# Use of Transactions within a Reactive Microservices Environment

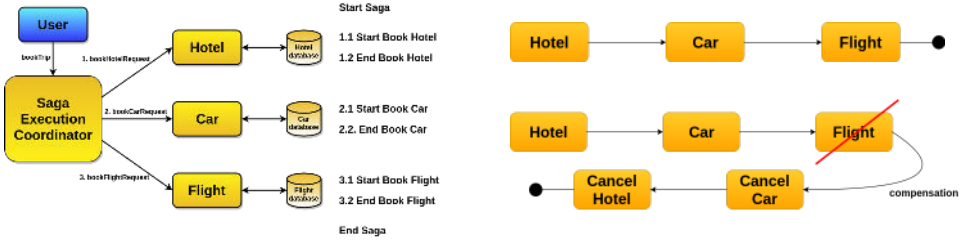
MASARYK UNIVERSITY
redhat.
Martin Štefanko
Advisor: Bruno Rossi, PhD

## PROBLEM STATEMENT

- **Transaction processing** is an inherent part of application development
- Particularly in the **distributed environment**, the utilization of transactions introduces several challenges that transaction management must be able to handle
- **Microservices** represent an emerging architectural style for the modern distributed application design which addresses concerns to stay responsive, elastic and resilient which is why the use of traditional locking transaction commit protocols may not be acceptable
- **The saga pattern** presents a suitable alternative solution to transaction processing that relaxes some of the ACID properties in order to promote availability

## SAGA PATTERN

- A saga is a **sequence of operations** that can be interleaved with other operations
- Each operation represents a unit of work that can be undone by the **compensation action** – the user defined action which can semantically undo the work performed by the original operation
- The saga guarantees that either all operations complete successfully or the corresponding compensation actions are run for all executed operations to cancel the partial processing




## CONTRIBUTIONS

### Saga development support

- The investigation of the current implementations of the saga pattern available for the enterprise Java applications
- Explored frameworks
  - Axon framework
  - Eventuate Event Sourcing (ES)
  - Eventuate Tram
  - Narayana Long Running Actions (LRA)

| Problem | Axon | Eventuate ES | Eventuate Tram | LRA |
|---|---|---|---|---|
| CQRS restriction | Yes | Yes | Optional | No |
| Asynchronous by default | Yes | Yes | No | No |
| Saga tracking and definition | No | No | Yes | No |
| Single point of failure | No | Yes | Yes | Yes* |
| Communication restrictions | Yes | Yes | Yes | No |
| Distributed by default | No | Yes | Yes | Yes |

- An example application has been created for all inspected frameworks – widely accepted by the respective communities
- We additionally performed an extensive performance testing of created examples by the custom performance test (several reported issues)

### LRA executor extension for Narayana LRA

- utilization of the LRA in the reactive microservices with main focus placed on the system responsivness
- **The asynchronous execution** is based on the user defined **LRA definition** which is passed to the executor service
- Easily extensible and flexible design, protocol and platform independent
- Supported be the quickstart example application

## FUTURE WORK

- Inclusion of the executor extension in the Narayana LRA implementation and in the currently forming Eclipse MicroProfile LRA specification
- Extensions of the LRA executor including new LRA definition formats, processing strategies, and new communication protocols support