Référentiel Coordonnées JAVA

Architecture et solutions techniques

Thomas Duchatelle (thomas.duchatelle-ext@yrnet.com)

Yves Rocher

February 15, 2013

Sommaire

Sources sous gestionnaire de version

Subversion, ou SVN

SVN

SVN est un gestionnaire de version. Il conserve l'historique de modifications des sources associés à des méta-données : historique, auteur, ...

Sources sous gestionnaire de version

Subversion, ou SVN

SVN

SVN est un gestionnaire de version. Il conserve l'historique de modifications des sources associés à des méta-données : historique, auteur, ...

URL :

```
1  http://subversion.yvesrocher.com:9030/REFERENTIEL_COORDONNEES/
```

Organisation des branches/tags

`trunk`, `branches`, `tags`, ...

Le *repository* s'organise suivant la convention SVN :

trunk Version en cours de développement

Organisation des branches/tags

`trunk`, `branches`, `tags`, ...

Le *repository* s'organise suivant la convention SVN :

- **trunk** Version en cours de développement
- **tags** Snaphot des sources pour chaque version livrée
  Exemple de nommage : `T_V3.1.1_RELEASE_Mep_2013_01_31`

Organisation des branches/tags

`trunk`, `branches`, `tags`, ...

Le *repository* s'organise suivant la convention SVN :

trunk Version en cours de développement

tags Snaphot des sources pour chaque version livrée
Exemple de nommage : `T_V3.1.1_RELEASE_Mep_2013_01_31`

branches Copies des tags afin d'apporter des corrections (support)

Outils et frameworks utilisés

Spring Inversion de contrôle, gestion des contextes, transactions

BDD

Outils et frameworks utilisés

Spring	Inversion de contrôle, gestion des contextes, transactions BDD
Hibernate	*Mapping Relationnel Objet*

Outils et frameworks utilisés

Spring	Inversion de contrôle, gestion des contextes, transactions BDD
Hibernate	*Mapping Relationnel Objet*
Hibernate Validator	Validation des données

Outils et frameworks utilisés

Spring	Inversion de contrôle, gestion des contextes, transactions BDD
Hibernate	*Mapping Relationnel Objet*
Hibernate Validator	Validation des données
SLF4J	API de log, utilise *LOG4J* en backend

Outils et frameworks utilisés

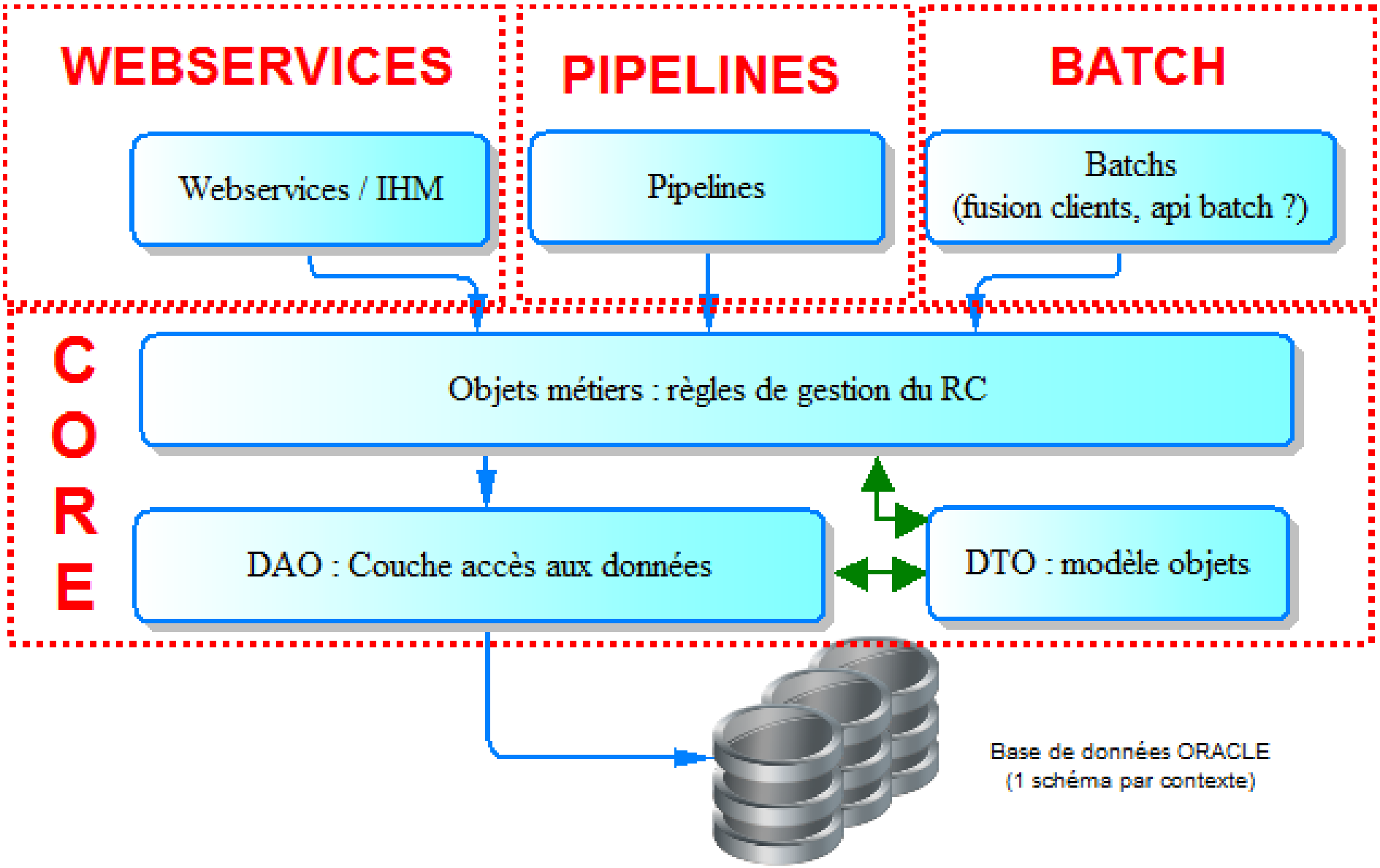
Spring	Inversion de contrôle, gestion des contextes, transactions BDD
Hibernate	*Mapping Relationnel Objet*
Hibernate Validator	Validation des données
SLF4J	API de log, utilise *LOG4J* en backend
Commons Apache	Utilitaires sur les chaines de caractères, la détection des fichiers, ...

