Architecture à 5 couches :
du modèle à l'implémentation JAVA

Introduction

Dans le numéro precedent, j'avais presenté un modele d'architecture a 5 couches et detaille ses avantages en terme d'evolutivité, de maintenabilite et de reutilisabilite. nous etions restes a un niveau architecture : objets, diagrammes, couches…

Je vous propose aujourd'hui d'explorer les enjeux de l'implémentation, de vous livrer des " best practices " et de vous decrire un framework java d'implementation. j'illustrerai ces concepts à l'aide d'un cas concret, et detaillerai le développement depuis la phase d'analyse jusqu'à la phase d'implementation.

Cet article s'adresse à des developpeurs java experimentes en architecture objet.

Rappel sur l'architecture à 5 couches :

Les architectures à couches proposent de regrouper les composants proposant le même type de comportement. Ces composants sont placés dans un même domaine, ou encore couche.

L'architecture à 5 couches que nous détaillons propose d'isoler :

- L'interface graphique dans la couche Client
- Le fonctionnel dans la couche Application
- Le métier dans la couche Entreprise
- La persistance Objet / Relationnel dans la couche Mapping
- Les données dans la couche Physique

Cette décomposition en couches permet d'élever progressivement le niveau d'abstraction depuis le format de stockage des données jusqu'à l'interface utilisateur.

En pratique, chaque couche est composée de domaines , qui regroupent des objets. En JAVA, les couches sont composées de package, lesquels regroupent des classes.

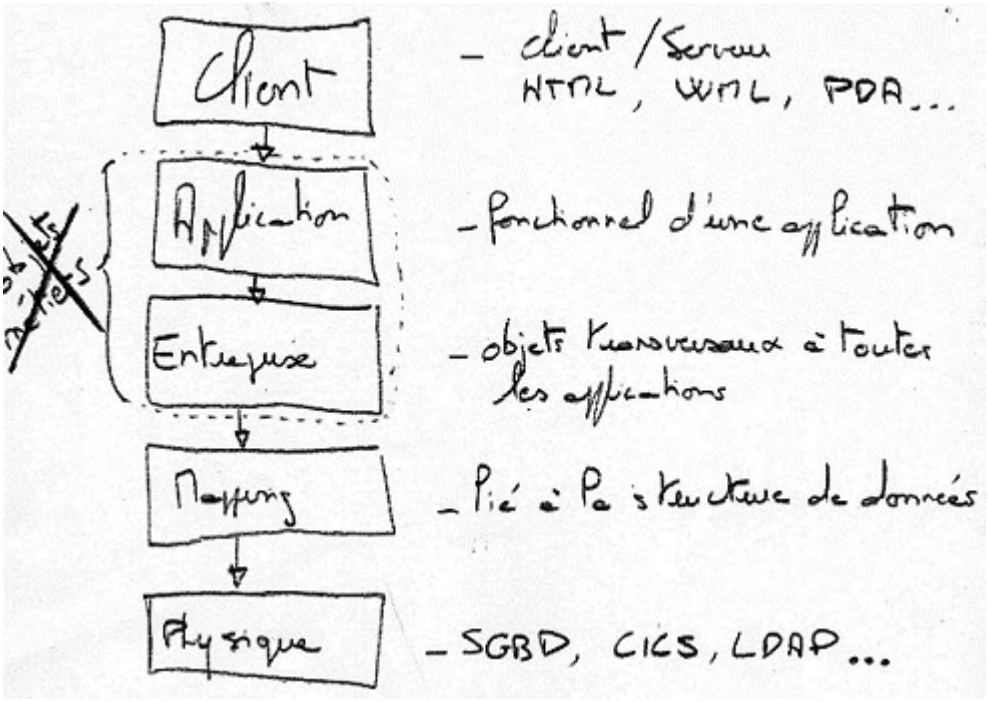

Illustration : Décomposition en couches

Analyse

Nous illustrerons l'implémentation de chaque couche par un exemple concret : une application WEB qui présente des news.

L'application présente dans des pages HTML des news, organisées en catégories, publiées par des auteurs, et lues par un utilisateur authentifié à travers un écran de login.
Un utilisateur peut effectuer des recherches de news.
Un auteur peut publier des news.

Illustration : Snapshot de http://www.application-servers.com/

Le premier travail consiste à représenter le dialogue Utilisateur avec le système. Ce dialogue est illustré par un cas d'utilisation, décrit par l'analyste. Dans le cas de notre portail de news, le scénario principal consiste en le login utilisateur, la lecture de news et la publication de news. Ce scénario peut être présenté dans un ou plusieurs diagrammes de séquence. A des fins de simplification, nous représentons l'ensemble du dialogue dans un seul diagramme de séquence.

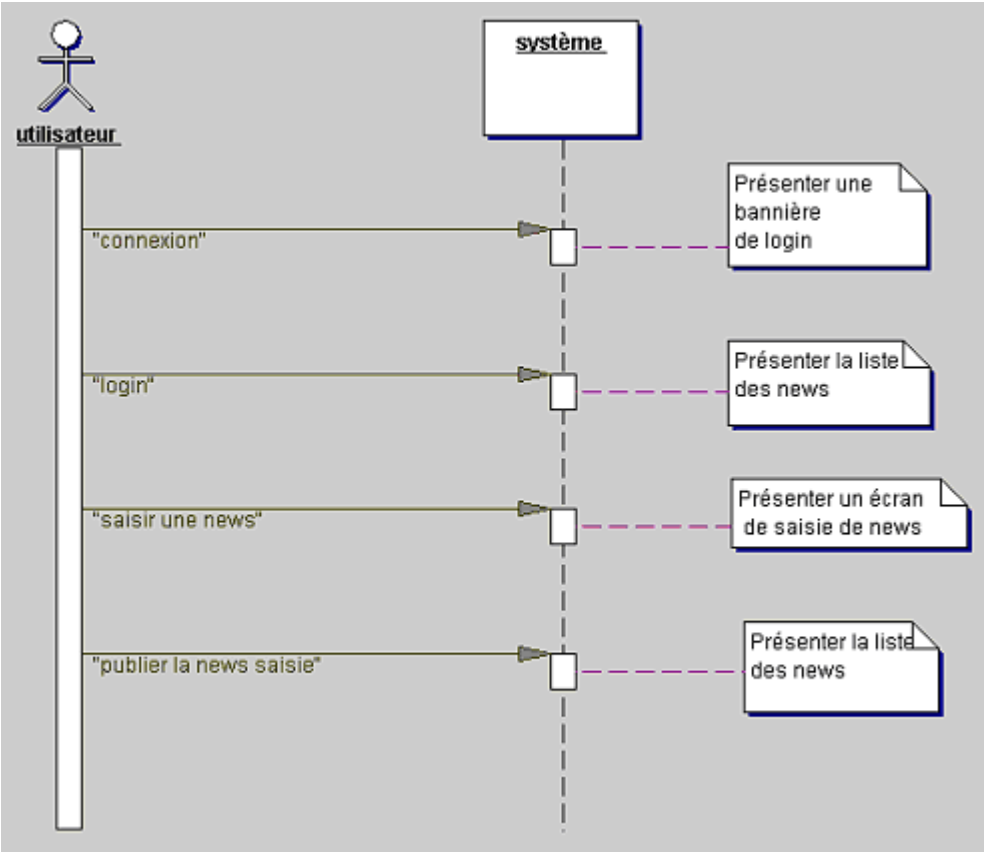

Illustration : diagramme de séquence du dialogue Utilisateur / Système

En formalisant le dialogue Utilisateur / Système, ce diagramme vient compléter les spécifications et permet de faire le lien entre les équipes de développement et les utilisateurs. Il est utile de s'en servir pour cadrer la réalisation, dans ce cas, il prendra la forme d'un document contractuel.

Le dialogue entre l'utilisateur et le système est présenté sous la forme de phrase entre guillemets, car dans le diagramme d'analyse, il ne s'agit pas de représenter le comportement d'implémentation du système à l'aide de méthodes, mais sa représentation du point de vue utilisateur.

Remarque :
La conception détaille le diagramme ci-dessus, en faisant intervenir les classes d'implémentation de haut niveau, c'est-à-dire, liée à l'architecture logicielle. Dans le cas de l'architecture à 5 couches, la conception fera apparaître les interfaces des services et objets utilisés au niveau de chaque couche

Détaillons l'implémentation de chaque couche, illustrée par cet exemple d'application.

La couche Physique

La couche Physique correspond à la structure physique des données: SGBD, annuaire LDAP, Transaction CICS, ..

Dans le cadre de notre application de news, je propose que nous utilisions une base de données relationnelles, qui contienne les news (table STORY), les catégories (table TOPIC), les auteurs (table AUTHOR) et les

utilisateurs (table USR).

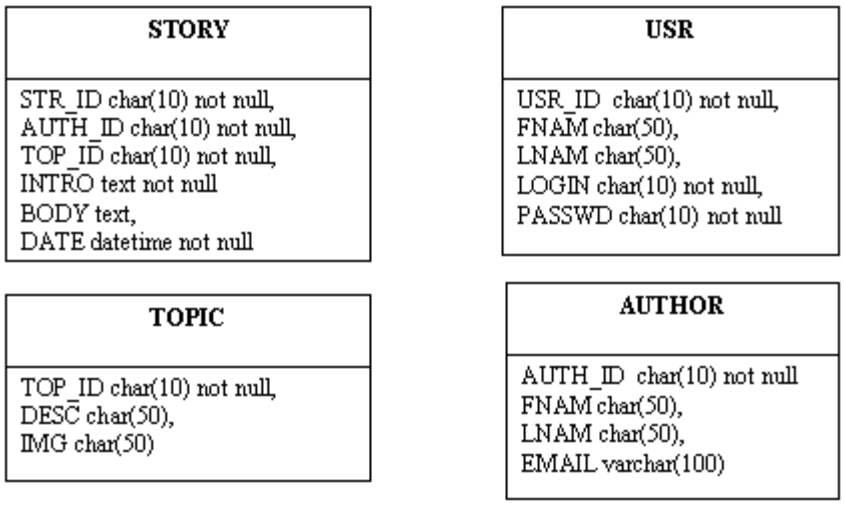

Illustration : Schema relationnel

Remarquez que les colonnes des tables STORY, USR, AUTHOR et TOPIC sont différentes des objets identifiés par l'analyse (si les noms de colonnes sont un peu abscons, c'est pour mieux illustrer le travail des couches supérieures). Il est de la responsabilité de la couche Entreprise de présenter ces tables sous la forme d'objets métier.

La couche Mapping

La couche Mapping transforme la représentation physique des données en une représentation Objet, en tenant compte du langage de programmation. Les expressions « mapping Relationnel / Objet » ou « mécanisme de persistance » sont utilisées.

Les appels nécessaires dans le cadre de notre application sont

1. Lecture de la table STORY pour récupérer la liste des news
2. Insertion d'une nouvelle news dans la table STORY
3. Lecture de la table USR pour réaliser l'authentification
4. Lecture de la table AUTHOR pour récupérer les informations sur l'auteur
5. Lecture de la table TOPIC pour récupérer des informations le topic.

En JAVA, on utilise le protocole JDBC pour effectuer ces appels SQL. Vous pouvez :

- implémenter vous-même les appels JDBC correspondant
- ou bien utiliser un outil de mapping Relationnel / Objet du type TopLink de WebGain, ou bien un OpenSource du type Castor d'Exolab. Ces outils génèrent automatiquement le code JDBC correspondant aux appels SQL (INSERT, DELETE, UPDATE, SELECT).

Implémenter son propre mécanisme de persistance offre un maximum de flexibilité mais une productivité et un niveau de service beaucoup plus faible que l'utilisation d'outils tiers. Par exemple, TopLink propose une gestion de cache, des mécanismes de réservation d'objet, le mappage des types natifs en type Java, le parcours des relations, de l'instantiation paresseuse, le clonage d'objets dans des espaces de modification …

Je vous conseille d'utiliser les règles suivantes dans la couche Mapping :

1. utiliser la même terminologie que la base : ceci permet de rapprocher plus facilement le modèle SQL des classes correspondantes,
2. nommer les classes de mappage avec des préfixes DB : cela permet de bien identifier les classes directement issues de la base,
3. ne pas modifier le code généré par l'outil : retoucher le code généré pose invariablement des problèmes de conservation lors de la génération suivante, c'est la responsabilité de la couche entreprise d'implémenter les transformations de type, les parcours de liens… Le code JAVA généré représente le modèle brut de données.
4. ne pas ajouter de code SQL au code généré par l'outil : le cas échéant, favoriser l'utilisation de procédures stockées.

