

RESTful HTTP on the JVM

Martin Eigenbrodt (@eigenbrodtm)

Stefan Tilkov (@stilkov)

innoQ

Intro

3 Programming Languages

3 Frameworks

2 Library Collections

60 Minutes

Showcase common tasks

Highlight strengths & weaknesses



Java / Scala

Full Stack

Async

Akka based



Java

JSR-339/JAX-RS 2.0 Implementation

Annotation-driven



Clojure

Ring, Clout, Compojure

It's just functions, so no logo



Liberator (formerly “compojure-rest”)

Inspired by Webmachine (Erlang)

Status machine for correct HTTP



Scala

Library

Embedded HTTP server

Async, Akka-based

Internal routing DSL

Simple Path Matching

Map path and method to code

Map return value to response



```
@Path("/paths")
public class Paths {
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    @Path("simple")
    public String simple() {
        return "Simple";
    }
}
```



```
# routes file
GET      /paths/simple      controllers.Paths.simple

object Paths extends Controller {
  def simple = Action {
    Ok ("Simple")
  }
}
```



(Ring + Clout)

```
(defn hello-world-app [req]
  {:status 200
   :headers {"Content-Type" "text/plain"}
   :body "Hello, World!"})

(hello-world-app {:uri "/foo"
                  :request-method :get})

> {...}
```



(Ring + Clout)

```
(defroutes  
  (GET "/paths/simple" [] hello-world-app))
```

```
(defroutes  
  (GET "/paths/simple" [] "Simple"))
```



```
class MyServiceActor extends HttpServiceActor
{
  def receive = runRoute(myRoute)

  def myRoute = pathPrefix("paths") {
    (path("simple")) {
      get {
        complete {
          "Simple"
        }
      }
    }
  }
}
```

Spray Route

Route = RequestContext => Unit

Directive = (o..n => Route) => Route

Bind dynamic path elements

Map path with Placeholder to code

`/users/{id}`



```
@Path ("twice/{value}")
```

```
@GET
```

```
public int twice (@PathParam ("value") int value) {  
    return value * 2;  
}
```



```
GET /paths/twice/:id  Controllers.Paths.twice(id : Int)
```

```
def twice (i : Int) = Action {  
  Ok((i*2).toString())  
}
```



(Ring + Clout)

```
(defroutes binding-routes
  (GET ["/paths/twice/:value",
        :value #"[0-9]+" ]
    [value]
    (str (* 2 (Integer/parseInt value))))))
```



```
path("twice" / IntNumber) { int =>
  get {
    complete {
      (int * 2).toString
    }
  }
}
```

Server-side link creation

Need to link to a resource

But should be DRY



```
// specific Resource
URI uri = UriBuilder.
    fromResource(Paths.class).
    path(Paths.class, "twice").
    resolveTemplate("value", 21).build();

// templated Link
URI uri = UriBuilder.
    fromResource(Paths.class).
    path(Paths.class, "twice").build();
```



```
# specific Resource
controllers.routes.Paths.twice(21).absoluteURL()

# Template
controllers.routes.Paths.twice(1234)
    .absoluteURL().toString
    .replaceAllLiterally("1234", "{value}")
```



(Route-one)

```
(defroute twice "/paths/twice/:value")

(defroutes binding-routes
  (compojure/GET [twice-template, :value #"[0-9]+" ] [value]
    (str (* 2 (Integer/parseInt value)))))

(defroutes link-routes
  (GET "/links/toTwice21" [:as r]
    (str (build-base-uri r) (twice-path {:value 21})))
  (GET "/links/toTwice" [:as r]
    (str (build-base-uri r)
      (route-pattern-to-uri-template twice-template))))
```



(Route-one)

```
(macroexpand-1 '(defroute twice "/paths/twice/:value"))

(do
  (def twice-template "/paths/twice/:value")
  (defn twice-path [& r]
    (path-for "/paths/twice/:value"
              (if (= 1 (count r)) (first r) (apply hash-map r))))
  (defn twice-url [& r]
    (url-for "/paths/twice/:value"
             (if (= 1 (count r)) (first r) (apply hash-map r)))))

(defn route-pattern-to-uri-template [pattern]
  (clojure.string/replace pattern #" :([\w]*)" "{$1}"))
```



