Modelling Discrete-Event Systems Using DEVS (2016 Fall)

Assignment 1:

Student Academic Registration System

# Srivastav Reddy Janapalli

# #101054575

# Carleton University

# Part 1

Student registration system model consists of mainly two major blocks. In this simulation model, student’s GPA, preferred courses and addition marks are assumed based on *probability* to simulate some real-world scenarios through web portal by filling the application. After filling the application There’s a chance that the application might be with drawn so application process takes 70% probability of submitting it.

The data is pushed to University’s block where it is queued for registrar’s approval. Once the registrar checks for minimum standards met by student based on probability it is either been accepted or rejected. If it has been accepted, the application is forwarded to Faculty advisor.

If the faculty finds the candidate suitable for the program and has a good GPA suitable for the position he will accept it. The GPA too is calculated completely on probability and once its rejected its marked as advisor rejected applicationID.

Student registration system keeps track of students, courses they have opted for and the status. Upon successful submission, a student ID is automatically assigned to the student for future reference purposes. Finally, within the registration period the Advisor do advising for the students by approving/rejecting requested course by the student.

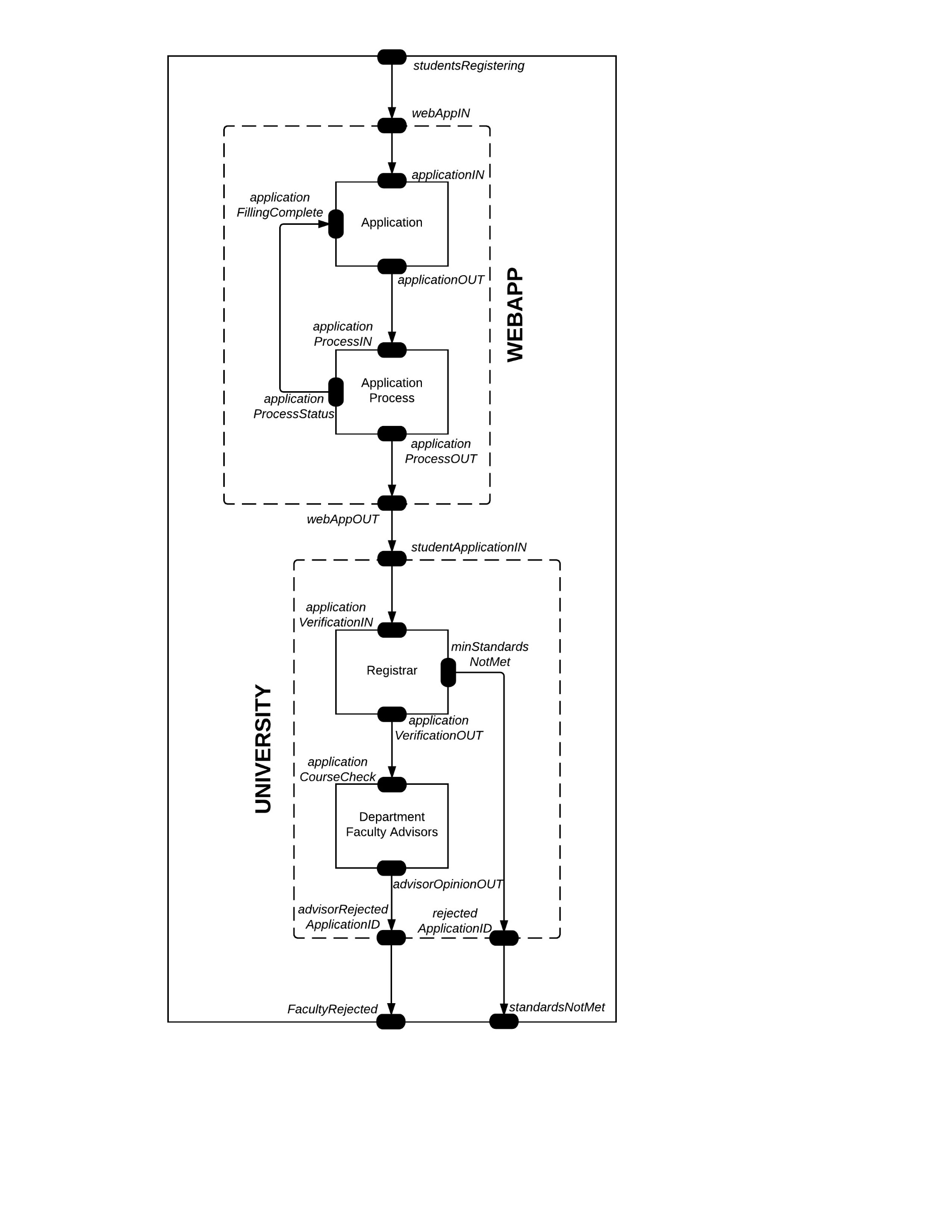


Figure 1: Student Academic Registration System (SARS)

**Model Structure:**

* WEBAPP model

This model is used to make all the students applications queue up, be processed in order and distribute those applications who want to apply for the designated university through university online process.

* UNIVERSITY model

This model is aimed to make applications go through a series of check process (Administrations registrar, department faculty / advisor check) before accepting the application. As the result, not so good student applications are selected out.

**Components**:

* application
* application process
* registrar
* faculty

**Behavior of component**

application: This atomic model is used to make all the applications holding for filling procedures in a queue. Once it receives the feedback from the application process model considering submitting documents and submitting transcripts, it will let the next applications to be filled.

applicationProcedure: It distributes the new applications to the university. If he successfully filled the applications and chosen to submit the application.

