk 7](#_Toc465624444)

[2.2.2 Turn\_away\_N\_ok1 8](#_Toc465624445)

[2.2.3 ExtraC1 8](#_Toc465624446)

[2.2.4 NewP1 8](#_Toc465624447)

[2.2.5 carCheck 9](#_Toc465624448)

[2.2.6 interview 9](#_Toc465624449)

[2.2.7 Turn\_away\_N\_ok2 9](#_Toc465624450)

[2.2.8 ExtraC\_NewP1 10](#_Toc465624451)

[2.2.9 extraChecks 10](#_Toc465624452)

[2.3 Testing Strategy 11](#_Toc465624453)

[3.0 Simulation Tests 11](#_Toc465624454)

[3.1 Atomic Model Tests 11](#_Toc465624455)

[3.1.1 Line Model Test 11](#_Toc465624456)

[3.1.2 decider Model Test 13](#_Toc465624457)

[3.1.3 secondaryDecider Model Test 15](#_Toc465624458)

[3.2 Coupled Model Tests 17](#_Toc465624459)

[3.2.1 frontDesk Model Test 17](#_Toc465624460)

[3.2.2 interview Model Test 18](#_Toc465624461)

[3.2.3 extraChecks Model Test 20](#_Toc465624462)

[3.3 Border Model Test 22](#_Toc465624463)

[References 24](#_Toc465624464)

# 1.0 Introduction

“The DEVS formalism for discrete-event systems can be used to model many different natural or artificial systems.” [1] In this course SYSC 5104, DEVS models can be implemented using two different software namely: CD++ and CDBoost. CD++ is frequently used by students because it works with a Windows operating system (OS) and Windows OSs are more common. On the other hand, CDBoost is a software that works only with a Linux operating system and thus is less commonly used by students. In this assignment, the author translated a DEVS model called “Border Crossing Model” created by a previous student of the course, Scott Stewart to CDBoost. The student created the formal specification for this model, implemented this model in CD++ and thoroughly tested this model. The in-depth details about the model are in the Border Crossing Model Final Report by Scott Stewart in [1] and thus, the author will refer to this report a lot throughout this document. A brief explanation of this model is given in the following section.

## 1.1 Border Crossing Model

The Border Crossing Model will be simply referred to as the border model from this point onwards. The border model was designed to demonstrate the different things that may happen when a person arrives at a border, such as: whether the person is ok to cross the border, if the person is turned away, or if the person will undergo extra checks before they either cross the border or are turned away. Figure 1 shows the conceptual model for the border crossing model. This model is a bit different from the border model showed in Figure 1 in [1], the actual names used in the CD++ implementation of the border model were included in the border model for clarity. As seen in Figure 1, a person is received into the border through the input port (*new\_person*) and the person is either let through (through ok output port) or turned away completely (through *turn\_away* output port).

The border model is made up of two coupled models named frontDesk and extraChecks. The frontDesk receives the people coming in from the input and looks at them in turn, deciding if they are okay to pass through the ‘*ok*’ output port, or if they should be turned away through ‘*turn\_away’* output port, or they have to go for more checks through the ‘*extra\_checks’* output port. The *ok* and *turn\_away* outputs of the frontDesk model are linked to the output ports of the border model which have the same names. The frontDesk model can be subdivided further into two atomic models, the ‘line’ and the ‘decider’ models. “The line model places the people coming into the input in a line, only allowing the person at the front of the line through if the decider is not busy.” [1] The person at the front of the line is passed to the decider through the line’s *line\_out* output port and the decider receives a new person through its *next\_person* input port. Also, the decider tells the line that it is not busy through its *is\_free* output port and the line receives this signal through its *ready\_for\_next* input port. The decider takes in the person from the head of the line and decides what to do with the person. If the person is required to have extra checks performed on them, they will be outputted through the FrontDesk’s *extra\_checks* port into the ExtraChecks’ *extra\_checks* input port.

