3C03 Concurrency: Concurrent Architectures - Announcer/Listener

Mark Handley

Outline

- Motivation
- Announcer-Listener
- Announcer-Listener Model
- Announcer-Listener Safety and Progress
- Announcer-Listener Implementation
- Summary
Motivation

- Notification of events:
  - Events originate in one location (announcer)
  - Communicated to multiple interested parties (listeners)
- Announcer does not know listeners.
- Listeners do not know announcer.
- Communication is managed by connector called event manager.

Announcer-Listener Architecture
Application Examples

- **User Interface Frameworks:**
  - AWT Listeners are ordinary objects.
  - Events are mouse clicks, button presses.
  - Events cause operations to be invoked in Listeners.

- **CORBA Event Service:**
  - Event Producers are Announcers
  - Event Channels are Event Managers
  - Event Consumers are Listeners
  - Used, for example in distributed stock tickers.

Filtering & Recursive Events

- **Listeners may only be interested in a subset of events**
  - They register with Event Manager using a “pattern” of events they wish to receive.

- **Listeners may themselves be announcers and forward events into subsequent Event Managers.**

- **Listeners do not have to be active processes.**
**Event Manager Model**

set Listeners = {a, b, c, d}
set Pattern = {pat1, pat2}

REGISTER = IDLE,
IDLE = ( register[p:Pattern] -> MATCH[p]
| announce[Pattern] -> IDLE),
MATCH[p:Pattern] = (announce[a:Pattern] ->
  if (a==p) then
    ( event[a] -> MATCH[p]
    | deregister -> IDLE)
  else
    MATCH[p]
| deregister -> IDLE).

||EVENTMGR = (Listeners:REGISTER)
  /{announce/Listeners.announce}.

**Announcer-Listener Model**

ANNOUNCER = (announce[Pattern] -> ANNOUNCER).

LISTENER(P='pattern) = (register[P] -> LISTENING),
LISTENING={ compute -> LISTENING
| event[P] -> LISTENING
| event[P] -> deregister -> STOP
}+(register[Pattern]).

||ANNOUNCER_LISTENER=( a:LISTENER('pat1)
| b:LISTENER('pat1)
| c:LISTENER('pat2)
| d:LISTENER('pat2)
| EVENTMGR
| ANNOUNCER).
Announcer-Listener Analysis

■ Safety-Properties:
  • Listeners receive events then and only then when they are registered for them
  • Listeners only receive events that match the patterns they have registered for

■ Progress-Properties
  • Announcer should be able to announce events independent of state of Listeners

Safety Properties:
- Listener only receives events when it’s registered.
- Listener only receives the events for which it registered.

property

SAFE = (register[p:Pattern] -> SAFE[p]),
SAFE[p:Pattern] = ( event[p] -> SAFE[p]
| deregister -> SAFE )

Progress Properties:
- The announcer can announce, no matter who is listening.

progress ANNOUNCE[p:Pattern] = {announce[p]}
Announcer-Listener Example

Simple game:
- Coloured Blocks are moving around on a canvas
- Hit them with a mouse press
- A hit block turns black and stops moving
- Blocks are threads that listen for mouse events
- Events are announced by the display canvas
- Events are generated by the AWT classes for Event handling

Announcer Listener Design

Diagram showing the relationship between Applet, EventDemo, Thread, BoxMover, BoxCanvas, MouseEvent, and MouseListener.
Summary

- Announcer-Listener
- Applications for Announcer-Listener
- Announcer-Listener Model
- Announcer-Listener Safety and Progress
- Announcer-Listener Implementation
- Summary