registrar: It could check whether the student met the minimum requirements of the university or if the applications are full and no more space is available of new students.

faculty: It could check if the students course options is suitable for his/her goals and want to give the admission or not.

# Part II

As you can see in figure 1, the SARS – Student Academic Registering System simulator has 1 input and 2 outputs. The *studentsregistering* means how many applications including the ones who have submitted the applications online and the ones that are still pending are go to university. The two outputs are showing the unsatisfied applications who fails in one of the series of checks, that is, registrars *minStandards*, advisors *advisorOpinion*. In terms of general overview,

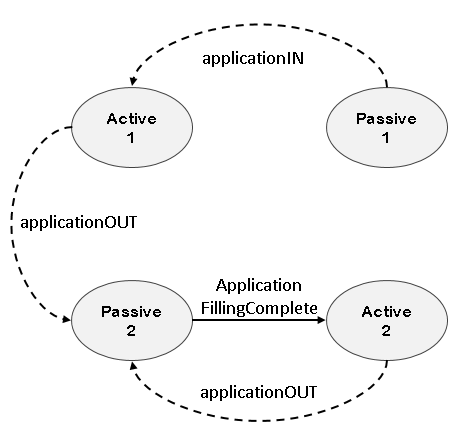
The application submission is made of two components: online submission and university verification. In addition, WEBAPP and UNIVERSITY models are decompressed to 2 components each for better understanding. WEBAPP has application and *applicationProcess* unit or atomic models connected in a series with a feedback links. Where as for the couple model UNIVERSITY is split in o two main atomic models, administrations block or most commonly known as *registrar* and the departments faculty who act as an advisors named as *advisor*, these are connected in series and each as their own output. These models are taken based on assumption that they have the similar behavior which can check if the application fit the standards of the university.

**Formal Specifications of atomic models**

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

DEVS = ( X, Y, S, δext, δint, λ, ta )

**Application**:



X = {applicationIN, applicationFillingComplete}

Y = {1, studentNumber}

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

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

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

δint (“active1”, 1) = “passive1”

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

λ(“active1”, applicationTime) = 1

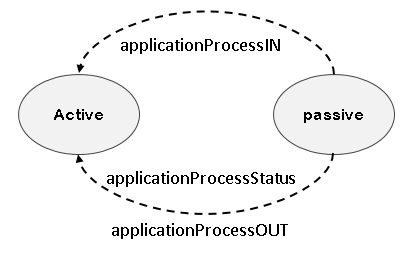
λ(“active2”, applicationTime) = applicationNum

ta(“active1”, applicationTime) = 5

ta(“active2”, applicationTime) = 5

ta(“passive”) = ∞

**applicationProcess**



X={applicationProcessIn}

Y= { applicationProcessOUT, applicationProcessStatus}

S= {“passive”, “active”}

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

δint (“active”) = “passive”