Build it yourself

Manage conditional requests

Client may request with
“If-None-Match”



```
@Path ("etag")
@GET
public Response etag(@Context Request request) {
    String entity = "This is the entity.";
    EntityTag eTag = new EntityTag(entity);
    ResponseBuilder rb =
        request.evaluatePreconditions(eTag);
    if (rb != null) {
        return rb.build();
    }
    return Response.ok(entity).tag(eTag).build();
}
```



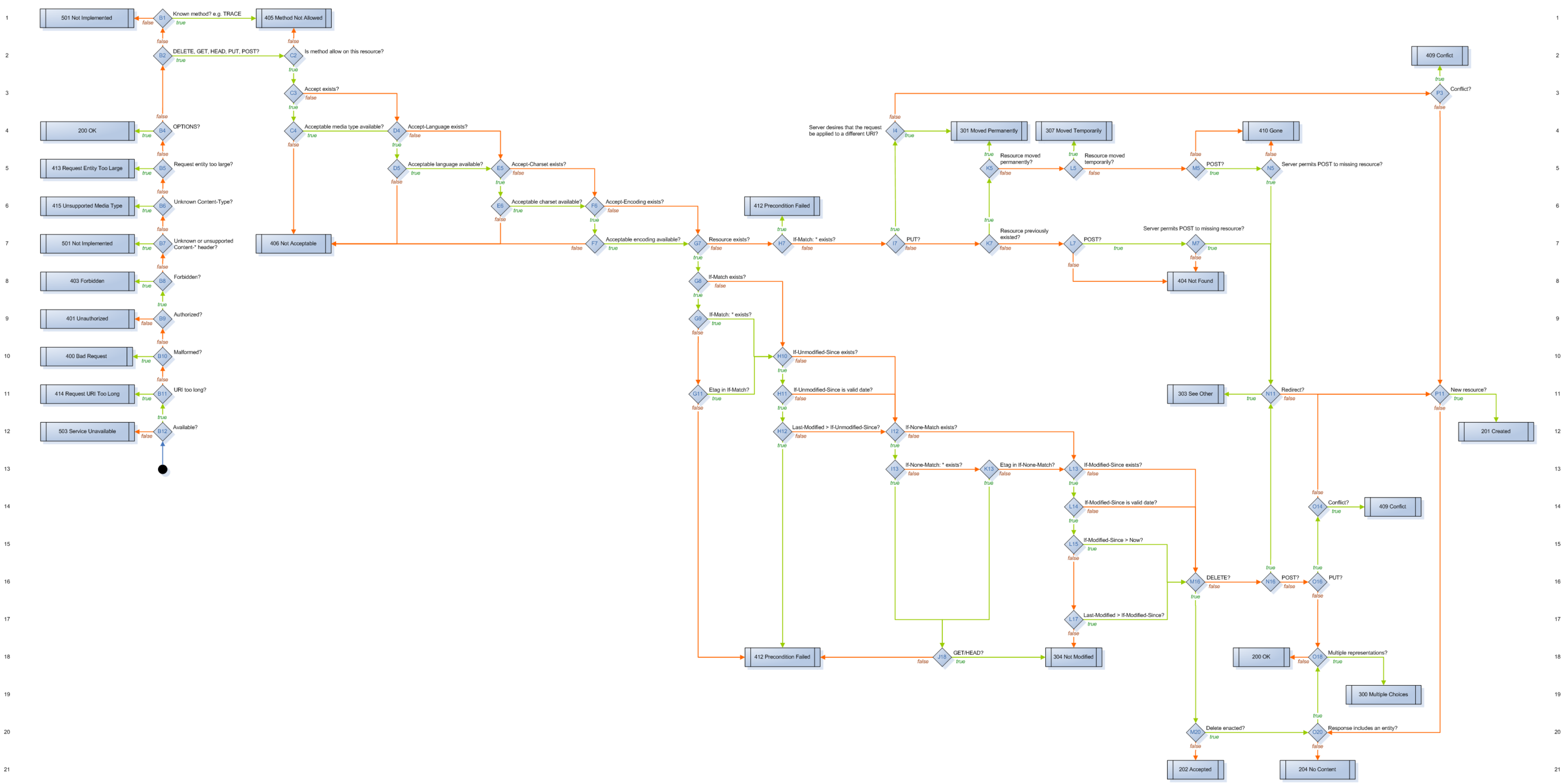
Nada



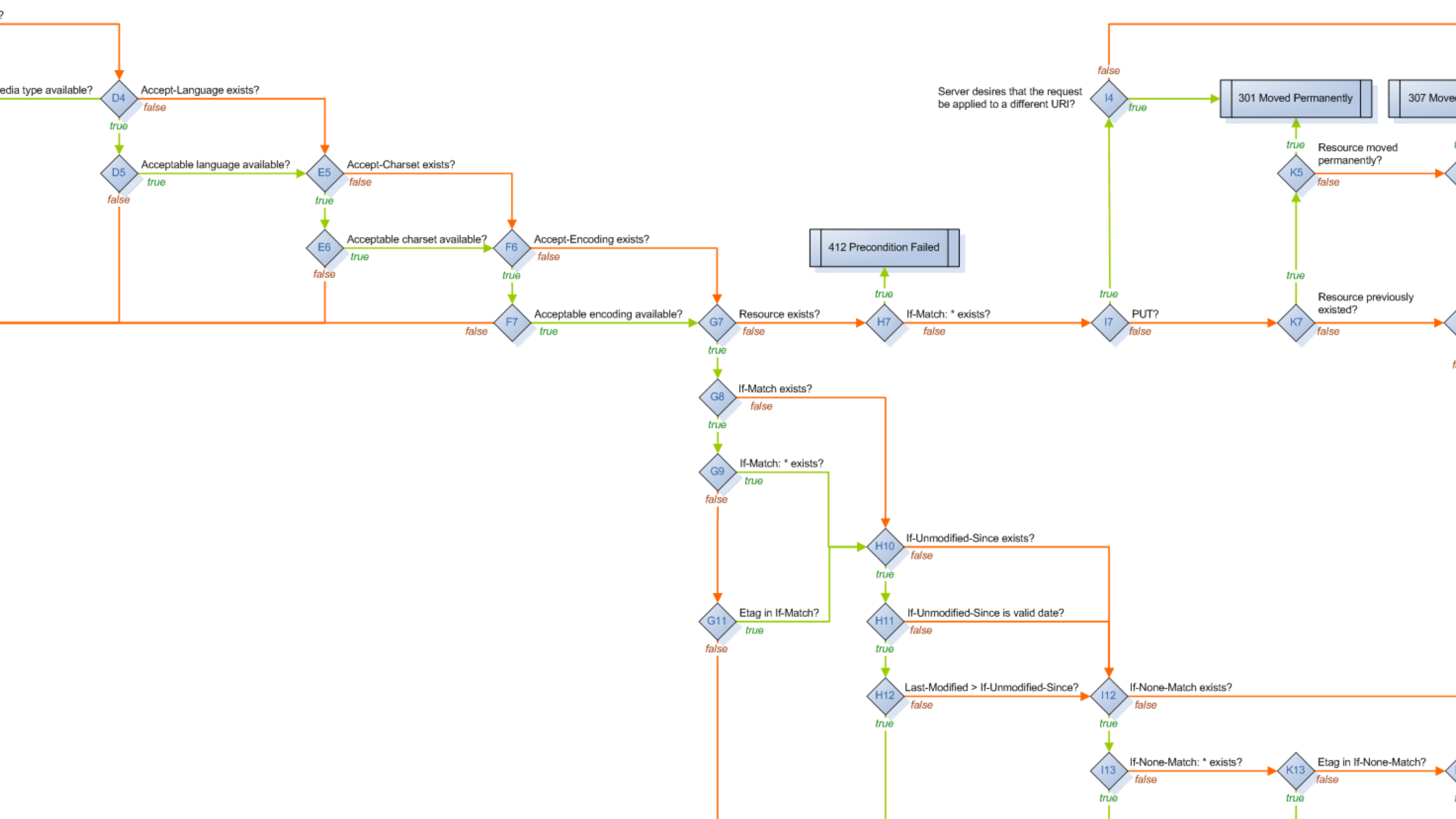
```
(defroutes caching-routes
  (ANY caching "/caching/etag" []
    (let [result "Some business entity"]
      (resource :available-media-types ["text/plain"]
        :etag (md5 result)
        :handle-ok {:data result}))))))
```



v3



<https://code.google.com/p/http-headers-status/>
<https://github.com/for-GET/http-decision-diagram>





```
( (path("etag") & get) {  
  val entity = "This is the entity."  
  val etag = EntityTag(entity)  
  conditional(etag, changeDate) {  
    complete { entity }  
  }  
})
```

