Domain Driven REST web-services

Felipe Gaúcho

fgaucho@gmail.com

Jfokus 2010, Stockholm/Sweden

“a server side presentation”

Domain Driven REST web-services

Felipe Gaúcho

fgaucho@gmail.com

Senior Software Engineer @ Netcetera AG (Switzerland)

JUG Advisor & Open Source Evangelist

Using Java since 1996

Using Glassfish since 2005

Owner & committer of the project “Arena PUJ”

Agenda

Data bridge between

the relational database and the HTTP interface

- **Object-Relational** impedance mismatch (Java ↔ SQL)
- **Object-Media Type** impedance mismatch (Java ↔ JSON, JSONP, XML, ?)

Agenda

- The Arena PUJ Project (quick overview)
- Arena PUJ Architecture
- Object-Relational mapping with JPA2
- Object-Mime Type mapping with JAXB
- Exposing a Java domain model with JAX-RS (Jersey Framework)
- Next step: hateoas

Arena PUJ Project

Prêmio **U**niversitário **J**ava

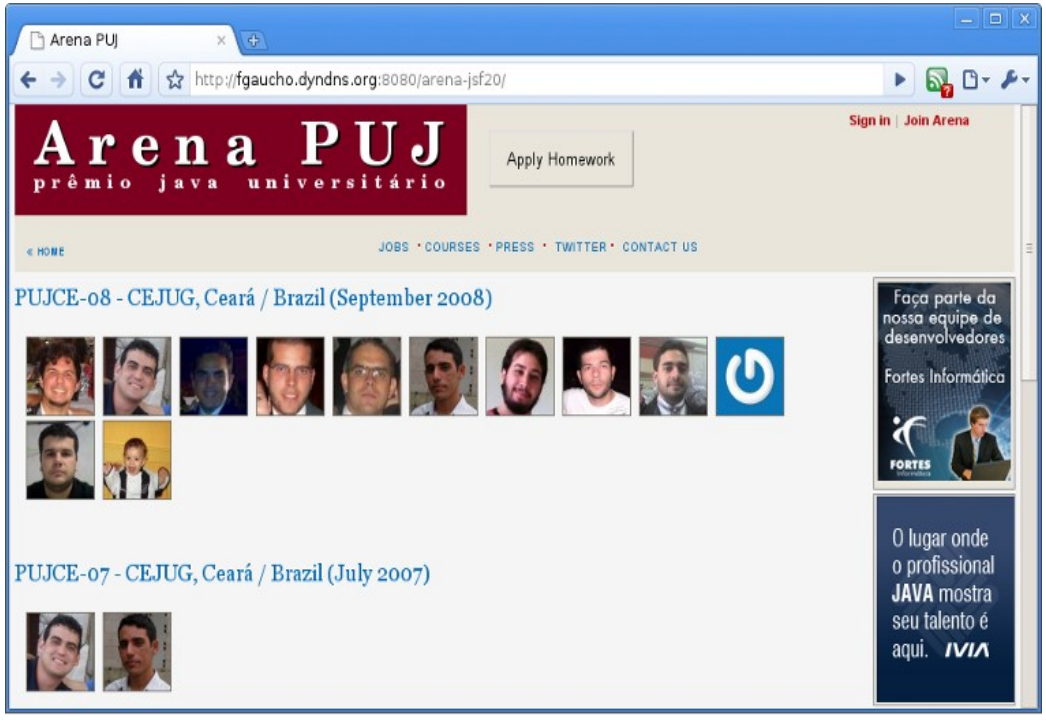
- Java Prize for University Students -

An academic competition in which students homeworks are evaluated by senior Java professionals.

kenai.com/projects/puj

“A CEJUG Initiative – please copy this idea in your local JUG”

Arena Quick Demo



Arena Quick Demo

- JSF 2.0: http://fgaucho.dyndns.org:8080/arena-jsf20/
- DWR: http://fgaucho.dyndns.org:8080/arena-dwr/
- Vaadin: http://fgaucho.dyndns.org:8080/arena-vaadin/
- Atom 1.0 http://fgaucho.dyndns.org:8080/arena-http/atom
- WADL: http://fgaucho.dyndns.org:8080/arena-http/application.wadl
- Hudson: http://fgaucho.dyndns.org:8080/hudson/
- Dev wiki: http://kenai.com/projects/puj/pages/Arena-dev

