Reactor pattern

T-106.420 Concurrent programming

Jan Lönnberg

Department of Computer Science
Helsinki University of Technology

15th November 2005

Reactor pattern

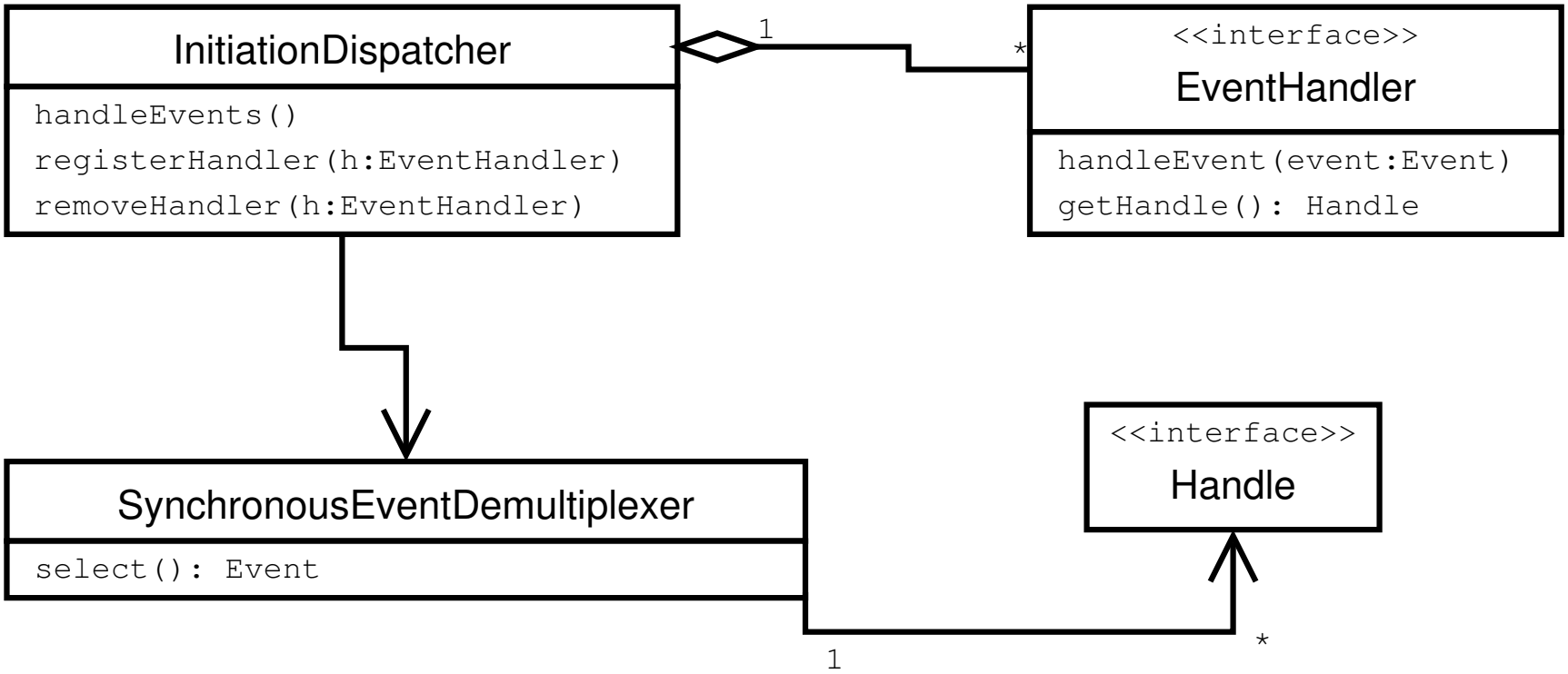
Purpose

- Receive incoming messages or requests from multiple concurrent clients.
- Process these messages using event handlers one at a time.
- Useful in e.g. servers, graphics systems, component frameworks.

Advantages

- Serialisation of events $\Rightarrow$ simpler concurrency in application.
- Modularity/portability improved.

Structure


InitiationDispatcher
handleEvents()
registerHandler(h:EventHandler)
removeHandler(h:EventHandler)
1
*
<<interface>>
EventHandler
handleEvent(event:Event)
getHandle(): Handle
SynchronousEventDemultiplexer
select(): Event
<<interface>>
Handle
1
*

Structure

Handle

- Receives events.
- E.g. a network connection, timer, user interface device, pipe.

Synchronous Event Demultiplexer

- `select()` waits until an event is received on a Handle and returns the event.
- Often implemented as part of an operating system.

Structure

Initiation Dispatcher

- Uses the Synchronous Event Demultiplexer to wait for events.
- Dispatches events to the Event Handlers.

Event Handler

- Application-specific event processing code.

Dynamics

Setup

- Create Initiation Dispatcher.
- Register Event Handlers with Initiation Dispatcher.

Main loop

- Call `handleEvents` in Initiation Dispatcher repeatedly.
- Initiation Dispatcher calls `select` in Synchronous Event Demultiplexer, blocking until an event is received.
- The Initiation Dispatcher calls `handleEvent` in the corresponding Event Handler, passing it the event.

End

- Unregister Event Handlers from Initiation Dispatcher.