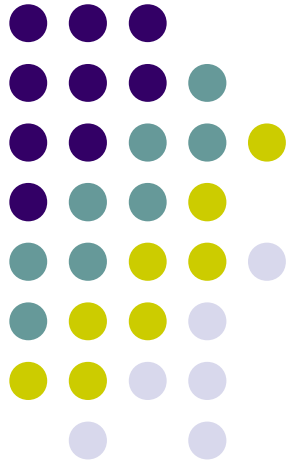
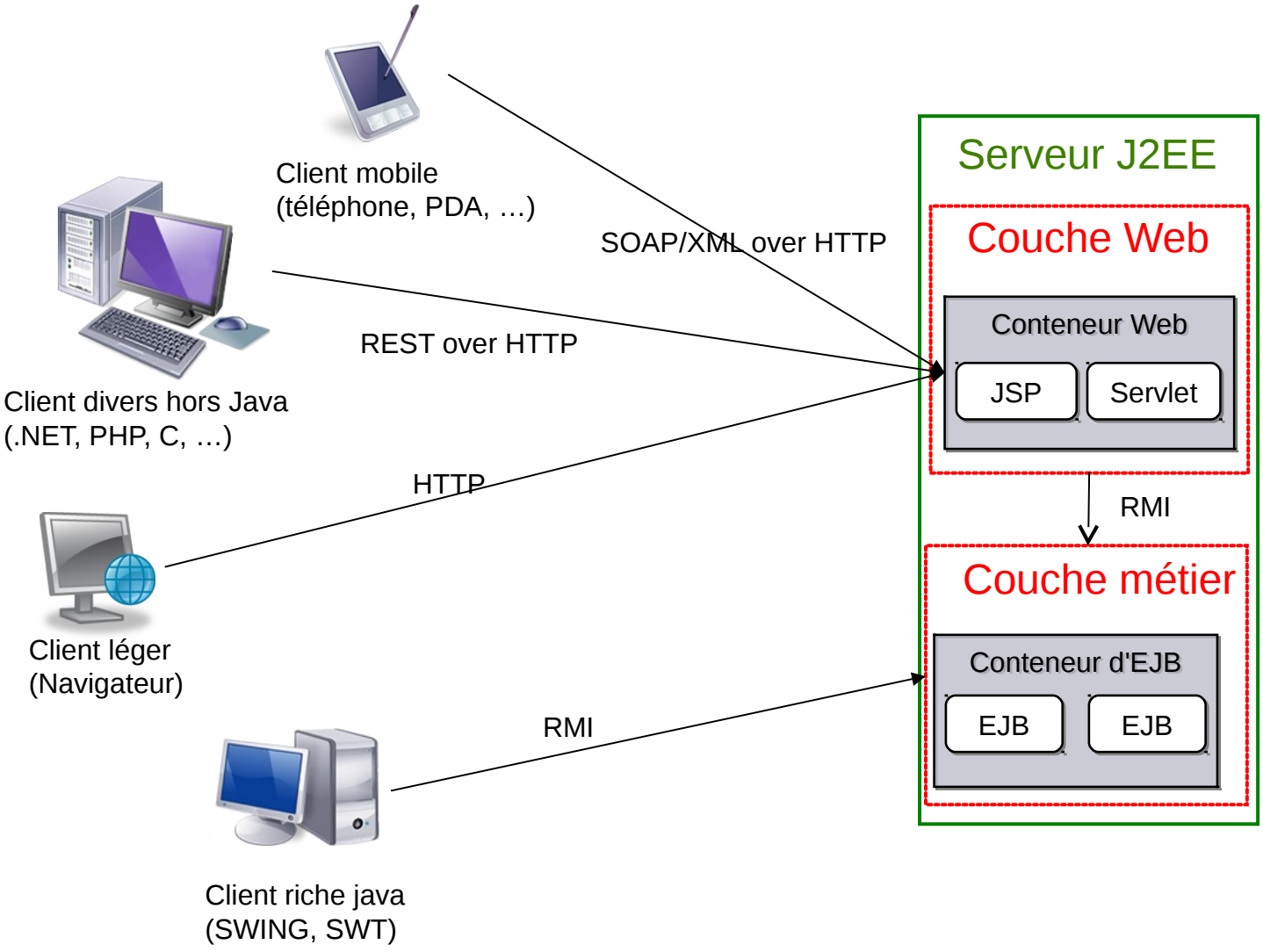
Architecture des systèmes d'information répartis

Representational State Transfer
REST

Sorina Ionica



Les web services … ?



REST

- Architecture orientée ressource proposé

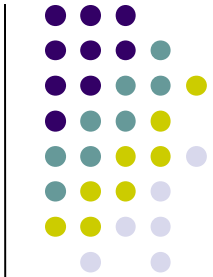
par Ray Fielding (le co-auteur d'Apache)

- Basée sur HTTP, les URIs, les liens
- Mecanisme analogue à une RPC (remote procedure call) mais beaucoup plus simple

REST

- Ray Fielding a définit les régles à respecter pour une architecture REST
- Les ressources sont identifiées par des URI
  - http://nirvana.com/prophetes identifie la liste des prophètes
  - http://nirvana.com/prophetes/13 identifie le 13-ème prophète
  - http://nirvana.com/prophetes/brian identifie le prophète brian
  - http://nirvana.com/prophetes/13/propheties identifie la liste des prophéties du 13-ème prophète

REST



Les ressources sont manipulées au travers des représentations

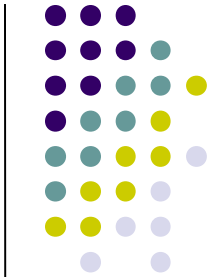
-une ressource peut avoir des représentations dans des formats divers : XML, CSV, JSON, HTML, CSV, ...

-Le client peut définir le(les) format de réponse souhaitée via l'entête Accept.

-Exemple :

```
GET /prophetes/13 HTML/1.1
Host: nirvana.com
Accept: application/xml
```

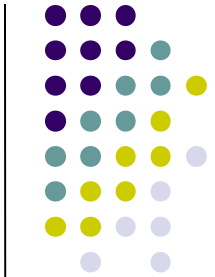
REST



Accéder aux ressources par 4 verbes HTTP correspondant aux 4 opérations CRUD :

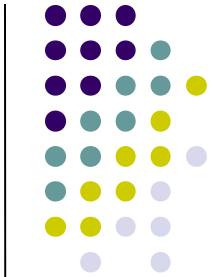
- Créer (create) => POST
- Afficher (read) => GET
- Mettre à jour (update) => PUT
- Supprimer (delete) => DELETE

Exemple



- GET http://nirvana.com/prophetes lit la liste des prophétes
- DELETE http://nirvana.com/prophetes/13 supprime le 13-ème prophéte
- POST http://nirvana.com/prophetes+body des données pour créer un prophéte
- PUT http://nirvana.com/prophetes/13 + body des donnes pour modifier le 13-ème prophéte

API Jersey



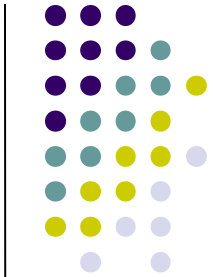
JAX-RS = spécification par le Java Community Process pour les services Restful en Java

- -décrit uniquement le coté serveur
- -`javax.ws.rs.*` = package de JAX-RS
- -implémentations : XCF d'Apache, RestEasy de Jboss, Jersey...

Jersey est l'implémentation d'Oracle pour cette spécification

- -fonctionne sur Tomcat, GlassFish, JBoss, ….
- -construit avec Apache Maven : ce qui simplifie la dépendance entre APIs fournit une Api coté client.

Exemple



Une simple classe POJO (Plain Old Java Object) :

- @Path spécifie de répondre aux requêtes d'URI finissant par *bonjour*

```
@Path("/bonjour")
public class Hello {
     …
}
```

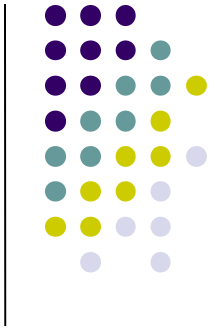
Exemple

Les annotations JAX-RS assurent la jonction entre la classe POJO et la requête ou réponse HTTP :

- @GET spécifie que la méthode répond à une requête GET
- @Produces spécifie le type de la réponse

```
package df.jersey.helloserver;
   @Path("/bonjour")
   public class Hello {
        @GET
        @Produces(MediaType.TEXT_HTML)
        public String sayHtmlHello() {
                  return "<html> " + "<title>" + "bonjour" + "</title>"
                  + "<body><h1>" + "bonjour" + "</h1></body>" + "</html> ";
   }
```

Exemple



Les annotations JAX-RS assurent la jonction entre la classe POJO et la requête ou réponse HTTP :

- On modifie @Path => L'URI est /prophetes/15
- @PathParam permet d'associer un morceau de l'URL de requête à un champ ou un paramètre.

```
@Path("/prophetes/{id}")
public String getProphete(@PathParam("id")  long id) {

        return « Bilou »+id ;
}
}
```

Les servlets - Développement

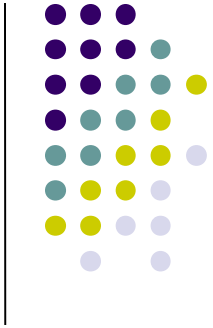
- Décrire la servlet dans le fichier ...\WEB-INF\web.xml

la classe du servlet, le package, l'URL de base

```
<web-app>
    <display-name>Hello</display-name>

     <servlet>
          <servlet-name>Hello</servlet-name>
          <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
     </servlet>
    <init-param>
      <param-name>jersey.config.server.provider.packages</paramname>
      <param-value>df.jersey.helloserver</paramvalue>
    </init-param>
     <servlet-mapping>
          <servlet-name>Hello</servlet-name>
          <url-pattern>/rest/*</url-pattern>
     </servlet-mapping>
</web-app>
```

Lancement de l'application

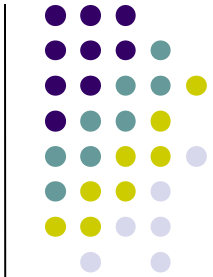


Lancer une requête

```
http://localhost:8008/cours-rest/rest/prophetes/15
```

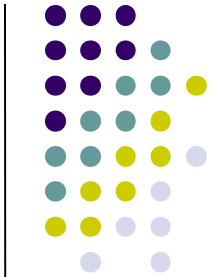
- http://localhost:8008 : correspond à l'URL sur laquelle le serveur (Tomcat le plus souvent) est lancé.
- /cours-rest est le nom du projet qui porte l'application
- /rest est l'url de base
- /prophetes/15 cette dernière partie de l'URL permet de s'adresser à notre service web tel que nous l'avons défini.

Régles



- Stateless : pas de gestion d'état
  - Le serveur ne stocke jamais l'états des applications, donc des requêtes.
  - Simplifie le service mais augmente le volume de communication
  - L'alternative est fournie par la régle HATEOAS

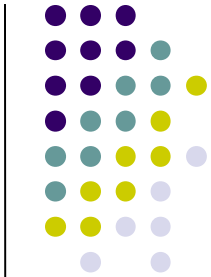
HypermediA as The Engine Of the Application State : HATEOAS



- Les liens hypermédia doivent permettre l’enchaînement des actions donc le changement d'état
- Exemple :

```
GET /medecins/doc/creneaux?date=20171221&status=dispo HTTP/1.1
```

HypermediA as The Engine of the Application State : HATEOAS

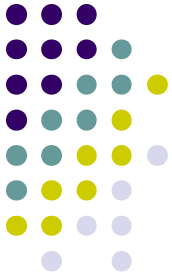


- Les liens hypermédia doivent permettre l’enchaînement des actions donc le changement d'état

Réponse :

```
<creneaux>
    <creneau>
        <id>1337</id>
        <heure>14h30</heure>
        ...
        <link rel="self" href="http://api.example.com/creneaux/1337" />
        <link rel="rdv" href="http://api.example.com/creneaux/1337/rdv" />
    </creneau>
    <creneau>
        <id>1645</id>
        ...
        <link rel="self" href="http://api.example.com/creneaux/1645" />
        <link rel="rdv" href="http://api.example.com/creneaux/1645/rdv" />
    </creneau>
    <link rel="self" href="http://api.example.com/medecins/doc/creneaux?date=20151021&status=ouvert" />
</creneaux>
```

WEB-ographie



- http://rest.elkstein.org/2008/02/what-is-rest.html
- http://opikanoba.org/tr/fielding/rest/ chapitre 5 de la thèse de R. Fielding sur REST
- http://mbaron.developpez.com/soa/jaxrs/

  Développer des Services Web REST avec Java : JAX-RS de Mickael Baron
- http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html

  The Java EE 7 Tutorial : Chapter 20 - Building RESTful Web Services with JAX-RS