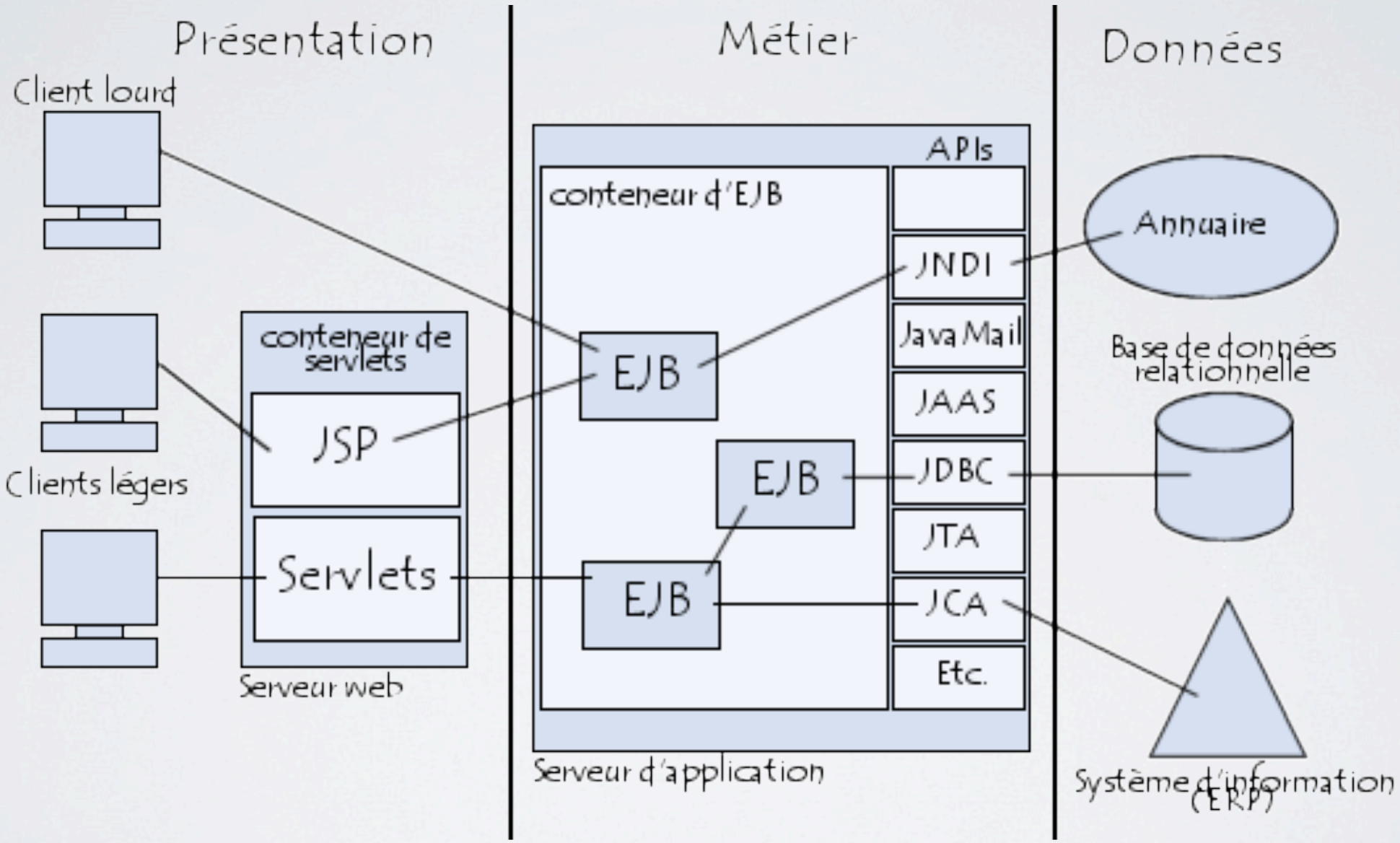
SPRING WEBFLOW

RAPPEL D'ARCHITECTURE



PROBLÉMATIQUE

Comment faire le lien entre mes vues graphiques et ma logique métier en respectant les contraintes (maintenance, coût & temps de dev., performance, portabilité, ...) de développement d'applications d'entreprise ?

LE DESIGN PATTERN FRONT-CONTROLLER

Lorsque l'on accède directement à une vue sans passer par un système centralisé, deux problèmes peuvent apparaître:

- Chaque vue va fournir ses propres services, ce qui entraîne une duplication du code;
- On mélange la partie navigation et la partie affichage

De plus, le contrôleur est plus compliqué à maintenir car il est dilué dans plusieurs endroits de l'application.

FRONT-CONTROLLER

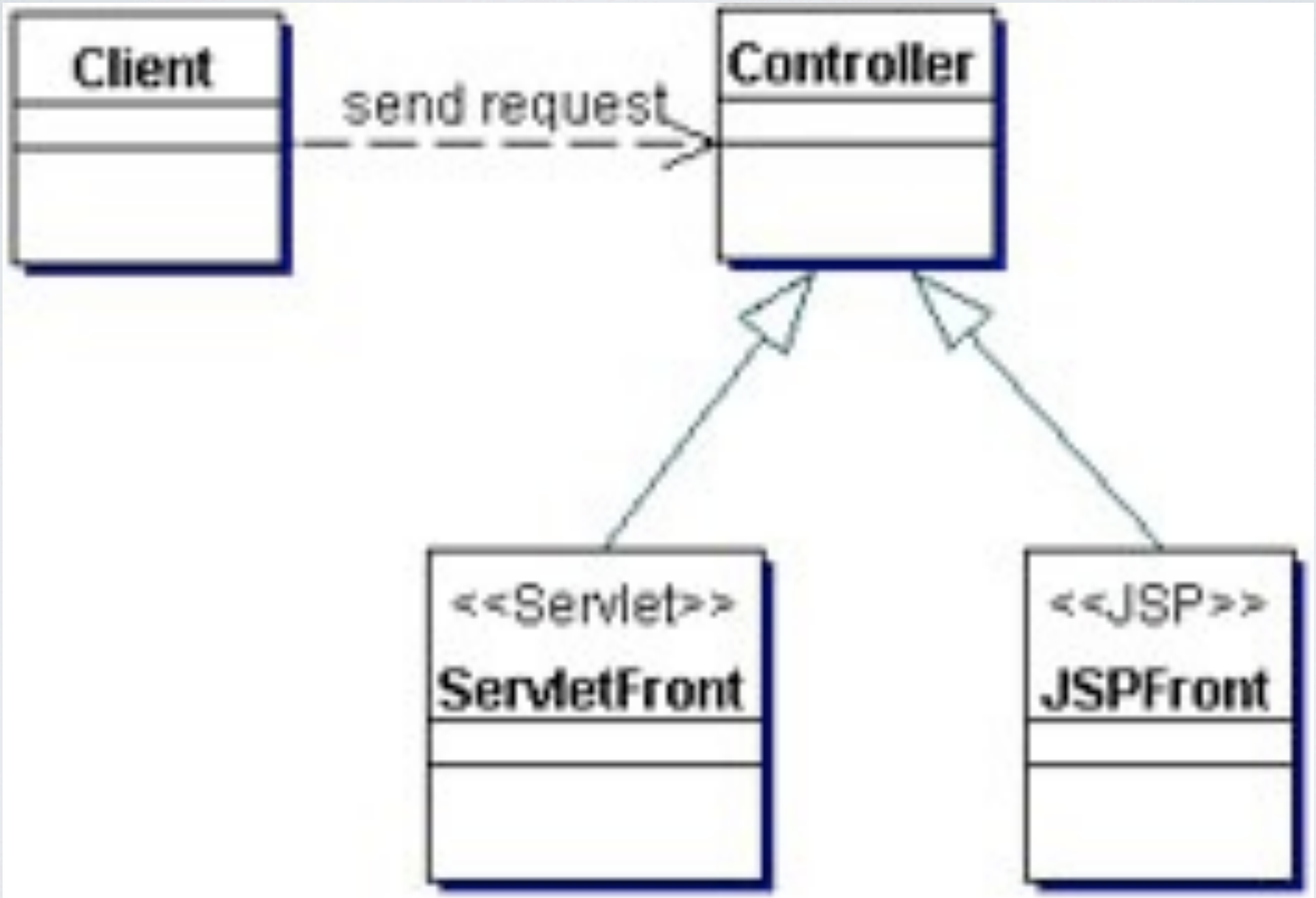
Utilisation d'un contrôleur comme point d'entrée de traitement des requêtes. Le contrôleur va gérer le traitement des requêtes (authentification et droits d'accès), la délégation des traitements métier, la gestion du choix de la vue appropriée, le traitement des erreurs et la gestion des stratégies de création de contenu.

FRONT-CONTROLLER

Le contrôleur va être couplé avec un dispatcheur en charge de la gestion des vues et de la navigation.

Le dispatcheur peut être directement encapsulé dans le contrôleur ou peut être vue comme un composant séparé.

SCHÉMA UML


Client
send request
Controller
<<Servlet>>
ServletFront
<<JSP>>
JSPFront

SPRING WEBFLOW

Spring WebFlow est une extension du pattern MVC permettant de définir la navigation entre les différentes pages d'une application Web.

C'est un front-controller !

SPRING WEBFLOW

- mécanisme (**Work Flow**) permettant:
  - décrire la navigation
  - limiter les portées au sein d'un processus
- **DSL** permettant de décrire le processus de navigation (XML)
- séparation claire entre:
  - logique de navigation et logique applicative
  - logique métier

POUR RAPPEL...

logique de navigation sans état !

VS.

logique applicative avec état!

AVANTAGES

- règles de navigation encapsulées dans un seul endroit
- découpage en couche des développements
- suivre la logique applicative sans entrer dans l'implantation
- réutilisation d'un processus à plusieurs endroits
- limitation des portées (optimisation de la mémoire)
- principe de conversation:
  - plus longue qu'une simple requête; plus courte qu'une session

ENRICHISSEMENT DE JSF

- comble certaines limitations de JSF:
  - navigation trop verbeuse
  - difficilement maintenable sur de gros projets
  - pas de variables de portées « Conversation »

AMELIORE LES COÛTS

- les flows sont rechargeables à chaud en phase de dev.
- syntaxe des flows est compréhensible par des non techniques
- réutilisation des flows sous forme de sous-flow
  - modularisation de pans entiers de navigation
- intégration naturelle avec Spring

DESCRIPTION D'UN FLOW

Un flow ou processus de navigation est tout simplement une machine à états finie !

DESCRIPTION D'UN FLOW

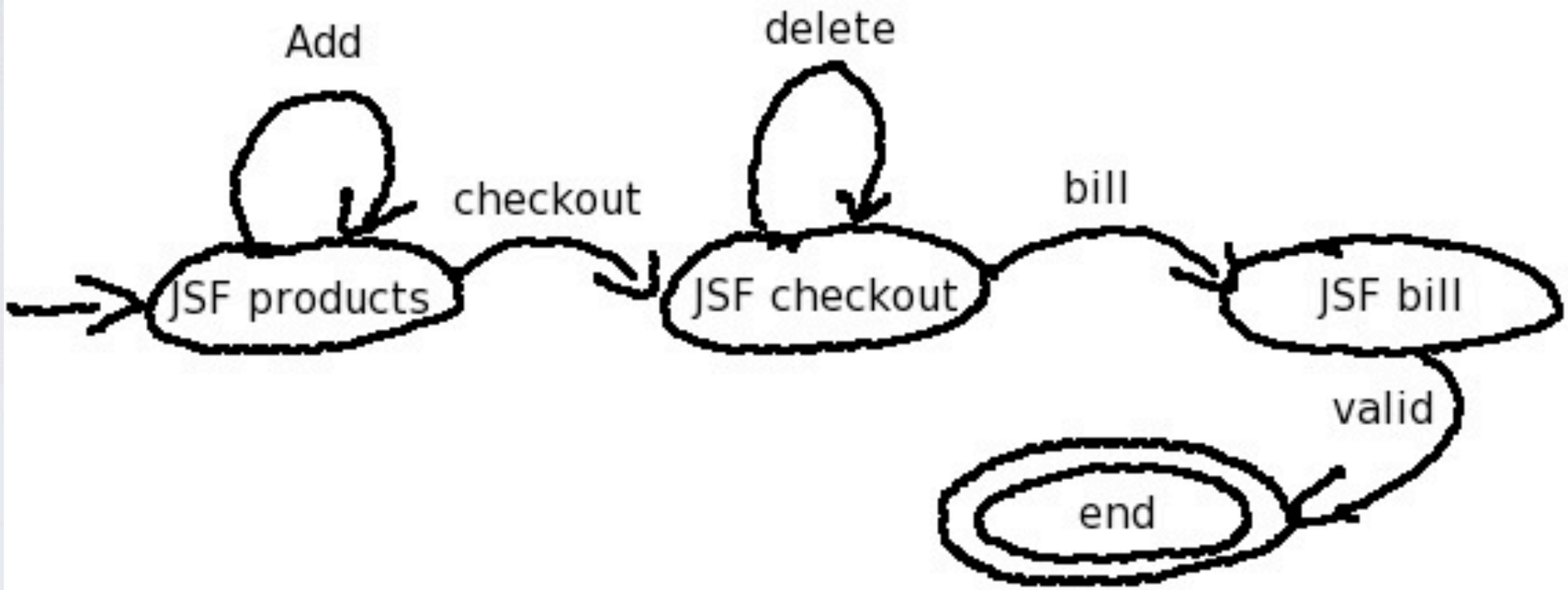
- décrit dans un fichier xml
- le nom du fichier est de la forme (*-flow.xml)
  - paradigm « Convention over Configuration »

