Appendix C

SUL, MUL and Hydrolift Artefacts

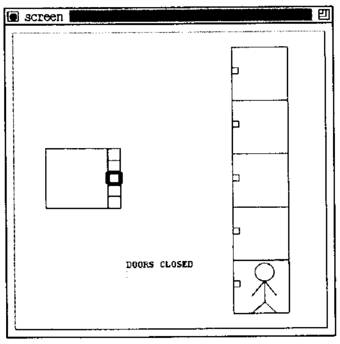
C.1 SUL artefacts

C.1.1 SUL LSD specification

```
agent door() {
state
  door
oracle
  brake
derivate
  door is (brake == ON) ? OPEN : CLOSED
}

agent landing(_F) {
state
  landButton
oracle
  floor direction brake
handle
  brake destination
protocol
  landButton{_F} == ON && _F == floor + direction && brake == OFF -> brake = ON,
  landButton{_F} == ON && direction == NIL -> destination = _F,
  floor == _F -> landButton{_F} = OFF
}

agent car(_F) {
state
  carButton
oracle
  floor direction brake
handle
  brake destination
protocol
  carButton{_F} == ON && _F == floor + direction && brake == OFF -> brake = ON,
  carButton{_F} == ON && direction == NIL -> destination = _F,
  floor == _F -> carButton{_F} = OFF
}
```

```
agent shaft() {
state
  floor destination direction
oracle
  brake
handle
  brake
derivate
  direction is (floor < destination) ?  UP :
               (floor > destination) ?  DOWN : NIL
protocol
  brake == OFF -> floor = floor + direction,
  brake == ON && direction != NIL -> brake = OFF
}
```

