MCV Actors
An Akka Experiment

Alexandre Zua Caldeira
zuacaldeira@gmail.com

September 16, 2016

Contents

Chapter 1

Overview

1.1 Introduction

1.2 *Java*, The Tool Not The Target

1.3 *Vaadin*, Alone With Java

1.4 *Akka*ros, 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
   `branch: feature/userUI/akkaUI`, `status: done`
   (b) Rename MyUI to WelcomeUI
   `branch: feature/userUI/akkaUI`, `status: done`
   (c) Make WelcomeUI extend AkkaUI
   `branch: feature/userUI/akkaUI`, `status: done`
   (d) Create class UserUI extends AkkaUI
   `branch: feature/userUI/akkaUI`, `status: done`
   (e) Merge akkaUI with userUI
   `branch: feature/userUI/akkaUI`, `status: doing`
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 static[1] 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.

Goals.

Tasks.

[1]Why `static`

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