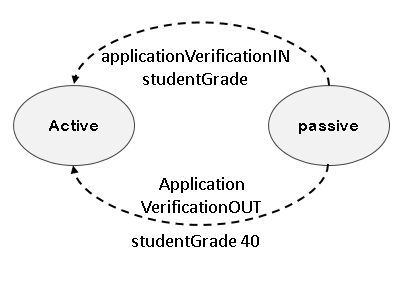
λ(“active”, applicationProcessTime)

= applicationNum if outport = applicationProcessStatus

= applicationProcessno if outport = applicationProcessOUT

ta(“active”, applicationProcessTime) = 5

**registrar**

****

S = {“passive”, “active”}

X = {applicationVerificationIN}

`Y = {applicationVerificationOUT, minStandardsNotMet}

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

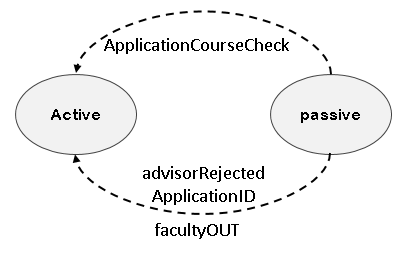
δint (“active”) = “passive”

λ(“active”, registrarTime) = studntGrade (>=40) if outport = applicationVerificationOUT

= studentGrade (<40) if outport = minStandardNotMet

ta(“active”, registrarTime) = 5

**Faculty**

****

S = {“passive”, “active”}

X = {applicationCourseCheck}

`Y = {facultyOut, advisorOpinionOUT}

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

δint (“active”) = “passive”

λ(“active”, facultyTime) = checkApplicationNum (satisfies) if outport = facultyOut

= checkApplicationNum (not) if outport = advisorOpinionOUT

ta(“active”, facultyTime) = 10

**Formal specifications of coupled models**

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

* **webApp**

X = {webAppIN }

Y = {webAppOUT}

D = {application, applicationProcess}

EIC = {(( application, “webAppIN”), (application, “applicationIN”)}

EOC = {( applicationProcess, “applicationProcessOUT”) , (application, “applicationOUT”)}

IC = {(( application, “applicationOUT”),( applicationProcess, “applicationProcessIN”)),

((applicationProcess,“applicationProcessStatus”),(application, “applicationFillingComplete” ))}

* **university**

X = {studentApplicationIN }

Y = {rejectedApplicationID, advisorRejectedApplicationID}

D = {registrar, faculty}

EIC = {(university, “studentApplicationIN”), (registrar, “applicationVerificationIN”)}

EOC = {((registrar, “minStandardsNotMet”), (university, “rejectedapplicationID”)),

((faculty, “advisorOpinionOUT”),(university, “advisorRejectedApplicationID”))}

IC = {(( registrar, “minStandardsNotMet”), (faculty, “applicationCourseCheck”))}

**Test Strategies**

The atomic models and coupled models will be tested using the “black box” testing method. Test cases are created by adding different combinations of inputs to the event file (.ev), run the simulation (.scp) and check whether the outputs in the output file (.out) are what we expected.

# Part III

To verify the atomic models, test cases are created to test these models. Please refer the project models for more in-depth analysis on the coding, I’ve stated comments and explanations.

