Elevator Design
Walk-through

18-540 Distributed Embedded Systems
John DeVale
September 22, 2000
Overview: Why Design

- Provides an intermediate step between specification and code that allows a level of rigidity to be placed on solution in such a fashion as to shed light on potential pitfalls in implementation without building so much infrastructure that avoiding the pitfalls become difficult.

- Easier to meet Spec

What happens when food implementation does not meet specification:
Describing a behavior in terms of state

- Isolate the basic functionality – What does it do at the highest level of abstraction
- Determine the messages it must get to transit from each state into the adjacent node of the digraph (Event Driven – What we are doing now)
  - AKA. Extract the conditions under which it changes what it is doing (Time Driven)
- Build the information into states and transitions
- Determine where it starts (Initialization State)
A simple example – Dispatch (our dispatch)

◆ Its behavioral specification:

BEHAVIORS:

◆ 1.1 DesiredFloor.f shall always be set to Target.
◆ 1.2 DesiredFloor.d shall always be set to Stop.
◆ 1.3 Whenever any DoorClosed [j] is False, Target shall be set equal to
  \((\text{CurrentFloor}-1) \mod \text{MaxFloors}) + 1\)
◆ 1.4 DesiredDwell shall always be set to a constant appropriate value for door open dwell.

◆ What does it really do?

• It orders the car to a new position (if the doors are not close)

  OR

• It doesn’t (if the doors are closed)
BEHAVIORS:

1.1 DesiredFloor.f shall always be set to Target.

1.2 DesiredFloor.d shall always be set to Stop.

1.3 Whenever any DoorClosed [j] is False, Target shall be set equal to
   \((\text{CurrentFloor}-1) \mod \text{MaxFloors}) + 1\)

1.4 DesiredDwell shall always be set to a constant appropriate value for door open dwell.
How does it get from one state to another

- The door ceases to be closed
- Or it ceases to be not closed

Order

| Target=(((CurrentFloor-1) mod MaxFloors)+1) |
| DesiredFloor=[Target,Stop] |
| DesiredDwell=3000ms |

Not Order

Door Closes

Door un-closes
Where does it start

This is problematic. Consider the following:

• The elevator starts on floor #1
• The elevator starts with desired floor=lobby, stop
• The car starts on the lobby (1st floor)
• The doors start shut

So?

• So there is no reason for the elevator to move or open its doors, thus the dispatcher will never order the car to a different floor
• Given our design, the only way to get the elevator moving is for someone in the Lobby to press a hall call button, causing the doors to open.

• Ooops. But, we have the opportunity to change the design before code is written, minimizing its painfulness.
Final Dispatch Design

Doors Open

Target=\(((\text{CurrentFloor}-1) \mod \text{MaxFloors})+1)\)
DesiredFloor=\[[\text{Target},\text{Stop}]\]
DesiredDwell=3000ms

Doors Closed

A()
B()
C()

InitState

Target=\(((\text{CurrentFloor}-1) \mod \text{MaxFloors})+1)\)
DesiredFloor=\[[\text{Target},\text{Stop}]\]
DesiredDwell=3000ms

Transition | Message | Guard Conditions |
---|---|---|
A | DoorClosed | DoorClosed=True |
B | DoorClosed | DoorClosed=False |
C | Atfloor | AtFloor[f,d]=None |
A bit more complicated – DoorMotorControl

BEHAVIORS:

1.1 If AtFloor[f,d] is None set Cycles[j] to zero.

1.2 If any DoorReversal[q] is True then: command DoorMotor[j] to Open; increment Cycles[j]; set Dwell[j] to an appropriate value.

1.3 If CurrentFloor equals DesiredFloor.f, and Drive is commanded to Stop, and Cycles[j] is zero then: command DoorMotor[j] to Open; increment Cycles[j]; set Dwell[j] to an appropriate value.

1.4 If CurrentFloor equals DesiredFloor.f, and Drive is commanded to Stop, and either (HallCall[CurrentFloor,DesiredFloor.d] is true) or (any HallCall[CurrentFloor,*) is true and DesiredFloor.d is stop), then: command DoorMotor[j] to Open; increment Cycles[j]; set Dwell[j] to an appropriate value.

1.5 If CurrentFloor equals DesiredFloor.f, and Drive is commanded to Stop, and CarCall[CurrentFloor] is True, then: command DoorMotor[j] to Open; increment Cycles[j]; set Dwell[j] to an appropriate value.

1.6 When DoorOpen[j] transitions from False to True: set CountDown[j] to Dwell[j]; command DoorMotor to Stop.

1.7 When DoorClosed[j] transitions from False to True: command DoorMotor to Stop.

1.8 When CountDown[j] transitions to zero: command DoorMotor to Close.
What does it really do?

- Tells motors to open doors
- Tells motors to shut doors
- Tells motor to stop
  - Could stop open
  - Could stop shut
Door Motor Control States

Motor Open
- Inc Cycle
- Set Dwell
- DoorMotor[j]=Open

Motor Stop/Open
- DoorMotor[j]=Stop
- Set Countdown

Motor Stop/Shut
- DoorMotor[j]=Stop
- If AtFloor[f,d]=None Cycles=0

Motor Close
- DoorMotor[j]=Close
To get from open to stop/open

- \( \text{DoorOpen}[j] = \text{True} \)

```
DoorOpen[j] = True
Motor Open
Inc Cycle
Set Dwell
DoorMotor[j] = Open
```

```
Motor Stop/Open
DoorMotor[j] = Stop
Set Countdown
```
From stop/open to close

