# **Modeling Discrete Event Systems Using DEVS**

# **(2012 Fall)**

**Assignment 1- Simulation of passengers’ boarding at airport**

Pei Jin

7014544

University of Ottawa

Content

**Modeling Discrete Event Systems Using DEVS (2012 Fall)** 1

Part I 3

**a)** **Description** 3

**b)** **Model structure** 3

**c)** **Components** 4

**d)** **Behavior of component** 4

Part II 4

Formal Specifications of atomic models 5

Formal specifications of coupled models 7

Test Strategies 8

Part III 8

**1.** **Atomic model “buffer”** 8

**2.** **Atomic model “proc”** 10

**3.** **Atomic model “luggage”** 11

**4.** **Atomic model “custom”** 13

**5.** **Atomic model “security”** 14

**6.** **Coupled Model “Queue”** 16

**7.** **Coupled Model “Servicecenter”** 17

**8.** **The top model “airportboarding”** 18

Conclusion 22

## Part I

1. **Description**

The goal of building this simulator is that I want to simulate the whole procedure of airport services that an international passenger will probably experience at the airport when he wants to travel abroad. Sometimes, there are too many international passengers showing up at the airport, and they are supposed to deal with check-in and a series of check service provided by airports in order before going aboard, thus simulating the whole service procedure is necessary, which will probably help the airport to handle thousands of people and filter passengers do not obey the rules of airports.

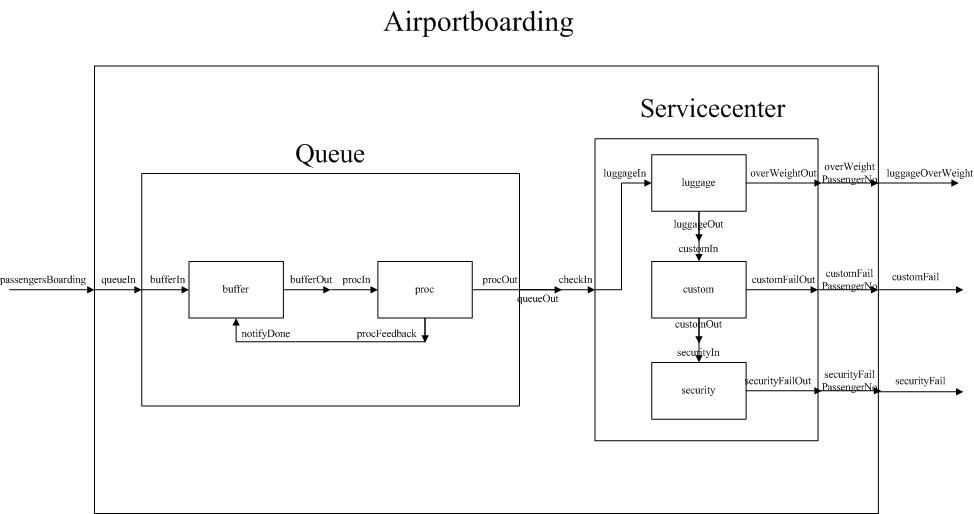


Figure 1: Structure of Airportboarding Simulator

1. **Model structure**
2. Queue model

This model is used to make all the passengers queue up, be processed in order and distribute those passengers who want to travel abroad (international passengers) to the service center.

1. Servicecenter model

This model is aimed to make travelers’ accept a series of check process (luggage, custom and security check) before going aboard. As the result, unsatisfied passengers are selected out.

1. **Components**
2. buffer
3. proc
4. luggage
5. custom
6. security
7. **Behavior of component**
8. buffer: This atomic model is used to make all the passengers waiting for handling procedures in a queue. Once it receives the feedback from the proc model, it will let the next passenger go ahead.
9. proc: It distributes the new passenger to the right service center. If he travels abroad, he will go to the service center for international passengers.
10. luggage: It could check whether the weight of passengers’ luggage is inside the recommended limit.
11. custom: It could check whether the passengers’ items satisfy standard
12. security: It could check whether things carried on the passenger are dangerous.

## Part II

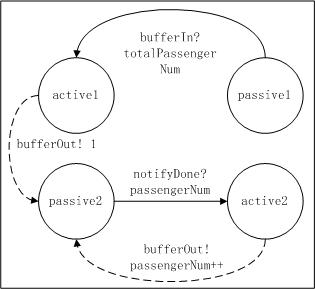
As we can see in the figure 1, the airport boarding simulator has 1 input and 3 outputs. The *passengersboarding* means how many passengers (including both the domestic and international passengers) will go aboard. The 3 outputs are showing the unsatisfied passenger who fails in one of the series of check (that is, luggage weight *(luggageOverWeight*), custom (*customfail*) and security (*securityfail*)). In terms of architecture, the airport boarding is made up of 2 components: Queue model and Servicecenter model. In addition, the Queue model is decomposed to the 2 components: buffer and proc, which indicates passengers are queuing up, receive the procedure provided by proc, and then know where they should go (the international service center or not). Also, the Servicecenter model consists of 3 components: luggage, custom and security. They are assumed to have the similar behavior which can check whether the passenger obey the recommended standard.

### Formal Specifications of atomic models

The more concrete DEVS formalism with port specifications is as follows:

DEVS = ( X, Y, S, δext, δint, λ, ta )

#### buffer



X = {bufferIn, notifyDone}

Y = {1, passengerNum}

S = {“active1”, “active2”, “passive1”, “passive2”}

δext (“passive1”,totalPassengerNum) = “active1”

δext (“passive2”, notifyDone) = “active2”

δint (“active1”, 1) = “passive1”

δint (“active2”, passengerNum++) = “passive2”

λ(“active1”, bufferTime) = 1

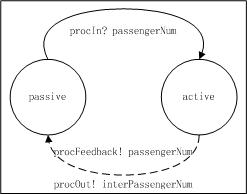
λ(“active2”, bufferTime) = passengerNum

ta(“active1”, bufferTime) = 5

ta(“active2”, bufferTime) = 5

ta(“passive”) = ∞

1. **proc**



X={procIn}

Y= {procOut, procFeedback}

S= {“passive”, “active”}

δext (“passive”, passengerNum) = “active”

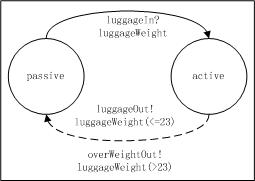
δint (“active”) = “passive”

λ(“active”, procTime) = passengerNum if outport = procFeedback

= internationalPassengerNum if outport = procOut

ta(“active”, procTime) = 5

1. **luggage**

****

S = {“passive”, “active”}

X = {luggageIn}

` Y = {luggageOut, overWeightOut}

δext (“passive”, luggageWeight) = “active”

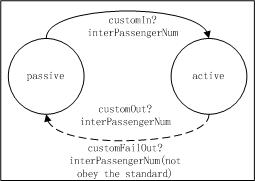
δint (“active”) = “passive”

λ(“active”, luggageTime) = luggageWeight (<=23) if outport = luggageOut

= luggageWeight (>23) if outport = overWeightOut

ta(“active”, luggageTime) = 5

1. **custom**

****

S = {“passive”, “active”}

X = {customIn}

` Y = {customOut, customFailOut}

δext (“passive”, interPassengerNum) = “active”

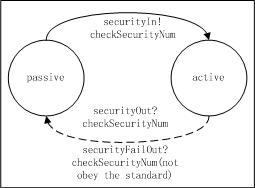
δint (“active”) = “passive”

λ(“active”, customTime) = interPassengerNum (satisfies) if outport = customOut

= interPassengerNum (not) if outport = customFailOut

ta(“active”, customTime) = 15

1. **security**



S = {“passive”, “active”}

X = {securityIn}

` Y = {securityOut, securityFailOut}

δext (“passive”, checkSecurityNum) = “active”

δint (“active”) = “passive”

λ(“active”, securityTime) = checkSecurityNum (satisfies) if outport = securityOut

= checkSecurityNum (not) if outport = securityFailOut

ta(“active”, securityTime) = 10

### Formal specifications of coupled models

N= (X, Y, D, EIC, EOC, IC) for the coupled model Queue, Servicecenter and Airportboarding Simulator are defined as follows:

#### Queue

X = {queueIn }

Y = {queueOut}

D = {buffer, proc}

EIC = {((Queue, “queueIn”), (buffer, “bufferIn”)}

EOC = {(proc, “procOut”) , (Queue, “queueOut”)}

IC = {((buffer, “bufferOut”),(proc, “procIn”)),

((proc, “procFeedback”),(buffer, “notifyDone”))}

1. **Servicecenter**

X = {checkIn }

Y = {overWeightPassengerNo, customFailPassengerNo, securityFailPassengerNo}

D = {luggage, custom, security}

EIC = {(Servicecenter, “checkIn”), (luggage, “luggageIn”)}

EOC = {((luggage, “overWeightOut”), (Servicecenter, “overWeightPassengerNo”)),

((custom, “customFailOut”), (Servicecenter, “customFailPassengerNo”)),

((security, “securityFailOut”),(Servicecenter, “securityFailPassengerNo”))}

IC = {((luggage, “luggageOut”),(custom, “customIn”)),

((custom, “customOut”), (security, “securityIn”))}

1. **Airportboarding**

X = {passengersBoarding};

Y = {luggageOverWeight, securityFail, customFail};

D = {queue, servicecenter};

EIC = {(Airportboarding, “passengersBoarding”), (Queue, “queueIn”)}

EOC = {((Servicecenter, “overWeightPassengerNo”), ( Airportboarding, “luggageOverWeight”)),

((Servicecenter, “customFailPassengerNo”), ( Airportboarding, “customFail”)),

((Servicecenter, “securityFailPassengerNo”),( Airportboarding, “securityFail”))}

IC = {((Queue, “queueOut”), ( Servicecenter, “checkIn”))

### Test Strategies

I will use the “black box” testing method to test atomic models and coupled models in order, which is the best method to check the internal transitions are correct or not. As for different atomic and coupled models, I will give different inputs to the corresponding event files (*.ev*), run each simulation and check whether outputs in output files (*.out*) are what I am expected to see.

As for my project, the sequence of testing should be following as:

1. “buffer” and “proc” atomic models

2. “Queue” coupled model

3. “luggage”, “custom” and “security” atomic models

4. “Servicecenter” coupled model

5. “Airportboarding” the top model

## Part III

In order to ensure the correctness of the whole simulation, analyze the behavior of the atomic model is of importance.

1. **Atomic model “buffer”**

The input of buffer should be a positive integer represents the number of passengers who will get on the plane. No matter how large is this integer, as long as it is a positive integer, it should start to send from 1 indicates the first passenger. After sending, it should wait for the feedback at the port “notifyDone”. Once it receives the feedback and then it could send the next passenger (2, 3, 4 …)

So I write the *buffer.ev* file as follows:

00:00:10:00 bufferIn 11

00:00:16:00 notifyDone 1

00:00:22:00 notifyDone 2

00:00:28:00 notifyDone 3

**00:00:32:00 notifyDone 4**

00:00:40:00 notifyDone 5

**00:00:42:00 notifyDone 6**

00:00:52:00 notifyDone 7

00:00:58:00 notifyDone 8

00:01:04:00 notifyDone 9

00:01:10:00 notifyDone 10

00:01:16:00 notifyDone 11

The 2 bold events (line 5 and 7) should not generate outputs because the “bufferTime” is 5 which means it needs 5 seconds to execute the passenger. If at that time, the feedback is arrived in advance, buffer will ignore it at current and handles the new one. Take the line 5 as an example, at 32 second; the input receives 4 but 3 has not finished, so buffer ignores the on-going one and handles 4. One should be noticed is that the elapsed time is changed from 4 to 0, so the output is 5 at 37 second. The output file *buffer.out* file shows the expected result as follows:

00:00:15:000 bufferout 1

00:00:21:000 bufferout 2

00:00:27:000 bufferout 3

00:00:37:000 bufferout 5

00:00:47:000 bufferout 7

00:00:57:000 bufferout 8

00:01:03:000 bufferout 9

00:01:09:000 bufferout 10

00:01:15:000 bufferout 11

1. **Atomic model “proc”**

The input of the proc should be an integer as well, and then it handles the data it received, making it BIT AND with 1. If the result is 1, it will send from the port “procFeedback” and the port “procOut”, otherwise it will send from the port “procFeedback” only. So the test file “proc.ev” is created as follows:

00:00:10:00 procIn 1

00:00:16:00 procIn 2

**00:00:19:00 procIn 6**

00:00:28:00 procIn 7

00:00:34:00 procIn 8

**00:00:35:00 procIn 16**

00:01:28:00 procIn 27

00:02:34:00 procIn 38

**00:02:36:00 procIn 39**

00:04:34:00 procIn 43

00:05:34:00 procIn 53

00:06:20:00 procIn 65

00:07:34:00 procIn 77

00:08:00:00 procIn 88

The 3 bold events (line 3, 6 and 9) should not generate outputs because the “procTime” is 5 which means it needs 5 seconds to execute the passenger. If at that time, a new input arrived in advance, buffer will ignore it because it is in the “active” mode at current. Take the line 3 as an example, at 19 seconds; the input receives 6 but 2 has not finished, so buffer ignores the 6 directly rather than handle it, which is opposite to the “buffer” model. One should be noticed is that the elapsed time does not change although the new data is coming, so the input 2 is sent at 21 second without changing. The output file *proc.out* file shows the expected result as follows:

00:00:15:000 procout 1

00:00:15:000 procfeedback 1

00:00:21:000 procfeedback 2

00:00:33:000 procout 7

00:00:33:000 procfeedback 7

00:00:39:000 procfeedback 8

00:01:33:000 procout 27

00:01:33:000 procfeedback 27

00:02:39:000 procfeedback 38

00:04:39:000 procout 43

00:04:39:000 procfeedback 43

00:05:39:000 procout 53

00:05:39:000 procfeedback 53

00:06:25:000 procout 65

00:06:25:000 procfeedback 65

00:07:39:000 procout 77

00:07:39:000 procfeedback 77

00:08:05:000 procfeedback 88

1. **Atomic model “luggage”**

The input of this model is odd integers indicates the international passengers. Then it handles the data it received, making it BIT AND with 31. If the result is larger than 23, it will send from the port “overWeightOut”, otherwise it will be sent from the port “luggageOut” only. So the test file “luggage.ev” is created as follows:

00:00:00:00 luggageIn 22

00:00:20:00 luggageIn 23

00:00:25:00 luggageIn 24

00:01:10:00 luggageIn 26

00:02:10:00 luggageIn 56

00:02:20:00 luggageIn 122

00:02:30:00 luggageIn 125

00:02:40:00 luggageIn 167

00:02:50:00 luggageIn 178

00:03:00:00 luggageIn 179

00:04:20:00 luggageIn 223

00:04:22:00 luggageIn 225

00:04:50:00 luggageIn 245

00:05:10:00 luggageIn 256

00:05:30:00 luggageIn 278

The output should generate different outputs depending on different integers. It is similar to the “proc” model when the new data is input but the former one does not be handled yet, so the new one will just be dropped, which makes sense in reality as well. The output file *luggage.out* file shows the expected result as follows:

00:00:09:000 luggageout 22

00:00:29:000 luggageout 23

00:01:19:000 overweightout 26

00:02:19:000 overweightout 56

00:02:29:000 overweightout 122

00:02:39:000 overweightout 125

00:02:49:000 luggageout 167

00:02:59:000 luggageout 178

00:03:09:000 luggageout 179

00:04:29:000 overweightout 223

00:04:59:000 luggageout 245

00:05:19:000 luggageout 256

00:05:39:000 luggageout 278

1. **Atomic model “custom”**

The input of this model is odd integers indicates the international passengers. Then it handles the data it received, making it BIT AND with 15. If the result is 15 which indicates the passenger does not satisfy the standard, but not all the unsatisfied passengers are sent from the “customFail” output, some of them will have 10% chance to escape the check implementing by random function, which reflects the true fact. Due to the low chance, so I put many integers which should be possibly sent from the “customFailOut”.

