ESUP Portail

Formation ESUP-Commons V2

Du 30 janvier au 2 février 2012

Céline Didier – Université de Lorraine
Jérôme Robbiano – Université d'Aix-Marseille

Généralités

Sondage

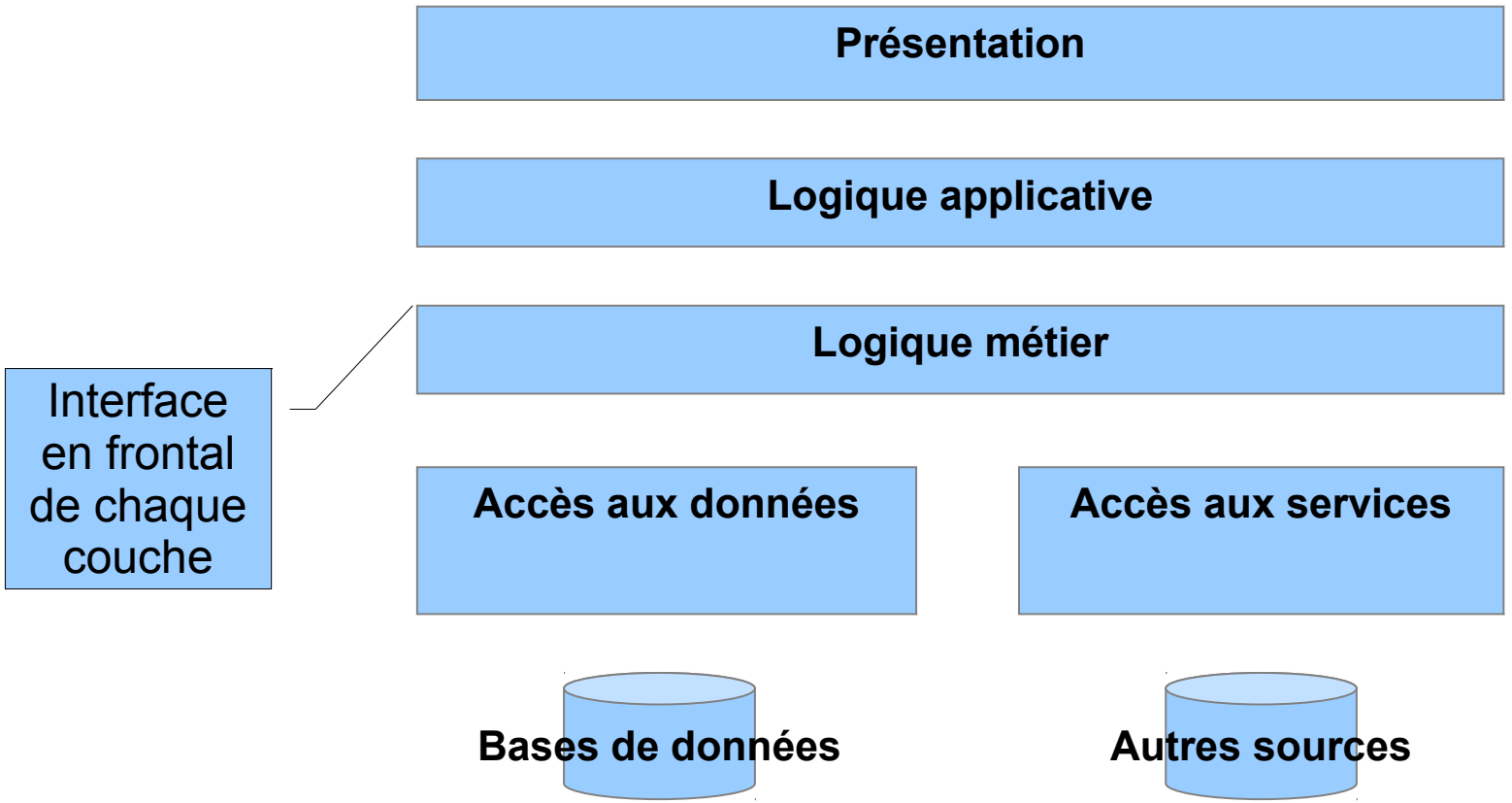
- Qui a déjà pratiqué ESUP-Commons V1 ?
  - .... / ....
- Qui pratique Spring ?
  - .... / ....
- Qui pratique JPA/Hibernate ?
  - .... / ....
- Qui pratique JSF ?
  - .... / ....
- Qui pratique Spring-MVC ?
  - .... / ....

Once upon a time…

- **Pascal Aubry lors de la formation V1**

  - Utilise dès que possible des outils de haut niveau !
    - Encore plus vrai en EC2 qui contient moins de code
  - Sépare bien tes couches !
  - Abstrais tes objets !
    - On continue à utiliser l'injection de dépendances
    - On continue à utiliser les interfaces
  - Relis ton code deux semaines après !
    - Sur cet aspect comme les 2 précédents maven améliore la modularité

Approche en couches



ESUP-Commons V2

- **Les objectifs**
  - Conserver le côté structurant de la démarche ESUP-Commons introduit avec la V1
    - Introduction de maven
  - Utiliser les technologies les plus récentes pour une meilleure productivité et offrir de nouveaux services
    - Spring 3, JPA, REST
  - Ne plus être dépendant d'une seule technologie de vue
- **Les composants**
  - Les librairies ESUP-Commons (core)
  - Des archetypes maven ESUP-Commons (blank)
  - Les applications ESUP-Commons d'exemple (example)

Compatibilité V1/V2

- **Structuration maven**
- **Changements liés à la techno de vue**
  - Étaient liés à JSF en V1
    - La gestion des exceptions --> portée en V2
    - La gestion des transactions --> portée en V2
    - La vérification de version de la base de données --> non portée en V2
  - JSF
    - La V1 était en JSF 1.1 et utilisait des balises e:
      - Ces balises ne sont pas portées en V2
    - La V2 utilise JSF 1.2 et 2.0
      - Facelet est ajouté en 1.2 et de base en 2.0

Maven

Maven

- **Introduction**
  - Maven est un outil permettant d'automatiser la gestion du projet
    - Mais pas seulement le build
  - Il est basé sur modèle de description du projet
    - Project Object Model matérialisé par un pom.xml

Maven
Convention plutôt que configuration

- **Introduction**
  - Si on respecte des conventions on n'a rien ou peu de choses à configurer
- **Exemple de la structure de fichiers**
  - Src
    - Main
      - Java
      - Resources
      - Webapp
    - Test
      - Java
  - Target

Maven
Gestion des dépendances

- **Chaque projet a une signature**
  - <project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.esupportail</groupId>
    <artifactId>example-domain-services</artifactId>
    <version>1.0-SNAPSHOT</version>
    <name>domain-service</name>
    </project>
  - Parfois noté
    - org.esupportail:example-domain-services:1.0-SNAPSHOT
- **Un projet peut dépendre d'autres projets**
  - <dependencies>
    <dependency>
    <groupId>org.esupportail</groupId>
    <artifactId>example-dao</artifactId>
    <version>1.0-SNAPSHOT</version>
    </dependency>
    </dependencies>

Maven
Gestion des dépendances

- **Les projets sont de différents type (<packaging>)**
  - Jar (librairie, valeur par défaut)
  - War (archive web)
  - Pom
    - Usages
      - Projet parent dont les valeurs sont potentiellement surchargées par le pom du projet
      - Utiliser pour factoriser un ensemble de dépendances
  - Maven-plugin
- **Les projets sont stockés sur des dépôts**
  - Sous la forme
    - D'un pom
    - D'une archive (jar, war)
    - Potentiellement : Javadoc ou source

Maven
Dépôts

- **Structuration**
  - <groupId[1]>/<groupId[2..n]>/<artifactId>/<version>/fichier
- **Dépôt de référence**
  - Maven est hiérarchique
    - Il s'appuie sur un super pom
      - Qui définit les comportements par défaut de maven
      - La localisation du dépôt de référence maven
        - http://repo1.maven.org/maven2
        - C'est là que l'on va chercher les dépendances
- **Dépôt local**
  - Localisation
    - ~/.m2/repository
  - Utiliser pour :
    - Cacher localement les dépendances
    - Mettre à disposition un projet des autres projets locaux

Maven
Dépôts

- **Dépôt intermédiaire**
  - Ex : ESUP
    - https://mvn.esup-portail.org
    - Sert au stockage des dépendances partagées par la communauté
      - Ex : esup-commons
    - Il est défini dans le pom.xml des projets
      - <repository>
        <id>esup</id>
        <url>https://mvn.esup-portail.org/content/repositories/releases</url>
        <snapshots><enabled>false</enabled></snapshots>
        <releases><enabled>true</enabled></releases>
        </repository>

Maven
Dépôts

- **Les dépôts sont de deux types**
  - SNAPSHOT
    - Pour les projets dont la version finit par « -SNAPSHOT »
    - Un projet peut y être poussé n fois
    - Une dépendance y est recherchée une fois par jour
      - -U force une nouvelle recherche
      - -o ne tente pas une nouvelle recherche
  - RELEASE
    - Pour les projets dont la version ne finit pas par « -SNAPSHOT »
    - Le projet ne peut y être poussé qu'une seule fois
    - Une dépendance n'est rapatriée localement qu'une seule fois

Maven
Cycle de vie

- **Liste ordonnée de phases**
  - Compile
    - Compile les sources (dans target)
  - Test
    - Lance les tests unitaires (contenus dans /src/test/java)
  - Package
    - Produit l'archive (dans target)
  - Install
    - Installe l'archive dans le dépôt local
  - Deploy
    - Déploie l'archive dans le dépôt distant
      - DistributionManagement/repository du pom.xml

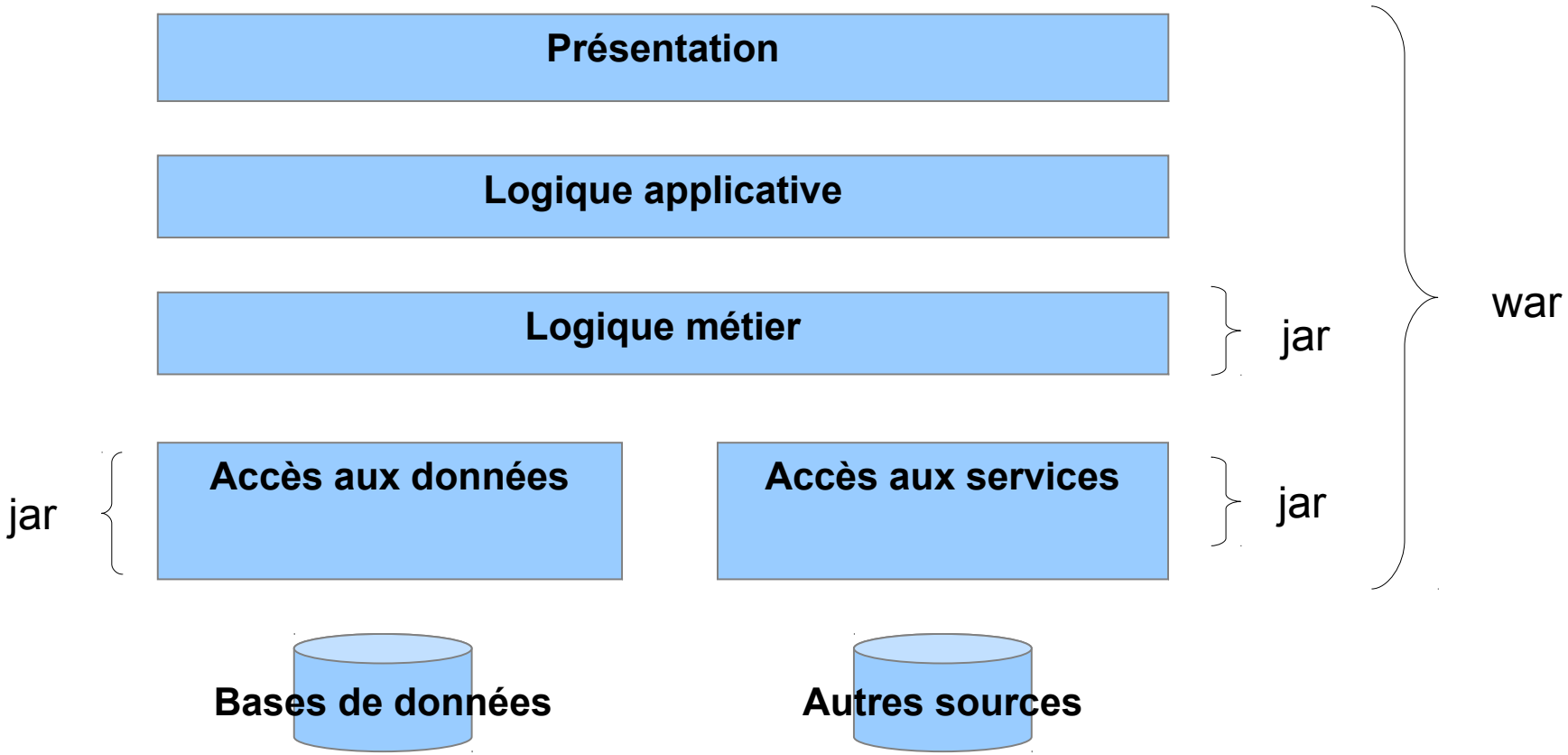
Maven Plugins

- **De base**
  - Maven utilise des plugins dont les goals (unités de traitement) sont liés aux phases du cycle de vie
    - compile --> compiler:compile
- **Un peu plus**
  - On peut aussi appeler spécifiquement un plugin
    - mvn eclipse:eclipse
      - Création d'un environnement éclipse
    - mvn release:prepare
      - Préparation d'une version (Vérifications, Changement des numéros, commit SVN)
    - mvn jetty:run
      - Lance le projet dans le serveur d'applications jetty
    - mvn archetype:generate
      - Génération d'un nouveau projet à partir d'un « moule »
  - On peut passer des paramètres à un plugin
    - Si la valeur par défaut dans le pom.xml ne convient pas
    - Ex. : mvn -Djetty.port=8085 jetty:run