- **Countdown expires**

  Motor Stop/Open

  DoorMotor[j]=Stop
  Set Countdown

  Countdown=0

  Motor Close

  DoorMotor[j]=Close
From close to stop/closed

- DoorClosed[j]=True

Motor Stop/Shut

DoorMotor[j]=Stop
If AtFloor[f,d]=None Cycles=0

DoorClosed=True

Motor Close

DoorMotor[j]=Close
From stop/closed to open

• Transition E
  
  • CurrentFloor=DesiredFloor and Drive=Stop and Cycles=0
  • CurrentFloor=DesiredFloor and Drive=Stop and CarCall[CurrentFloor]=True
  • CurrentFloor=DesiredFloor and Drive=Stop and HallCall[CurrentFloor, DesiredFloor.d]=True
  • CurrentFloor=DesiredFloor and Drive=Stop and HallCall[CurrentFloor, *] and DesiredFloor.d=Stop

Motor Stop/Shut

  DoorMotor[j]=Stop
  If AtFloor[f,d]=None Cycles=0

Motor Open

  Inc Cycle
  Set Dwell
  DoorMotor[j]=Open

E()
Where does that leave us?

**Transition Message Guard Conditions**

<table>
<thead>
<tr>
<th>Transition</th>
<th>Message</th>
<th>Guard Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>DoorOpen</td>
<td>$\text{DoorOpen}[j]=\text{True}$</td>
</tr>
<tr>
<td></td>
<td>Countdown</td>
<td>$\text{Countdown}=0$</td>
</tr>
<tr>
<td>C</td>
<td>DoorClosed</td>
<td>$\text{DoorClosed}[j]=\text{True}$</td>
</tr>
<tr>
<td>D</td>
<td>AtFloor</td>
<td>$\text{CurrentFloor}=\text{DesiredFloor}$ and $\text{Drive}=\text{Stop}$ and $\text{Cycles}=0$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or $\text{CurrentFloor}=\text{DesiredFloor}$ and $\text{Drive}=\text{Stop}$ and $\text{CarCall}[\text{currentFloor}]=\text{True}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or $\text{CurrentFloor}=\text{DesiredFloor}$ and $\text{Drive}=\text{Stop}$ and $\text{HallCall}[\text{currentFloor}, \text{DesiredFloor.d}]=\text{True}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or $\text{CurrentFloor}=\text{DesiredFloor}$ and $\text{Drive}=\text{Stop}$ and $\text{HallCall}[\text{currentFloor},^*]$. and $\text{DesiredFloor.d}=\text{Stop}$</td>
</tr>
</tbody>
</table>
Not quite enough…

- We still haven’t handled the door reversal case

<table>
<thead>
<tr>
<th>Transition</th>
<th>Message</th>
<th>Guard Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>DoorReversal</td>
<td>DoorReversal[q]=True</td>
</tr>
<tr>
<td>B</td>
<td>DoorOpen</td>
<td>DoorOpen[j]=True</td>
</tr>
<tr>
<td>C</td>
<td>Countdown</td>
<td>Countdown=0</td>
</tr>
<tr>
<td>D</td>
<td>DoorClosed</td>
<td>DoorClosed[j]=True</td>
</tr>
<tr>
<td>E</td>
<td>AtFloor</td>
<td>CurrentFloor=DesiredFloor and Drive=Stop and Cycles=0</td>
</tr>
</tbody>
</table>
One last problem

- Cycles can currently never be reset to zero
- When the doors close, AtFloor is still true
- Thus the closed/stop state must receive AtFloor messages
The Final Design - DoorMotorControl

Transition | Message          | Guard Conditions                                                                 
------------|------------------|----------------------------------------------------------------------------------
A           | DoorReversal     | DoorReversal[q]=True                                                             
B           | DoorOpen         | DoorOpen[j]=True                                                                
C           | Countdown        | Countdown=0                                                                     
D           | DoorClosed       | DoorClosed[j]=True                                                              
E           | AtFloor          | CurrentFloor=DesiredFloor and Drive=Stop and Cycles=0 or CurrentFloor=DesiredFloor and Drive=Stop and CarCall[currentFloor]=True or CurrentFloor=DesiredFloor and Drive=Stop and HallCall[currentFloor,DesiredFloor.d]=True or CurrentFloor=DesiredFloor and Drive=Stop and HallCall[currentFloor,*] and DesiredFloor.d=Stop AtFloor[f,d]=None
F           | AtFloor          |                                                                                

Transition Message Guard Conditions

Motor Stop/Open
DoorMotor[j]=Stop
Set Countdown

Motor Stop/Shut
DoorMotor[j]=Stop
If AtFloor[f,d]=None Cycles=0

Motor Open
Inc Cycle
Set Dwell
DoorMotor[j]=Open

Motor Close
DoorMotor[j]=Close

Transition Message Guard Conditions

A DoorReversal DoorReversal[q]=True
B DoorOpen DoorOpen[j]=True
C Countdown Countdown=0
D DoorClosed DoorClosed[j]=True
E AtFloor CurrentFloor=DesiredFloor and Drive=Stop and Cycles=0 or CurrentFloor=DesiredFloor and Drive=Stop and CarCall[currentFloor]=True or CurrentFloor=DesiredFloor and Drive=Stop and HallCall[currentFloor,DesiredFloor.d]=True or CurrentFloor=DesiredFloor and Drive=Stop and HallCall[currentFloor, *] and DesiredFloor.d=Stop AtFloor[f,d]=None
F AtFloor
Recap

- Isolate what it does
- Figure out how it changes “modes”
- Complete a “Sanity” Check