So the test file “custom.ev” is created as follows:

00:00:00:00 customIn 15

00:00:11:00 customIn 31

00:00:31:00 customIn 47

00:00:56:00 customIn 63

00:00:58:00 customIn 64

00:01:50:00 customIn 79

00:02:10:00 customIn 111

00:02:30:00 customIn 112

00:03:10:00 customIn 123

00:03:30:00 customIn 127

00:03:35:00 customIn 143

00:04:00:00 customIn 175

00:04:30:00 customIn 177

00:05:00:00 customIn 239

00:05:20:00 customIn 303

00:05:36:00 customIn 367

00:05:56:00 customIn 368

The result of BIT AND algorithm should decide which port the passenger should be sent from, so most of them (15, 31, 47, 63, 79, 111, 127, 143, 175, 239, 303, 367) should be sent from the port “customFailOut”, while there is 10% possibility that they could be sent from the port “customOut”. When the new data is input but the former one does not be handled yet, so the new one will just be dropped (e.g. line 5). The output is not deterministic due to the random function. The following is an example of the output file *custom.out.* In this output file,the event with **bold** fonts means this passenger escapes the airport custom check.

00:00:15:000 customfailout 15

00:00:46:000 customfailout 47

00:01:11:000 customfailout 63

00:02:05:000 customfailout 79

00:02:25:000 customfailout 111

00:02:45:000 customout 112

00:03:25:000 customout 123

00:03:45:000 customfailout 127

**00:04:15:000 customout 175**

00:04:45:000 customout 177

00:05:15:000 customfailout 239

00:05:35:000 customfailout 303

00:05:51:000 customfailout 367

00:06:11:000 customout 368

1. **Atomic model “security”**

This model has the similar function to the “custom”. The input of this model is odd integers indicates the international passengers. Then it handles the data it received, making it BIT AND with 63. If the result is 49 which indicates the passenger does not satisfy the standard, but not all the unsatisfied passengers are sent from the “securityFail” output, some of them will have 20% chance to escape the check implementing by random function, which reflects the true fact. Due to the low chance, so I put many integers which should be possibly sent from the “securityFailOut”.

So the test file “security.ev” is created as follows:

00:00:00:00 securityIn 46

00:03:00:00 securityIn 47

00:04:00:00 securityIn 48

00:05:00:00 securityIn 49

00:06:00:00 securityIn 51

00:07:00:00 securityIn 53

00:08:00:00 securityIn 113

00:09:00:00 securityIn 177

00:10:00:00 securityIn 241

The result of BIT AND algorithm should decide which port the passenger should be sent from, so most of them (49, 113, 177, 241, 305, 369) should be sent from the port “securityFailOut”, while there is 20% possibility that they could be sent from the port “securityOut”. The output is not deterministic due to the random function. The following is an example of the output file *security.out.* In this output file,the event with **bold** fonts means this passenger escapes the airport security check.

00:00:10:000 securityout 46

00:03:10:000 securityout 47

00:04:10:000 securityout 48

00:05:10:000 securityfailout 49

00:06:10:000 securityfailout 51

00:07:10:000 securityfailout 53

00:08:10:000 securityfailout 113

00:09:10:000 securityfailout 177

**00:10:10:000 securityout 241**

1. **Coupled Model “Queue”**

The coupled model Queue is the second top model which consists of “buffer” and “proc” atomic models. All passengers enter into the queue model, and then go into the “buffer” which calculates how many passengers will go boarding, after which they enter the “proc” notifies where international passengers should go next ( the other coupled model “Servicecenter”). At the same time, once the “proc” has handled one passenger, it will notify the buffer that it is empty now and next passenger could come on.

The input of this coupled model is “queueIn”, a positive integer indicating the number of passengers who want to get on the plane. The output is “queueOut” which indicates the number of passengers who want to travel abroad.

The test file “*queue.ev*” is simple with only one line as follows. 15 passengers want to get on the airplane.

