SYSC 5104 – Methodologies for Discrete Event Modeling and Simulation

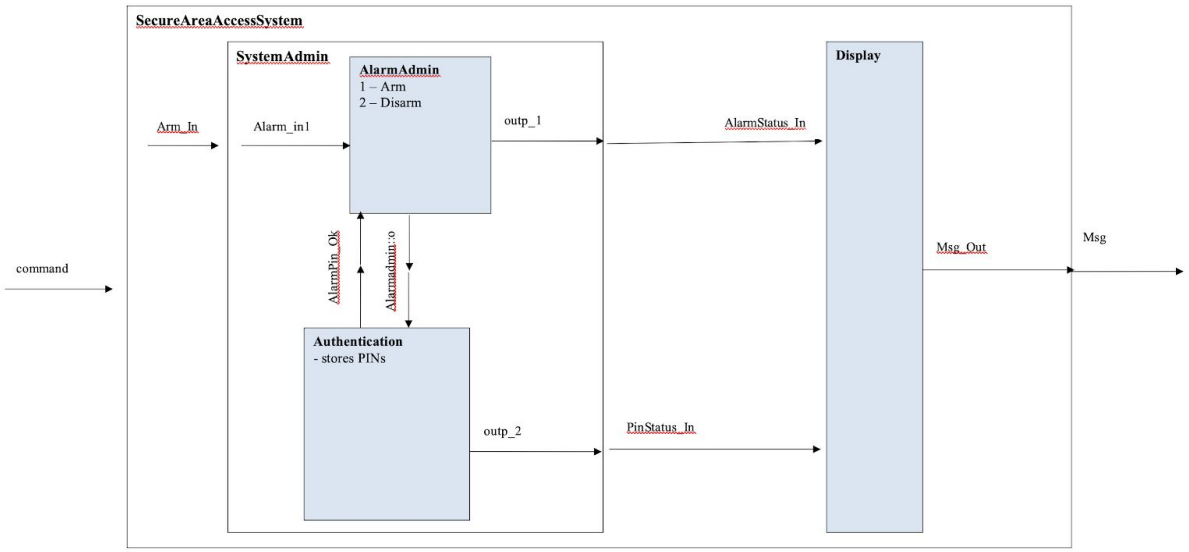
Secure Area Access System

Model Developed Initially : Tania Pendergast - 23 June, 2008

Name: Prathmesh Ranaut	**Name: KHURRAM SHAFIQ**
Carleton University	**University of Ottawa**
Student ID: 101138167	**Student ID: 300093289**
prathmeshranaut@cmail.carleton.ca	**kshaf094@uottawa.ca**

Model Description

The following is the complete structure for our secure area access system adapted from the CD++ implementation by Tania Pendergast. We will implement this model in Cadmium.



Model Components:

1. Top Model: It consists of receiving an Arm/Disarm command and processing to display the following messages:
    1. 0 => Alarm disarmed message
    2. 1 => Alarm armed message
    3. 2 => PIN prompt message
    4. 3 => Invalid action message
    5. 4 => Invalid PIN error message
    6. 5 => Door temporarily unlocked message

2. SystemAdmin: It's job is to receive the command from Top model and send output to Display model.
   Takes Arm/Disarm command.

3. AlarmAdmin: This model receives the command from SystemAdmin and Authentication model to send output to display model.
   Takes Arm/Disarm command and requests pin from Authentication and based on the output, sends output on outp_1 port.
4. Authenticator: This model authenticates the AlarmModel. This model receives input from AlarmAdmin and it sends back output to AlarmAdmin for further processing.

