Event-Driven Architecture

Software Architecture

Richard Thomas

March 31, 2025

Definition 0. Event

Something that has happened or needs to happen.

Definition 0. Event Handling

Responding to notification of an event.

Definition 0. Asynchronous Communication

Sending a message to a receiver and not waiting for a response.

Comment on how this enables parallel processing.

Responsiveness

- Synchronous Communication
  - Send message
  - *Wait* for response
  - Continue processing

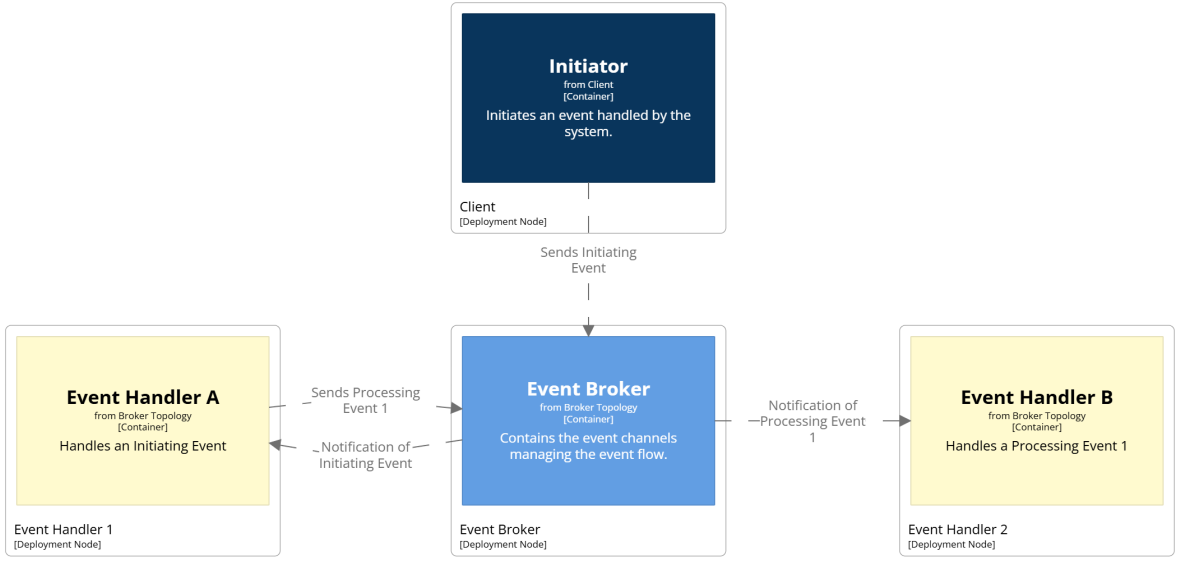
Responsiveness

- Synchronous Communication 
  - Send message
  - *Wait* for response
  - Continue processing
- Asynchronous Communication 
  - Send message
  - Continue processing
  - *Optionally* receive response
  - *Complex* error handling 

Definition 0. Event-Driven Architecture

Asynchronous distributed system that uses event processing to *coordinate* actions in a larger business process.

Event-Driven Architecture



Comment on how each container is deployed in its own compute node.

Terminology

Initiating Event Starts the business process

Terminology

Initiating Event	Starts the business process
Processing Event	Indicates next step in the process can be performed