Outils et frameworks utilisés

Spring	Inversion de contrôle, gestion des contextes, transactions BDD
Hibernate	*Mapping Relationnel Objet*
Hibernate Validator	Validation des données
SLF4J	API de log, utilise *LOG4J* en backend
Commons Apache	Utilitaires sur les chaines de caractères, la détection des fichiers, ...
JUNIT	Tests unitaires (utilisé avec *Mockito* et *FestAssert*)

Référentiel Coordonnées

1 application, 4 modules



Modules du référentiel coordonnées

Core Coeur de l'application (jar) :

Modules du référentiel coordonnées

Core Coeur de l'application (jar) :

- Modèle objets : représentation *client* / *coordonnées*

Modules du référentiel coordonnées

Core Coeur de l'application (jar) :

- Modèle objets : représentation *client* / *coordonnées*
- couche d'accès au données : structure de la base, gestion des schémas

Modules du référentiel coordonnées

Core Coeur de l'application (jar) :

- Modèle objets : représentation *client* / *coordonnées*
- couche d'accès au données : structure de la base, gestion des schémas
- règles de gestion : calcul des consentements, règles de mises à jour, dé-duplication

Modules du référentiel coordonnées

Core Coeur de l'application (jar) :

- Modèle objets : représentation *client* / *coordonnées*
- couche d'accès au données : structure de la base, gestion des schémas
- règles de gestion : calcul des consentements, règles de mises à jour, dé-duplication

Webservices Application Java EE (war) exposant des webservices

Modules du référentiel coordonnées

Core Coeur de l'application (jar) :

- Modèle objets : représentation *client* / *coordonnées*
- couche d'accès au données : structure de la base, gestion des schémas
- règles de gestion : calcul des consentements, règles de mises à jour, dé-duplication

Webservices Application Java EE (war) exposant des webservices

- correspondance *modèle objet interne* $\leftrightarrow$ *contrat WSDL*

Modules du référentiel coordonnées

Core Coeur de l'application (jar) :

- Modèle objets : représentation *client* / *coordonnées*
- couche d'accès au données : structure de la base, gestion des schémas
- règles de gestion : calcul des consentements, règles de mises à jour, dé-duplication

Webservices Application Java EE (war) exposant des webservices

- correspondance *modèle objet interne* $\leftrightarrow$ *contrat WSDL*
- Interface WEB : suivi des pipelines

Modules du référentiel coordonnées

Core Coeur de l'application (jar) :

- Modèle objets : représentation *client* / *coordonnées*
- couche d'accès au données : structure de la base, gestion des schémas
- règles de gestion : calcul des consentements, règles de mises à jour, dé-duplication

Webservices Application Java EE (war) exposant des webservices

- correspondance *modèle objet interne* $\leftrightarrow$ *contrat WSDL*
- Interface WEB : suivi des pipelines

Pipelines Intégration de données par fichiers CSV (jar exécutable)

Modules du référentiel coordonnées

Core Coeur de l'application (jar) :

- Modèle objets : représentation *client* / *coordonnées*
- couche d'accès au données : structure de la base, gestion des schémas
- règles de gestion : calcul des consentements, règles de mises à jour, dé-duplication

Webservices Application Java EE (war) exposant des webservices

- correspondance *modèle objet interne* $\leftrightarrow$ *contrat WSDL*
- Interface WEB : suivi des pipelines

Pipelines Intégration de données par fichiers CSV (jar exécutable)

- Démarrage/Arrêt d'un *démon*

Modules du référentiel coordonnées

Core Coeur de l'application (jar) :

- Modèle objets : représentation *client* / *coordonnées*
- couche d'accès au données : structure de la base, gestion des schémas
- règles de gestion : calcul des consentements, règles de mises à jour, dé-duplication

Webservices Application Java EE (war) exposant des webservices

- correspondance *modèle objet interne* $\leftrightarrow$ *contrat WSDL*
- Interface WEB : suivi des pipelines

Pipelines Intégration de données par fichiers CSV (jar exécutable)

- Démarrage/Arrêt d'un *démon*
- Détection de l'arrivée de fichiers

Modules du référentiel coordonnées

Core Coeur de l'application (jar) :

- Modèle objets : représentation *client* / *coordonnées*
- couche d'accès au données : structure de la base, gestion des schémas
- règles de gestion : calcul des consentements, règles de mises à jour, dé-duplication

Webservices Application Java EE (war) exposant des webservices

- correspondance *modèle objet interne* $\leftrightarrow$ *contrat WSDL*
- Interface WEB : suivi des pipelines

Pipelines Intégration de données par fichiers CSV (jar exécutable)

- Démarrage/Arrêt d'un *démon*
- Détection de l'arrivée de fichiers
- EAI et lecture de fichier

Modules du référentiel coordonnées

Core Coeur de l'application (jar) :

- Modèle objets : représentation *client* / *coordonnées*
- couche d'accès au données : structure de la base, gestion des schémas
- règles de gestion : calcul des consentements, règles de mises à jour, dé-duplication

Webservices Application Java EE (war) exposant des webservices

- correspondance *modèle objet interne* $\leftrightarrow$ *contrat WSDL*
- Interface WEB : suivi des pipelines

Pipelines Intégration de données par fichiers CSV (jar exécutable)

- Démarrage/Arrêt d'un *démon*
- Détection de l'arrivée de fichiers
- EAI et lecture de fichier
- Parallélisation des traitements

Modules du référentiel coordonnées

Core Coeur de l'application (jar) :

- Modèle objets : représentation *client* / *coordonnées*
- couche d'accès au données : structure de la base, gestion des schémas
- règles de gestion : calcul des consentements, règles de mises à jour, dé-duplication

Webservices Application Java EE (war) exposant des webservices

- correspondance *modèle objet interne* $\leftrightarrow$ *contrat WSDL*
- Interface WEB : suivi des pipelines

Pipelines Intégration de données par fichiers CSV (jar exécutable)

- Démarrage/Arrêt d'un *démon*
- Détection de l'arrivée de fichiers
- EAI et lecture de fichier
- Parallélisation des traitements

Batch Exports, traitements de types : fusion, statistiques, ...

Organisation des modules

En utilisant *Maven*

Maven

Apache Maven est un logiciel de gestion de projet. Basé sur le concept de *Project Object Model* (POM), il gère le processus de compilation, rapports, ...

Organisation des modules
En utilisant *Maven*

Maven

Apache Maven est un logiciel de gestion de projet. Basé sur le concept de *Project Object Model* (POM), il gère le processus de compilation, rapports, ...

Le Référentiel Coordonnées utilise *Maven* pour :

- la gestion des dépendances
- le packaging : compilation, tests automatique, archives zip et ear
- le déploiement des pipelines et batch (FTP)
- configuration du poste de travail : *Eclipse*
- tests en local des Webservices et IHM

Projets et sous-projets *Maven*

Le référentiel coordonnées est répartie sur 6 projets *Maven* :

- **addressrepository** : *version, dépendances, modules (pom)*

Projets et sous-projets *Maven*

Le référentiel coordonnées est répartie sur 6 projets *Maven* :

- **addressrepository** : *version, dépendances, modules (pom)*
  - **address-core** : *coeur de l'application* (jar)
    package : `net.yvesrocher.services.address.core`

Projets et sous-projets *Maven*

Le référentiel coordonnées est répartie sur 6 projets *Maven* :

- **addressrepository** : *version, dépendances, modules (pom)*
  - **address-core** : *coeur de l'application* (jar)
    package : `net.yvesrocher.services.address.core`
  - **address-pipelines** : *démon (jar exécutable)*
    package : `net.yvesrocher.services.address.pipelines`

Projets et sous-projets *Maven*

Le référentiel coordonnées est répartie sur 6 projets *Maven* :

- **addressrepository** : *version, dépendances, modules (pom)*
  - **address-core** : *coeur de l'application* (jar)
    package : `net.yvesrocher.services.address.core`
  - **address-pipelines** : *démon (jar exécutable)*
    package : `net.yvesrocher.services.address.pipelines`
  - **address-webservices** : *War du webservices*
    package : `net.yvesrocher.services.address.webservices`

Projets et sous-projets *Maven*

Le référentiel coordonnées est répartie sur 6 projets *Maven* :

- **addressrepository** : *version, dépendances, modules (pom)*
  - **address-core** : *coeur de l'application* (jar)
    package : `net.yvesrocher.services.address.core`
  - **address-pipelines** : *démon (jar exécutable)*
    package : `net.yvesrocher.services.address.pipelines`
  - **address-webservices** : *War du webservices*
    package : `net.yvesrocher.services.address.webservices`
  - **address-ear** : *Package le war en un EAR*

Projets et sous-projets *Maven*

Le référentiel coordonnées est répartie sur 6 projets *Maven* :

- **addressrepository** : *version, dépendances, modules (pom)*
  - **address-core** : *coeur de l'application* (jar)
    package : `net.yvesrocher.services.address.core`
  - **address-pipelines** : *démon (jar exécutable)*
    package : `net.yvesrocher.services.address.pipelines`
  - **address-webservices** : *War du webservices*
    package : `net.yvesrocher.services.address.webservices`
  - **address-ear** : *Package le war en un EAR*
  - **address-batch** : *Batchs utilisant le coeur V3 (jar exécutable)*
    package : `net.yvesrocher.services.address.batch`

Compilation et déploiement

Compilation

Maven exécute les tests unitaires et crée les binaires s'il n'y a pas d'erreur.
Les binaires sont présents dans le répertoire `target` de chaque module.

Compilation et déploiement

Compilation

Maven exécute les tests unitaires et crée les binaires s'il n'y a pas d'erreur.
Les binaires sont présents dans le répertoire `target` de chaque module.

Commande Maven de compilation et déploiement sur FTP :

```
1  mvn clean install -DdeployPipelines
```

Sommaire

Configuration

Inversion de Contrôle, avec *Spring* !

Spring

Spring fournit l'inversion de contrôle : cycle de vie des *beans*, injection de dépendances et gestion de la configuration.

Configuration

Inversion de Contrôle, avec *Spring* !

Spring

Spring fournit l'inversion de contrôle : cycle de vie des *beans*, injection de dépendances et gestion de la configuration.

Les fichiers de configuration se trouvent dans `src/main/resources` :

- `spring` : fichiers de configuration *Spring* :

Configuration

Inversion de Contrôle, avec *Spring* !

Spring

Spring fournit l'inversion de contrôle : cycle de vie des *beans*, injection de dépendances et gestion de la configuration.

Les fichiers de configuration se trouvent dans `src/main/resources` :

- `spring` : fichiers de configuration *Spring* :
  - `addressrepository-core.xml` : inclue les fichiers nécessaires

Configuration

Inversion de Contrôle, avec *Spring* !

Spring

Spring fournit l'inversion de contrôle : cycle de vie des *beans*, injection de dépendances et gestion de la configuration.

Les fichiers de configuration se trouvent dans `src/main/resources` :

- `spring` : fichiers de configuration *Spring* :
  - `addressrepository-core.xml` : inclue les fichiers nécessaires
  - `addressrepository-businessservice.xml` : paramètre le coeur pour être intégré à un autre Webservice

Configuration

Inversion de Contrôle, avec *Spring* !

Spring

Spring fournit l'inversion de contrôle : cycle de vie des *beans*, injection de dépendances et gestion de la configuration.

Les fichiers de configuration se trouvent dans `src/main/resources` :

- `spring` : fichiers de configuration *Spring* :
  - `addressrepository-core.xml` : inclue les fichiers nécessaires
  - `addressrepository-businessservice.xml` : paramètre le coeur pour être intégré à un autre Webservice
- `config` : fichiers de paramétrage (*properties*)

Gestion des contextes du RC

Porté *Spring* : `context`

Scope `context`

En plus des scopes classiques `singleton` et `prototype`, le scope `context` défini un singleton d'un contexte.

Gestion des contextes du RC

Porté *Spring* : `context`

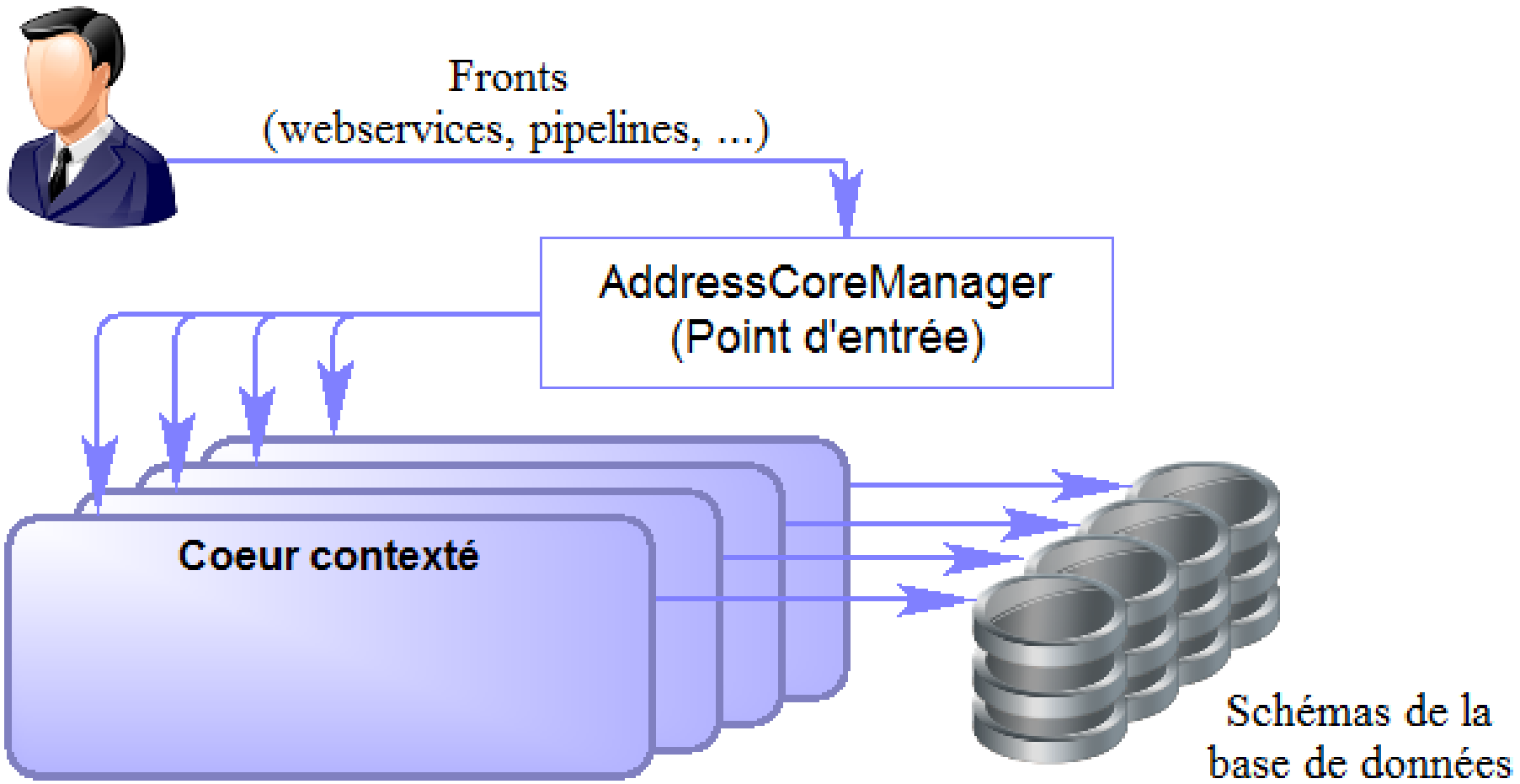
Scope `context`

En plus des scopes classiques `singleton` et `prototype`, le scope `context` défini un singleton d'un contexte.

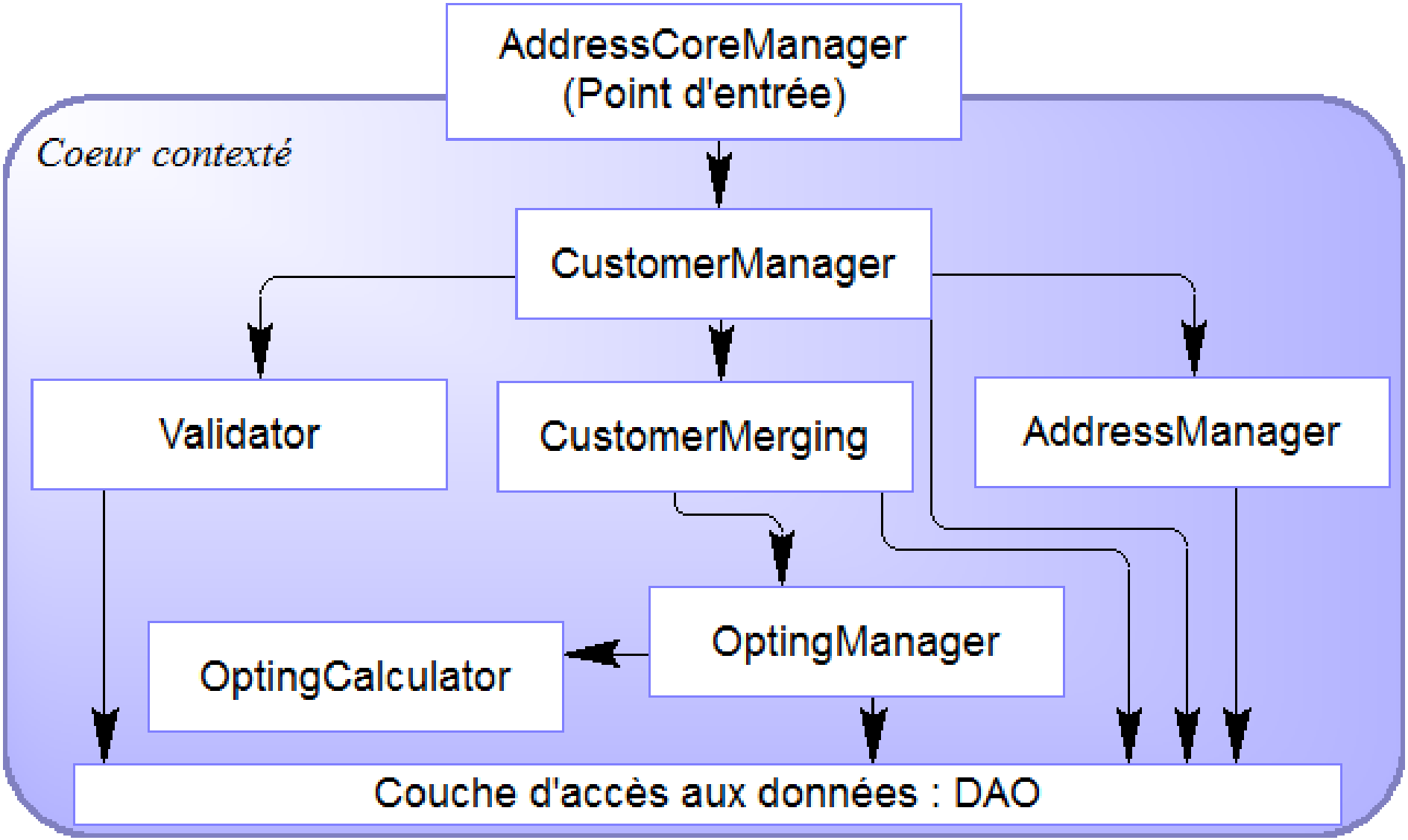
- Seule une instance de bean contexté est créée pour chaque contexte.

Distribution sur les contextes

Une instance du coeur est créée pour chaque contexte



Cartographie du coeur



Briques applicatives du RC

- **CustomerManager** Point d'entrée du coeur
  - Exécute la validation
  - Recherche les doublons
  - Distribue : nouveau client, mise à jour, prospect
- **Validator** Valide les données d'un client, de ses comptes et coordonnées.
- **AddressManager** Traitement des coordonnées prospectes.
- **CustomerMerging** Confronte les données présentes en base à la mise à jour
- **OptingManager** Règle générale sur les consentements (propagations, flag)
- **OptingCalculator** Calculs des consentements

Couche d'accès aux données

Object Relational Mapping par *Hibernate*

Hibernate

Le framework *Hibernate* est utilisé pour gérer la relation entre le modèle objet et la base de données.

Couche d'accès aux données

Object Relational Mapping par *Hibernate*

Hibernate

Le framework *Hibernate* est utilisé pour gérer la relation entre le modèle objet et la base de données.

Session Factory

La `SessionFactory` d'Hibernate est configuré par Spring dans le fichier `context-persistence.xml`.
Elle a pour scope le **context** : une session factory par schéma.

Configuration des sources de données

Les sources de données sont fournie à *Spring* par le `DatasourceProvider`.
Il se repose sur 2 implémentations de *IDatasourceFactory* :

- `FileDatasourceFactoryImpl` : fichier properties présent sur UNIX (pipelines, batch)
- `JndiDatasourceFactoryImpl` : datasources présentes dans un dictionnaire JNDI

Pattern : *Open Session In View*

Open Session In View

Le pattern *Open Session In View* consiste à ouvrir la session le plus tôt possible, et la fermée le plus tard possible. Ainsi, les relations lazy ne sont chargées qu'au dernier moment et seulement si nécessaire.

Pattern : *Open Session In View*

Open Session In View

Le pattern *Open Session In View* consiste à ouvrir la session le plus tôt possible, et la fermée le plus tard possible. Ainsi, les relations lazy ne sont chargées qu'au dernier moment et seulement si nécessaire.

Ouverture des sessions

L'annotation `@OpenSession` permet d'ouvrir la session pour la méthode annotée, ou toutes les méthodes de la classes annotée.

Pattern : *Open Session In View*

Open Session In View

Le pattern *Open Session In View* consiste à ouvrir la session le plus tôt possible, et la fermée le plus tard possible. Ainsi, les relations lazy ne sont chargées qu'au dernier moment et seulement si nécessaire.

Ouverture des sessions

L'annotation `@OpenSession` permet d'ouvrir la session pour la méthode annotée, ou toutes les méthodes de la classes annotée.

Transactions

En revanche, les transactions sont positionnées pour garantir la cohérence de la base de données. Elle sont configurée par l'annotation `@Transactional`

Tests unitaires dans le RC

Outils utilisés

JUNIT Framework de tests unitaires

Tests unitaires dans le RC

Outils utilisés

JUNIT	Framework de tests unitaires
FestAssert	Écriture des assertions dans les tests

Tests unitaires dans le RC

Outils utilisés

JUNIT	Framework de tests unitaires
FestAssert	Écriture des assertions dans les tests
Mockito	Isole les beans business afin de pouvoir les isoler avant de les tester

Tests unitaires dans le RC

Outils utilisés

JUNIT	Framework de tests unitaires
FestAssert	Écriture des assertions dans les tests
Mockito	Isole les beans business afin de pouvoir les isoler avant de les tester
HSQL	Base de données en mémoire vive pour tester les DAO

Tests unitaires dans le RC

Outils utilisés

JUNIT	Framework de tests unitaires
FestAssert	Écriture des assertions dans les tests
Mockito	Isole les beans business afin de pouvoir les isoler avant de les tester
HSQL	Base de données en mémoire vive pour tester les DAO
DBUnit	Charge/Décharge les données de tests dans la base HSQL

API de test
Éléments principaux de l'API

Éléments développés dans le RC pour faciliter l'écriture des tests

- classes parentes pré-configurée

API de test
Éléments principaux de l'API

Éléments développés dans le RC pour faciliter l'écriture des tests

- classes parentes pré-configurée
- sur-couche de *DBUnit*

API de test
Éléments principaux de l'API

Éléments développés dans le RC pour faciliter l'écriture des tests

- classes parentes pré-configurée
- sur-couche de *DBUnit*
- assertions spécifique sur les DTO (avec *FestAssert*)

API de test
Éléments principaux de l'API

Éléments développés dans le RC pour faciliter l'écriture des tests

- classes parentes pré-configurée
- sur-couche de *DBUnit*
- assertions spécifique sur les DTO (avec *FestAssert*)
- méthodes de génération de données de tests

Classes parentes de tests unitaires

Classes principales à surcharger :

- **SimpleJunitTest** : Super-classe principale des tests unitaire dans le RC. Initialise le contexte Spring

Classes parentes de tests unitaires

Classes principales à surcharger :

- **SimpleJunitTest** : Super-classe principale des tests unitaire dans le RC. Initialise le contexte Spring
- **DBUnitTest** : Initialise la BDD HSQL et configure Hibernate pour l'utiliser

Classes parentes de tests unitaires

Classes principales à surcharger :

- **SimpleJunitTest** : Super-classe principale des tests unitaire dans le RC. Initialise le contexte Spring
- **DBUnitTest** : Initialise la BDD HSQL et configure Hibernate pour l'utiliser
- **ContextedJunitTest** : permet l'injection de *beans contextés* dans la classe de test

Sur-couche de DBUnit

Annotation `@DatabaseScripts`

DBunit

DBunit est un outils chargeant, vidant ou exportant le contenu d'une BDD, à partir ou vers des fichiers XML.

Sur-couche de DBUnit

Annotation `@DatabaseScripts`

DBunit

DBunit est un outils chargeant, vidant ou exportant le contenu d'une BDD, à partir ou vers des fichiers XML.

L'annotation `@DatabaseScripts`, développée pour le RC :

- positionnée sur les méthodes de tests, ou au niveau classe
- définit les fichiers à utiliser pour charger la BDD
- les `locations` sont héritées de la classe, et des classes parentes
- pas d'héritage si `inheritsLocations` est `true`

Sommaire

Module webservices RC

Module Webservices du RC

Ce module est déployé sur un serveur d'applications (type *Websphère*).
Il expose des webservices et fait le lien avec le coeur RC

Module webservices RC

Module Webservices du RC

Ce module est déployé sur un serveur d'applications (type *Websphère*).
Il expose des webservices et fait le lien avec le coeur RC

- Pas de logique métier en dehors de la règle A : seul le compte correspondant au réseau de l'appel est pris en compte.

Module webservices RC

Module Webservices du RC

Ce module est déployé sur un serveur d'applications (type *Websphère*).
Il expose des webservices et fait le lien avec le coeur RC

- Pas de logique métier en dehors de la règle A : seul le compte correspondant au réseau de l'appel est pris en compte.
- Package des classes non générées : `net.yvesrocher.services.address.webservices`

Module webservices RC

Module Webservices du RC

Ce module est déployé sur un serveur d'applications (type *Websphère*).
Il expose des webservices et fait le lien avec le coeur RC

- Pas de logique métier en dehors de la règle A : seul le compte correspondant au réseau de l'appel est pris en compte.
- Package des classes non générées : `net.yvesrocher.services.address.webservices`
- la majorité du module sont les *mapper* : DTO $\leftrightarrow$ WSDL

IHM de monitoring

Avec *Spring MVC*

Spring MVC

Dispatcher : en fonction de l'URL d'appel, trouve la méthode la plus appropriée. La méthode réalise l'action et indique la vue à utiliser.

IHM de monitoring

Avec *Spring MVC*

Spring MVC

Dispatcher : en fonction de l'URL d'appel, trouve la méthode la plus appropriée. La méthode réalise l'action et indique la vue à utiliser.

- package des *contrôleurs* : `net.yvesrocher.services.address.webservices.servlet.controller`

Sommaire

Module des pipelines

Pipelines

Application *standalone* de type démon. Elle détecte l'arrivée de fichiers, détermine leur type et les traite.

Module des pipelines

Pipelines

Application *standalone* de type démon. Elle détecte l'arrivée de fichiers, détermine leur type et les traite.

Parties des pipelines :

- **démon** : démarrage et arrêt de l'application

Module des pipelines

Pipelines

Application *standalone* de type démon. Elle détecte l'arrivée de fichiers, détermine leur type et les traite.

Parties des pipelines :

- **démon** : démarrage et arrêt de l'application
- **configuration** : quels sont les répertoires à scanner, quels traitements

Module des pipelines

Pipelines

Application *standalone* de type démon. Elle détecte l'arrivée de fichiers, détermine leur type et les traite.

Parties des pipelines :

- **démon** : démarrage et arrêt de l'application
- **configuration** : quels sont les répertoires à scanner, quels traitements
- **gestion des traitements** : limite du nombre de traitements simultanés, global et par contexte

Démon pipelines

Démon

Déployé dans une archive ZIP, contient un jar, ses dépendances et son paramétrage : fichiers de configuration, fichiers *scriptella*.
La partie démon est implémenté dans la classe `main`.

Démon pipelines

Démon

Déployé dans une archive ZIP, contient un jar, ses dépendances et son paramétrage : fichiers de configuration, fichiers *scriptella*.
La partie démon est implémenté dans la classe `main`.

- Écoute sur un port, configurable : `Socket TCP`
- Transmission de chaine de caractères à travers la `socket` pour arrêter le démon, connaitre sa version ou son état.

Démon pipelines

Démon

Déployé dans une archive ZIP, contient un jar, ses dépendances et son paramétrage : fichiers de configuration, fichiers *scriptella*.
La partie démon est implémenté dans la classe `main`.

- Écoute sur un port, configurable : `Socket TCP`
- Transmission de chaine de caractères à travers la `socket` pour arrêter le démon, connaitre sa version ou son état.

Évolutions possibles

Normaliser cette communication par RPC Java (Remote Procedure Call).

Configuration des pipelines

Déclaration des flux d'entrée standard

```
1  <pipelinesConfiguration>
2    <dataInputs>
3      <csvFileFlux id="classicCsv">
4        <encoding>UTF-8</encoding>
5        <separator>;</separator>
6        <quote>''</quote>
7      </csvFileFlux>
8    </dataInputs>
9
10   <pipelines>
11     <!-- Pipelines standardises -->
12     <pipeline enabled="true">
13       <context>
14         <brandId>YR</brandId>
15         <countryId>RU</countryId>
16       </context>
17       <name>YR/RU/standard/update</name>
18       <location>YRRU/standard</location>
19       <discriminator>[A-Za-z]{4}_(SYN|REP|PRO)_.*</discriminator>
20       <standardMethod />
21     </pipeline>
22
23   </pipelines>
24 </pipelinesConfiguration>
```

Configuration des pipelines

Déclaration d'un flux non standard

```
1   ...
2       <pipeline enabled="true">
3         <name>YR/RU/_reprise/isam</name>
4         <location>YRRU/scriptella</location>
5         <discriminator>[A-Za-z]{4}_REP_ISAM_.*</discriminator>
6         <context>
7           <brandId>YR</brandId>
8           <countryId>RU</countryId>
9           <language>ru</language>
10        </context>
11        <scriptellaMethod csvFile="classicCsv">
12          <mapperScript>isam/reprise/isam_yrru.mapper.js</mapperScript>
13          <nature>REPRISE</nature>
14          <pipelineName>20_diamant</pipelineName>
15          <properties>
16            <processType>REPRISE</processType>
17            <applicationCode>ISAM</applicationCode>
18            <conCode>VPM</conCode>
19            <pchCode>VPM</pchCode>
20            <copScopeDefault>VPM</copScopeDefault>
21            <bevCode>REP</bevCode>
22            <styCode>SI</styCode>
23            <copRequester>ISAM</copRequester>
24          </properties>
25        </scriptellaMethod>
26      </pipeline>
27  ...
```

Gestion des traitements