Proactor Pattern

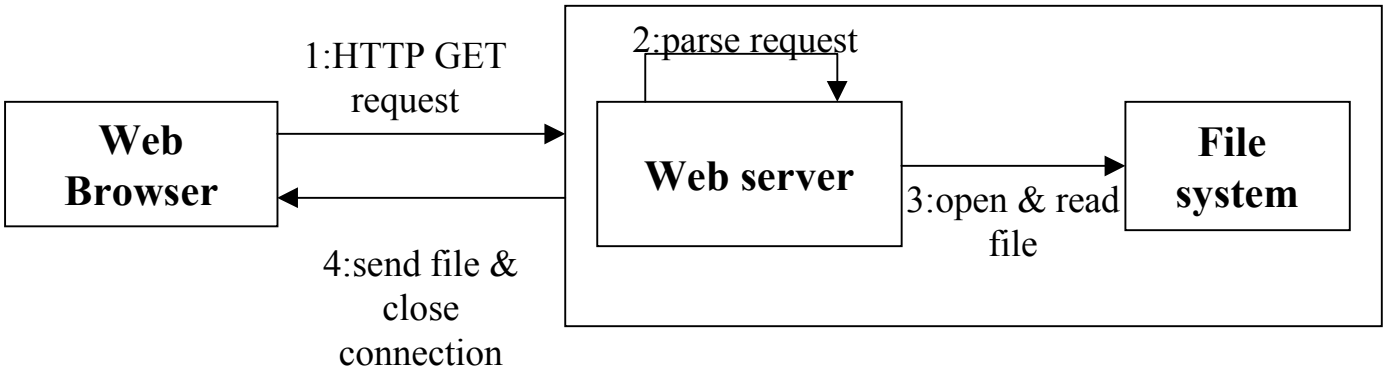
Tik-109.450 Object-Oriented Protocol Engineering

Chengyuan Peng
pcy@tml.hut.fi

- Functions of Proactor Pattern
  - Allows event-driven applications to efficiently demultiplex and dispatch service requests triggered by the completion of asynchronous operations, to achieve the performance benefits of concurrency without incurring certain of its liabilities
- Context:
  - An ***event-driven application*** that receives and processes multiple service requests asynchronously.

An Example

- A high-performance Web server that processes HTTP requests sent from multiple remote Web browsers simultaneously.

