



East West University

Course Title : Software Quality Assurance
Course Code : CSE 435
Section : 01
Semester : Fall 2020

Project topic: Online Ambulance Service

Submitted By

Nasif Wasek Fahim

2017-1-60-037

Mueem Nahid Ibn Mahbub

2017-1-60-089

Md. Jubayer Hossain Abir

2017-1-60-084

Submission Date

07/01/2021

Online Ambulance Service

Introduction:

This is an integrated service that provides all information about the location of the ambulance and its routes for the public. The proposed system is a web-based application that provides information regarding ambulance quality, routes, fair. The system contains an online order system where service recipients can order any nearby ambulance service and service providers can receive the given order. There is also an admin module where the admin can add ambulance, scrap ambulance, and any update. The admin is a panel that will be consisted of a group of authorized persons.

1st Sequence Diagram:

Login System Sequence Diagram:

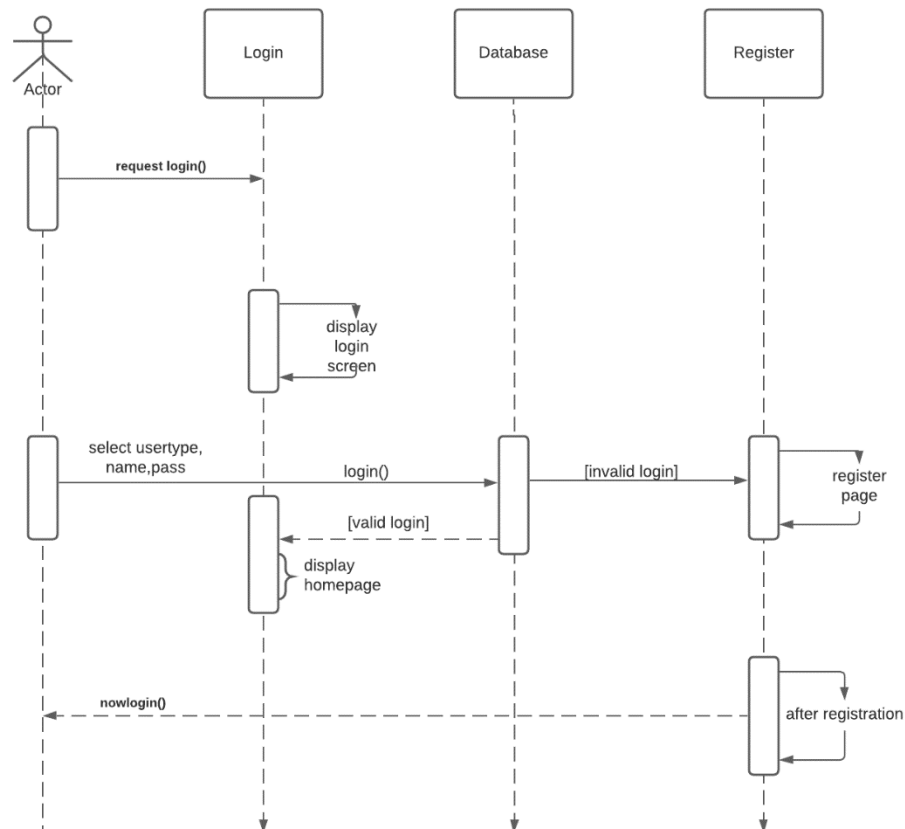


Fig: Login System

The functionality of the Login Sequence Diagram:

In this system actor (admin, service receiver, service provider) would try to log in to the system and for that, the actor would enter user type, username, and password. After this, the information would be verified and checked into the database. If the information is legit then the actor would be able to enter the homepage. Otherwise, the system would take the user to the registration page for registration.

PROMELA Code for the Login System:

```
mtype = { req2log, dis_log_pg};
```

```
chan toU = [2] of {mtype, bit};
```

```
chan toL = [2] of {mtype, bit};
```

```
proctype User(chan in, out)
```

```
{
```

```
    bit sndbit, recbit;
```

```
    do
```

```
        :: out !req2log, sndbit ->
```

```
            in ?dis_log_pg, recbit;
```

```
    od
```

```
}
```

```
proctype Login(chan in, out)
```

```
{
```

```
    bit recbit;
```

```
    do
```

```
        :: in ? req2log(recbit) ->
```

```
            out ! dis_log_pg(recbit);
```

