Event-Driven Programming and State Machines

Gabriel Hugh Elkaim
Fall 2015
Traditional Program Structures

START

IF data set

FOR, WHILE DO/WHILE

(END)

COMMAND LINE PROGRAMS

main

{ while (c) 

{ }

} main/ISR

gui

Gabriel Hugh Elkaim – Fall 2015

CMPE 118/218 – Intro. to Mechatronics
Asynchronous — any input can happen at any time

Simultaneous — inputs & outputs

Sequence of inputs & outputs are *UNKNOWN, UNKNOWABLE, RE: ORDERABLE*

No end, No exit
Inputs - Sensors - switches, light sensors, temperature, timers
User inputs - Push button, keypad, communications

Outputs - Update a display
- Turn something on/off (motors)
- In general, change something physical

RTOS - Real-Time Operating System
- Vx works ~ $30k/year
- QNX - $12k
Events and Services Framework (1.4)

Formalized methodology - concept first framework

Excellent method for implicitly write down programs (state machines)

Emphasize **DESIGN** first

- Make it clear how to define your low-level functions

- Makes **DEBUGGING** easier
Events and Services Framework (2.4)

**Rule #1**

Tasks break down into **only** two fundamental classes:

1. Event Detector
2. Service

\[
\text{While \ (time\ Not\ Expired(1)) \ }
\]

//rest of code
What is an Event?

Some detectable change

open

Event

switch bounce

closed

1-3 mm

12.5 mm
What Happens with Noise?
Add Hysteresis

![Graph showing hysteresis behavior with threshold levels](image)

- On
- Off
- High Threshold
- Signal Plus Noise
- Low Threshold
Events and Services Framework (3.4)

Corollary to Rule #1

- Keep event dispatchers and service routines as short and fast as possible

- Make them non-blocking!!
Events and Services Framework (4.4)

Complete Program Structure

Initialize all hardware/software

while (i) {
    Test for events << read & do
    Service events

}
State Machines (1.4)

- Description of an Abstract Machine

- At any point in time it can be in only one (and only one) of a fixed number of states.

- Next state in a progression depends only on the current state and the input events.

- Idealized, instantaneous transitions from one state to the next.
State Machines (2.4)

- Useful tool for describing the behavior of an event-driven program.
- Allows you to explore the behavior **before** you implement any code at all.
- Naturally fit into Event/Service framework.
Gerund - "in" State Machines (3.4)
Dining Philosophers Problem

1987
State Machines (4.4)

"uni"    "match"

Diagram of state machine with two states: 'Start' and 'End'. An arrow labeled 'send (grant) to

...and another arrow labeled 'catch' leading to a state labeled 'Match'.

Gabriel Hugh Elkaim – Fall 2015
CMPE 118/218 – Intro. to Mechatronics
Example: Smart Combination Lock

Combination = 1-1-8
Example: Smart Combination Lock

Combination = 1-1-8

- Close Lock
- Open
- Timeout
- Twist Unlock
- None Right
- 1 Right
- 2 Right
- 3 Right
- 8

Start Timer

NA

NA

1

1

1

1

8

NA

NA

1
ES – Software Events and Services (1.4)

ES Framework - supplies services

events → service

State Machine
Nested if-then-else
static enum State
if (state == locked) {
  if (cmd == "1") {
    all state = open right;
  } else if (state == ...}

Nested Case Study:
static enum State
switch (state) {
case locked:
  switch (cmd) {
    case "1":
      break;
    ;;
  }
}
ES – Software Events and Services (2.4)

- **ES Configure.h**: modify this header file to point to your own event checkers and service functions.
- **Initialize ES** by calling: `ES_Init();`
- **Event-Checking Functions**
  - All EventCheckers must return **TRUE** when a new event has been detected and **FALSE** otherwise
  - EventCheckers post an **ES_Event** type to the appropriate queue
    - `ES_Event.EventType`: an enum with all events listed
    - `ES_Event.EventParam`: a 16 bit parameter to go along with the event
  - **System level events**: `ES_INIT`, `ES_TIMEOUT(#)`, `ES_TIMERSTOPPED(#)`, `ES_TIMERACTIVE(#)`
- **Service Functions**
  - Run to Completion (RTC), non-blocking code
  - Called when a new event shows up in associated event queue
  - Can be a simple service or a state machine
ES – Software Events and Services (4.4)

- **Timer Functions** start, stop, or load a new time into the timer
  - `ES_Timer_InitTimer(#, time)`: Sets the countdown time in ms and starts the countdown for timer #. Posts an `ES_TIMERACTIVE(#)` event.
  - `ES_Timer_SetTimer(#, time)`: Sets the countdown time in ms for timer #, but does NOT start the timer.
  - `ES_Timer_StartTimer(#, time)`: Starts timer #. Posts an `ES_TIMERACTIVE(#)` event.
  - `ES_Timer_StopTimer(#, time)`: Stops the countdown time in ms for timer #. Posts an `ES_TIMERSTOPPED(#)` event.

- **ES_Timer_GetTime()** gets the FreeRunningTimer (1ms ticks)

- **User Timers** system timers 0 is a general purpose user timers
- **User Timer Functions** check if expired, running, or stopped (valid for #0)
  - `IsTimerExpired(#)`: returns TRUE if timer # expired, FALSE otherwise.
  - `IsTimerActive(#)`: returns TRUE if timer # active, FALSE otherwise.
  - `IsTimerStopped(#)`: returns TRUE if timer # stopped, FALSE otherwise.
  - `GetUserTimerState(#)`: returns the ES_EventTyp_t for timer #.
Roach Library (1.2)

- You need to initialize the functions by calling
  `Roach_Init();`

- Functions available for controlling the motors (see documentation for full details):
  `Roach_LeftMtrSpeed(x); Roach_RightMtrSpeed(x);`
  - $x$ is a number from -10 (reverse) to 10 (forward)

- Functions available for checking the bumpers:
  `Roach_ReadFrontLeftBumper(); Roach_ReadFrontRightBumper();`
  `Roach_ReadRearLeftBumper(); Roach_ReadRearRightBumper();`

- Function available for reading the bumpers all at once:
  `unsigned char Roach_ReadBumpers();`
Roach Library (2.2)

- Function available for reading the light level:
  ```c
  unsigned int Roach_LightLevel();
  0 - 1023
  ```

- Function available for reading the battery level:
  ```c
  unsigned int Roach_BatteryVoltage();
  ```

- Functions available for outputting to the LEDs (only for new roaches):
  ```c
  Roach_LEDSSet(x), uint16_t Roach_LEDSGet()
  - x is a pattern of 12 bits
  - NEW_ROACH needs to be defined at the project level
  ```

- Functions available for outputting to the LEDs (only for new roaches):
  ```c
  Roach_BarGraph(x),
  - x is a number between 0 and 12
  ```
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.
Questions?
Announcements

Quiz Today!!

Handouts - Sensor Data Sheet

Movie Prints - After class today.

Lab Door do NOT get propped open.