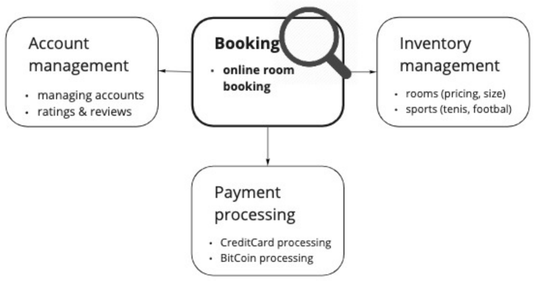


Demo

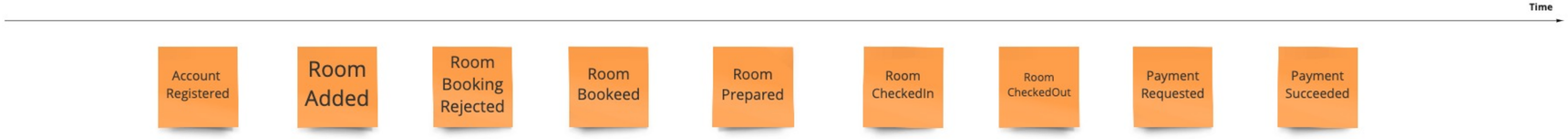
https://github.com/AxonIQ/axoniq-hotel

Event model

"What a system is supposed to do from start to finish, on a time line and with no branching"

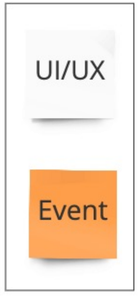
1 moving part

Step 1: "A plausible story made of events"

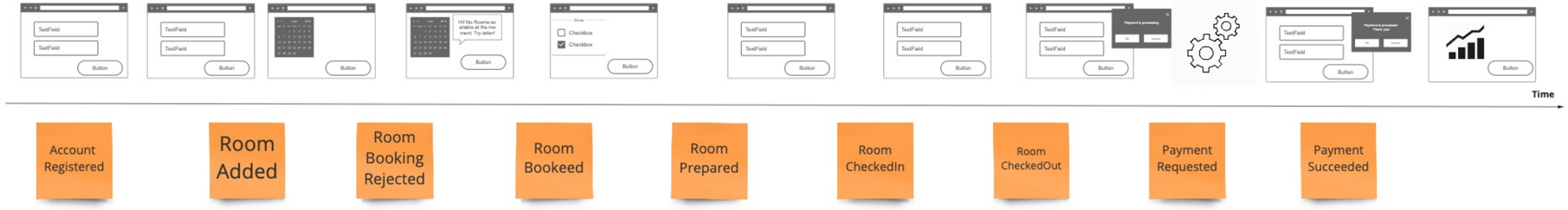


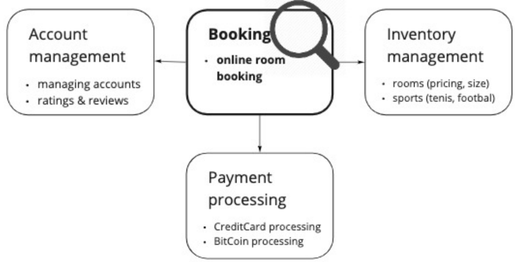
---

2 moving parts



Step 2: "Adding wireframes to address visual learners."