- dans le répertoire src/main/webapp/WEB-INF/flow**s**
- les vues sont placées au même endroit
- les fichiers de localisation aussi
- les sous-flows dans un sous-répertoire

DESCRIPTION D'UN FLOW

- le fichier xml va décrire:
  - les états de la machine à états finie
  - les transitions entre ses états
  - les différentes actions déclenchées

EXEMPLE


Add
delete
checkout
bill
JSF products
JSF checkout
JSF bill
valid
end

EXEMPLE

```
<flow>
  <view-state id=«products»>
    <transition on=«add»>...</transition>
    <transition on=«checkout» to=«checkout»>...</transition>
  </view-state>

  <view-state id=«checkout»>
    <transition on=«delete»>...</transition>
    <transition on=«bill» to=«bill»>...</transition>
  </view-state>

  <view-state id=«bill»>
    <transition on=«valid» to=«end»>...</transition>
  </view-state>

  <end-state id=«end»>
    ...
  </end-state>
</flow>
```

DIFFÉRENTS TYPES D'ÉTATS

- start-state
- view-state
- end-state
- decision-state
- action-state

START-STATE

- état de départ d'un flow
- il peut ne pas être défini
- le premier état sera alors l'état de départ
- regroupe les traitements à faire pour initialiser un flow
  - renseigné les valeurs des variables par exemple

```
<start-state>...</start-state>
```

VIEW-STATE

- correspond à une page que l'on souhaite afficher
- quand on entre dans un view-state on affiche la page correspondant à l'état
- l'utilisateur va générer des événements
- ces événements vont faire changer le flow d'état

Lors de l'affichage le flow est en pause !

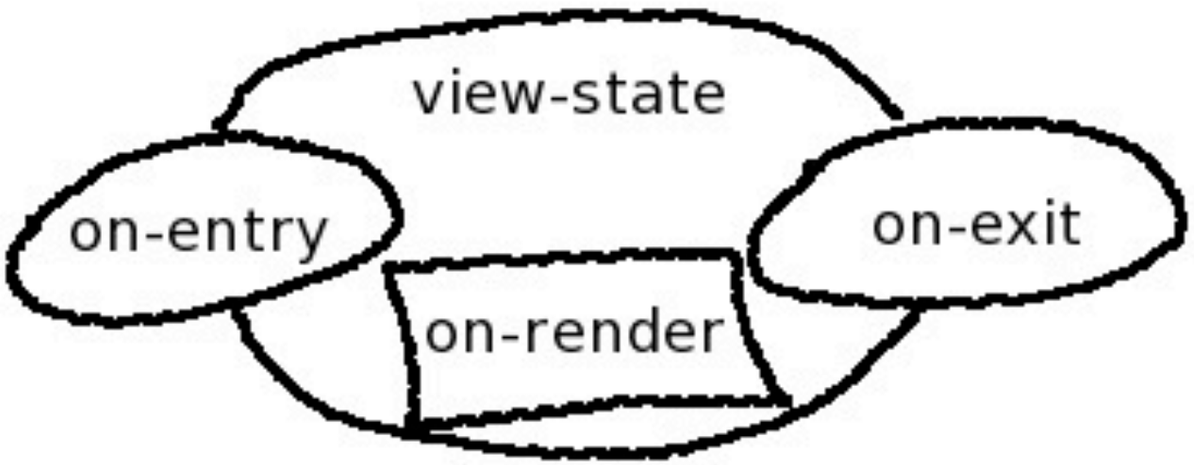
EXEMPLE

```
<view-state id=«toto» view=«toto»>
  <transition on=«connect» to=«connect»>
    <evaluate expression=«...» value=«...»/>
  </transition>
</view-state>
```

