cegeka

Saga pattern within MassTransit

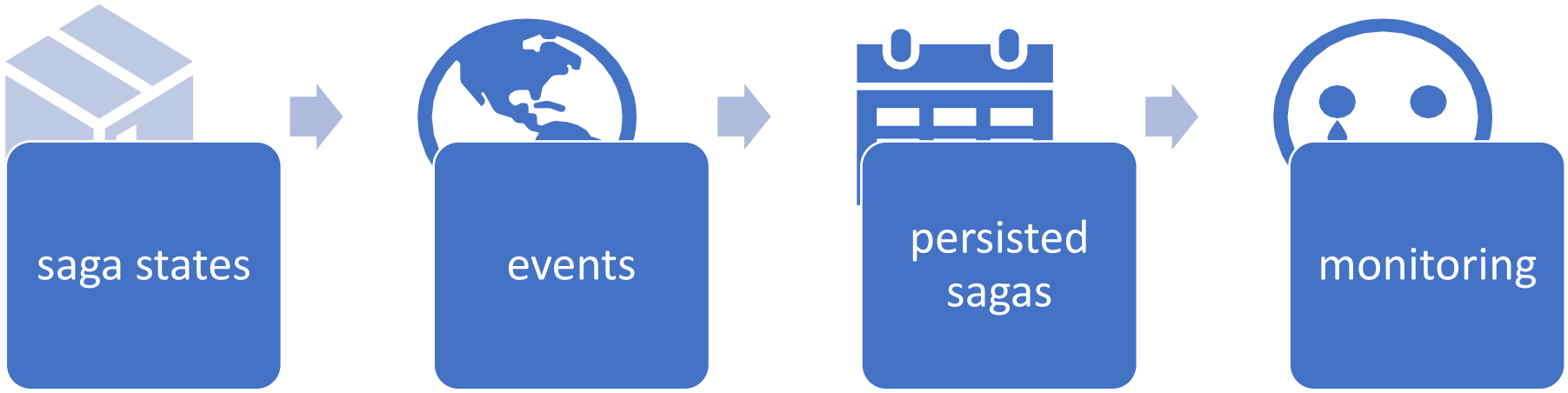
Balan Denis

2021.05.06

cegeka

Preface

Getting started with Saga Pattern within MassTransit
in a document generation example.



Document Generation Sample
Gathering requirements

- The client, with a dispatcher, must have the ability to send an **event to initiate the generation of a set** of related documents to an entity (e.g. insurance policy).

- **Objective**: To build a system that is
  - scalable
  - always reporting on the status of document generation
  - be fault tolerance
  - consistent - changes must be atomic

Context

We are developing a distributed document generation solution.

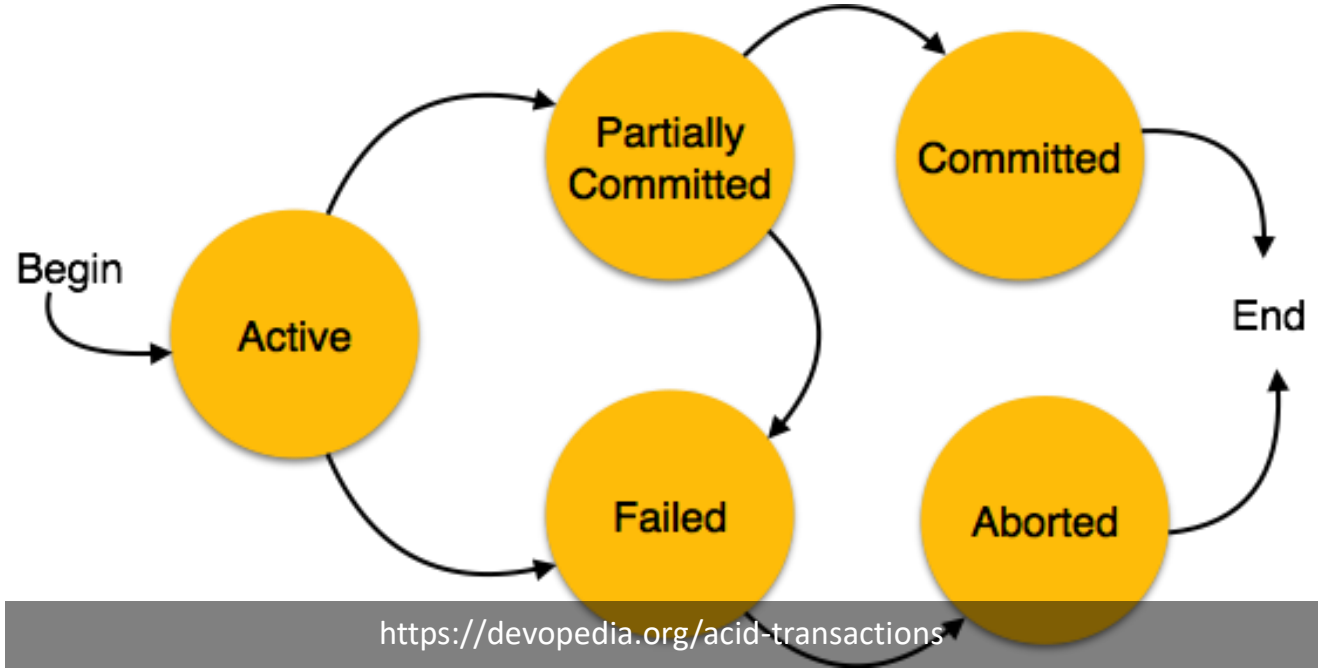
The application must ensure that each document is successfully generated, or else the entire set of documents is marked as erroneous.

Since data collection, generation, uploading, etc. are in different micro-services, the application must use a solution to orchestrate the entire workflow.

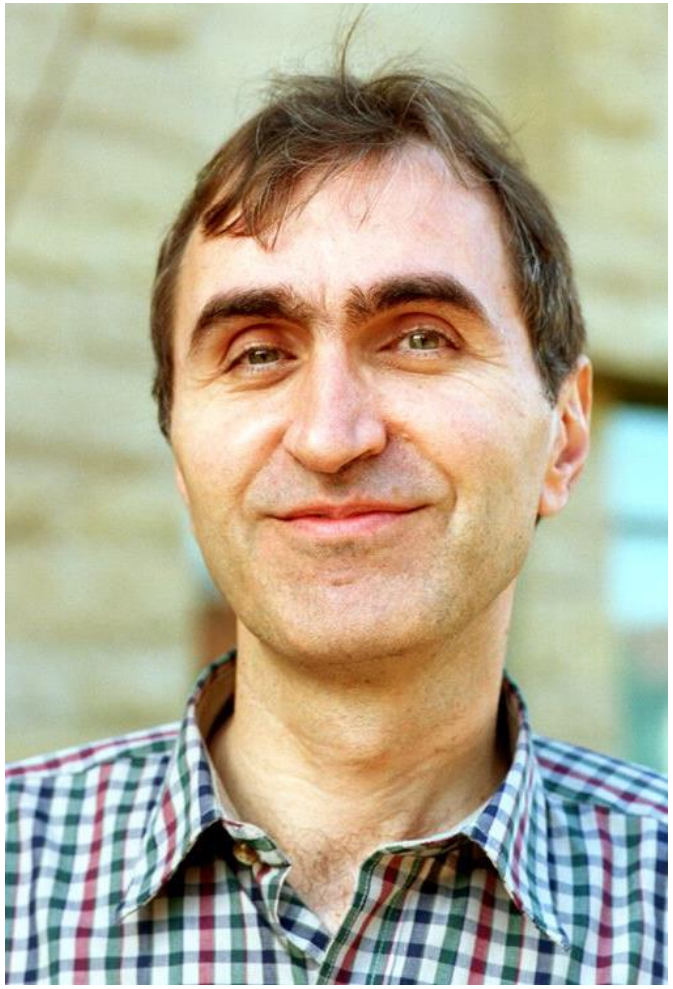
Before we start

- Transactions take a long time from initiation until completion. The existence of multiple databases and/or communication with multiple services makes difficult to maintain data consistency.

