MVC 1.0 - Hands-on LAB

Christian Kaltepoth

Senior Software Developer, ingenit

@chkal

Ivar Grimstad

Principal Consultant, Cybercom Sweden

@ivar_grimstad

- Introduction
- Part 1 - The Basics
- Part 2 - Core MVC 1.0 Features
- Part 3 - Useful MVC 1.0 Features
- Summary

Part 1
Introduction

Revised Java EE 8 Proposal

- No Change to Plan
- Propose to Drop
- Propose to Add

CDI 2.0 (JSR 365)
- Bootstrap API for Java SE
- Async events
- Observer ordering

JSON-B 1.0 (JSR 367)
- JSON <-> object mapping

JMS 2.1 (JSR 368)
- Flexible JMS MDBs
- Improved XA support

Servlet 4.0 (JSR 369)
- HTTP/2 support

JAX-RS 2.1 (JSR 370)
- Reactive enhancements
- Server-sent events
- Non-blocking I/O
- Client-side circuit breakers

MVC 1.0 (JSR 371)
- Action-based MVC framework

JSF 2.3 (JSR 372)
- Small-scale new features
- Community-driven improvements

Management 2.0 (JSR 373)
- REST-based APIs

JSON-P 1.1 (JSR 374)
- JSON Pointer and Patch
- Java Lambda support

Security 1.0 (JSR 375)
- Authentication/authorization APIs
- OAuth, OpenID support
- Secret management

Bean Validation 2.0 (JSR 380)
- Collection constraints
- Date/Time support
- Community-requested features

Configuration
- Standard for externalizing application configuration

Health Checking
- Standard for client-side health reporting

JavaOne ORACLE

Rationale for Proposed Changes

New Functionality

- Cloud apps make many remote REST calls. Need a **client-side circuit breaker** added to JAX-RS
- Need a **secret vault** because there's no way to do this today using **standards**
- **Need OAuth and OpenID** support because those technologies have rapidly emerged as standards
- Need **externalized configuration store** to make applications retargetable across environments
- Need basic **multi-tenancy** support to accommodate needs of more complex apps and offer higher density
- **Need standard way of health checking** Java-based apps

Dropped Functionality

- **JMS** is no longer very relevant in cloud. Proposed to stay at JMS 2.0 standard (vs. upgrading to JMS 2.1).
- Cloud apps often ship headless, making **MVC** largely irrelevant
- Current **Management** JSR not widely used

JSR #371
Model-View-Controller (MVC 1.0) Specification
Transfer Ballot

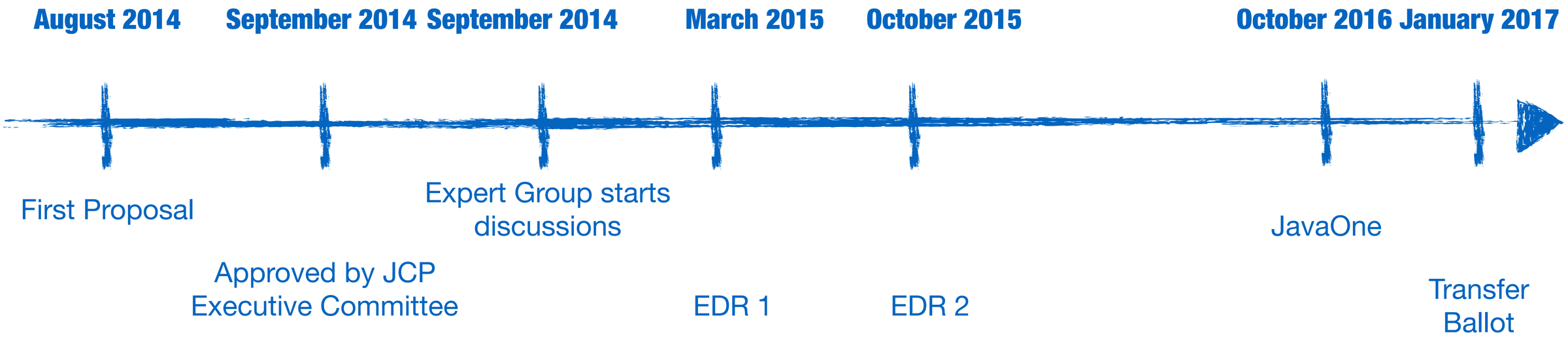
Ballot duration: 2017-01-17 to: 2017-01-30

Special Vote Instructions:

Santiago Pericasgeertsen, and Manfred Riem, Co-Sped. Leads, representing Oracle, request a JSR Transfer Ballot of JSR 371: Model-View-Controller (MVC 1.0) Specification, to Ivar Grimstad, an individual.