C.1.2 SUL visualization/animation



C.1.3 SUL DoNaLD script

The following DoNaLD script defines the SUL visualization.

```
%donald
###########
#LIFT USER#
###########

openshape man
within man {
  circle head
  line body, leftarm, rightarm, leftleg, rightleg
  point manpos
  int rad
  manpos = {0,0}
  rad = 15
  head = circle(manpos, rad)
  body = [manpos - {0,rad}, manpos - {0,50}]
  leftarm = [manpos - {0,rad}, manpos - {20,40}]
  rightarm = [manpos - {0,rad}, manpos - {-20,40}]
  leftleg = [manpos - {0,50}, manpos - {20,70}]
  rightleg = [manpos - {0,50}, manpos - {-20,70}]
}

boolean inlift
inlift = false

openshape person
within person {
  shape person
  point put, one
  put = {~/carpos.1 + 100, ~/carpos.2 + 150}
  one = {800, 10}
  person = if ~/inlift then trans( scale(~/man, 2), put.1, put.2 ) else
           trans( scale(~/man, 2), one.1, one.2+(~/floor*180) )
}

##########
#LIFT CAR#
##########

openshape box
within box {
  int width, length
  point p, q, b, d
  line top, bot, left, right
  b = {0, 0}
  d = b + {width, 0}
  p = b + {0, length}
  q = b + {width, length}
  width, length = 100, 100
  top = [p, q]
  bot = [b, d]
  left = [p, b]
  right = [q, d]
}

int floor
floor = 1

point carpos
carpos = {100, 50+(180*(floor-1)) }

openshape car
within car {
  point corner
  corner = ~/carpos
  shape car
  car = trans( scale(~/box, 2), corner.1, corner.2)
  int X, Y
  X = 200
  real ratio
  ratio = 0.4

  openshape button1
  within button1 {
    shape button1
    boolean light
    light = true
    button1 = trans( scale(~/~/box, ~/ratio), ~/corner.1+~/X,~/corner.2)
  }

  openshape button2
  within button2 {
    shape button2
    button2 = trans( scale(~/~/box, ~/ratio), ~/corner.1+~/X, ~/corner.2+~/ratio*100)
    boolean light
    light = false
  }

  openshape button3
  within button3 {
    shape button3
    button3 = trans( scale(~/~/box, ~/ratio), ~/corner.1+~/X, ~/corner.2+~/ratio*200)
    boolean light
    light = false
  }

  openshape button4
  within button4 {
    shape button4
    button4 = trans( scale(~/~/box, ~/ratio), ~/corner.1+~/X, ~/corner.2+~/ratio*300)
    boolean light
    light = false
  }

  openshape button5
  within button5 {
    shape button5
    button5 = trans( scale(~/~/box, ~/ratio), ~/corner.1+~/X, ~/corner.2+~/ratio*400)
    boolean light
    light = false
  }
}

? A_car_button1_button1 is (carButton_1 == ON) ? "linewidth=5" : "linewidth=0";
? A_car_button2_button2 is (carButton_2 == ON) ? "linewidth=5" : "linewidth=0";
? A_car_button3_button3 is (carButton_3 == ON) ? "linewidth=5" : "linewidth=0";
? A_car_button4_button4 is (carButton_4 == ON) ? "linewidth=5" : "linewidth=0";
? A_car_button5_button5 is (carButton_5 == ON) ? "linewidth=5" : "linewidth=0";

##########
#LANDINGS#
##########

int ceiling, wall
ceiling = 950
wall = 700
real ratio
ratio = 1.8

openshape floor1
within floor1 {
  shape floor1
  floor1 = trans( scale(~/box, ~/ratio), ~/wall, ~/ceiling-~/ratio*500)
  openshape button
  within button {
    shape button
    boolean light
    light = false
    button = trans( scale(~/~/box, 0.2), ~/~/wall, ~/~/ceiling-~/~/ratio*450)
  }
}

openshape floor2
within floor2 {
  shape floor2
  floor2 = trans( scale(~/box, ~/ratio), ~/wall, ~/ceiling-~/ratio*400)
  openshape button
  within button {
    shape button
    boolean light
    light = false
    button = trans( scale(~/~/box, 0.2), ~/~/wall, ~/~/ceiling-~/~/ratio*350)
  }
}
```

```
openshape floor3
within floor3 {
  shape floor3
  floor3 = trans( scale(~/box, ~/ratio), ~/wall, ~/ceiling-~/ratio*300)
  openshape button
  within button {
    shape button
    boolean light
    light = false
    button = trans( scale(~/~/box, 0.2), ~/~/wall, ~/~/ceiling-~/~/ratio*250)
  }
}

openshape floor4
within floor4 {
  shape floor4
  floor4 = trans( scale(~/box, ~/ratio), ~/wall, ~/ceiling-~/ratio*200)
  openshape button
  within button {
    shape button
    boolean light
    light = false
    button = trans( scale(~/~/box, 0.2), ~/~/wall, ~/~/ceiling-~/~/ratio*150)
  }
}

openshape floor5
within floor5 {
  shape floor5
  floor5 = trans( scale(~/box, ~/ratio), ~/wall, ~/ceiling-~/ratio*100)
  openshape button
  within button {
    shape button
    boolean light
    light = false
    button = trans( scale(~/~/box, 0.2), ~/~/wall, ~/~/ceiling-~/~/ratio*50)
  }
}

? A_floor1_button_button is (landButton_1 == ON) ? "linewidth=5" : "linewidth=0";
? A_floor2_button_button is (landButton_2 == ON) ? "linewidth=5" : "linewidth=0";
? A_floor3_button_button is (landButton_3 == ON) ? "linewidth=5" : "linewidth=0";
? A_floor4_button_button is (landButton_4 == ON) ? "linewidth=5" : "linewidth=0";
? A_floor5_button_button is (landButton_5 == ON) ? "linewidth=5" : "linewidth=0";
```