Dans le cas de notre exemple, vous aurez mappé les tables précédentes dans les objets suivants :

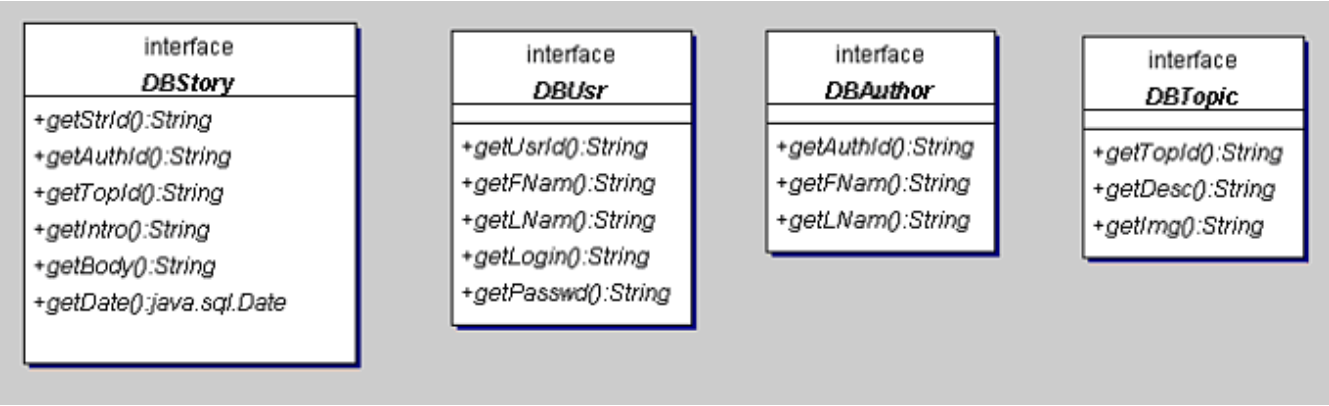

Illustration : Objets de mappage

La couche Entreprise

La couche Entreprise regroupe les objets communs et structurants pour toutes les applications, et garantit l'intégrité des données.

La couche Entreprise est fortement réutilisable d'une application à une autre, parce qu'elle est dénuée des aspects fonctionnels (qui sont de la responsabilité de la couche Application).

Cette couche comprend les tests de cohérence, la mécanique transactionnelle et la traçabilité des appels. Si la base de données n'est pas internationalisée, la couche Entreprise ne possède pas de paramètres de localisation.

En pratique, on trouve au niveau de la couche Entreprise :

- des objets, dont la persistence est assurée par les objets de la couche mapping,
- des services, qui proposent les méthodes de modification, création et suppression d'objet, et les opérations de recherche
- et des exceptions levées en cas de violation des règles d'intégrité dans les services

Les objets Entreprise, candidats dans notre exemple sont : News, CategoryTopic, Author, User. Pour ces objetsil n'y a pas de préfixe, car ils correspondent au vocabulaire que le client a l'habitude d'utiliser.

Les objets Entreprise disposent uniquement d'accesseurs en lecture ; l'écriture est assurée par les services, qui vérifient en même temps le respect des règles d'intégrité. Ils ne présentent pas leur clé, ni aucun autre champ

technique, mais uniquement les champs métiers.

Les objets Entreprise sont organisés selon leur rôle. Dans notre exemple, News, CategoryTopic et Author, relatifs au domaine “ news ” seront regroupés dans le package “ entreprise.news ”, et User dans le package “ entreprise.organisation ”, par exemple.