00:00:00:00 queueIn 15

The output should be a series of odd integer numbers which indicate international passengers. As I simulate at the time 00:00:00:00, going through the bufferTime and procTime for each passenger, the time of output should be at 00:00:10:00 for the first passenger. Sequentially, the second passenger ( the integer 2 ) implemented the BIT AND algorithm and then the result is 0 which means he is the domestic passenger, so this is not sent from the port “queueOut”. That is the reason why the third passenger is sent out at the 00:00:30:000 (after handling 2 passengers). The following is the output file *queue.out.*

00:00:10:000 queueout 1

00:00:30:000 queueout 3

00:00:50:000 queueout 5

00:01:10:000 queueout 7

00:01:30:000 queueout 9

00:01:50:000 queueout 11

00:02:10:000 queueout 13

00:02:30:000 queueout 15

1. **Coupled Model “Servicecenter”**

The coupled model Queue is also the second top model which consists of “luggage”, “custom” and “security” atomic models. All international passengers enter into the servicecenter model, and then go into the “luggage” which calculates how much the weight of passengers’ luggage take, after which they enter the “custom” and “security” atomic models to receive the corresponding check. In any of three check programs, the passenger does not obey the rule of airport; the passenger will probably fail to get on the airplane. But some of them has the certain probability to escape such things, if they are lucky enough.

The input of this coupled model is “checkIn”, a positive integer indicating the number of international passengers who want to get on the plane. The outputs are “overWeightPassengerNo”, “customFailPassengerNo” and “securityFailPassengerNo” which indicate the number of passengers who do not obey the rules set by the airport.

The test file “*servicecenter.ev*” is not simple as the “Queue” coupled model as follows.

00:10:00:00 checkIn 15

00:20:00:00 checkIn 29

00:30:00:00 checkIn 31

00:40:00:00 checkIn 47

00:50:00:00 checkIn 49

00:55:00:00 checkIn 109

01:10:00:00 checkIn 111

01:30:00:00 checkIn 113

01:50:00:00 checkIn 127

01:55:00:00 checkIn 177

02:10:00:00 checkIn 341

03:40:00:00 checkIn 343

04:40:00:00 checkIn 135

05:50:00:00 checkIn 327

06:00:00:00 checkIn 233

The output should be many unsatisfied passengers who fail in any of three check processes, but it is not fixed due to the random functions. According to the result, I simulate at the time 00:10:00:00. The No.15 passenger is not checked successfully as its random number is larger than 0.9. The No.29 passenger is checked successfully during the first stage check, taking 9 seconds (luggageTime) and the time of output should be at 00:20:09:000. As for the No.47 passenger, he experiences 2 steps of check and is picked up because he does not obey the custom rule, taking 9 seconds luggageTime and 15 seconds (customTime). The following is an example of output files *servicecenter.out.*

00:20:09:000 overweightpassengerno 29

00:30:09:000 overweightpassengerno 31

00:40:24:000 customfailpassengerno 47

00:50:34:000 securityfailpassengerno 49

01:10:24:000 customfailpassengerno 111

01:50:09:000 overweightpassengerno 127

01:55:34:000 securityfailpassengerno 177

1. **The top model “airportboarding”**

The top model consists of 2 coupled models “Queue” and “Servicecenter”. The input of the top model should be an integer (larger than 0) which indicates the total number of passengers. The output should be international passengers who fail in any of three check process. In order to get a larger samples to analyze, I set the “passengersBoarding” equal to 567. So the file *airportboarding.ev* is created as follows:

00:00:00:00 passengersBoarding 567

After going through the “Queue” model, total passengers are divided into 2 parts- international and domestic passengers. International passengers are represented by the odd numbers. Then they receive a series of checks implemented by the algorithm BIT AND, and then some passengers will fail in one of three check process, but some will escape due to the random function. In terms of output time, it can be obtained by calculation (for the first 24 passengers who do not enter the servicecenter, they need (5(bufferTime) + 5(procTime))×24 = 240s. For the No.25 passenger, the required time is 5(bufferTime) + 5(procTime)+ 9(luggageTime)=19s. So the first output “luggageoverweight 25” should be at the time 00:04:19:000 (240s+19s=259s). The following is an example of output files *airportboarding.out.*

00:04:19:000 luggageoverweight 25

00:04:39:000 luggageoverweight 27

00:04:59:000 luggageoverweight 29

00:05:19:000 luggageoverweight 31

00:08:44:000 securityfail 49

00:09:39:000 luggageoverweight 57

00:09:59:000 luggageoverweight 59

00:10:19:000 luggageoverweight 61

00:10:39:000 luggageoverweight 63

00:13:34:000 customfail 79

00:14:59:000 luggageoverweight 89

00:15:19:000 luggageoverweight 91

00:15:39:000 luggageoverweight 93

00:15:59:000 luggageoverweight 95

00:18:54:000 customfail 111

00:19:24:000 securityfail 113

00:20:19:000 luggageoverweight 121

00:20:39:000 luggageoverweight 123

00:20:59:000 luggageoverweight 125

00:21:19:000 luggageoverweight 127

00:24:14:000 customfail 143

00:25:39:000 luggageoverweight 153

00:25:59:000 luggageoverweight 155

00:26:19:000 luggageoverweight 157

00:26:39:000 luggageoverweight 159

00:29:34:000 customfail 175

00:30:59:000 luggageoverweight 185

00:31:19:000 luggageoverweight 187

00:31:39:000 luggageoverweight 189

00:31:59:000 luggageoverweight 191

00:34:54:000 customfail 207

00:36:19:000 luggageoverweight 217

00:36:39:000 luggageoverweight 219

00:36:59:000 luggageoverweight 221

00:37:19:000 luggageoverweight 223

00:40:14:000 customfail 239

00:41:39:000 luggageoverweight 249

00:41:59:000 luggageoverweight 251

00:42:19:000 luggageoverweight 253

00:42:39:000 luggageoverweight 255

00:45:34:000 customfail 271

00:46:59:000 luggageoverweight 281

00:47:19:000 luggageoverweight 283

00:47:39:000 luggageoverweight 285

00:47:59:000 luggageoverweight 287

00:50:54:000 customfail 303

00:51:24:000 securityfail 305

00:52:19:000 luggageoverweight 313

00:52:39:000 luggageoverweight 315

00:52:59:000 luggageoverweight 317

00:53:19:000 luggageoverweight 319

00:56:14:000 customfail 335

00:57:39:000 luggageoverweight 345

00:57:59:000 luggageoverweight 347

00:58:19:000 luggageoverweight 349

00:58:39:000 luggageoverweight 351

01:01:34:000 customfail 367

01:02:04:000 securityfail 369

01:02:59:000 luggageoverweight 377

01:03:19:000 luggageoverweight 379

01:03:39:000 luggageoverweight 381

01:03:59:000 luggageoverweight 383

01:06:54:000 customfail 399

01:08:19:000 luggageoverweight 409

01:08:39:000 luggageoverweight 411

01:08:59:000 luggageoverweight 413

01:09:19:000 luggageoverweight 415

01:12:14:000 customfail 431

01:12:44:000 securityfail 433

01:13:39:000 luggageoverweight 441

01:13:59:000 luggageoverweight 443

01:14:19:000 luggageoverweight 445

01:14:39:000 luggageoverweight 447

01:17:34:000 customfail 463

01:18:59:000 luggageoverweight 473

01:19:19:000 luggageoverweight 475

01:19:39:000 luggageoverweight 477

01:19:59:000 luggageoverweight 479

01:22:54:000 customfail 495

01:23:24:000 securityfail 497

01:24:19:000 luggageoverweight 505

01:24:39:000 luggageoverweight 507

01:24:59:000 luggageoverweight 509

01:25:19:000 luggageoverweight 511

01:28:14:000 customfail 527

01:29:39:000 luggageoverweight 537

01:29:59:000 luggageoverweight 539

01:30:19:000 luggageoverweight 541

01:30:39:000 luggageoverweight 543

01:33:34:000 customfail 559

01:34:04:000 securityfail 561

### Conclusion

The Airportboarding Simulator model simulates the process of boarding of international passengers. In addition, the hierarchical test for both atomic models and coupled models generate the expected results. The output data is reasonable and the time of output can be calculated according to the time set in each model. The information of luggage, custom and security of passengers can be obtained by passengers’ number (in binary) in my model. The execution results and the behavior of the model match the specifications of models. The Airportboarding model works exactly as I expect according to the specifications.