Scott Stewart - 100850700

wtime@live.ca

Methedologies for Modelling discrete event systems

Border Crossing Model – 5/3/2016

Table of Contents

[1.0 Discrete-Event Dynamic System Model Proposal 2](#_Toc444970556)

[2.0 Conceptual Model 3](#_Toc444970557)

[2.1 Formal Specifications: Atomic Models 4](#_Toc444970558)

[2.1.1 Line 4](#_Toc444970559)

[2.1.2 Decider 4](#_Toc444970560)

[2.1.3 SecondaryDecider 5](#_Toc444970561)

[2.2 Formal Specifications: Coupled Models 6](#_Toc444970562)

[2.2.1 FrontDesk and Car 6](#_Toc444970563)

[2.2.2 Interview 6](#_Toc444970564)

[2.2.3 ExtraChecks 6](#_Toc444970565)

[2.3 Testing Strategy 7](#_Toc444970566)

[3.0 Simulation Analysis 7](#_Toc444970567)

[3.1 Atomic Model Analysis 7](#_Toc444970568)

[3.1.1 Line Model Analysis 7](#_Toc444970569)

[3.1.2 Decider Model Analysis 8](#_Toc444970570)

[3.1.3 Secondary Decider Model Analysis 10](#_Toc444970571)

[3.2 Coupled Model Analysis 12](#_Toc444970572)

[3.2.1 FrontDesk Model Analysis 12](#_Toc444970573)

[3.2.2 Interview Model Analysis 13](#_Toc444970574)

[3.2.3 ExtraChecks Model Analysis 14](#_Toc444970575)

[3.3 Border Model Analysis 15](#_Toc444970576)

# 1.0 Discrete-Event Dynamic System Model Proposal

The DEVS formalism for discrete-event systems can be used to model many different natural or artificial systems. The system that we will be exploring in detail during this assignment is a conventional border crossing, which gates the flow of people travelling from one country to another. This model needs to be able to demonstrate the different things that may happen when a person arrives at a border, such as: whether they are let through, if they are turned away, or if there are extra checks required before making a decision. The number of people who have successfully crossed as well as how long it takes them to cross then needs to be recorded for analysis.



Figure 1: Conceptual model for one-way border crossing

Figure 1 shows the conceptual model for the border crossing. When a person arrives the top model passes them to the ‘FrontDesk’ coupled model, which passes them to the ‘line’ atomic model. The ‘line’ holds all of the people in order of arrival, and will only pass the person at the front of the line to the ‘decider’ when it receives the signal from the ‘decider’ that it is ready for the next person. The ‘decider’ takes the person and, after a time delay, uses a random number generator to tell if they are instantly turned away from the border, are allowed to cross, or have to be passed further into the ‘extra checks’ coupled model. Whichever option it chooses, the ‘decider’ will then send a signal to the line that it is ready for the next person.

The ‘extra checks’ coupled model is composed of two separate coupled models which determine whether the person ultimately is allowed to cross or is denied: the ‘car check’ model takes the person from the desk and then after a time delay will allow them to cross, send them for additional checks, or turn them away. If a person arrives and the ‘car check’ is busy, it will put them in a line internally. If the person needs more checks, they are passed to an ‘interview’ model, which after a time delay will allow them to cross or ultimately turn them away. Both the ‘interview’ and ‘car check’ models have internal atomic models that hold onto the people entering them, making sure none of the people are lost.

The “turn away” and “ok” outputs from the coupled models are finally sent through the outputs of the border model, indicating if the person ultimately is turned away or is allowed to pass through the border.

## 2.0 Conceptual Model

In order to properly code our Border Crossing model, a conceptual model must first be developed to demonstrate our expected behavior. As seen from Figure 1 several atomic models and coupled models are needed in order to receive the person at the border through the input port (*new\_person*) and determined if they should be accepted through the border successfully (through *ok*) or should instead be turned away completely (through *turn\_away*).