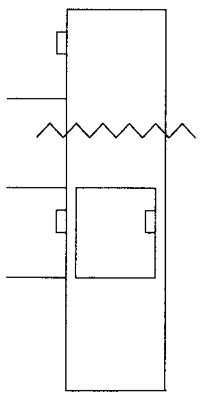
C.1.4 SUL ADM script

The following script defines the ADM entities for the SUL animation.

```
%adm
entity door() {
definition
  door is (brake == ON) ? OPEN : CLOSED
}

entity landing(_F) {
action
  landButton{_F} == ON && _F == floor + direction && brake == OFF -> brake = ON,
  landButton{_F} == ON && direction == NIL -> destination = _F,
  floor == _F -> landButton{_F} = OFF
}
```

```
entity car(_F) {
action
  carButton{_F} == ON && _F == floor + direction && brake == OFF -> brake = ON,
  carButton{_F} == ON && direction == NIL -> destination = _F,
  floor == _F -> carButton{_F} = OFF
}

entity shaft() {
definition
  direction is (floor < destination) ?  UP :
               (floor > destination) ?  DOWN : NIL
action
  brake == OFF -> floor = floor + direction,
  brake == ON && direction != NIL -> brake = OFF
}

# instantiate new entities
door()
shaft()
car(1)
car(2)
car(3)
car(4)
car(5)
landing(1)
landing(2)
landing(3)
landing(4)
landing(5)
```

C.1.5 SUL sketch



C.1.6 SUL statement of requirements

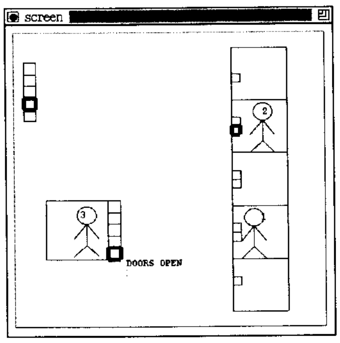
On each landing there is a button and in the car there is a button for each floor. The user makes a request for the car to come to his landing by pressing a button. The shaft mechanism moves the car to his landing and opens the door. The user enters the car and presses a button. The shaft mechanism moves the car to the landing he requested and opens the door. The user exits the car. For safety the door is opened and closed by the brake ensuring that the door is only open whilst the brake is on.

C.2 MUL artefacts

C.2.1 MUL LSD specification

```
agent door() {
state
  door
oracle
  brake
derivate
  door is (brake == ON) ? OPEN : CLOSED
}

agent landing(_F) {
state
  landButton
oracle
  floor direction brake
handle
  brake destination
protocol
  landButton{_F} == UP && _F == floor + 1 && brake == OFF -> brake = ON,
  landButton{_F} == DOWN && _F == floor - 1 && brake == OFF -> brake = ON,
  landButton{_F} != OFF && direction == NIL -> destination = _F,
  floor == _F -> landButton{_F} = OFF
}

agent car(_F) {
state
  carButton
oracle
  floor direction brake
handle
  brake destination
protocol
  carButton{_F} == ON && _F == floor + direction && brake == OFF -> brake = ON,
  carButton{_F} == ON && direction == NIL -> destination = _F,
  floor == _F -> carButton{_F} = OFF
}

agent shaft() {
state
  floor destination direction
oracle
  brake
handle
  brake
derivate
  direction is (floor < destination) ?  UP :
               (floor > destination) ?  DOWN : NIL
protocol
  brake == OFF -> floor = floor + direction,
  brake == ON && direction != NIL -> brake = OFF
}
```

C.2.2 MUL visualization/animation



C.2.3 MUL DoNaLD redefinitions

The following DoNaLD script redefines the SUL visualization as the MUL visualization by redefining the person shape to represent three people.

```
%donald
############
#LIFT USERS#
############

boolean inlift, inliftB, inliftC
inlift = false
inliftB = false
inliftC = false

openshape person
within person {
  shape person, person2, person3
  point put, one
  label p1, p2, p3
  p1 = if ~/inlift then label("1", put) else
           label("1", {one.1,one.2+(~/floor*180)})
  p2 = if ~/inliftB then label(" 2", put) else
           label(" 2", {one.1,one.2+(~/floorB*180)})
  p3 = if ~/inliftC then label("  3", put) else
           label("  3", {one.1,one.2+(~/floorC*180)})
  put = {~/carpos.1 + 100, ~/carpos.2 + 150}
  one = {800, 10}
  person = if ~/inlift then trans( scale(~/man, 2), put.1-30, put.2 ) else
           trans( scale(~/man, 2), one.1-30, one.2+(~/floor*180) )
  person2 = if ~/inliftB then trans( scale(~/man, 2), put.1, put.2 ) else
           trans( scale(~/man, 2), one.1, one.2+(~/floorB*180) )
  person3 = if ~/inliftC then trans( scale(~/man, 2), put.1+30, put.2 ) else
           trans( scale(~/man, 2), one.1+30, one.2+(~/floorC*180) )
}
```