VIEW-STATE

- Action effectuée lors de l'entrée d'un view-state
  - `<on-entry></on-entry>`
- Action effectuée avant l'affichage de la vue
  - `<on-render></on-render>`
- Action effectuée lors de la sortie d'un view-state
  - `<on-exit></on-exit>`

ACTIONS VIEW-STATE



RENDERING DE FRAGMENTS

on peut demander de ne réafficher qu'une partie d'une vue

```
<transition on="next">
 <evaluate expression="..." />
 <render fragments="totoFragment" />
</transition>
```

Très important pour les performances !

END-STATE

- permet de définir la page à afficher ou les actions à effectuer à la fin d'un flow
- lors du passage dans un end-state les données liées à l'instance du flow ne sont plus accessibles

```
<end-state>
   ...
</end-state>
```

DECISION-STATE

- exécution d'un code non visuel
- dépend de données présentes dans le flow
- peut être associé au contrôleur dans un MVC

```
<decision-state>
   ...
</decision-state>
```

EXEMPLE

```
<decision-state id="isToto">
    <if test="t.isToto()"
        then="toto"
        else="titi" />
</decision-state>
```

ACTION-STATE

- exécution d'un code non visuel
- fait appel à des méthodes d'actions à l'entrée de ce type d'état
- comparable à la partie contrôleur dans un MVC
- va chercher de l'information pour réaliser son routage

```
<action-state>
  <evaluate expression=«...» value=«...» />
  ...
</action-state>
```

APPEL À UNE ACTION

Dans l'action state du flow:

```
<evaluate expression="multiAction.actionMethod1" />
```

Code Java de l'action:

```
public class CustomMultiAction extends MultiAction {
    public Event actionMethod1(RequestContext context) {
        ...
    }

    public Event actionMethod2(RequestContext context) {
        ...
    }
}
```

TRAITEMENT DES EXCEPTIONS

```
public class BookingAction {
    public String makeBooking(Booking booking, RequestContext context) {
        try {
            BookingConfirmation confirmation = bookingService.make(booking);
            context.getFlowScope().put("confirmation", confirmation);
            return "success";
        } catch (RoomNotAvailableException e) {
            context.addMessage(new MessageBuilder().error().
                .defaultText("No room is available at this hotel").build());
            return "error";
        }
    }
}
```

ACTION-STATE VS. DECISON-STATE

- decision-state permet de prendre une décision de routage en fonction d'éléments existants

- action-state permet de prendre une décision de routage en fonction du résultat de l'exécution d'une ou plusieurs actions

TRANSITIONS GLOBALES

Transitions valides quelque soit l'état du flow !

Exemple:

```
<global-transitions>
  <transition on=«toto» to=«titi» />
</global-transitions>
```

VARIABLES & PORTÉES

Traditionnellement, 4 types de portée:

- portée application
- portée session
- portée request
- portée page

PROBLÉMATIQUE

La portée session est trop étendue car elle couvre l'ensemble des interactions d'un utilisateur avec l'application !

La portée request est souvent trop brève car elle se limite à une interaction requête / réponse !

DE NOUVELLES PORTÉES...

De la plus restreinte à la plus globale:

- request
- flash
- view
- flow
- conversation

LES PORTÉES WEBFLOW

Ces nouvelles portées permettent:

- réduction du risque d'erreurs dus à un partage trop global
- réutilisation d'une partie de l'application dans un autre contexte
- pas d'impact ou remise en cause du reste de l'application

Une portée est définie pour chaque variable déclarée au sein du flow !

PORTÉE FLOW

- un flow est un processus métier indépendant représentant un cas d'utilisation particulier de l'application

- un flow est constitué de quelques vues et des données associées qui sont stockées

- la portée flow correspond de l'état de départ du flow à l'état fin du flow

PORTÉE VIEW

- une vue correspond à une page qui visualise des informations

- une portée vue correspond à l'entrée dans le view-state à la sortie du view-state

PORTÉE CONVERSATION

- une conversation est stockée dans la session HTTP
- une variable à une durée de vie de l'état de départ du flow parent à l'état fin du flow parent
- elle est partagée par le flow parent et tout ses sous-flows

Comment passer des paramètres à un flow ?