To checkout the source code from the Mercurial repository:

hg clone https://kenai.com/hg/puj~arena

Domain Driven REST web-services

First things first

“show me the code”

You start with a simple POJO

```
public class User {
    private String login;
    private String password;
}
```

Make your POJO persistable with JPA annotations

```
@Entity
public class User {

    @Id
    private String login;

    @Column(length = 32)
    private String password;

}
```

```
create table User (
        login varchar(255) not null,
        password varchar(32),
        primary key(login)
);
```

The EntityManager is a generic DAO

```
@Stateless
public class UserFacade {

        @PersistenceContext
        protected EntityManager manager;

        public User write( User entity ) {
                manager.persist(entity);  // 1 line of code
                return entity;
        }

        public User read( Object primaryKey ) {
                return manager.find(type, primaryKey);   // 1 line of code
        }

}
```

@ManagedBean

A ManagedBean supports a small set of basic services such as resource injection, lifecycle callbacks and interceptors.

Make your POJO serializable with JAXB annotations

```
@XmlRootElement
@Entity
public class User {

    @XmlAttribute
    @Id
    private String login;

    @XmlElement
    @Column(length = 32)
    private String password;

}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<user login="fgaucho">
     <password>secret</password>
</user>
```

jersey.dev.java.net

Jersey serializes the JAXB objects in
XML and JSON formats out of the box.

It uses the Accept header
to select the response format.
(default is XML)

HTTP resources with Jersey

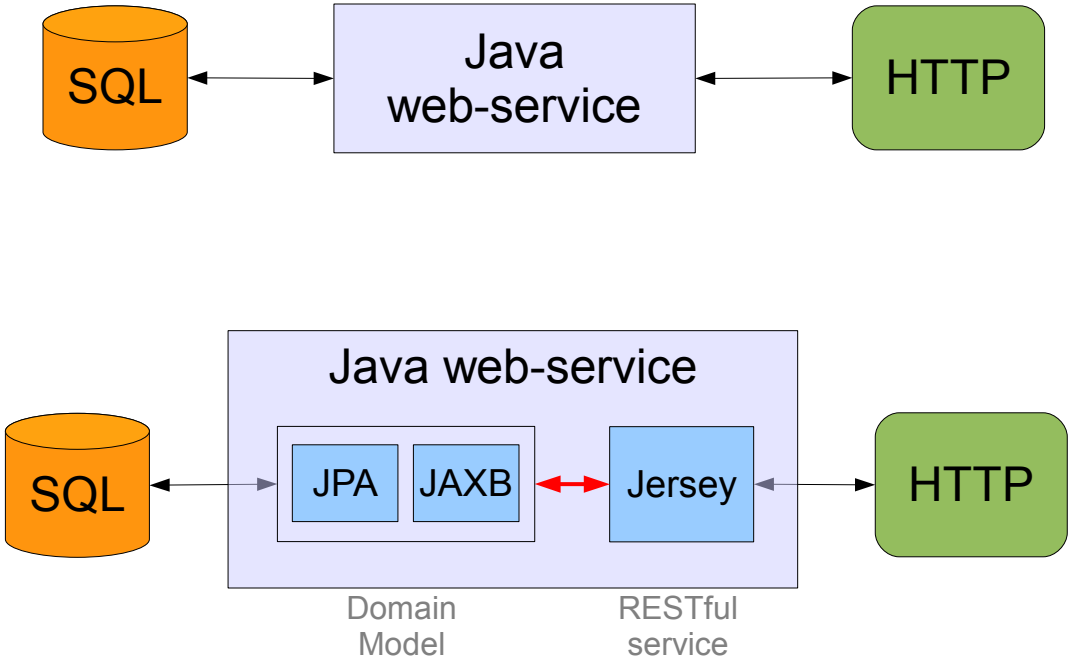
```
@Path("user")
public class PujUserResource {

    @PUT                                          // PUT http://server/app/user ...
    @Produces( “application/json” )
    public User write(User entity) {
        return facade.write(entity);
    }

    @GET                                          // GET http://server/app/user/fgaucho
    @Produces( “application/json” )
    @Path("{id}")
    public User read(@PathParam("id") String login) {
        return facade.read( (Object) login);      // 1 line of code
    }

}
```

The HTTP bridge made simpler



Domain Driven REST web-services

Where is the Business Layer ?