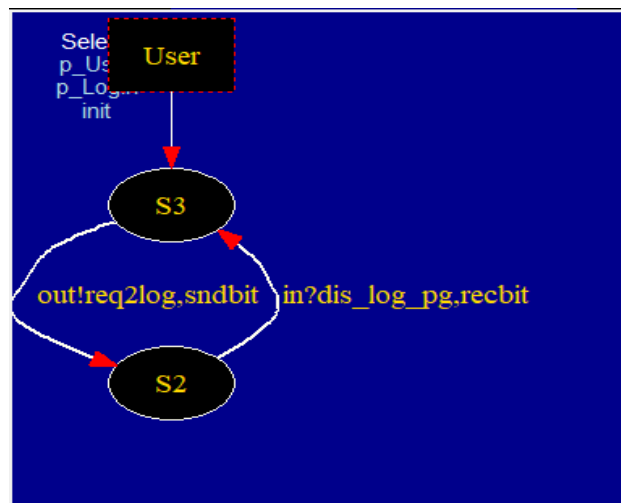
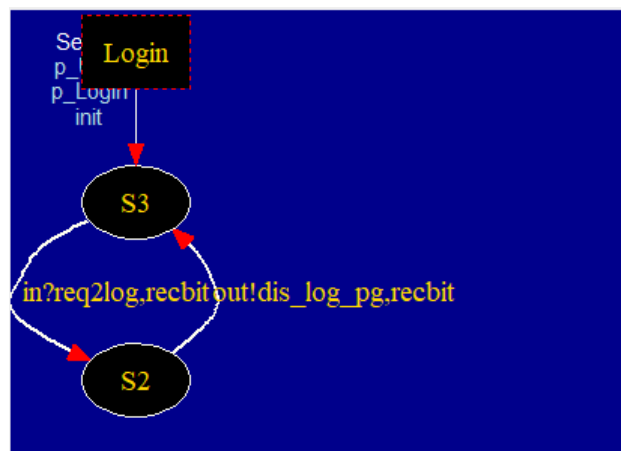
```

    od
}

init
{
    run User (toU, toL);
    run Login (toL, toU);
}

```

Screenshots of the Process Automata:



log-in.pml

Spin Version 6.5.1 -- 20 December 2019 -- iSpin Version 1.1.4 -- 27 November 2014

Edit/View
Simulate / Replay
Verification
Swarm Run
<Help>
Save Session
Restore Session
<Quit>

Mode
A Full Channel
Output Filtering (reg. exp.s)
(Re)Run

☒ Random, with seed: 123
☒ blocks new messages
process ids:

☐ Interactive (for resolution of all nondeterminism)
☐ loses new messages
queue ids:

☐ Guided, with trail: log-in.pml.trail
browse
☐ MSC+stmt
var names:

initial steps skipped: 0
MSC max text width: 20
tracked variable:

maximum number of steps: 10000
MSC update delay: 25
track scalings:

☒ Track Data Values (this can slow)

Background command executed:
spin -p -s -r -X -v -n123 -l -g -u10000 log-in.pml

Step Forward
Step Backward

Save in: msc.ps

```

1      mtype = { req2log, dis_log_pg;
2          chan toU = [2] of {mtype, bit};
3          chan toL = [2] of {mtype, bit};
4
5          proctype User(chan in, out)
6          {
7              bit sndbit, recbit;
8              do
9                  :: out !req2log, sndbit ->
10                     in ?dis_log_pg, recbit;
11
12             od

```

```

99      2?req2log.0
100      2?req2log.0
101      2?dis_log_pg.0
102      2?req2log.0
103      2?req2log.0
104      2?dis_log_pg.0
105      2?req2log.0
106      2?dis_log_pg.0
107      2?req2log.0
108      2?dis_log_pg.0
109      2?req2log.0
110      2?dis_log_pg.0
111      2?req2log.0
112      2?req2log.0

```

[variable values, step 112]

```

94:   proc 2 (Login:1) log-in.pml:19 (state 1) [in?req2log,recbit]
95:   proc 2 (Login:1) log-in.pml:20 (state 2) [out?dis_log_pg,recbit]
96:   proc 1 (User:1) log-in.pml:10 (state 2) [in?dis_log_pg,recbit]
97:   proc 1 (User:1) log-in.pml:9 (state 1) [out?req2log,sndbit]
98:   proc 2 (Login:1) log-in.pml:19 (state 1) [in?req2log,recbit]
99:   proc 2 (Login:1) log-in.pml:20 (state 2) [out?dis_log_pg,recbit]
100:   proc 1 (User:1) log-in.pml:10 (state 2) [in?dis_log_pg,recbit]
101:   proc 1 (User:1) log-in.pml:9 (state 1) [out?req2log,sndbit]
102:   proc 2 (Login:1) log-in.pml:19 (state 1) [in?req2log,recbit]
103:   proc 2 (Login:1) log-in.pml:20 (state 2) [out?dis_log_pg,recbit]
104:   proc 1 (User:1) log-in.pml:10 (state 2) [in?dis_log_pg,recbit]
105:   proc 1 (User:1) log-in.pml:9 (state 1) [out?req2log,sndbit]
106:   proc 2 (Login:1) log-in.pml:19 (state 1) [in?req2log,recbit]
107:   proc 2 (Login:1) log-in.pml:20 (state 2) [out?dis_log_pg,recbit]
108:   proc 1 (User:1) log-in.pml:10 (state 2) [in?dis_log_pg,recbit]
109:   proc 1 (User:1) log-in.pml:9 (state 1) [out?req2log,sndbit]
110:   proc 2 (Login:1) log-in.pml:19 (state 1) [in?req2log,recbit]
111:   proc 2 (Login:1) log-in.pml:20 (state 2) [out?dis_log_pg,recbit]
112:

```

[queues, step 112]

```

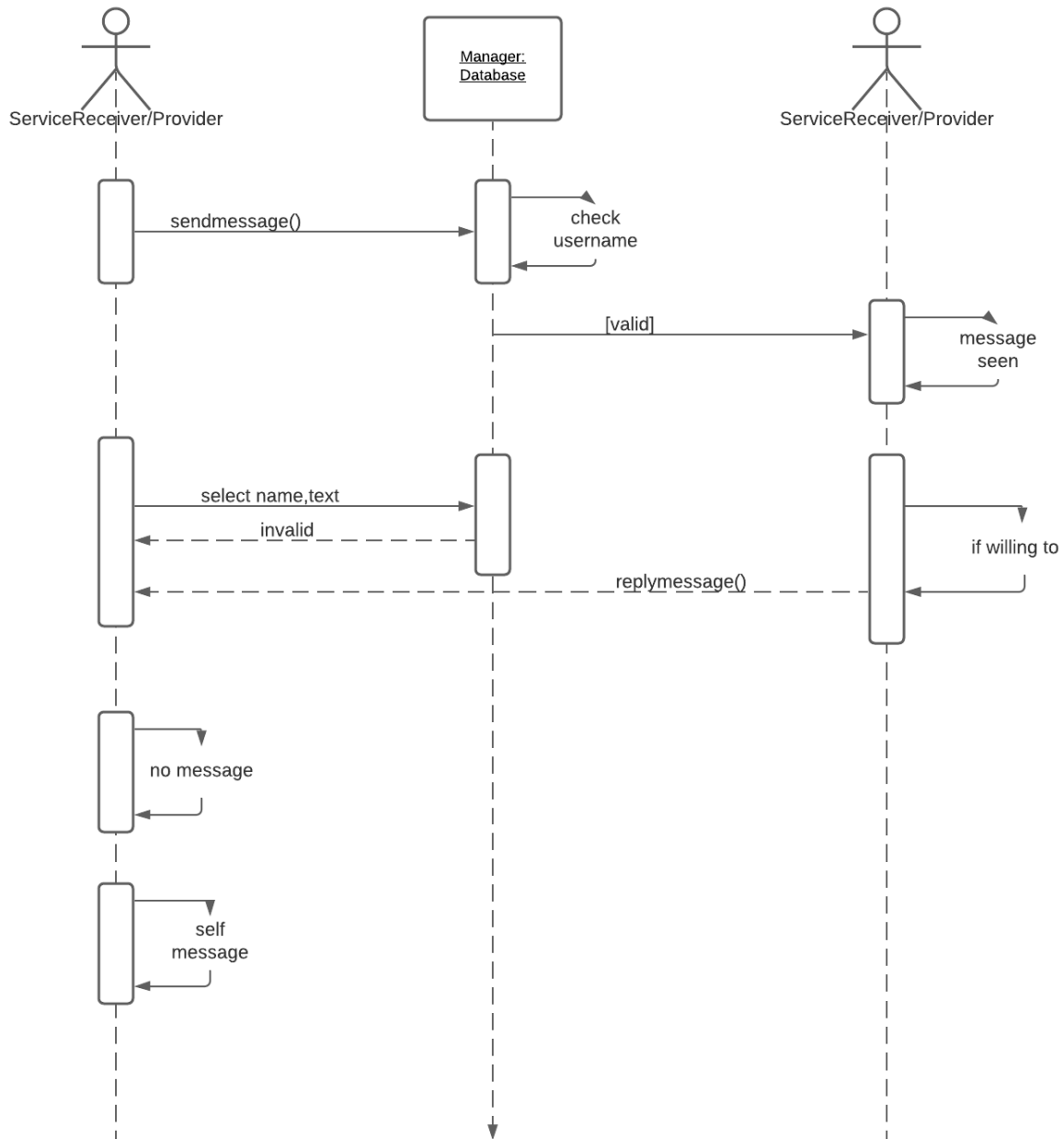
q 1 :: (Login(2):out):
q 2 :: (Login(2):in):