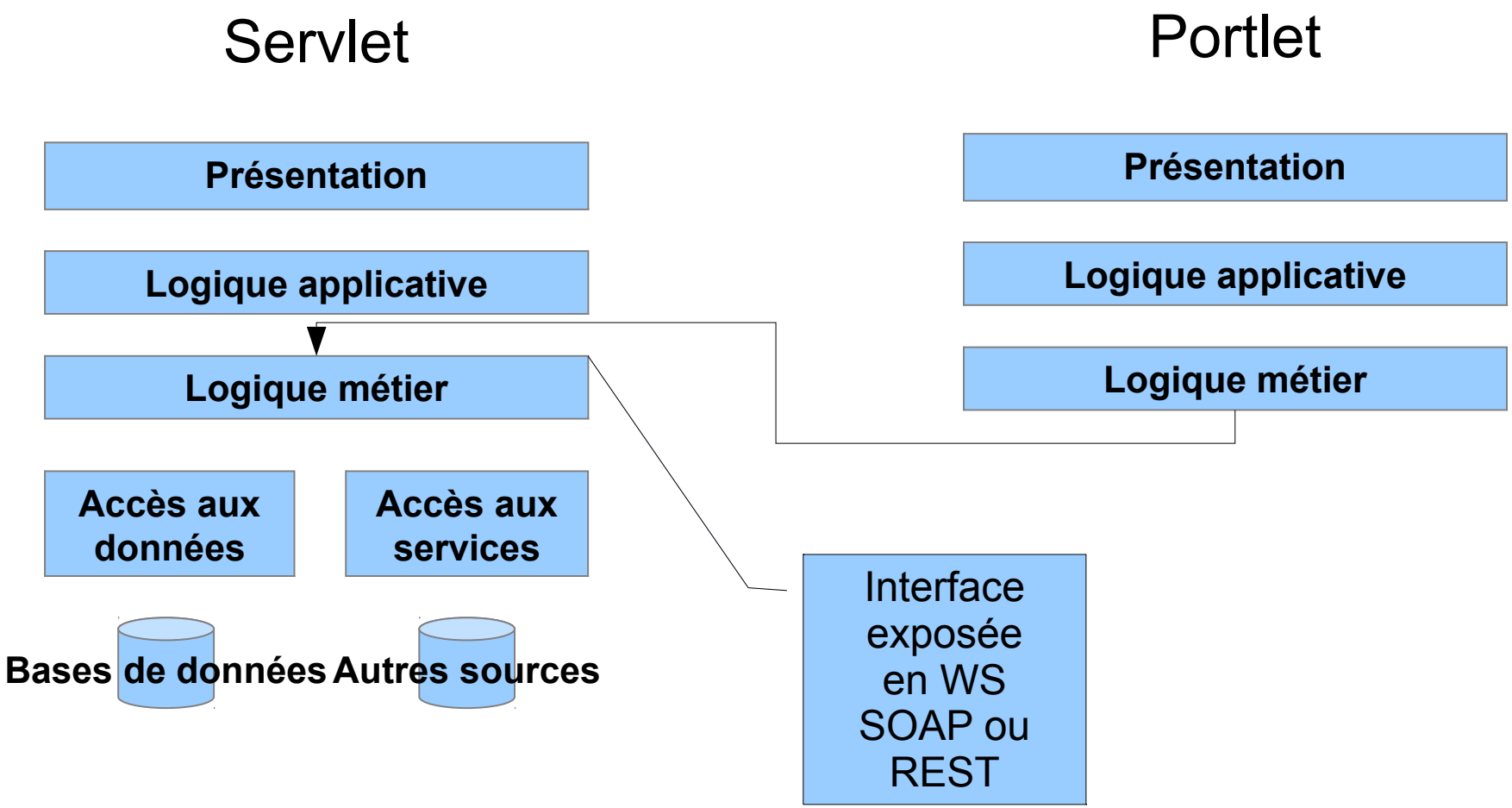
Maven
Projets modulaires

- **Deux notions**
  - Projet parent (balise <parent>)
    - Permet l'héritage de configuration (pom.xml du parent) entre plusieurs projets
      - Ex : properties/esupcommons.version
  - Module
    - Dans le pom.xml
      - modules/module
    - Permet de lancer des commandes maven sur tous les sous-projets
- **Dans ESUP-Commons V2**
  - On utilise les deux notions au sein d'un même projet parent (core, example et blank)

Approche modulaire



Approche modulaire



Spring

Spring

- **Introduction**
  - Spring est un conteneur léger dont le rôle est de gérer :
    - L'inversion de contrôle
      - C'est le conteneur qui instancie les objets
        - Notés bean dans les fichiers de configuration
    - L'injection de dépendances
      - C'est le conteneur qui injecte les dépendances
        - Appel des méthodes pour positionner les attributs

Spring
Les fichiers de configuration

- **Point d'entrée**
  - Le web.xml
    - <context-param>
      <param-name>**contextConfigLocation**</param-name>
      <param-value>
      classpath:/properties/applicationContext.xml
      </param-value>
      </context-param>
- **Import de fichiers**
  - Dans le applicationContext.xml pour en faciliter la maintenance
    - <import resource="i18n/i18n.xml" />
    - <import resource="classpath*:META-INF/esup-formation-domainservices-domain.xml" />
- **Annotations**
  - Spring permet de plus en plus de passer par des annotations
    - ESUP-Commons V2 utilise encore beaucoup les fichiers XML pour la définition des beans
    - On peut imaginer une évolution pour les définitions ne nécessitant pas une intervention de l'exploitant

Spring
Définition des beans

- **Déclaration d'un bean**
  - <bean id="domainService"
    class="org.esupportail.example.domain.DomainServiceImpl">
    <property name="daoService" ref="daoService" />
    </bean>
- **Portée du bean**
  - Ici la balise bean n'a pas d'attribut scope
    - Le bean est donc de scope singleton (valeur par défaut)
  - Les autres scopes fréquemment utilisés
    - Session
    - Request
- **Programmation par interface**
  - Ici le bean domainService correspond à la classe DomainServiceImpl
    - Cette classe implémente une interface DomainService
    - On peut lui substituer une autre implémentation (Ex : DomainServiceMock)

Spring
L'injection de données

- **Chaîne de caractères**
  - <property name="email" value="webmaster@domain.edu"/>
- **Autre bean**
  - <property name="ldapService" ref="ldapService"/>
- **Liste**
  - <property name="servers">
    <list>
    <ref bean="smtpServer1" />
    <ref bean="smtpServer2" />
    </list>
    </property>

Spring
Beans “abstraits” = héritage configuration

- **Beans abstraits**
  - <bean
    id="abstractLdapAwareBean" abstract="true">
    <property name="ldapService" ref="ldapService" />
    </bean>
- **Héritage**
  - <bean id="controller"
    class="org.esupportail.formation.web.controllers.TaskController"
    parent="abstractLdapAwareBean">
    <property name="x" value="y" />
    </bean>

Spring
Vérification des beans

- **Implémenter l'interface InitializingBean**
  - public void afterPropertiesSet() {
    if (!StringUtils.hasText(this.x)) {
    x = DEFAULT_X;
    }
    Assert.notNull(this.y, "property y of class " +
    this.getClass().getName() + " can not be null");
    }

Spring
Externaliser la configuration

- **Définition d'un propertyConfigurer**
  - <bean id="propertyConfigurer"
    class="org.springframework.beans.factory.
    config.PropertyPlaceholderConfigurer">
    <property name="locations">
    <list>
    <value>classpath:/properties/defaults.properties</value>
    <value>classpath:/properties/config.properties</value>
    </list>
    </property>
    </bean>
- **Ordre de prise en compte des fichiers de propriétés**
  - default --> config
- **Définition des variables**
  - smtp.host=stmp.mon-univ.fr
- **Utilisation des variables**
  - <property name="host" value="${smtp.host}" />

Spring
Externaliser la configuration 2

- **Usage avancé**
  - <bean id="propertyConfigurer"
    class="org.springframework.beans.factory.
    config.PropertyPlaceholderConfigurer">
    <property name="locations">
    <list>
    <value>classpath:/properties/defaults.properties</value>
    <value>classpath:/properties/config.properties</value>
    <value>file:${application.config.location}</value>
    </list>
    </property>
    <property name="ignoreResourceNotFound" value="true" />
    </bean>
- **le fichier properties est renseigné au lancement du tomcat**
  - Ex : -Dapplication.config.location=/tmp/foo.properties

Logs

Logs

- **Généralités**
  - Configuration
    - src/main/resources/log4j.properties
  - Niveaux de log
    - TRACE, DEBUG, INFO, WARN ou ERROR
- **Création du logger**
  - Classe statique
    - private static final Logger LOGGER =
      new LoggerImpl(NonClasse.class);
  - Autres classes
    - private final Logger logger = new LoggerImpl(getClass());
- **Utilisation**
  - logger.error("Nous avons un problème");
  - if (logger.isDebugEnabled())
    logger.debug("set language " + locale +
    " for user '" + currentUser.getId() + "'");

Tests

Tests
Intégration Maven

- **Lancer des tests**
  - Explicitement via mvn test
  - Mais aussi via mvn package car les tests font partie du cycle de vie maven
- **Ne pas lancer les tests**

  -DskipTests=true
- **Code de test**
  - Dans src/test/java
- **Utilisation de Junit**
  - <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.8.2</version>
    <type>jar</type>
    <scope>test</scope>
    </dependency>

Tests
Code de test

- **Utilisation des annotations**
  - JUnit
    - Pour cibler les méthodes contenant des tests
      - @Test public void getUsers() {
    - Initialiser certaines données
      - @Before public void setUp() {
  - Spring
    - Pour charger le contexte
      - @RunWith(SpringJUnit4ClassRunner.class)
      - @ContextConfiguration(locations="classpath*:META-INF/testApplicationContext.xml")
    - Accéder à certains beans
      - @Autowired
        public void setLdapUserService(LdapUserService ldapUserService) {
          this.ldapUserService = ldapUserService;
        }

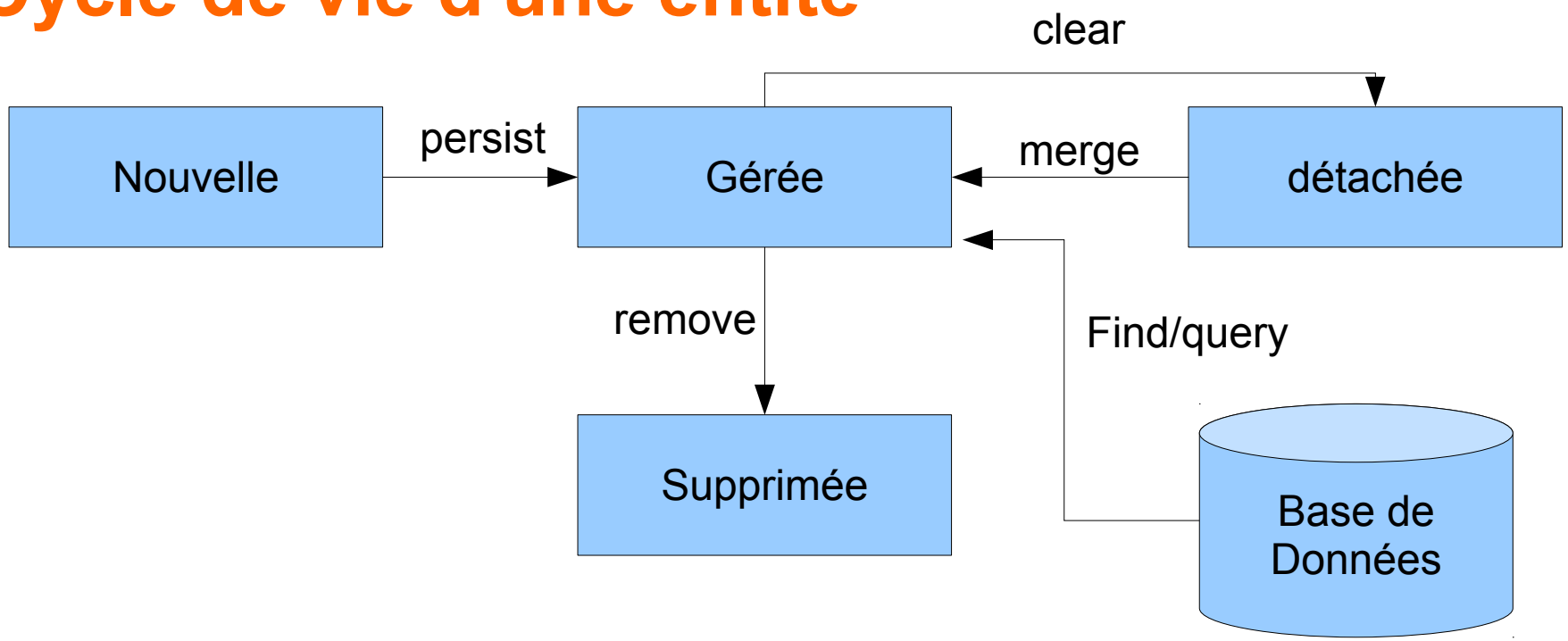
JPA

JPA
Généralités

- **ORM**
  - Object-Relational Mapping = Mapping Objet-Relationnel
- **Les normes**
  - EJB 1 et 2
    - EJB Session et entité
    - Hibernate a une approche différente
  - EJB 3
    - Les EJB entité disparaissent au profit de JPA
    - JPA est très proche de Hibernate
    - Jboss fait partie du groupe de spécification de JPA
    - Hibernate est une des implémentations de JPA
      - C'est le choix de ESUP-Commons V2

JPA
Cycle de vie

- **Notion de session du contexte de persistance**
  - En général maintenu le temps de la requête
    - A ne pas confondre avec la session applicative
  - En fin de session les instances gérées sont enregistrées en base
- **Cycle de vie d'une entité**



JPA
Mapping

- **Par annotation**
  - Pratique
    - http://www.objectdb.com/api/java/jpa/annotations
  - Sur les classes
    - @Entity
      - Précise que la classe est une entité à persister
  - Sur les champs
    - @Id
      - Clé primaire
    - @GeneratedValue
      - Valeur générée automatiquement
    - @Column
      - Permet de donner les caractéristiques de colonne qui stockera le champ (Ex : nom, null, taille, etc.)
    - @Transient
      - Le champ n'est pas persisté

JPA
Mapping 2

- **Les relations**
  - Annotations
    - @ManyToMany, @ManyToOne, @OneToMany, @OneToOne
  - Navigabilité
    - Une relation peut être unidirectionnelle ou bidirectionnelle
  - Lazy loading
    - Par défaut les enfants ne sont chargés que tardivement
      - Attention au cas où la session a pris fin. On peut avoir besoin de rattacher une entité à la nouvelle session (merge).
  - Cascade
    - En général on n'enregistre pas explicitement une entité (fait automatiquement en fin de session). De plus ses enfants (ajoutés, supprimés, modifiés) sont aussi enregistrés automatiquement.
      - cascade=ALL <--> cascade={PERSIST, MERGE, REMOVE, REFRESH, DETACH}

JPA
Mapping 3

- **Les requêtes**
  - Déclarées par annotation
    - ```
      @NamedQueries({
       @NamedQuery(
        name="tasksForUser",
        query="SELECT t FROM Task t WHERE t.owner.login = :userLogin"
       )
      })
      ```
  - Possibilité de passer des paramètres (:userLogin)
  - Possibilité de naviguer dans les objets (t.owner.login)
      ```

JPA
Configuration

- **Maven**
  - Déclaration d'une dépendance avec esup-commons2-jpa
- **Spring (dao)**
  - Support des annotations
    - <bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor" />
  - Branchement sur le mécanisme de gestion des transactions
    - <bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager"> <property name="entityManagerFactory" ref="entityManagerFactory" /> </bean>

JPA
Configuration 2

- **Spring (dao)**
  - Gestionnaire d'entité JPA
    - <bean id="entityManagerFactory"
      class="....LocalContainerEntityManagerFactoryBean">
      <property name="dataSource" ref="**${datasource.bean}**" />
      <property name="jpaVendorAdapter" ref="**jpaVendorAdapter**" />
      <property name="persistenceXmlLocation"
      value="classpath*:META-INF/**esup-formation-dao-persistence.xml**" />
      <property name="jpaProperties" ref="**jpaProperties**" />
      </bean>
  - Data source JNDI (${datasource.bean})
    - <jee:jndi-lookup id="JNDIDataSource"
      jndi-name="${jndi.datasource}" **lookup-on-startup="false"**
      expected-type="javax.sql.DataSource"/>
    - A noter le lookup-on-startup="false" qui permet de ne pas faire une recherche JNDI si le bean n'est pas référencé

JPA
Configuration 3

- **Spring (dao)**
  - Data source JDBC (${datasource.bean})
    - <bean id="JDBCDataSource"
      class="org.apache.commons.dbcp.BasicDataSource"
      destroy-method="close" **lazy-init="true"**>
      <property name="driverClassName"
        value="${jdbc.connection.driver_class}" />
      <property name="maxActive" value="100" />
      <property name="maxIdle" value="30" />
      <property name="maxWait" value="100" />
      <property name="url" value="${jdbc.connection.url}" />
      <property name="username" value="${jdbc.connection.username}" />
      <property name="password" value="${jdbc.connection.password}" />
      </bean>
    - A noter le lazy-init="true" qui permet de ne pas instancier le bean s'il n'est pas référencé

JPA
Configuration 4

- **Spring (dao)**
  - Le fournisseur JPA
    - <bean id="jpaVendorAdapter"
      class="...jpa.vendor.HibernateJpaVendorAdapter">
      <property name="showSql" value="true" />
      <property name="generateDdl" value="true" />
      <property name="database" value="${jpa.database.type}" />
      </bean>
  - Les paramètres de ce fournisseur
    - <util:properties id="jpaProperties">
      <prop key="hibernate.cache.provider_class">...NoCacheProvider</prop>
      <prop key="hibernate.cache.use_query_cache">false</prop>
      <prop key="hibernate.cache.use_second_level_cache">false
      </prop>
      </util:properties>

JPA
Configuration 5

- **JPA**
  - persistence.xml
    - <persistence .../...>

```
<persistence-unit name="manager1"
   transaction-type="RESOURCE_LOCAL">
  <class>org.esupportail.formation.domain.beans.Task</class>
 </persistence-unit>
</persistence>
```

JPA
Configuration 6

- **Spring (transaction)**
  - Au niveau de la couche métier
  - Les transactions
    -
```
<aop:config>
  <aop:pointcut id="domainMethods"
    expression="execution(* org.esupportail.*.domain.DomainServiceImpl.*(..))"/>
    <aop:advisor advice-ref="txAdvice"
      pointcut-ref="domainMethods" />
</aop:config>
<tx:advice id="txAdvice" transaction-manager="txManager">
  <tx:attributes>
    <tx:method name="add*" propagation="REQUIRED" />
    <tx:method name="delete*" propagation="REQUIRED" />
    <tx:method name="update*" propagation="REQUIRED" />
    <tx:method name="*" propagation="SUPPORTS" read-only="true"/>
  </tx:attributes>
</tx:advice>
```

JPA
DAO

- **Accès au contexte de persistance**
  - @PersistenceContext
    public void setEntityManager(EntityManager em) {
      this.entityManager = em;}
- **CRUD**
  - public void addTask(Task task) {
    entityManager.persist(task);}
  - public Task getTask(long id) {
    Task task = entityManager.find(Task.class, id);
    return task;}
  - Rien à faire en update sauf si l'instance est détachée
  - public void deleteTask(Task task) {
    Task t = entityManager.find(Task.class, task.getId());
    entityManager.remove(t);}

JPA
DAO

- **Rattacher une instance**
  - public Task updateTask(Task task) {
    return entityManager.merge(task);}
    - L'objet détaché est passé en paramètre
    - La fonction renvoie l'objet rattaché
- **Rechercher des objets**
  - public List<Task> getTasksForUser(User u) {
    Query q = entityManager.createNamedQuery("tasksForUser");
    q.setParameter("userLogin", u.getLogin());
    return (List<Task>)q.getResultList();
    }

JPA
Quelques bonnes pratiques

- **Clé métier != clé primaire**
  - Pour permettre une évolution simple du MPD
    - Définir une clé primaire auto-générée
    - Assurer la cohérence des données par une contrainte d'unicité sur la clé métier
- **hashCode() et equals()**
  - Si on stocke des objets en tant que List, Map ou Set alors il est requis d'implémenter hashCode() et equals() sur ces objets
  - hashCode() et equals() utilisent les éléments de la clé métier

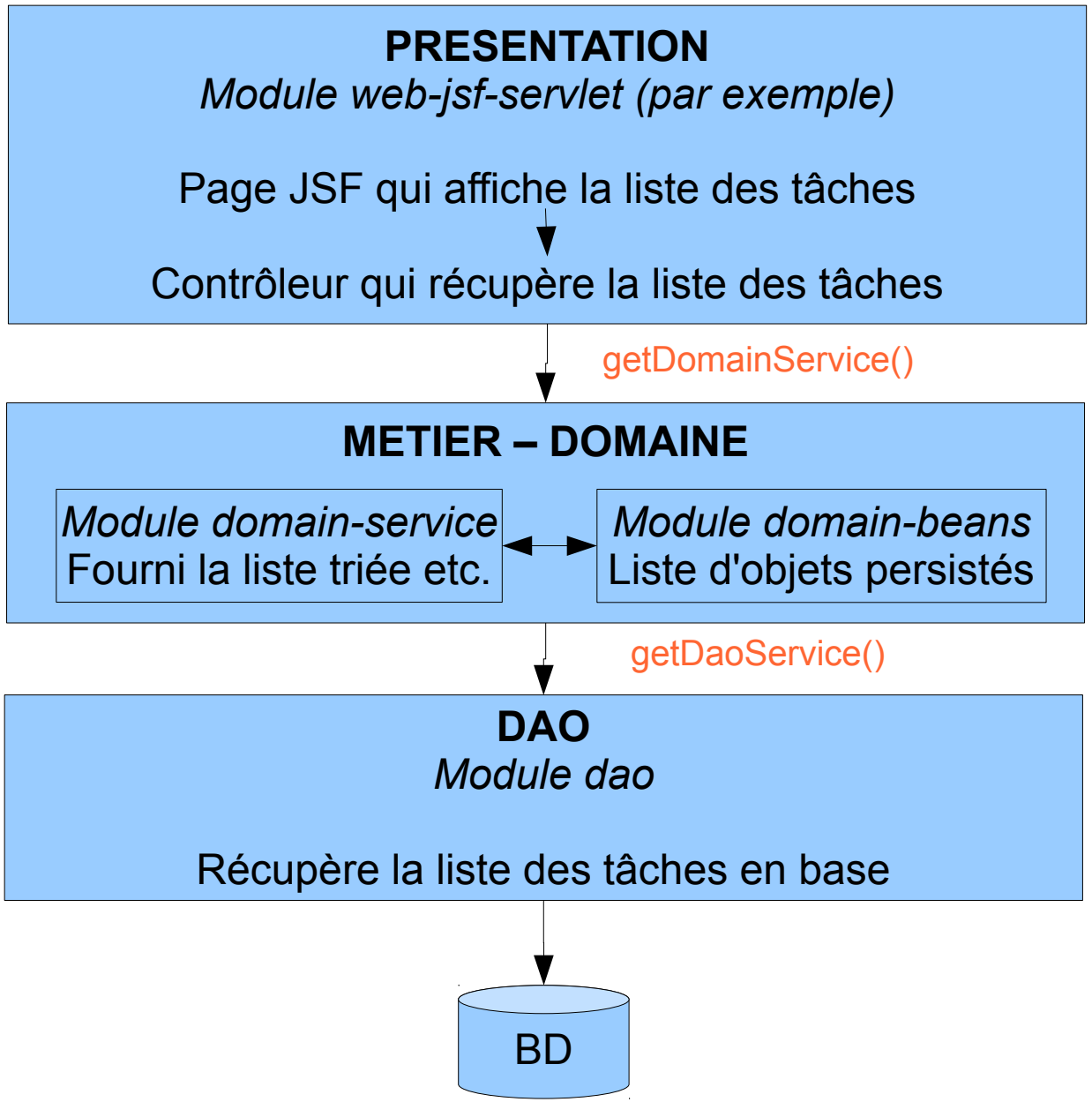
Vue, Métier, Données
Spring, Maven
Les recettes

Les recettes
cas classique

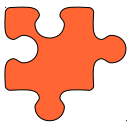
- **Expression du besoin**
  - La vue a besoin d'accéder au métier
    - Ex : obtenir une liste des tâches
  - Note :
    - Le métier, pour faire son traitement, a besoin d'accéder aux données
- **Besoin technique**
  - Les couches métier et données doivent être compilées et à jour
    - mvn clean install
  - La vue doit avoir accès à la couche métier
    - Ajout de la dépendance maven dans le pom.xml de la vue
  - Le contrôleur a besoin que l'objet domainService soit instancié et lui soit rattaché
    - Configuration Spring du contrôleur
  - Note :
    - Dans le cadre de l'interaction métier/données, l'objet daoService doit être instancié et rattaché à l'implémentation correspondant à domainService

Les recettes
Séparation en couches/modules

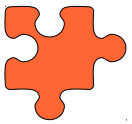
Exemple d'un cas classique :



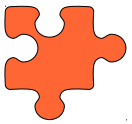
MAVEN



mvn clean install



mvn clean install



mvn clean install

Les recettes
cas classique 2

- **domainService rattaché au contrôleur**
  - Les contrôleurs JSF de EC étendent AbstractDomainAwareBean
  - Configuration Spring
    - <bean id="userController"
      class="org.esupportail.formation.web.controllers.UserController"
      parent="abstractContextAwareController"
      scope="session">
    - AbstractContextAwareController a comme parent abstractDomainAwareBean
    - <bean id="abstractDomainAwareBean" abstract="true"
      parent="abstractJsfMessagesAwareBean">
      <property name="domainService" ref="domainService" />
      </bean>
  - Code java
    - Via l'héritage de AbstractDomainAwareBean notre contrôleur a accès à une méthode getDomainService()
      - getDomainService().getTasks();

Les recettes
cas classique 3

- **Encore faut-il avoir codé getTASK()**
  - Mode opératoire
    - Ajouter la déclaration dans l'interface (DomainService)
      - public List<Task> getTasks();
    - Eclipse signalera les implémentations en erreur (par ex. DomainServiceImpl)
      - Eclipse proposera d'ajouter les méthodes non encore implémentées
      - Coder ensuite le corps de la méthode
  - Note
    - Dans un premier temps on peut coder un DomainServiceMock qui n'a pas besoin d'accéder à la couche de persistance

Les recettes
cas classique 4

- **daoService rattaché à l'implémentation de domainService**
  - Configuration Spring
    - <bean id="domainService"
      class="org.esupportail.formation.domain.DomainServiceImpl">
      <property name="daoService" ref="daoService" />
      </bean>
  - Code java
    - private DaoService daoService;
    - public void setDaoService(DaoService daoService) {
      this.daoService = daoService;
      }
    - public List<Task> getTasks() {
      List<Task> ret = daoService.getTasks();
      //faire un traitement métier sur les tâches ici
      return ret;
      }

Les recettes
cas classique 5

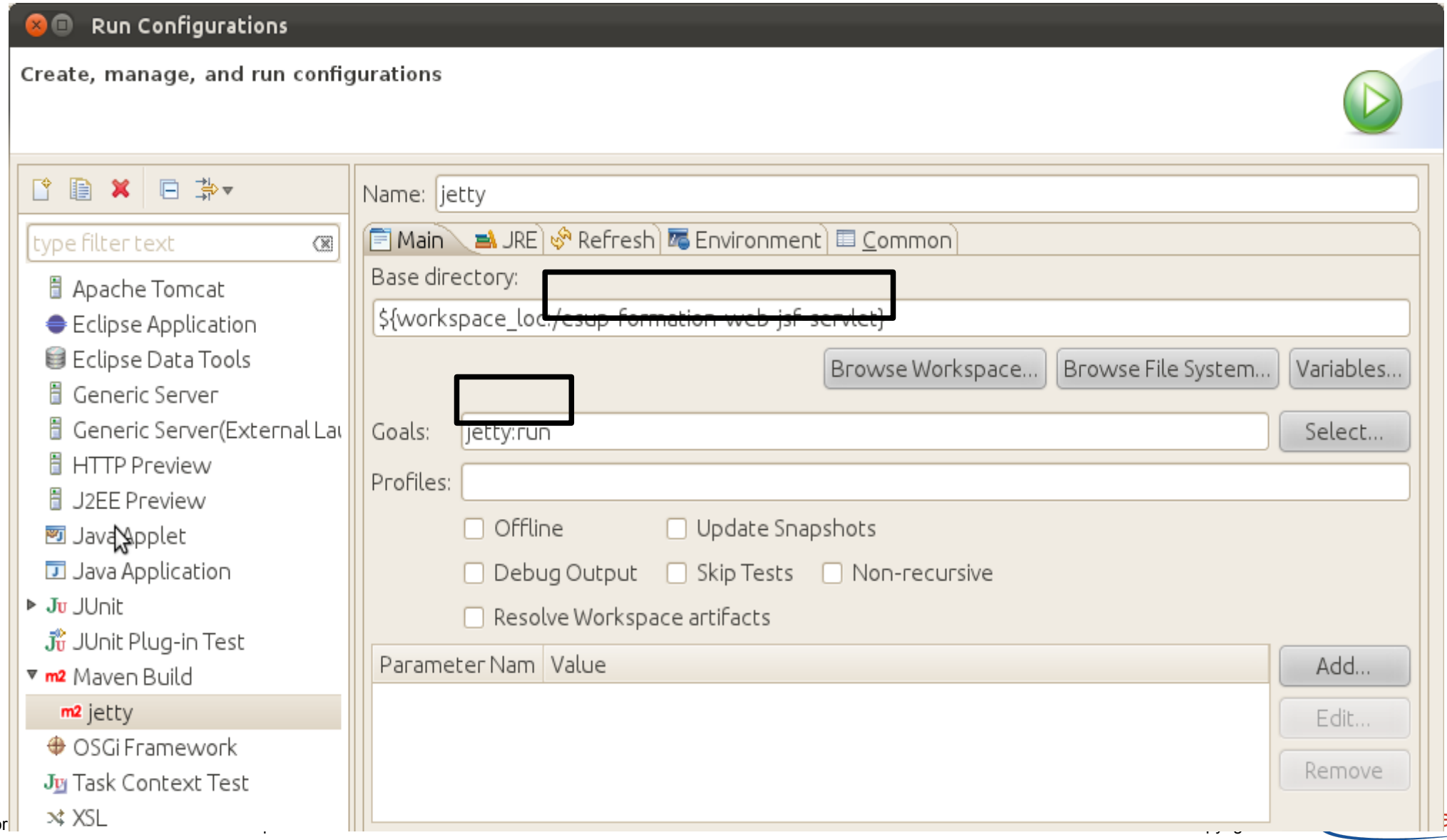
- **Les dépendances**
  - La vue doit avoir accès à la couche métier
    - Pom.xml de esup-formation-web-jsf-servlet
      - <dependency>
        <groupId>org.esupportail.formation</groupId>
        <artifactId>esup-formation-domain-services</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        </dependency>
  - La couche métier doit avoir accès à la couche données
    - Pom.xml de esup-formation-domain-services
      - <dependency>
        <groupId>org.esupportail.formation</groupId>
        <artifactId>esup-formation-dao</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        </dependency>
  - Note :
    - Ces dépendances sont recherchées dans le repository maven local. Si le code des couches métier ou données a évolué alors **ne pas oublier le mvn clean install**

Les recettes
Configurer Eclipse

- **On va avoir besoin de 2 commandes maven**
  - Mvn jetty:run sur notre module cible (Ex : esup-formation-web-jsf-servlet)
    - Pour pouvoir tester via un navigateur
  - Mvn clean install sur le parent
    - Pour installer nos modules dans le repository local
    - Ceci rendra ces modules « accessibles » par notre module cible
- **On peut configurer eclipse pour ces 2 commandes**

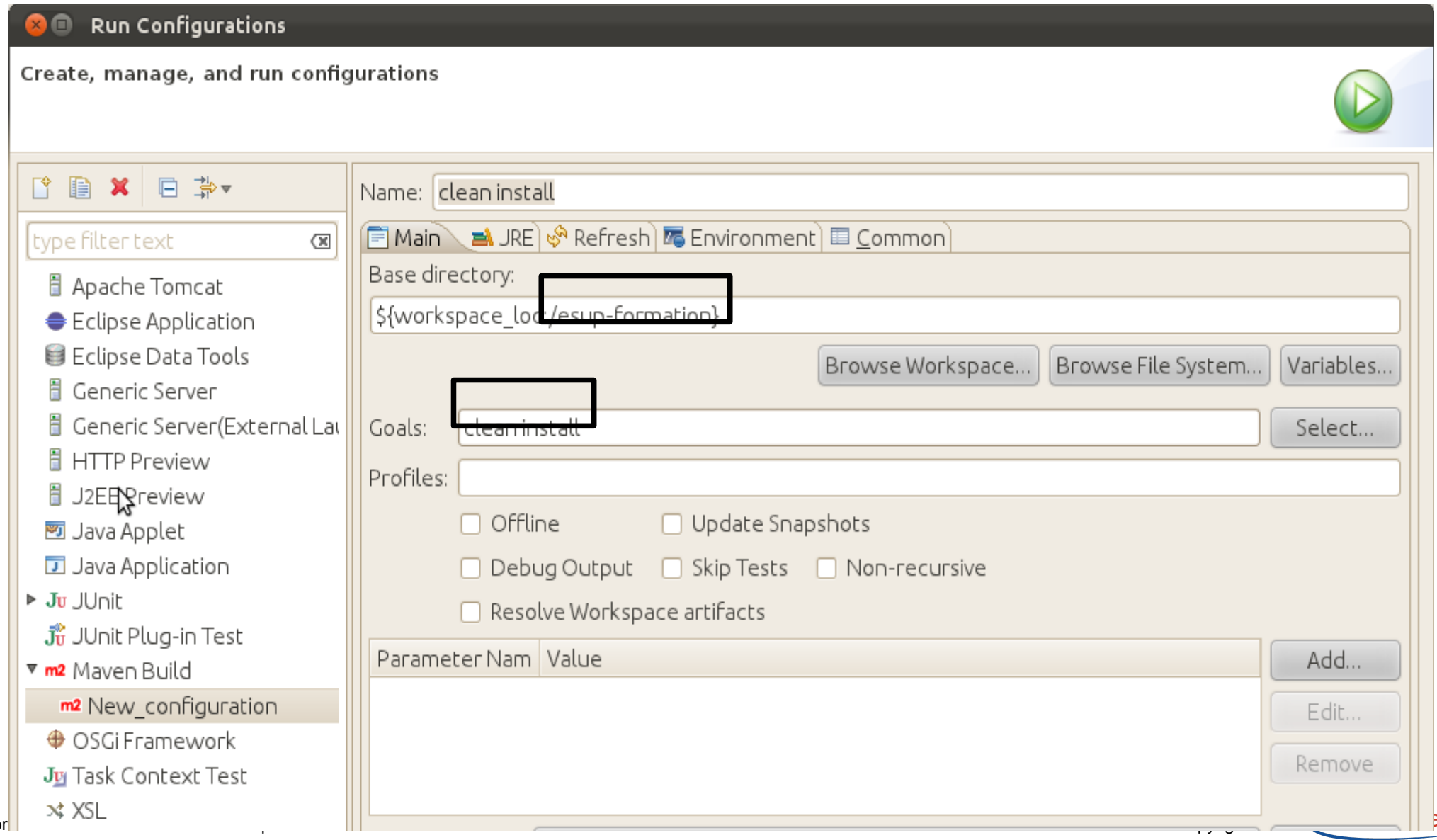
Les recettes
Configurer Eclipse 2

- **Mvn jetty:run**



Les recettes
Configurer Eclipse 3

- **Mvn clean install**



Les recettes
Configurer Eclipse 4

- **Mode debug**
  - Mvn jetty:run est accessible en mode debug
  - 2 avantages
    - On peut poser des points d'arrêt
    - On peut faire du remplacement de code à chaud !
      - Même celui des dépendances !
      - NB : Ne fonctionne pas si :
        - On ajoute une nouvelle classe
        - On ajoute une méthode à une classe
        - On change le type retourné par une méthode
  - Attention
    - Penser à lancer un mvn clean install
      - Dans les cas où le remplacement de code à chaud ne fonctionne pas
      - Après un stop de votre jetty
        - Le code remplacé à chaud n'est pas dans le repository maven local

Les recettes
Configurer Eclipse 5

- **Visiblement c'est un bug de m2e**
  - !!! On devrait directement avoir les sources dans la conf par défaut du « source lookup »
- **Accès au code source des dépendances ????**
  - Des autres modules
    - Pour le debug ou le remplacement à chaud
  - Des autres dépendances
    - Par Ex. EC2
    - Pour le debug

JSF

JSF
Généralité

- **Approche**
  - La vue s'appuie sur des composants
    - On peut utiliser des librairies tierces
  - Basé sur un cycle de vie avec des événements que l'on peut intercepter
    - Fonctionnement avancé (utilisé en interne par EC-Core)
  - La navigation est gérée par des règles
    - JSF 2 introduit une notion de règles par défaut (peu ou pas utilisée dans EC2)

Versions de JSF

	JSF 1.2	JSF 2.0
Servlet	OK	OK
uPortal 3 (portlet 1.0)	OK Via portlet bridge (JSR 301)	Pas possible
uPortal 4 (portlet 2.0)	OK Via portlet bridge (JSR 329)	OK Via porlet bridge (Spécification officielle pas sortie) Pas encore testé par ESUP

JSF
Configuration

- **WEB-INF/web.xml**
  - Paramètres javax.faces.*
  - Servlet javax.faces.webapp.FacesServlet
- **WEB-INF/faces-config.xml**
  - Permet de configurer JSF
  - Permet de surcharger le fonctionnement par défaut de JSF. ESUP-Commons l'utilise par ex. pour :
    - La résolution des EL (messages mais aussi l'accès aux beans Spring -en remplacement de ceux gérés par JSF-)
    - Certaines fonctionnalités (support Spring en mode portlet, vues mobiles, exceptions, etc.)
  - Pourrait être utilisé pour la navigation
- **WEB-INF/navigation-rules.xml**
  - Navigation

JSF
Affichage d'une page

Id	Titre
2	lw - Titre de tache
4	vx - Titre de tache
5	kb - Titre de tache
6	tm - Titre de tach
7	ma - Titre de tach

```
<?xml version="1.0" encoding="UTF-8"?
<!DOCTYPE html .../...
<ui:composition .../..
 template="/stylesheets/pageTemplate.xhtml"
 <ui:define name="content"
  <h:dataTable border="1" var="task"
   value="#{taskController.publicTasks}"
   rendered="#{!empty taskController.publicTasks}">
   <h:column>
    <f:facet name="header">
     <h:outputText value="Id" />
    </f:facet>
    <h:outputText value="#{task.id}" />
   </h:column>

   .../...
```

JSF
Affichage d'une page 2

- **Vue**
  - <h:dataTable border="1" var="task"
    value="**#{taskController.publicTasks}**"
    rendered="**#{!empty taskController.publicTasks}**">
- **Spring**
  - properties/web/controllers.xml
    - <bean id="taskController"
      class="org.esupportail.formation.web.controllers.TaskController"
      parent="abstractContextAwareController"
      scope="session"/>
- **Java**
  - TaskController.java
    - public List<Task> getPublicTasks() {
      return getDomainService().getPublicTasks();
      }

JSF
Affichage d'une page 3

- **Vue**
  - <h:outputText value="**#{task.id}**" />
- **Provenance de l'objet task**
  - <h:dataTable border="1" var="**task**"
    value="**#{taskController.publicTasks}**"
    rendered="**#{!empty taskController.publicTasks}**"
- **Java**
  - Task.java (dans le module esup-formation-domain-beans)
    - public long getId() {
      return id;
      }

JSF
Navigation


```
<h:commandLink
  action="#{userController.goToUserManagerPage}>
  <h:outputText value="gestion des utilisateurs"/>
</h:commandLink>
```

```
Contrôleur :
public String goToUserManagerPage() {
  return "go_userManagerPage";
}
```

```
<navigation-case>
  <from-outcome>go_userManagerPage</from-outcome>
  <to-view-id>/stylesheets/userManager.xhtml</to-view-id>
  <redirect/>
</navigation-case>
```

JSF
Facelet

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE .../...>
<html .../...>
 <f:view locale="#{sessionController.locale}">
  <h:head>
   <title>Titre</title>
  </h:head>
  <h:body styleClass="fl-theme-app">
   <div id="body">
    <div class="app-body">
     <ui:insert name="content" />
.../...
```

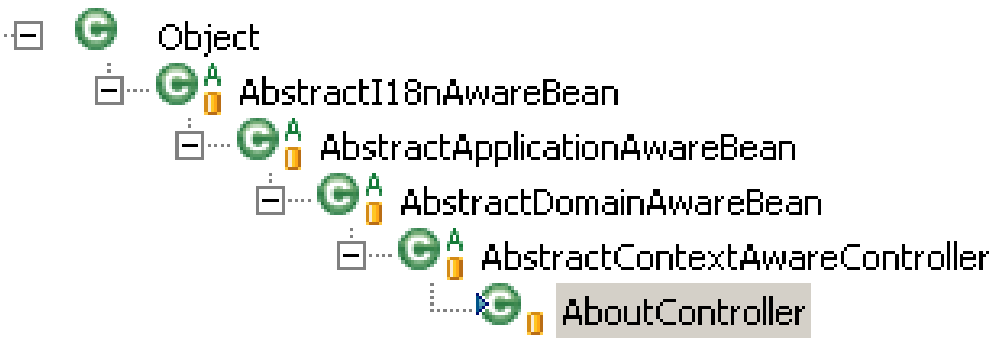
Template utilisé par la vue

Définition des zones du template

```
<?xml version="1.0" encoding="UTF-8"?
<!DOCTYPE html .../...
<ui:composition .../..
 template="/stylesheets/pageTemplate.xhtml"
 <ui:define name="content"
  <h:dataTable border="1" var="task"
   value="#{taskController.publicTasks}"
   rendered="#{!empty taskController.publicTasks}">
   <h:column>
.../...
```

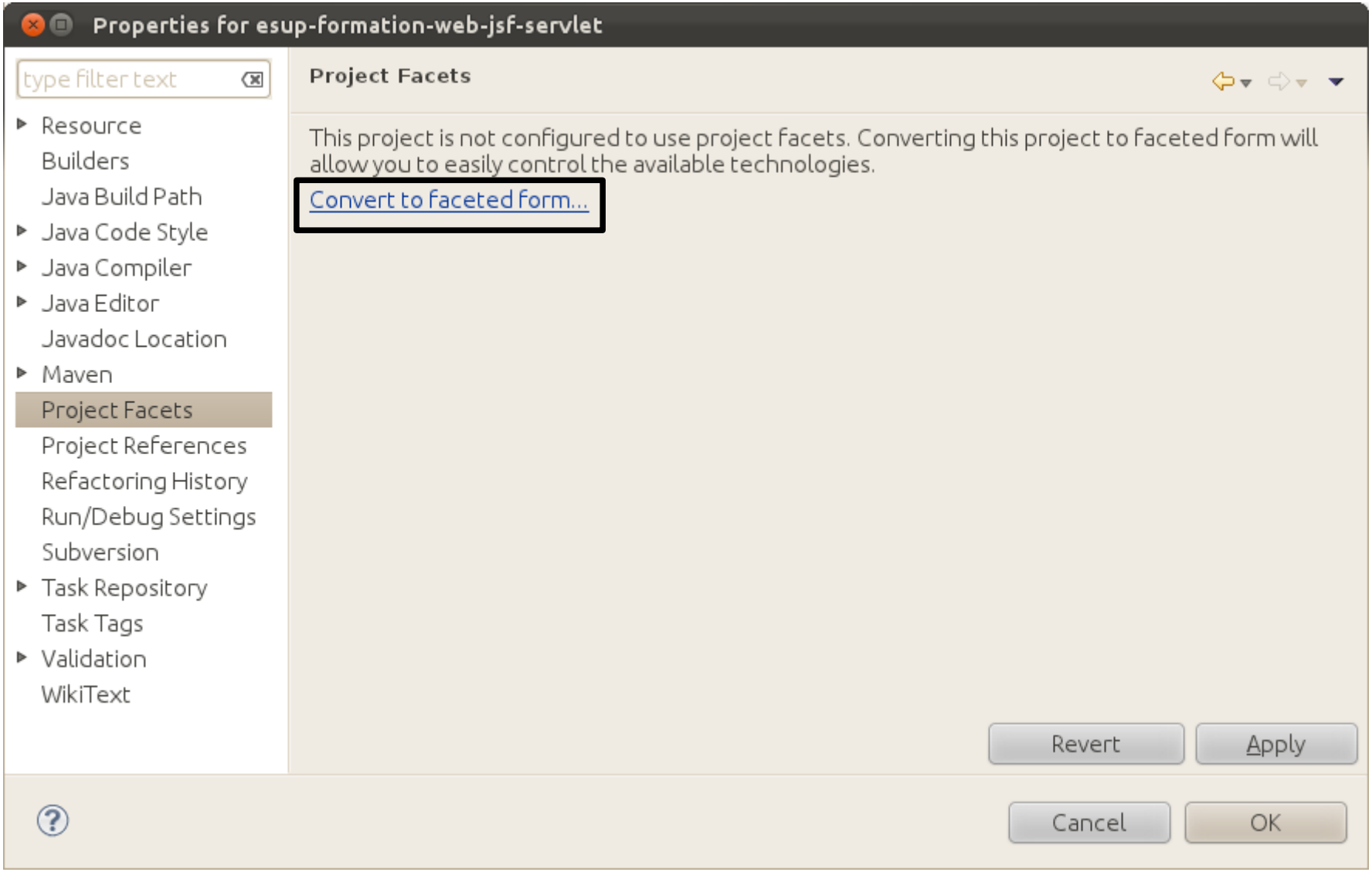
JSF
i18n

- **Configuration**
  - Properties/i18n/i18n.xml
  - Fichier de properties (_en, _fr)
    - Custom <-- Messages <-- Commons
    - MESSAGE.USER_NOT_FOUND = utilisateur {0} non trouvé !
  - Accès
    - Depuis le code java
      - Ajout d'un message JSF
        - addErrorMessage("uid", "MESSAGE.USER_NOT_FOUND", userId);
      - Récupération d'une chaîne
        - msg = getString("MY.MESSAGE" [, locale] [, param]*);
    - Depuis une vue
      - <h:outputText value="#{msgs['TASK.MANAGER.NOTASKS']}" />

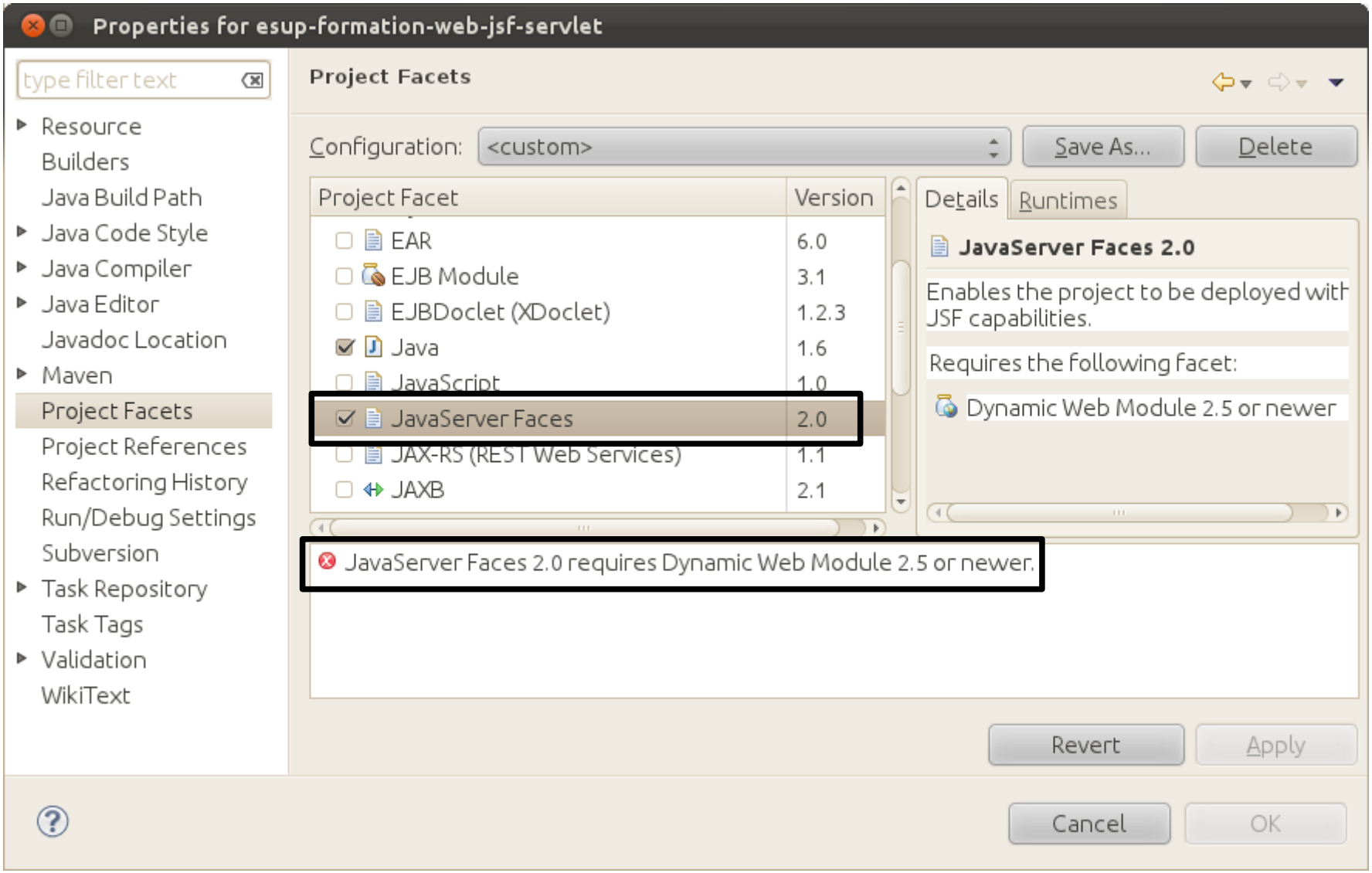


Astuce JSF

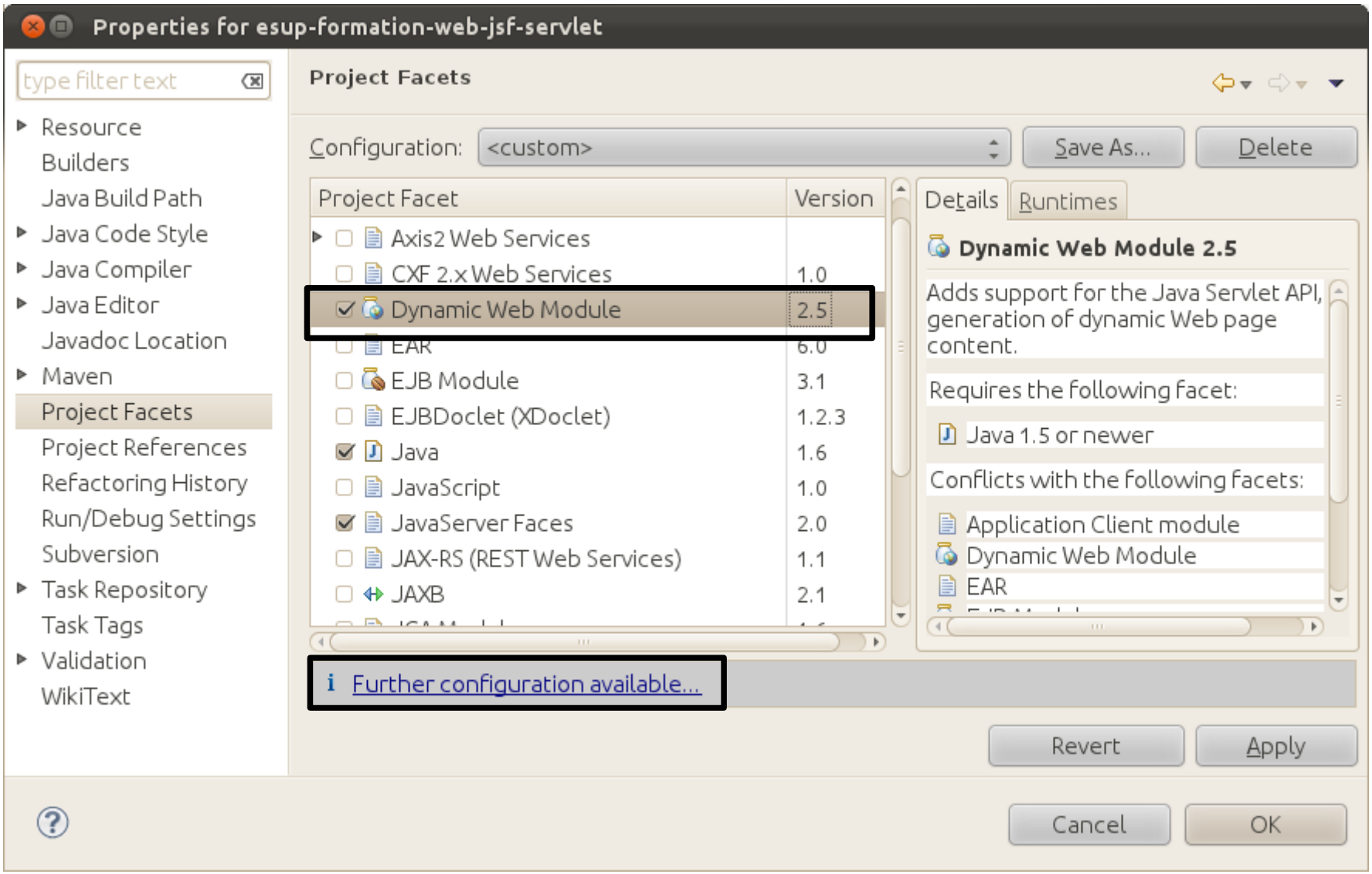
Les recettes
Complètement du code JSF



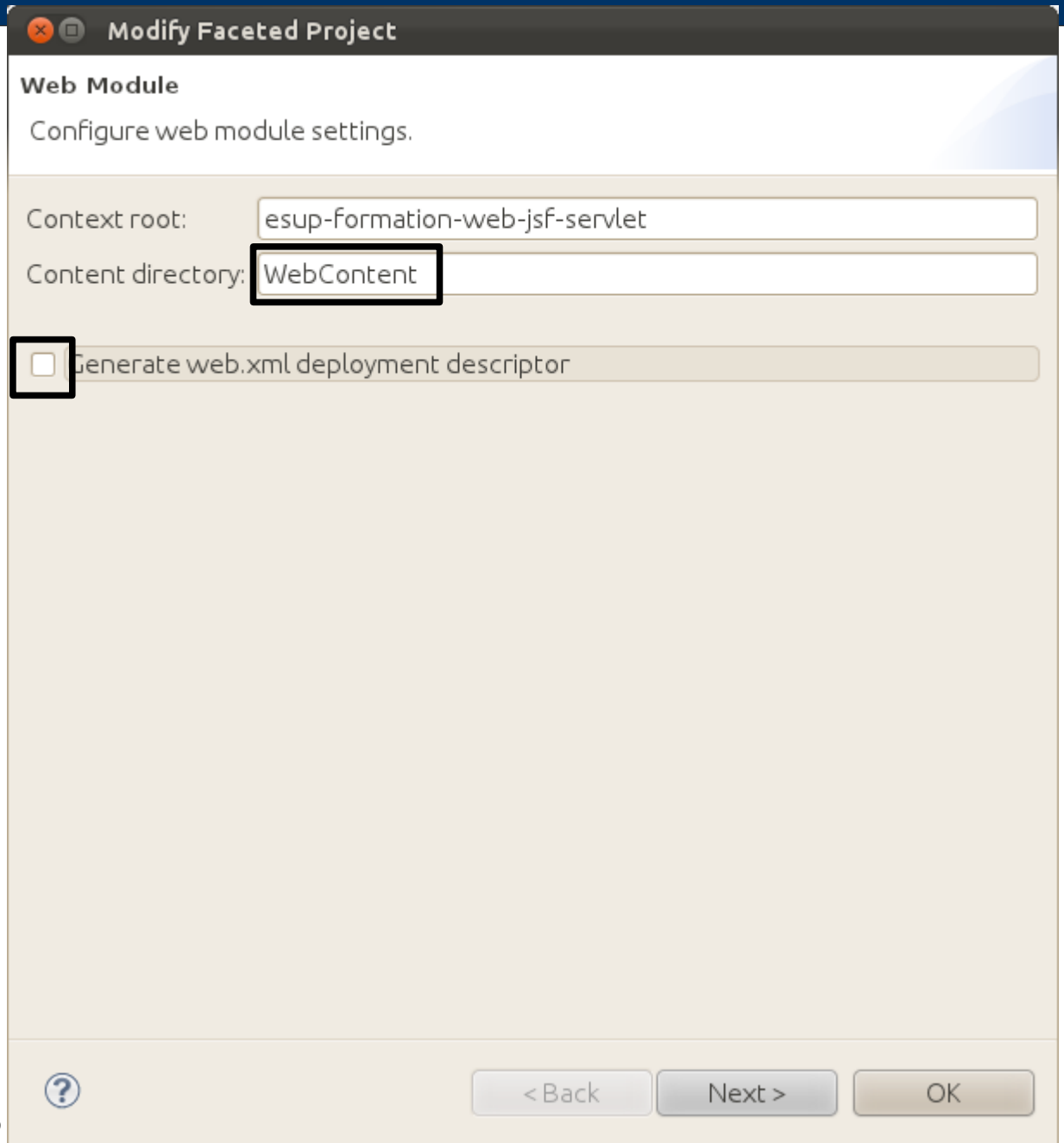
Les recettes
Complètement du code JSF 2



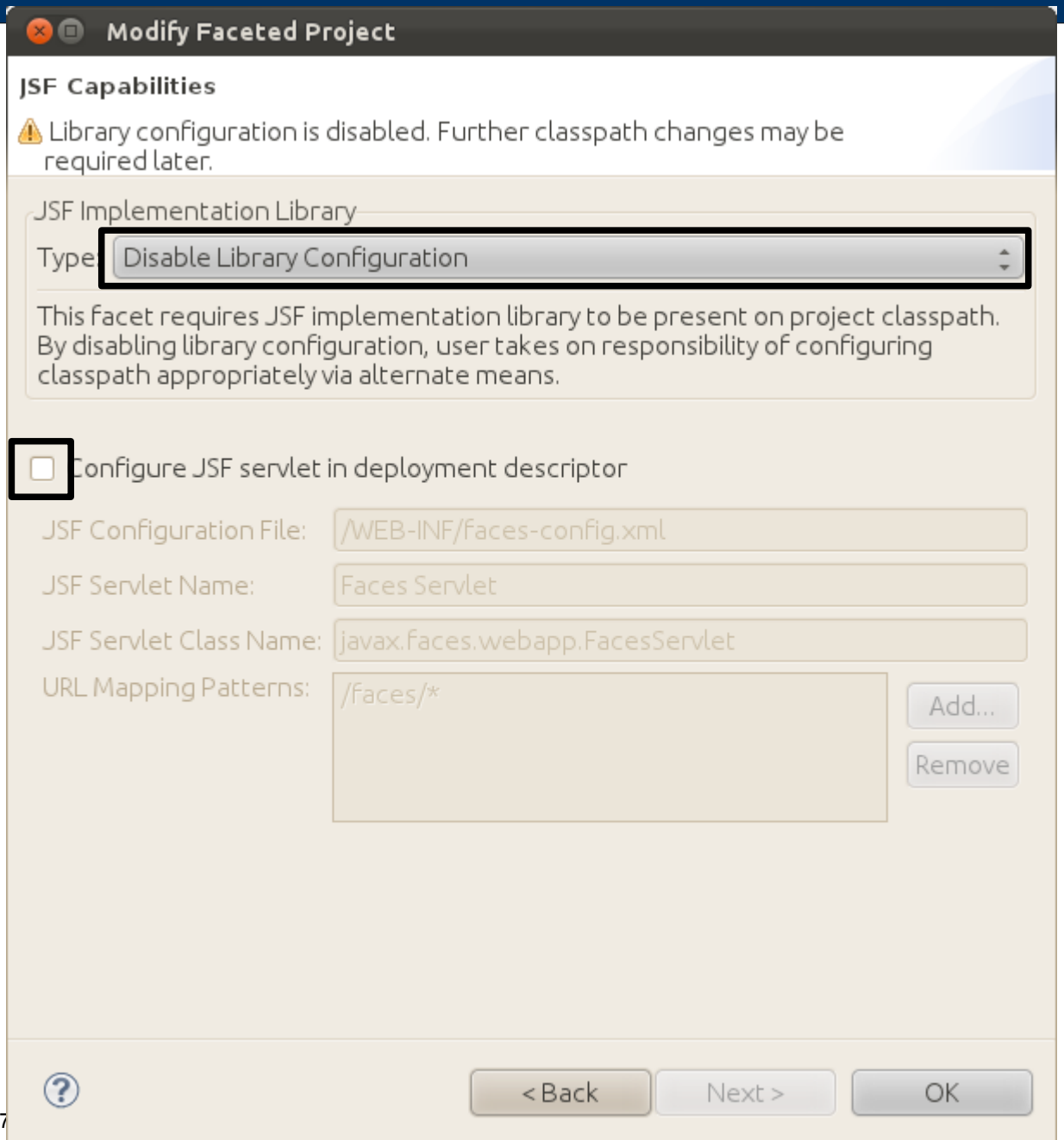
Les recettes
Complètement du code JSF 3



Les recettes
Complètement du code JSF 4

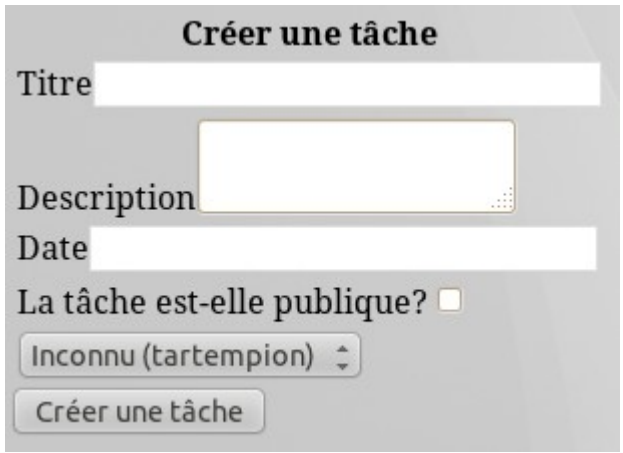


Les recettes
Complètement du code JSF 5



Les formulaires JSF

JSF
Affichage d'un formulaire




```
<?xml version="1.0" encoding="UTF-8"?
<!DOCTYPE html .../...
<ui:composition .../..
 template="/stylesheets/pageTemplate.xhtml"
 <ui:define name="content"
  <h:form>
   <h:panelGrid id="grid" columns="1">
    <h:panelGroup>
     <h:outputLabel for="title"
      value="Titre" />
     <h:inputText id="title"
      value="#{taskController.currentTask.title}"
      required="#{true}" />
     <h:message for="title" />
    </h:panelGroup>
.../...
    <h:commandButton action="#{taskController.addTask}"
     value="Créer une tâche" />
.../...
```

JSF
Affichage d'un formulaire 2

- **Vue**
  - <h:inputText id="title"
    value="**#{taskController.currentTask.title}**"
    required="#{true}" />
- **Spring**
  - properties/web/controllers.xml
    - Définition de taskController
- **Java**
  - TaskController.java
    - private Task currentTask = new Task();
    - public Task getCurrentTask() {
      return currentTask;}
  - Task.java
    - public void setTitle(String title) {
      this.title = title;}

JSF
Affichage d'un formulaire 3

- **Vue**
  - <h:commandButton action="**#{taskController.addTask}**"
    value="#{msgs['TASK.MANAGER.ADD.TASK']}" />
- **Spring**
  - properties/web/controllers.xml
    - Définition de taskController
- **Java**
  - TaskController.java
    - public void addTask() {
      getDomainService().addTask(currentTask);
      }
    - NB : addTask() pourrait renvoyer une chaîne
      - CF. chapitre « JSF - Navigation »

JSF
Convertisseurs

- **Souvent implicites**
  - Interger, String, Boolean, etc.
- **Fournis par JSF (ou des librairies tierces)**
  - <h:inputText id="date"
    value="#{taskController.currentTask.date}">
    <f:convertDateTime type="date" pattern="ddMMyyyy"/>
    </h:inputText>
- **Ecrits spécifiquement**
  - <h:selectOneListbox value="#{var}" converter="#{userConverter}">
  - UserConverter est un bean Spring
  - public class UserConverter implements Converter
    - public Object getAsObject
    - public String getAsString
- **Immediate=true**
  - Si un bouton a cet attribut les conversions et validations sont ignorées (cas du bouton annuler par exemple)

JSF
Validateurs

- **JSF offre un mécanisme proche des convertisseurs**
  - <f:validateLength minimum="5"/>
  - validator="#{taskController.validateTitle}
    - public void validateTitle() throws ValidationException
- **JSF 2 est aussi compatible avec JSR 303**
  - Ajout de la dépendance
    - org.hibernate:hibernate-validator:4.0.2.GA
  - On annote directement l'objet lié au formulaire
    - @NotNull
      @Size(max = 15, min = 5, message="{err1}")
      private String title;
    - Err1 étant défini dans un fichier ValidationMessages_xx.properties dans le classpath

Etc. JSF

JSF
Ajax

- **Avec JSF 2**
  - <h:inputText id="title"
    value="#{taskController.currentTask.title"
    required="#{true}">
    <f:ajax event="**keyup**" render="**outAjax**" />
    </h:inputText>
    <h:outputText id="**outAjax**"
    value="#{taskController.currentTask.title}" />
- **Via des librairies tierces**

JSF
Exceptions

- **Configuration**
  - JSF (WEB-INF/faces-config.xml)
    - <exception-handler-factory>
      org.esupportail.commons.exceptions.ExceptionHandlerFactory
      </exception-handler-factory>
  - Spring (properties/exceptionHandling/exceptionHandling.xml)
    - ExceptionViews
      - Type d'exception --> règle de navigation
- **La vue d'exception**
  - Contient un appel à #{exceptionController.restart}
    - org.esupportail.commons.web.controllers.ExceptionController
    - Méthode restart()
      - Appelle la méthode reset() de tous les beans implémentant Resettable
      - Renvoie la valeur applicationRestarted à traiter par les règles de navigation

URL directe

URL directe
Généralité

- **Contexte**
  - Imaginé dans le cadre de JSF qui ne gérait pas bien cette fonctionnalité en version 1.X
- **Objectifs**
  - Permettre de générer des liens directs
    - A mettre dans les vues et/ou dans des mails
  - Obtenir des liens qui fonctionnent en servlet et/ou en portlet et avec ou sans CAS et/ou shib
- **Remarques JSF 2**
  - JSF 2 intègre de base et simplement une fonctionnalité équivalente
    - Pas décrite ici (utilisation de balises f:viewParam)
    - Sans doute non fonctionnelle en mode portlet (à vérifier)
- **Principe**
  - Un générateur d'URL et un intercepteur d'URL

URL directe
Configuration

- **Configuration**
  - Générateur d'URL
    - Dans properties/deepLinking/urlGenerator.xml
    - <bean id="servletUrlGenerator"
      class="...ServletUrlGeneratorImpl" lazy-init="true">
      <property name="casUrl" value="${cas.url}" />
      <property name="casLoginUrl"
        value="${urlGeneration.casLoginUrl}" />
      <property name="servletCasLoginUrl"
        value="${urlGeneration.casLoginUrl}" />
      <property name="servletGuestUrl"
        value="${urlGeneration.servletGuestUrl}" />
      </bean>
  - Il existe d'autres implémentations
    - Ex : UportalUrlGeneratorImpl

URL directe
Configuration 2

- **Configuration**
  - intercepteur d'URL
    - Via un phase listener dans webapp/WEB-INF/faces-config.xml
      - <phase-listener>
        org.esupportail.commons.jsf.DeepLinkingPhaseListener
        </phase-listener>
    - Dans properties/deepLinking/deepLinking.xml
      - Déclaration de UrlPatternDescriptor

URL directe Configuration 3

- **UrlPatternDescriptor**
  - <bean id="urlTaskDetail"
    class="org.esupportail.commons.jsf.UrlPatternDescriptor">
    <property name="params">
    <list><value>taskId</value></list>
    </property>
    <property name="actionBinding" >
    <bean class="org.esupportail.commons.jsf.ActionBinding">
    <property name="action" value="taskController.goUrlTask"/>
    <property name="args">
    <list><value>java.lang.String</value></list>
    </property>
    </bean>
    </property>
    </bean>
  - Définit :
    - Une liste de paramètres permettant la sélection de cet UrlPatternDescriptor
    - une action à exécuter (sous forme d'une EL)

URL directe
Utilisation

- **Génération de l'URL directe**
  - Code
    - Map<String, String> params = new HashMap<String, String>();
      params.put("taskId", "11");
      String url = getUrlGenerator().guestUrl(params);
  - Existent aussi casUrl() et shibbolethUrl()
- **Action liée à une URL directe**
  - Code
    - public String goUrlTask(String taskId){
      detailledTask = getTaskFromDomainService(
      new Long(taskId).longValue());
      return "go_taskDetailPage";}
  - Renvoie une navigation rule

Gestion du mail

Gestion du mail

- **Configuration**
  - properties/smtp/smtp.xml
    - Définit un smtpService
- **Utilisé de base dans les exceptions**
- **Appel depuis du code**
  - smtpService.send(
    new InternetAddress("celine.didier@uhp-nancy.fr"),
    "CREATION D'UNE TACHE",
    "Une nouvelle tâche vient d'être créée",
    "Une nouvelle tâche vient d'être créée");
  - Possibilité de joindre des fichiers

Authentification

Authentification

- **Configuration**
  - Dans le fichier auth.xml du module domain-services on trouve :
    - Un bean authenticator dont la classe (AuthenticatorImpl) implémente l'interface Authenticator et donc la méthode User getUser()
      - Authenticator se charge de retrouver les informations de l'utilisateur (objet EC AuthInfo), de les mettre en cache et de les injecter dans un objet métier de l'application (Ex : User)
    - Un bean authenticationService (référencé par authenticator)
      - Dans EC il existe différentes implémentations de authenticationService
        - CasFilterAuthenticationService
        - OfflineFixedUserAuthenticationService
- **Dans le code**
  - On utilise authenticator qui renvoie un objet métier
    - if (authenticator.getUser() != null)
      login = authenticator.getUser().getLogin();
      else
      login "Invité";

LDAP

LDAP

- **Configuration Maven**
  - Ajouter cette dépendance :
    - <dependency>
      <groupId>org.esupportail</groupId>
      <artifactId>esup-commons2-ldap</artifactId>
      <version>${esupcommons.version}</version>
      <exclusions>
       <exclusion>
        <artifactId>spring-tx</artifactId>
        <groupId>org.springframework</groupId>
       </exclusion>
      </exclusions>
      </dependency>
    - On est obligé de mettre cette exclusion car la version de Spring LDAP qui est utilisée par esup-commons2-ldap fait référence à une ancienne version de Spring

LDAP

- **Configuration Spring**
  - Dans un fichier (à créer) ldap.xml on trouvera :
    - Un bean ldapUserService utilisant SearchableLdapUserServiceImpl fourni par EC. Il a comme propriétés :
      - Des paramètres de configuration (attributs à rechercher, etc.)
      - Des services EC (i18nService, cacheManager)
      - Une référence à ldapTemplate
    - Un bean ldapTemplate utilisant LdapTemplate fourni par Spring.
      - Il référence un contextSource
    - Un bean contextSource utilisant MultiUrlLdapContextSource fourni par EC
      - En fait MultiUrlLdapContextSource étend LdapContextSource de Spring en facilitant la saisie de n sources LDAP
      - Il permet de configurer la connexion au LDAP (url, user, passwd, timeout, etc.)
- **Dans le code**
  - LdapUser uLdap = ldapUserService.getLdapUser(user.getLogin());
    String displayName=uLdap.getAttribute("cn");

Web Services

Web Services

- **Généralité**
  - Depuis EC2 on n'utilise plus Xfire mais CXF
    - CXF est une implémentation possible de JAX-WS
- **Configuration Maven**
  - Ajouter cette dépendance :
    - <dependency>
      <groupId>org.esupportail</groupId>
      <artifactId>esup-commons2-ws-cxf</artifactId>
      <version>${esupcommons.version}</version>
      <type>pom</type>
      </dependency>

Web Services 2

- **Configuration Spring**
  - Dans le fichier domain.xml ajouter :
    - Un espace de nom jaxws
      - xmlns:jaxws="http://cxf.apache.org/jaxws"
      - xsi:schemaLocation="http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd »
    - Des imports spécifiques à CXF
      - <import resource="classpath:META-INF/cxf/cxf.xml" />
<import resource="classpath:META-INF/cxf/cxf-extensionsoap.xml" />
<import resource="classpath:META-INF/cxf/cxf-servlet.xml" />
    - Une déclaration pour configurer l'URL de réponse et la classe d'implémentation du WS
      - <jaxws:endpoint id="domainService.remoteService" implementor="**#domainService**" address="**/TodoService**">
</jaxws:endpoint>

Web Services 3

- **Configuration servlet**
  - Dans le fichier web.xml ajouter :
    - La déclaration de la servlet
      - <servlet>
        <servlet-name>CXFServlet</servlet-name>
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
        <servlet-name>CXFServlet</servlet-name>
        <url-pattern>/services/*</url-pattern>
        </servlet-mapping>
- **Dans le code**
  - Sur l'interface DomainService :
    - @WebService
    - C'est tout !

Web Services
Génération du code client

- **Dans le pom.xml du client**
  - <plugin>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-codegen-plugin</artifactId>
    <executions>
    <execution>
    <id>generate-sources</id>
    <phase>generate-sources</phase>
    <configuration>
    <wsdlOptions>
    <wsdlOption>
    <wsdl>${basedir}/src/main/etc/TodoService.wsdl</wsdl>
    </wsdlOption>
    </wsdlOptions>
    </configuration>
    <goals>
    <goal>wsdl2java</goal>
    </goals>
    </execution>
    </executions>
    </plugin>

Web Services
Génération du code client 2

- **Dans le code**
  - DomainService ds = new DomainServiceImplService(
    new URL(wsdl)).getDomainServiceImplPort();
    - DomainService fait partie du code généré
    - wsdl est l'URL du WS à utiliser
      - pas forcément celui ayant servi à la génération du code
  - List<Task> tasks = ds.getTasks();

Services REST

REST

- **Généralité**
  - Representational State Transfer
    - On utilise les méthodes HTTP (Get, Post, Delete, etc.)
    - Les données sont formatées en XML ou JSON
  - On utilise aussi CXF pour REST
    - CXF est une implémentation possible de JAX-RS
- **Configuration Maven**
  - Ajouter cette dépendance :
    - <dependency>
      <groupId>org.esupportail</groupId>
      <artifactId>esup-commons2-rs-cxf</artifactId>
      <version>${esupcommons.version}</version>
      <type>pom</type>
      </dependency>

REST 2

- **Configuration Spring**
  - Dans le fichier domain.xml ajouter :
    - Un espace de nom jaxrs
      - xmlns:jaxrs="http://cxf.apache.org/jaxrs"
      - http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd
    - Des imports spécifiques à CXF
      - <import resource="classpath:META-INF/cxf/cxf.xml" />
      - <import resource="classpath:META-INF/cxf/cxf-servlet.xml" />

REST 3

- **Configuration Spring 2**
  - Une déclaration pour configurer l'URL de réponse
    - <jaxrs:server id="domainServiceRest"
      address="/rest">
      <jaxrs:serviceBeans>
        <ref bean="domainService" />
      </jaxrs:serviceBeans>
      <jaxrs:providers>
        <ref bean="jsonProvider" />
      </jaxrs:providers>
    </jaxrs:server>
  - La déclaration du provider JSON
    - <bean id="jsonProvider"
      class="org.codehaus.jackson.jaxrs.JacksonJaxbJsonProvider" />

REST 4

- **Configuration servlet**
  - Dans le fichier web.xml ajouter :
    - La déclaration de la servlet
      - la même que pour les Web Services (car c'est toujours CXF)
- **Dans le code**
  - Sur l'interface DomainService :
    - Au niveau de la classe
      - @Path("/domainService/")
      - @Produces("application/json")
    - Sur les méthodes
      - @GET @Path("/users/{id}")
        User getUser(@PathParam("id") String id)
      - @GET @Path("/users")
        List<User> getUsers()

Déploiement portlet

Déploiement portlet
Les étapes

- **Construction du war**
  - Mvn package
- **Déploiement du war dans le serveur d'application**
  - Dans l'environnement ESUP-Porail
    - Ant portlet.deploy -DportletApp=/chemin/targetproj.war
    - NB : Cette commande modifie le web.xml de votre projet
      - Ajout d'une servlet (org.apache.pluto.core.PortletServlet)
- **Création d'un contexte tomcat pour ce war**
  - Modification sur server.xml
- **Enregistrement de cette portlet dans uPortal**

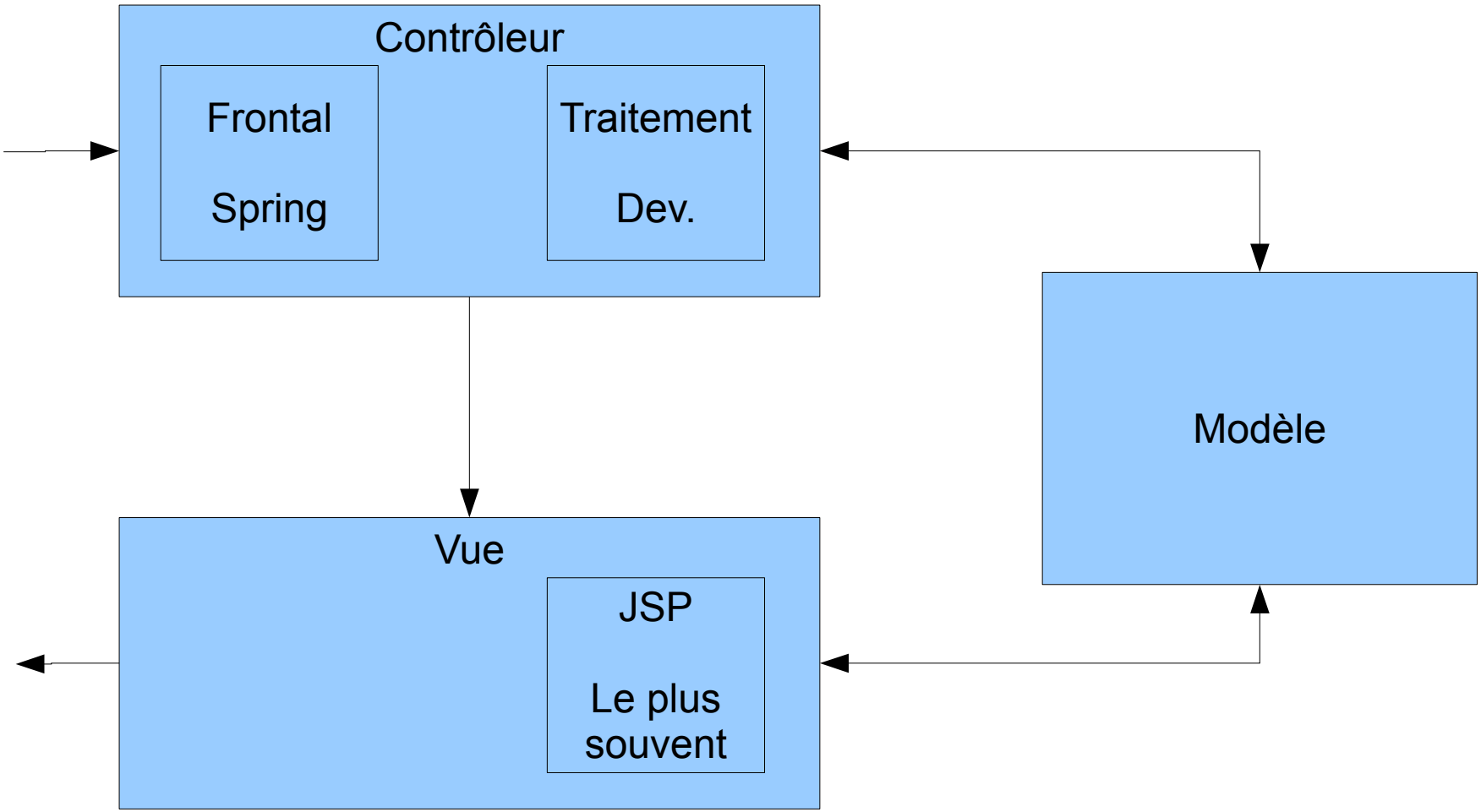
Déploiement portlet
Spécificités liées aux portlets

- **Accès aux attributs utilisateurs**
  - Portlet.xml
    - portlet/user-attribute/name
  - Usage
    - L'uid récupéré par le portletAuthenticationService de EC2
- **Les préférences**
  - Portlet.xml
    - portlet/portlet-preferences/preference/name et value
  - Usage
    - Moyen simple de configurer une portlet
      - Un administrateur peut éditer une préférence au moment de l'installation
        - N instances possibles avec des configurations différentes
    - On peut rendre une préférence modifiable par l'utilisateur
      - A nous de créer la vue pour saisir l'information
      - Mais le portail se charge de la persistance en base

Spring-MVC

Spring-MVC
Généralité

- **Patron MVC2**



Spring-MVC Configuration

- **Contrôleur frontal**
  - Portlet.xml
    - <portlet-class>
      org.springframework.web.portlet.DispatcherPortlet
      </portlet-class>
  - Web.xml
    - <servlet-class>
      org.springframework.web.servlet.ViewRendererServlet
      </servlet-class>
    - Sur http://www.ibm.com/developerworks/websphere/library/techarticles/0802_patil-pt1/0802_patil-pt1.html
      - The ViewRendererServlet is the bridge servlet for portlet support. During the render phase, DispatcherPortlet wraps PortletRequest into ServletRequest and forwards control to ViewRendererServlet for actual rendering. This process allows Spring Portlet MVC Framework to use the same View infrastructure as that of its servlet version

Spring-MVC Configuration 2

- **Contrôleur de traitement**
  - PortletContext.xml
    - NB : Ce fichier est référencé par le paramètre contextConfigLocation de la DispatcherPortlet dans le portlet.xml
    - <context:component-scan base-package="org.esupportail.truc"/>
      - Permet la détection des @Controller dans le package org.esupportail.truc
      - NB : Le DispatcherPortlet défini précédemment permet aussi la détection des @RequestMapping positionnés dans le contrôleur pour répondre aux différentes requêtes

Spring-MVC Configuration 3

- **Résolution des vues**
  - ApplicationContext.xml
    - <bean id="jspViewResolver"
      class="...**servlet**.view.InternalResourceViewResolver">
      <property name="cache" value="true"/>
      <property name="viewClass"
      value="...**servlet**.view.JstlView"/>
      <property name="prefix" value="/WEB-INF/jsp/"/>
      <property name="suffix" value=".jsp"/>
      </bean>

Spring-MVC
Exemple de page

- ```
  <%@ include file="/WEB-INF/jsp/include.jsp"%>

    <h2>${userFromEC2.login} !</h2>

      <ul>
        <c:forEach var="task" items="${taskList}" >
          <li>${task.id} - ${task.title}</li>
        </c:forEach>
      </ul>

  ```
  ```

Spring-MVC
i18n

- **Configuration**
  - Comme avec JSF via un bean Spring :
    - <bean id="messageSource"
      class="...ResourceBundleMessageSource">
      - NB : Le bean DOIT se nommer messageSource
- **Dans la vue**
  - <spring:message code="view.ticket.message.alarm"/>

Spring-MVC Navigation

- **Annotation du contrôleur**
  - Les méthodes sont annotées par @RequestMapping
    - Ex. portlet : @RequestMapping("VIEW")
  - Ces méthodes renvoient un objet ModelAndView qui contient :
    - Le nom de la vue à utiliser
    - Une map contenant des objets manipulables dans la vue.
      - Ex :
        - ModelMap model = new ModelMap();
List<Task> list= domainService.getTasks(...)
model.put("taskList", list);
return new ModelAndView("view", model);
        - <c:forEach var="task" items="${taskList}" >

Licence