- Because the data is in different databases held by different services, it is not possible to simply use a local ACID transaction.



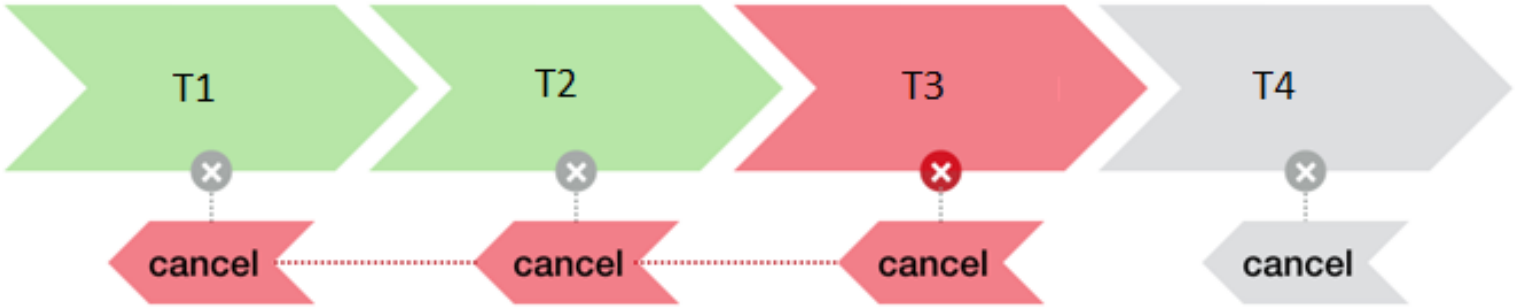
https://devopedia.org/acid-transactions



Hector Garcia-Molina, Kenneth Salem Princeton University 1987

Sagas are Long Lived Transactions

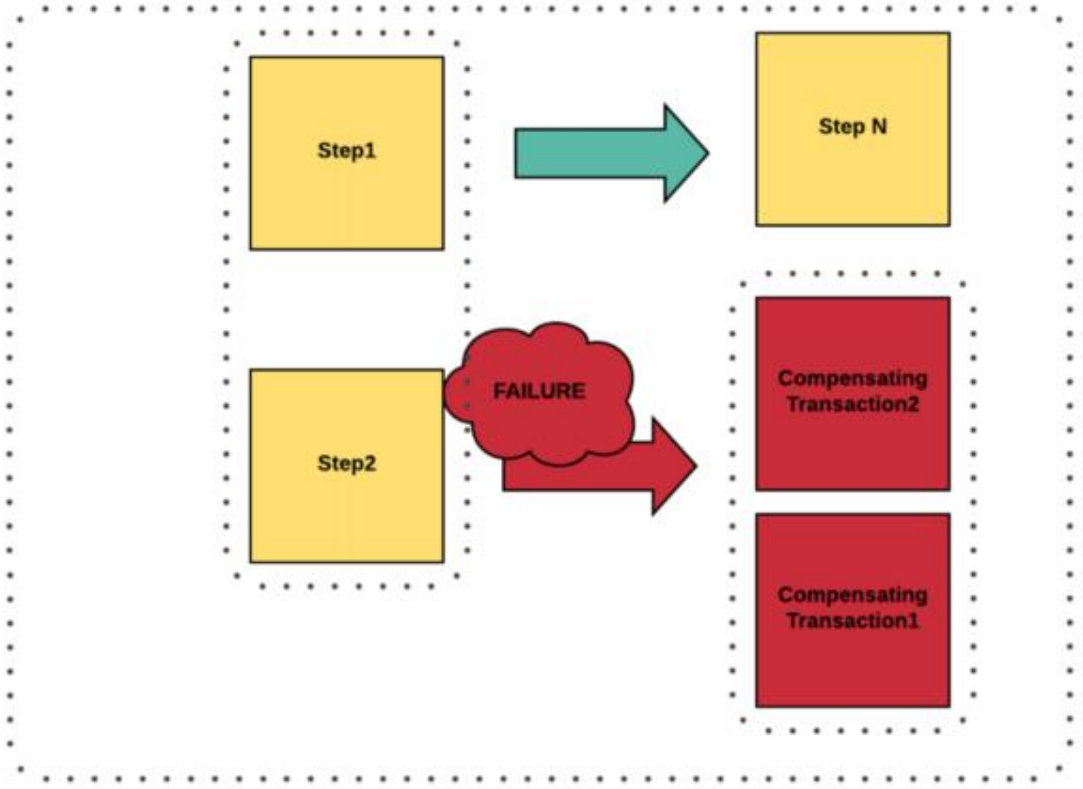
- “A Saga is a Long Lived Transaction that can be written as a sequence of transactions that can be interleaved.
- All transactions in the sequence complete successfully or compensating transactions are ran to amend a partial execution.”



https://www.cs.cornell.edu/andru/cs711/2002fa/reading/sagas.pdf

Sagas are a Failure Management Pattern

- The saga pattern is a **failure management pattern that helps establish consistency** in distributed applications, and **coordinates transactions** between multiple microservices to maintain data consistency