* **Atomic model “application”**

The input of application should be a positive integer represents the number of applications who will get on the university admission. 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 application. After sending, it should wait for the feedback at the port “applicationFillingComplete”. Once it receives the feedback and then it could send the next application (2, 3, 4 …)

So I write the application.ev file as follows:

00:00:10:00 applicationIN 11

00:00:16:00 applicationFillingComplete 1

00:00:22:00 applicationFillingComplete 2

**00:00:28:00 applicationFillingComplete 3**

00:00:32:00 applicationFillingComplete 4

00:00:40:00 applicationFillingComplete 5

**00:00:42:00 applicationFillingComplete 6**

00:00:52:00 applicationFillingComplete 7

00:00:58:00 applicationFillingComplete 8

00:01:04:00 applicationFillingComplete 9

00:01:10:00 applicationFillingComplete 10

00:01:16:00 applicationFillingComplete 11

The 2 bold events (line 3 and 8) should not generate outputs because the “applicationTime” is 5 which means it needs 5 seconds to execute the application time. 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 application.out file shows the expected result as follows:

00:00:15:000 applicationOUT 1

00:00:21:000 applicationOUT 2

00:00:27:000 applicationOUT 3

00:00:37:000 applicationOUT 5

00:00:47:000 applicationOUT 7

00:00:57:000 applicationOUT 8

00:01:03:000 applicationOUT 9

00:01:09:000 applicationOUT 10

00:01:15:000 applicationOUT 11

* **Atomic model “applicationProcess”**

The input of the applicationProcess 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 “applicationProcessStatus” and the port “applicationProcessOUT”, otherwise it will send from the port “applicationProcessStatus” only. So the test file “applicationProcess.ev” is created as follows:

00:00:10:00 applicationProcessIN 1

**00:00:16:00 applicationProcessIN 2**

**00:00:19:00 applicationProcessIN 6**

00:00:28:00 applicationProcessIN 7

00:00:34:00 applicationProcessIN 8

00:00:35:00 applicationProcessIN 16

00:01:28:00 applicationProcessIN 27

00:02:34:00 applicationProcessIN 38

**00:02:36:00 applicationProcessIN 39 \***

00:04:34:00 applicationProcessIN 43

00:05:34:00 applicationProcessIN 53

00:06:20:00 applicationProcessIN 65

00:07:34:00 applicationProcessIN 77

00:08:00:00 applicationProcessIN 88

The 3 bold events (line 3, 6 and 9) should not generate outputs because the “applicationProcessTime” is 5 which means it needs 5 seconds to execute the application. If at that time, a new input arrived in advance, application/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 “application/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 applicationProcess.out file shows the expected result as follows:

00:00:15:000 applicationProcessOUT 1

00:00:15:000 applicationProcessStatus 1

00:00:21:000 applicationProcessStatus 2

00:00:33:000 applicationProcessOUT 7

00:00:33:000 applicationProcessStatus 7

00:00:39:000 applicationProcessStatus 8

00:01:33:000 applicationProcessOUT 27

00:01:33:000 applicationProcessStatus 27

00:02:39:000 applicationProcessStatus 38

00:04:39:000 applicationProcessOUT 43

00:04:39:000 applicationProcessStatus 43

00:05:39:000 applicationProcessOUT 53

00:05:39:000 applicationProcessStatus 53

00:06:25:000 applicationProcessOUT 65

00:06:25:000 applicationProcessStatus 65

**00:07:39:000 applicationProcessOUT 77**

00:07:39:000 applicationProcessStatus 77

00:08:05:000 applicationProcessStatus 88

* **Atomic model “Registrar”**

The input of this model is integers indicates the student applications. Then it handles the data it received, making it BIT AND with 30 or 40. If the result is larger than 25, it will send from the port “minStandardsNotMet”, otherwise it will be sent from the port “applicationVerificationOUT” only. So the test file “Registrar.ev” is created as follows:

00:00:00:00 applicationVerificationIN 22

00:00:20:00 applicationVerificationIN 23

00:00:25:00 applicationVerificationIN 24

00:01:10:00 applicationVerificationIN 26

**00:02:15:00 applicationVerificationIN 56**

00:02:20:00 applicationVerificationIN 122

00:02:30:00 applicationVerificationIN 125

00:02:40:00 applicationVerificationIN 167

00:02:50:00 applicationVerificationIN 178

**00:03:05:00 applicationVerificationIN 179**

00:04:20:00 applicationVerificationIN 223

00:04:22:00 applicationVerificationIN 225

00:04:50:00 applicationVerificationIN 245

00:05:10:00 applicationVerificationIN 256

00:05:30:00 applicationVerificationIN 278

The output should generate different outputs depending on different integers. It is similar to the “applicationProcess” 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 registrar.out file shows the expected result as follows:

00:00:09:000 applicationVerificationOUT 22

**00:00:29:000 applicationVerificationOUT 23**

00:01:19:000 minstandardsNotMet 26

00:02:19:000 minstandardsNotMet 56

00:02:29:000 minstandardsNotMet 122

00:02:39:000 minstandardsNotMet 125

00:02:49:000 applicationVerificationOUT 167

00:02:59:000 applicationVerificationOUT 178

00:03:09:000 applicationVerificationOUT 179

**00:04:29:000 minstandardsNotMet 223**

00:04:59:000 applicationVerificationOUT 245

00:05:19:000 applicationVerificationOUT 256

00:05:39:000 applicationVerificationOUT 278

* **Atomic model “registrar”**

This model has the similar function to the “registrar”. The input of this model is integers indicates the student applications. Then it handles the data it received, making it BIT AND with 63. If the result is 49 which indicates the application does not satisfy the standard, but not all the unsatisfied applications are sent from the “advisorRejectedApplicationID” 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 “advisorOpinionOUT”.

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

00:00:00:00 applicationCourseCheck 46

00:03:00:00 applicationCourseCheck 47

00:04:00:00 applicationCourseCheck 48

00:05:00:00 applicationCourseCheck 49

**00:06:00:00 applicationCourseCheck 51**

00:07:00:00 applicationCourseCheck 53

00:08:00:00 applicationCourseCheck 113

00:09:00:00 applicationCourseCheck 177

00:10:00:00 applicationCourseCheck 241

The result of BIT AND algorithm should decide which port the application should be sent from, so most of them (49, 113, 177, 241, 305, 369) should be sent from the port “advisorOpinionOUT”, while there is 20% possibility that they could be sent from the port “faultyOUT”. The output is not deterministic due to the random function. The following is an example of the output file faculty.out. In this output file, the event with bold fonts means this application escapes the rigorous background checks if you have an amazing recommendation from some famous and well known folks.

00:00:10:000 facultyOUT 46

00:03:10:000 facultyOUT 47

**00:04:10:000 facultyOUT 48**

00:05:10:000 advisorRejectedApplicationID 49

00:06:10:000 advisorRejectedApplicationID 51

00:07:10:000 advisorRejectedApplicationID 53

00:08:10:000 advisorRejectedApplicationID 113

00:09:10:000 advisorRejectedApplicationID 177

**00:10:10:000 facultyOUT 241**

* **Coupled Model “webApp”**

The coupled model webApp is the second top model which consists of “application” and “applicationProcess” atomic models. All applications go into the webApp model, and then go into the “application” which calculates how many universitys will go boarding, after which they enter the “applicationprocess” notifies where student applications should go next ( the other coupled model “university”). At the same time, once the “applicationProcess” has handled one application, it will notify the application that it is empty now and next application could come on.

The input of this coupled model is “webappIN”, a positive integer indicating the number of applications who want to get on the university admission. The output is “webappOut” which indicates the number of applications who want to get admission and have successfully submitted the application with 70% probability.

The test file “webApp.ev” is simple with only one line as follows. 15 applications want to get the admission.

00:00:00:00 webAppIN 15

The output should be a series of integer numbers which indicate student applications. As I simulate at the time 00:00:00:00, going through the appplicationTime and applicationProcessTime for each application. the time of output should be at 00:00:10:00 for the first application. Sequentially, the second application ( the integer 2 ) implemented the probability of 70 % and then the result is 0/1 which means he has submitted the application or not, so this is not sent from the port “webAppOUT”. In this case he accepted, That is the reason why the second application is sent out at the 00:00:20:000 (after handling 2 applications). The following is the output file webapp.out.