The Border model is broken up into two different components: FrontDesk and ExtraChecks. The FrontDesk receives the people coming in from the input and looks at them in turn, deciding if they are ok to pass, are turned away completely, or have to go through extra checks. This model can be subdivided further into two atomic models, the Line and the Decider. The Line 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. 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’ *new\_person* input port.

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 Car check and the Interview. The Car is a coupled model that 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 Car’s *extra\_checks* port, it will then go into the Interview’s *new\_person* input port. Interview 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. Interview’s and Car’s 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.

## 2.1 Formal Specifications: Atomic Models

The formal specification for atomic models is: <S, X, Y, int, ext, , ta>. There are three different atomic models used in the Border model: Line, Decider, and SecondaryDecider.

### 2.1.1 Line

S = {ready, not\_ready, list<people>}

X = {new\_person, ready\_for\_next}

Y = {line\_out}

int (ready AND list is not empty) = not\_ready, remove front person from the list

ext (ready\_for\_next) = ready

ext (new\_person) { add the value from *new\_person* and put it at the end of the list }

(ready AND list is not empty) { send the person at the front of the line through *line\_out* }

ta(not\_ready OR list is empty) = INFINITY

ta(ready AND list is not empty) = 0

### 2.1.2 Decider

S = {passive, active}

X = {next\_person}

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

int (active) = passive

ext (next\_person, passive) = active

ext (next\_person, active) = active

(active)

{ Send person from *next\_person* to *ok* with 80% chance

Send person from *next\_person* to *extra\_checks* with 10% chance

Send person from *next\_person* to *turn\_away* with 10% chance

Regardless of the above, send *next\_person* to *is\_free*.

}

ta(passive) = INFINITY

ta(active) = delay (normal distribution)

### 2.1.3 SecondaryDecider

S = {passive, active}

X = {next\_person}

Y = {ok, turn\_away, is\_free}

int (active) = passive

ext (next\_person, passive) = active

ext (next\_person, active) = active

(active)

{ Send person from *next\_person* to *ok* with 50% chance

Send person from *next\_person* to *turn\_away* with 50% chance

Regardless of the above, send *next\_person* to *is\_free*.

}

ta(passive) = INFINITY

ta(active) = delay (normal distribution)

## 2.2 Formal Specifications: Coupled Models

The formal specifications for the coupled models is: <X, Y, D, EIC, EOC, IC, SELECT>. There are four different coupled models: FrontDesk, ExtraChecks, Car, and Interview. FrontDesk and Car both use the same coupled model, so they will be defined together.

### 2.2.1 FrontDesk and Car

X = {new\_person}

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

D = {line, decider}

EIC = {(self.new\_person, line.new\_person)}

EOC = {(decider.ok, self.ok), (decider.extra\_checks, self.extra\_checks), (decider.turn\_away, self.turn\_away)}

IC = {(line.line\_out, decider.next\_person), (decider.is\_free, line.ready\_for\_next)}

SELECT: ({decider, line}) = decider;

### 2.2.2 Interview

X = {new\_person}

Y = {ok, turn\_away}

D = {line, secondaryDecider}

EIC = {(self.new\_person, line.new\_person)}

EOC = {(secondaryDecider.ok, self.ok), (secondaryDecider.turn\_away, self.turn\_away)}

IC = {(line.line\_out, secondaryDecider.next\_person), (secondaryDecider.is\_free, line.ready\_for\_next)}

SELECT: ({secondaryDecider, line}) = secondaryDecider;

### 2.2.3 ExtraChecks

X = {new\_person}

Y = {ok, turn\_away}

D = {Car, Interview}

EIC = {(self.new\_person, Car.new\_person)}

EOC = {(Car.ok, self.ok), (Interview.ok, self.ok), (Car.turn\_away, self.turn\_away), (Inverview.turn\_away, self.turn\_away)}

IC = {(Car.extra\_checks, Interview.new\_person)}

SELECT: ({Car, Interview}) = Interview;

## 2.3 Testing Strategy

A testing strategy needed to be formulated and executed in order to determine if the overall model was working as intended. In order to verify this, the model was tested as follows:

* Each of the atomic models is to be tested using a variety of inputs through an event file (.ev) and the resultant output file (.out) has to be analyzed to determine if the behavior of the model matched what was required.
* The coupled models should be constructed and then analyzed in the same manner as the atomic models: input an event file and examine the results through the output file. If the behavior of the coupled model matched what was expected, it could be used in further coupled model tests.
* Once all of the individual components are tested, the full border crossing model needs to be tested with a variety of inputs and analyzed. Any abnormal behavior has to be noted and remarked upon.