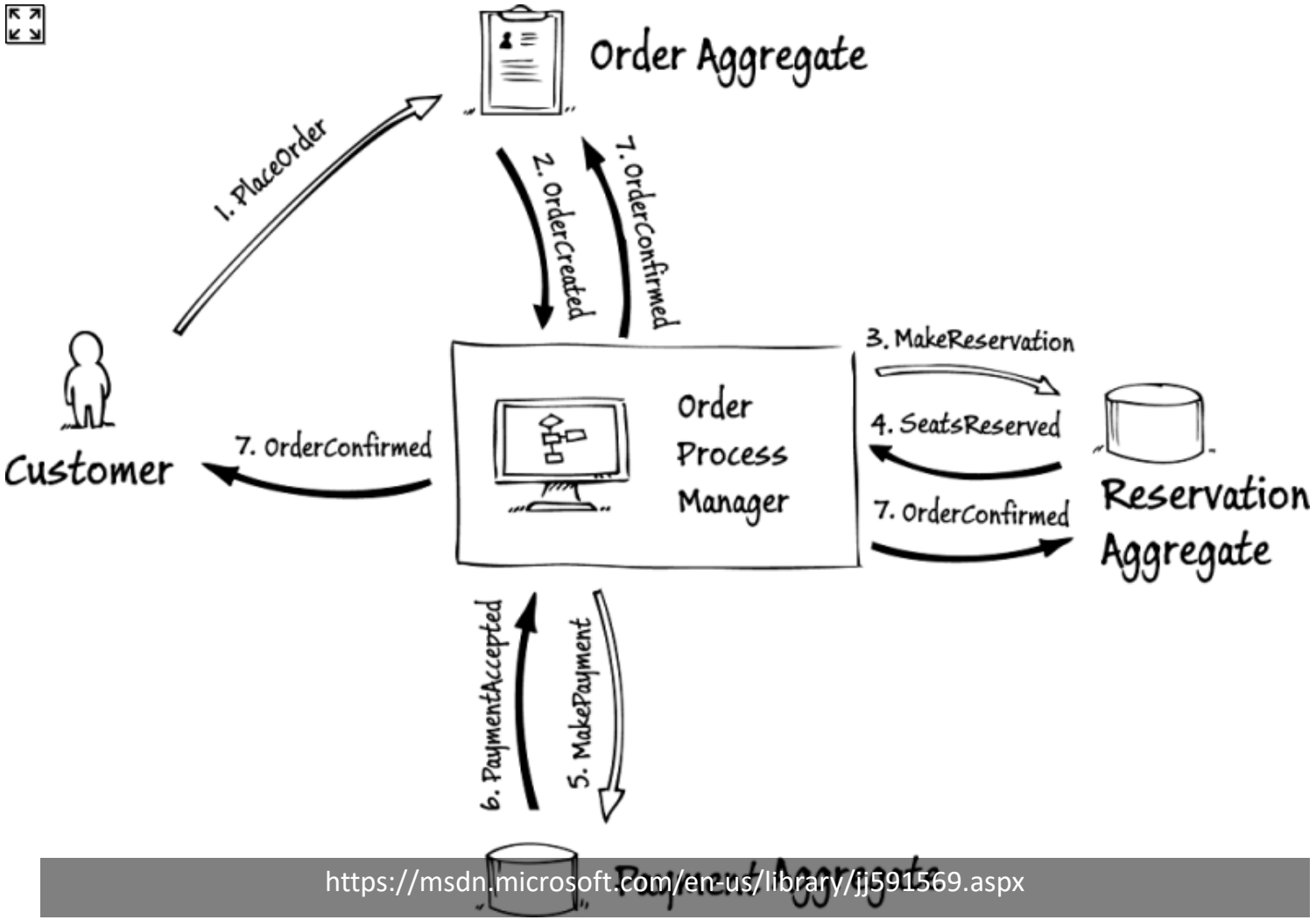


https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/saga-pattern.html

https://medium.com/@chaosgears/saga-patterns-inside-step-functions-world-b330c40fb9d5

Saga as Process Manager

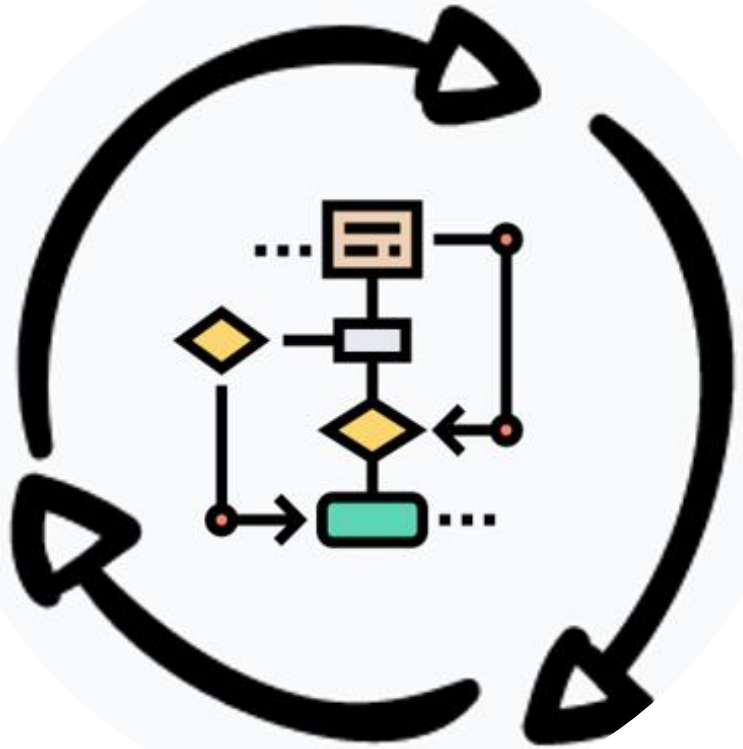
A Saga is a Collection of Sub-Transactions.