00:00:10:000 webappout 1

00:00:20:000 webappout 2

00:00:30:000 webappout 3

00:00:40:000 webappout 4

00:00:50:000 webappout 5

00:01:00:000 webappout 6

00:01:10:000 webappout 7

00:01:20:000 webappout 8

00:01:30:000 webappout 9

00:01:40:000 webappout 10

00:01:50:000 webappout 11

00:02:00:000 webappout 12

00:02:10:000 webappout 13

00:02:20:000 webappout 14

00:02:30:000 webappout 15

* **Coupled Model “university”**

The coupled model university is also the second top model which consists of “registrar” and “Faculty” atomic models. All student applications enter into the university model, and then go into the “registrar” which calculates whether the applications should be accepted or not, after which they enter the “faculty” atomic model to receive the corresponding final application check. In any of three check programs, the application does not obey the rule of university standards and sheer luck; the application will probably fail to secure a seat in that university. But some of them has the certain probability to escape such things, if they are lucky enough.

The input of this coupled model is “studentapplicationIN”, a positive integer indicating the number of student applications who want to get an admission. The outputs are “rejected applicationID” and “advisorRejectedApplicationID” which indicate the number of applications who do not satisfy the conditions.

The test file “university.ev” is not simple as the “webapp” coupled model as follows.

**00:10:00:00 studentApplicationIN 15**

00:20:00:00 studentApplicationIN 29

00:30:00:00 studentApplicationIN 31

00:40:00:00 studentApplicationIN 47

**00:50:00:00 studentApplicationIN 49**

00:55:00:00 studentApplicationIN 109

01:10:00:00 studentApplicationIN 111

01:30:00:00 studentApplicationIN 113

01:50:00:00 studentApplicationIN 127

01:55:00:00 studentApplicationIN 177

02:10:00:00 studentApplicationIN 341

03:40:00:00 studentApplicationIN 343

04:40:00:00 studentApplicationIN 135

05:50:00:00 studentApplicationIN 327

06:00:00:00 studentApplicationIN 233

The output should be some unsatisfied applications who fail in any of these check processes, but it is not fixed due to the random functions. Per the result, I simulate at the time 00:10:00:00. The No.15 application is not accepted successfully as its random number is larger than 25. The No.47 application is checked successfully during the first stage check, taking 9 seconds (registrarTime) and the time of output should be at 00:20:09:000. As for the No.47 university, he experiences 2 steps of check and is picked up because he does not obey the faculty rule, taking 9 seconds facultyTime. The following is an example of output files university.out.

00:20:09:000 rejectedapplicationid 29

00:30:09:000 rejectedapplicationid 31

00:50:19:000 advisorrejectedapplicationid 49

01:50:09:000 rejectedapplicationid 127

01:55:19:000 advisorrejectedapplicationid 177

* **The top model “sars”**

The top model consists of 2 coupled models “webApp” and “university”. The input of the top model should be an integer (larger than 0) which indicates the total number of applications. The output should be student applications who fail in any of two check process. In order to get a larger samples to analyze, I set the “studentsRegistering” equal to 500. So the file sars.ev is created as follows:

00:00:00:00 studentsRegistering 500

After going through the “webapp” model, total applications are divided into 2 parts- submitted and not submitted applications. student Applications are represented by the numbers. Then they receive a series of checks implemented by the algorithm BIT AND, and then some Applications will fail in one of two 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 Applications who do not submit to the university, they need (5(applicationTime) + 5(applicatitionProcessTime))×24 = 240s. For the No.25 Application, the required time is 5(applicationTime) + 5(applicatitionProcessTime)+ 9(registrarTime)=29s. So the first output “standardsnotmet 26” should be at the time 00:04:29:000 (240s+29s=269s). The following is an example of output files sars.out.

00:04:29:000 standardsnotmet 26

00:04:39:000 standardsnotmet 27

00:04:49:000 standardsnotmet 28

00:04:59:000 standardsnotmet 29

00:05:09:000 standardsnotmet 30

00:05:19:000 standardsnotmet 31

00:09:49:000 standardsnotmet 58

00:09:59:000 standardsnotmet 59

00:10:09:000 standardsnotmet 60

00:10:19:000 standardsnotmet 61

00:10:29:000 standardsnotmet 62

00:10:39:000 standardsnotmet 63

00:15:09:000 standardsnotmet 90

00:15:19:000 standardsnotmet 91

00:15:29:000 standardsnotmet 92

00:15:39:000 standardsnotmet 93

00:15:49:000 standardsnotmet 94

00:15:59:000 standardsnotmet 95

00:20:29:000 standardsnotmet 122

00:20:39:000 standardsnotmet 123

00:20:49:000 standardsnotmet 124

00:20:59:000 standardsnotmet 125

00:21:09:000 standardsnotmet 126

00:21:19:000 standardsnotmet 127

00:25:49:000 standardsnotmet 154

00:25:59:000 standardsnotmet 155

00:26:09:000 standardsnotmet 156

00:26:19:000 standardsnotmet 157

00:26:29:000 standardsnotmet 158

00:26:39:000 standardsnotmet 159

00:31:09:000 standardsnotmet 186

00:31:19:000 standardsnotmet 187

00:31:29:000 standardsnotmet 188

00:31:39:000 standardsnotmet 189

00:31:49:000 standardsnotmet 190

00:31:59:000 standardsnotmet 191

00:36:29:000 standardsnotmet 218

00:36:39:000 standardsnotmet 219

00:36:49:000 standardsnotmet 220

00:36:59:000 standardsnotmet 221

00:37:09:000 standardsnotmet 222

00:37:19:000 standardsnotmet 223

00:41:49:000 standardsnotmet 250

00:41:59:000 standardsnotmet 251

00:42:09:000 standardsnotmet 252

00:42:19:000 standardsnotmet 253

00:42:29:000 standardsnotmet 254

00:42:39:000 standardsnotmet 255

00:47:09:000 standardsnotmet 282

00:47:19:000 standardsnotmet 283

00:47:29:000 standardsnotmet 284

00:47:39:000 standardsnotmet 285

00:47:49:000 standardsnotmet 286

00:47:59:000 standardsnotmet 287

00:52:29:000 standardsnotmet 314

00:52:39:000 standardsnotmet 315

00:52:49:000 standardsnotmet 316

00:52:59:000 standardsnotmet 317

00:53:09:000 standardsnotmet 318

00:53:19:000 standardsnotmet 319

00:57:49:000 standardsnotmet 346

00:57:59:000 standardsnotmet 347

00:58:09:000 standardsnotmet 348

00:58:19:000 standardsnotmet 349

00:58:29:000 standardsnotmet 350

00:58:39:000 standardsnotmet 351

01:03:09:000 standardsnotmet 378

01:03:19:000 standardsnotmet 379

01:03:29:000 standardsnotmet 380

01:03:39:000 standardsnotmet 381

01:03:49:000 standardsnotmet 382

01:03:59:000 standardsnotmet 383

01:08:29:000 standardsnotmet 410

01:08:39:000 standardsnotmet 411

01:08:49:000 standardsnotmet 412

01:08:59:000 standardsnotmet 413

01:09:09:000 standardsnotmet 414

01:09:19:000 standardsnotmet 415

01:13:49:000 standardsnotmet 442

01:13:59:000 standardsnotmet 443

01:14:09:000 standardsnotmet 444

01:14:19:000 standardsnotmet 445

01:14:29:000 standardsnotmet 446

01:14:39:000 standardsnotmet 447

01:19:09:000 standardsnotmet 474

01:19:19:000 standardsnotmet 475

01:19:29:000 standardsnotmet 476

01:19:39:000 standardsnotmet 477

01:19:49:000 standardsnotmet 478

01:19:59:000 standardsnotmet 479

# Conclusion

The SARS model simulates the process of student applications. 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 registrar and faculty advisors of applications can be obtained by applicationID number (in binary) in my model. The execution results and the behavior of the model match the specifications of models. The SARS model works exactly as I expect according to the specifications. ☺

\* \* THE END \* \*