

EPELFI AMALFI

Architecture Applicative

Conception Technique : Principes et définitions

Historique des versions			
Version	Création	Description des évolutions	Auteur(s)
1.0	25 Mai 2006	Création du document	RCo, COA, JLF
1.1	25 mai 2007	Complément sur les traces	NLu
1.2	6 juillet 2007	Complément sur communication SSEE Complément console surveillance applicative Complément Impression. Complément sécurité applicative.	RCo
1.2.1	12 juillet 2007	Complément sécurité applicative. Complément Accès GED pour SSEE	Rco
1.2.2	8 juillet 2008	Mise à jour de la surveillance applicative	TMo
2.0	26/01/2011	TMAC-1451 Surcharge CPU : en cas d'utilisation des touches "back" ou "refresh", invalider la session utilisateur	DPI
3.0	10/11/2011	TMAC-1873 : Journalisation des actions effectuées dans AMALFI (SC+SS2E)	
4.0	27/02/2012	TMAC-1373 : précision en cas de concurrence d'accès	MIG
5.0	22/03/2012	TMAC-2093 Double soumission : mise en place de la solution définie en réunion le 27/09/2011 -> Retour arrière sur la TMAC-1451	NAD/PAK
5.1	29/03/2012	Prise en compte PVAT-472	NAD/DPI

Référence	Auteur		Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK		1/77

6.0	15/11/2016	Prise en compte TMAC-4728	PAK
6.2	05/12/2016	Prise en compte des retours de l'EPELFI	PAK

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			2/77

Table des matières

1.	Introduction	9
1.1.	Objectifs du document	9
1.2.	Limitations	9
2.	Architecture AMALFI	9
2.1.	Principes généraux	9
2.1.1.	L'architecture applicative : un modèle en couches, la topologie	10
2.1.2.	Vue déployée des couches de l'architecture	11
2.1.3.	Vue des principaux composants par couches de l'architecture	12
2.1.4.	Vue détaillée des composants par couches de l'architecture	13
2.1.5.	Principe du modèle de base des objets métiers (versions)	16
2.1.5.1.	Les domaines de données	17
2.1.5.2.	La gestion des identifiants et des versions.....	18
2.1.6.	Accès aux données et sécurité	19
2.2.	Principes et définitions transverses	19
2.2.1.	La gestion des erreurs (Exceptions)	20
2.2.1.1.	Les erreurs attendues	20
2.2.1.2.	Les erreurs inattendues	21
2.2.2.	Les traces applicatives et de débogage	21
2.2.3.	Les traces applicatives	21
2.2.3.1.	Rapports journaliers.....	21
2.2.3.2.	Structure du fichier de traces applicatives	22
2.2.3.2.1.	Données communes.....	22
2.2.3.2.2.	Données spécifiques par événement	22
2.2.3.2.2.1.	L'appel des services	22
2.2.3.2.2.2.	Le démarrage d'un processus	22
2.2.3.2.2.3.	L'exécution d'une action de processus	23
2.2.3.2.2.4.	L'arrêt d'un processus	23
2.2.3.2.2.5.	La réactivation d'un processus	23
2.2.3.2.2.6.	Données Console	23
2.2.3.2.3.	Evénements Erreur	24
2.2.3.2.3.1.	Erreurs attendues	24
2.2.3.2.3.2.	Erreurs inattendues	24

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			3/77

2.2.3.2.4.	Mise en œuvre pour la supervision applicative	24
2.2.3.3.	Trace du système de supervision	25
2.2.4.	Les traces de déboguage	25
2.2.4.1.	Traces techniques	25
2.2.4.2.	Traces des erreurs (Exceptions).....	25
2.2.5.	Les traces d'audit	25
2.2.6.	La console de surveillance applicative	25
2.2.7.	Le composant d'assemblage et d'initialisation de l'application	26
2.2.8.	La sécurité applicative	27
2.3.	Principes et définitions du Front	29
2.3.1.	Principes et définitions de la couche navigation	31
2.3.1.1.	Struts	31
2.3.1.2.	Classes Action et Formulaire Struts	31
2.3.1.3.	Une classe Action Struts mais plusieurs actions Struts	31
2.3.1.4.	Les processus métiers (vus de la navigation)	31
2.3.1.5.	Le couple Action Struts / Processus Métier.....	32
2.3.1.6.	La session utilisateur (vue de la navigation)	32
2.3.1.7.	De l'action Struts vers le Processus Métier	32
2.3.1.8.	La clé de la navigation est l'état du Processus métier	32
2.3.1.9.	Un mécanisme de pile pour les processus métiers	32
2.3.1.10.	Enchaînement d'écrans	33
2.3.1.11.	Le formulaire Struts	33
2.3.1.12.	Construction du formulaire HTML dans la JSP	33
2.3.1.13.	Prise en compte du formulaire reçu dans la requête	33
2.3.1.14.	Contrôle de la navigation hors actions applicatives (boutons navigateur)	34
2.3.1.15.	Les exceptions (inattendues) vues par l'action Struts	35
2.3.1.16.	Synthèse pour la couche navigation	35
2.3.1.16.1.	Les transitions Page - Action - Processus	35
2.3.1.16.2.	Les interactions Page - Formulaire Données du Processus.....	36
2.3.2.	Principes et définitions de la couche coordination	38
2.3.2.1.	Le contexte Utilisateur.....	38
2.3.2.2.	Les processus métiers	39
2.3.2.3.	Le cache et les données métiers.....	39
2.3.2.3.1.	Les données métiers	39

