CMPE-013/L

Event Driven Programming
State Machines
Gabriel Hugh Elkaim
Winter 2014

Traditional Program Structures

Programming Embedded Systems (1.2)

Programming Embedded Systems (2.2)
Events and Services Framework (1.4)
- Concept Framework (Paradigm Methodology)
- Essential way of logically and data processing
- Simplify Define First
- Make it clear how to define key level factors
- Make DEPENDENT DEPENDENT

Events and Services Framework (2.4)
Rule #1
- Tasks break down into only two:
  1. End Debugger
  2. Service Finish

    for i = 0 to completion
    end

What is an Event?

What Happens with Noise?
Add Hysteresis

Events and Services Framework (3.4)
- Keep Event Detection at Amis' Finches
- Make as Puitive (Fair)
- Make Them Non-blocking

Events and Services Framework (4.4)

A Mechanical Cockroach
A Mechanical Cockroach

- Light Goes On
- Drive Forward
- Light Goes Off
- Stop
- Contact Object
  - Turn left, drive in reverse for 3 sec.
  - Reverse Timer Expires
  - Turn straight, simulate Light Goes Off event

State Machines (1.4)

- Describe an abstract machine
- At any point in time it can be in only one state
  - a fluid number of states
- Next state is a function of the current state
  - and the input (input)
  - Symbolic, changing behavior from one state to the next.
State Machines (2.4)
- Useful for studying behavior of an event driven program
- Allows you to explore behavior before you write your code.
- Not directly into the actual event driven framework.

State Machines (3.4)

State Machines (4.4)

State Diagram Conventions (1.2)
State Diagram Conventions (2.2)

Example: Smart Combination Lock

Combination = 0-1-3

Example Code: IF-ELSE

```c
void SmartCombinationLock(iface index)
{
    // Possible states and transitions
    switch (iface.state)
    {
        case Open:
            // ... transitions...
            break;
        case Close:
            // ... transitions...
            break;
        case Unlock:
            // ... transitions...
            break;
        case Timeout:
            // ... transitions...
            break;
        default:
            // Default case
            break;
    }
}
```
Example Code: CASE

```c
//

Example Code: CASE

```
Microwave Oven

(a) The Open button opens the door, stops cooking, and holds time.
(b) The Start button clears the timer. If cooking was active, it is disabled.
(c) The Start button starts cooking for whatever time has been set. While cooking, the timer decrements to zero and turns off cooking when it reaches zero.
(d) The Popcorn button forces a time of 3 minutes to be set into the timer.
(e) The Defrost button forces the power level to 50%.
(f) Time is set by twisting the dial and then pressing the button at the center of the dial. No action takes place until the button is pressed. At that point, the time on the dial is entered into the timer.
(g) There is a switch connected to the door to show whether it is open or closed.

Pseudo-Code (PDL)

- PDL = Program Design Language
- Pseudo-code is written in ENGLISH.
- Doesn’t use the syntax of any particular programming language.
- It is a written, low-level exploration of an implementation of an algorithm.
- It can form the first level of comments for your code.

Roach Library

- You need to initialize the functions by calling 
  RoachInit();
- Functions available for controlling the motors (see documentation for full details):
  LeftMtrSpeed(x); RightMtrSpeed(x);
  - x is a number from -10 (reverse) to 10 (forward)
- Functions available for checking the bumpers:
  uchar ReadBumpers();
- Function available for reading the light level:
  uchar LightLevel();

Questions?