



# ***C340 Concurrency: Concurrent Architectures - Announcer/Listener***

***Wolfgang Emmerich***



## ***Outline***

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

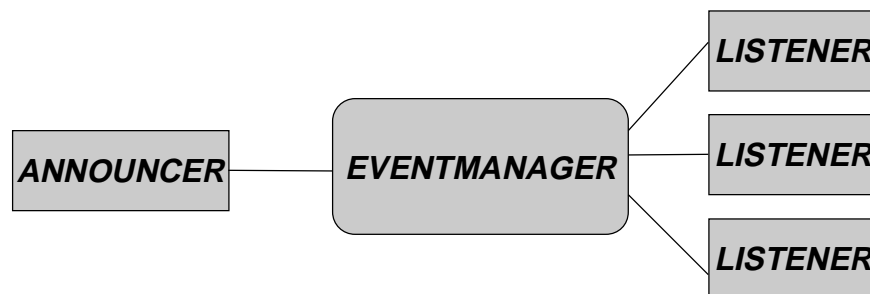


## Motivation

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



## Announcer-Listener Architecture





## ***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***



## ***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***



## 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}.
```

© Wolfgang Emmerich, 1998/99

LTSA 7



## Announcer-Listener Model

```
ANNOUNCER = (announce[Pattern]->ANNOUNCER).
LISTENER(P='pattern)=(register[P]0>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)
                       | |EVENTMANAGER
                       | |ANNOUNCER).
```

© Wolfgang Emmerich, 1998/99

8



## 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*



## Announcer-Listener Analysis

### ■ **Safety Properties:**

property

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

### ■ **Progress Properties:**

```
progress ANNOUNCE[p:Pattern]={announce[p]}
```





## *Summary*

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