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			4/77

2.3.2.3.2.	Le cache.....	40
2.3.2.3.3.	Les transactions	40
2.3.2.4.	Synthèse de la couche coordination.....	41
2.3.2.4.1.	Exemple de comportement de la pile de processus métiers	41
2.3.3.	Principes et définitions de la partie présentation	42
2.3.3.1.	Ergonomie, Charte graphique et Accessibilité	43
2.3.3.2.	Présentation des données à l'utilisateur	44
2.3.3.3.	Édition des états imprimés.....	44
2.3.3.3.1.	Les impressions génériques.....	45
2.3.3.3.2.	Les impressions spécifiques	45
2.4.	Principes et définitions de l'accès aux services	46
2.4.1.	Vue générale de l'accès aux services en EJB (Site Central)	47
2.4.2.	Vue générale de l'accès aux services en mode local (Test)	48
2.4.3.	Vue générale de l'accès aux services en MDB (SSEE)	49
2.4.3.1.	Rappel des choix et contraintes pour le SSEE.....	49
2.4.3.2.	Vue logique de la solution de communication pour le SSEE.....	51
2.4.3.3.	Vue technique de la solution de communication pour le SSEE	52
2.4.3.3.1.	Diagramme de classes	52
2.4.3.3.2.	Diagrammes de collaboration.....	54
2.4.4.	La partie Front de la couche d'accès aux services	56
2.4.4.1.	Le gestionnaire de services	56
2.4.4.2.	La fabrique des proxys de services	57
2.4.4.3.	Le proxy de service local.....	57
2.4.4.4.	Le proxy de service EJB	57
2.4.4.5.	Le proxy de service JMS	57
2.4.5.	La couche back d'accès aux services	57
2.4.5.1.	L'EJB Accès services (MDB pour SSEE, Session pour SC)	57
2.4.5.2.	La fabrique des services métiers.....	58
2.4.6.	Schéma de synthèse des principes et définitions	58
2.5.	Principes et définitions du Back	59
2.5.1.	Principes et définitions de la couche service	60
2.5.1.1.	Le contexte d'appel de service	61
2.5.1.2.	Le contexte du service	61
2.5.1.3.	Le cache et les données métiers.....	61

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			5/77

2.5.2.	Principes et définitions de la couche persistance	61
2.5.2.1.	Principes structurants.....	62
2.5.2.1.1.	Classification des objets	62
2.5.2.1.2.	Concept de version	62
2.5.2.1.3.	Référence entre Objets	63
2.5.2.1.4.	Référence vers un objet dont le numéro Amalfi est momentanément inconnu	64
2.5.2.1.5.	Concept d'objet temporaire	64
2.5.2.1.6.	Objet versionnable et temporaire	65
2.5.2.1.7.	Concurrence d'accès	65
2.5.2.1.8.	Objets du référentiel	65
2.5.2.1.9.	Référence d'un item vers des items d'autres énumérations	66
2.5.2.1.10.	Collection d'items référençant un item donné	66
2.5.2.2.	Organisation des données dans la base.....	66
2.5.2.2.1.	Objets Lf.....	66
2.5.2.2.2.	Utilisation d'un discriminant.....	67
2.5.2.2.3.	Discriminant d'un objet référencé.....	67
2.5.2.2.4.	Timestamp et User technique	67
2.5.2.2.5.	Table des versions publiées.....	68
2.5.2.2.6.	Table des versions temporaires	68
2.5.2.2.7.	Table Index des entités publiées.....	69
2.5.2.2.8.	Table Index des entités temporaires	69
2.5.2.2.9.	Master de toutes les versions.....	70
2.5.2.2.10.	Division d'une table en sous tables	71
2.5.2.2.11.	Objets du référentiel	71
2.5.2.2.12.	Table des versions	71
2.5.2.2.13.	Objets du registre des dépôts.....	73
2.5.2.3.	Module d'accès à la persistance	73
2.5.2.3.1.	Les méthodes d'accès à la persistance	73
2.5.2.3.2.	Obtention d'un identifiant valide	74
2.5.2.3.3.	Chargement d'objets par leur identifiant.....	74
2.5.2.3.4.	Persistance d'objets en base de données	74
2.5.2.3.5.	Recherche d'objets en base de données	74
2.6.	Accès GED	74

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			6/77

2.7. Le contrôle des performances

76

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			7/77

Table des figures

Figure 1 : Topologie du modèle en couche de l'architecture applicative	10
Figure 2 : Vue déployée des couches de l'architecture applicative	11
Figure 3 : Vue de l'architecture applicative des principaux composants dans les couches	12
Figure 4 : Vue détaillée de l'architecture applicative des composants dans les couches.....	14
Figure 5 : Les grands domaines du modèle de données	17
Figure 6 : La gestion des versions par l'exemple	18
Figure 7 : Accès aux données et sécurité	19
Figure 8 : Principes de la couche transverse	20
Figure 9 : Architecture logique de la console de surveillance.	26
Figure 10 : Principes de la sécurité applicative (Structure de données).....	28
Figure 11 : Principes de la sécurité applicative (Dynamique)	29
Figure 12 : Principes du Front.....	29
Figure 13 : Vue générale des couches Navigation et Coordination	30
Figure 14 : Principes de la couche navigation.....	31
Figure 15 : Exemple de navigation	Erreur ! Signet non défini.
Figure 16 : Exemple de transition Page - Action - Processus	35
Figure 17 : Exemple d'interaction Page - Formulaire	37
Figure 18 : Principes de la couche coordination	38
Figure 19 : Comportement de la pile Processus et interactions Processus-Action-JSP	42
Figure 20 : Principes de la présentation	42
Figure 21 : Vue générale des couches navigations et présentation du point de vue de la présentation des informations.....	43
Figure 22 : Diagramme de synthèse de la construction d'un PDF pour impression.	46
Figure 23 : Principes de l'accès aux services	46
Figure 24 : Vue générale de la couche d'accès aux services en mode distant synchrone (EJB)	48
Figure 25 : Vue générale de la couche d'accès aux services en mode local	49
Figure 26 : Synthèse des principes et définitions de la couche d'accès aux services	58
Figure 27 : Principes du Back	59
Figure 28 : Vue générale des couches Service et Persistance	59
Figure 29 : Principes de la couche service	60
Figure 30 : Principes de la couche persistance.....	61
Figure 31 : Le concept de version	63

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			8/77

Figure 32 : Table des versions et Index des entités	64
Figure 33 : Concept d'objet temporaire	65
Figure 34 : Les objets versionnables et temporaires	65
Figure 35 : Référence d'un item vers des items d'autres énumérations.....	66
Figure 36 : Exemple de table de versions publiées	68
Figure 37 : Exemple de table de versions temporaires.....	69
Figure 38 : Exemple de table index d'entités publiées.....	69
Figure 39 : Exemple de table index d'entités temporaires	70
Figure 40 : Exemple d'énumération sans référence	72
Figure 41 : Exemple d'énumération avec référence(s)	72
Figure 42 : synthèse simplifié de l'accès GED pour le SC.	75
Figure 43 : synthèse simplifié de l'accès GED pour le SSEE.....	76
Figure 44 : Le contrôle des performances	76

1. Introduction

1.1. *Objectifs du document*

Ce document regroupe les principes et définitions de l'architecture de l'application Amalfi. Cette architecture étant découpée en couches, chaque couche donnera lieu à une partie du document. La première partie étant réservée aux concepts généraux n'appartenant pas spécifiquement à une couche.