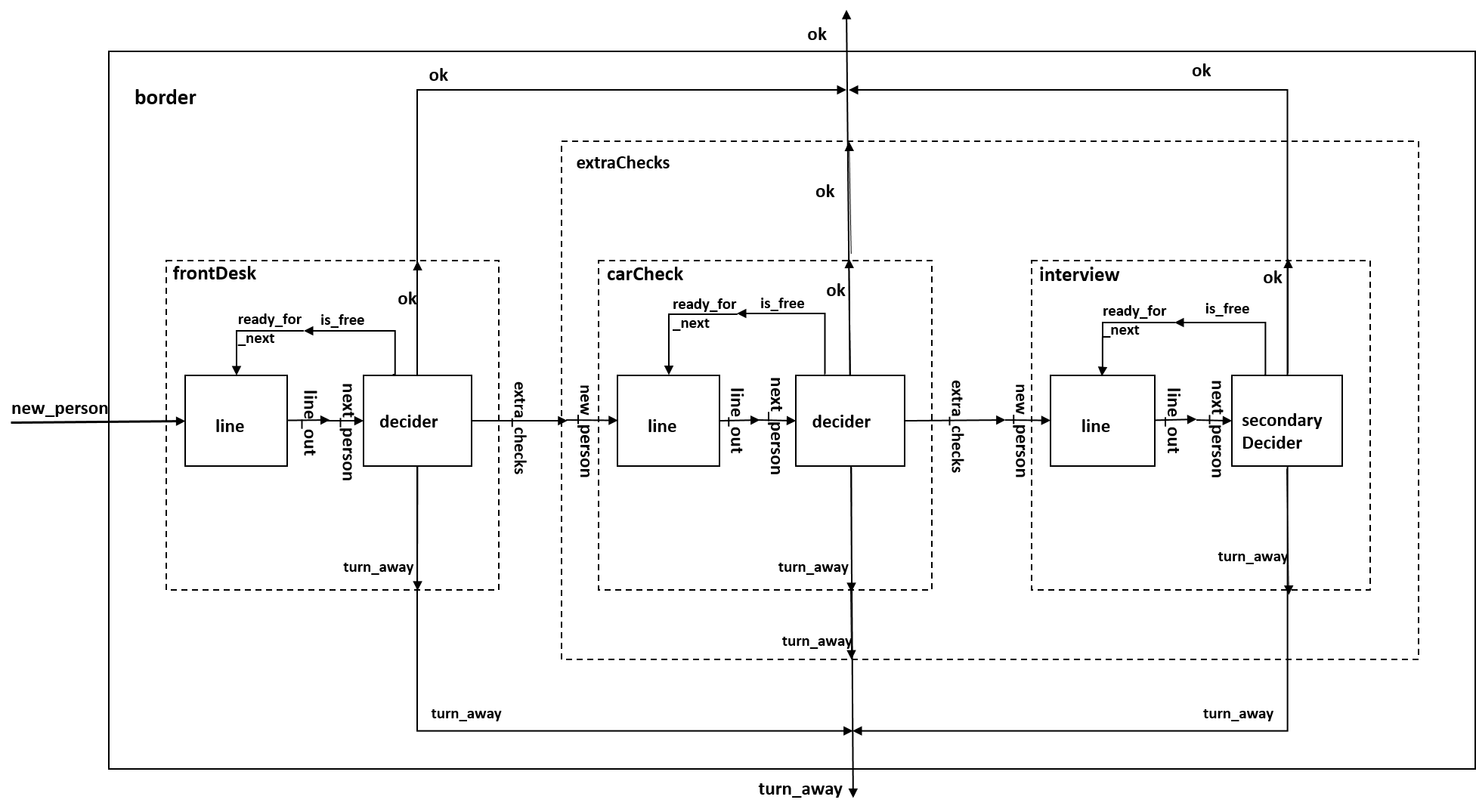


Figure : Conceptual model for one-way border crossing (Figure 1 of Border Crossing Model Final Report by Scott Stewart but modified by Ifeoluwa Oyelowo)

“The extraChecks model, as the name implies, places further checks on the people passing through it, ultimately deciding if they will be allowed to cross the border or are denied. It is composed of two different submodels: the carCheck and the interview which are both coupled models. The carCheck is composed the same way as the frontDesk: a line atomic model and a decider atomic model. The behavior is the exact same as the frontDesk, putting people in a line if the decider is busy. If the person at the decider is outputted through the carCheck’s *extra\_checks* port, it will then go into the interview’s *new\_person* input port. The interview coupled model is composed of a line atomic model and a modified decider atomic model, labeled the secondaryDecider. This atomic model has only two output ports: *ok* and *turn\_away*, removing the *extra\_checks* port because this coupled model is the last model the person will go through for extra checks. The interview and carCheck output ports *ok* and *turn\_away* are both routed to the ports with the same name on ExtraChecks, which then links those ports to the output ports of the border model.” [1]

# 2.0 Border Crossing Model implemented in CDBoost

The border crossing model explained in section 1.1 was implemented in CD++. However, to implement the same model in CDBoost, a modified version of the model shown in Figure 1 is used. This new model, shown in Figure 2 supports the limitation of CDBoost which is that all models in CDBoost must have just one output port and one input port. To support models which have more than one input or output ports, a message structure which has a port and value element is used. The message structure is defined in the message.cpp and message.hpp files which are in the data\_structure folder. The message structure came with the CDBoost template and is not the author’s work. If for instance, the decider model in Figure 1 which has 4 outputs wants to send an output through its *ok* port, the decider will send a message through its output port and specify the port name to be *ok* and also specify the value which the *ok* port is sending.

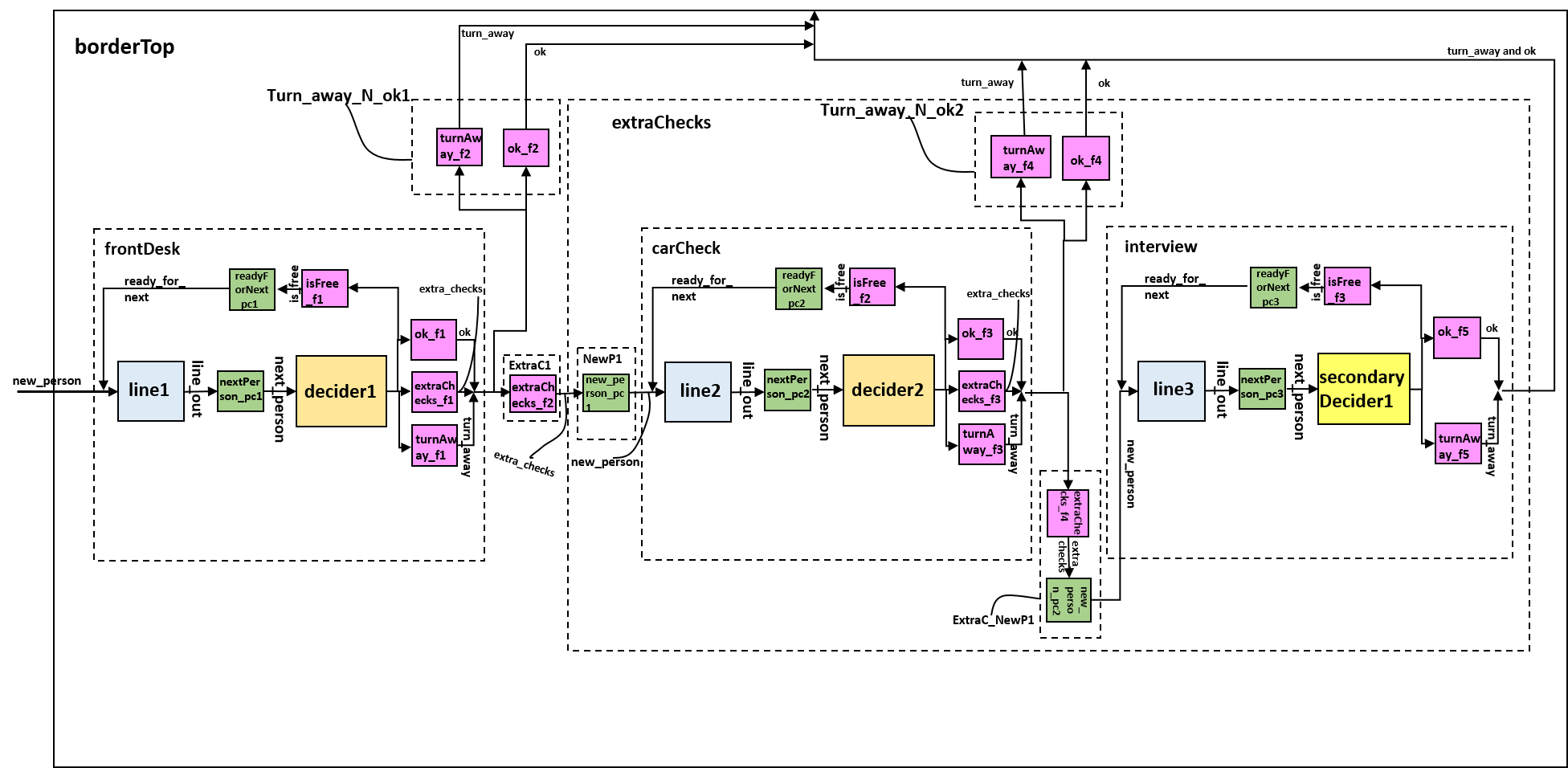


Figure : Border crossing model implemented in CDBoost



Figure : Legend of Figure 2

Since each model only has one output port in CDBoost, if a model has more than one output to be sent through its output port, filters have to be used to ensure that the outputs of each model go to the right places. To send an output, a model will send a message specifying the port’s name and the value being sent through the port. The filter model is an atomic model which came with the CDBoost template and is not the author’s work. The filter model can be found in the vendors folder. Consider the decider model which has four outputs. The *is\_free* output goes to the input of the line model. A filter model called isfree\_f1 must be placed between the decider1 and the line1 models to ensure that when the output of the decider1 model is from the *is\_free* port (and this would be specified in the message sent out through the decider1’s output port), the value would go to the line1 model.