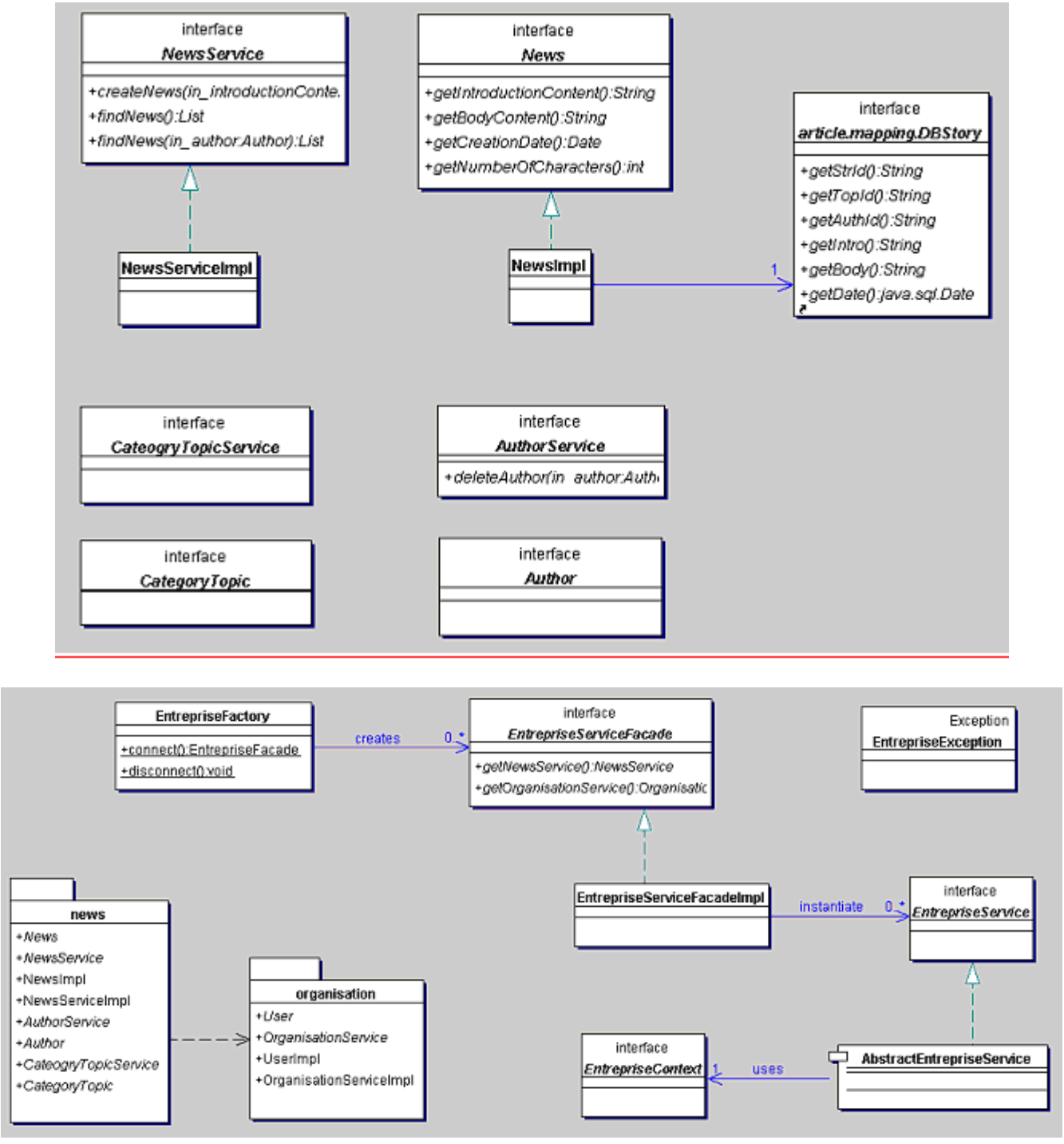

Illustration : Topic et Author ne sont pas détaillés, pour gagner de la place

La couche Entreprise présente ses objets sous la forme d'interfaces. C'est à l'implémentation de réaliser l'accès aux objets de la couche Mapping. Les objets d'implémentation peuvent encapsuler un ou plusieurs objets mapping (fonction de la technologie de mapping Objet / Relationnel et voir article en référence).

Les attributs des objets Entreprise sont de types évolués, c'est à l'objet Entreprise de faire la transformation depuis les types primitifs retournés par les objets encapsulés de la couche Mapping.

En général, on trouve les types de champs suivants :

- types primitifs JAVA : String, int
- collections JAVA : List, Map, Set
- interface d'autres objets métiers
- types contraints : bornes, énumérations, nullable / not nullable
- types métiers, par exemple, Degré de vétusté d'une news, Niveau de complexité du sujet…

Exemple de l'objet News :

```
package entreprise.news ;

/** Objet Entreprise News
 */
// L'interface EntrepriseObjet est une interface de marquage
public interface News implements EntrepriseObject {

  /**
   */
  getIntroductionContent() : String
  getBodyContent() : String
  getCreationDate() : java.util.Date
  getNumberOfCharacters() : int
}
```

Interface de marquage

Interface vide qui permet d'étiqueter des objets.Utiliser une interface de marquage permet :

- de spécifier que le type de l'objet attendu est plus fin qu'Objet
- d'identifier rapidement les impacts sur le code, le jour où l'interface est enrichie.
- Dans la couche Entrerpise, l'interface EntrepriseObject permet de reconnaître les objets Entreprise.

Les services Entreprise permettent de réaliser les opérations de recherche (méthodes find), de création (create), de suppression (delete), de modification (update) et de parcours de liens (navigation).
Ces opérations sont présentées sous la forme d'une interface, qui décrit le contrat du service. Les méthodes d'un service retournent des objets du domaine du service et acceptent en argument des objets du même ou d'autres services ; c'est là qu'intervient le couplage des packages, prenez soin de bien identifier vos relations.

Selon le type de relations entre objets, la navigation a lieu :

- Directement au niveau de l'objet Entreprise, dans le cas d'une relation de type aggrégation ou composition, par exemple, news.getBodyContent(), pour récupérer le contenu de la news
- A travers un service, dans le cas d'une association (1..1, 1..N…), par exemple, authorService.getAuthor(News in_news), pour atteindre l'auteur de la news

Dans notre exemple, l'interface du service « NewsService » se présente comme suit :

```
package entreprise;

/** Service d'accès aux News, TopicCategory et Author
 */
public interface NewsService implements EntrepriseService {

  /** Création d'une news
   */
  public News createNews(
    String in_introductionContent,
    String in_bodyContent,
    Date in_creationDate,
    Author in_author,
    Topic in_topic)
    throws entreprise.EntrepriseException ;

  /** Recherche de tous les news
   */
  public java.util.List findNews()
    throws entreprise.EntrepriseException ;

  /** Recherche des news d'un auteur
   * Retourne une liste vide s'il n'y a pas de news
   *  pour l'auteur spécifié */

  public java.util.List findNews(Author)
    throws entreprise.EntrepriseException;
}
```

Remarque :
Pour augmenter les performances de vos services Entreprise, mettez en place des pools (maximiser les temps de création d'instances) et des caches, d'objets Entreprise. Il est simple d'introduire ces patterns car toutes les méthodes de récupération d'objets sont regroupées au niveau des services Entreprise.

Pour isoler au maximum la couche Application de l'implémentation de la couche Entreprise, il est important que les signatures des méthodes des interfaces des objets et services Entreprise ne soient pas soumises à modification, en fonction du code développé dans les classes d'implémentation.
Pour minimiser les impacts de l'implémentation sur les interfaces, on veillera à ce que toutes les méthodes des interfaces :

- utilisent des types Collections ou Natifs, jamais de classes d'implémentation. Une erreur fréquente est de retourner des Vector par exemple, plutôt qu'une List ou un Set. Ceci contraint à effectuer dans ce cas une copie si on récupérait une List d'un autre type, plutôt que de retourner cette List directement.
- lèvent des exceptions : ainsi si une nouvelle implémentation introduit la levée d'une exception que l'implémentation précédente ne nécessitait pas, la signature de l'interface Entreprise n'est pas modifiée !
- De plus, les exceptions levées doivent toutes être de type EntrepriseException ; ainsi si une nouvelle exception se présente lors de l'implémentation, la couche Entreprise peut la lever sans que la couche Application subisse de modification. Par ailleurs, le polymorphisme de JAVA vous permet de retourner des exceptions, sous classe de EntrepriseException. Dans ce cas, préciser dans le code de la classe d'implémentation, la nature des exceptions que vous levez. Illustrons ce dernier point par un exemple :

```
/** Suppression d'un auteur dans le service AuthorService.
 * Lève une IntegrityException si il existe des news référençant
 * cet auteur
 */

public void deleteAuthor(Author in_author)
    throws EntrepriseException{

  // Vérifier les arguments
  if (in_author == null) {

    // EntrepriseImplementationException sous-classe
    // EntrepriseException
    throw new EntrepriseImplementationException(« Null argument in_author ») ;
  }

  // Recherche de news liées à l'author
  if (newsService.findNews(in_author).size() > 0) {
    // EntrepriseIntegrityException sous-classe
    // EntrepriseException
    throw new EntrepriseIntegrityException(« Please, remove the news of » + in_author + «, first ») ;
  }

  // Supprimer l'author en appelant la couche Mapping
  ….
}
```

Au niveau de l'implémentation des services, il est utile d'introduire un contexte, qui maintienne les informations de connexion vers la couche mapping, le code de l'utilisateur à des fins de traçabilité, et d'autres objets utilitaires. Nommer cet objet EntrepriseContext.