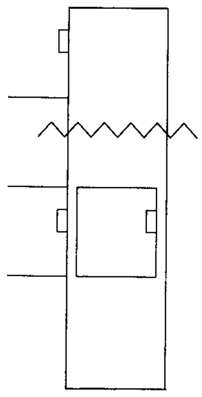
C.2.4 MUL ADM redefinitions

The following ADM script redefines entities in the SUL animation by instantiating new user entity definitions and redefining the `landing` entity.

```
%adm
entity userIn(_U) {
```

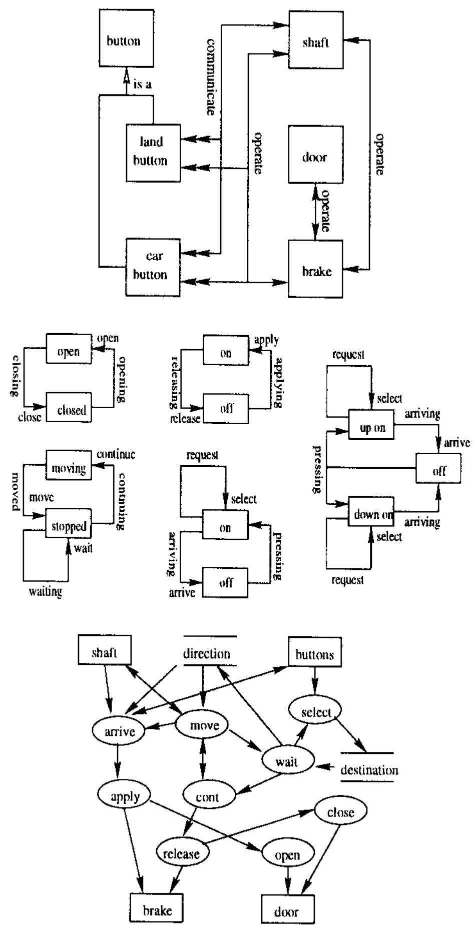
```
definition
  floor{_U} is floor
action
  Rand(2) == 1 && door == OPEN -> delete_userIn(_U); floor{_U} = floor; userOut(_U),
  Rand(2) == 1 -> execute("carButton_"//str(Rand(5))//" = ON;")
}

entity userOut(_U) {
action
  Rand(2) == 1 && door == OPEN && floor == floor{_U} -> delete_userOut(_U); userIn(_U),
  Rand(2) == 1 -> execute("landButton_"//str(floor{_U})//" = UP;"),
  Rand(2) == 1 -> execute("landButton_"//str(floor{_U})//" = DOWN;")
}

entity landing(_F) {
action
  landButton{_F} == UP && _F == floor + 1 && brake == OFF -> brake = ON,
  landButton{_F} == DOWN && _F == floor - 1 && brake == OFF -> brake = ON,
  landButton{_F} != OFF && direction == NIL -> destination = _F,
  floor == _F -> landButton{_F} = OFF
}

# instantiate new entities
userIn(1)
userIn(2)
userIn(3)
```

C.2.5 MUL sketch



C.2.6 MUL statement of requirements and models

On each landing there is an up and a down button. In the car there is a button for each floor. Users make requests for the car to come to their landing or go to another landing by pressing these buttons. The shaft mechanism moves the car towards a destination landing stopping whenever the brake is applied. The brake is applied whenever the car arrives at a landing requested by a user (for requests from landings the direction matters). On arriving at the destination landing the shaft mechanism selects the next destination. The shaft mechanism releases the brake and starts the car moving again. For safety the door is opened and closed by the brake ensuring that the door is only open whilst the brake is on.



open	open door
close	close door
apply	apply brake generate opening
release	generate closing release brake
select	change destination
arrive	reset button generate applying
continue	generate releasing generate moved
move	generate arriving move car generate waiting or continuing
wait	generate request change direction generate waiting or continuing

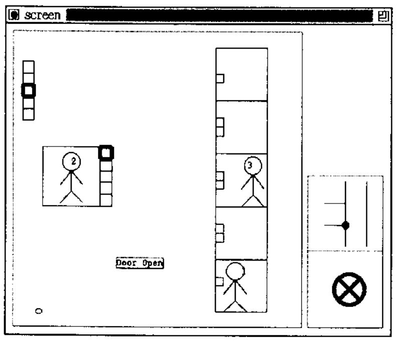
C.3 Hydrolift artefacts

C.3.1 Hydrolift LSD specification

```
agent door() {
state
  door
oracle
  brake
derivate
  door is (brake == ON) ? OPEN : CLOSED
}

agent landing(_F) {
state
  landButton
oracle
  sensed brake
handle
```

```
  brake
protocol
  landButton{_F} == UP && sensed{_F - 1} == UP && brake == OFF -> brake = ON,
  landButton{_F} == DOWN && sensed{_F + 1} == DOWN && brake == OFF -> brake = ON,
  sensed{_F} != NIL -> landButton{_F} = OFF
}

agent car(_F) {
state
  carButton
oracle
  chan2
handle
  chan1
protocol
  carButton{_F} == ON -> chan1 = _F,
  chan2 == _F -> carButton{_F} = OFF
}

agent pump() {
state
  change target
oracle
  brake pressure chan1
handle
  brake pressure chan2
derivate
  k = 100,
  change is (pressure < target) ? k :
            (pressure > target) ? -k : 0
protocol
  target == pressure + change && brake == OFF -> brake = ON,
  change == 0 -> target = chan1*k,
  pressure == target -> chan2 = target/k,
  brake == OFF -> pressure = pressure + change,
  brake == ON && change != 0 -> brake = OFF
}
```

C.3.2 Hydrolift visualization/animation



C.3.3 Hydrolift DoNaLD redefinitions

The following DoNaLD script redefines the MUL visualization as the Hydrolift visualization by adding shapes for the pump and valve.

```
######
#PUMP#
```

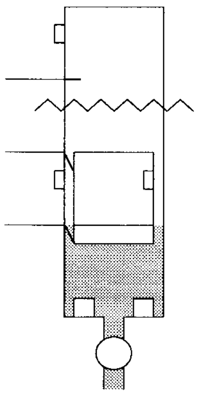
```
######

openshape pumpshape
within pumpshape {
  circle base
  point pos
  int radius
  boolean on
  base = circle(pos, radius)
  pos = {460,350}
  radius = 40
  on = false
  line one, two
  point p1, p2, p3, p4
  int const
  one = [ p1 , p2 ]
  two = [ p3 , p4 ]
  const = 29
  p1 = pos - {const, const}
  p2 = pos + {const, const}
  p3 = pos + {-const, const}
  p4 = pos + {const, -const}
}

? A_pumpshape_one is (change > 0) ? "linewidth=5" : "linewidth=0";
? A_pumpshape_two is (change > 0) ? "linewidth=5" : "linewidth=0";
? A_pumpshape_base is (change > 0) ? "linewidth=5" : "linewidth=0";

#######
#VALVE#
#######

openshape diaphragm
within diaphragm {
  line pipe1, pipe2, pipe3, pipe4, pipe5
  point one, two, three, four, five, six, seven, eight
  int height, width
  point pos
  height = 300
  width = 100
  pos = {200, 200}
  two = pos + {0, height}
  three = pos + {-width, 0}
  four = pos + {-width, 100}
  pipe1 = [ pos, two]
  pipe2 = [ three, four ]
  five = four + {0, 100}
  six = three + {0, height}
  pipe3 = [ five, six ]
  seven = five - {100, 0}
  eight = four - {100, 0}
  pipe4 = [ seven, five ]
  pipe5 = [ eight, four ]
}

openshape valve
within valve {
  circle base
  int radius
  base = circle(~/diaphragm/four, radius)
  radius = 10
  line valvepos
  point end
  int change
  boolean open
  change = if open then ~/diaphragm/width else 0
  end = ~/diaphragm/five + {change, 0}
  valvepos = [ ~/diaphragm/four, end ]
  open = false
}
```

C.3.4 Hydrolift ADM redefinitions

The following ADM script redefines the entities in the MUL animation by instantiating the entity definitions for the pump and sensor and changing the definitions of the landing and car entities.

