Recitation #4

18-649 Distributed Embedded Systems
Friday 19-Sep-2014
Announcements and Administrative Stuff

◆ Project 4 posted

◆ TA office hours
  ◆ http://www.ece.cmu.edu/~ece649/admin.html#info
  ◆ Monday: PH 126A 5:00-6:00 (Sajjan)
  ◆ Tuesday: WEH 5328 5:00-6:00 (Felix)
  ◆ Wednesday: WEH 5310 6:00-7:00 (Patrick)
  ◆ Thursday: PH A22 5:00-6:00 (Jeff)
  ◆ Friday: WEH 5328 5:00-6:00 (Felix)

◆ Submission Mistakes
  • Please place portfolio files in the project root directory with no additional directories.
    – Correct: proj3\(portfolio files)
    – Incorrect: proj3\portfolio\(portfolio files)
  • Minimum Contribution chart in peer review folder.
TA Office Hours

- **If you have questions about grading on a project**
  - Go see the TA that graded your project if possible

- **For grade correction requests or disputes**
  - You must submit a written (paper) request including:
    - Your name
    - TA name that graded the assignment
    - Specific issue with grading
  - Within 1 week of when the grade is posted to blackboard
    - We’ll be a little flexible with projects 1&2 since it took a while to settle down office hours
Anyone have to update sequence diagrams to add missed behaviors?
  • This is expected
  • Good design process helps identify these bugs before implementation!

Some common things some might have missed:
  • Turning hall and car button lights OFF
    – If you see the button has already lit up, would you press it again?
  • Setting car position indicator
    – How does the passenger know when to get off the elevator?
  • What about safety cases?

Other notes:
  • Why do mHallLight and mCarLight exist?
    – Typically used for fancy dispatchers and fault tolerance
    – For state chart traceability, you can mark these as “future expansion”
      » But, any reasonable approach is fine so long as it is consistently applied
Project 4 Overview

- Convert your event-triggered requirements to time-triggered
- Create state charts using time-triggered requirements
- Traceability between requirements and state charts
- Log any changes to requirements, sequence diagrams, etc.
An event triggers a message to be sent ONCE
  • E.g. “Passenger presses a button”

Controllers take actions when they receive a particular message
  • Receiving a message is an event that triggers some action

Controllers can only act on one new message at a time
  • If actions require more than one message, controller has to store them
Now: Time-Triggered

- Think of messages as periodic updates of system state variables
  - E.g. Repeatedly check “Is the button currently pressed?”

- Controllers take actions based upon the current state of the system
  - Controllers run control loops at regular intervals
  - Constantly monitor the most recent values of messages
    - Actions performed once the most recent values match a particular set of conditions

- Controllers keep the most recent copy of messages
  - Current state = most recent copies of messages
Another Magic Formula

◆ Time-triggered system

• (Null or <message value>, … <message value>)
  and (Null or <variable value test>, … <variable value test>)
  shall result in <message transmitted>, …
  <variable value assigned>

• Can trigger on zero or more messages; zero or more variables
  – Need one or more total triggers
  – OK for left hand side trigger to ONLY be a state variable (or always be true)
  – Right hand side can have zero or more messages; zero or more variable values
  – “Shall” and “should” are both acceptable

• OK to assign multiple messages, OK to assign multiple values

• EVERY VERB GETS A NUMBER
Correct and Incorrect TT Requirement Examples

◆ Correct:

R1. If X and Y then
   R1.a. M shall be set to \( m \)
   R1.b. N shall be set to \( n \)

• One number per verb
• Reminder: Trace to the sub-numbered bullets

◆ Wrong:

R1. If X and Y then \( M \) shall be set to \( m \) and \( N \) shall be set to \( n \)

*Problem: More than one verb per traceable numbered requirement*
Time-Triggers Requirements Guidance

- **Use typical message format to refer to the most recent copy**
  - You don’t have to explicitly store the newest copy

- **Example:**
  R1. If (mAtFloor\[g,b\] is true) and (mDesiredFloor.f = = g), then
    R1.a. mCarCall\[g,b\] shall be set to false, and
    R1.b. CarLight\[g,b\] shall be set to false, and
    R1.c. mCarLight\[g,b\] shall be set to false.

- **Time-triggered requirements act on the current state of the system**
  - Don’t refer to a message “being received” or some other event
How Does This Impact Sequence Diagrams?

- **Message arcs represent the change in value**
  - Event-triggered: The time when a single message value is broadcast
  - Time-triggered: The time when a periodic message value changes
  - So, the number of message arcs should remain about the same

- **Time-triggered requirements may simplify your sequence diagrams**
  - You may not need to explicitly store variables now
  - Some of your variable assignment bubbles might need to be removed

- **Update sequence diagrams if a behavior is changed, added, or removed**

- **Yes, if you modify sequence diagrams you must update traceability**
  - *You must enter each change in the issues log if it is a defect rather than an enhancement*
    - (Until mid-semester, almost everything you change will be due to finding a defect)
State Charts

◆ **Event-Triggered:**
  - Arcs are taken in response to received message
  - Asynchronous state machine
    - Only does something when an event occurs
    - Action inside a state takes place exactly *once* per arc transition
  - Switch statements for state machine are executed once per arriving arc

◆ **Time-Triggered:**
  - Arcs are taken periodically if conditions are true
  - Synchronous state machine
    - Does something on regular period regardless of changes
    - Actions inside state occur repeatedly (every period)
  - Switch statement for state machine executed once per period