# 3.0 Simulation Analysis

## 3.1 Atomic Model Analysis

### 3.1.1 Line Model Analysis

As discussed previously, the Line atomic model should take inputs from the *new\_person* port and place them in an internal ordered list. If the Line receives any input from the *ready\_for\_next* port, it should output the person at the head of the line through the port *line\_out* immediately and then passivate itself. If an input from *ready\_for\_next* is received while the internal list is empty, an input from *new\_person* should immediately send the new person through the port *line\_out* and then passivate itself again. Finally, the internal list must respect the order of arrivals, regardless of how long the people are in the line.

The event file *line.ev* was used in conjunction with the .ma file *lineMA.ma* in order to test all of the cases. As a side note, because the model is assumed to be completely empty at the beginning of the simulation, the line should pass the first person through immediately.

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

The event in bold is an extra *ready\_for\_next* input that should not cause any extra outputs, since at 00:01:10:00 the line should already be empty. At 00:01:20:00, 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 received at 00:02:00:00 in the input port should not be seem through the output port because it did not receive any additional *ready\_for\_next* signals. The outputs from this simulation were examined through the file *lineMAOUT.out*.

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

From this file, we can see that the Line atomic model did in fact follow the expected behavior of the model.

## 3.1.2 Decider Model Analysis

The Decider atomic model is slightly different and for faultless execution it requires that it is hooked up to a Line atomic model. The Decider has one input *next\_person* to represent people coming up to a desk, has three outputs (*ok*, *extra\_checks*, and *turn\_away*) to represent how the person was handled, and an output *is\_free* to signal the line that the next person can come.

The Decider for the test, using the event file *decider.ev* and the .ma file *deciderMA.ma*, had a time advance function with a normal distribution: its mean value was 5 seconds and its standard deviation was 4 seconds. The event file put twenty people through the input of the decider at an interval of 10 seconds between people. In order to consider the test successful, we needed to see that each of the “decision” output ports had at least one event and that for every decision an output is sent through *is\_free* to tell the Line that the next person may come.

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

Above is the event file that was used to test the Decider model. As it can be seen from the output file *deciderMAOUT.out*, the input 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.

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

From the rest of the output file, it can be seen that each of the “decision” output ports was visited at least once, with *ok* receiving a lot more outputs than either *extra\_checks* or *turn\_away*. Also, *is\_free* is fired each time the “decision” ports were fired, indicating that the Decider is properly signaling that more people can come to it. From the above output, it is safe to say that the Decider model is behaving as expected.

### 3.1.3 Secondary Decider Model Analysis

The SecondaryDecider atomic model is very similar to the Decider. The only real change is that it has only two “decision” output ports: *turn\_away* and *ok*. Because of this, the testing of the SecondaryDecider is very similar to that of the Decider.

The event file *secondaryDecider.ev* was used in conjunction with the .ma file *secondaryDeciderMA.ma* for the initial testing. The event file has only ten different inputs into the *next\_person* input port at 10 second intervals.

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

From this simulation, we get the output file *secondaryDeciderMAOUT.ma*.

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

We can see from this simulation that, similarly to the Decider atomic model, 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.