1.2. *Limitations*

On postule dans ce document que les principes généraux de JEE (java enterprise edition), des framework Struts, Spring et de la conception orientée objet sont connus du lecteur.

2. Architecture AMALFI

2.1. *Principes généraux*

Cette partie présente les principes généraux utilisés dans l'architecture applicative. Ces principes seront repris et détaillés dans les parties suivantes.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			9/77

2.1.1. L'architecture applicative : un modèle en couches, la topologie

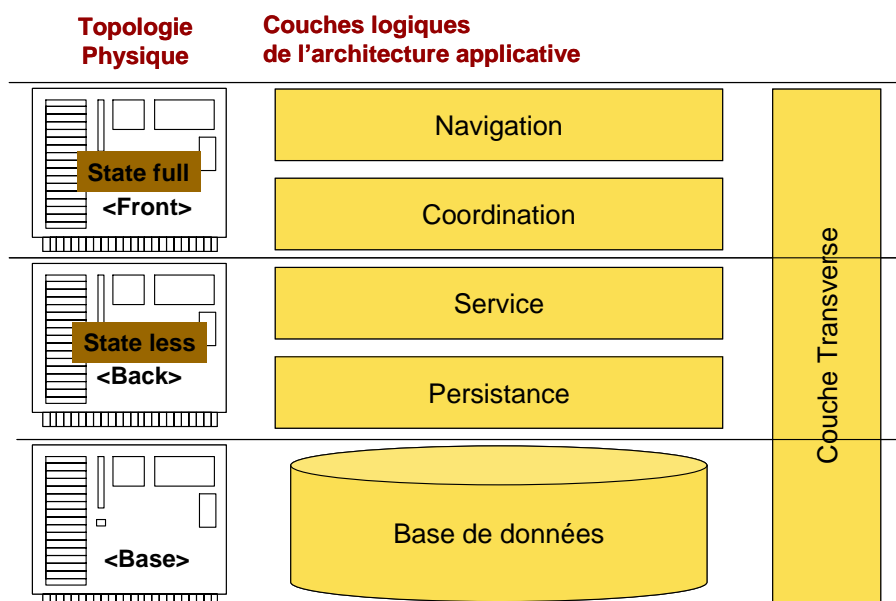


Figure 1 : Topologie du modèle en couche de l'architecture applicative

L'architecture de l'application Amalfi V2 suit un modèle en couches, chaque couche ayant un rôle spécifique. Les couches permettent une construction plus simple et plus logique des briques de déploiement (EAR), sur les serveurs (logique) Front et Back.

La couche Navigation a pour rôle de gérer la navigation dans l'application entre les différents écrans. Elle repose sur le Framework Struts de l'organisation Apache qui met en œuvre le paradigme MVC2 (Modèle – Vue – Contrôleur version 2). Cette couche manipule directement des concepts J2EE tels que la requête http, les servlets, les JSP, la session http, ...

La couche Coordination permet de définir la logique applicative en terme de processus fonctionnels (groupe d'écrans). Ces processus fonctionnels traitent les requêtes utilisateurs : c'est-à-dire qu'ils vérifient les règles de gestion et appellent des services pour lire et modifier la base de données. Cette couche est dite Statefull puisque entre deux requêtes d'un utilisateur elle conserve les informations contextuelles du processus en les stockant dans le contexte utilisateur (1 contexte par utilisateur). Cette couche est découplée des concepts J2EE.

La couche Service définit les services utilisés par la couche Coordination. Elle est dite Stateless puisque les services ne conservent pas d'information contextuelle entre 2 appels. Elle utilise la couche Persistance pour accéder (en lecture ou modification) en base. Cette couche ne manipule pas de concepts J2EE.

Un composant d'accès aux services se retrouve entre la couche Coordination et la couche Service, il permet à la couche Coordination d'appeler la couche Service en masquant la utilisation du middleware : ce rôle est important car le SC et le SSEE n'utilise pas le même (EJB/RMI pour SC et JMS pour SSEE).

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			10/7 7

La couche Persistance définit les objets données (les objets métiers) et de faire le mapping avec les tables de la base de données. Elle offre aussi à la couche Service l'opportunité de récupérer (pour lecture ou modification) ou créer des données en base. Cette couche ne manipule pas de concepts J2EE.

Enfin la couche transverse regroupe les composants utilisés par toutes les couches comme les exceptions. Il n'y a pas de concepts J2EE non plus dans cette couche.

Ainsi les couches Navigation et Coordination sont déployées dans le serveur « logique » « Front », les couches Service et Persistance sont déployées dans le serveur « logique » « Back ». La couche transverse est commune au Front et au Back, elle est donc déployée dans les deux. Le modèle en couches permet aussi de cantonner les concepts J2EE à certaines couches. Les autres couches sont non J2EE et peuvent être testées de façon plus simple (sans déployer dans un conteneur J2EE).