https://msdn.microsoft.com/en-us/library/jj591569.aspx

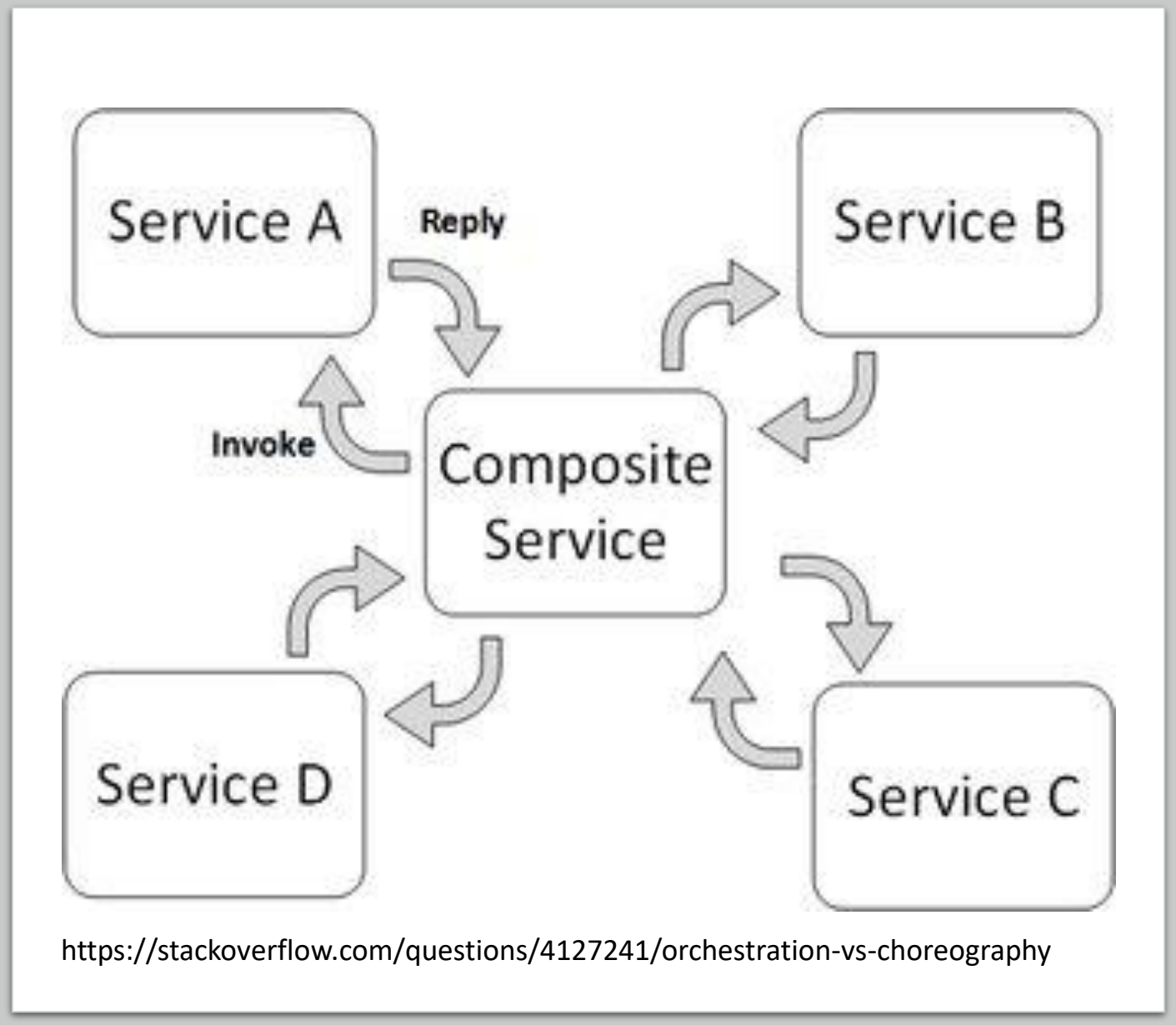
Saga on Sagas

- It's possible to implement Saga on Saga via invoking child Saga initialization message **from parent one**, while parent is in a certain state **until child will not finish**.



Implementation types - Orchestration

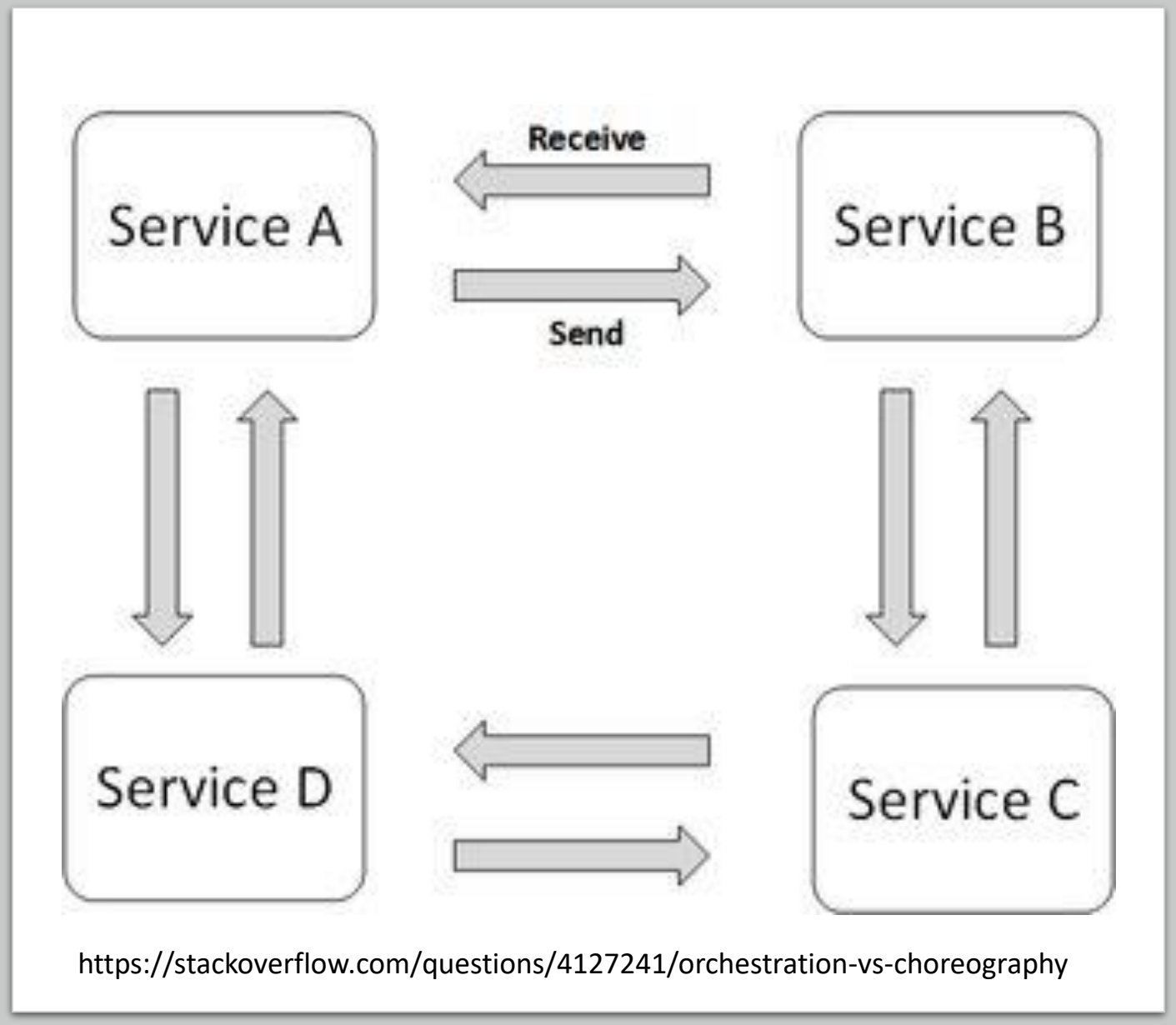
- Orchestration - Service orchestration represents a single centralized executable business process (the orchestrator) that coordinates the interaction among different services.
- The orchestrator is responsible for invoking and combining the services



https://stackoverflow.com/questions/4127241/orchestration-vs-choreography

Implementation types - Choreography

- Choreography - Service choreography is a global description of the participating services, which is defined by exchange of messages, rules of interaction and agreements between two or more endpoints.
- Choreography employs a decentralized approach for service composition



https://stackoverflow.com/questions/4127241/orchestration-vs-choreography

Benefits & drawbacks

of Orchestration

- Centralized logic: this can be good and bad
- Easier to understand the workflow since its defined in a central location
- Full control over the workflow steps
- Centralized point of failure reporting
- Easier to debug and test