“and validation, security, logging, ...”

Domain Driven REST web-services

- Code a Business Layer
  - Prefer a Generic @Stateless DAO
- Deploy to a Java EE 6 container
  - Security
  - Dependency Injection
  - Load balance
  - HTTP monitoring & logging
  - Resources management: JDBC, JavaMail, JMS.

“Trade off between abstraction and its costs”

Domain Driven REST Design

JPA
JAXB

- Transaction
- Optimistic Locking
- Bean validation
- Domain Model
- HTTP Representation

@Stateless
DAO

- Business logic
- Hypermedia workflow

JAX-RS

- Format Validation
- HTTP Headers
- Response codes

Glassfish v3

- Security (authentication & authorization)
- Data Source, JavaMail, JMS
- Logging & Monitoring

Domain Driven REST web-services

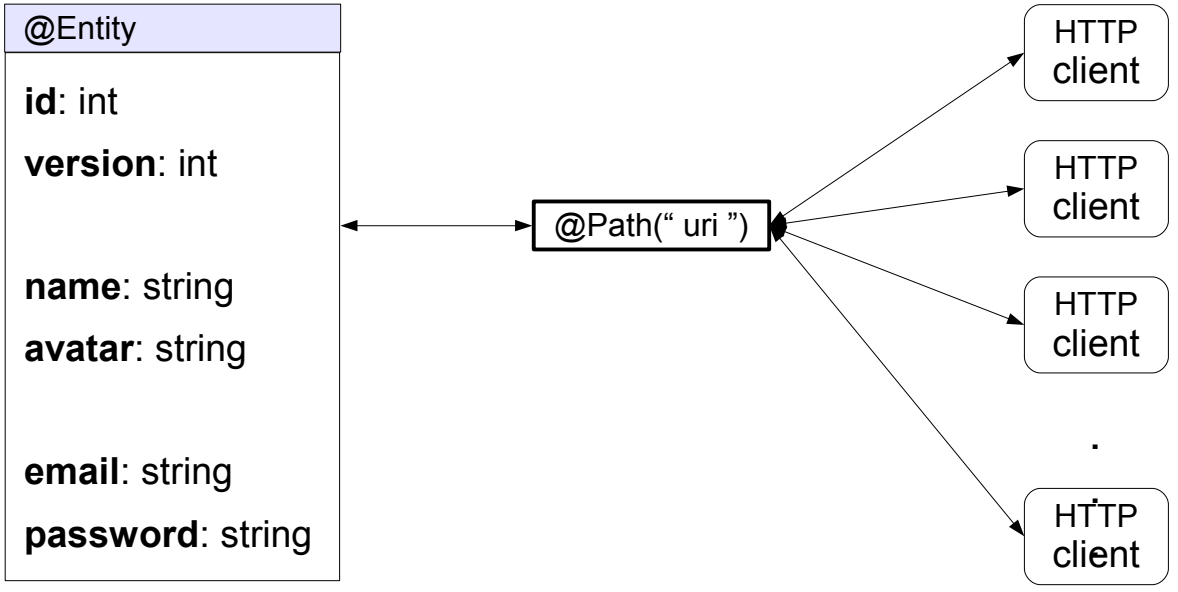
How about security ?