Another thing considered in developing the model in Figure 2 is that CDBoost supports different port names using a portConversor atomic model. Again this model came with the CDBoost template and is not the author’s work. This model can be found in the vendors folder. Consider the line atomic model in the frontDesk coupled model in Figure 1 which has an output port called *line\_out* which should be an input to the decider model. However, the decider’s input port is called *next\_person*. In this case, the portConversor will be placed between the line and the decider, to ensure that when the line sends its output through the *line\_out* port, the decider receives that value through its *next\_person* port. In Figure 2 above, this portConversor is called nextPerson\_pc1.

Note that the models in Figure 2 are color coded and the legend is shown in Figure 3. For instance, all models coloured pink are instances of the filter atomic model and though they have different names, they have the same behaviour. The same applies for the other atomic models.

Note also, that in CDBoost, atomic models cannot be directly coupled to coupled models. This is why in Figure 2, the coupled models ExtraC1, Turn\_away\_N\_ok1 and so on exists. For instance, the coupled model ExtraC1 has just one atomic model called extraChecks\_f2. This atomic model must be in a coupled model in order to be coupled to the extraChecks coupled model. All coupled models are bounded by dashed lines.

All the models (both atomic and coupled models) in Figure 1 are in Figure 2. However, due to some limitations of CDBoost, some additional models explained above were added to the model in Figure 2. The functionality of extraChecks and frontDesk coupled models remain the same.

## 2.1 Formal Specifications: Atomic Models

The formal specification for atomic models is: <S, X, Y, int, ext, , ta>. There are five different atomic models used in the Border model: line, decider, secondaryDecider, filter, portConversor. The line, decider and secondaryDecider models are exactly the same models as the ones already defined in the Border Crossing Model Final report by Scott Stewart in [1] and will not be redefined in this report. Also, the portConversor and filter models are not the author’s work and will not be defined in this report.

## 2.2 Formal Specifications: Coupled Models

The formal specifications for the coupled models is: <X, Y, D, EIC, EOC, IC, SELECT>. There are nine different coupled models: frontDesk, extraChecks, carCheck, interview, ExtraC1, Turn\_away\_N\_ok1, Turn\_away\_N\_ok2, NewP1, ExtraC\_NewP1. frontDesk, extraChecks, carCheck and interview are different from the coupled models defined in the Border Crossing Model Final report by Scott Stewart in [1] and would be redefined in this report.

### 2.2.1 FrontDesk

In CDBoost, the coupled models are completely defined by the atomic models which form the coupled models. Therefore instead of the external input coupling (EIC) of frontDesk to be seen as (self.new\_person, line.new\_person), line1 is the EIC of frontDesk as that model has the input *new\_person* which is the same input for frontDesk. In the same way the EOC of frontDesk will be ok\_f1, extraChecks\_f1 and turnAway\_f1 as these atomic models have the output ports *ok, extra\_checks* and *turn\_away* respectively which are the same for frontDesk. Finally, models in CDBoost have only one input and one output port and therefore, the internal coupling of frontDesk are defined in terms of the models. For instance, the internal coupling between line1 and nextPerson\_pc1 will be {line1, nextPerson\_pc1} and not {line1.line\_out, nextPerson\_pc1.line\_out}. This is the way all the coupled models will be defined. The formal specification for the frontDesk model is shown below:

X = {new\_person}

Y = {ok, extra\_checks, turn\_away}

D = {line1, decider1, nextPerson\_pc1, readyForNext\_pc1, isFree\_f1, ok\_f1, extraChecks\_f1, turnAway\_f1}

EIC = {line1}

EOC = {ok\_f1, extraChecks\_f1, turnAway\_f1}

IC = {{line1, nextPerson\_pc1}, {nextPerson\_pc1, decider1}, {decider1, isFree\_f1},

{isFree\_f1, readyForNext\_pc1}, {readyForNext\_pc1, line1}, {decider1, ok\_f1},

 {decider1, extraChecks\_f1}, {decider1, turnAway\_f1}}

SELECT: ({line1, decider1, nextPerson\_pc1, readyForNext\_pc1, isFree\_f1, ok\_f1, extraChecks\_f1, turnAway\_f1}) = decider1;

### 2.2.2 Turn\_away\_N\_ok1

X = {extra\_checks, ok, turn\_away}

Y = {ok, turn\_away}

D = {turnAway\_f2, ok\_f2}

EIC = {turnAway\_f2, ok\_f2}

EOC = {turnAway\_f2, ok\_f2}

IC = {}

SELECT: ({turnAway\_f2, ok\_f2}) = ok\_f2;

### 2.2.3 ExtraC1

X = {ok, extra\_checks, turn\_away}

Y = {extra\_checks}

D = {extraChecks\_f2}

EIC = {extraChecks\_f2}

EOC = {extraChecks\_f2}

IC = {}

SELECT: ({extraChecks\_f2}) = extraChecks\_f2;

### 2.2.4 NewP1

X = {extra\_checks}

Y = {new\_person}

D = {new\_person\_pc1}

EIC = {new\_person\_pc1}

EOC = {new\_person\_pc1}

IC = {}

SELECT: ({new\_person\_pc1}) = new\_person\_pc1;

### 2.2.5 carCheck

X = {new\_person}

Y = {ok, extra\_checks, turn\_away}

D = {line2, decider2, nextPerson\_pc2, readyForNext\_pc2, isFree\_f2, ok\_f3, extraChecks\_f3, turnAway\_f3}

EIC = {line2}

EOC = {ok\_f3, extraChecks\_f3, turnAway\_f3}

IC = {{line2, nextPerson\_pc2}, {nextPerson\_pc2, decider2}, {decider2, isFree\_f2},