@Controller
to the Community !

History


August 2014
September 2014
September 2014
March 2015
October 2015
October 2016
January 2017
First Proposal
Approved by JCP Executive Committee
Expert Group starts discussions
EDR 1
EDR 2
JavaOne
Transfer Ballot

History

January 2017

Transfer Ballot

April 2017

Transfer Complete

May 2017

Christian co-spec lead

May 2017

Apache License 2.0

June 2017

Infrastructure Complete

October 2017

Intention announced

Recent Activities

Transfer approved by the Executive Committee ✓

Thanks!

Finalize Transfer ✓

Done!

New Infrastructure Setup ✓

Done!

Licensing ✓

Done!

Ongoing Activities

Formally move infrastructure from java.net ✓
Done!

Bring in Christian Kaltepoth as co spec-lead ✓
Done!

Revise the Schedule ✓

Adopt-a-JSR

THE APACHE® SOFTWARE FOUNDATION

Apache License 2.0

Adopt-a-JSR

Write Code!

Give us Feedback

Blog

Tweet

Create a Logotype ✓

Secure | https://www.mvc-spec.org

JSR 371: Model-View-Controller Specification

MVC 1.0 | Home | News | Specification | Learn | Contribute | Ozark (RI) | JCP | Logos and Artwork

Overview

Model-View-Controller, or MVC for short, is a common pattern in Web frameworks where it is used predominantly to build HTML applications. The model refers to the application's data, the view to the application's data presentation and the controller to the part of the system responsible for managing input, updating models and producing output.

Web UI frameworks can be categorized as action-based or component-based. In an action-based framework, HTTP requests are routed to controllers where they are turned into actions by application code; in a component-based framework, HTTP requests are grouped and typically handled by framework components with little or no interaction from application code. In other words, in a component-based framework, the majority of the controller logic is provided by the framework instead of the application.

The API defined by this specification falls into the action-based category and is, therefore, not intended to be a replacement for component-based frameworks such as JavaServer Faces (JSF), but simply a different approach to building Web applications on the Java EE platform.

The MVC API is layered on top of JAX-RS and integrates with existing EE technologies like CDI and Bean Validation.

HelloWorld Example

```
@Path("hello")
public class HelloController {

    @Inject
    private User user;

    @GET
```

Latest news

2017-11-28

New MVC 1.0 Logo!

We have a new Logo for MVC 1.0!

read more

2017-05-17

Apache License, Version 2.0 Selected

The Specification, JavaDoc, API, RI and TCK will be licensed using The Apache License 2.0 (https…

