Event-Driven Programming and State Machines

Gabriel Hugh Elkaim
Traditional Program Structures

```c
main()
{
    if (condition) {
        // Code
    }
    while (condition) {
        // Code
    }
}

mainISR
{
    // ISR code
}

GUI

CLI
```
Programming Embedded Systems (1.2)

Asynchronous - any input can happen at any time
Simultaneous - inputs and output
Sequence of inputs and outputs are UNKNOWN, UNKNOWABLE, RE-ORDERABLE

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

Inputs - Sensors: lots of them
User inputs - push button, keypads, communications.

Outputs: - update a display, turn something on or off (motors), in general change something physical

POSIX

RTOS - Real Time Operating System
VxWorks - 40K/year/seat
QNX
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 your low level functions
make debugging easier!!
Events and Services Framework (2.4)

RULE #1:

All tasks break down into ONLY two fundamental classes:

(1) Event Detector
(2) Service

FAST, ATOMIC, NON-BLOCKING!!!

DON'T DO THIS: while (!IsTimerExpired()) { ; }
What is an Event?

Some detectable change.

EVENT

Switch bounce

1-3 msec

125 msec

Debouncing

0 µs

5-50
What Happens with Noise?
Add Hysteresis

![Graph showing hysteresis with On, Off, High Threshold, and Signal Plus Noise regions.](image)
Events and Services Framework (3.4)

**Correlary to RULE #1:**

Keep event detectors and service functions as short as possible.

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

Complete Embedded Program Structure

Initialize all hardware/software
while (1) {
    TestForEvents() -- round robin fashion
    ServiceEvents (state machine)
}
State Machines (1.4)

- 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 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 the code
- Natural fit into the events and services framework
"GERUND" "ING"
State Machines (3.4)

- Initial Transition
- Action
- Event
- Event (guard)
- Event (guard)!
Dining Philosophers Problem

1987

NAREC
ES_Framework → Simplest RTOS

event handlers → SERVICE
(State machine)
State Machines (4.4)

UML

[UMlet]
[Draw.io]
quantum leaps

"Case"

State Flow
State Diagram Conventions (1.2)

Nested if-then-else

static enum state

if (state == 2 locked) {
    if (enable == 1) {
        state = open correct;
    }
    else if (state == ...)
Nested case statement

static enum state

switch (state) {
    case locked:
        switch (event) {
            case 1:
                state = one-right;
                break;
            break;
        }
}
Example: Smart Combination Lock

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

Combination = 1-1-8

- **Open**
- **Close Lock**
- **Timeout**
- **Twist Unlock**
- **3 Right**
- **1 Right**
- **2 Right**
- **8 Right**

- Start Timer

- NA

- !1

- NA

- !1

- NA

- !8

- NA

- 1

- NA

- 1
ES – Software Events and Services (1.4)
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(#), ES_EMPTY, 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
- **Run ES Framework** by calling **ES_Run().** Only returns with an error.
ES – Software Events and Services (3.4)

Entry

- All Queues Empty? (No)
  - Pull event from highest priority non-empty queue
  - Pass that event to the associated service run routine

- All Queues Empty? (Yes)
  - Add System Event to the appropriate queue.

- Any System Events Pending? (Yes)
  - Call User Event Checking Routines. These routines will post to queues when events are detected.

- New Event Detected? (Yes)
  - Move to the next user event checking routine in the list
  - Last Event checker done?

- New Event Detected? (No)
  - Last Event checker done?
    - Yes
      - End flowchart
    - No
      - Go back to Entry

- Any System Events Pending? (No)
  - End flowchart
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 -100(reverse) to 100(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 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?
Introduction to Sensors

Gabriel Hugh Elkaim
Mechantronic Systems

The world

World to signal sensor

Decision maker

MC

Number to world

Actuator

Hands
World to Signal

Sensor $\rightarrow$ Transducer $\rightarrow$ convert an physical phenomenon into another.

Signal Conditioning $\rightarrow$ used to add this to all of our sensors. Most simple (unamplified)

Example:

- Thermal
- Pin/or
- CdS light sensor

$\mu V \rightarrow V$