Spring

SoC – IoC – DI

Thomas Duchatelle (duchatelle.thomas@gmail.com)

Capgemini, pour Yves Rocher

December 13, 2012

Sommaire

Séparation des préoccupations

SoC : Separation of Concerns

Pris isolément, chaque problème est plus facile à traiter.

Séparation des préoccupations

SoC : Separation of Concerns

Pris isolément, chaque problème est plus facile à traiter.

Découpage de l'application pour isoler les problématiques :

- persistance
- services métier
- présentation (IHM Web)
- appel webservice
- ...

Séparation des préoccupations

SoC : Separation of Concerns

Pris isolément, chaque problème est plus facile à traiter.

Découpage de l'application pour isoler les problématiques :

- persistance
- services métier
- présentation (IHM Web)
- appel webservice
- ...

Beans

Pour chaque nature de problématique : conception de "composants spécialisés", de *briques applicatives*.

Cas concret

Embauche d'un nouvel employé

Nouvelle embauche

Intégration dans le SI d'un nouvel employé : création de son matricule, email et insertion dans le système des ressources humaines.

Cas concret
Embauche d'un nouvel employé

Nouvelle embauche

Intégration dans le SI d'un nouvel employé : création de son matricule, email et insertion dans le système des ressources humaines.

Processus métier pour l'embauche d'un nouveau client :

1. un utilisateur autorisé renseigne le nom, prénom et intitulé du poste du nouvel employé

Cas concret
Embauche d'un nouvel employé

Nouvelle embauche

Intégration dans le SI d'un nouvel employé : création de son matricule, email et insertion dans le système des ressources humaines.

Processus métier pour l'embauche d'un nouveau client :

1. un utilisateur autorisé renseigne le nom, prénom et intitulé du poste du nouvel employé
2. le système génère le matricule de l'employé : identifiant unique

Cas concret
Embauche d'un nouvel employé

Nouvelle embauche

Intégration dans le SI d'un nouvel employé : création de son matricule, email et insertion dans le système des ressources humaines.

Processus métier pour l'embauche d'un nouveau client :

1. un utilisateur autorisé renseigne le nom, prénom et intitulé du poste du nouvel employé
2. le système génère le matricule de l'employé : identifiant unique
3. le système génère l'email de l'employé : à partir de son nom et prénom, unique aussi

Cas concret
Embauche d'un nouvel employé

Nouvelle embauche

Intégration dans le SI d'un nouvel employé : création de son matricule, email et insertion dans le système des ressources humaines.

Processus métier pour l'embauche d'un nouveau client :

1. un utilisateur autorisé renseigne le nom, prénom et intitulé du poste du nouvel employé
2. le système génère le matricule de l'employé : identifiant unique
3. le système génère l'email de l'employé : à partir de son nom et prénom, unique aussi
4. toutes ces données sont conservées dans le Référentiel Employés

Cas concret
Embauche d'un nouvel employé

Nouvelle embauche

Intégration dans le SI d'un nouvel employé : création de son matricule, email et insertion dans le système des ressources humaines.

Processus métier pour l'embauche d'un nouveau client :

1. un utilisateur autorisé renseigne le nom, prénom et intitulé du poste du nouvel employé
2. le système génère le matricule de l'employé : identifiant unique
3. le système génère l'email de l'employé : à partir de son nom et prénom, unique aussi
4. toutes ces données sont conservées dans le Référentiel Employés
5. le système informe l'application des Ressources Humaines de la création de nouvel employé

Méga script !

Un script PHP suffit

- Un tel processus pourrait être écrit en un seul script PHP...

Méga script !

Un script PHP suffit

- Un tel processus pourrait être écrit en un seul script PHP...
- Mais :

Méga script !
Un script PHP suffit

- Un tel processus pourrait être écrit en un seul script PHP...
- Mais :
  - difficulté d'écrire le script

Méga script !
Un script PHP suffit

- Un tel processus pourrait être écrit en un seul script PHP...
- Mais :
  - difficulté d'écrire le script
  - longueur et lisibilité du script ?