2.1.2. Vue déployée des couches de l'architecture

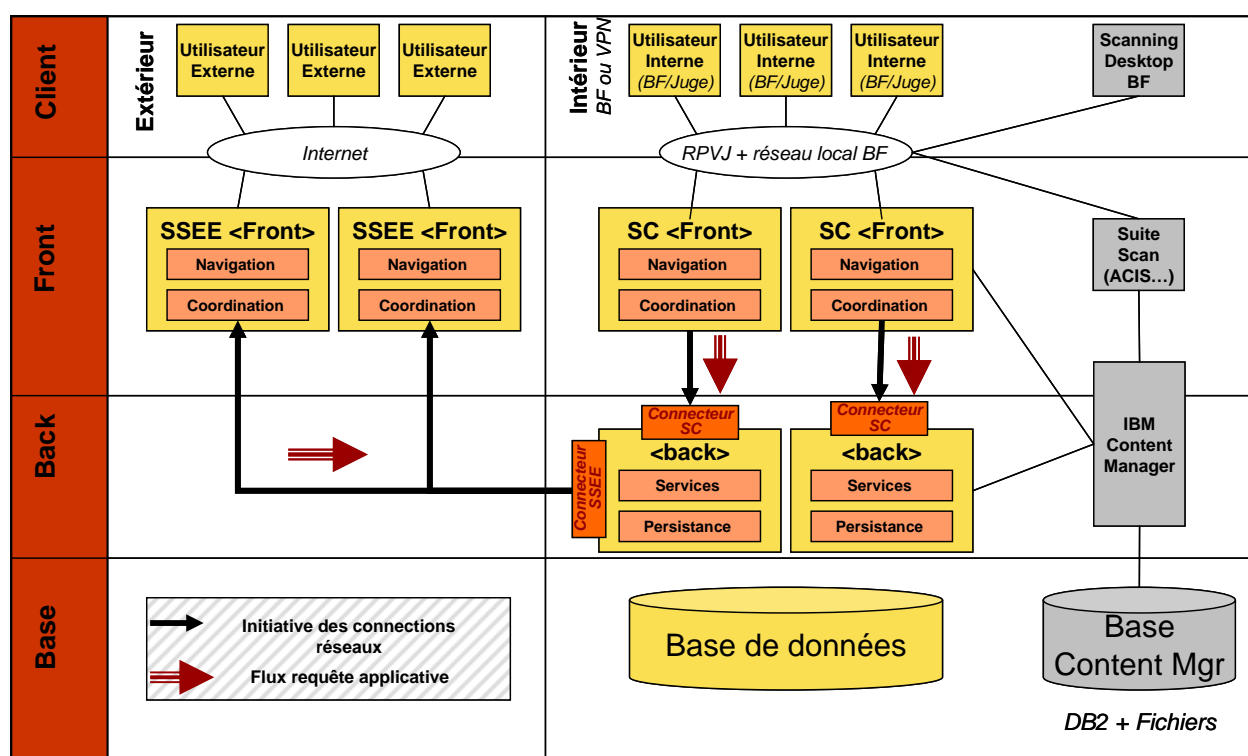


Figure 2 : Vue déployée des couches de l'architecture applicative

Le schéma ci-dessus montre une infrastructure logique simplifiée où les serveurs logiques ont été doublés pour montrer la possibilité de répartition de la charge et de gestion de la robustesse. Le déploiement des différentes couches de l'architecture applicative apparaît dans chacun des serveurs logiques.

La colonne de droit décrit le site central avec :

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			11/7 7

- sa base de données,
- accédée par les serveurs logiques « back » contenant les couche Persistance et Services
- La couche Service du back est accédée par le « front » SC via un connecteur dédié (technologie EJB).
- Les serveurs logiques « front » hébergent les couches coordination et navigation.
- La couche navigation reçoit les requêtes des utilisateurs internes à travers le RPVJ.

Le système site central inclus aussi une chaîne de numérisation / GED avec des éléments déployés dans les bureaux fonciers. L'application site central interagit avec la GED.

La colonne de gauche décrit le SSEE qui ne possède que des serveurs logiques « front ». Le SSEE a la même base de données que le SC et les serveurs logiques « back » du SC peuvent être réutilisés (il est tout de même possible de mettre en place des serveurs back dédiés au SSEE). La connexion entre le front SSEE et le back passe par un connecteur SSEE qui ne propose que les services nécessaires au SSEE. Le connecteur SSEE s'appuie sur la technologie JMS, dont l'infrastructure garantit que l'initiative des appels réseaux se fait par le back vers le front SSEE conformément aux exigences de sécurité.