of Choreography

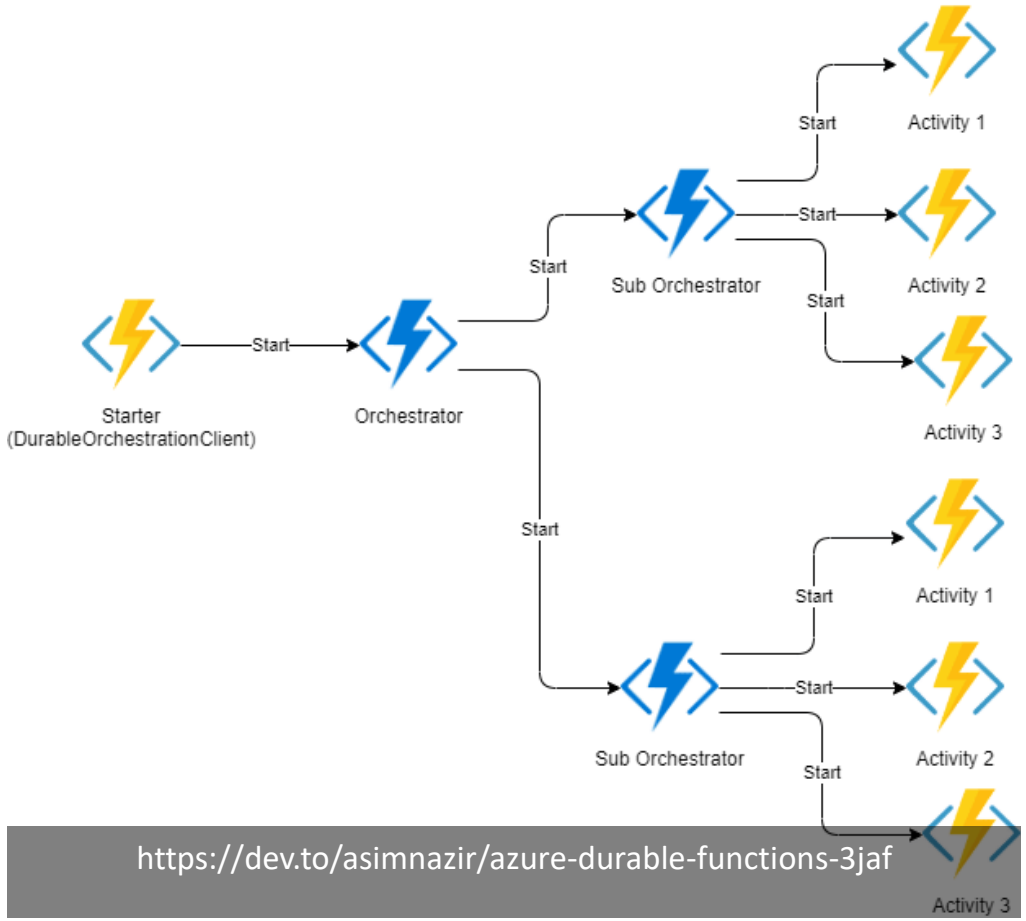
- No centralized logic: this can be good and bad
- Useful for small/simple workflows
- Difficult to conceptualize if a lot of services are involved.
- Hader to debug & test if a lot of services are involved

This means that a choreography differs from an orchestration with respect to where the logic that controls the interactions between the services involved should reside outside.

Ready to use products

- Azure Functions (+ Durable Functions)
- AWS Lambda (+ Step Functions)
- Google Cloud Functions (+ Workflows)
- https://temporal.io/ (fork from Azure Functions)

- K8S
  - Knative + https://kogito.kie.org/
  - https://argoproj.github.io/argo-workflows/
  - https://camunda.com/
  - https://kubeless.io/
  - https://fission.io/ (workflow)

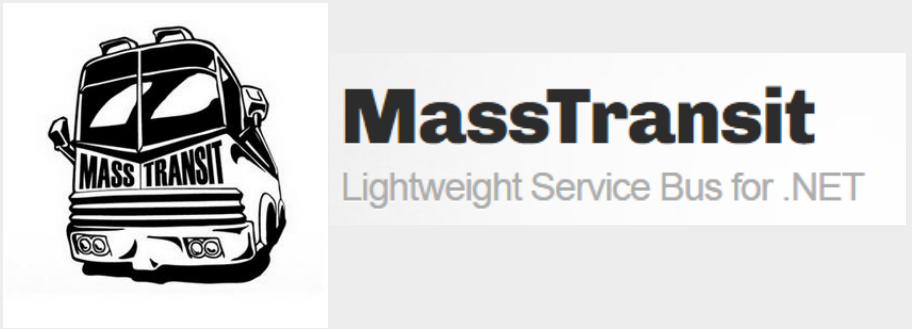


https://dev.to/asimnazir/azure-durable-functions-3jaf

.NET ecosystem focused

NServiceBus + Saga

MassTransit + Saga



Transport for both - RabbitMQ

MESSAGE BROKER VS MESSAGE BUS

Just another layer of abstraction

Messaging in .NET via MassTransit by Arturs Karbone

Message bus is broker agnostic (an abstraction above message brokers).

- Like ORM DB (EntityFramework/Dapper/etc.) providers in .NET

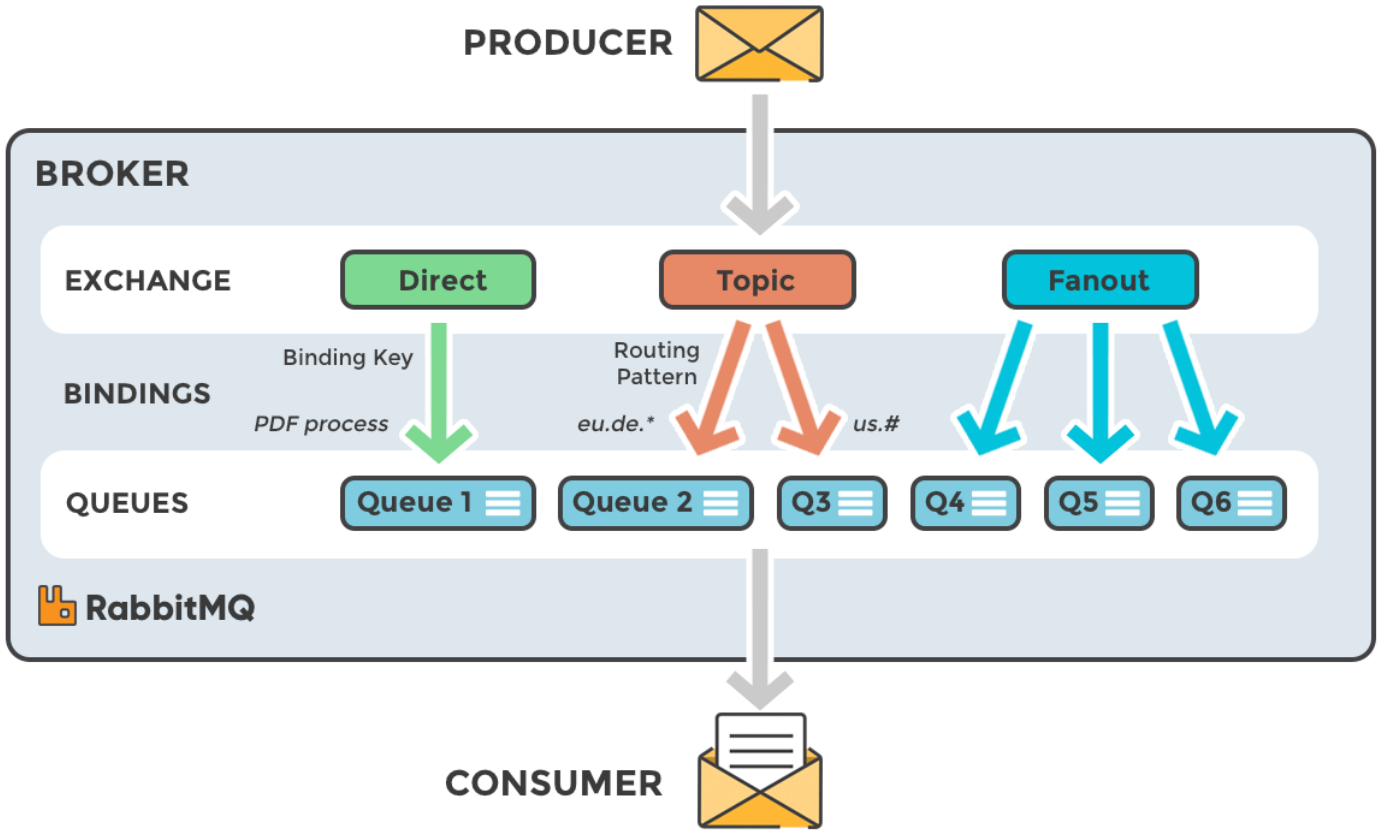
Message bus takes care of low - level details like