Méga script !
Un script PHP suffit

- Un tel processus pourrait être écrit en un seul script PHP...
- Mais :
  - difficulté d'écrire le script
  - longueur et lisibilité du script ?
  - tests de tous les cas

Séparation des préoccupations

Division de la problématique en petites sous problématique

Proposition de découpage :

Séparation des préoccupations

Division de la problématique en petites sous problématique

Proposition de découpage :

- *IHM* (couche de présentation) : propose une interface intuitive à l'utilisateur afin de récolter les données

Séparation des préoccupations

Division de la problématique en petites sous problématique

Proposition de découpage :

- *IHM* (couche de présentation) : propose une interface intuitive à l'utilisateur afin de récolter les données
- *Gestionnaire des Employés* (objet métier) : détient les règles et le processus de création d'un employé

Séparation des préoccupations

Division de la problématique en petites sous problématique

Proposition de découpage :

- *IHM* (couche de présentation) : propose une interface intuitive à l'utilisateur afin de récolter les données
- *Gestionnaire des Employés* (objet métier) : détient les règles et le processus de création d'un employé
- *Générateur de matricules* (objet métier) : détient les règles de génération d'un identifiant unique

Séparation des préoccupations

Division de la problématique en petites sous problématique

Proposition de découpage :

- *IHM* (couche de présentation) : propose une interface intuitive à l'utilisateur afin de récolter les données
- *Gestionnaire des Employés* (objet métier) : détient les règles et le processus de création d'un employé
- *Générateur de matricules* (objet métier) : détient les règles de génération d'un identifiant unique
- *Générateur d'email* (objet métier) : génère un email à partir du nom et prénom.

Séparation des préoccupations

Division de la problématique en petites sous problématique

Proposition de découpage :

- *IHM* (couche de présentation) : propose une interface intuitive à l'utilisateur afin de récolter les données
- *Gestionnaire des Employés* (objet métier) : détient les règles et le processus de création d'un employé
- *Générateur de matricules* (objet métier) : détient les règles de génération d'un identifiant unique
- *Générateur d'email* (objet métier) : génère un email à partir du nom et prénom.
- *DAO Employés* (objet d'accès aux données) : persiste l'employé et détermine si un email est disponible