INPUT / OUTPUT D'UN FLOW

- Chaque flow peut définir un contrat bien définie de variables en entrée et en sortie

- Chaque flow peut récupérer des variables en entrée quand il commence

- Chaque flot peut retourner des variables en sortie quand il s'arrête

INPUT / OUTPUT D'UN FLOW

- on utilise l'élément input pour déclarer une variable d'entrée d'un flow
- on utilise l'élement output pour déclarer une variable de sortie d'un flow
- dans les deux cas on peut déclarer un type
- dans les deux cas on peut déclarer une valeur

EXEMPLES: INPUT

```
<input name="hotelId" type="long" />
```

```
<input name="hotelId" value="flowScope.hotels.hotelId" />
```

```
<input name="hotelId" type="long" value="flowScope.hotelId"
required="true" />
```

EXEMPLES: OUTPUT

```
<end-state id="bookingConfirmed">
    <output name="bookingId" />
</end-state>
```

```
<output name="confirmation" value="booking.confirmation" />
```

VARIABLES IMPLICITES

- flowScope
- requestScope
- viewScope
- requestScope
- flashScope
- conversationScope
- requestParameters
- currentUser
- currentEvent
- messageContext
- resourceBundle

EXEMPLE FLOWSCOPE

```
<evaluate expression="searchService.findHotel(id)"
result="flowScope.hotel" />
```

```
<evaluate expression="searchService.findHotel(id)"
result="conversationScope.hotel" />
```

EXEMPLE CURRENTUSER

```
<evaluate expression="bookingService.createBooking(id,
currentUser.name)" result="flowScope.booking" />
```

CONVERTIR

- conversion des données dans le form (string) vers le bon type de données dans le modèle

- des convertisseurs sont déjà définis pour les types de données primitifs (entiers, réels, énumérations et dates)

- on peut souhaiter rajouter un convertisseur spécifique

CONVERTIR

- étendre d'une classe déjà définie
  - StringToObject
- deux méthodes sont à implanter:

```
protected abstract Object toObject(String s, Class Clazz) throws
Exception;

protected abstract String toString(Object object) throws
Exception;
```

EXEMPLE

```
public class StringToMonetaryAmount extends StringToObject {

    public StringToMonetaryAmount() {
        super(MonetaryAmount.class);
    }

    @Override
    protected Object toObject(String string, Class clazz) {
        return MonetaryAmount.valueOf(string);
    }

    @Override
    protected String toString(Object object) {
        MonetaryAmount amount = (MonetaryAmount) object;
        return amount.toString();
    }
}
```

ENREGISTREMENT D'UN CONVERTISSEUR

ajout d'un service fournissant la ou les nouvelles conversions

```
@Component("conversionService")
public class ApplicationConversionService extends
DefaultConversionService
{
    @Override
    protected void addDefaultConverters() {
        super.addDefaultConverters();
        this.addConverter(new StringToMonetaryAmount());
        this.addConverter("monetaryAmount", new
StringToMonetaryAmount());
    }
}
```

ENREGISTREMENT D'UN CONVERTISSEUR

modification du fichier xxx-webflow-config.xml

```
<faces:flow-builder-services id="facesFlowBuilderServices"
development="true" conversion-service="conversionService"/>
```

VALIDER

- validation des objets saisis par les utilisateurs
- implantation d'un valideur
- ${model}Validator
  - convention over configuration!
  - @Component
- méthode validate${state}
  - state correspond à l'id du view-state

EXEMPLE

```
@Component
public class BookingValidator {
    public void validateEnterBookingDetails(Booking booking,
ValidationContext context) {
        MessageContext messages = context.getMessageContext();
        if (booking.getCheckinDate().before(today())) {
            messages.addMessage(new MessageBuilder().error
().source("checkinDate").
                    defaultText("Check in date must be a future
date").build());
        } else if (!booking.getCheckinDate().before
(booking.getCheckoutDate())) {
            messages.addMessage(new MessageBuilder().error
().source("checkoutDate").
                    defaultText("Check out date must be later than
check in date").build());
        }
    }
}
```

SUPPRESSION DE LA VALIDATION

```
<view-state id="chooseAmenities" model="booking">
    <transition on="proceed" to="reviewBooking">
    <transition on="back" to="enterBookingDetails"
validate="false" />
</view-state>
```