2.1.3. Vue des principaux composants par couches de l'architecture

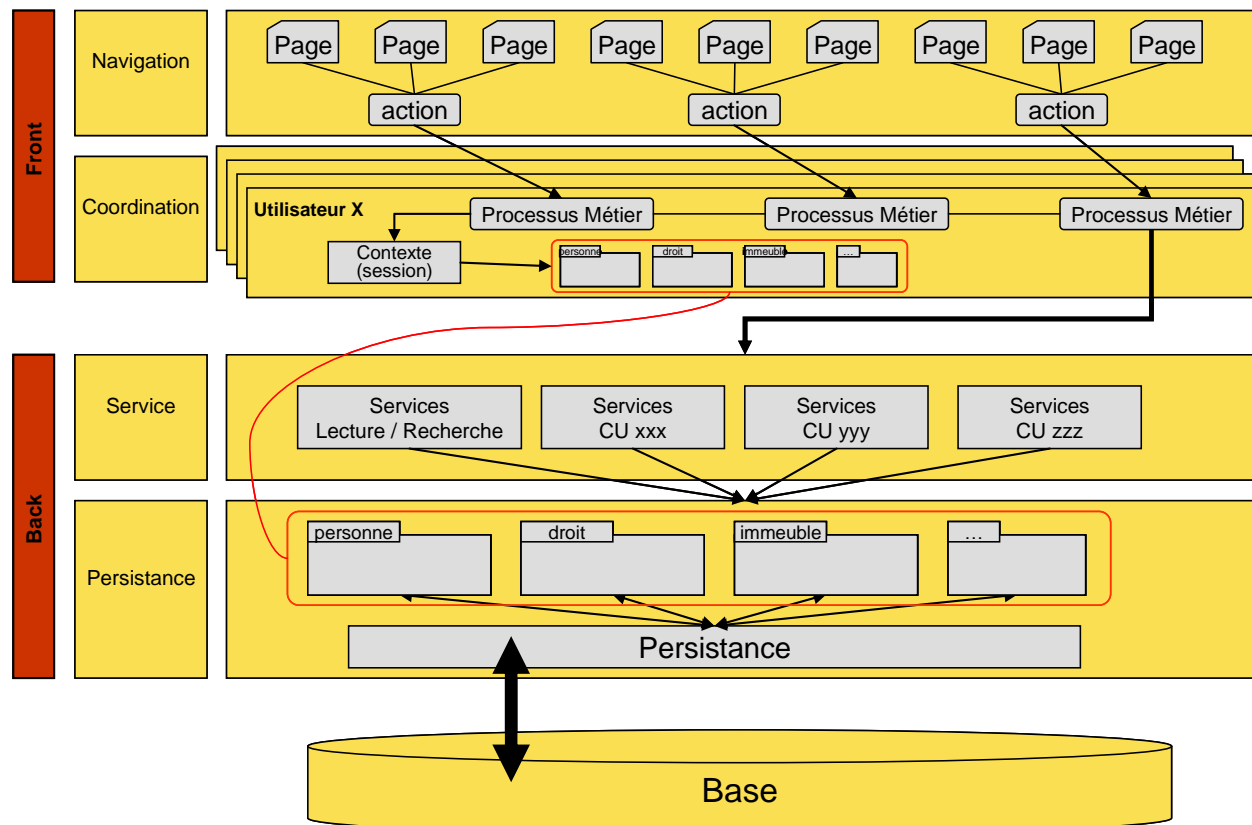


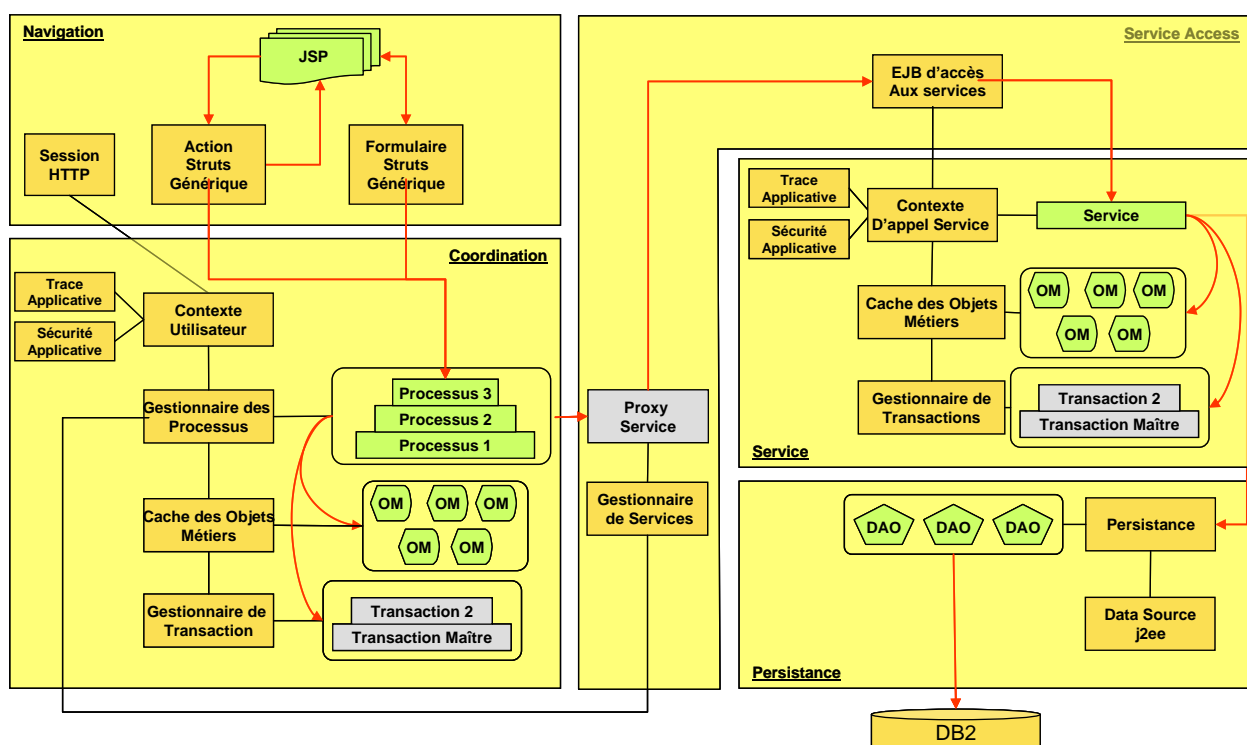
Figure 3 : Vue de l'architecture applicative des principaux composants dans les couches

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			12/7 7

Le schéma ci-dessous montre les principaux composants de chacune des couches.

- ✓ **La couche Navigation** contient les pages (JSP/écrans). Chaque page peut déclencher des actions (Struts) vers le processus qui les gère dans la couche coordination
- ✓ **La couche Coordination** contient les processus métiers et un contexte par utilisateur qui inclut les objets métiers (structure de données) en cours d'utilisation. Les processus métiers reçoivent les actions en provenant des pages et exécutent les traitements demandés en s'appuyant sur des objets métiers les services. A la fin du traitement le processus métier retourne à la l'utilisateur une page. Remarque : un processus métier peut sous-traiter une partie des pages et des traitements à un autre processus. Les processus sont le lieu privilégié pour l'implémentation de la logique métier (cas d'utilisation).
- ✓ **La couche Service** contient des services qui se répartissent en 2 catégories: des services de lecture et recherche des objets métiers et des services dédiés aux cas d'utilisation. Comme vu précédemment la couche service ne conserve pas de données utilisateur entre 2 appels. Pour l'accès à la base de données les services utilisent la couche de persistance. La vocation de la couche service est de permettre l'ajout de traitement fonctionnel nécessitant beaucoup d'information de la base de données.
- ✓ **La couche Persistance** contient des composants permettant de lire, insérer, modifier, chercher et supprimer des objets métier (ou objets du domaine) dans la base de données. Remarque : les objets métiers sont transverses, ils sont utilisés par la plupart des couches.