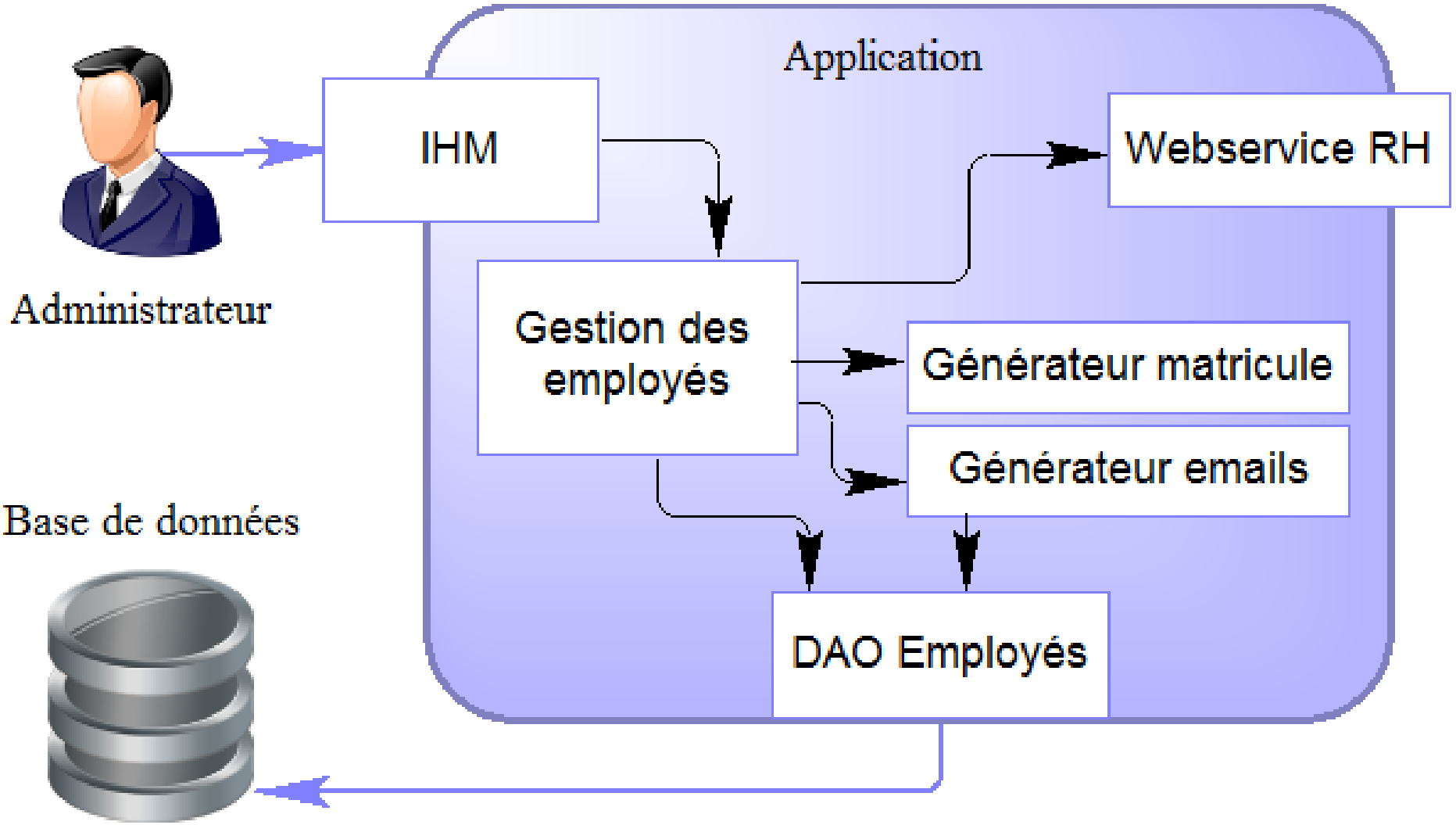
Séparation des préoccupations

Division de la problématique en petites sous problématique

Proposition de découpage :

- *IHM* (couche de présentation) : propose une interface intuitive à l'utilisateur afin de récolter les données
- *Gestionnaire des Employés* (objet métier) : détient les règles et le processus de création d'un employé
- *Générateur de matricules* (objet métier) : détient les règles de génération d'un identifiant unique
- *Générateur d'email* (objet métier) : génère un email à partir du nom et prénom.
- *DAO Employés* (objet d'accès aux données) : persiste l'employé et détermine si un email est disponible
- *Connecteur WS RH* (objet métier) : gère la connexion avec le webservice de l'application des ressources humaines.

Architecture de l'exemple



Sommaire

Comment gérer autant debriques applicatives ?

Approche naïve

Instanciation des dépendances du bean :

```
1   public class EmployeeManager {
2
3     private EmployeeNumberGenerator employeeNumberGenerator;
4
5     private EmailGenerator emailGenerator;
6
7     private EmployeeDAO employeeDAO;
8
9     public EmployeeManager(Datasource datasource) {
10      // Generateur de matricule n'a pas de dependance
11      employeeNumberGenerator = new EmployeeNumberGenerator();
12
13      employeeDAO = new EmployeeDAO();
14      employeeDAO.setDatasource(datasource);// confiruration les datasources !!
15
16      emailGenerator = new EmailGenerator();
17      emailGenerator.setEmployeeDAO(employeeDAO); // ajout des dependance
18    }
19  }
```

Comment gérer autant de *briques applicatives* ?

Approche naïve

Pas de singleton possible

Une nouvelle instance d'un bean est créée à chaque fois.

Comment gérer autant de *briques applicatives* ?

Approche naïve

Pas de singleton possible

Une nouvelle instance d'un bean est créée à chaque fois.

Couplage fort

La dépendance doit connaitre l'implémentation de ses dépendances, ainsi que les dépendances des dépendances (et ainsi de suite) !

Comment gérer autant de briques applicatives ?

Approche par *Factory*

Utilisation d'une fabrique d'objets

```
1  public class EmployeeManager {
2
3    private IEmployeeNumberGenerator employeeNumberGenerator = Factory.
         getEmployeeNumberGenerator();
4
5    private IEmailGenerator emailGenerator = Factory.getEmailGenerator();
6  }
```

Comment gérer autant de *briques applicatives* ?

Approche par *Factory*

Mutualisation de la création d'objets

- Les méthodes sont réutilisables.

Comment gérer autant de *briques applicatives* ?

Approche par *Factory*

Mutualisation de la création d'objets

- Les méthodes sont réutilisables.
- Les implémentations des beans ne sont connues que de la fabrique : utilisation *interfaces*.

Comment gérer autant de *briques applicatives* ?

