

MCV Actors
An Akka Experiment

Alexandre Zua Caldeira
zuacaldeira@gmail.com

September 16, 2016

Contents

1	Overview	5
1.1	Introduction	5
1.2	<i>Java</i> , The Tool Not The Target	5
1.3	<i>Vaadin</i> , Alone With Java	5
1.4	<i>Akkaros</i> , MVC Actors	5
1.5	<i>Akkaria</i> , A World To Live In	5
2	The Akka Experiment	7
2.1	Experiment	7
2.1.1	Day 1: September 16, 2016	7

Chapter 1

Overview

1.1 Introduction

1.2 *Java*, The Tool Not The Target

1.3 *Vaadin*, Alone With Java

1.4 *Akkaros*, MVC Actors

1.5 *Akkaria*, A World To Live In

Chapter 2

The Akka Experiment

2.1 Experiment

2.1.1 Day 1: September 16, 2016

Problem summary. The architecture on Section 1.5 describes a system with *Akka* actors integrated in a *Vaadin* web application. Today's story shows a possible implementation of such integration. It's tasks are:

1. Preparation
 - (a) Create abstract class AkkaUI
 - (b) Rename MyUI to WelcomeUI
 - (c) Make WelcomeUI extend AkkaUI
2. Specification
 - (a) Determine the communication protocol in form of a session type specification
 - (b) Determine the client-side projection of the communication protocol
 - (c) Determine the server-side projection of the communication protocol
3. Test and Implementation
 - (a) Create tests that asserts about the behaviour expected by the specification, both on client and server sides. This tests should verify that:
 - i. All expected messages are received
 - ii. All messages are processed in the order predefined by the session type
 - iii. If termination is mandatory, assert about termination status

- (b) Create `WelcomeMVCActor`, a subclass of `MVCActor`, as a static¹ inner class of `WelcomeUI`. This actor will implement the MVC pattern of this architecture:
 - i. Implement the communication protocol inside the `onReceive()`, as asynchronously as possible.²
 - ii. Use a *session type based finite state machine* to guide communication dealing with message processing order.
 - iii. 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
 - iv. Define server-side business actors as `BusinessActors`
 - v. Implement the server-side projection of the asynchronous communication protocol in the `onReceive()` method.
- 4. UIs act as presenters and contains views. Each UI must define a subclass of `MVCActor`, defining one client-side `MVCActor`, that must be provided as a inner class

Goals.

Tasks.

¹Why `static`

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