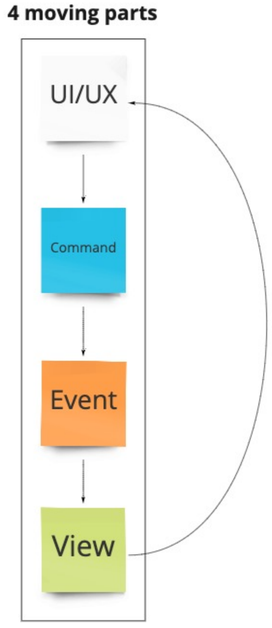
Demo

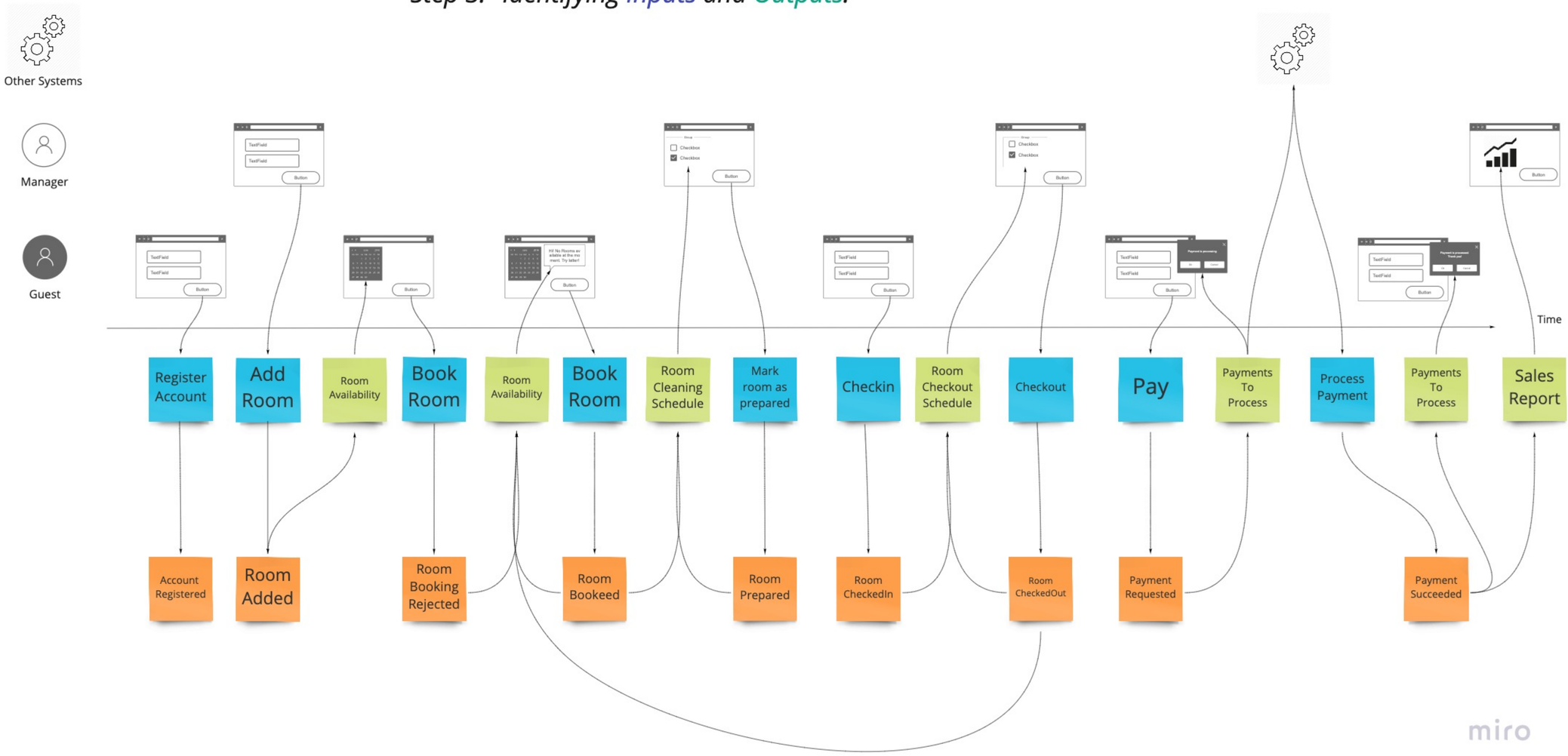
https://github.com/AxonIQ/axoniq-hotel

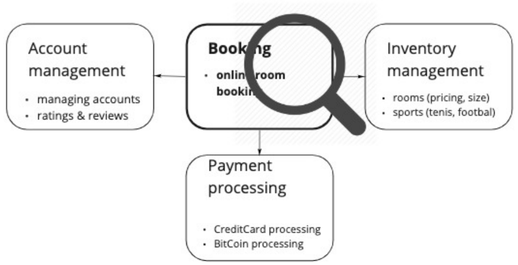
Event model

"What a system is supposed to do from start to finish, on a time line and with no branching"

Step 3: "Identifying Inputs and Outputs."







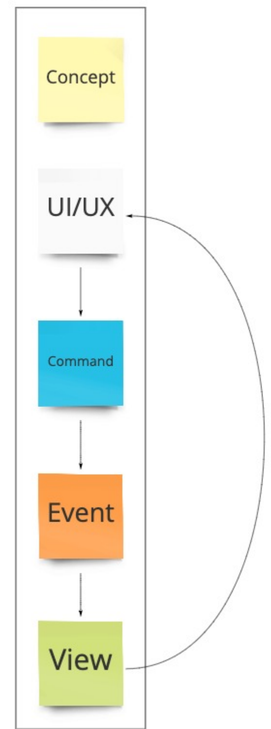
Demo

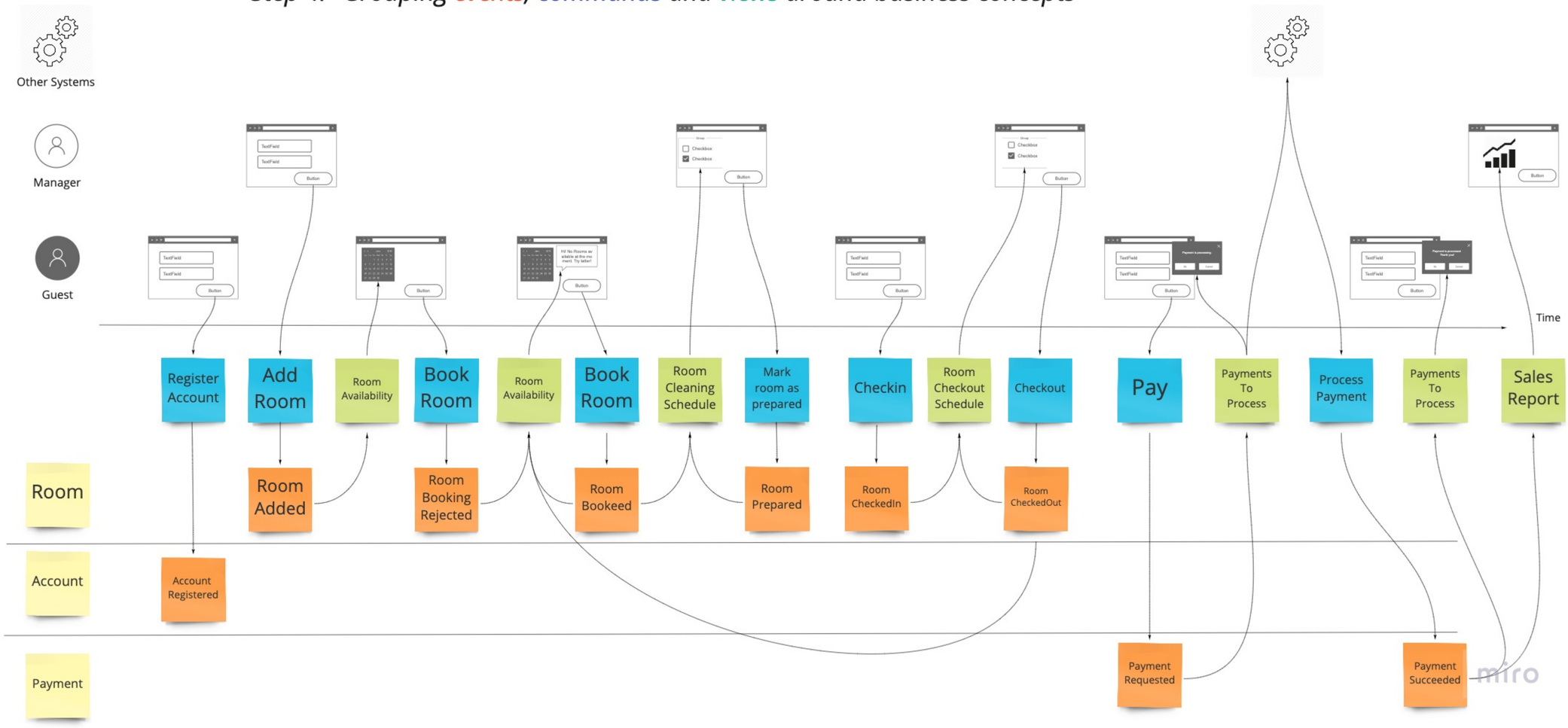
https://github.com/AxonIQ/axoniq-hotel

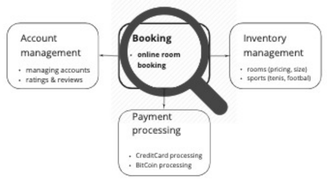
Event model

"What a system is supposed to do from start to finish, on a time line and with no branching"

Step 4: "Grouping events, commands and views around business concepts"







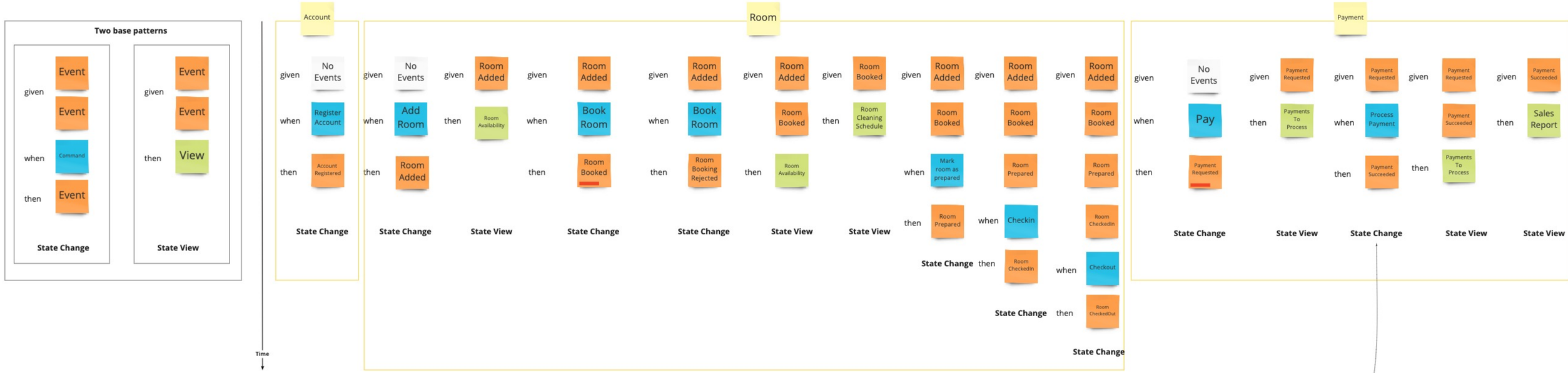
Demo

https://github.com/AxonIQ/axoniq-hotel

Event Model - Specification by example

"Collaborative approach to defining requirements"

Step 5: "Being more explicit about each State Change and State View we gain deeper understanding of the system requirements"



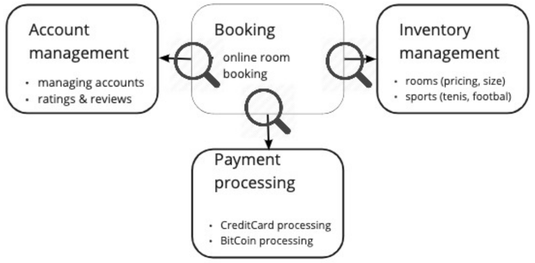
Transition to the (Java) source code is immediate. We are able to reflect the presented white board in a series of `acceptance` tests very fast, without loosing any information.

Axon Framework Test Fixture - Example

```
@Test
void processPaymentTest() {
    UUID accountId = UUID.randomUUID();
    UUID paymentId = UUID.randomUUID();
    PaymentRequested paymentRequested = new PaymentRequested(paymentId, accountId, BigDecimal.TEN);
    ProcessPaymentCommand processPaymentCommand = new ProcessPaymentCommand(paymentId);
    PaymentSucceeded paymentSucceeded = new PaymentSucceeded(paymentId);

    testFixture
            .given (paymentRequested)
            .when (processPaymentCommand)
            .expectEvents (paymentSucceeded);
}
```