DEVS Formalism

AlarmAdmin

X = { ArmReq_In, DisarmReq_In, AlarmPin_Ok }
Y = { Admin_Out, Alarm_Status1, Alarm_Status2, Error, Alarm_Req }
S = { state, processingTime (0,0,10), enum Status status={disarmed, armed},
enum Request request={disarm, arm, invalid, none},
enum Errors err={invalid_action, invalid_pin, no} }
δext (s,e,x)
{

```
        switch (port) {
          case Arm:
            if (message == 1 && state.status == Status::Disarmed) {
              state.request = Arm;
              state.working = true;
            } else {
              assert("Invalid State sent to Arm - AlarmAdmin");
            }
            break;
          case Disarm:
            if (message == 1 && state.status == Status::Armed) {
              state.request = Disarm;
              state.working = true;
            } else {
              assert("Invalid State sent to Disarm - AlarmAdmin");
            }
            break;
          case Pin:
            if (message == 1 && state.status == Status::Disarmed) {
              state.request = None;
              state.status = Status::Armed;
              state.working = true;
            }
            if (message == 0 && state.status == Status::Armed) {
              state.request = None;
              state.status = Status::Disarmed;
              state.working = true;
            }
            if (message == 2) {
              //Invalid pin
              state.request = None;
              state.working = true;
```

```
            }
            break;
        }
}

δint (s)
{
        if (state.request == Arm || state.request == Disarm) {
          state.next_internal = preparationTime;
        } else {
          state.request = None;
          state.nextInternal = std::numeric_limits<TIME>::infinity();
        }
}

λ(s)
{
        if (state.request == Arm) {
          out_aux = Message_t(0, 1);
        } else if (state.request = Disarm) {
          out_aux = Message_t(1, 1);
        } else if (state.request == None) {
          out_aux = Message_t(2, 1);
        }
}
```

Authentication

X = {Pin_In, Alarm_Req, Alarm_Status2}
Y = {AlarmPin_Ok, Error, DoorPin_Ok}
S = {state, randnumber, dist, processingTime (0,0,0,10), enum PinCheck pinchk={disarm_valid,
arm_valid, invalid, door_valid, door_invalid, none}, enum Request request={disarm, arm, nada},
enum Errors err={invalid_action, invalid_pin, no}, enum Status status={disarmed, armed} }

```
δext (s,e,x)
{
        switch (port) {
          case 0:
            if (state.request == Request::None) {
              if (message == 0) {
                state.status = Disarmed;
```

```
        } else if (message == 1) {
            state.status = Armed;
        } else {
            assert(false && "Invalid message passed to Pin Model");
        }
        state.request = Request::None;
        state.pinCheck = PNone;
        state.nextInternal = std::numeric_limits<TIME>::infinity();
    } else {
        assert(false && "Invalid message/port reset with request None");
        state.nextInternal -= e;
    }
    break;
case 1:
    if (message == 1) {
        double randNumber = (double) rand() / (double) RAND_MAX;
        if (state.request == Arm) {
            if (randNumber <= 0.9) {
                state.pinCheck = ArmValid;
            } else {
                state.pinCheck = Invalid;
            }
        } else if (state.request == Request::Disarm) {
            if (randNumber <= 0.9) {
                state.pinCheck = DisarmValid;
            } else {
                state.pinCheck = Invalid;
            }
        } else {
            if (randNumber <= 0.9) {
                state.pinCheck = DoorValid;
            } else {
                state.pinCheck = DoorInvalid;
            }
        }
        state.nextInternal = preparationTime;
    }
    break;
case 2:
    if (message == 0) {
        state.request = Arm;
        state.nextInternal = std::numeric_limits<TIME>::infinity();
    }
    if (message == 1) {
        state.request = Disarm;
```

```
                state.nextInternal = std::numeric_limits<TIME>::infinity();
            }
            state.nextInternal -= e;
        }

}
δint (s)
{
        state.pinCheck = PNone;
        state.request = Request::None;
        state.nextInternal = std::numeric_limits<TIME>::infinity();
}
λ(s)
{
        switch (i.pinCheck) {
          case DisarmValid:
            out_aux = Message_t(0, 1);
            break;
          case ArmValid:
            out_aux = Message_t(0, 2);
            break;
          case Invalid:
            out_aux = Message_t(0, 3);
            break;
          case DoorValid:
            out_aux = Message_t(0, 4);
            break;
          case DoorInvalid:
            out_aux = Message_t(0, 5);
            break;
          case PNone:
            out_aux = Message_t(0, 6);
            break;
        }
}
```