Si votre application utilise des transactions, vous pourrez utiliser ce contexte pour y placer le contexte transactionnel. Même si nous n'abordons pas les transactions dans cet article, notez les points suivants :

1. Dans le cas d'une transaction courte, démarrer et terminer la transaction dans une même méthode de la couche Application avec une clause finally qui garantisse que la transaction est toujours terminée (que ce soit un commit ou un rollback).

2. Dans le cas d'une transaction longue, créer un service de suivi de contexte transactionnel qui assure la bonne terminaison de la transaction, en ajoutant un timeout, au-delà duquel la transaction est rollbackée.

En ce qui concerne les services eux-même, il est pratique de proposer une classe abstraite, qui contienne ce contexte.

```
package entreprise ;

/** Squelette des services de la couche Entreprise
 */
public abstract class AbstractEntrepriseService
    implements EntrepriseService {

  /** Entreprise Context utilisé par le service
   */
  protected EntrepriseContext context = null ;

  /**  Constructeur
  */
  protected AbstractEntrerpiseService(EntrerpiseContext in_context) {
    super() ;
    context = in_context ;
  }
}
```

Nous disposons donc des services Entreprise : NewsService et OrganisationService. Créons maintenant une " Facade " au niveau du package « entreprise », qui délivre ces services.

```
package entreprise ;

/** Facade sur les services de la couche Entreprise */
public class EntrepriseServiceFacade {

  /** Retourne une instance NewsService
  */
  public NewsService getService()  throws EntrepriseException ;

  /** Retourne une instance OrganisationService
  */
  public OrganisationService getOrganisationService()
    throws EntrepriseException ;
}
```

Je vous conseille de créer une dernière classe qui se présente comme un point de passage obligé pour entrer dans la couche Entreprise. Je vous propose de nommer cette classe EntrepriseFactory.
Tout est envisageable au niveau de l'implémentation, à étudier en fonction de vos contraintes d'implémentation :

- Pool de connexions,
- Contexte transactionnel…

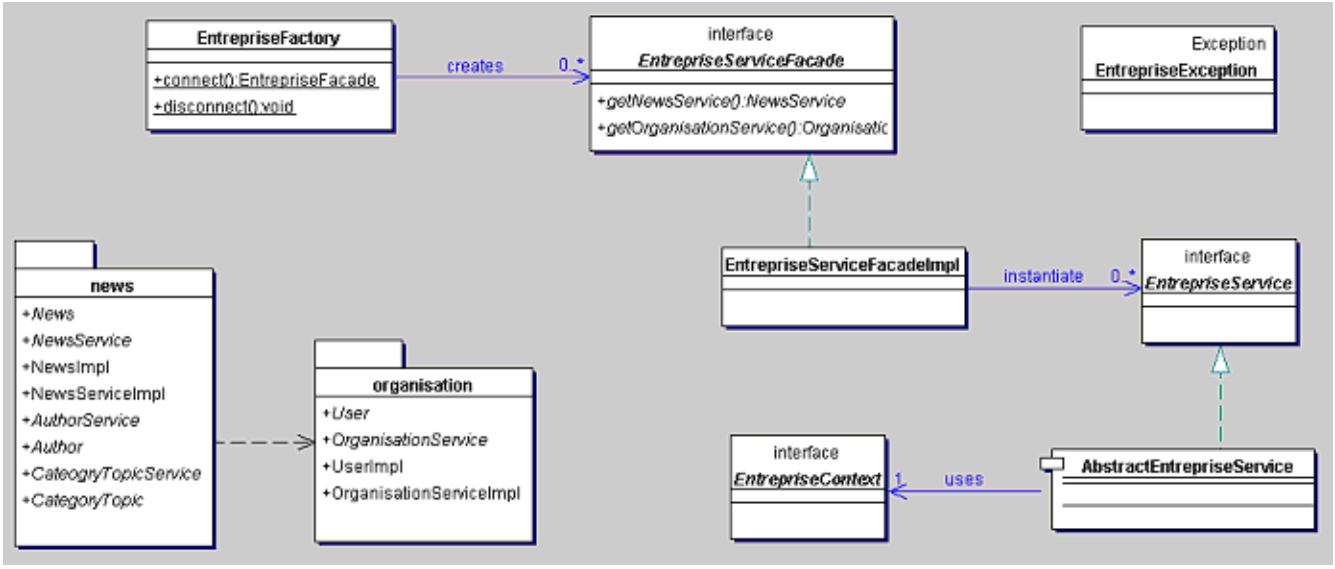

Illustration : structure de la couche Entreprise

Couche Application

La couche Application contient la logique fonctionnelle de l'application, que ses services implémentent en appelant les services Entreprise.
La couche Application est modifiée au rythme de l'évolution des spécifications fonctionnelles.

Dans l'exemple du Portail de News, la couche Application dispose de 2 services :

- AuthenticationService : réalise l'authentification de l'utilisateur, utilisé par la bannière de login (couche Client)
- PublishingService : fonctions de publication et visualisation de news

En plus d'isoler le fonctionnel, la couche Application permet de proposer plusieurs domaines fonctionnels reposant sur la même couche Entreprise.
Si l'on reprend notre exemple, on pourrait imaginer une deuxième application, chargée de l'administration du portail. On trouverait alors des fonctions d'administration d'auteurs et de catégories. Ces services peuvent s'appuyer sur les services AuthorService et TopicCategoryService dans modification de ceux-ci. On retrouve ici le but premier de la couche Entreprise : la réutilisabilité.
La couche Application a quant à elle des objectifs de maintenabilité et d'évolutivité, face au fonctionnel.