2.1.4. Vue détaillée des composants par couches de l'architecture



Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			13/7 7

Figure 4 : Vue détaillée de l'architecture applicative des composants dans les couches

Le schéma ci-dessus présente de manière plus détaillée la décomposition en composants des couches de l'architecture. Les composants verts (JSP, Processus, Objet Métier, Service et DAO) sont directement dédiés à l'implémentation des cas d'utilisation : ils peuvent être spécifiques à un cas d'utilisation. Les autres composants font partie de l'architecture applicative et sont implémentés une fois pour toutes.

✓ **La couche Navigation** contient les éléments J2EE dédiés à l'interaction avec les utilisateurs :

- la session http : permet de stocker des informations importantes entre deux requêtes. C'est en fait le point d'accès au contexte utilisateur (le contexte est un objet dans la session).
- l'action Struts : est prise en charge par une classe Java générique dans cette architecture. Toutes les requêtes sont routées vers cette classe action, qui trouve dans la requête les informations nécessaires pour activer le processus qui doit la traiter. La classe action est générique, mais il y a une action (objet) par processus, en effet les actions Struts sont nommées et déclarées dans un fichier de configuration : on a ainsi une action par processus métier qui regroupe les « forward » vers les pages prises en charge par le processus. A la fin de son exécution le processus métier retourne un état à l'action Struts, cet état correspond à une JSP que l'action envoie.
- le formulaire Struts : est aussi pris en charge par une classe générique dans cette architecture. Cette classe permet d'accéder au processus en cours d'exécution. La page JSP peut ainsi afficher les éléments proposés par le processus (souvent des objets métiers). Le formulaire correspond aussi à un formulaire HTML qui permet de récupérer les saisies utilisateur dans la requête.
- Les autres composants : la couche navigation contient aussi des composants absents du schéma comme :
 - un convertisseur permettant de passer les données du format saisi par l'utilisateur au format électronique supporté par l'application, et vice versa (exemple : les dates et les numériques).
 - Des taglibs, librairies de tags de type « html » facilitant l'écriture des JSP.
 - Des feuilles de styles (CSS) simplifiant la prise en compte de la charte graphique et de l'accessibilité.

✓ **La couche coordination** contient les éléments dédiés à l'implémentation de la logique métier :

- le contexte utilisateur : c'est un objet Java simple, placé dans la session http de l'utilisateur. Cet objet est le point d'accès unique aux autres composants (ci-dessous) de la couche coordination,
- le gestionnaire de processus : ordonnance les différents processus métier en cours d'exécution. Un processus métier implémente la logique du traitement à

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			14/7 7

effectuer lors des actions utilisateur. Comme vue précédemment un processus métier peut sous-traiter tout ou partie d'un traitement à un autre processus. Le gestionnaire de processus organise la navigation entre processus à l'aide d'une pile. Ainsi, lorsqu'un processus sous-traite un traitement à un autre processus, le processus sous-traitant est ajouté en dessus de pile. Lorsqu'un processus se termine il est supprimé de la pile et le processus précédent reprend la main. Ainsi, un processus sous-traitant peut rendre la main au processus appelant sans le connaître.

- le cache des objets métiers : permet de conserver des objets métiers entre plusieurs requêtes http et ainsi d'éviter des allers-retours entre le Front et le Back. En fait, pour obtenir des objets métiers, les processus s'adressent toujours au cache. Soit le cache contient l'objet et le retourne. Soit il appelle le service de lecture en base puis conserve l'objet ainsi obtenu avant de le retourner au processus. Afin d'éviter une trop grande consommation mémoire par utilisateur, le cache est vidé chaque fois que l'utilisateur revient sur le processus accueil (1^{er} processus de la pile).
 - le gestionnaire de transactions permet de gérer des transactions en mémoire (pas en base) au niveau du processus métier. Un processus métier peut démarrer une transaction, tous les changements effectués dans cette transaction seront journalisés en mémoire par le gestionnaire. Il est possible d'imbriquer les transactions. Ainsi un processus peut :
 - annuler une transaction : tous les objets du cache reviennent dans leur état du début de la transaction
 - valider une transaction : toutes les modifications de la transaction validée sont intégrées dans la transaction englobante. Si il s'agit de la transaction globale (pas de transaction englobante), le service de mise à jour du cas d'utilisation correspondant au processus est appelé avec la liste des objets modifiés pour une mise à jour de la base : c'est à ce moment-là, que commence la véritable transaction base de donnée.
 - La sécurité applicative : permet de veiller à ce que les actions de l'utilisateur soient conformes à ces droits. Elle est appelée à chaque démarrage de processus et à chaque action de l'utilisateur et elle possède un droit de veto.
 - La trace applicative : permet d'alimenter les journaux (fichiers de log) de l'application à des fins de suivis de l'activité applicative. Elle est donc appelée pour le démarrage des processus et des actions utilisateurs, mais aussi lors des erreurs techniques applicatives (erreurs techniques inattendues, hors fautes de saisies utilisateurs, prévues par les règles de gestion).
- ✓ **Accès aux services** se retrouve entre la couche Coordination et la couche Service. Il permet la prise en charge du middleware (franchissement du réseau) de manière transparente. Il contient :
- un gestionnaire de services : lorsqu'un processus veut appeler un service, il s'adresse à ce gestionnaire en lui disant quel service il veut. Le gestionnaire donne alors au processus un représentant (proxy) du service. Du point de vue du processus ce proxy est le service lui-même. Mais le proxy n'est qu'un représentant, il possède la même interface (API) que le service souhaité, mais

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			15/7 7

lorsque on l'appelle, il prend en charge la traversée du réseau et déclenche le véritable service situé dans la partie back.