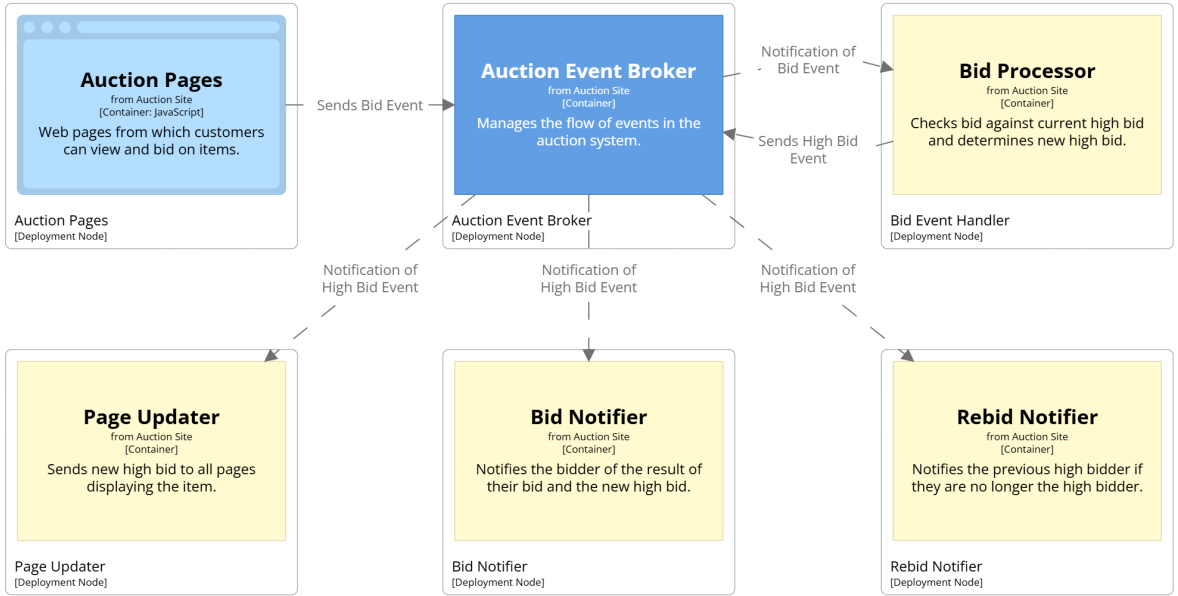
Terminology

Initiating Event	Starts the business process
Processing Event	Indicates next step in the process can be performed
Event Channel	Holds events waiting to be processed

Terminology

Initiating Event	Starts the business process
Processing Event	Indicates next step in the process can be performed
Event Channel	Holds events waiting to be processed
Event Handler	Processes an event • Step, or part of a step, in the business process

Auction Example



- Auction Event Broker has an API Gateway component to receive client requests and components to manage the event channels.
- Step through event process.
- Highlight asynchronous messages and parallel processing.
- Bid Processor could send back a high bid event or an async message.

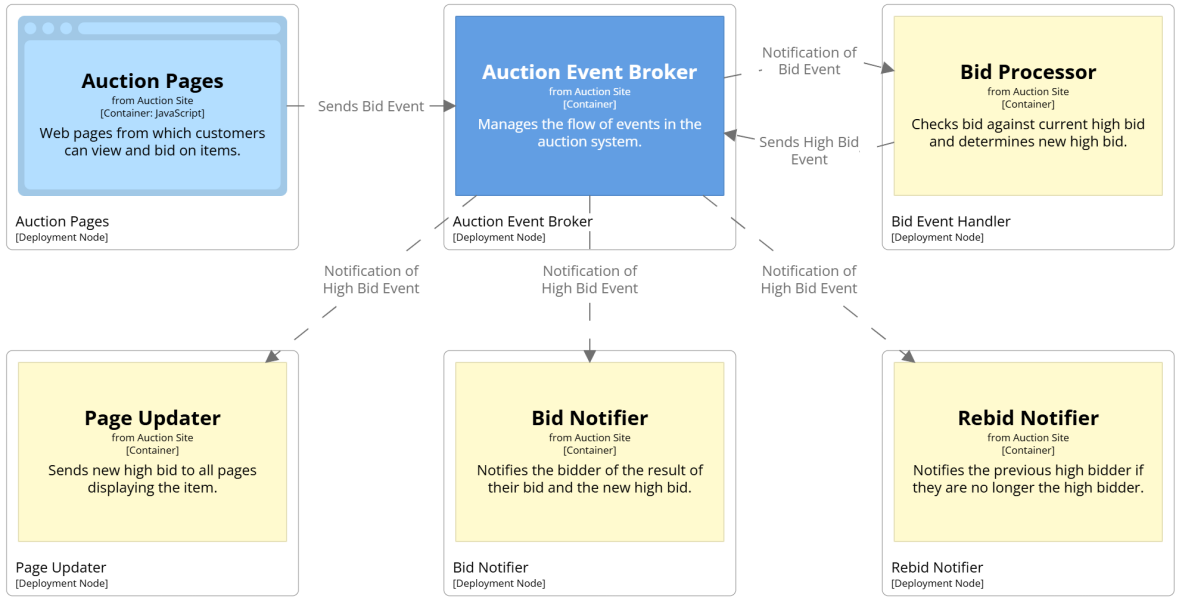
Definition 0. Event Handler Cohesion Principle

Each event handler is a simple cohesive unit that performs a *single* processing task.

Definition 0. Event Handler Independence Principle

Event handlers should not depend on the *implementation* of any other event handler.

Auction Example – Error Handling



- Ask:
- How to handle Bid Processor failing?
  - Need to restart & recover
- How to handle Rebid Notifier failing?
  - Need to restart – Could losing events be acceptable?
- How to handle Event Broker failing?
  - Need to restart & recover – without losing events

Topologies

Broker All events received by event broker

- Notifies event handlers of events
- Event handlers send processing events when they finish processing

Topologies

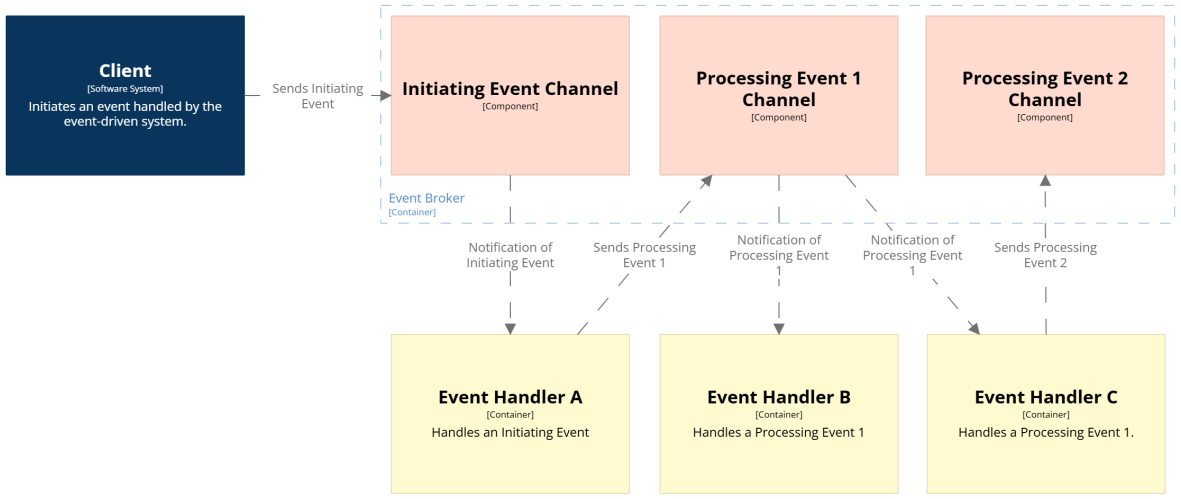
Broker
All events received by event broker

- Notifies event handlers of events
- Event handlers send processing events when they finish processing

Mediator
Manages business process

- Event queue of initiating events
- Event mediator sends processing events to event handlers
- Event handlers send async messages to mediator to report process finished

Broker Topology

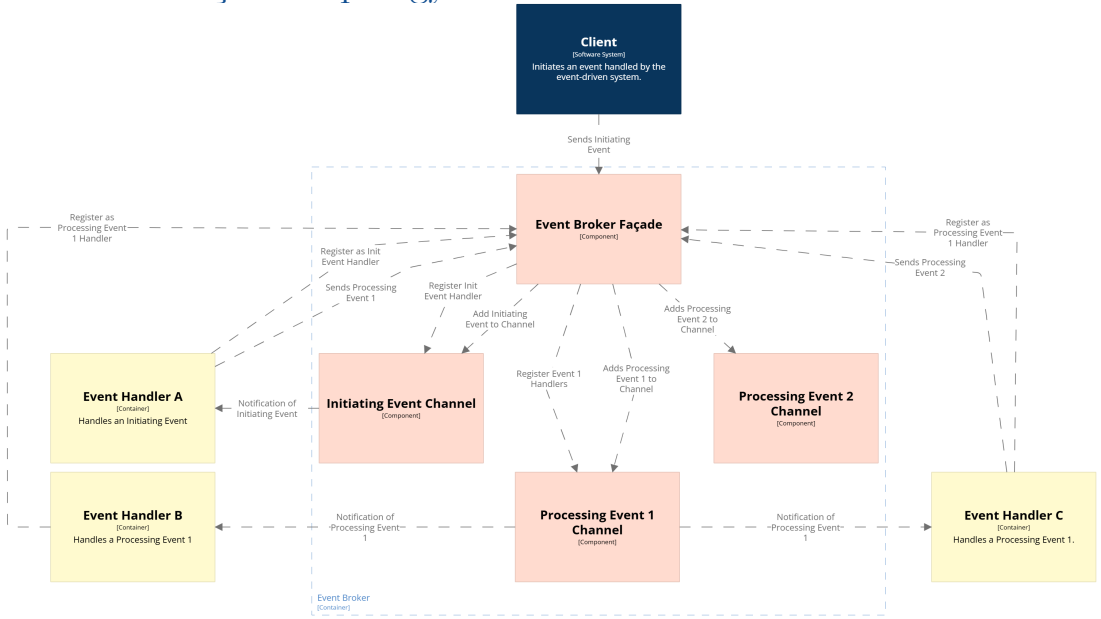


- Step through event process
- Channels facilitate message flow
  - Commonly a lightweight message broker (e.g. RabbitMQ, ...)
- Send final processing event, even if it is not handled
  - Easier to *extend* in the future

Event Broker Façade

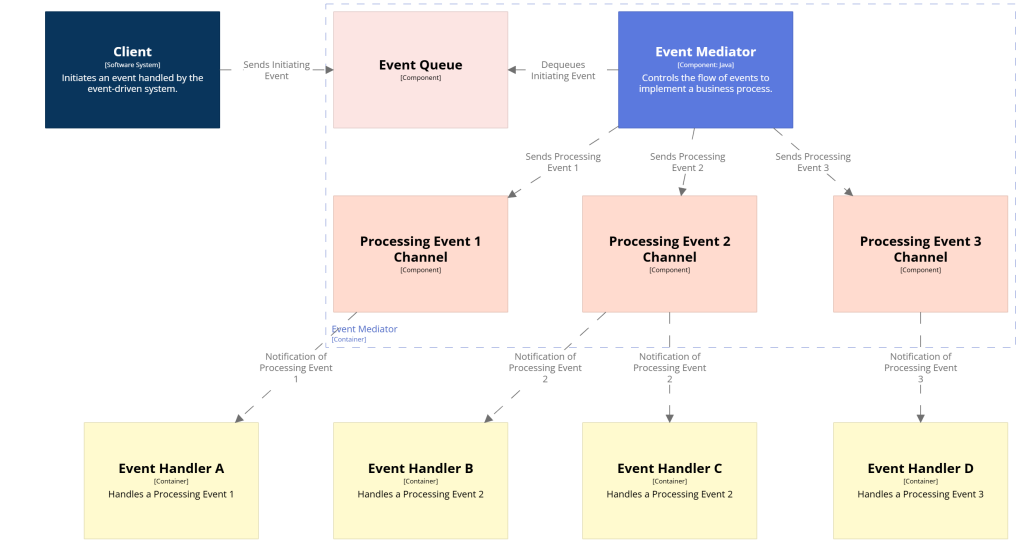
- Event handlers register to *listen* for events
- Receives events and *directs* them to the correct channel

Broker with Façade Topology



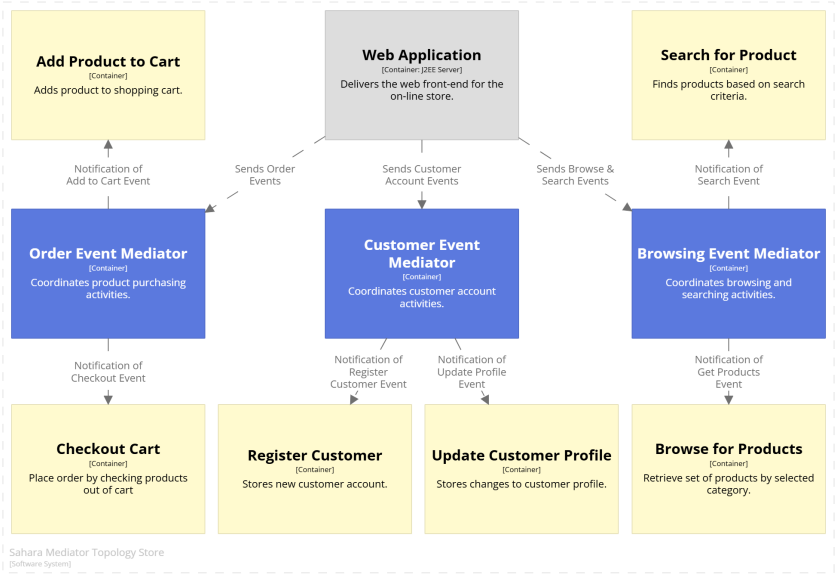
- Event processing & event handling are the same
- Event Handlers register to listen for events, rather than being connected directly to Channels
  - Additional layer of abstraction
- Step through event process

Mediator Topology



- Step through event process.
- Highlight process control performed by mediator.

Sahara Mediator Topology



- Step through event process.
- Multiple mediators is common – *one* per domain.
- Discuss internals of mediators: event queue and event channels.

Extensibility

- New behaviour for existing event
  - Broker: Implement event handler & register with broker
    - Existing ignored event hooks
  - Mediator: Implement event handler & modify mediator logic

Extensibility

- New behaviour for existing event
  - Broker: Implement event handler & register with broker
    - Existing ignored event hooks
  - Mediator: Implement event handler & modify mediator logic
- New event
  - Broker: Implement event & event handler, create event channel, modify broker façade
  - Mediator: Implement event & event handler, modify mediator logic

Scalability

- Event handlers deployed independently
  - Scaled independently to manage load

Scalability

- Event handlers deployed independently
  - Scaled independently to manage load
- Event broker federated
  - Distributed across multiple compute nodes

Scalability

- Event handlers deployed independently
  - Scaled independently to manage load
- Event broker federated
  - Distributed across multiple compute nodes
- Event mediators for different domains
  - Distributes loads by domain
    (e.g. browse & search, account, & order events)
    - Scaled independently to manage load

Queues

- Channels can be implemented as queues
  - FIFO behaviour

Queues

- Channels can be implemented as queues
  - FIFO behaviour
- Multiple front of queue pointers
  - For each event handler

Queues

- Channels can be implemented as queues
  - FIFO behaviour
- Multiple front of queue pointers
  - For each event handler
- Event removed when event handlers finish
  - Retry if a handler fails

Queues

- Channels can be implemented as queues
  - FIFO behaviour
- Multiple front of queue pointers
  - For each event handler
- Event removed when event handlers finish
  - Retry if a handler fails
- Events persist until removed
  - Recovery from broker failure

Streams

- Channels can be implemented as streams
  - Events are saved permanently

Streams

- Channels can be implemented as streams
  - Events are saved permanently
- Handlers notified when event added to stream
  - Observer pattern

Streams

- Channels can be implemented as streams
  - Events are saved permanently
- Handlers notified when event added to stream
  - Observer pattern
- Handlers process events at their own pace
  - Cardiac arrest alarm vs. heart rate graph

Streams

- Channels can be implemented as streams
  - Events are saved permanently
- Handlers notified when event added to stream
  - Observer pattern
- Handlers process events at their own pace
  - Cardiac arrest alarm vs. heart rate graph
- Events history
  - Redo processing
  - Review processing activities

Queues vs. Streams

- Queue
  - Known steps in business process
  - Easier sequencing of steps in business process
  - “Exactly once” semantics
  - eCommerce system

Queues vs. Streams

- Queue
  - Known steps in business process
  - Easier sequencing of steps in business process
  - “Exactly once” semantics
  - eCommerce system
- Stream
  - Very large number of events or handlers
  - Handlers can ignore events
  - Analysis of past activity
  - Event sourcing

Broker dumb pipe

Broker events have occurred

Broker vs. Mediator Topologies

Broker dumb pipe

Broker events have occurred

Mediator smart pipe

Mediator events are commands to process

Broker Advantages

- Scalability
- Reliability
- Extensibility
- Low coupling

Broker vs. Mediator Topologies

Broker Advantages

- Scalability
- Reliability
- Extensibility
- Low coupling

Mediator Advantages

- Complex business process logic
- Error handling
- Maintain process state
- Error recovery

Emphasise the *real* advantage of Broker is low coupling and slightly easier extensibility.

Pros & Cons

Modularity Event Handlers 

Extensibility 

Reliability Event Handlers 

Interoperability Events 

Scalability Event Handlers 

Security 

Simplicity 

Deployability 

Testability Complex Interactions 

- Broker & Mediator are both *very* scalable
- Due to simple event management through message queues
  - Broker can handle a slightly higher load
  - Broker is slightly easier to scale
- Mediator has more internal processing, so requires greater resources