“I don't want to expose the whole database”

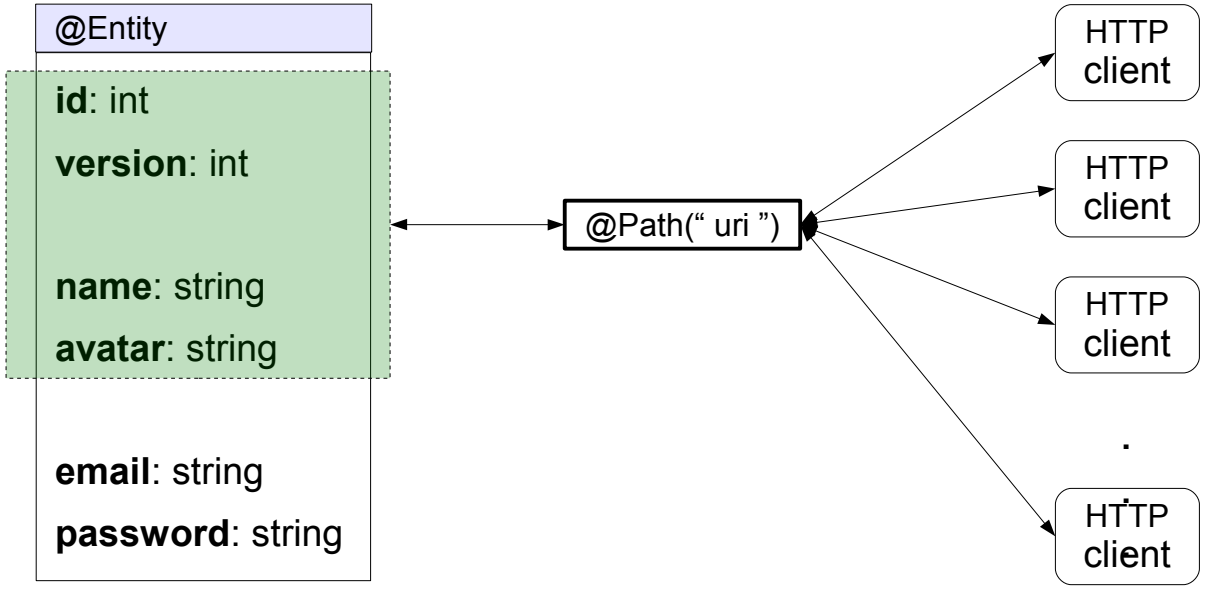
Data encapsulation

“which table column is visible to which client”

All data exposed on the web



Hidden fields



Hidden fields

```
@XmlRootElement
@Entity
public class User {

    @XmlAttribute
    @Id
    private String login;

    @XmlTransient
    @Column(length = 32)
    private String password;

}
```

“static and transient fields are @XmlTransient by default”

To hide fields on the Resource Representation

@XmlTransient

```
@XmlTransient
@Column
private String password;
```

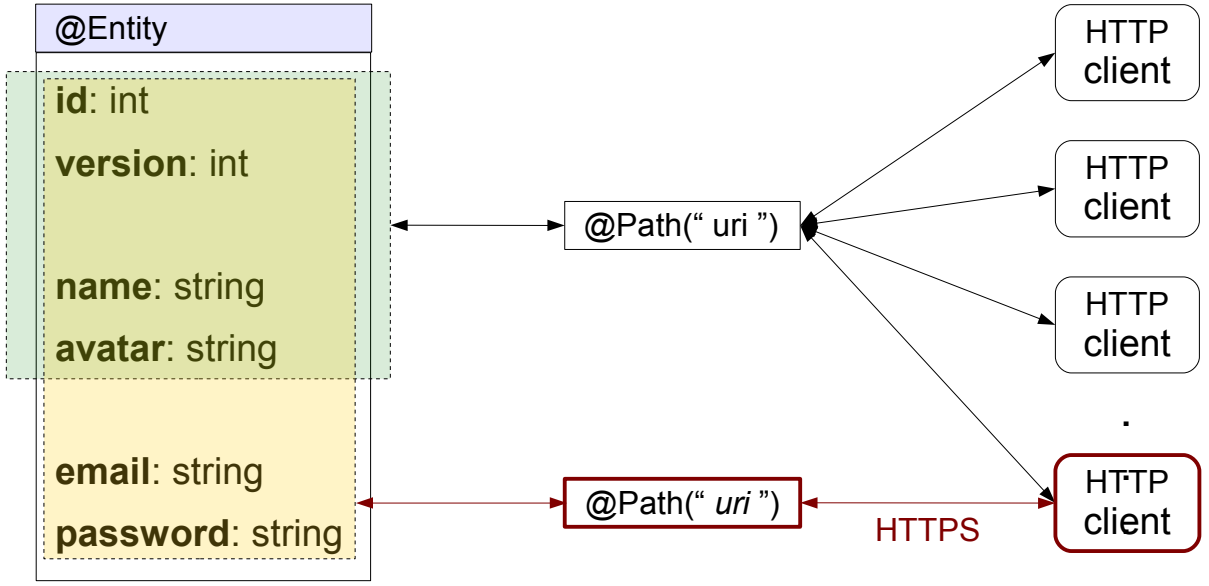
Issues:

- HTTP request with partial dataset
- Partial entity cannot be persisted

Mitigation:

- Use JPA @NamedQuery to update the data

Different views per HTTP resource



Different views per HTTP resource

```
@XmlRootElement
@Entity
public class User {

    @XmlAttribute
    @Id
    private String login;

    @XmlTransient
    @Column(length = 32)
    private String password;

}
```

2 views

```
@XmlRootElement
@Entity
public class User {
    @XmlAttribute
    @Id
    private String login;
}
```

```
@XmlRootElement
@Entity
public class UserDetail extends User {
    @XmlElement
    @Column(length = 32)
    private String password;

}
```

“Discriminator column (DTYPE)”

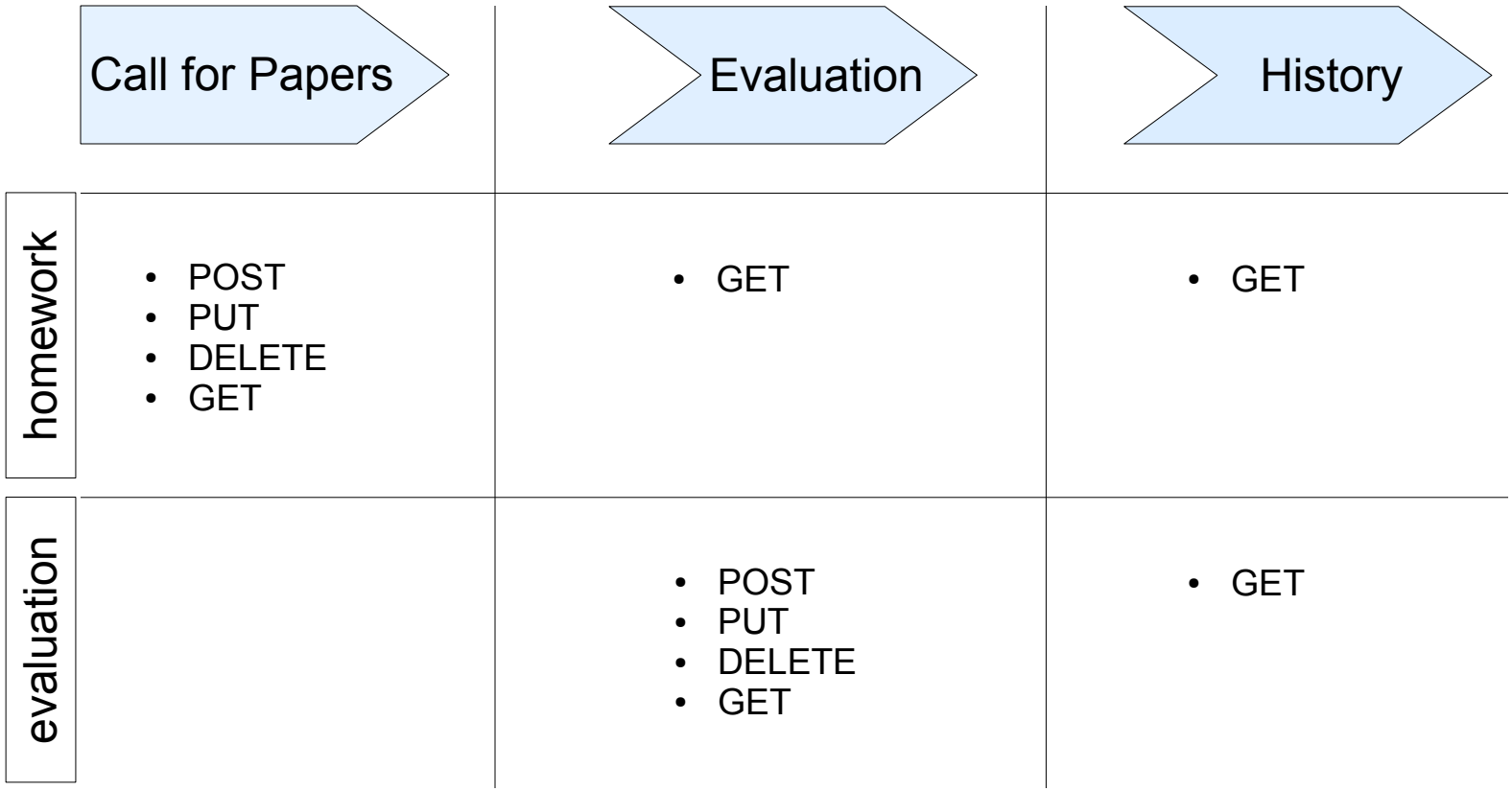
Domain Driven REST web-services

What about HATEOAS ?

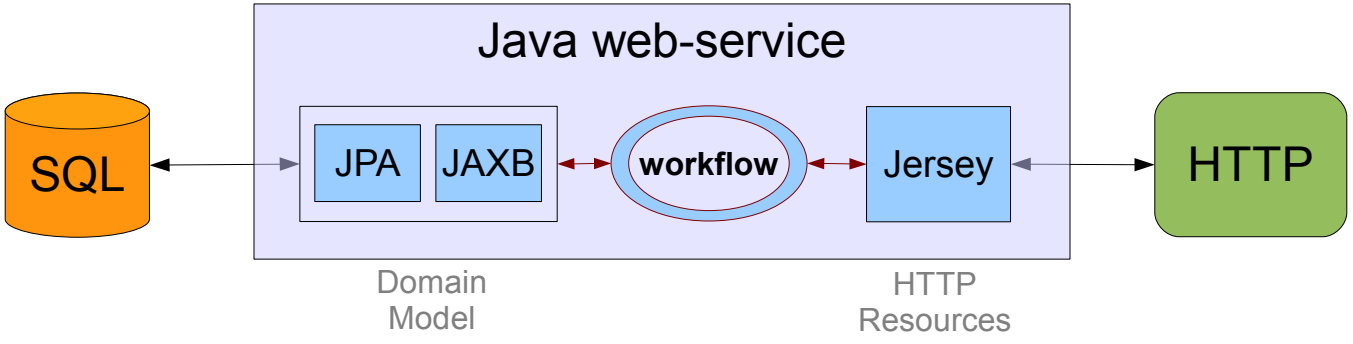
“Hypermedia as the Engine of the Application State”

HATEOAS

PUJ Competition phases (Application States)



HATEOAS



- DB insert → add hyperlinks
- DB delete → remove hyperlinks
- DB update → update hyperlinks

Example:

```
@Override
public PujCompetitionEntity create(PujCompetitionEntity entity) throws Exception {
        Collection<PujLinkEntity> links = entity.getLink();
        links.add(new PujLinkEntity("/" + entity.getName(), "itself"));
        return super.create(entity);
}
```

HATEOAS

There is no framework today for supporting 100% REST in the Java Platform.

Java EE 6 + a bit of creativity can lead us to such a framework.

to be continued...

Thanks for coming !

Arena PUJ
CEJUG Open Source Project

project leader
Felipe Gaúcho

developers
Leonardo Sombriks
Felipe Gaúcho

community collaboration
CEJUG, RSJUG, SOUJAVA, GOJAVA, RIOJUG

special thanks
Glassfish team @ SUN Microsystems,
Paul Sandoz & Jersey mailing list,
FishCat team

And many other contributors that made Arena PUJ possible.
You can learn with this project,
just checkout the code and have fun !

Arena PUJ Architecture