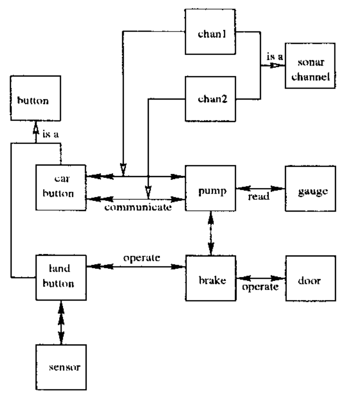
```
%adm
entity landing(_F) {
action
  landButton{_F} == UP && sensed{_F - 1} == UP && brake == OFF -> brake = ON,
  landButton{_F} == DOWN && sensed{_F + 1} == DOWN && brake == OFF -> brake = ON,
  sensed{_F} != NIL -> landButton{_F} = OFF
}

entity car(_F) {
action
  carButton{_F} == ON -> chan1 = _F,
  chan2 == _F -> carButton{_F} = OFF
}

entity sensor(_F) {
definition
  sensed{_F} is ( pressure == _F*k && change == k ) ? UP :
                ( pressure == _F*k && change == -k ) ? DOWN : NIL
}

entity pump() {
definition
  k = 100,
  change is (pressure < target) ? k :
            (pressure > target) ? -k : 0
action
  target == pressure + change && brake == OFF -> brake = ON,
  change == 0 -> target = chan1*k,
  pressure == target -> chan2 = target/k,
  brake == OFF -> pressure = pressure + change,
  brake == ON && change != 0 -> brake = OFF
}

# instantiate new entities
sensed_0 = 0
sensor(1)
sensor(2)
sensor(3)
sensor(4)
sensor(5)
sensed_6 = 0
pump()
```

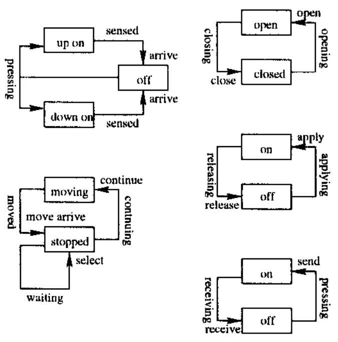
C.3.5 Hydrolift sketch

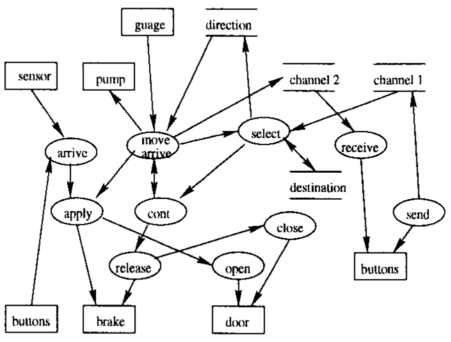


C.3.6 Statement of requirements and models

On each landing there is an up and a down button. In the car there is a button for each floor. Users make requests for the car to come to their landing or go to another landing by pressing these buttons. The shaft mechanism consists of sensors for detecting the direction of the car, sonar for communicating between a pump and the car, and a guage situated with the pump at the base of the shaft for detecting the pressure of the water in the shaft. The sonar has a channel from car to pump (channel 1) and a channel from pump to car (channel 2) which carry floor numbers. The pump moves the car towards a destination landing stopping whenever the brake is applied. The brake is applied whenever the car arrives at a landing requested by a user (for requests from landings the direction matters). The landing buttons are reset locally and the car buttons are reset by a signal on sonar channel 2. On arriving at the destination landing the pump selects the next destination from those transmitted on sonar channel 1. The pump releases the brake and starts the car moving again. For safety the door is opened and closed by the brake ensuring that the door is only open whilst the brake is on.







`open`	`open door`
`close`	`close door`

`apply`	`apply brake`
	`generate opening`

`release`	`generate closing`
	`release brake`

`receive`	`reset button`
`send`	`queue destination`

`select`	`change destination and direction`
	`generate waiting or continuing`

`arrive (landing)`	`reset button`
	`generate applying`

`arrive (pump)`	`generate receiving`
	`generate applying`

`continue`	`generate releasing`
	`generate moved`

`move`	`generate arriving`
	`move car`
	`generate waiting or continuing`