Approche par *Factory*

Mutualisation de la création d'objets

- Les méthodes sont réutilisables.
- Les implémentations des beans ne sont connues que de la fabrique : utilisation *interfaces*.

Comment gérer autant de *briques applicatives* ?

Approche par *Factory*

Mutualisation de la création d'objets

- Les méthodes sont réutilisables.
- Les implémentations des beans ne sont connues que de la fabrique : utilisation *interfaces*.

Couplage toujours important

Dépendance vis à vis de la fabrique.

Comment gérer autant de *briques applicatives* ?

Approche par *Factory*

Mutualisation de la création d'objets

- Les méthodes sont réutilisables.
- Les implémentations des beans ne sont connues que de la fabrique : utilisation *interfaces*.

Couplage toujours important

Dépendance vis à vis de la fabrique.

Configuration

Comment faire passer la source de données ?

- méthode statiquement : peu intuitif et source d'erreurs

Comment gérer autant de *briques applicatives* ?

Approche par *Factory*

Mutualisation de la création d'objets

- Les méthodes sont réutilisables.
- Les implémentations des beans ne sont connues que de la fabrique : utilisation *interfaces*.

Couplage toujours important

Dépendance vis à vis de la fabrique.

Configuration

Comment faire passer la source de données ?

- méthode statiquement : peu intuitif et source d'erreurs
- argument de la méthode : couplage fort

Comment gérer autant de *briques applicatives* ?

Approche idéale : injection de dépendances

Objectifs de l'approche idéale :

- L'`EmployeeManager` ne crée pas ses dépendances
- Il ne récupère pas ses dépendances d'un tiers

Comment gérer autant de *briques applicatives* ?

Approche idéale : injection de dépendances

Objectifs de l'approche idéale :

- L'`EmployeeManager` ne crée pas ses dépendances
- Il ne récupère pas ses dépendances d'un tiers

Comment ?

- Il va être créé par un composant externe (équivalent d'une fabrique)
- Ce composant externe va lui injecter les dépendances dont il a besoin

Comment gérer autant de *briques applicatives* ?

Approche idéale : injection de dépendances

Injection de dépendances

Le concept d'injection de dépendances est d'instancier un bean, et de lui injecter, par constructeur ou par setter ses dépendances.

Gestion de la configuration

Approche naïve

Accès direct

Chaque bean est responsable de sa configuration : il utilise sa propre méthode et y accède lui même.

Gestion de la configuration

Approche naïve

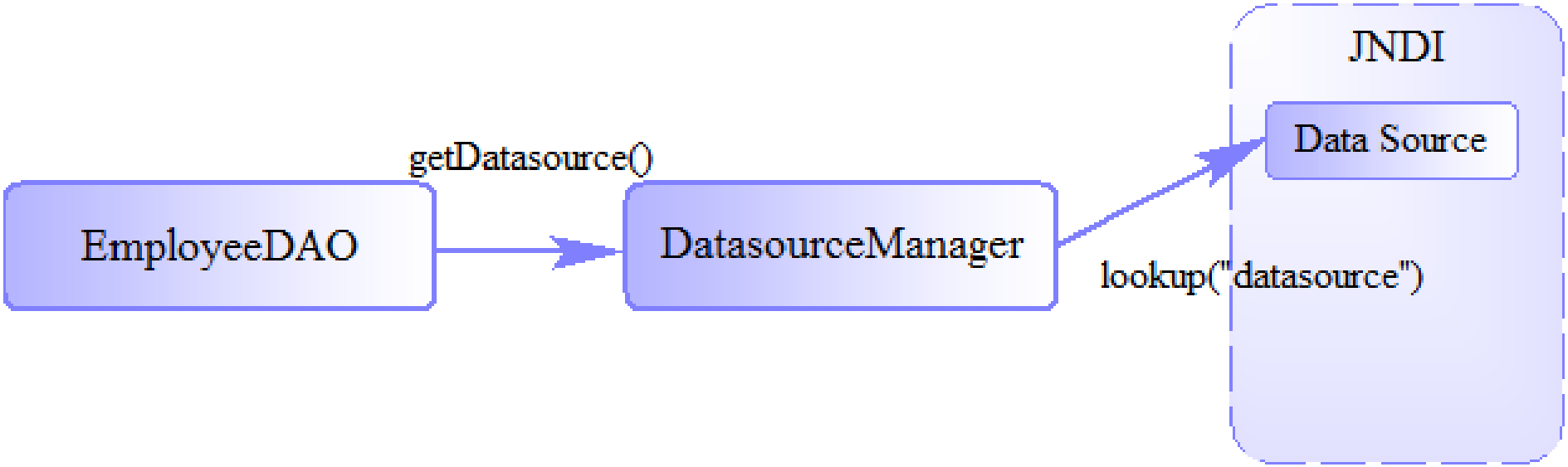
Accès direct

Chaque bean est responsable de sa configuration : il utilise sa propre méthode et y accède lui même.

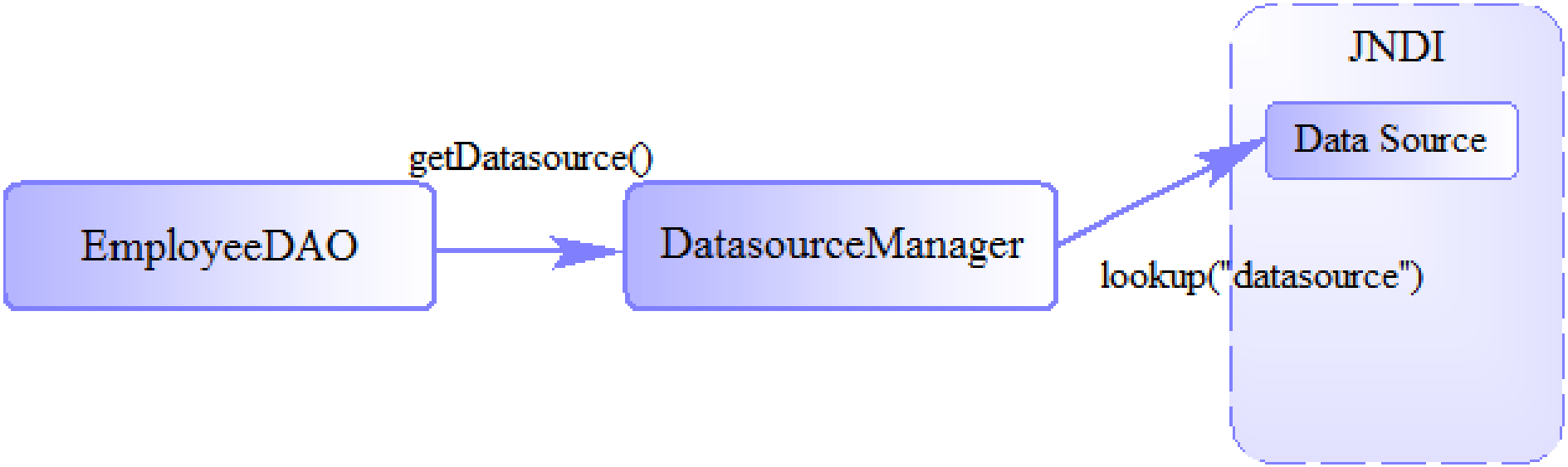
Exemple

La couche d'accès aux données utilise un "`ConnectionManager`" qui récupère les sources de données par *JNDI*.