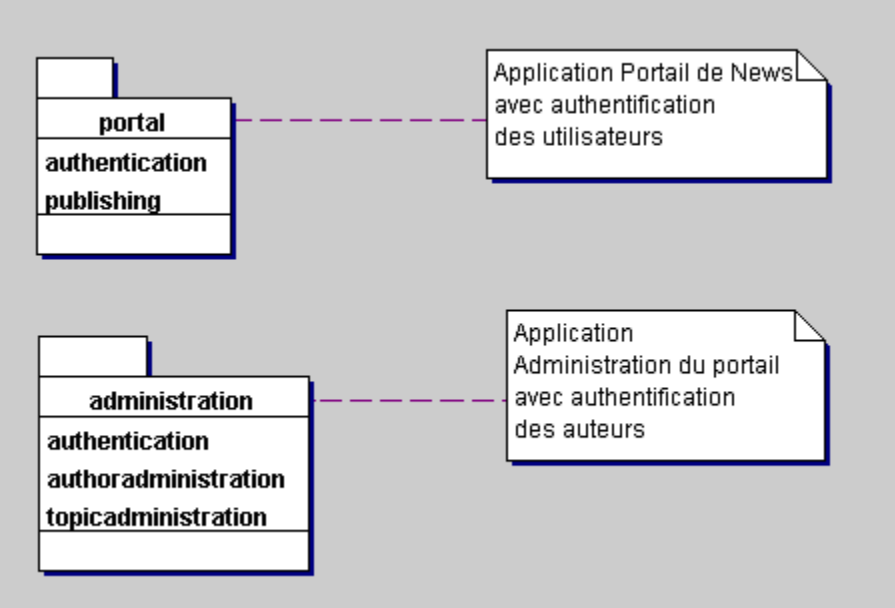

Illustration : Les services de la couche Application

Les 2 services « authentication » sont indépendants : l'un interroge les User de la couche Entreprise, l'autre peut être built-in (root / root).

Revenons sur l'exemple du portail.

Le service « Authentication » s'appuie uniquement sur le service Organisation de la couche service, et propose la méthode authenticate pour authentifier l'utilisateur à partir d'un couple (login, password). Une partie de son implémentation pourrait être réalisée comme suit :

```
/** Authentification selon les paramètres spécifiés
 */
public boolean authenticate(String in_login, String in_password)
throws ApplicationException {

  // Toujours commencer par vérifier les arguments, c'est un gage
  // de robustesse pour l'application développée
  if ((in_login == null) || (in_passwd == null)) {
    throw new ApplicationException("null argument ") ;
  }

  // On aura préalablement récupéré une EntrepriseServiceFacade
  // avec le code
  // entrepriseServiceFacade  = EntrepriseFactory.connect()
  try {
    // Récupération du service organisation
    OrganisationService l_service = entrepriseServiceFacade.getOrganisationService() ;

    // Récupération de l'utilisateur
```

```
    User l_user = l_service.findUser(in_login) ;

    // Authentification
    String l_passwd = l_user.getPassword() ;
    return (in_passwd.equals(l_passwd)) ;
  }
  catch (EntrepriseException e) {
    // Les services Application catchent les exceptions de
    // niveau entreprise pour retourner des applications
    // de leur niveau
    throw new ApplicationException(" Problème lors de l'authentification de l'utilisateur") ;
  }
}
```

Remarquez que :

La couche Application traite les EntrepriseException et remonte des exceptions de type Application. Les erreurs renvoyées par la couche Application sont à destination de l'utilisateur, les libellés doivent donc être pertinents, internationalisable,… Pour bien traiter ce point il est souvent recommander de s'appuyer sur un framework. Dans notre cas, afin de limiter la complexité de l'exemple, les messages renvoyés à l'utilisateur sont réduits au minimum.

A aucun moment, la couche Application ne présume des mécanismes de stockage des logins et mots de passe utilisateur ; ceux-ci pourraient être déplacés dans un annuaire LDAP sans entraîner aucune modification de la couche application, et à fortiori de la couche Client (c'est bien l'objectif de l'architecture à 5 couches J).

Passons maintenant au service « Publishing ».
Le service Publishing permet à la couche Client :

- De récupérer la liste des news
- De publier une nouvelle news

Dans le service Publishing, une news est constituée de :

- Un tître
- Une catégorie
- Un auteur
- Un contenu
- Une date de soumission

L'on comprend que la vue applicative est différente de la vue Entreprise. C'est la responsabilité de la couche Application de construire le fonctionnel à partir des services exposés par la couche Entreprise

Le modèle d'implémentation de la couche Application ressemble à celui de la couche Entreprise, on retrouve les notions :

- d'objets Application, qui présentent une vue fonctionnelle des objets Entreprise
- de services Application, qui exposent les fonctions
- d'exceptions de niveau Application, qui sont utilisées si les contraintes fonctionnelles ne sont pas respectées.

