Transactions for Microservices?

Tom Jenkinson

January 2018

Overview

- Transactions for Microservices and why they are needed
- Introducing Eclipse MicroProfile and Narayana LRA
- Live coding demo
- Conclusions

System faults

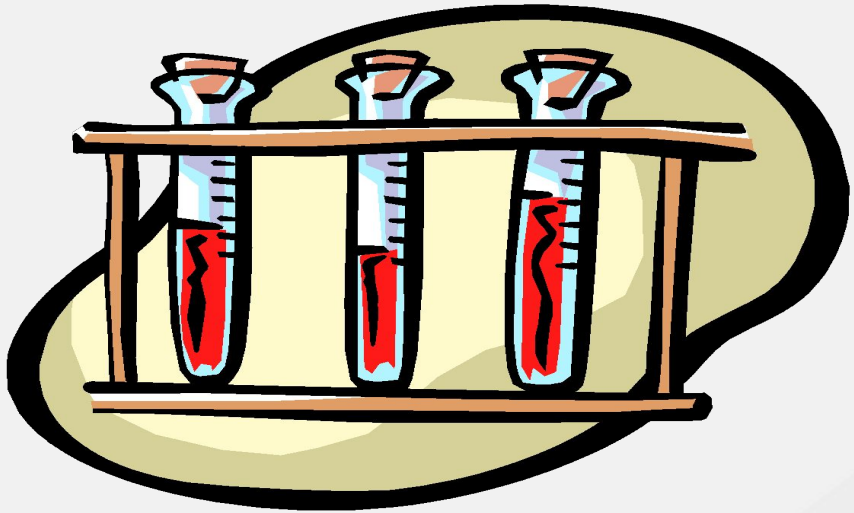
- Machines (and software!) fail…
  - Things get better with each generation
    - Individual machines have a lower probability of failure
    - Ran at scale the system as a whole sees an increase in the number of failures
- Failures of centralized systems are difficult to handle
  - But failures of distributed systems are much more difficult!
    - Microservices are all about distributed systems

What is a transaction and how can it help me?

- Mechanistic aid to achieving “**correctness**”
  - Provides an “**all-or-nothing**” property to work that is conducted within its **scope**
  - **Guarantees** shared state is protected from conflicting updates from multiple users
- There are many models…

Defining a transaction model...

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability



Transactions for Microservices?

- Microservice interactions are typically complex
  - Involving many parties
  - Possibly extending to different organisations
  - Potentially lasting for hours or days
- They must be coordinated somehow to deal with failures in these interactions
- So can full-ACID transactions help us to achieve “correctness”?

Well, yes but..

- Full-ACID transactions implicitly assume
  - One service can **block** another service
  - You can cope with resources being locked for periods of **time**
- Therefore full-ACID does not work optimally in
  - Loosely coupled environments!
  - Long duration activities!

If full-ACID isn’t suitable, what can we do?

- Relax some of the properties!
  - Let’s take a simple case of booking a seat on a plane and travel insurance….

Relax atomicity

- Sometimes it may be desirable to cancel some work without affecting the remainder
- Similar to nested transactions
  - Work performed within scope of a nested transaction is provisional
  - Failure does not affect enclosing transaction
- In our example:
  - It may still be preferential to get the airline seat even without travel insurance
    - The travel insurance might have been a nested transaction

Relax consistency

- ACID transactions are all about strong global consistency
  - All participants remain in lock-step
- But that does not scale well
  - Replication researchers have known about this for years
    - Weak consistency replication protocols developed for large scale (number of replicas and physical deployment)
  - Weak/eventual consistency is now a generally accepted approach
- In our example:
  - It might have very little consequence to other users of the system if the reservation of the airline seat is seen by other users of the system before the purchase of the insurance assuming eventually all updates are consistent

Relax isolation

- Internal isolation or resources should be a decision for the service provider
  - E.g., commit early and define compensation activities
  - However, it does impact applications
    - Some users may want to know a priori what isolation policies are used
- Undo can be whatever is required
- Regarding the example:
  - It might have very little impact to other users of the system if the reservation of the airplane seat is seen by other users but subsequently released back to the pool of available seats if a transaction is undone

Relax durability

- Although this is possible in some models its use is not appropriate in the use cases we try to address
- For example, if durability is relaxed:
  - We might be able to book the airline seat but subsequently a power loss of the transaction manager means we never complete the travel insurance booking