Exemple de l'approche directe



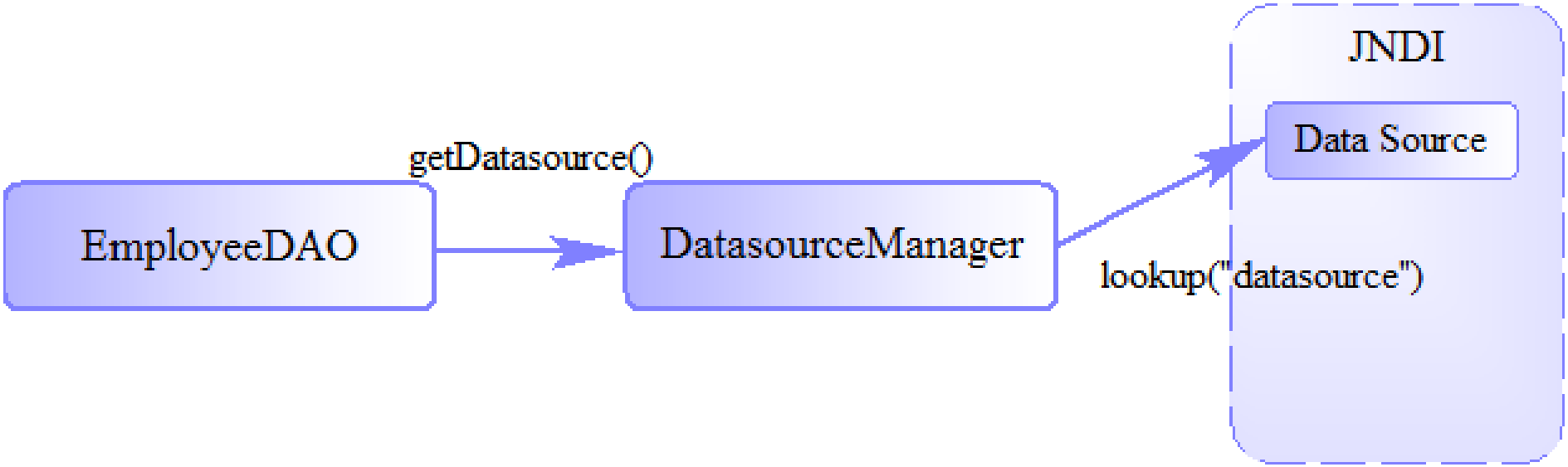
Exemple de l'approche directe



Différences sur les environnements :

- Sur un serveur d'application **OK**.

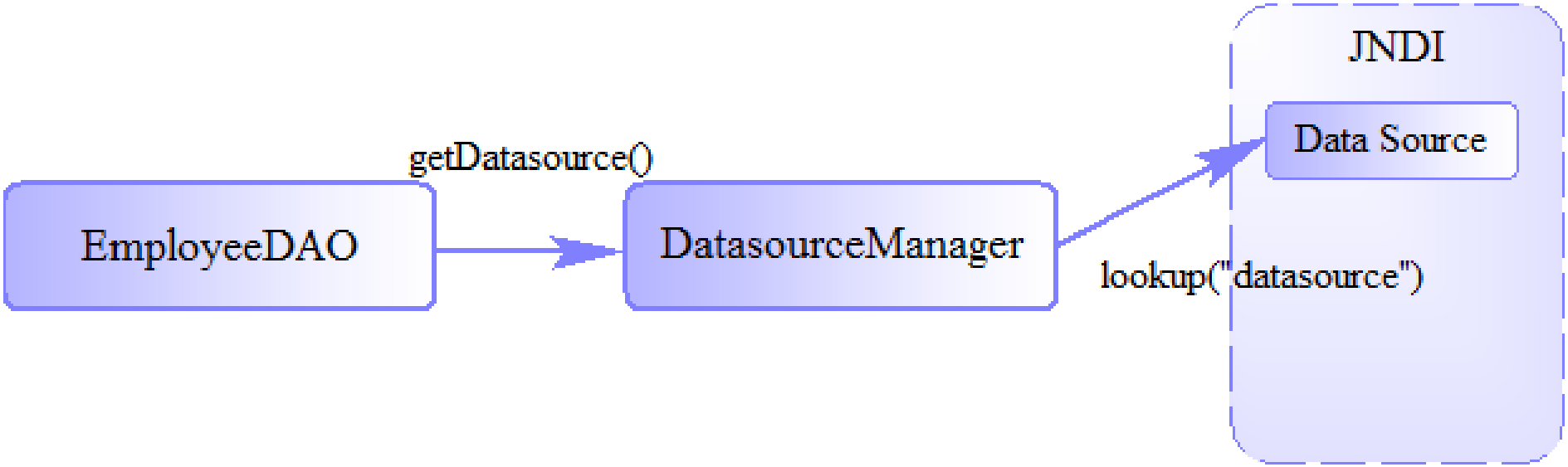
Exemple de l'approche directe



Différences sur les environnements :

- Sur un serveur d'application **OK**.
- Serveur local : nécessite de paramétrer les sources de données

