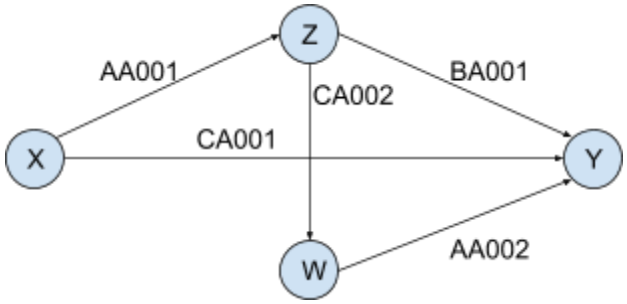
Learning Objectives

- Implementing actor model using Akka
- Integrating play framework with Akka
- RESTful web service
- Two-phase commit protocol

Story

We are implementing a prototype of an airline ticket booking system. The system has two roles: airline operator and booking coordinator. In our prototype, there are three airline operators, which are AA (American Airlines), BA (British Airways), and CA (Air China). They operate five flights among four cities. AA operates AA001 and AA002, BA operates BA001, and CA operates CA001 and CA002, as shown in the figure below:



The booking coordinator is aiming to satisfy users' travel demands from X to Y, and it always tries to book a trip with the smallest number of segments. That means it tries booking CA001 first; if failed(no seat or airline operator breakdown), it tries booking AA001+BA001; if still failed, it tries booking AA001+CA002+AA002. The table lists the number of available seats on each flight:

Flight Number	# Available seats
AA001	3
AA002	1
BA001	1
CA001	1
CA002	1

The system accepts requests via RESTful interface. We use actor model to process requests. There are two types of actors in the system: airline actors and the booking actor.

An airline actor represents an airline operator. It tracks the number of available seats on each flight it operates, accepts and responds to *Hold/Confirm* requests. A *Hold* request tries to hold a seat on a flight for booking, and a corresponding *Confirm* request then confirms the booking. If no *Confirm* is requested after a *Hold* request for a certain amount of time, the airline actor will cancel the hold.

The booking actor uses two-phase commit protocol to perform booking. In the first phase, it tries to hold a seat for each segment. If all of them succeed, it confirms them in the second phase.

This project is inspired by the paper Pardon & Pautasso, 2014. This paper may help you understand the scenario and techniques, especially the two-phase commit protocol. However, this project is different. Please closely read instructions.

Deliverables & Grading Parameters

NOTE: The following parts are mainly for grading purposes. Please submit your whole project as one .zip file.

Part 0: Valid Submission (5%)

A valid submission means your submitted project executes out of box. In other words, it can be executed using 'sbt run' on a clean computer with only JDK and sbt installed. Make sure your project:

- Is submitted completely;
- Has no compiling error;
- Includes essential libraries, either in your code repository or configured as remote maven libraries in build.sbt script; and
- Includes no absolute path.

A safe way is to test your program on another computer before you submit.

Please submit your whole project as ONE .zip file via Moodle, other forms of submission will not be accepted. Refer to *Guidelines* section for guides of reducing submission size.

Part 1: A Play Framework based web service with RESTful APIs (10%)

Your program should provide specified RESTful APIs. That is, it should respond to specified RESTful requests and its responses comply with the API specification. Refer to the API section for the specification of APIs.

Part 2: Akka actors with debugging APIs (25%)

Your program should construct three airline actors (AA, BA, and CA) and one booking actor, along with their debug APIs. Refer to the API section for specification of APIs. You will take risk of no credit for this part if you do not provide debug API. The business logic is not considered in this part.

Part 3: Airline actors (25%)

Implement the business logic of airline actors, which includes but is not limited to:

- Recording available seats of each flight
- Responding *Hold* and *Confirm* requests
- Cancelling holds when timeout
- Debug APIs of airline actors

Part 4: Booking actor (25%)

Implement the booking logic of the booking actor, which includes but is not limited to:

- Responding to a booking request, trying to satisfy the request with the smallest number of segments
- Performing booking using two-phase commit protocol
- Linking RESTFul APIs to booking actor
- Debug APIs of booking actor

Part 5: Error handling (10%)

You should deal with errors occurring in different components:

- FAIL response from actors
- Timeout in an actor's response.

In the booking actor, you should keep an event log for recovery purposes. Record at least the following events:

- Responses of *Hold* requests, including:
    - Success or failure
    - Expiration time
- Responses of *Confirm* requests
- Partially failed requests, which happen when
    - one confirm fails but at least one confirm succeeds in the current transaction.
    - one *Hold* expires without successful confirm but some confirms succeed in the current transaction.

Record a timestamp and a transaction ID with each log entry. The transaction ID should be unique for each transaction.

You are not required to implement recovery mechanism in this project.

Ingredients

- JDK 8 (required)
- sbt (ver 0.13.15 or higher)
- Play (ver 2.6.2 or higher)
- Akka (ver 2.5.4 or higher)
- sqlite-jdbc (ver 3.20.0 or higher)

API

RESTful API

Request	Response	Description

GET /trips	{
	“status”: “success”,
	“trips”: [<list of trip IDs>]
	}
		Get a list of trips booked.

GET /trips/:tripID	{
	“status”: “success”,
	“segments”: [<list of flight code>]
	}
	or
	{
	“status”: “error”,
	“message”: <error message>
	}
		Get a list of segments of a trip. A segment is represented by its flight.

GET /operators	{
	“status”: “success”,
	“operators”: [<list of operator codes>]
	}
		Get a list of airline operators.

GET /operators/:operator/flights	{
	“status”: “success”,
	“flights”: [<list of flight codes>]
	}
	or
	{
	“status”: “error”,
	“message”: <error message>
	}
		Get a list of flights operated by an airline operator.

GET /operators/:operator/flights/:flight	{
	“status”: “success”,
	“seats”: <number of available seats>
	}
		Get the number of available seats on a flight.

	of
	{
	“status”: “error”,
	“message”: <error message>
	}	

POST /trip/:from/:to	{
	“status”: “success”,
	“tripID”: <trip ID>
	}
	or
	{
	“status”: “error”,
	“message”: <error message>
	}
		Book a trip. **Currently, the $from and $to should always be X and Y. If not, return an error.**

Debugging API

These APIs are for debug purpose, mainly for mocking fail and timeout scenarios of actors. In the following APIs, *$airline* can be *AA*, *BA*, or *CA*.

Request	Response	Description
POST /actor/$airline/confirm_fail	{ “status”: “success”}	After this request is posted, corresponding airline actor will reply fail to subsequent *Confirm* requests without actual processing.
POST /actor/$airline/confirm_no_response	{ “status”: “success”}	After this request is posted, corresponding airline actor will not reply to subsequent *Confirm* requests without actual processing.
POST /actor/$airline/reset	{ “status”: “success”}	After this request is posted, the actor will reset to normal.

Guidelines

The guidelines are mainly for Java, many of them may also apply to both Java and Scala though. You are free to choose between Java and Scala.

- ### Set RESTful Routes

Take *GET /operators/:operator/flights/:flight* as an example. First, add following line into routes.conf:

GET /operators/:operator/flights/:flight
controllers.HomeController.getFlight(operator: String, flight: String)

In the *HomeController*, add a function:

```
public Result getFlight(String operator, String flight){
    return ok("operator: " + operator + ", flight: " + flight);
}
```

Try to visit *http://localhost:9000/operators/AA/flights/AA001*
You should see *operator: AA, flight: AA001*

Refer to: http://www.baeldung.com/rest-api-with-play

- Use Akka

Akka is already part of the Play Framework, so you do not need to install it. Just import it when necessary:

```
import akka.actor.*;
import akka.japi.*;
```

For quick Akka examples, refer to:
http://developer.lightbend.com/guides/akka-quickstart-java/?_ga=2.96077404.1561246931.1504127451-2077373114.1503939704

- Talk to Akka actor in Play

Refer to: https://www.playframework.com/documentation/2.7.x/JavaAkka

- Reduce the size of submission

The size of submission should be far less than the locker limit if you set and clean it properly. Here are some tips for size reduction:

- Do not include binary libraries in your project. If you need to use a third party library, add it into *build.sbt* as a dependency. For example:

  libraryDependencies += "org.xerial" % "sqlite-jdbc" % "3.28.0"
  The library will be downloaded and installed from maven repository when next time `*sbt run* ` being executed.

- Remove target and intermediate files. For a sbt project, this can be achieved by deleting nested target directories, such as *./target*, *./project/target*, and *./project/project/target*. Refer to: https://stackoverflow.com/questions/4483230/an-easy-way-to-get-rid-of-everything-generated-by-sbt