- Queues/Exchanges/Bindings
- Acknowledgements
- Serialization/Deserialization
- Error handling
- Retries
- Auditing

Developers can concentrate on sending and receiving messages

Message bus – usually is language specific

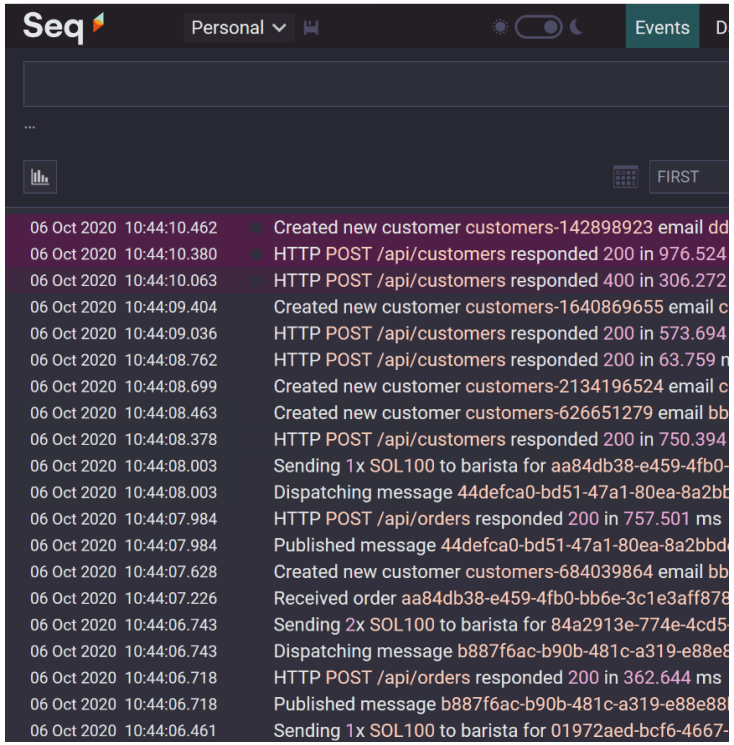
TRANSPORT THEORY



https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html

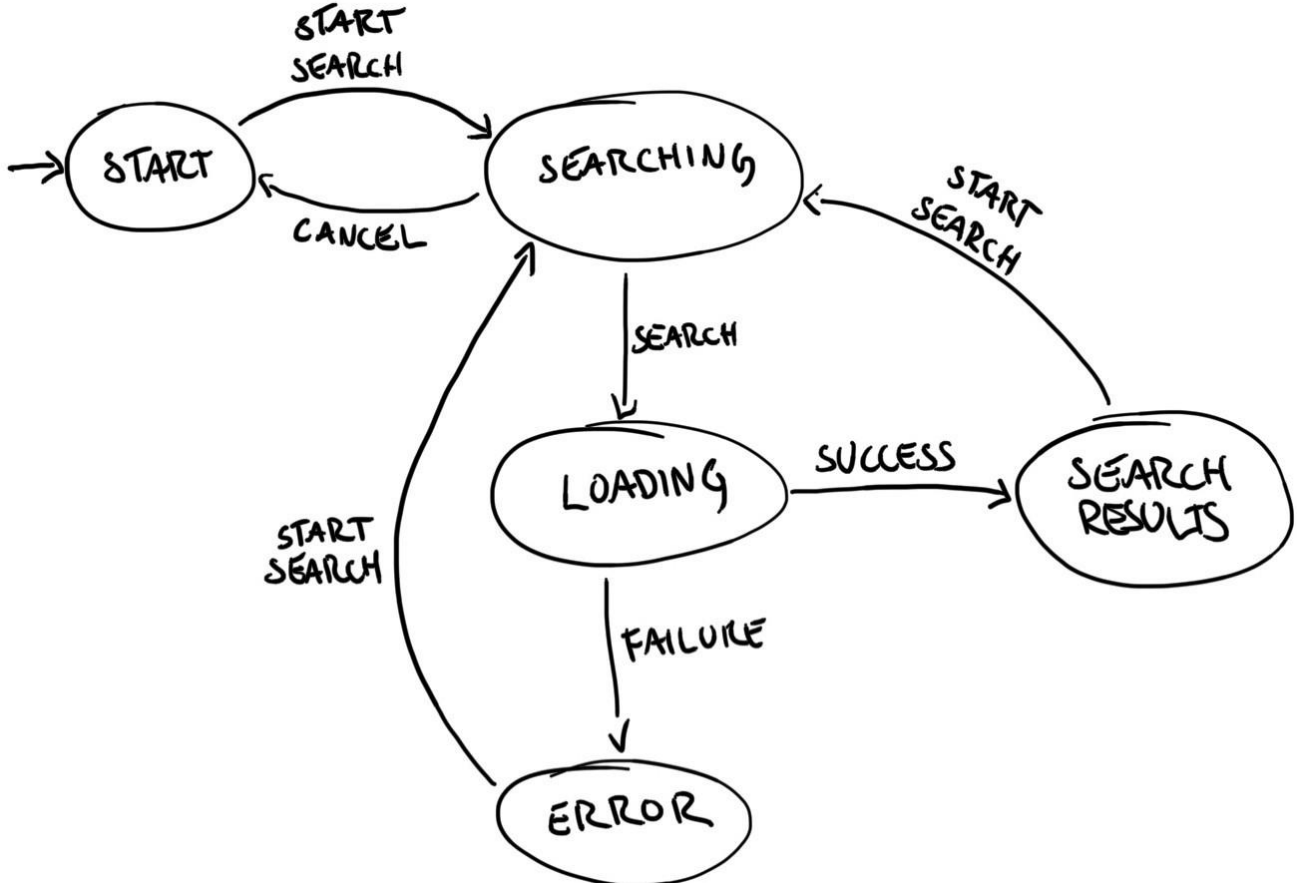
Document Generation Sample
Gathering requirements

- **Purpose**: Document generation software.
- Once finished he must receive the response (either generated urls **or** failed status)
- **Indicators**: Need a clear way of visualizing process - **SEQ**



Document Generation Sample
Gathering requirements

- **Actions**: Define transitions and states.
  - splitting a saga into several instances (fork)
  - setting possible states of a single document
  - defining events that trigger the change of states of an instance
  - sending the result to the receiver (join)