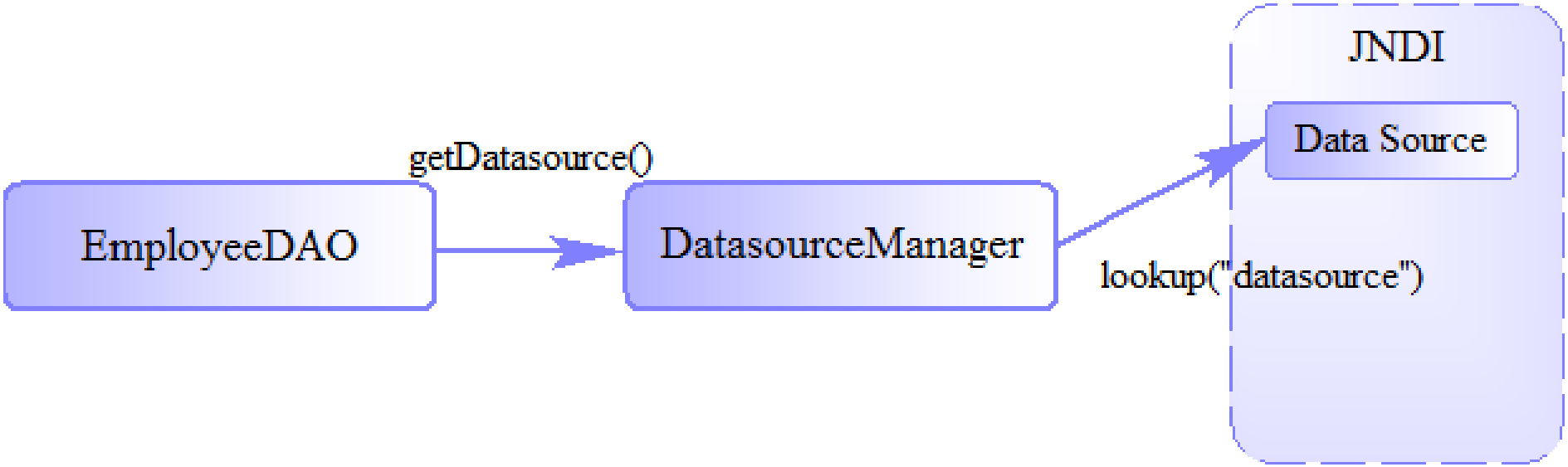
Exemple de l'approche directe



Différences sur les environnements :

- Sur un serveur d'application **OK**.
- Serveur local : nécessite de paramétrer les sources de données
- Tests local (unitaires) : nécessite de forcer la configuration. Lourdeur d'écriture des tests.

Exemple de l'approche directe



Différences sur les environnements :

- Sur un serveur d'application **OK**.
- Serveur local : nécessite de paramétrer les sources de données
- Tests local (unitaires) : nécessite de forcer la configuration. Lourdeur d'écriture des tests.
- En mode standalone (batch) : création d'un "faux" contexte JNDI renseigné à partir d'un autre système de configuration !

Conclusion de cette première approche

Cohérence

Aucune gestion globale : cohérence des méthodes entre les beans système. Possible duplications de code et impossibilité de partage d'une même configuration.

Conclusion de cette première approche

Cohérence

Aucune gestion globale : cohérence des méthodes entre les beans système. Possible duplications de code et impossibilité de partage d'une même configuration.

Couplage fort

Aucune flexibilité propre à l'environnement d'exécution n'est permise.