- Les proxys de services : comme vue précédemment ce sont des représentants des services qui permettent l'activation du service dans le back après franchissement du réseau. Leur objectif est de masquer la technologie employée pour traverser le réseau. Ainsi, il y aura plusieurs modes de communication avec le back : Pour l'application site central, l'architecture utilise un EJB (Session Stateless) pour accéder aux services du back.
- ✓ **La couche Service** reprend pratiquement les mêmes composants que la couche coordination. Les processus sont remplacés par les services. Chacun des composants réutilisés jouent le même rôle et répond aux mêmes besoins. Les différences principales sont :
 - la non conservation d'information entre 2 appels de services (Stateless),
 - comme on ne conserve pas de contexte entre 2 appels, il n'est pas nécessaire de mettre en place un mécanisme de pile pour les services.
 - Le cache d'objet métier s'adresse directement à la persistance pour lire ou mettre à jour des objets en base de données.
- ✓ **La couche Persistance** contient tout le code nécessaire à la lecture et la mise à jour de la base de données. Ce code se trouve essentiellement dans 2 composants :
 - les DAO (Data Access Object) : il y a un DAO par classe métier (Droit, Copropriété ...), il contient le code Java / SQL nécessaire à la lecture et mise à jour de la base pour l'objet en question.
 - Un moteur de persistance qui est en fait un Ordonnanceur, qui appelle successivement les DAO des objets à mettre à jour dans la base de données.

Tous ces composants sont définis de manière plus précise dans les parties suivantes.

2.1.5. Principe du modèle de base des objets métiers (versions)

Ce chapitre présente les grands choix de structurations de la base de données.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			16/7 7

2.1.5.1. Les domaines de données

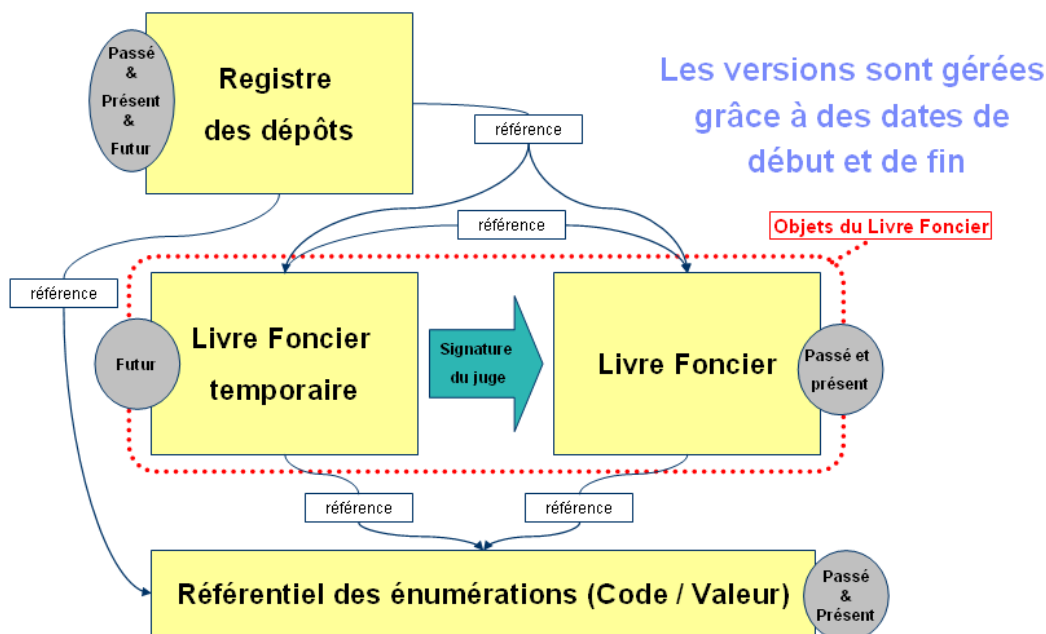


Figure 5 : Les grands domaines du modèle de données

Ce schéma montre trois grandes familles d'objets métiers :

- ✓ Le domaine des objets du registre des dépôts qui représentent les requêtes, aussi bien dans le passé (requêtes déjà publiées), dans le présent (requêtes à publier) que dans le futur (requêtes à publier, mais nécessitant la publication d'autres requêtes au préalable). Ces objets référencent les objets du Livre Foncier (temporaire ou publié) et des énumérations
- ✓ Le domaine des objets du livre foncier est divisé en deux sous domaines:
 - les objets du livre foncier (publiés) : ils représentent le présent et le passé. Ce sont les informations en cours de validité ou qui furent valide par le passé suite à des publications de requête.
 - Les objets du livre foncier temporaire, ils représentent les informations telles que le souhaiterais une requête qui n'a pas encore été publiée. Ils représentent donc le futur (probable).

Les objets passent du sous domaine temporaire au sous domaine livre foncier grâce à la signature du juge.

- ✓ Le domaine Référentiel des énumérations : il s'agit d'informations assez stables, comme les circonscriptions foncières, les bureaux fonciers Ces objets représentent le passé et le présent, en effet certaines valeurs peuvent être modifiées (modification des labels), la version précédente est alors conservée. D'autres valeurs peuvent être radiées (suppression d'un bureau foncier par exemple), la valeur est alors inutilisable par les nouveaux objets mais reste utilisée par les anciens objets.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			17/7 7

2.1.5.2. La gestion des identifiants et des versions

Les identifiants du livre foncier : appelons Entité l'ensemble des versions d'objets successifs. Le Numéro Amalfi est un identifiant métier pour les entités du livre Foncier. Nous appelons Clé Amalfi l'identifiant d'une version d'objet dans le livre Foncier: Elle est composée d'un numéro Amalfi et de la version. Afin de simplifier le numérotage en base de données la version est sous une forme de date : « dtDebut ». De plus pour faciliter le chaînage des versions, on ajoute une date de fin qui reprend le « dtDebut » (version) de l'objet précédent.

Relations dans le livre Foncier : dans le livre Foncier (temporaire ou publié) la plupart des relations ne dépendent pas de la version, en fait les liens se font au niveau Entité. Les objets du registre des dépôts sont sans version. Les relations du Registre des Dépôts vers le livre Foncier (lien Requête / Objet Foncier) se font souvent au niveau des objets (incluant la version). Entre n'importe quel objet (Livre ou Registre) les liens vers le référentiel (énumération) se font sans version (seulement avec le code de l'énuméré).