read more

2017-04-22

Transfer Process Complete

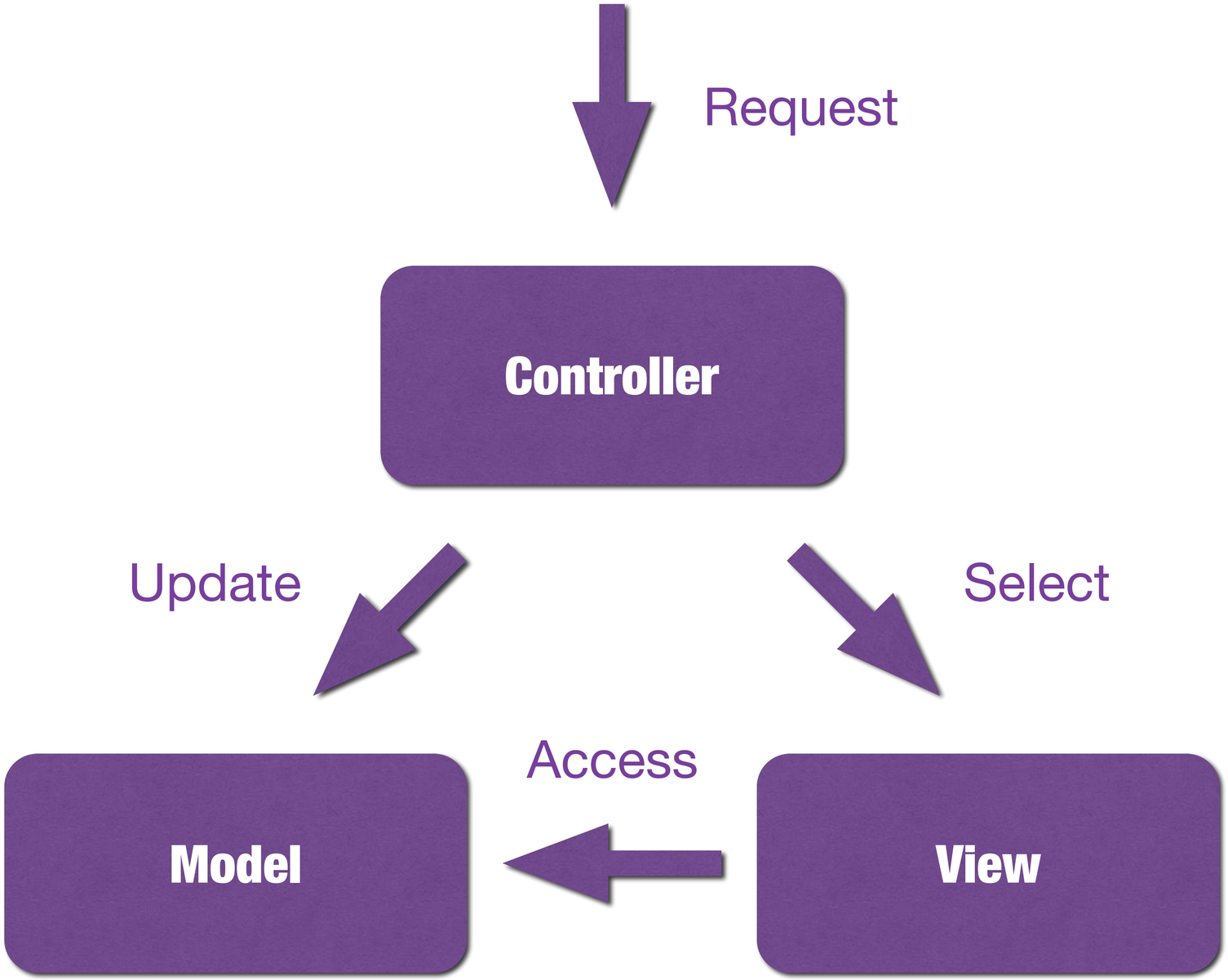
The transfer of JSR 371 from Oracle is now complete!