```

[illegible]

2nd Sequence Diagram:

Inbox System Sequence Diagram:



Sequence diagram of
Inbox System

Fig: Inbox System

The functionality of the Inbox Sequence Diagram:

In this system, the service receiver can send messages to the service provider in case of any need and vice versa. The system also allows service receivers to send message to other service receivers and the other way around. There is also a feature where the user can message oneself. First, a user selects a username to text and sends the text. After the message is sent by the user, the user will have to wait for a response. After the message is seen by the recipient then the recipient can reply, and the sender would get the message.

PROMELA Code for the Inbox System:

```
mtype {MSG, ACK};
```

```
chan toS = [2] of {mtype, bit};
```

```
chan toR = [2] of {mtype, bit};
```

```
proctype Sender(chan inp, out)
```

```
{
```

```
    bit sendbit, rcvbit;
```

```
    do
```

```
        :: out ! MSG, sendbit ->
```

```
            inp ? ACK, rcvbit;
```

```
            if
```

```
                :: rcvbit == sendbit ->
```

```
                    sendbit = 1-sendbit
```

```
                :: else
```

```
            fi
```

```
    od
```

```
}
```

```
proctype Receiver(chan inp, out)
```

```

{
    bit recvbit;
    do
        :: inp ? MSG(recvbit) ->
            out ! ACK(recvbit);
    od
}

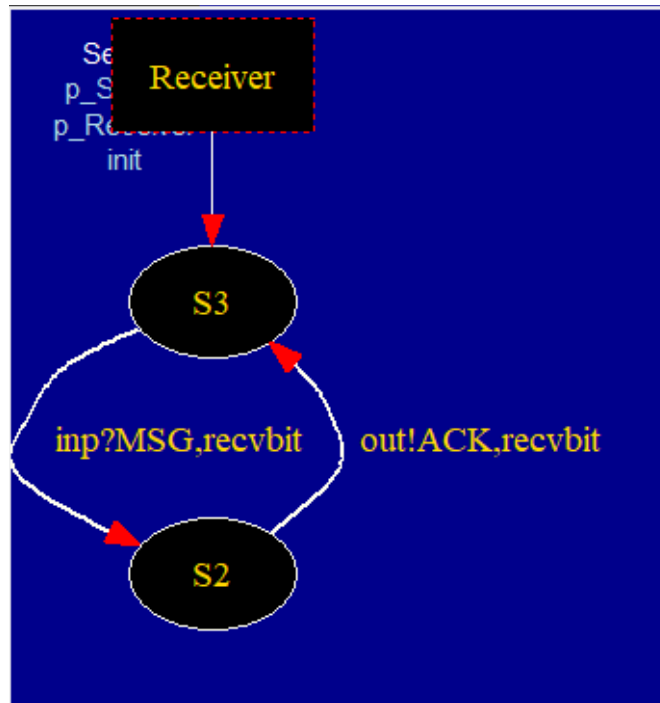
```