Display

X = {PinPrompt_In, AlarmStatus_In, Error_In, DoorOk_In }
Y = {Msg_Out}
S = {state, message, processingTime (0,0,0,30), displayTime (0,0,10,0), enum Status={disarmed, armed}
enum Display={ disarmedMsg, armedMsg, pinPrompt, invalidAction, invalidPin, doorUnlock} }

```
δext (s,e,x)
{
        switch (port) {
          case 0:
            state.display = PINMsg;
            state.working = true;
            state.next_internal = TIME("00:00:10");
            break;
          case 1:
            if (message == 1) {
              state.display = ArmedMsg;
              state.status = Armed;
              state.working = true;
              state.next_internal = state.next_internal - e;
            } else if (message == 0) {
              state.display = DisarmedMsg;
              state.status = Disarmed;
              state.working = true;
              state.next_internal = state.next_internal - e;
            }
            break;
          case 2:
            if (message == 1) {
              state.display = InvalidPin;
              state.working = true;
              state.next_internal = state.next_internal - e;
            } else if (message == 0) {
              state.display = InvalidAction;
              state.working = true;
              state.next_internal = state.next_internal - e;
            }
            break;
          case 3:
            state.display = DoorUnlocked;
            state.working = true;
            state.next_internal = TIME("00:00:10");
            break;
        }

}

δint (s)
{
        if (state.display == ArmedMsg || state.display == DisarmedMsg || state.display ==
PINMsg) {
```

```
            // DO nothing
            state.next_internal = std::numeric_limits<TIME>::infinity();
            state.working = false;
        }

        if (state.working) {
            if (state.display == InvalidPin
                || state.display == InvalidAction
                || state.display == DoorUnlocked) {
                state.next_internal = preparationTime;
                state.working = false;
            }
        } else {
            if(state.status == Armed) {
                state.display = ArmedMsg;
                state.working = false;
                state.next_internal = std::numeric_limits<TIME>::infinity();
            } else if (state.status == Disarmed) {
                state.display = DisarmedMsg;
                state.working = false;
                state.next_internal = std::numeric_limits<TIME>::infinity();
            }
        }

}
λ(s)
{
        out_aux = Message_t(0, state.display);
}
```

Testing Strategy:
We will be using individual sets of inputs to test the atomic models and will then test the coupled and top model in black box.

Testing Models

Alarm Admin:

00:00:20 0 1 00:00:50 2 1 00:01:10 0 1 00:01:40 1 1 00:02:00 2 0 00:02:20 1 1 00:02:40 0 1 00:03:10 2 2 00:03:40 0 1 00:04:00 2 1	00:00:00:000 State for model input_reader is next time: 00:00:00:000 State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: None 00:00:00:000 State for model input_reader is next time: 00:00:20:000 State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: None 00:00:20:000 State for model input_reader is next time: 00:00:30:000 State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: Arm 00:00:30:000 State for model input_reader is next time: 00:00:30:000 State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: Arm 00:00:40:000 State for model input_reader is next time: 00:00:30:000 State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: Arm 00:00:50:000 State for model input_reader is next time: 00:00:20:000

	State for model alarmAdmin is AlarmAdmin Status:Armed; Request: None 00:01:00:000 State for model input_reader is next time: 00:00:20:000 State for model alarmAdmin is AlarmAdmin Status:Armed; Request: None 00:01:10:000 State for model input_reader is next time: 00:00:30:000 State for model alarmAdmin is AlarmAdmin Status:Armed; Request: None 00:01:40:000 State for model input_reader is next time: 00:00:20:000 State for model alarmAdmin is AlarmAdmin Status:Armed; Request: Disarm 00:01:50:000 State for model input_reader is next time: 00:00:20:000 State for model alarmAdmin is AlarmAdmin Status:Armed; Request: Disarm 00:02:00:000 State for model input_reader is next time: 00:00:20:000 State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: None 00:02:10:000 State for model input_reader is next time: 00:00:20:000 State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: None 00:02:20:000 State for model input_reader is next time: 00:00:20:000

	State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: None 00:02:40:000 State for model input_reader is next time: 00:00:30:000 State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: Arm 00:02:50:000 State for model input_reader is next time: 00:00:30:000 State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: Arm 00:03:00:000 State for model input_reader is next time: 00:00:30:000 State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: Arm 00:03:10:000 State for model input_reader is next time: 00:00:30:000 State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: None 00:03:20:000 State for model input_reader is next time: 00:00:30:000 State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: None 00:03:40:000 State for model input_reader is next time: 00:00:20:000 State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: Arm 00:03:50:000 State for model input_reader is next time: 00:00:20:000

