MCV Actors
An Akka Experiment

Alexandre Zua Caldeira
zuacaldeira@gmail.com

September 17, 2016

Contents

Part I

Overview

Chapter 1

Introduction

Chapter 2

Java, The Tool Not The Target

Chapter 3

Vaadin, Alone With Java

Chapter 4

*Akka*ros, MVC Actors

Chapter 5

Akkaria, A World To Live In

Part II

The Akka Experiment

Chapter 6

Methodology

Chapter 7

Development Log

7.1 Day 1: September 17, 2016

7.1.1 Summary

The architecture on Section 5 describes a system with *Akka* actors integrated in a *Vaadin* web application. Today's story shows a possible implementation of such integration.

7.1.2 Preparation

Task 7.1.1. Create abstract class AkkaUI
`branch: feature/userUI/akkaUI`
`status: done`

Task 7.1.2. Rename MyUI to WelcomeUI
`branch: feature/userUI/akkaUI`
`status: done`

Task 7.1.3. Make WelcomeUI extend AkkaUI
`branch: feature/userUI/akkaUI`
`status: done`

Task 7.1.4. Create class UserUI extends AkkaUI
`branch: feature/userUI/akkaUI`
`status: done`

Task 7.1.5. Merge akkaUI with userUI
`branch: feature/userUI/akkaUI`
`status: todo`

7.1.3 Specification

Task 7.1.6. Determine the communication protocol in form of a session type specification
`branch: feature/task/specification`
`status: todo`

Task 7.1.7. Determine the client-side projection of the communication protocol
`branch: feature/task/specification/client`
`status: todo`

Task 7.1.8. Determine the server-side projection of the communication protocol
`branch: feature/task/specification/server`
`status: todo`

7.1.4 Test

Create tests that asserts about the behaviour expected by the specification, both on client and server sides. This tests should verify that:

Task 7.1.9. All expected messages are received
`branch: feature/task/test/`
`status: todo`

Task 7.1.10. All messages are processed in the order predefined by the session type

Task 7.1.11. If termination is mandatory, assert about termination status

7.1.5 Implementation

Task 7.1.12. Create WelcomeMVCActor, a subclass of MVCActor, as a static[1] inner class of WelcomeUI. This actor will implement the MVC pattern of this architecture:

Task 7.1.13. Implement the communication protocol inside the `onReceive()`, as asynchronously as possible.[2]

Task 7.1.14. Use a *session type based finite state machine* to guide communication dealing with message processing order.

Task 7.1.15. Store incoming messages locally to decide how to proceed and react to them when; messages make the fsm to advance in the session type performing a state transition

Task 7.1.16. Define server-side business actors as BusinessActors

Task 7.1.17. Implement the server-side projection of the asynchronous communication protocol in the `onReceive()` method.

7.1.6 Evolution

[1] Why `static`

[2] Use `tell` and `forward` actor communication patterns and reserve the `ask` communication pattern for special cases.