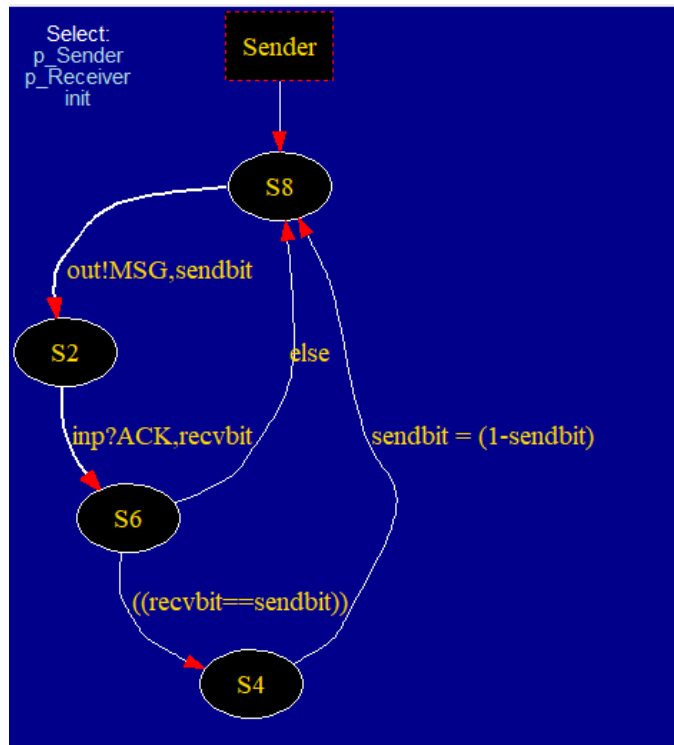
```

init
{
    run Sender(toS, toR);
    run Receiver(toR, toS);
}

```

Screenshots of the Process Automata:





Screenshot of the inbox.pml Simulation:

inbox.pml

Spin Version 6.5.1 -- 20 December 2019 -- iSpin Version 1.1.4 -- 27 November 2014

Mode: Random, with seed: 123. A Full Channel. Output Filtering (reg. exps.): (Re)Run. Background command executed: spin -p -s -r -X -v -n123 -l -g -u10000 inbox.pml

Options: ☒ Random, with seed: 123. ☐ Interactive (for resolution of all nondeterminism). ☐ Guided, with trail: inbox.pml.trail. Initial steps skipped: 0. Maximum number of steps: 10000. Track Data Values (this can be slow): ☒. ☐ blocks new messages. ☐ loses new messages. ☐ MSC+stmnt. MSC max text width: 20. MSC update delay: 25. Tracked variable: . Track scaling: .

1 mtype (MSG, ACK);
2
3 chan toS = [2] of {mtype, bit};
4 chan toR = [2] of {mtype, bit};
5
6 proctype Sender(chan inp, out)
7 {
8 bit sendbit, rcvbit;
9 do
10 :: out ! MSG, sendbit ->
11 inp ? ACK, rcvbit;
12 if

986 1?ACK,1
987 2!MSG,0
993 2?MSG,0
994 1!ACK,0
995
996 1?ACK,0
996 2!MSG,1
1002 2?MSG,1
1003 1!ACK,1
1004
1005 1?ACK,1

[variable values, step 1007]
Receiver(2):rcvbit = 1
Sender(1):rcvbit = 1
Sender(1):sendbit = 0

990: proc 1 (Sender:1) inbox.pml:14 (state 4) [sendbit = (1-sendbit)]
993: proc 1 (Sender:1) inbox.pml:10 (state 1) [out!MSG,sendbit]
994: proc 2 (Receiver:1) inbox.pml:24 (state 1) [inp?MSG,rcvbit]
995: proc 2 (Receiver:1) inbox.pml:25 (state 2) [out!ACK,rcvbit]
996: proc 1 (Sender:1) inbox.pml:11 (state 2) [inp?ACK,rcvbit]
998: proc 1 (Sender:1) inbox.pml:13 (state 3) [((rcvbit==sendbit))]
999: proc 1 (Sender:1) inbox.pml:14 (state 4) [sendbit = (1-sendbit)]
1002: proc 1 (Sender:1) inbox.pml:10 (state 1) [out!MSG,sendbit]
1003: proc 2 (Receiver:1) inbox.pml:24 (state 1) [inp?MSG,rcvbit]
1004: proc 2 (Receiver:1) inbox.pml:25 (state 2) [out!ACK,rcvbit]
1005: proc 1 (Sender:1) inbox.pml:11 (state 2) [inp?ACK,rcvbit]
1006: proc 1 (Sender:1) inbox.pml:13 (state 3) [((rcvbit==sendbit))]
1007: proc 1 (Sender:1) inbox.pml:14 (state 4) [sendbit = (1-sendbit)]

[queues, step ~]
q 1 :: (Sender(1):inp):
q 2 :: (Sender(1):out):

Screenshot of the inbox.pml Diagram:

