Event-Driven Architecture

April 4, 2022

Richard Thomas

Presented for the Software Architecture course
at the University of Queensland

THE UNIVERSITY
OF QUEENSLAND
AUSTRALIA

Event-Driven Architecture

Software Architecture

April 4, 2022

Richard Thomas

1 Introduction

Event-driven is an asynchronous architectural style. It reacts to events, which is different to the common view of many designs which respond to requests.

Events provide a mechanism to manage asynchronous communication. An event is sent to be handled, and the sender can continue with other tasks while the event is being processed. If necessary, the handler can send a message back to the sender indicating the result of the event processing.

Each event handler can be implemented in its own independent execution environment. This allows each type of handler to be easily scaled to handle its load. Asynchronous communication means that event generators do not need to wait for the handler to process the event. Handlers can generate their own events to indicate what they have done. These events can be used to coordinate steps in a complex business process.

The service-based architecture example for the Sahara eCommerce system is designed to perform synchronous processing of requests. A customer *requests* to view a product's details through a REST API and receives a response with the result. Similarly, a customer adding a product to their shopping cart is another request.

Adding auctions to the Sahara eCommerce system is a simple example that benefits from an event-driven architecture. A customer sends a message to the Sahara eCommerce system initiating the *event* of making a bid. Upon receiving the bid event, the system checks the bid against the current high bid and determines the new high bid. The system then generates a new high bid event. This event triggers actions in the system to update the high bid and notify the bidder of the result of their bid. It may also trigger an action to notify the previous high bidder that they are no longer the high bidder.

Multiple customers could submit bids for the same item at almost the same time (e.g. bid sniping). These bid events are queued to be processed in the order they are received. This queuing mechanism allows event handlers to process messages that arrive faster than the handler can process them. It assumes that the message load will reduce in time and that the event handler will process all messages in the queue.

There are two basic approaches to implementing an event-driven architecture. The terminology is that these are different *topologies*, as they have different high-level structures. The *broker topology* is the simpler of the two and is optimised for performance, responsiveness, scalability, extensibility, and low coupling. The *mediator topology* is more complex but is designed to provide reliability, process control, and error handling.

2 Broker Topology

The broker topology consists of four elements.

Initiating Event starts the flow of events.

Event Broker has *channels* that receive events waiting to be handled.

Event Handler accepts and processes events.

Processing Events sent by an event handler when it has finished processing an event.

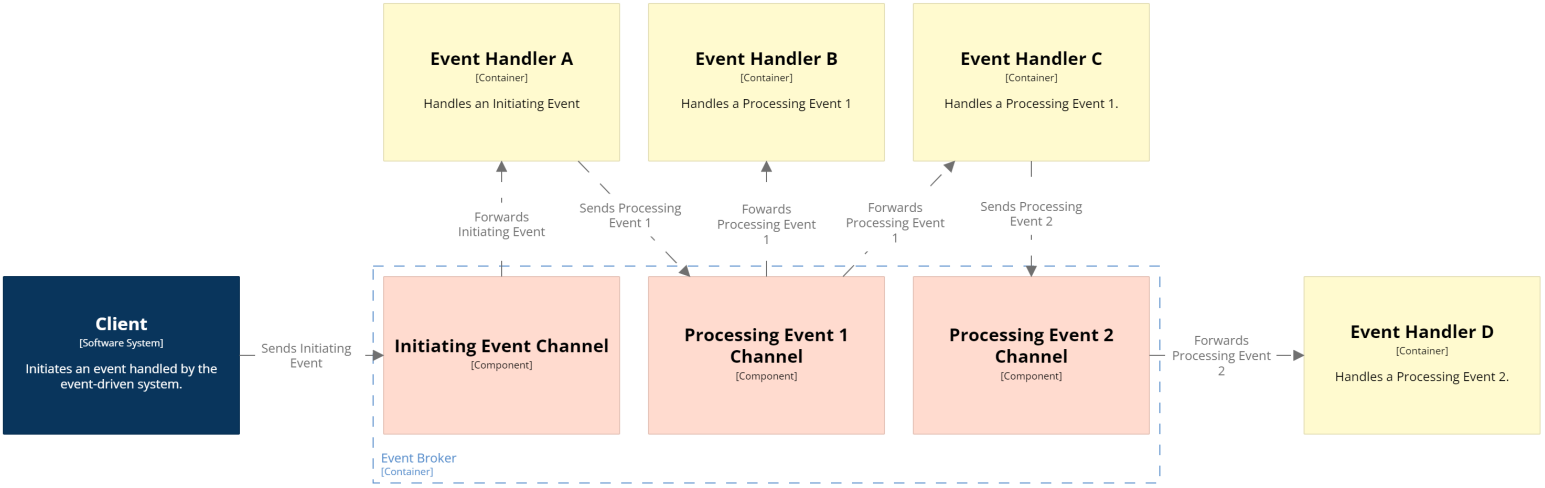

Figure 1: Basic broker topology.

In figure 1, the *Client* sends the *Initiating Event* to the *Initiating Event Channel* in the *Event Broker*. *Event Handler A* accepts the event and processes it. Upon completion of handling the *Initiating Event*, *Event Handler A* sends *Processing Event 1* to the appropriate channel in the event broker. Event handlers B and C accept this processing event and perform their actions. When event handler C finishes processing it sends Processing Event 2 to its channel. Event handler D accepts this event and performs its processing.

3 Design Considerations

A service-based architecture is typically used for medium-sized systems.

4 Service-Based Principles

There are a couple of principles which should be maintained when designing a service-based architecture to produce a simple, maintainable, deployable and modular designs.

Definition 1. Independent Service Principle

Services should be independent, with no dependencies on other services.

5 Extensions

There are a few common variations of the service-based architecture to consider.

5.1 Separate Databases

The first variation we will consider is to have separate databases for each service.

6 Conclusion

Service-based architecture is an approach to designing a distributed system that is not too complex. Domain services provide natural modularity and deployability characteristics in the architecture design. Well

designed service APIs improve the encapsulation and hide implementation details of the services.