Embedded Systems and IoT

Ingegneria e Scienze Informatiche - UNIBO
a.a 2022/2023
Lecturer: Prof. Alessandro Ricci

[module-2.4] EVENT-BASED ARCHITECTURES

FROM INTERRUPTS TO EVENT-BASED ARCHITECTURES

- The interrupt mechanism can be exploited to design high-level event-based architectures
  - interrupts per se can be conceived as low-level events interrupting the control flow of the super-loop to execute the corresponding interrupt (event) handlers
  - interrupts make it possible to avoid polling in managing sensors and input
    - it is not the processor that continuously read the status of the sensor, but conceptually the sensor which notifies changes as soon as they occur
- High-level event-based architectures
  - **observer pattern**
  - **asynchronous FSM**

OBSERVER PATTERN

- Elements
  - **source** or *generator* of events
    - it provides an interface/API for observers (listers) to register/ subscribe in order to be notified when an event occurs
    - e.g. a tactile button as a source
  - **events** *generated*
    - e.g. *button pressed* event
  - **observer** (or listener) of a source
    - it provides an interface/API to be notified of the events
    - e.g. buttonPressed method in an interface
- Designing an implementation using interrupts
  - used inside source components, bound to events that can occur
  - the interrupt handler calls the listeners registered on the source component

OBSERVER PATTERN: EXAMPLE

- Example available on the repo (module-lab-2.4/pattern-observer-example)
  - tactile button as source
- Parts
  - Observer pattern library
    - abstract class **EventSource**
      - represents a generic base event source, generating events using interrupts
    - class InterruptDispatcher
      - functions as a bridge between interrupt handlers and the event source
  - Event-based Tactile Button component
    - Button interface
    - AbstractButton abstract class
      - base class for event-based button
    - ButtonImpl implementation

REMARKS

- ***The listener codes is executed by interrupt handlers***, executing in interrupt context
  - this could be a strong limitation (as discussed in previous modules)
    - listeners cannot contain long-term computations or use functions/procedures that need interrupts to be enabled
- To avoid race conditions, it is necessary to make the super-loop code accessing variables modified by listeners atomic
  - by disabling and re-enabling interrupts

ASYNCHRONOUS FSMs

- Exploiting interrupts to design and implements event-triggered FSMs (as introduced in module-2.2)
  - differently from synchronous FSMs, the evaluation of reactions is done when an event occurs, generated by interrupt
  - there is not the notion of period, like in synchronous FSMs
  - we can eventually model/simulate a periodic behaviour by means of temporal events generated by a timer

ASYNCHRONOUS FSMs: AN EXAMPLE

- Example in the repo (module-lab-2.4/button-led-async-fsm)
  - implementation of a button-led system based on async FSMs
- Parts
  - Library async-fsm
    - classes: **Event**, **EventSource**, **Observer**
      - representing events generated by interrupts, event sources and event observers
    - InterruptDispatcher class
      - functions as a bridge between interrupt handlers and event sources
    - abstract class **AsyncFSM**
      - base class for implementing specific async FSMs
      - functions as observer of events generated by event source components
      - uses an event-queue to keep track of events
  - Tactile button
    - event source

REMARKS

- In this case there is ***full uncoupling between event generation (done by interrupts) and reactions & actions (executed by the super loop control flow)***
  - we don't have the limitations due to the execution of listeners in interrupt handlers
    - nevertheless, we fully preserve the reactivity/ responsiveness of the system
  - race conditions are avoided by construction, since the interrupt handlers do not execute codes accessing variables
- The design/implementation is based on the *Reactor pattern*, also called ***event-loop architecture***