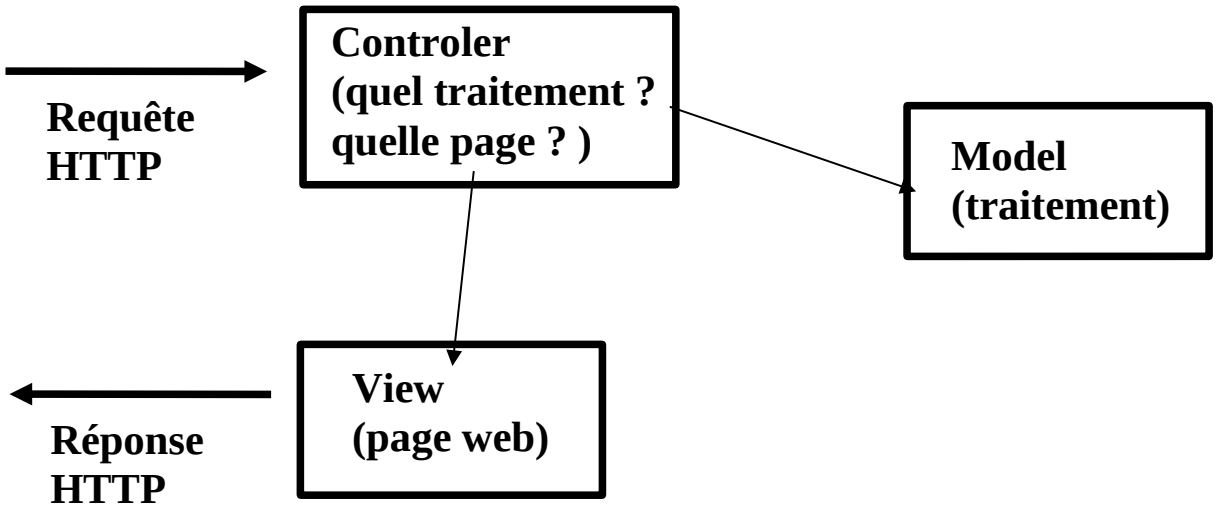
Applications Web dynamiques
Enterprise Java Applications

Daniel Hagimont

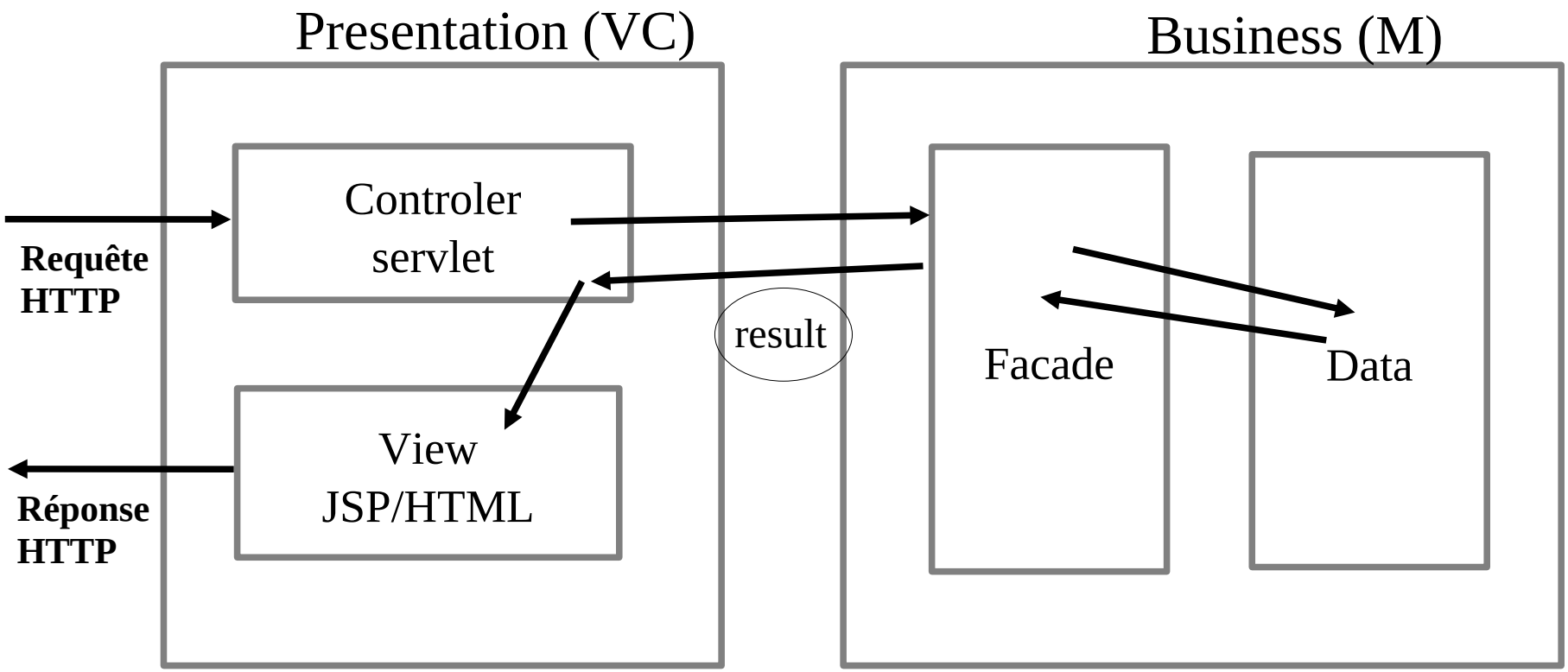
https://www.google.fr/search?q=daniel+hagimont+home+page

Rappel : Modèle MVC

- Model View Controler
- Séparation entre
  - Le Contrôleur : servlet qui aiguille les requêtes
  - La Vue : pages JSP pour l'affichage à l'écran
  - Le Modèle : les classes (beans) qui traitent les données



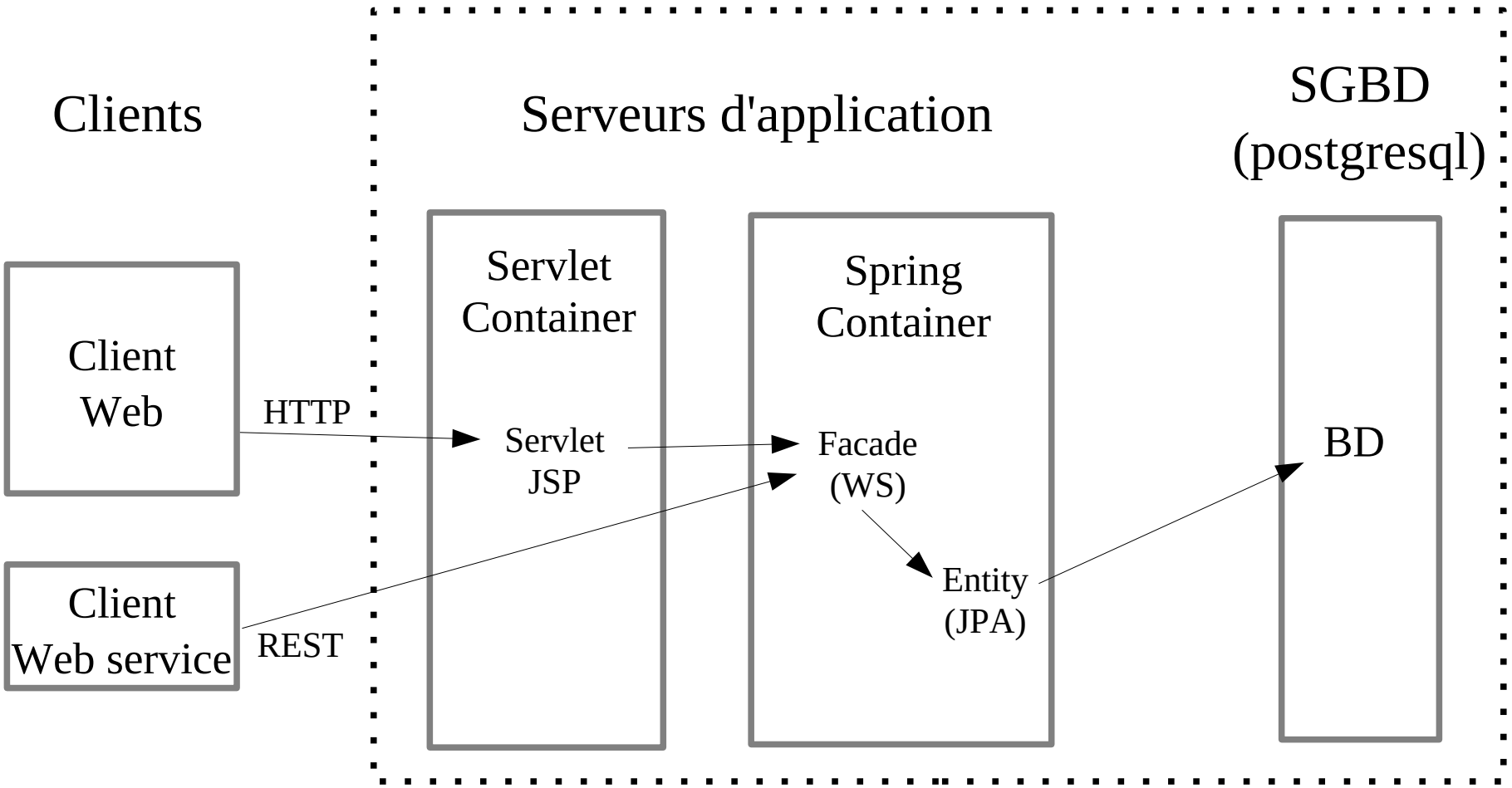
Rappel : Architecture souhaitée



Préambules

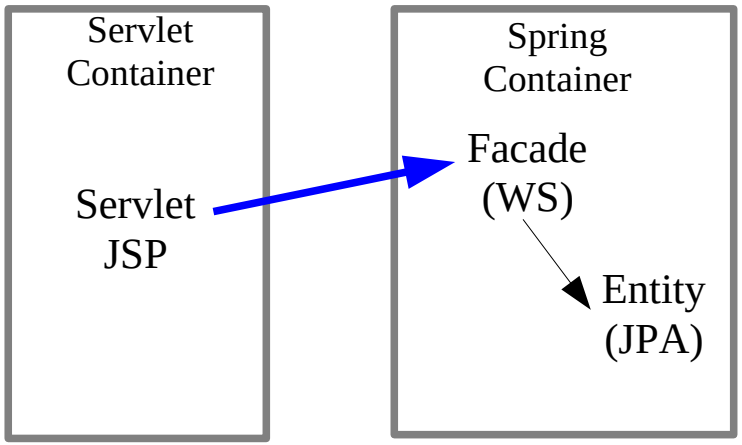
- Auparavant, mon cours reposait sur les EJB (Enterprise Java beans) qui fournit :
  - Des session beans pour structurer la Facade
    - Accessibles avec Java RMI ou des Web Services
  - Des entity beans pour gérer des Data dans une base de données
- Je suis passé à Spring
  - En raison de sa popularité dans les entreprises
  - Il fournit plus ou moins les mêmes concepts
    - Des REST web services pour construire la Facade
    - Des entity beans (on parle de Java Persistence API)

Architecture souhaitée



Structuration de la Facade avec Spring

- Voir le cours Intergiciel du semestre précédent
  - La servlet appelle la Facade sous la forme d'un web service REST
  - La Facade peut être appelée depuis n'importe quelle application (pas que la servlet)
  - La Facade est juste une fonction (réplication dans un datacenter)
- Exemple sur l'application bancaire



Exemple : les données manipulées (Data)

```
public class Compte {
   private int num;
   private String nom;
   private int solde;

   public Compte() {}

   public Compte(int num, String nom, int solde) {
       this.num = num; this.nom = nom; this.solde = solde;
   }

   public String toString() {
       return "Compte [num="+num+", nom="+nom+", solde="+solde+"]";
   }

   // setters and getters
```

Exemple : implantation de la Facade (service REST)

```
@RestController
public class Facade {
    private Map<Integer, Compte> comptes = new Hashtable<Integer, Compte>();

    @PostMapping("/addcompte")
    public void addCompte(@RequestBody Compte c) {
        comptes.put(c.getNum(), c);
    }

    @GetMapping("/consultercomptes")
    public Collection<Compte> consulterComptes() {
        return comptes.values();
    }

    @GetMapping("/consultercompte")
    public Compte consulterCompte(@RequestParam("num") int num) throws RuntimeException {
        Compte c = comptes.get(num);
        if (c == null) throw new RuntimeException("Compte introuvable");
        return c;
    }
```

Exemple : implantation de la Facade (service REST)

```
    @PostMapping("/debit")
    public void debit(@RequestParam("num") int num, @RequestParam("montant") int montant)
                                                                    throws RuntimeException {
        Compte c = consulterCompte(num);
        if (c.getSolde()<montant) throw new RuntimeException("Solde insuffisant");
        c.setSolde(c.getSolde()-montant);
    }

    @PostMapping("/credit")
    public void credit(@RequestParam("num") int num, @RequestParam("montant") int montant) {
        Compte c = consulterCompte(num);
        c.setSolde(c.getSolde()+montant);
    }

    public Facade() {
        addCompte(new Compte(1, "dan", 2000));
        addCompte(new Compte(2, "alain", 4000));
        addCompte(new Compte(3, "luc", 6000));
    }
}
```

Exemple : le controleur (interface cliente RestEasy)

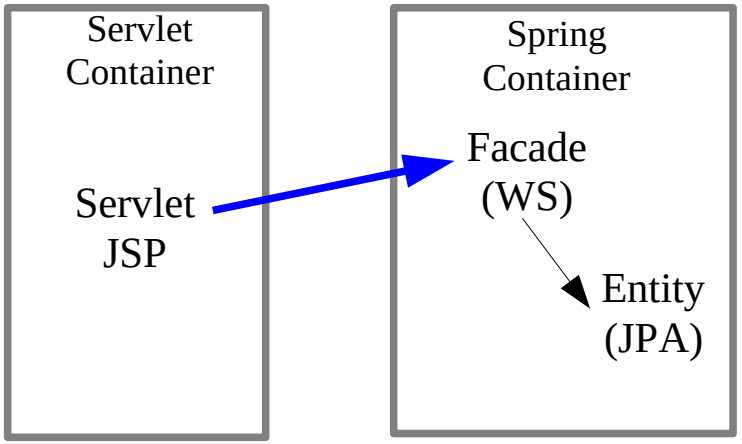
```
@Path("/")
public interface Facade {

   @POST
   @Path("/addcompte")
   @Consumes({ "application/json" })
   public void addCompte(Compte c);

   @GET
   @Path("/consultercomptes")
   @Produces({ "application/json" })
   public Collection<Compte> consulterComptes();

   @GET
   @Path("/consultercompte")
   @Produces({ "application/json" })
   public Compte consulterCompte(@QueryParam("num") int num) throws RuntimeException;

   @POST
   @Path("/debit")
   public void debit(@QueryParam("num") int num, @QueryParam("montant") int montant) throws RuntimeException;

   @POST
   @Path("/credit")
   public void credit(@QueryParam("num") int num, @QueryParam("montant") int montant);

}
```

Exemple : le controleur (servlet)

```
@WebServlet("/Controller")
public class Controller extends HttpServlet {

   final String path = "http://localhost:8080/bank-spring";

   Facade facade;

   public Controller() {
      super();
      ResteasyClient client = new ResteasyClientBuilder().build();
      ResteasyWebTarget target = client.target(UriBuilder.fromPath(path));
      facade = target.proxy(Facade.class);
   }

   protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                                          throws ServletException, IOException {
      try {
         String action=request.getParameter("action");
         if (action.equals("consulter")) {
            int num=Integer.parseInt(request.getParameter("num"));
            request.setAttribute("num", num);
            request.setAttribute("compte", facade.consulterCompte(num));
         }
```

Conclusion

- On a une séparation plus claire entre
  - Le front-end : comprend la vue (V) et le controleur (C)
    - Sous la forme de servlet/JSP
  - Le back-end : le modèle
    - Sous la forme d'un WS REST



Conclusion

- Suite : on voudrait gérer les données dans une BD
  - Sans avoir à programmer avec JDBC