Recursive routing

**“eat” first segment of a path and pump rest
back into the routing engine**



```
// In some Controller
@Path("tree")
public SubResource tree() {
    // Could pass parameters here
    return new SubResource();
}

public class SubResource {
    @GET public String done() {...}

    // Recursion!
    @Path("{element}")
    public SubResource sub(@PathParam("element") String name) {
        // Perhaps do something with "name" or pass it ..
        return new SubResource();
    }
}
```



(Ring + Clout)

```
(defroutes path-routes
  (GET dynamic "/paths/tree/*" [& more]
    (str "Content of /paths/tree/" (:* more))))
```



Sub resource possible with additional routes file

no recursion

one could implement the “Routes” trait



```
pathPrefix("tree") {  
  subResource("/tree")  
}  
  
def subResource(name: String): Route = {  
  pathEndOrSingleSlash {  
    get {  
      complete {  
        s"Content of ${name}"  
      }  
    } ~  
    pathPrefix(Segment) { seg =>  
      subResource(s"${name}/${seg}")  
    }  
  }  
}
```

Content negotiation and marshalling

**Return correct Content based on accept
header**

Create response from domain object



```
@GET  
@Produces ({MediaType.APPLICATION_JSON,  
MediaType.APPLICATION_XML})  
public Data getData() {  
    Data data = new Data();  
    data.data = "Hello World";  
    return data;  
}  
  
@XmlRootElement  
public class Data {  
    @XmlValue  
    public String data;  
}
```



```
(defmethod representation/render-map-generic "application/xml"  
  [data context]  
  (emit-str  
    (map (fn [k] (element k {} (get data k))) (keys data))))))
```

```
(defroutes contneg-routes  
  (ANY contneg "/contneg" []  
    (resource :available-media-types ["application/json"  
                                     "application/xml"]  
             :handle-ok {:data "Hello World"})))
```



```
def get = Action { implicit request =>
  render {
    case Accepts.Xml () =>
      Ok (<data>Hello World</data>)
    case Accepts.Json () =>
      Ok (Json.obj ("data" -> "Hello World"))
  }
}
```



```
(path("conneg")) {  
  get {  
    complete { Data("Hello World") }  
  }  
}
```

```
object Data {  
  ...  
  implicit val marshaller =  
    Marshaller[Data] { (data, ctx) =>  
      ctx.tryAccept(supportedContentTypes) match {  
        // Delegate  
        case Some(Json) => jsonMarshaller(data, ctx)  
        case Some(Xml) => xmlMarshaller(data, ctx)  
        case _ => ctx.rejectMarshalling(supportedContentTypes)  
      }  
    }  
}
```

Summary

Code generation

Reflection

Functions & Composition

Type Classes

Macros

Decision FSM

Thank you!
Questions?
Comments?

Martin Eigenbrodt, @eigenbrodtm
martin.eigenbrodt@innoq.com
Phone: +49 0171 3333 085

Stefan Tilkov, @stilkov
stefan.tilkov@innoq.com
Phone: +49 170 471 2625



innoQ Deutschland GmbH
Krischerstr. 100
40789 Monheim am Rhein
Germany
Phone: +49 2173 3366-0

Ohlauer Straße 43
10999 Berlin
Germany
Phone: +49 2173 3366-0

Robert-Bosch-Straße 7
64293 Darmstadt
Germany
Phone: +49 2173 3366-0

Radlkofersstraße 2
D-81373 München
Germany
Telefon +49 (0) 89 741185-270

innoQ Schweiz GmbH
Gewerbestr. 11
CH-6330 Cham
Switzerland
Phone: +41 41 743 0116