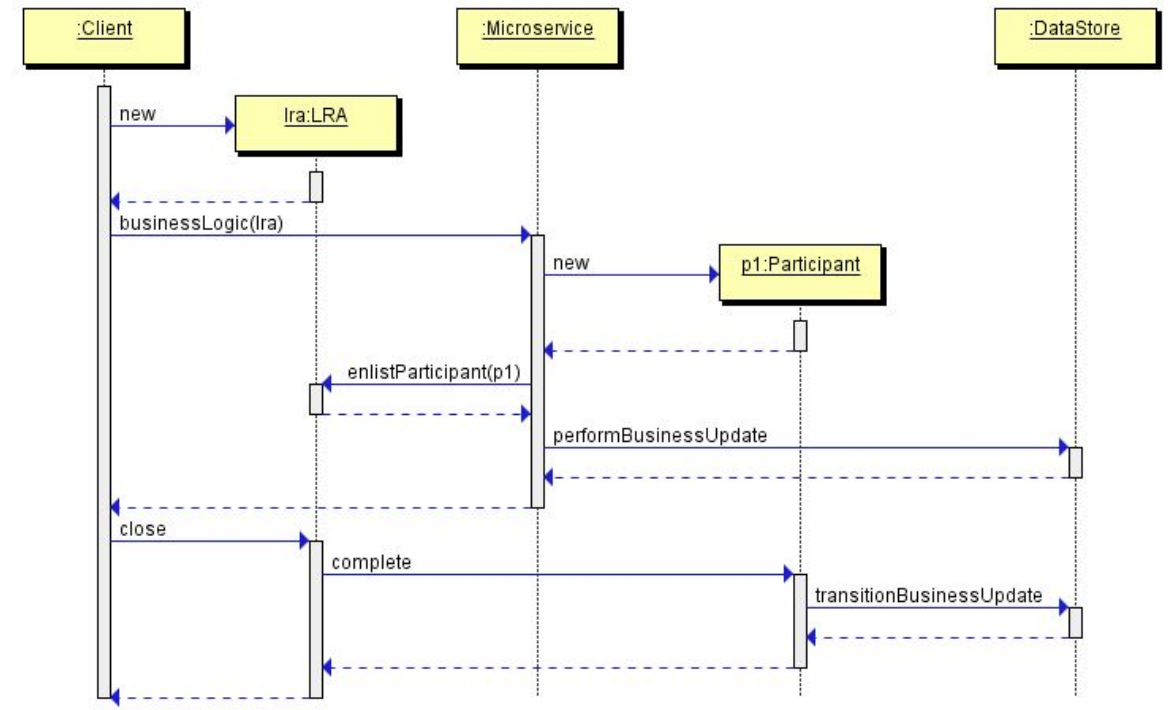
How to choose the best set of properties?

- We approached the MicroProfile community
- We looked at typical use cases
  - Booking of tickets
  - Booking a trip such as a taxi, hotel and flight
- We knew we needed to address scalability concerns associated with full-ACID
  - Removing locking means we will be relaxing the isolation property
    - This means that partial updates are visible and need to be catered for in-app
  - Would like to ensure that as much work as possible that is done by the system is useful so try to transition the system state forward at all times - only undo what is necessary

The result: Long Running Action

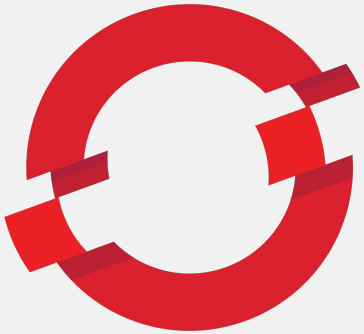
- A draft specification contributed to the MicroProfile community
- Defines an API for to coordinate activities allowing both
  - Long running activity
  - Loosely coupled processes
- Provides:
  - CDI annotations which can be applied to JAX-RS resources
  - A pure java based coordination API
- Built on the premise that each action defines a corresponding “undo” action
  - Forward compensation during activity is allowed
    - Keep business process making forward progress