Payment Requested
Process Payment
Payment Succeeded

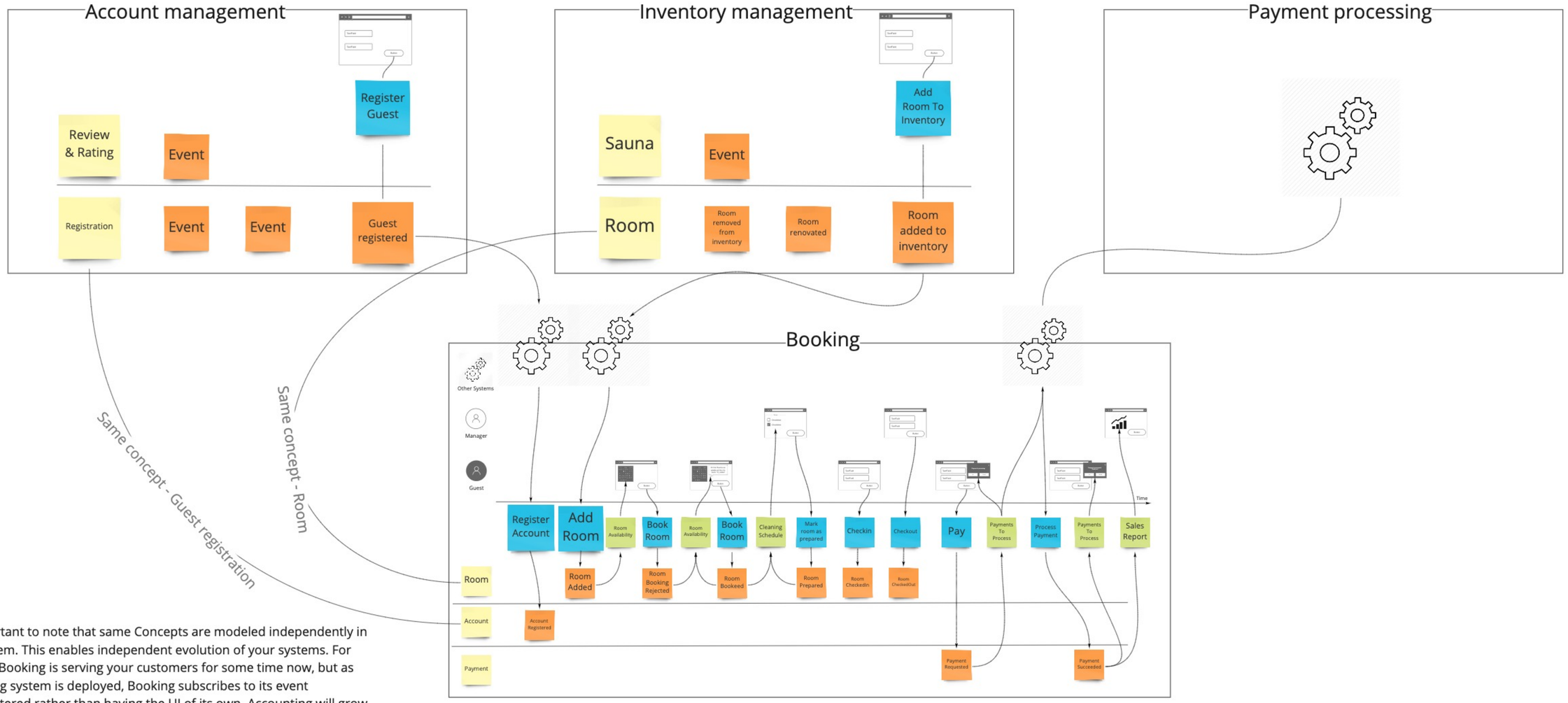


Demo

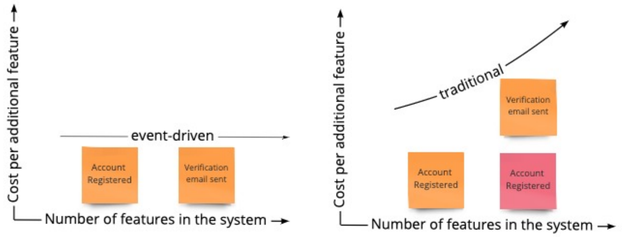
https://github.com/AxonIQ/axoniq-hotel

Systems Landscape - Integrations

"It's often useful to understand how all of these software systems fit together within the bounds of an enterprise"



It is important to note that same Concepts are modeled independently in each System. This enables independent evolution of your systems. For example, Booking is serving your customers for some time now, but as Accounting system is deployed, Booking subscribes to its event GuestRgistered rather than having the UI of its own. Accounting will grow without affecting Booking very much !



