CITS4401 Software Requirements and Design
Event-driven systems

Lecturer: Arran Stewart

Event-driven systems

Event-driven systems

Typical flow of control for a command-line application:

1. Start
2. Read input
3. Perform computation
4. Write output
5. Exit

Event-driven systems

Many systems today (e.g. GUIs, mobile apps, web apps) do not follow a "procedural" flow of control like this, but are instead event-driven.

Event-driven flow of control

1. Start
2. Begin event loop
3. Has a new event been generated?
   (e.g. a mouse click)
   If yes:
   - identify event handlers
   - dispatch event to them

   If event was “terminate”:
   - Goto step 4.

   Else:
   - repeat step 3.
4. Tidy up and exit

Events

- An *event* is just a piece of data (usually an object, in OO systems) that acts as a *signal* to another object (known as a *listener*)
- When events are sent, we usually refer to this as "emiting" or "firing" the event.

Listeners

A listener object performs specified actions in response to an event.

- Multiple listeners can respond to one event
- One listener can respond to multiple sorts of events
- Typically, a listener object will tell the framework that it is "subscribing" to listen for some particular set of events.

The observer pattern

- This may remind you of the Observer pattern.
- But note that listeners do not directly observe the objects that generate events.
- Rather, the listeners observe the event framework, and the event framework observes the objects that generates events.
- Generators and listeners are somewhat *decoupled* from each other. This makes it easier to change system behaviour in the future.

Event-driven systems

- In a typical command-line program, the application has control at all stages of execution.
- The user may interact with the system, at pre-programmed times.
- In a typical event-driven system:

  After initialisation of the system (e.g. creating GUI objects), control is passed to the event framework.
- The system waits for the user actions and other events, then calls methods of appropriate objects to *handle* those events.

Event-driven systems

Besides GUIs, examples of event-driven systems are:

- Network servers
  - these wait for connections or other events
- Stream parsers
  - these parse some type of document, and the "events" are encountering particular elements in the document
  - e.g. "When you see a 'Heading3' element, call this particular method to handle it."

Event-driven programming

- Event-driven programming introduces a different sort of flow of control to what you may be used to.
- *Events* can occur at unpredictable (rather than pre-specified) times.
- The control structure of event-driven programs can be thought of as being "inverted": the event framework (or operating system) has control, not the application.

Event-driven programming

- Typically, the system will wait for an event (e.g. a user action, a network connection, some length of time elapsing), then call the appropriate method in your program.
- In a GUI program, classes called *widgets* represent items displayed on-screen, that also can detect a user's input and emit events.
- The parts of the program that respond to the events are called *handlers*.

Events can represent things like:

- user input e.g. a key press
- internal events e.g. timeouts
- a signal from sensors
- hardware being attached or removed

More on events in the workshop.

Event-driven architecture

We will look more at architecture soon – it describes the high-level design of a system.

A typical architecure for event-driven systems is the “Observe and React” architecture.

Observe and React architecture

- Description (Sommerville): The input values of a set of sensors of the same types are collected and analyzed. These values are displayed in some way. If the sensor values indicate that some exceptional condition has arisen, then actions are initiated to draw the operator's attention to that value and, in certain cases, to take actions in response to the exceptional value.
- Stimuli: Values from sensors attached to the system"
- Responses: Outputs to display, alarm triggers, signals to reacting systems
- Used in: Monitoring systems, alarm systems

Observe and React process structure