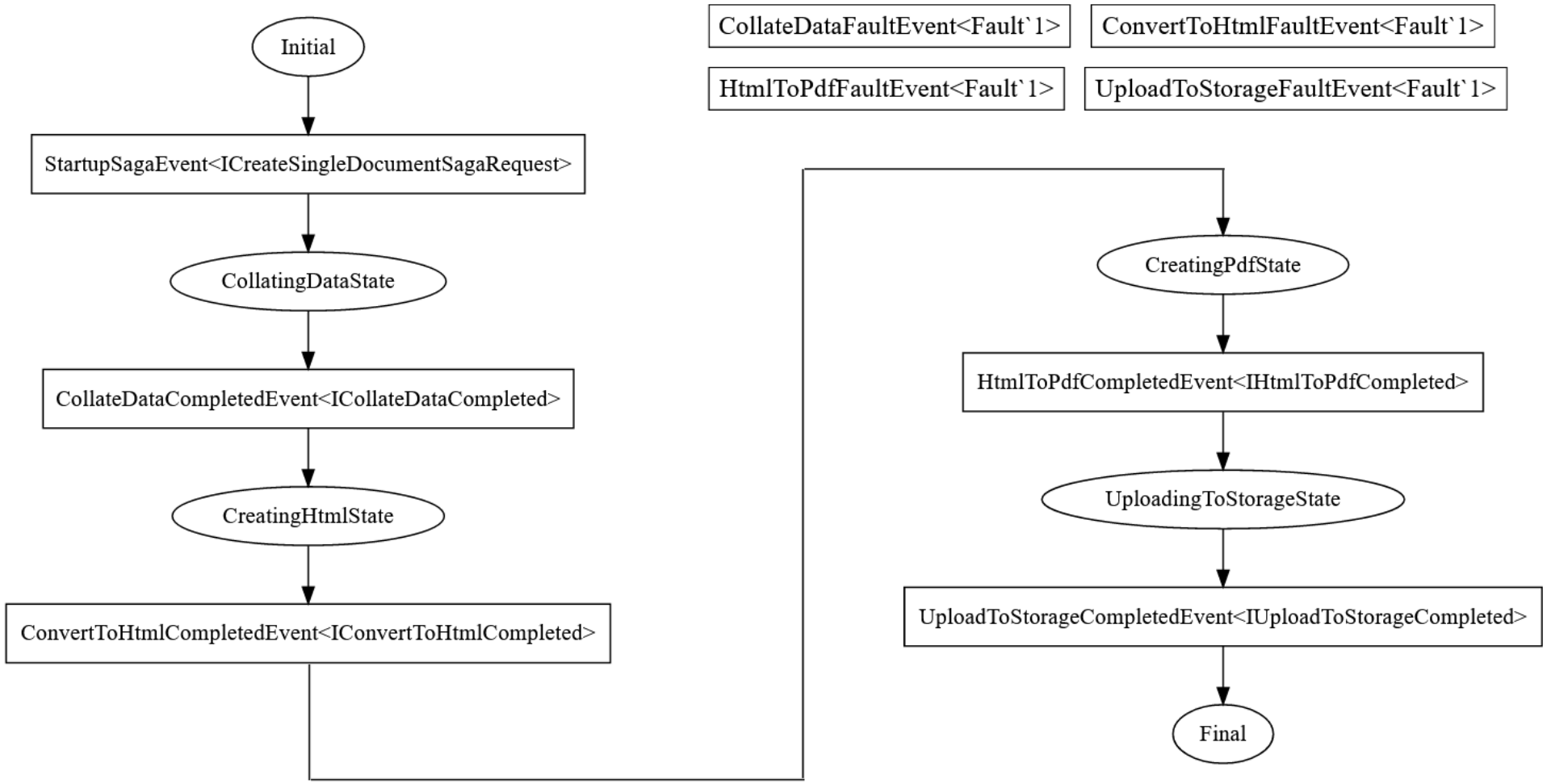


DGMS.Saga.Service.CreateSingleDocument

States & Events & Transitions

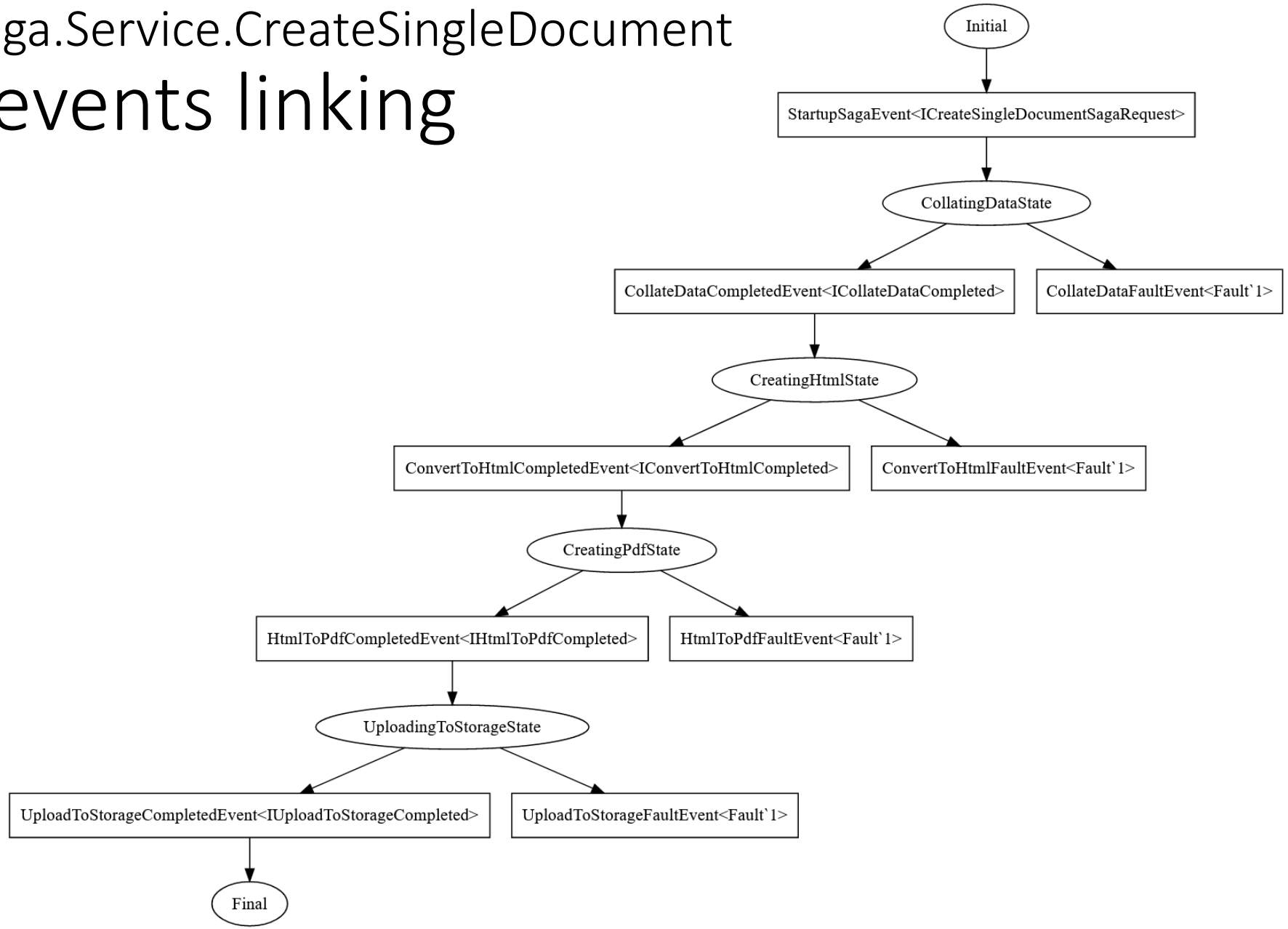
DGMS.Saga.Service.CreateSingleDocument

Fault events



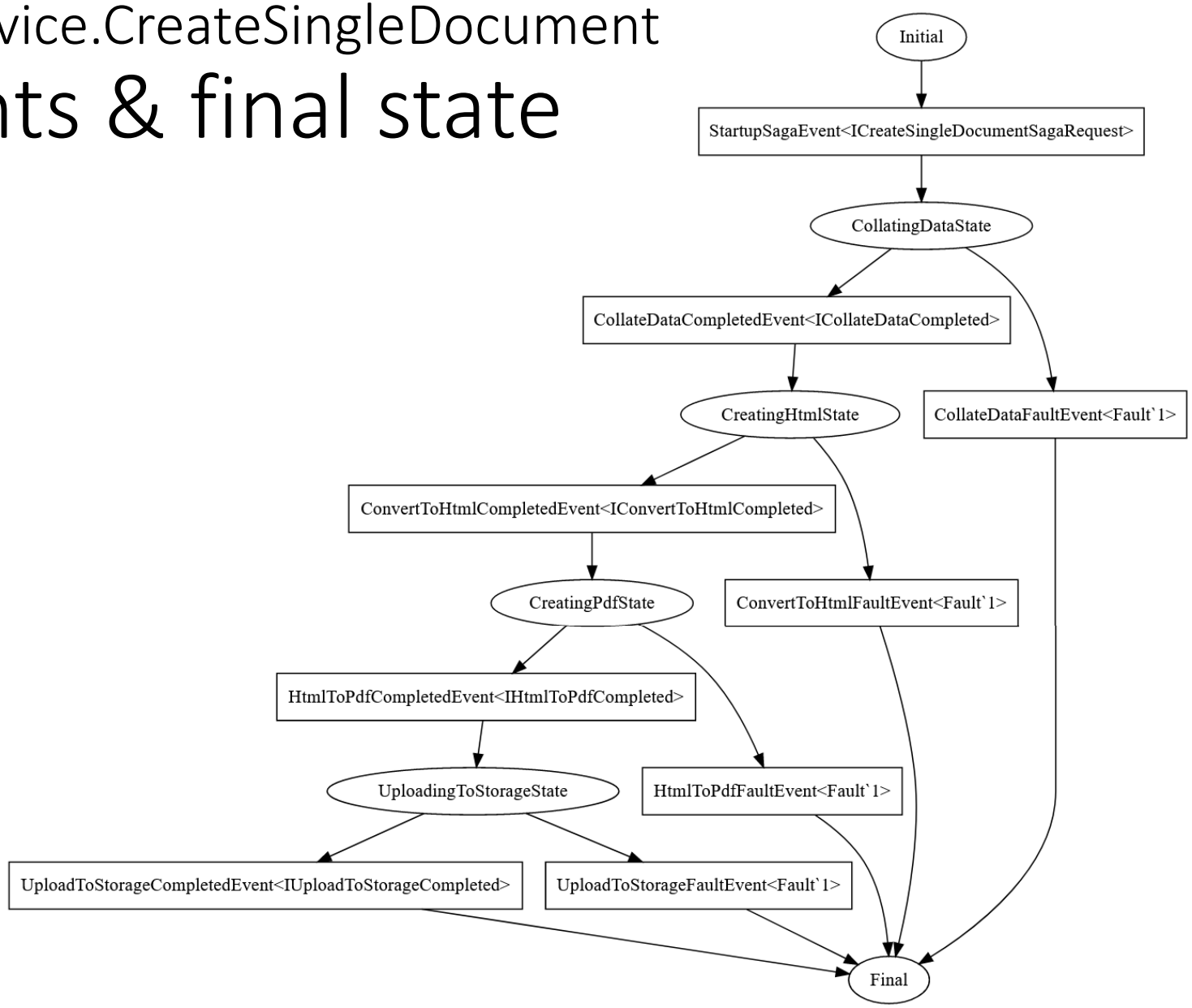
DGMS.Saga.Service.CreateSingleDocument

Fault events linking



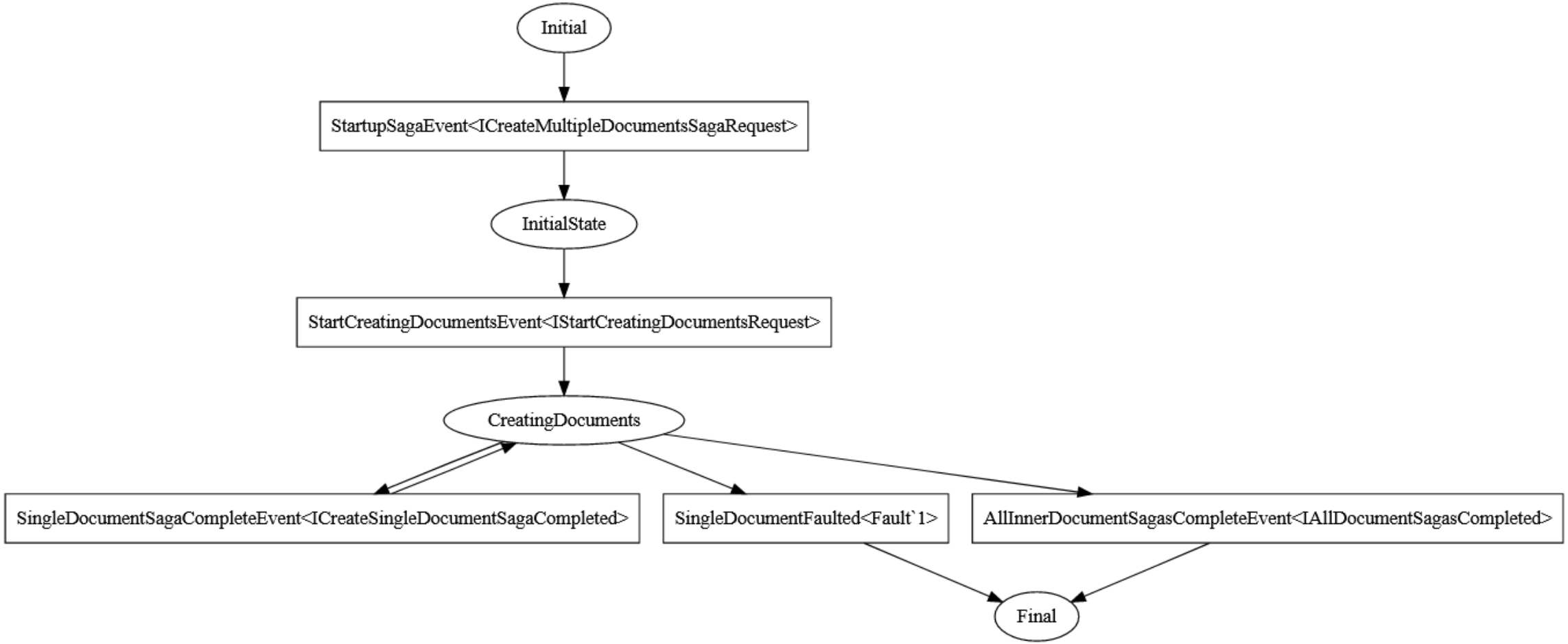
DGMS.Saga.Service.CreateSingleDocument

Fault events & final state



DGMS.Saga.Service.CreateMultipleDocuments

Saga on Saga




Demo

Questions

cegeka

Thanks for attention

Email: denis.balan@cegeka.com
Twitter: @DenisBalans
LinkedIn: https://www.linkedin.com/in/denis-balan/
GitHub: https://github.com/DenisBalan/