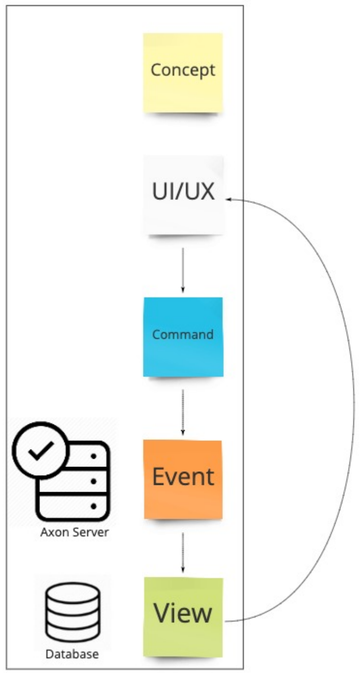
Demo

https://github.com/AxonIQ/axoniq-hotel

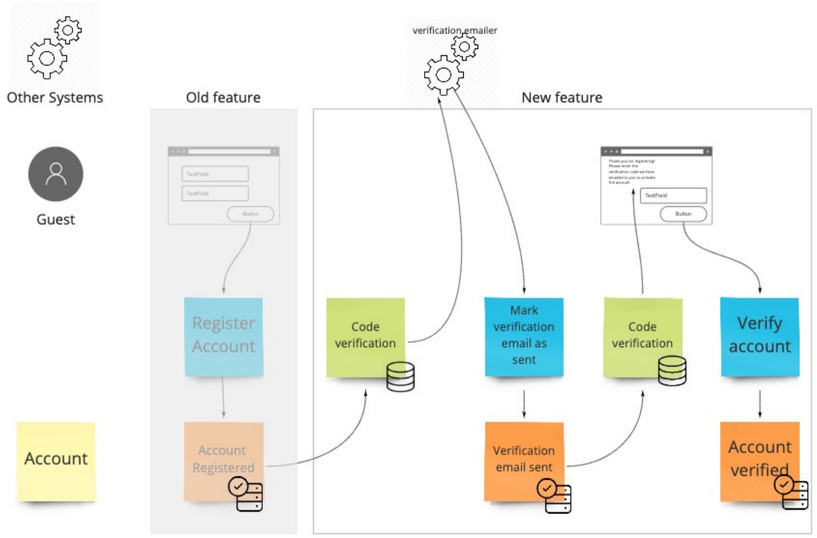
Event model

"Cost per additional feature - Event-Driven vs Traditional Systems "

5 moving parts



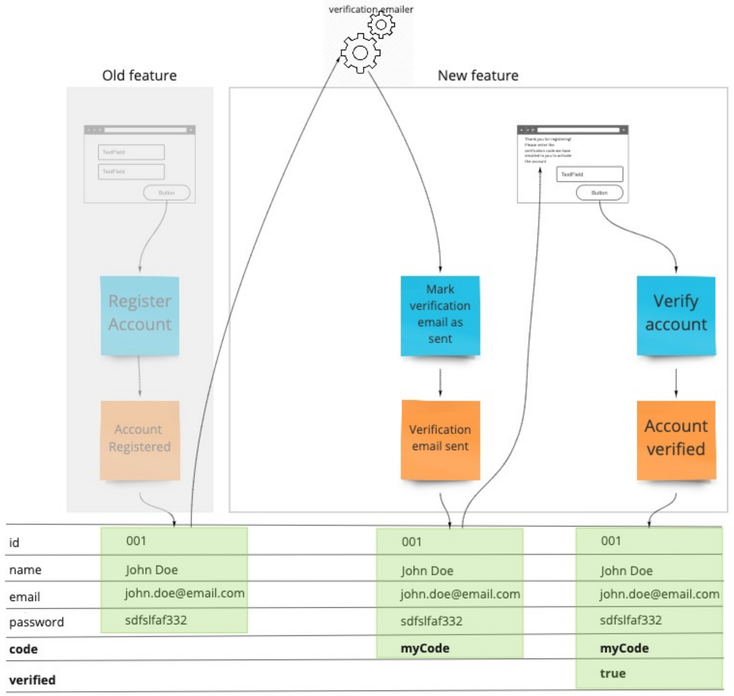
Event-Driven



The query model is continuously updated to contain a certain representation of the current state (**state view**), based on the events. This way, every feature in the workflow has its own view (own table, own DB schema, ...), keeping features independent and making `cost per additional feature` flat. This is **CQRS.**

CQRS enables/unlocks Event Sourcing! Event Sourcing mandates that the state change of the application isn't explicitly stored in the database as the new state (overwriting the previous state) but as a series of events. This way you don't loose any data/information. Everything that happened in the system is stored.
Information is far more valuable then the price of the storage these days, Don't throw it away!

Traditional



Being 'efficient' with storage requires re-opening the design of existing tables as we add new features to our system. It is this rework that is responsible for features costing more and more as the size of the whole system grows.