{isFree\_f2, readyForNext\_pc2}, {readyForNext\_pc2, line2}, {decider2, ok\_f3},

 {decider2, extraChecks\_f3}, {decider2, turnAway\_f3}}

SELECT: ({line2, decider2, nextPerson\_pc2, readyForNext\_pc2, isFree\_f2, ok\_f3, extraChecks\_f3, turnAway\_f3}) = decider2;

### 2.2.6 interview

X = {new\_person}

Y = {ok, turn\_away}

D = {line3, secondaryDecider1, nextPerson\_pc3, readyForNext\_pc3, isFree\_f3, ok\_f5, extraChecks\_f5, turnAway\_f5}

EIC = {line3}

EOC = {ok\_f5, turnAway\_f5}

IC = {{line3, nextPerson\_pc3}, {nextPerson\_pc3, secondaryDecider1}, {secondaryDecider1, isFree\_f3}, {isFree\_f3, readyForNext\_pc3}, {readyForNext\_pc3, line3}, {secondaryDecider1, ok\_f5}, {secondaryDecider1, extraChecks\_f5}, {secondaryDecider1, turnAway\_f5}}

SELECT: ({line3, secondaryDecider1, nextPerson\_pc3, readyForNext\_pc3, isFree\_f3, ok\_f5, extraChecks\_f5, turnAway\_f5}) = secondaryDecider1;

### 2.2.7 Turn\_away\_N\_ok2

X = { extra\_checks, ok, turn\_away}

Y = {ok, turn\_away}

D = {turnAway\_f4, ok\_f4}

EIC = {turnAway\_f4, ok\_f4}

EOC = {turnAway\_f4, ok\_f4}

IC = {}

SELECT: ({turnAway\_f4, ok\_f4}) = ok\_f4;

### 2.2.8 ExtraC\_NewP1

X = {ok, extra\_checks, turn\_away}

Y = {new\_person}

D = {new\_person\_pc2, extraChecks\_f4}

EIC = {extraChecks\_f4}

EOC = {new\_person\_pc2}

IC = {extraChecks\_f4, new\_person\_pc2}

SELECT: ({extraChecks\_f4, new\_person\_pc2}) = new\_person\_pc2;

### 2.2.9 extraChecks

X = {extra\_checks}

Y = {ok, turn\_away}

D = {NewP1, carCheck, interview, Turn\_away\_N\_ok2, ExtraC\_NewP1}

EIC = {NewP1}

EOC = {interview, Turn\_away\_N\_ok2}

IC = {NewP1, carCheck}, {carCheck, ExtraC\_NewP1}, {carCheck, Turn\_away\_N\_ok2} , {ExtraC\_NewP1, interview}}

SELECT: ({NewP1, carCheck, interview, Turn\_away\_N\_ok2, ExtraC\_NewP1}) = carCheck;

## 2.3 Testing Strategy

As previously stated, the border model implemented in CDBoost in this report has been previously implemented in CD++. The border model had thoroughly been tested by the previous student. As a result, the testing strategy used to test the CDBoost border model was to test the model implemented in CDBoost using the same test files which the previous student used and compare the output results. The model was tested as follows:

• Each of the atomic models were tested using a variety of inputs through a text file (.txt). These inputs were obtained from the event file used in the CD++ implementation of the model. The resultant CDBoost output file (.txt) was compared with the output file (.out) from the CD++ project.

• The coupled models were also constructed and then tested in the same manner as the atomic models. If the behavior of the coupled model matched the CD++ implementation of the model, it could be used in further coupled model tests.

• Once all of the individual components were tested, the final in Figure 2 border crossing model was tested using the same tests which the original author used and the outputs were compared.

# 3.0 Simulation Tests

## 3.1 Atomic Model Tests

### 3.1.1 Line Model Test

The behaviour of the line model was already explained in [1] and will not be explained again here, the same applies to all the other atomic and coupled models tested. The inputs to the line model from the CD++ implementation of the model (from the event file (.ev)) and the corresponding inputs to the line model from the CDBoost implementation of the model are shown in Table 1 below. In the Assignment1\_Border\_Crossing package which contains all the project files for the CDBoost implementation of the border project. The following path leads to the line input test file: *Assignment1\_Border\_Crossing>>* *test >> line >> line\_input\_test.txt.*

Table : Inputs to the line model

|  |  |
| --- | --- |
| Inputs (.ev file) – CD++  (*line.ev)* | Inputs (.txt file) – CDBoost (*line\_input\_test.txt.)* |
| 00:00:10:00 new\_person 1  00:00:20:00 new\_person 2  00:00:30:00 new\_person 3  00:00:40:00 new\_person 4  00:00:45:00 ready\_for\_next -1  00:00:50:00 new\_person 5  00:00:55:00 ready\_for\_next -1  00:01:00:00 ready\_for\_next -1  00:01:10:00 ready\_for\_next -1  00:01:20:00 ready\_for\_next -1  **00:01:30:00 ready\_for\_next -1**  00:01:50:00 new\_person 6  00:02:00:00 new\_person 7 | 10 1 new\_person 1  20 1 new\_person 2  30 1 new\_person 3  40 1 new\_person 4  45 1 ready\_for\_next -1  50 1 new\_person 5  55 1 ready\_for\_next -1  60 1 ready\_for\_next -1  70 1 ready\_for\_next -1  80 1 ready\_for\_next -1  **90 1 ready\_for\_next -1**  110 1 new\_person 6  120 1 new\_person 7 |

Table 1 shows that the inputs to the CDBoost model and the CD++ model are the same. The only slight difference between the two inputs is the way the time is specified in CDBoost. The time class in CDBoost accepts time as a numerator in seconds and a denominator. As a result the time for each of the inputs in the CD++ .ev file was converted to seconds. The time in seconds was used as the numerator and 1 was used as the denominator for all the inputs in the .txt file for the CDBoost model.

“Because the model is assumed to be completely empty at the beginning of the simulation, the line should pass the first person through immediately. The event in bold is an extra *ready\_for\_next* input that should not cause any extra outputs, since at 70/1 the line should already be empty. At 80/1, the *ready\_for\_next* signal should allow the next person in the line through the output, but since the line is empty the next person should go through immediately. Finally, the person (person 7) received at 120/1 in the input port should not be seen through the output port because it did not receive any additional *ready\_for\_next* signals.” [1]

The output of the line model can be found in the line\_test\_output.txt file and this file can be obtained using following the path: *Assignment1\_Border\_Crossing>> test >> line >> line\_test\_output.txt.* The resulting outputs from the CDBoost line model (in a .txt file) and the outputs from the CD++ model (in a (.out) file) are shown in Table 2 below.