Long Running Actions: Actors and interactions



Live coding demo

Notable tech used in the demo

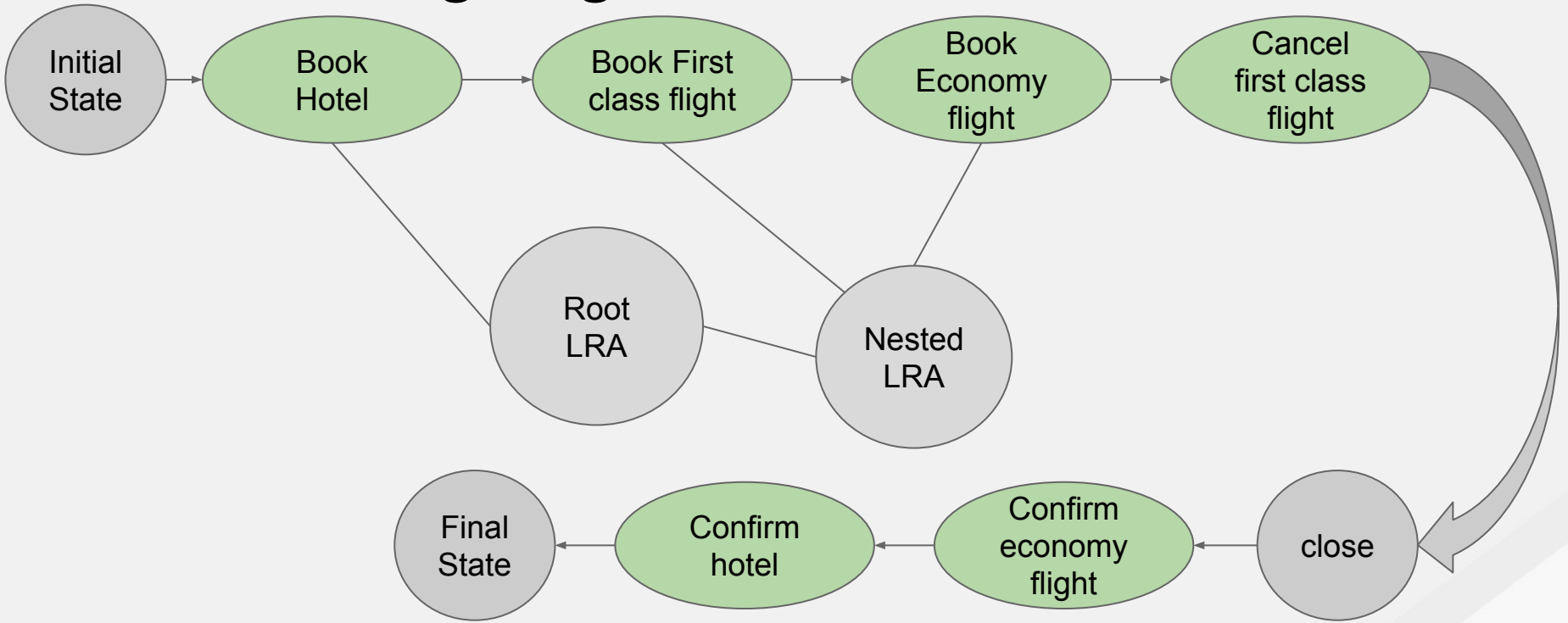


OPENSHIFT

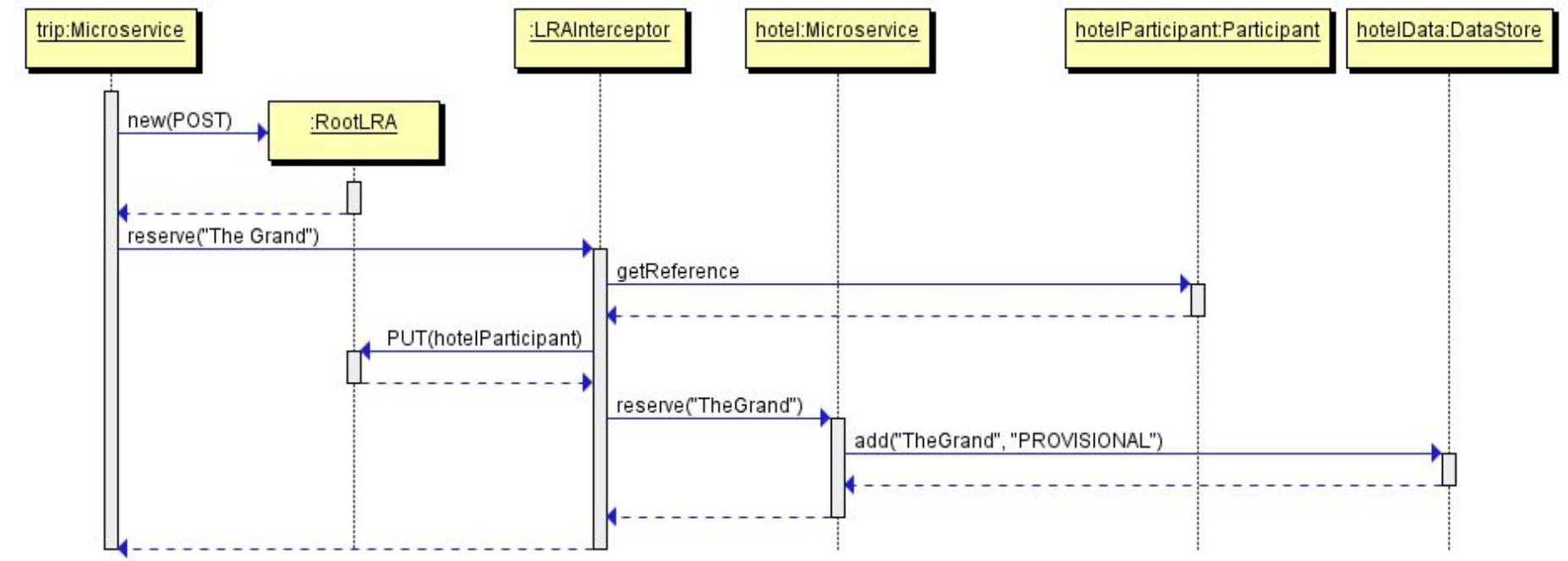
WildFly SWARM

narayana

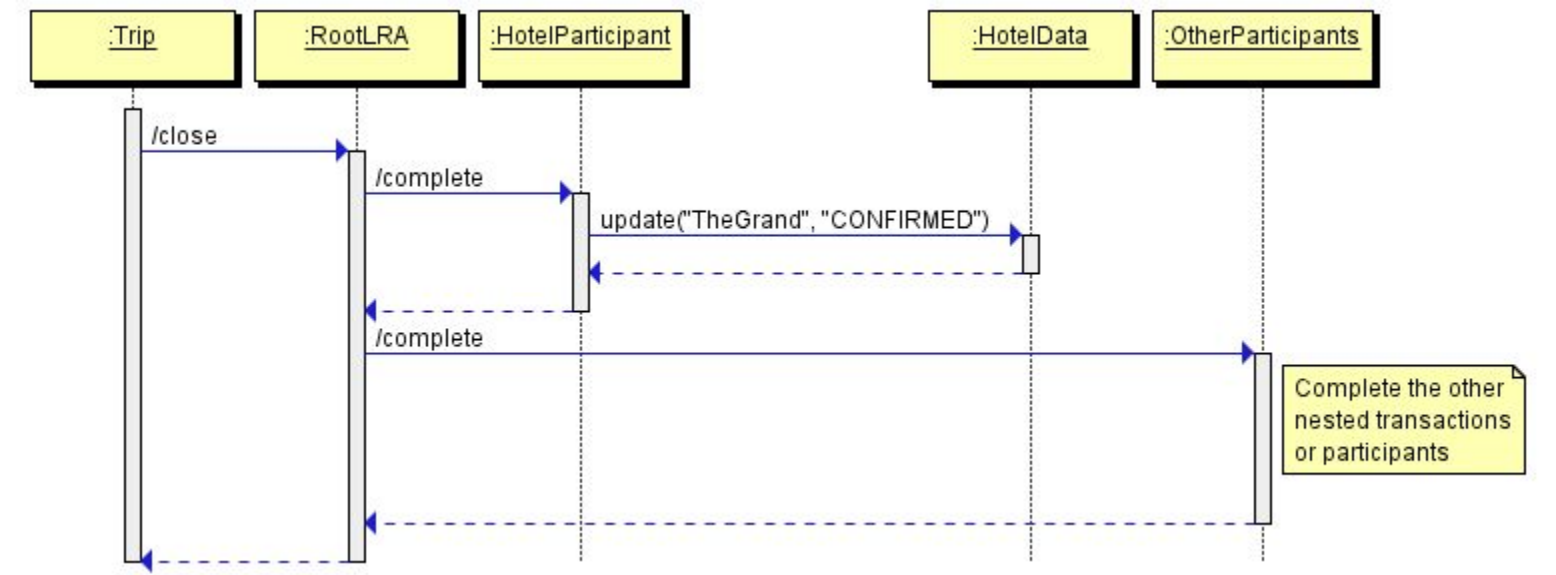
What we are going to do?



Deep-dive: Enlisting the Hotel Microservice



Deep-dive: Completing the booking



Running the demo!

Over to the IDE :)

Conclusions

- Transactions for Microservices?
  - Yes!
  - There are many transaction models that don’t necessitate full-ACID
    - Long Running Action specification
- Future work in MicroProfile
  - If you don’t agree with this model then get involved and help define more

Where can I find out more?

Narayana.io/community

- Forums
- IRC
- Blog
- https://github.com/jbosstm/conferences/201801-transactions-microservices/

MicroProfile mailing list

- “The need for transactions in the MicroProfile”
  - https://groups.google.com/forum/#!starred/microprofile/CJirjFkM9Do

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos