Event-Driven Programming and State Machines

Gabriel Hugh Elkaim
Traditional Program Structures

start

if/else

if

end

main

while

main/

ISR

everything happens in the ISR

GUT
Asynchronous — any input can happen at any time

Simultaneous — inputs & outputs

Sequence of inputs & outputs are unknown, unknowable, reorderable.

No end, no exit.
Programming Embedded Systems (2.2)

Inputs - Sensors (lots of them)

User Inputs - push buttons, keyboard, comms link

Outputs - update a display, turn something on/off (motors), in general change something physical in the world.

RTOS - Real Time Operating System
Events and Services Framework (1.4)

Formalized methodology – conceptual framework

Excellent method for implementing event driven programs (state machines)

**EMPHASIZE DESIGN FIRST**

Make it clear how to define our lower level entities

Make debugging much easier.
Events and Services Framework (2.4)

Rule #1

All tasks break down into TWO fundamental classes:

1. Event Detection
2. Service

Fast, atomic, non-blocking II.

Which:

```c
! Is Time Expired()
```
What is an Event?

Four detectable changes:

- Open
- Event
- 1-3 mm
- Debouncing

Closed

5-50
What Happens with Noise?
Add Hysteresis

Graph showing hysteresis with thresholds:
- High Threshold
- Signal Plus Noise
- Low Threshold

Graph axes range from 0 to 1 on the x-axis and y-axis.
Correct to date 1

Keep event detectors and service functions as short as possible.

Make them non-blocking.
Complete Embedded Program Structure:

initialize all hardware/software

while (1) {
    Test for Events () — round robin fashion
    Service Events () — single machine
}
Description of an abstract machine

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

Next state in a progression depends only on the current state and the inputs (events).

Idealized instantaneous transition from one state to 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 the code.
- Natural fit into the event-driven software framework.

Gabriel Hugh Elkaim
State Machines (3.4)

INITIAL

STATE

ENTRY/EXIT

ACTION

EVENT

(guard)

ACTION
Dining Philosophers Problem

Harel
1987
Announcements

(1) Be gentle on Reach / Ballin

(2) Groups — don’t join until you have a plan

- Don’t make student sign outside of created area.

(3) Not in this room on Thursday.
State Machines (4.4)

ES_Framework $\rightarrow$ VERY SIMPLIFIED RTOS

email deducers $\rightarrow$ SERVICE
(State Machine)
State Diagram Conventions (1.2)

Tools
- Quantrum Leaps
- "CAFE"
- State Flow

UnC
State Diagram Conventions (2.2)

```plaintext
Nested if-then-else

static enum State
if (state = 5 lockd) {
  if (load = 5 l) {
    state = on中存在的;
  }
  else if (state = 5 ...)
```
Nakib can submit:

```c
static enum state
switch (state) {
  case load:
    switch (cmd) {
      case 1:
        state = on_right;
        break;
      default:
        break;
    }
    break;
}
```
Example: Smart Combination Lock

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

Combination = 1-1-8
ES – Software Events and Services (1.4)

EVENT ↓
Top Level ↓
Bottom → FS_NO EVENT

Post Event
ES – Software Events and Services (2.4)

- **ESConfigure.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(#), ES_ENTRY, ES_EXIT`
- **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 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();
  ```

- 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 format needs to be defined at the LED 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?
Event Detection

Service → Posts an Event

@timzar
LiFePO₄
Introduction to Sensors

Gabriel Hugh Elkaim
Mechantronic Systems

The world

Sensor

World to signal

Digital signal

MC

Decision

Number to world

Actuation
World to Signal

Sensor Transducer \( \rightarrow \) converts one physical quantity into another

Signal Conditioning \( \rightarrow \) need to add this to all of our sensors \((\mu \text{V} \sim \text{mV} \rightarrow \text{V})\)

Examples:
- Thermocouples
- PIR/IR
- OdS light sensors
Sensor Development

Very very very expensive to develop ~ $1 Billion

- Military (cost insensitive)
- Automotive (cost sensitive)
- Medical/Pharma (~ development)
- Applied Physics
- Cell phone