Table : Outputs from the line model

|  |  |
| --- | --- |
| Outputs (.ev file) – CD++  (*lineMAOUT.out*) | Outputs (.txt file) – CDBoost (*line\_test\_output.txt)* |
| 00:00:10:000 line\_out 1  00:00:45:000 line\_out 2  00:00:55:000 line\_out 3  00:01:00:000 line\_out 4  00:01:10:000 line\_out 5  00:01:50:000 line\_out 6 | 10/1 line\_out 1  45/1 line\_out 2  55/1 line\_out 3  60/1 line\_out 4  70/1 line\_out 5  110/1 line\_out 6 |

The outputs from both the CDBoost implementation and the CD++ implementation of the model are similar and thus, the line model behaved as expected.

### 3.1.2 decider Model Test

The inputs to the decider model from the CD++ implementation of the model (from the event file (.ev)) and the corresponding inputs to the decider model from the CDBoost implementation of the model are shown in Table 3 below. The following path leads to the decider input test file: *Assignment1\_Border\_Crossing>> test >> decider >> decider\_input\_test.txt.*

Table : Inputs to the decider model

|  |  |
| --- | --- |
| Inputs (.ev file) – CD++  (*decider.ev)* | Inputs (.txt file) – CDBoost (*decider\_input\_test.txt)* |
| 00:00:10:00 next\_person 1  00:00:20:00 next\_person 2  00:00:30:00 next\_person 3  00:00:40:00 next\_person 4  00:00:50:00 next\_person 5  00:01:00:00 next\_person 6  00:01:10:00 next\_person 7  00:01:20:00 next\_person 8  00:01:30:00 next\_person 9  00:01:40:00 next\_person 10  00:01:50:00 next\_person 11  00:02:00:00 next\_person 12  00:02:10:00 next\_person 13  00:02:20:00 next\_person 14  00:02:30:00 next\_person 15  **00:02:40:00 next\_person 16**  00:02:50:00 next\_person 17  00:03:00:00 next\_person 18  00:03:10:00 next\_person 19  00:03:20:00 next\_person 20 | 10 1 next\_person 1  20 1 next\_person 2  30 1 next\_person 3  40 1 next\_person 4  **50 1 next\_person 5**  60 1 next\_person 6  **70 1 next\_person 7**  80 1 next\_person 8  90 1 next\_person 9  100 1 next\_person 10  110 1 next\_person 11  120 1 next\_person 12  130 1 next\_person 13  140 1 next\_person 14  **150 1 next\_person 15**  160 1 next\_person 16  **170 1 next\_person 17**  180 1 next\_person 18  190 1 next\_person 19  200 1 next\_person 20 |

Again, the two input files are the same with the syntax for the time being the only difference. As in the CD++ decider model, the time advance function for the CDBoost decider model was made to have a normal distribution: its mean value was 5 seconds and its standard deviation was 4 seconds. The input file put twenty people through the input of the decider at an interval of 10 seconds between people. Table 3 shows the event file and the text file that were used to test the decider model.

The outputs of the decider model can be found in the *decider\_test\_output.txt* file and this file can be obtained using following the path: *Assignment1\_Border\_Crossing>> test >> decider >> decider\_test\_output.txt.* The resulting outputs from the CDBoost decider model (in a .txt file) and the outputs from the CD++ model (in a (.out) file) are shown in Table 4 below.

Table : Outputs from the decider model

|  |  |
| --- | --- |
| Outputs (.ev file) – CD++  (*deciderMAOUT.out*) | Outputs (.txt file) – CDBoost (*decider\_test\_output.txt)* |
| 00:00:14:951 ok 1  00:00:14:951 is\_free 1  00:00:20:184 ok 2  00:00:20:184 is\_free 2  00:00:30:831 ok 3  00:00:30:831 is\_free 3  00:00:45:141 ok 4  00:00:45:141 is\_free 4  00:00:51:730 ok 5  00:00:51:730 is\_free 5  00:01:05:642 ok 6  00:01:05:642 is\_free 6  00:01:18:399 extra\_checks 7  00:01:18:399 is\_free 7  00:01:27:620 ok 8  00:01:27:620 is\_free 8  00:01:36:784 ok 9  00:01:36:784 is\_free 9  00:01:44:322 ok 10  00:01:44:322 is\_free 10  00:01:52:603 turn\_away 11  00:01:52:603 is\_free 11  00:02:07:748 ok 12  00:02:07:748 is\_free 12  00:02:12:778 turn\_away 13  00:02:12:778 is\_free 13  00:02:27:712 ok 14  00:02:27:712 is\_free 14  00:02:37:719 ok 15  00:02:37:719 is\_free 15  00:02:55:337 turn\_away 17  00:02:55:337 is\_free 17  00:03:03:133 ok 18  00:03:03:133 is\_free 18  00:03:10:183 ok 19  00:03:10:183 is\_free 19  00:03:22:676 extra\_checks 20  00:03:22:676 is\_free 20 | 11/1 ok 1  11/1 is\_free 1  20/1 ok 2  20/1 is\_free 2  33/1 extra\_checks 3  33/1 is\_free 3  46/1 ok 4  46/1 is\_free 4  64/1 extra\_checks 6  64/1 is\_free 6  86/1 extra\_checks 8  86/1 is\_free 8  92/1 ok 9  92/1 is\_free 9  104/1 ok 10  104/1 is\_free 10  120/1 ok 11  120/1 is\_free 11  122/1 extra\_checks 12  122/1 is\_free 12  140/1 ok 13  140/1 is\_free 13  150/1 ok 14  150/1 is\_free 14  165/1 ok 16  165/1 is\_free 16  183/1 turn\_away 18  183/1 is\_free 18  193/1 ok 19  193/1 is\_free 19  205/1 extra\_checks 20  205/1 is\_free 20 |

“As it can be seen from the output file (from the CD++ implementation of the decider model) *deciderMAOUT.out*, input 16 in bold did not get passed through the decider model. This is because due to the normal distribution of the time advance function, person 17 was inputted to the decider before the decider could output person 16, replacing that person. This shows that the decider needs another model to regulate its inputs.” [1]

A similar behavior can be seen in the output file of the CDBoost decider model. The inputs in bold (inputs 5,7,15 and 17) did not get passed through the decider model. The reason is the same as in the paragraph above. It is fine that different inputs in the CDBoost and CD++ implementation of the model did not get passed through the decider because the time used to generate the outputs is generated by a random function and thus, the fact that the two models behave similarly (they both did not pass 1 or more of the inputs) is enough to conclude that the CDBoost decider model is correct.

### 3.1.3 secondaryDecider Model Test

The test of the secondaryDecider is similar to that of the decider. The inputs to the secondaryDecider model from the CD++ implementation of the model (from the event file (.ev)) and the corresponding inputs to the secondaryDecider model from the CDBoost implementation of the model are shown in Table 5 below. Again these two files are similar. The path to access the *secondaryDecider\_input\_test.txt* fileis similar to the path used to access the *decider\_input\_test.txt.*

Table : Inputs to the secondaryDecider model

|  |  |
| --- | --- |
| Inputs (.ev file) – CD++  (*secondaryDecider.ev)* | Inputs (.txt file) – CDBoost (*secondaryDecider\_input\_test.txt)* |
| 00:00:10:00 next\_person 1  00:00:20:00 next\_person 2  00:00:30:00 next\_person 3  00:00:40:00 next\_person 4  00:00:50:00 next\_person 5  00:01:00:00 next\_person 6  00:01:10:00 next\_person 7  00:01:20:00 next\_person 8  00:01:30:00 next\_person 9  00:01:40:00 next\_person 10 | 10 1 next\_person 1  20 1 next\_person 2  30 1 next\_person 3  40 1 next\_person 4  50 1 next\_person 5  60 1 next\_person 6  70 1 next\_person 7  80 1 next\_person 8  90 1 next\_person 9  100 1 next\_person 10 |

The test of the secondaryDecider is similar to that of the decider. However, the secondaryDecider should only have outputs ‘*ok*’ and ‘*turn\_away’*. The resulting outputs from the CDBoost secondaryDecider model (in a .txt file) and the outputs from the CD++ model (in a (.out) file) are shown in Table 6 below.

Table : Outputs from the secondaryDecider model

|  |  |
| --- | --- |
| Outputs (.ev file) – CD++  (*secondaryDeciderMAOUT.out*) | Outputs (.txt file) – CDBoost (*secondaryDecider\_test\_output.txt)* |
| 00:00:14:951 turn\_away 1  00:00:14:951 is\_free 1  00:00:20:184 ok 2  00:00:20:184 is\_free 2  00:00:30:831 ok 3  00:00:30:831 is\_free 3  00:00:45:141 ok 4  00:00:45:141 is\_free 4  00:00:51:730 turn\_away 5  00:00:51:730 is\_free 5  00:01:05:642 ok 6  00:01:05:642 is\_free 6  00:01:18:399 turn\_away 7  00:01:18:399 is\_free 7  00:01:27:620 ok 8  00:01:27:620 is\_free 8  00:01:36:784 turn\_away 9  00:01:36:784 is\_free 9  00:01:44:322 ok 10  00:01:44:322 is\_free 10 | 18/1 ok 1  18/1 is\_free 1  28/1 ok 2  28/1 is\_free 2  39/1 ok 3  39/1 is\_free 3  46/1 ok 4  46/1 is\_free 4  53/1 turn\_away 5  53/1 is\_free 5  63/1 ok 6  63/1 is\_free 6  77/1 turn\_away 7  77/1 is\_free 7  84/1 ok 8  84/1 is\_free 8  98/1 ok 9  98/1 is\_free 9  107/1 ok 10  107/1 is\_free 10 |

From the above table in both output files, “an output is fired through the port *is\_free* every time the person is sent through either *ok* or *turn\_away*. Also, only those two ports are outputted from – we do not have an additional “decision” port like decider since this atomic model is used to give the final verdict of whether the person is allowed past the border.” [1]

The outputs from both the CDBoost implementation and the CD++ implementation of the secondaryDecider model are similar and thus, the secondaryDecider model behaved as expected.

## 3.2 Coupled Model Tests

After testing the atomic models, the coupled models were put together and tested. The coupled models were put together in main.cpp files. To access the file where the frontDesk coupled model was defined, go to the following path: *Assignment1\_Border\_Crossing>>* *test >> frontDesk >> main.cpp.* The other coupled models can be accessed in a similar way. After creating the coupled models, these models were tested in a similar way as the atomic models.

### 3.2.1 frontDesk Model Test

Note that the frontDesk and carCheck coupled models have the same behaviour and thus only the frontDesk model was tested. The frontDesk coupled model is made up of the following models: line1, decider1, nextPerson\_pc1, readyForNext\_pc1, isFree\_f1, ok\_f1, extraChecks\_f1 and turnAway\_f1. Refer to Figure 2 to see how the atomic models are connected in the frontDesk model. The behaviour of the frontDesk model was already explained in [1] and will not be explained again here, the same applies to all the other coupled models tested.

The inputs to the frontDesk model are stored in a text file called *frontDesk\_input\_test.txt.* This file can be accessed in a similar way as the atomic models test input files were accessed. The inputs to the frontDesk model were based on the inputs used to test the CD++ implementation of the model. The inputs to the CD++ frontDesk model are stored in an event file *frontDesk.ev.* The two input files are shown in Table 7 below.

Table : Inputs to the frontDesk model

|  |  |
| --- | --- |
| Inputs (.ev file) – CD++  (*frontDesk.ev)* | Inputs (.txt file) – CDBoost (*frontDesk\_input\_test.txt)* |
| 00:00:10:00 new\_person 1  00:00:20:00 new\_person 2  00:00:30:00 new\_person 3  00:00:40:00 new\_person 4  00:00:50:00 new\_person 5  00:00:50:00 new\_person 6  00:00:50:00 new\_person 7  00:00:50:00 new\_person 8  00:00:50:00 new\_person 9  00:00:50:00 new\_person 10  00:00:50:00 new\_person 11  00:03:00:00 new\_person 12 | 10 1 new\_person 1  20 1 new\_person 2  30 1 new\_person 3  40 1 new\_person 4  50 1 new\_person 5  50 1 new\_person 6  50 1 new\_person 7  50 1 new\_person 8  50 1 new\_person 9  50 1 new\_person 10  50 1 new\_person 11  180 1 new\_person 12 |

Like in previous tests for the atomic models, if the outputs of the CDBoost frontDesk (this is stored in a text file *frontDesk\_test\_output.txt)* match the outputs of the CD++ frontDesk (this is stored in an out file called *frontDeskMAOUT.out*), this means that the frontDesk model behaves correctly. The two output files are shown below in Table 8.

Table : Outputs from the frontDesk model

|  |  |
| --- | --- |
| Outputs (.ev file) – CD++  (*frontDeskMAOUT.out*) | Outputs (.txt file) – CDBoost (*frontDesk\_test\_output.txt)* |
| 00:00:16:951 ok 1  00:00:22:184 extra\_checks 2  00:00:32:831 ok 3  00:00:47:141 ok 4  00:00:53:730 ok 5  00:01:01:372 turn\_away 6  00:01:11:771 ok 7  00:01:21:391 ok 8  00:01:30:175 turn\_away 9  00:01:36:497 ok 10  00:01:41:100 ok 11  00:03:09:748 ok 12 | 23/1 ok 1  32/1 turn\_away 2  35/1 extra\_checks 3  50/1 ok 4  58/1 ok 5  70/1 ok 6  77/1 ok 7  90/1 extra\_checks 8  100/1 ok 9  110/1 ok 10  122/1 ok 11  188/1 ok 12 |

“From the output file (*frontDeskMAOUT.out)*, we can see that all of the people are seen in order. After the fifth person, it can also be seen that although seven people arrive at the same time they are handled at different times. Additionally, all three output ports do experience traffic at roughly the correct rate. Therefore, we can see that the coupled model is performing correctly.” [1]

The two output files shown in Table 8 do not generate the exact same outputs. But one can see from the *frontDesk\_test\_output.txt* file that the frontDesk model behaves in a similar manner as described in the above paragraph. Therefore, we can conclude that the CDBoost frontDesk model is correct.

### 3.2.2 interview Model Test

The interview coupled model behaves very similar to the frontDesk coupled model and it consists of the following atomic models: line3, secondaryDecider1, nextPerson\_pc3, readyForNext\_pc3, isFree\_f3, ok\_f5, extraChecks\_f5, turnAway\_f5. Refer to Figure 2 to see how the atomic models are connected in the interview model. The interview coupled model has “two “decision” ports which are *ok* and *turn\_away*, which occur with a 50/50 split.” [1]

The inputs to the CDBoost interview model are stored in a text file called *interview\_input\_test.txt.* This file can be accessed in a similar way as the frontDesk’s input test file was accessed. The inputs to the interview model were based on the inputs used to test the CD++ implementation of the model. The inputs to the CD++ interview model are stored in an event file *interview.ev.* The two input files are shown in Table 9 below.

Table : Inputs to the interview model

|  |  |
| --- | --- |
| Inputs (.ev file) – CD++  (*interview.ev)* | Inputs (.txt file) – CDBoost  (*interview \_input\_test.txt)* |
| 00:00:10:00 new\_person 1  00:00:20:00 new\_person 2  00:00:30:00 new\_person 3  00:00:40:00 new\_person 4  00:00:50:00 new\_person 5  00:00:50:00 new\_person 6  00:00:50:00 new\_person 7  00:00:50:00 new\_person 8  00:00:50:00 new\_person 9  00:00:50:00 new\_person 10  00:00:50:00 new\_person 11  00:03:00:00 new\_person 12 | 10 1 new\_person 1  20 1 new\_person 2  30 1 new\_person 3  40 1 new\_person 4  50 1 new\_person 5  50 1 new\_person 6  50 1 new\_person 7  50 1 new\_person 8  50 1 new\_person 9  50 1 new\_person 10  50 1 new\_person 11  180 1 new\_person 12 |

Like in the frontDesk test, if the outputs of the CDBoost interview model (this is stored in a text file *interview\_test\_output.txt)* match the outputs of the CD++ interview model (this is stored in an out file called *interviewMAOUT.out*), this means that the interview model behaves correctly. The two output files are shown below in Table 8.

Table : Outputs from the interview model

|  |  |
| --- | --- |
| Outputs (.ev file) – CD++  (*interviewMAOUT.out*) | Outputs (.txt file) – CDBoost (*interview\_test\_output.txt)* |
| 00:00:16:951 ok 1  00:00:22:184 ok 2  00:00:32:831 ok 3  00:00:47:141 ok 4  00:00:53:730 ok 5  00:01:01:372 turn\_away 6  00:01:11:771 turn\_away 7  00:01:21:391 turn\_away 8  00:01:30:175 turn\_away 9  00:01:36:497 turn\_away 10  00:01:41:100 ok 11  00:03:09:748 ok 12 | 17/1 turn\_away 1  32/1 ok 2  43/1 turn\_away 3  53/1 turn\_away 4  57/1 ok 5  70/1 turn\_away 6  79/1 turn\_away 7  90/1 ok 8  96/1 ok 9  105/1 ok 10  108/1 ok 11  183/1 ok 12 |

“The output (*interviewMAOUT.out*) of the model shows us that the coupled model is performing as expected, sending the people inputted through the input port *new\_person* to either the output port *ok* or *turn\_away* with roughly the same chance. The people also seem to stay in the line in the correct order, and will only be outputted after the person in front of them leaves and when the random amount of time occurs.” [1]

Again the outputs from the CDBoost interview model are not exactly the same as that of the CD++ interview model but the outputs are similar. The outputs in *interview\_test\_output.txt* show that people leave the interview model in roughly the same order and that a person leaves through the *ok* port or *turn\_away* port with roughly the same chance. Therefore the model behaves correctly.

### 3.2.3 extraChecks Model Test

The extraChecks coupled model is made up of the following models: NewP1, carCheck, interview, Turn\_away\_N\_ok2, ExtraC\_NewP1. Refer to Figure 2 to see how the atomic models are connected in the extraChecks model. The extraChecks coupled model behaves the same way as it is explained in section 3.2.3 in [1].

The inputs to the extraChecks model are stored in a text file called *extraChecks\_input\_test.txt.* This file can be accessed in a similar way as the previous models’ input test files were accessed. The inputs to the extraChecks model were based on the inputs used to test the CD++ implementation of the model. The inputs to the CD++ extraChecks model are stored in an event file *extraChecks.ev.* The two input files are shown in Table 11 below.

Table : Inputs to the extraChecks model

|  |  |
| --- | --- |
| Inputs (.ev file) – CD++  (*extraChecks.ev)* | Inputs (.txt file) – CDBoost  (*extraChecks \_input\_test.txt)* |
| 00:00:10:00 new\_person 1  00:00:20:00 new\_person 2  00:00:30:00 new\_person 3  00:00:40:00 new\_person 4  00:00:50:00 new\_person 5  00:00:50:00 new\_person 6  00:00:50:00 new\_person 7  00:00:50:00 new\_person 8  **00:00:50:00 new\_person 9**  00:00:50:00 new\_person 10  00:00:50:00 new\_person 11  00:00:50:00 new\_person 12  00:03:00:00 new\_person 13 | 10 1 new\_person 1  20 1 new\_person 2  30 1 new\_person 3  40 1 new\_person 4  50 1 new\_person 5  50 1 new\_person 6  50 1 new\_person 7  50 1 new\_person 8  **50 1 new\_person 9**  50 1 new\_person 10  50 1 new\_person 11  50 1 new\_person 12  180 1 new\_person 13 |

Like in previous tests, if the outputs of the CDBoost extraChecks model (this is stored in a text file *extraChecks\_test\_output.txt)* match the outputs of the CD++ extraChecks model (this is stored in an out file called *extraChecksMAOUT.out*), this means that the extraChecks model behaves correctly. The two output files are shown below in.

Table : Outputs from the extraChecks model

|  |  |
| --- | --- |
| Outputs (.ev file) – CD++  (*extraChecksMAOUT.out*) | Outputs (.txt file) – CDBoost (*extraChecks\_test\_output.txt)* |
| 00:00:14:951 ok 1  00:00:20:184 ok 2  00:00:30:831 ok 3  00:00:45:141 ok 4  00:00:51:730 turn\_away 5  00:00:57:372 ok 6  00:01:05:771 turn\_away 7  00:01:13:391 ok 8  00:01:24:497 ok 10  00:01:32:245 ok 11  00:01:35:023 ok 12  **00:01:37:778 turn\_away 9**  00:03:07:712 ok 13 | 15/1 ok 1  29/1 ok 2  36/1 ok 3  43/1 ok 4  61/1 ok 5  71/1 ok 6  72/1 ok 7  74/1 ok 8  81/1 turn\_away 10  81/1 ok 11  84/1 ok 12  **96/1 turn\_away 9**  181/1 ok 13 |

In the CD++ extraChecks model test, an interesting behaviour of the model was discovered. “After the simulation was run, the output file *extraChecksMAOUT.out* was analyzed. From this file, we can see some interesting behavior: when person #9 (bolded in both the input and output files) came to the ExtraChecks model, the Car check sent them further to the Interview. While person #9 was in the Inverview (waiting an average of 20 seconds), persons 10 through 12 came to the ExtraChecks area and left before person #9 did. This demonstrates some of the model’s more interesting behavior, where a person may be leap-frogged if said person had to do more checks.” [1]

Interestingly, the CDBoost version of the extraChecks behaved the exact same way. With person 9 coming out of the model after person 10 to 12 had left. In addition, no person is lost through the model even though the persons do not come out in the same order as they entered the model. It is safe to say that the CDBoost implementation of the extraChecks model is correct.

## 3.3 Border Model Test

After all the models (coupled and atomic) had been tested, they were coupled together like in Figure 2 and the top model was tested. This model was defined in the main.cpp file stored in the borderTop folder. The borderTop coupled model behaves the same way as it is explained in [1] and will not be explained again here.

The inputs to the borderTop model are stored in a text file called *borderTop\_input\_test.txt.* This file can be accessed in a similar way as the previous atomic and coupled models input test files were accessed. The inputs to the borderTop model were based on the inputs used to test the CD++ implementation of the model. The inputs to the CD++ borderTop model are stored in an event file *border.ev.* The two input files are shown in Table 13 below.

Table : Inputs to the borderTop model

|  |  |
| --- | --- |
| Inputs (.ev file) – CD++  (*border.ev)* | Inputs (.txt file) – CDBoost  (*borderTop\_input\_test.txt)* |
| 00:00:10:00 new\_person 1  00:00:20:00 new\_person 2  00:00:30:00 new\_person 3  **00:00:40:00 new\_person 4**  00:00:50:00 new\_person 5  00:00:50:00 new\_person 6  00:00:50:00 new\_person 7  00:00:50:00 new\_person 8  00:00:50:00 new\_person 9  00:00:50:00 new\_person 10  00:00:50:00 new\_person 11  00:03:00:00 new\_person 12 | 10 1 new\_person 1  20 1 new\_person 2  30 1 new\_person 3  40 1 new\_person 4  50 1 new\_person 5  50 1 new\_person 6  50 1 new\_person 7  50 1 new\_person 8  50 1 new\_person 9  50 1 new\_person 10  50 1 new\_person 11  180 1 new\_person 12 |

Like in previous tests, if the outputs of the CDBoost borderTop model (this is stored in a text file *borderTop \_test\_output.txt)* match the outputs of the CD++ borderTop model (this is stored in an out file called *borderMAOUT.out*), this means that the borderTop model behaves correctly. The two output files are shown below in Table 14.

Table : Outputs from the borderTop model

|  |  |
| --- | --- |
| Outputs (.out file) – CD++  (*borderMAOUT.out*) | Outputs (.txt file) – CDBoost (*borderTop\_test\_output.txt)* |
| 00:00:16:951 ok 1  00:00:22:184 ok 2  00:00:32:831 turn\_away 3  00:01:00:399 ok 5  **00:01:09:513 ok 4**  00:01:10:019 ok 6  00:01:18:803 ok 7  00:01:25:125 ok 8  00:01:29:728 ok 9  00:01:42:254 ok 10  00:01:56:907 ok 11  00:03:14:039 ok 12 | 18/1 ok 1  30/1 ok 2  39/1 ok 3  48/1 ok 4  61/1 ok 5  68/1 ok 6  75/1 ok 7  81/1 ok 8  87/1 turn\_away 9  99/1 ok 10  101/1 ok 11  185/1 ok 12 |

“From the outputs in *borderMAOUT.out*, we can see that person #4 was required to go through a lot more extra checks than person #5, and although #4 arrived 10 seconds earlier than #5 he left 10 seconds after #5. The event in question is in bold. This shows that even though it does not happen very often (about 1% of the time with the current probabilities of the Decider atomic model) it is possible for one person to end up leaving the Border after someone else, even if they came before that person.” [1]

Unlike the outputs in *borderMAOUT.out,* the outputs from the CDBoost implementation of the border crossing model in *borderTop\_test\_output.txt* show that people leave the border in order. Now this does not mean that the borderTop model is incorrect because people leave the decider and secondaryDecider at random times and thus there might be no time occurrences like that described in the previous paragraph: when a person leaves the border earlier than someone who went into the border before him/her. However, there might be a case where this happens such as the case in *borderMAOUT.out* in Table 14. To prove this, the CD++ implementation of the border model was rerun and the following test results in Table 15 were generated. As can be seen in Table 15, all the outputs come out in order. It is safe to say that the CDBoost implementation of the border crossing model is correct.

Table : Alternative outputs from the borderTop model

|  |
| --- |
| Outputs (.out file) – CD++  (*borderMAOUT\_1.out*) |
| 00:00:16:951 ok 1  00:00:22:184 ok 2  00:00:32:831 ok 3  00:00:47:141 ok 4  00:00:53:730 ok 5  00:01:01:372 ok 6  00:01:19:391 ok 7  00:01:20:555 ok 8  00:01:26:877 ok 9  00:01:31:480 ok 10  00:01:41:228 ok 11  00:03:31:209 turn\_away 12 |

# References

|  |  |
| --- | --- |
| [1] | S. Stewart, "Border Crossing Model," Ottawa, 2016. |