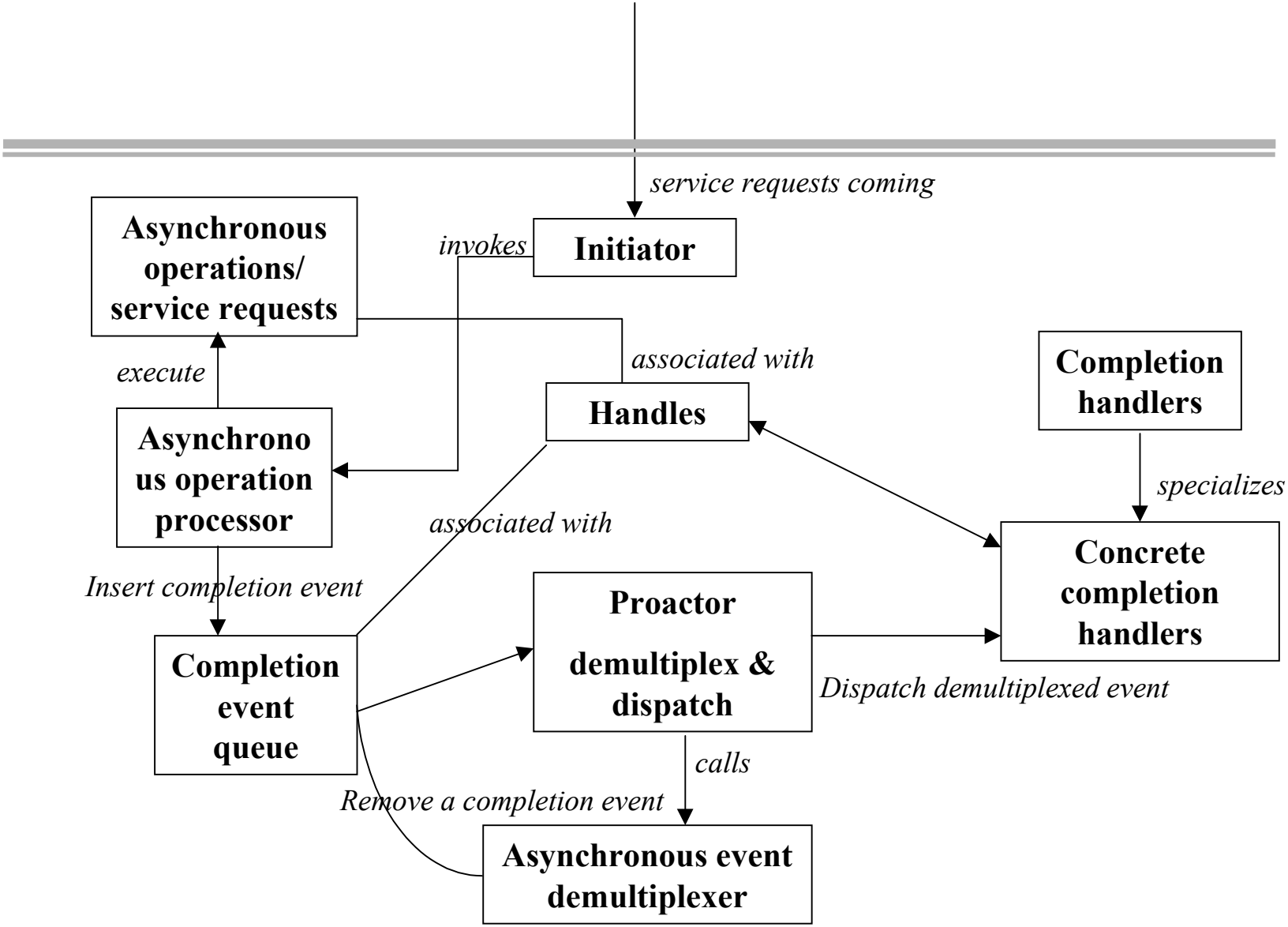


A high-performance Web server

- Problem:
  - The ***performance*** of event-driven applications, particularly servers, in a distributed system can often be improved by processing multiple service requests ***asynchronously***.
  - When asynchronous service processing completes, the application must handle the corresponding ***completion events*** delivered by the operating system to indicate the end of the asynchronous computations.

- Forces.
  - ***To improve scalability and latency***, an application should process multiple completion events simultaneously without allowing long-duration operations to delay other operation processing unduly.
  - ***To maximize throughput***, any unnecessary context switching, synchronization, and data movement among CPUs should be avoided.
  - ***Integrating*** new or improved services with existing completion event demultiplexing and dispatching mechanism should require minimal effort.
  - ***Application code*** should largely be shielded from the complexity of multi-threading and synchronization mechanisms.

- Solution:
  - ***Split*** application services into two parts: ***long-duration operations*** that execute asynchronously;***completion handlers*** that process the results of these operations when they finish:
  - ***Integrate*** the ***demultiplexing*** of completion events, which are delivered when asynchronous operations finish, with their ***dispatch*** to the completion handlers that process them.
  - ***Decouple*** completion event demultiplexing and dispatching mechanisms from the ***application-specific*** processing of completion events within completion handlers.



The chain of proactor pattern activities

- Participants
  - An **initiator:**
    - an entity local to an application
    - ***invokes*** asynchronous operations on an asynchronous operation processor.
  - **Asynchronous operations:**
    - ***Defines*** long-duration operation that can be executed ***asynchronously***
    - Used to implement a service

- Handles:
  - ***identify*** operating system resources that can be the target of asynchronous operation invocations or a source of completion events
- Asynchronous operation processor:
  - ***implemented*** by an operating system kernel.
  - ***executes*** Asynchronous operations.
  - ***generates*** the corresponding completion event
  - ***inserts*** this event into the completion event queue when an Asynchronous operation finishes executing

- Completion event queue
  - ***buffers*** completion events while they are waiting to be removed by an asynchronous event demultiplexer to their associated completion handler.
- Asynchronous event demultiplexer:
  - ***waits*** for completion events to be inserted into a completion event queue when an asynchronous operation has finished executing
  - then ***removes*** one or more completion event results from the queue and returns to its caller.

- A proactor:
  - provides an ***event loop*** for an application process or thread
  - In this loop, a proactor ***calls*** an asynchronous event demultiplexer to ***dequeue*** a completion event
  - then ***demultiplexes and dispatches*** the completion events to its associated completion handler
- A completion handler:
  - defines an ***interface*** for processing results of asynchronous operations.

– Concrete completion handlers:
  - ***process*** results of asynchronous operations in an application-specific manner.
  - ***specializes*** the completion handler to define a particular application service by implementing the inherited methods
  - ***associated*** with a handle that it can use to invoke asynchronous operations itself.
  - Potentially ***invokes*** additional asynchronous operations

- Implementation:
  - the nine participants be decomposed into two layers
    - Demultiplexing/dispatching infrastructure layer
      - ***performs*** generic, application-independent strategies for executing asynchronous operations
      - ***demultiplexes and dispatches*** completion events from these asynchronous operations to their associated completion handlers
    - Application layer
      - ***defines*** asynchronous operations and concrete completion handlers that perform application-specific service processing

- Benifits:
  - Separation of concerns:
  - Portability:
  - Encapsulation of concurrency mechanisms:
  - Decoupling of threading from concurrency:
  - Performance:
  - Simplification of application synchronization

- Liabilities:
  - **Restricted applicability:**
    - can be applied most efficiently if the OS supports asynchronous operations ***natively***.
    - If the OS does not provide this support, it is possible to emulate the semantics of the Proactor pattern using ***multiple threads*** within the proactor implementation.
  - **Complexity of programmimg, debugging, and testing:**
    - ***hard*** to program applications and high-level system services using asyncronous mechanisms, due to the ***separation*** in time and space between operation invocation and completion.

- ***hard*** to debug and test because the inverted flow of control oscillates between the proactive framework infrastructure and the method callbacks on application-specific handlers.

– **Scheduling, controlling, and canceling asynchronously running operations**

- ***Initiators*** may be unable to control the scheduling order in which asynchronous operations are executed by an asynchronous operation processor.

- Known Uses:
  - Completion ports in Windows NT
  - The POSIX AIO family of asynchronos I/O operations
  - ACE Proactor Framework
  - Operating system device driver interrupt-handling mechanisms
  - Phone call initiation via voice mail