Table of Contents

Long-Running User Flows (10 pages)

> In the long run, the pessimist may be proven right, but the optimist has a better time on the trip.
>
> — Daniel Reardon

In the previous chapters, we have looked at handling commands and queries within the context of a single aggregate. All the scenarios we have looked at thus far, have been limited to a single interaction. However, not all capabilities can be implemented in the form of a simple request-response interaction, requiring coordination across multiple external systems or human-centric operations or both. In other cases, there may be a need to react to triggers that are indeterministic (those that may only occur conditionally) and/or be time-bound (**deadline**). This may require managing business transactions across multiple bounded contexts that may run over a long duration of time, while continuing to maintain consistency (**saga**).

There are at least two common patterns to implement the saga pattern:

- **Explicit orchestration**: A designated component acts as a centralized coordinator — where the system relies on the coordinator to react to domain events to manage the flow.
- **Implicit choreography**: No single component is required to act as a centralized coordinator — where the components simply react to domain events in other components to manage the flow.

By the end of this chapter, you will have learned how to implement sagas using both techniques. You will also have learnt how to handle exceptions using retries, compensating actions and deadlines. You will finally be able to appreciate when/whether to choose an explicit orchestrator or simply stick to implicit choreography without resorting to the use of potentially expensive distributed transactions.

Technical requirements

- JDK 1.8+ (We have used Java 17 to compile sample sources)
- Spring Boot 2.4.x
- mvvmFX 1.8 (https://sialcasa.github.io/mvvmFX/)
- JUnit 5.7.x (Included with spring boot)
- TestFX (for UI testing)
- OpenJFX Monocle (for headless UI testing)
- Project Lombok (To reduce verbosity)
- Axon server to act as an event store
- Maven 3.x

Continuing our design journey

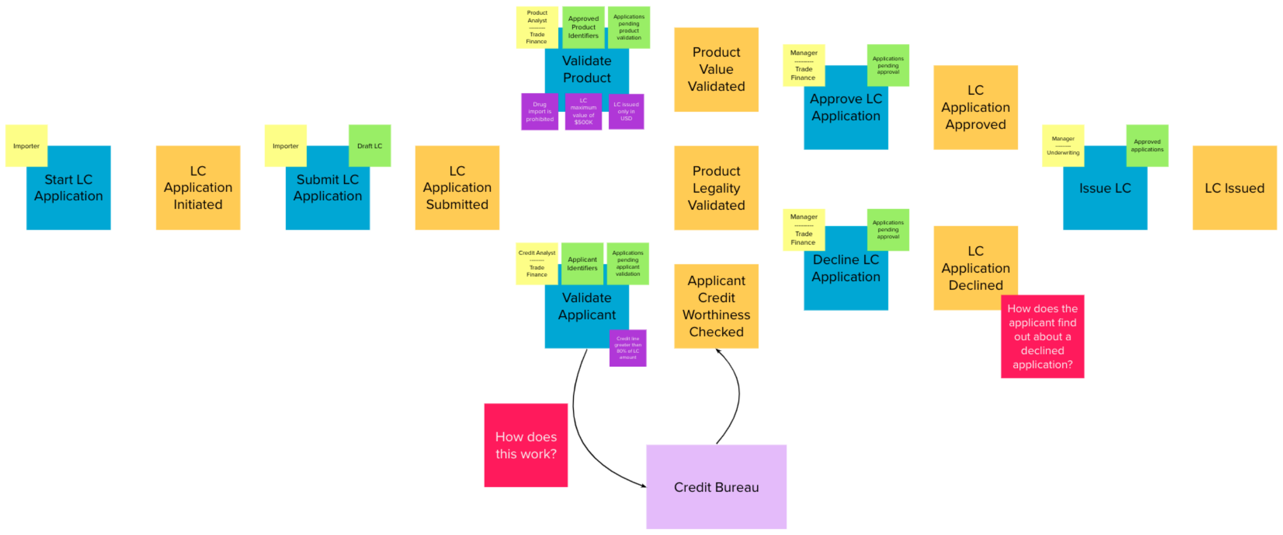
Importer
Start LC Application
LC Application Initiated
Importer
Draft LC
Submit LC Application
LC Application Submitted
Product Analyst ---------- Trade Finance
Approved Product Identifiers
Applications pending product validation
Validate Product
Drug import is prohibited
LC maximum value of $500K
LC issued only in USD
Product Value Validated
Product Legality Validated
Credit Analyst ---------- Trade Finance
Applicant Identifiers
Applications pending applicant validation
Validate Applicant
Credit line greater than 80% of LC amount
Applicant Credit Worthiness Checked
How does this work?
Credit Bureau
Manager ---------- Trade Finance
Applications pending approval
Approve LC Application
LC Application Approved
Manager ---------- Trade Finance
Applications pending approval
Decline LC Application
LC Application Declined
How does the applicant find out about a declined application?
Manager ---------- Underwriting
Approved applications
Issue LC
LC Issued

Figure 1. Recap of eventstorming session

Implementing sagas

Orchestration

Choreography

Orchestration versus choreography

Handling distributed exceptions

Types of exceptions

Business exceptions

System exceptions

Recovery

Automated recovery

Manual recovery

Compensating actions

Retries

Handling deadlines

Summary

Questions

Further reading

Title	**Author**	**Location**
Example	Author	https://www.example.com
Example	Author	https://www.example.com