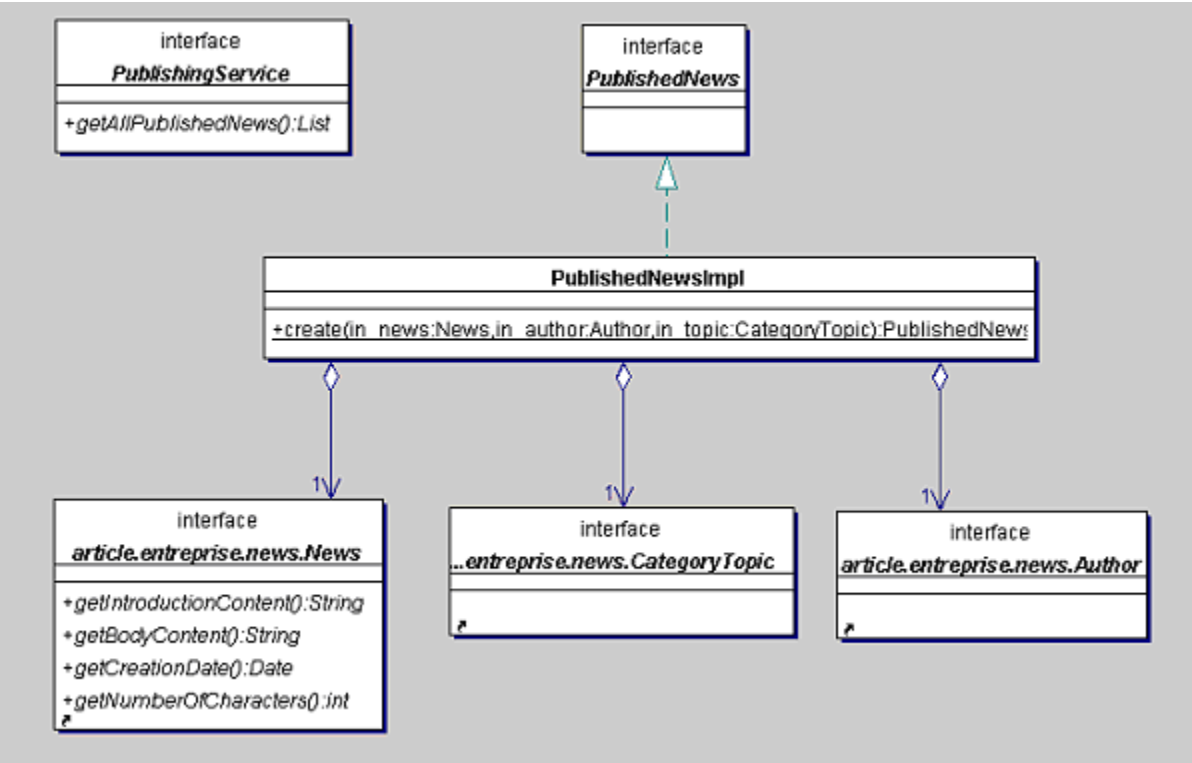

Illustration : Service publishing

Voici un exemple d'implémentation du service Publishing :

```
package application.publishing ;

/** Implémentation du service  Publishing
 */
public class PublishingServceImpl
    extends AbstractApplicationService
    implements PublishingService {

  /** Récupération de la liste des news
  */
  public List getAllNews () throws ApplicationException{
    try {
      // Récupérer la facade EntrepriseServiceFacace
      // stockée au niveau de l' AbstractApplicationService
      // par exemple
      List l_news =
        getEntrepriseFacade().getNewsService().findNews() ;
      Vector l_listOfPublishedNews = new Vector(l_news.size()) ;

      Iterator l_iterator = l_news.iterator() ;
      while (l_iterator.hasNext) {
        News l_news = (News) l_iterator.next() ;
        Author l_author =
```

```
          getEntrepriseFacade().getAuthorService().
          findAuthor(l_news) ;
        CategoryTopic l_ category = getEntrepriseFacade().getCategoryTopicService().findTopic(l_news) ;
        PublishedNews publishedNews = PublishedNewsImpl.create(l_news, l_author, l_category) ;
 ListOfPublishedNews.addElement(l_publishedNews) ;
}

return listOfPublishedNews ;
}
catch (EntrepriseException e) {
 throw new ApplicationException(« Impossible de récupérer la liste des news ») ;
}
 }
}
```

et l'implémentation de l'object application « PublishedNews »

```
package application.publishing ;

import entreprise.news.* ;
import entreprise.organisation.* ;

/** Implémentation de PublishedNews
 */
public class PublishedNewsImpl implements PublishedNews {
 /** Constructeur
· Parce qu'un constructeur ne peut pas vérifier ses arguments avant
· d'appeler le constructeur de sa super-classe,
· Il est indispensable que les arguments soient vérifiés avant l'appel au constructeur
· Pour assurer ceci, on spécifie le « modificateur » protected
· Et on crée une factory à travers la méthode de classe « create »
  */
 protected PublishedNewsImpl (News in_news, entreprise .Author in_author, entreprise.TopicCategory
in_category) {
super() ;

// affectation des champs de la news
title = in_news.getIntroductionContent() ;
  author = in_author.getFirstName() + « » «+ in_author.getLastName() ;
  topic =  in_category.getName() ;
  dateCreation = in_news.getCreationDate() ;
  content = in_news.getBodyContent() ;
}

 /** Monteur
   */
 public static PublishedNewsImpl create (News in_news, entreprise .Author in_author, entreprise.TopicCategory
in_category) throws ApplicationException {
  // Vérification des arguments
```

```
 if ((in_news == null) || (in_author == null) || (in_category == null)) {
  throw new ApplicationException(«Null argument specified ») ;
 }

  // Construction de la news
  try  {
   // Selon le choix d'implémentation, on utilisera une composition ou bien une copie des champs des objets
entreprise
return new PublishedNewsImpl (in_news, _in_author, in_category) ;
  }
catch (EntrepriseException e ){
 throw new ApplicationException(« Problème lors de la  récupération des champs de la news ») ;
}

}
}
```

Modificateur (« Modifier » en anglais)
Spécifie si un champ, une classe ou une méthode est public, protected, package ou private

Pour clore l'implémentation de la couche Application, ajouter les objets de :

- ApplicationContext : chargé de suivre le contexte transactionnel dans le cas de transaction longue
- AbstractApplicationService : classe de référence pour les services de la couche Application
- ApplicationServiceFacade : délivre l'accès aux services de la couche Application
- et ApplicationFactory : délivre une ApplicationServiceFacade, point de passage obligé de la couche Application où on pourra par exemple, inclure des logs, des pools…

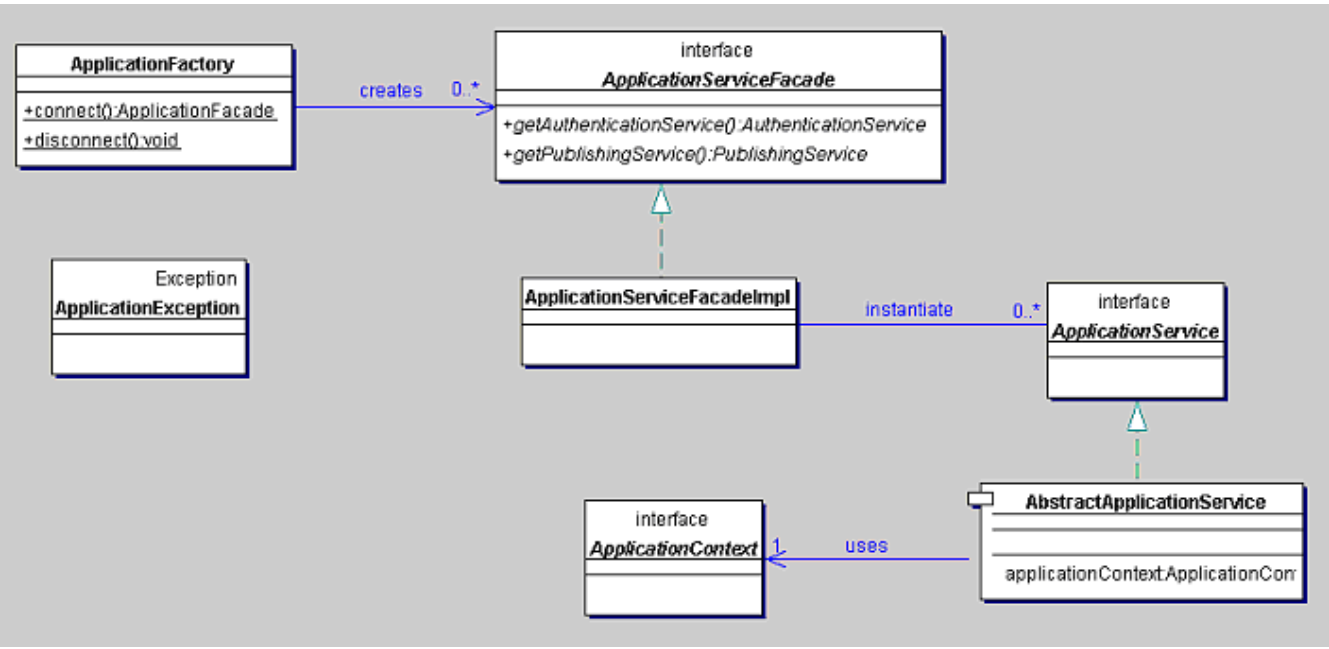

Illustration : Structure de la couche Application

Couche client

Disons que vous avez 3 possibilités à ce niveau :