	State for model alarmAdmin is AlarmAdmin Status:Disarmed; Request: Arm 00:04:00:000 State for model input_reader is next time: inf State for model alarmAdmin is AlarmAdmin Status:Armed; Request: None 00:04:10:000 State for model input_reader is next time: inf State for model alarmAdmin is AlarmAdmin Status:Armed; Request: None

Authentication:

00:00:10 0 0 00:00:30 2 0 00:00:45 1 1 00:00:59 0 1 00:01:30 2 1 00:01:50 1 1	00:00:00:000 State for model input_reader is next time: 00:00:00:000 State for model authentication is Authentication PinCheck: PNone; Status:Disarmed; Request: None 00:00:00:000 State for model input_reader is next time: 00:00:10:000 State for model authentication is Authentication PinCheck: PNone; Status:Disarmed; Request: None 00:00:10:000 State for model input_reader is next time: 00:00:20:000 State for model authentication is Authentication PinCheck: PNone; Status:Disarmed; Request: None 00:00:30:000 State for model input_reader is next time: 00:00:15:000

	State for model authentication is Authentication PinCheck: PNone; Status:Disarmed; Request: Arm 00:00:45:000 State for model input_reader is next time: 00:00:14:000 State for model authentication is Authentication PinCheck: ArmValid; Status:Disarmed; Request: Arm 00:00:55:000 State for model input_reader is next time: 00:00:14:000 State for model authentication is Authentication PinCheck: PNone; Status:Disarmed; Request: None 00:00:59:000 State for model input_reader is next time: 00:00:31:000 State for model authentication is Authentication PinCheck: PNone; Status:Armed; Request: None 00:01:30:000 State for model input_reader is next time: 00:00:20:000 State for model authentication is Authentication PinCheck: PNone; Status:Armed; Request: Disarm 00:01:50:000 State for model input_reader is next time: inf State for model authentication is Authentication PinCheck: DisarmValid; Status:Armed; Request: Disarm 00:02:00:000 State for model input_reader is next time: inf State for model authentication is Authentication PinCheck: PNone; Status:Armed; Request: None

Display:

00:00:10 0 1 00:00:30 1 1 00:01:00 0 1 00:01:15 1 0 00:01:30 2 1 00:02:00 1 0 00:02:30 3 1	00:00:00:000 State for model input_reader is next time: 00:00:00:000 State for model display is Display Status:Disarmed; Display: Disarmed 00:00:00:000 State for model input_reader is next time: 00:00:10:000 State for model display is Display Status:Disarmed; Display: Disarmed 00:00:10:000 State for model input_reader is next time: 00:00:20:000 State for model display is Display Status:Disarmed; Display: Enter Pin 00:00:20:000 State for model input_reader is next time: 00:00:20:000 State for model display is Display Status:Disarmed; Display: Disarmed 00:00:30:000 State for model input_reader is next time: 00:00:30:000 State for model display is Display Status:Armed; Display: Armed 00:01:00:000 State for model input_reader is next time: 00:00:15:000 State for model display is Display Status:Armed; Display: Enter Pin 00:01:10:000 State for model input_reader is next time: 00:00:15:000 State for model display is Display Status:Armed; Display: Armed 00:01:15:000 State for model input_reader is next time: 00:00:15:000 State for model display is Display Status:Disarmed; Display: Disarmed 00:01:30:000 State for model input_reader is next time: 00:00:30:000

	State for model display is Display Status:Disarmed; Display: Invalid Pin 00:02:00:000 State for model input_reader is next time: 00:00:30:000 State for model display is Display Status:Disarmed; Display: Disarmed 00:02:30:000 State for model input_reader is next time: inf State for model display is Display Status:Disarmed; Display: Door unlocked 00:02:40:000 State for model input_reader is next time: inf State for model display is Display Status:Disarmed; Display: Door unlocked 00:02:50:000 State for model input_reader is next time: inf State for model display is Display Status:Disarmed; Display: Disarmed

The Secure Area System simulates a system where you have to enter a PIN to access a door.