◆ **What’s the difference?**
  - What happens when you increment a variable within a state in an event-triggered state machine vs time-triggered?
State Charts

◆ **Create state charts based on your time-triggered requirements**
  - Each state must set all outputs of the control interface in every state
  - Make decisions based ONLY on the current state of the system
  - Have mutually excluding transitions
    - No two guard statements can be simultaneously true on arcs from same state
    - Implicit “stay in same state” guard condition if no other guards are true
  - Note that action inside a state happens every time state chart is evaluated
    - So if you have “set light to on” and the state chart runs at 10x/second, the light gets an “on” command 10 times per second
  - For now you can run state charts as fast as you want
    - (In general run them at least as fast as the fastest message repetition rate)

◆ **Create three tables per state chart**
  - State activities table
  - Transitions table
  - Traceability for states and transitions to requirements
  - See examples
State Charts

◆ Forbidden
  
  • No actions on arcs
    – All actions performed in the state
  
  • No entry actions (actions occurring only once upon entry)
  
  • No branches in transitions
    – Just make more than one transition

◆ Avoid:

  • Using a state variable to collapse states
    – Break it down into two separate states
    – Compact does not mean easier to read / understand / implement!

  • Nested state charts
    – There’s examples of how to do it correctly in the Soda Machine
    – Still not recommended
ButtonControl Time Triggered Statechart

<table>
<thead>
<tr>
<th>Transition #</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2.1</td>
<td>mButton[s] ← True AND mEmpty[s] ← False</td>
</tr>
<tr>
<td>T2.2</td>
<td>mVend ← True AND mEmpty[s] ← False</td>
</tr>
<tr>
<td>T2.3</td>
<td>mVend ← True AND mEmpty[s] ← True</td>
</tr>
<tr>
<td>T2.4</td>
<td>FlashCounter &gt; FlashLimit</td>
</tr>
<tr>
<td>T2.5</td>
<td>FlashCounterLimit ← 0</td>
</tr>
<tr>
<td>T2.6</td>
<td>mEmpty ← True</td>
</tr>
<tr>
<td>T2.7</td>
<td>mEmpty ← False</td>
</tr>
</tbody>
</table>

**State IDLE**
- Do:
  - Set ButtonLight[s] to True.
  - Set mButton[s] to False.
  - Set FlashCounter to 0.

**State EMPTY**
- Do:
  - Set ButtonLight[s] to False.
  - Set mButton[s] to False.
  - Set FlashCounter to 0.

**State VEND**

**State FLASH_OFF**
- Do:
  - Set ButtonLight[s] to False.
  - Set mButton[s] to True.
  - Increment FlashCounter.

**State FLASH_ON**
- Do:
  - Set ButtonLight[s] to True.
  - Set mButton[s] to True.
  - Decrement FlashCounter.
ButtonControl Time Triggered Statechart

- Each state gets a name
- All transitions are numbered
- Each state updates all interface outputs (and possibly variables)

State IDLE
Do:
Set ButtonLight[s] to True.
Set mButton[s] to False.
Set FlashCounter to 0.

State EMPTY
Do:
Set ButtonLight[s] to False.
Set mButton[s] to False.
Set FlashCounter to 0.

State VEND

State FLASH_OFF
Do:
Set ButtonLight[s] to False.
Set mButton[s] to True.
Increment FlashCounter.

State FLASH_ON
Do:
Set ButtonLight[s] to True.
Set mButton[s] to True.
Decrement FlashCounter.
A Brief Word Nested State Charts

- They’re tricky
  - Can make implementation and traceability a pain too sometimes

- Avoid nested state charts (the stuff in the blue box)
  - Your state charts aren’t going to be complex enough to need this
Traceability

- **Forward:**
  - Does every requirement map to at least one state or transition?

- **Backward:**
  - Does every state or transition map to at least one requirement?

- **Include this table in your behavioral requirements**

<table>
<thead>
<tr>
<th>States</th>
<th>R2.1</th>
<th>R2.2</th>
<th>R2.3</th>
<th>R2.4a</th>
<th>R2.4b</th>
<th>R2.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>EMPTY</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>VEND</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLASH_OFF</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLASH_ON</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transitions</th>
<th>R2.1</th>
<th>R2.2</th>
<th>R2.3</th>
<th>R2.4a</th>
<th>R2.4b</th>
<th>R2.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2.1</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>T2.2</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2.3</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>T2.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>T2.6</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
Traceability Updates and Issues Log

◆ If you change or add a behavior, update your sequence diagrams

◆ Update your issues log

◆ Retrace sequence diagram arcs to requirements to state charts

◆ We require end-to-end traceability
  • It takes longer than you would like, make sure you leave time for it!
Notes On Defect Tracking

◆ If you find a problem while you are working on something, don’t bother logging it
  • Defects “count” once you try to unit test, peer review, or check code in
  • In other words, start counting defects when you think an item is ready to push to the next phase

◆ For peer review record defects on a peer review log
  • Only promote to the Issue log if not fixed by the weekly due date (i.e., for every “not fixed” entry in a review log there should be an entry in the issue log added that week)
  • When reporting defects in presentation metrics, include peer review defect count, even if defect was closed that week

◆ For tests, record defects in test log AND issue log
  • You can add all review defects to issue log if you want for consistency, but it is optional
Questions?