- utiliser un AGL, ce qui correspond généralement à modèle évènementiel, où vous définissez le comportement lié à chaque événement, à travers un langage de script plus ou moins lié à l'AGL, ou natif (C++, Java…)
- utiliser un modèle MVC (où vous distinguerez les objets " Vue " et " Controleur " les données étant quant à elles générées par la couche application).
- ajouter une couche navigation, cette couche est responsable de l'enchaînement des écrans. Introduire une telle couche est déterminant si vous souhaitez piloter la navigation, ce qui est de plus en plus souvent le cas à l'ère de la personnalisation.

Remarque : vous pouvez cumuler modèle MVC et couche navigation.

Personnalisation
Il se dit beaucoup de choses autour de ce sujet. D'une façon générale il s'agit de proposer à l'utilisateur des fonctionnalités adaptées à ses besoins. Il est donc nécessaire d'un côté de cerner les besoins de cet utilisateur (profiling) et d'un autre côté d'être capable d'adapter l'application en fonction du profil de l'utilisateur. Ainsi, on pourra par exemple, pré-renseigner certains écrans de saisie, voire ajouter ou supprimer des écrans à l'application qui est présentée à l'utilisateur.

Personnalisation

Il se dit beaucoup de choses autour de ce sujet. D'une façon générale il s'agit de proposer à l'utilisateur des fonctionnalités adaptées à ses besoins. Il est donc nécessaire d'un côté de cerner les besoins de cet utilisateur (profiling) et d'un autre côté d'être capable d'adapter l'application en fonction du profil de l'utilisateur.

Ainsi, on pourra par exemple, pré-renseigner certains écrans de saisie, voire ajouter ou supprimer des écrans à l'application qui est présentée à l'utilisateur.

Nous ne détaillerons pas la couche client car son implémentation est directement liée à la technologie et à l'architecture utilisée. Dans tous les cas, retenez que la couche Client se contente d'effectuer des appels à la couche Application.

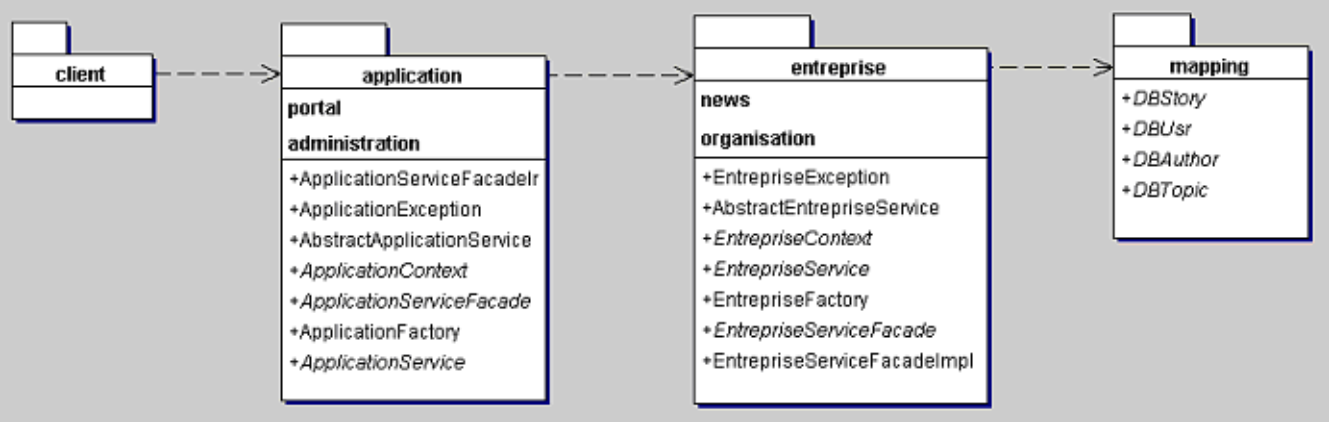

Illustration : Enchaînement des couches dans le cas de l'application Portail de News

Créer son framework technique
En isolant interfaces et classes des packages de premier niveau dans un groupe depackage distinct, vous pourrez réutiliser ce modèle dans vos prochains développements.
L'on parle alors de « framework technique ».

Placer vos classes et interfaces dans des packages du type :
<societe>.framework.multicouche.<nomdelacouche>

Exemple :
fr.improve.framework.multicouche.client
fr.improve.framework.multicouche.application
fr.improve.framework.multicouche.entreprise
fr.improve.framework.multicouche.mapping

Framework technique
Un framework technique regroupe les composants techniques, qui implémentent les contraintes d'architecture logicielle, tout en restant détachés du fonctionnel.

Développer des frameworks techniques est non seulement un gage de qualité, mais facilite aussi l'intégration de nouveaux développeurs.

Conclusion
Je vous ai présenté un exemple d'implémentation du modèle à 5 couches. Cet exemple reposant fortement sur l'utilisation d'interfaces, vous pouvez partir de ce squelette pour créer votre framework technique qui intégrera vos contraintes de :

- sécurité
- configurabilité
- transactionnel (mono ou multi référentiels)
- et les outils couramment utilisés par vos équipes

POUR ALLER PLUS LOIN

Mapping Relationnel / Objet par Scott Ambler
http://www.ambysoft.com/mappingObjects.pdf

UML en action
http://www.amazon.fr/exec/obidos/ASIN/2212091273/402-1083506-8339333

L'outil TopLink
http://www.webgain.com/products/toplink/

L'outil Castor
http://castor.exolab.org/

Auteur

De formation Télécom, Stève SFARTZ est consultant senior chez IMPROVE SA. SUN Certified JAVA Programmer et IBM VisualAge for JAVA Certified Programmer depuis 1998, Stève met en place des architectures JAVA depuis 1996 (Serveur d'application J2EE, Transactionnel, Client / Serveur JFC, architectures INTRA / INTERNET, architectures Objet / UML). Stève anime par ailleurs des conférences et des formations sur les technologies VisualAge, WebSphere, Processus de développement Objet, Architecture Objet UML...