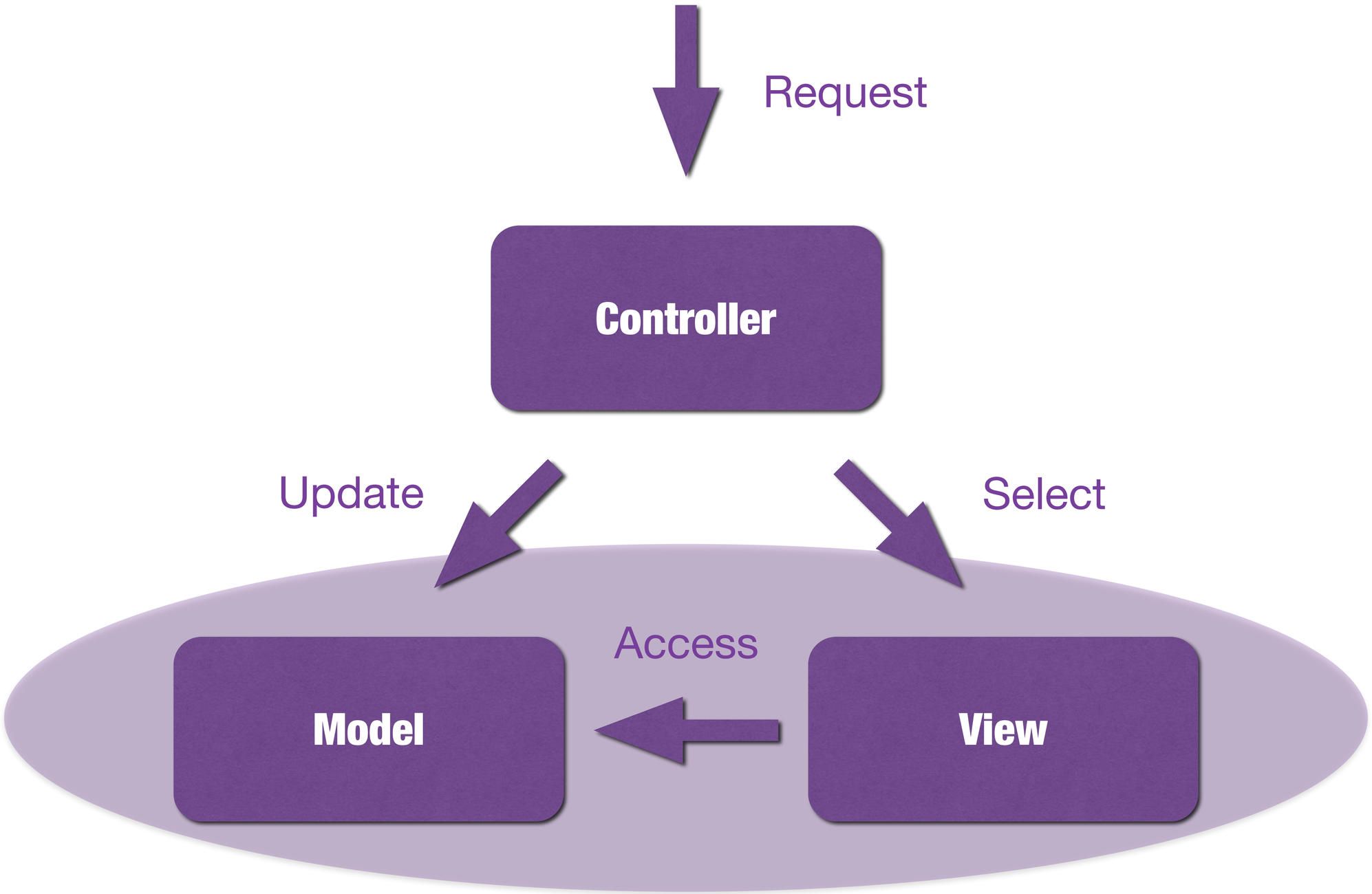
read more

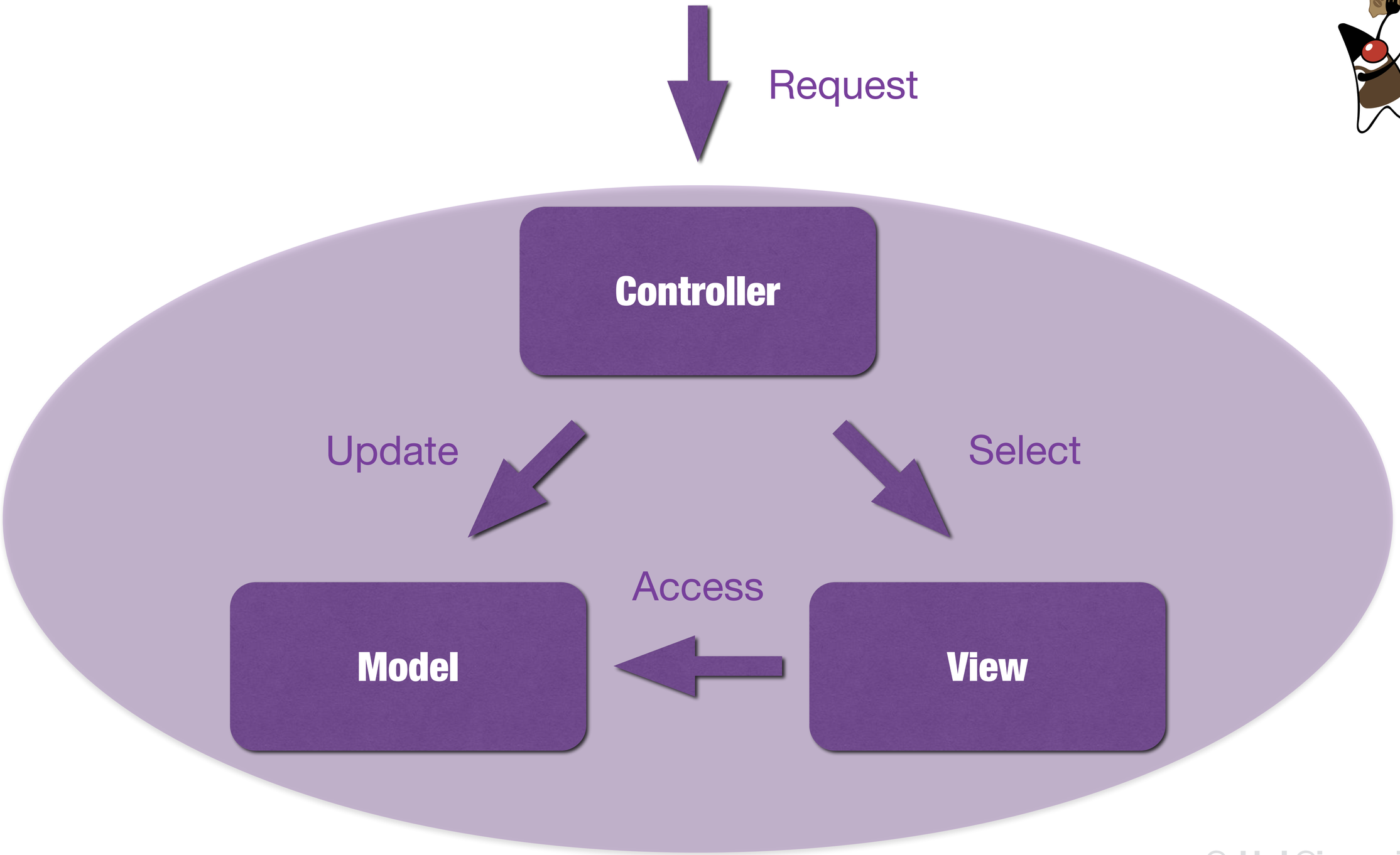
MVC 1.0 - The Basics

@chkal @ivar_grimstad

Action-based MVC


Request
Controller
Update
Select
Access
Model
View


Request
Controller
Update
Select
Access
Model
View


Request
Controller
Update
Select
Access
Model
View

Existing Java EE Technologies

Key Decisions

Key Decision

Build MVC 1.0 on top of JAX-RS

@chkal @ivar_grimstad

Controllers

Controller

```
public class HelloController {



}
```

Controller

```
@Path(“hello”)
public class HelloController {



}
```

Controller

```
@Controller
@Path(“hello”)
public class HelloController {



}
```

Views

View

```
@Controller
@Path(“hello”)
public class HelloController {



}
```

View

```
@Controller
@Path("hello")
public class HelloController {




    @GET
    public String hello() {
        return "hello.jsp";
    }
}
```

View

```
@Controller
@Path("hello")
public class HelloController {




    @GET
    public Response hello() {
        return Response.status(OK).entity("hello.jsp").build();
    }
}
```

View

```
@Controller
@Path("hello")
public class HelloController {



    @View("hello.jsp")
    @GET
    public void hello() {

    }
}
```

View

```
@View("hello.jsp")
@Controller
@Path("hello")
public class HelloController {




    @GET
    public void hello() {

    }
}
```

Models

Model

```
@View("hello.jsp")
@Controller
@Path("hello")
public class HelloController {




    @GET
    public void hello() {

    }
}
```

Model

```
@View("hello.jsp")
@Controller
@Path("hello")
public class HelloController {

    @Inject
    private Models model;

    @GET
    public void hello() {
      model.put("message", "Hello Cologne!");
    }
}
```

Model

```
<%@page contentType="text/html"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
   <head>
     <title>MVC 1.0 Hello Demo</title>
   </head>
   <body>
      <h1>Hello ${greeting}</h1>
   </body>
</html>
```

Part 1

https://github.com/ivargrimstad/mvc-hol