## 3.2 Coupled Model Analysis

### 3.2.1 FrontDesk Model Analysis

Now that the atomic models have been tested, the atomic models need to be put together in order to determine if they work well together. The first coupled model is the FrontDesk, which is used the same way for the Car coupled model, as discussed previously. The FrontDesk uses a Line in order to keep hold of the people being sent through its input and send them to the Decider when the Decider is free. If more than one person arrives while the Decider is busy, the model should hold on to the extra people and serve them in order, sending them ultimately through the three “decision” ports: *ok*, *turn\_away*, and *extra\_checks*, with an 80%, 10%, and 10% probability, respectively.

The coupled model is defined in its .ma file *frontDeskMA.ma*, and uses the event file *frontDesk.ev* for its inputs.

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

This event file inputs five people in a row into the FrontDesk in 10 second increments. At the fifth person, it instead simulates if seven people arrive at the exact same time. It then waits for a long time and inputs one more person. Below is the output file, *frontDeskMAOUT.out*, which shows the response to these inputs.

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

From the output file, 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.

### 3.2.2 Interview Model Analysis

The Interview coupled model is very similar to the FrontDesk coupled model. The coupled model is made up of a Line and a SecondaryDecider atomic model, linked up together in the same way as the Line and Decider models that the FrontDesk had. Additionally, the two “decision” ports are *ok* and *turn\_away*, which occur with a 50/50 split. Otherwise, the behavior of the Interview model behaves the same way as the FrontDesk model.

The coupled model is defined in its .ma file *interview.ma*, and uses the event file *inverview.ev* for its inputs. To test its similarity, the same inputs are used.

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

Below is the output of the Inverview simulation, which is obtained from *interviewMAOUT.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 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

The output 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.

### 3.2.3 ExtraChecks Model Analysis

The ExtraChecks coupled model is more complex than the previous models. It consists of an Interview and a Car coupled model, where the Car’s *extra\_checks* output port is connected to the Interview’s *new\_person* input port. The idea with this model is that when a person requires extra checks they must be sent through the ExtraChecks coupled model. First the person has to submit to a car inspection (the Car model) where they will either be ok to pass the boarder, be turned away, or require further checks. If they need extra checks, the person will need to go through an interview (the Interview model) where they will finally be labeled as ok to pass through the border or will ultimately be turned away.

In this coupled model, defined by the .ma file *extraChecksMA.ma*, the Decider model in the Car coupled model has a time advance function with a mean value of 5 seconds and the SecondaryDecider in the Interview coupled model has a time advance function with a mean value of 20 seconds: this is done so that it can be easily seen if the person is sent through extra checks to the Interview or if they are turned away/labeled ok through the Car model. The event file used for this simulation was the *extraChecks.ev* file.

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

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.

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

From the above output, we can see that the ExtraChecks coupled model is working as intended, successfully pushing the people through either the output port *ok* or *turn\_away*. Additionally, we can see that although the people do not all leave in the same order (which is an interesting feature of the model), none of the people are lost through the model. That said, a jail coupled model would be very interesting to add to the model.

### 3.3 Border Model Analysis

Once all of the models have been thoroughly tested, the models were put together in order to create the Border model. This top model was constructed by taking the *extra\_checks* output port from the FrontDesk coupled model and feeding it into the *new­\_person* input port of the ExtraChecks coupled model. The *ok* and *turn\_away* output ports of the FrontDesk and ExtraChecks models are fed through the output ports of the same name of the Border model, and the *new\_person* input port of the Border model is fed to the *new\_person* input port of the FrontDesk model. The coupled model is defined by its .ma file, *borderMA.ma*.

Because the Border model takes in people through the *new\_person* port similarly to the other coupled models, an event file *border.ev* was constructed for testing. This event file consisted of four people entering the Border with a 10 second increment between them, followed by seven people entering at the same time, and finally one more person entering a long time after.

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

Below is the output file generated from the simulation, *borderMAOUT.out*. From this output, 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.

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

From this test, we can see that the Border model meets the specifications that were outlined in the design document. The atomic models Line, Decider, and SecondaryDecider all behaved as expected when testing them individually as well as in different configurations in various coupled models. What is more, when coupled together as outlined in Section 1.0, the resulting Border coupled model followed the expected behavior very well with no modifications needed to the overall design.