Gestion de la configuration

Approche idéale

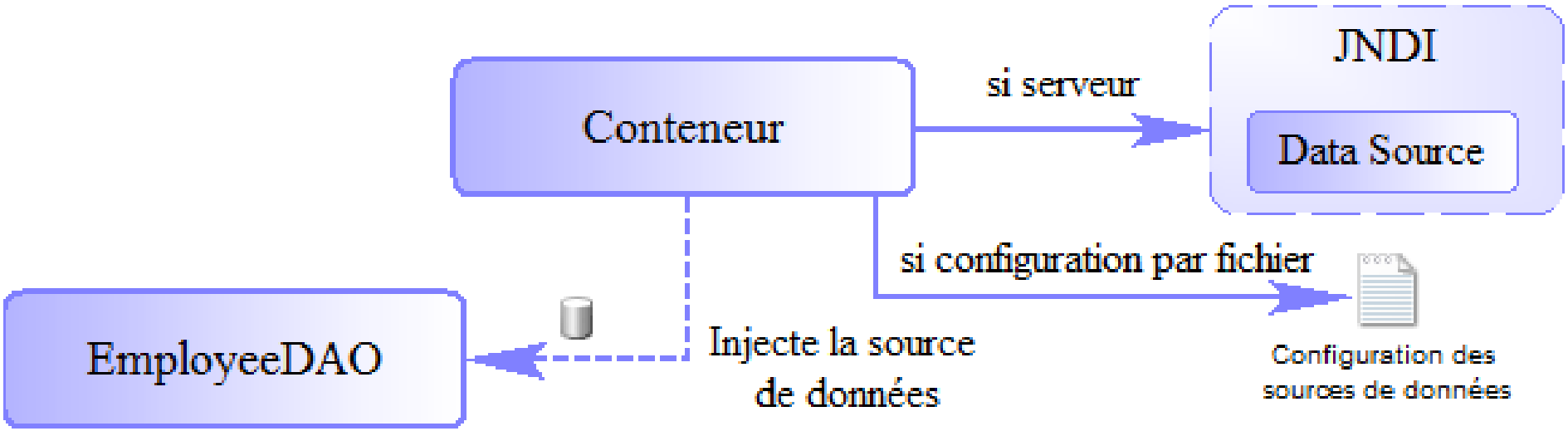
Injection de la configuration

La configuration est paramétrée de façon globale (properties, jndi, ...) et est distribuée à tous les beans qui en ont besoin.

Gestion de la configuration
Approche idéale

Injection de la configuration

La configuration est paramétrée de façon globale (properties, jndi, ...) et est distribuée à tous les beans qui en ont besoin.



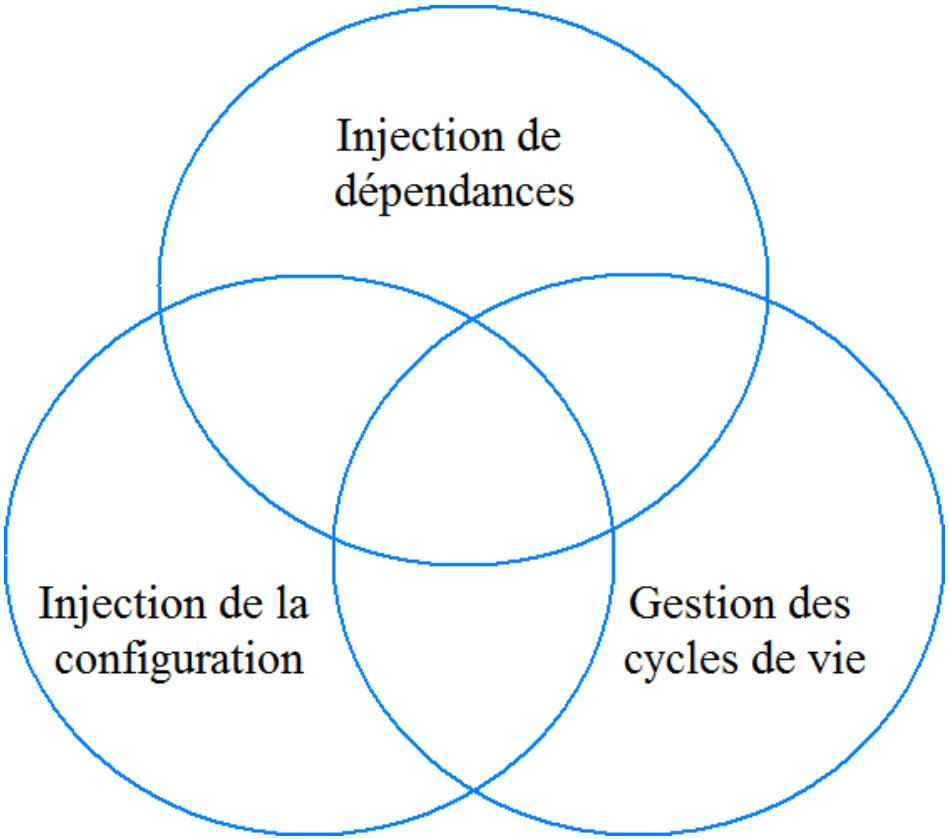
Cycle de vie

Cycle de vie

L'instanciation et la destruction des objets sont gérés par le conteneur. Il gère lui même les singletons.

Inversion de Contrôle

Association de 3 grands patterns



Les 3 grands patterns :

- **Injection de dépendances**
- **Injection de la configuration**
- **Gestion des cycles de vie**

Sommaire

Spring = Conteneur léger !

Définitions ...

Conteneur

Infrastructure prenant en charge la création d'objets et la mise en relation d'objets via des fichiers de configuration.

Fin

Merci, des questions ?