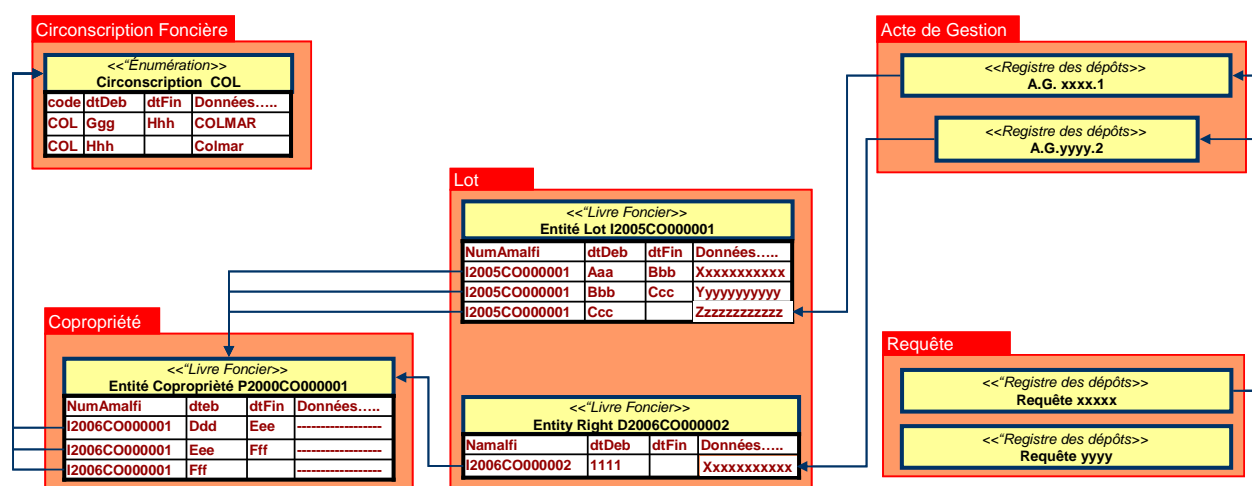


Figure 6 : La gestion des versions par l'exemple

Ces concepts sont vus en détails dans la partie sur la couche Persistance plus loin dans ce document.

2.1.6. Accès aux données et sécurité

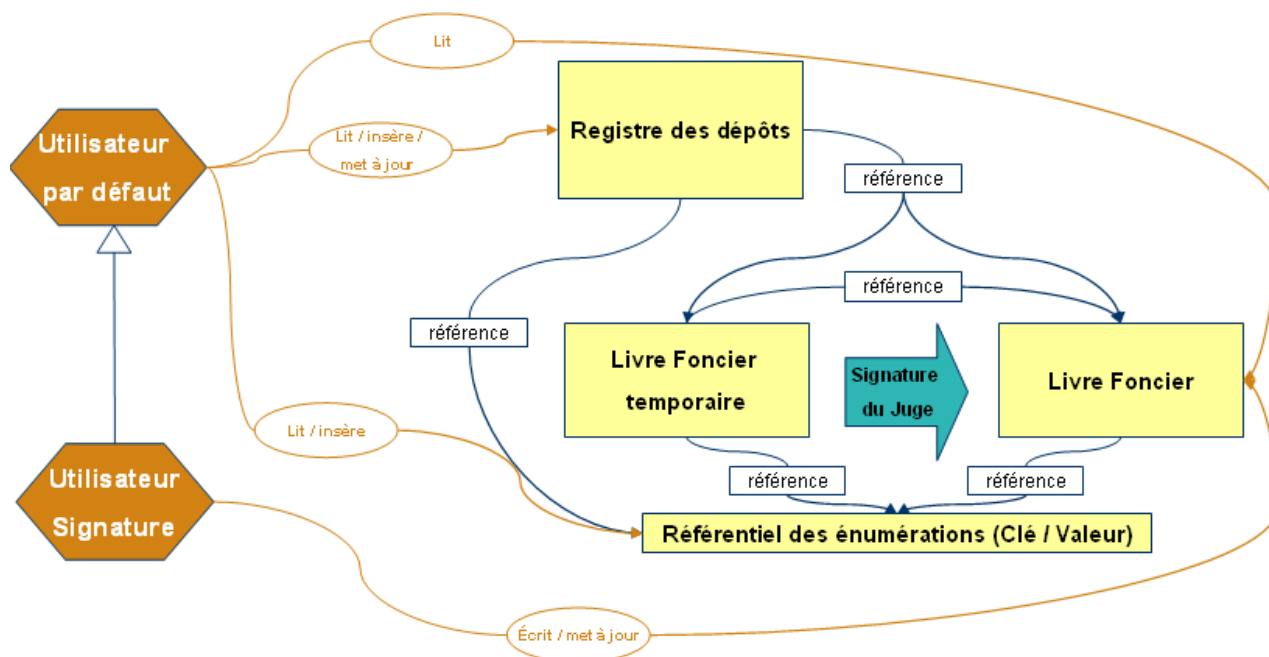


Figure 7 : Accès aux données et sécurité

Il existe deux types d'utilisateurs (techniques) de l'application Amalfi v2 vus de la base de données :

- ✓ L'utilisateur par défaut, qui a uniquement des droits en lecture sur le livre foncier, des droits en lecture, mise à jour et insertion sur le registre des dépôts (il peut créer des requêtes) et des droits en lecture et mise à jour sur le référentiel des énumérations.
- ✓ L'utilisateur « signature », qui a les mêmes droits que l'utilisateur par défaut, plus des droits étendus sur le livre foncier puisqu'il peut y écrire ou mettre à jour des données (cet utilisateur technique ne sera utilisé que par les processus de signature : dédié aux Juges).

2.2. Principes et définitions transverses

Dans cette partie sont abordés les principes de la couche transverse c'est-à-dire les composants pouvant être utilisés par toutes les couches de l'architecture.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			19/7 7

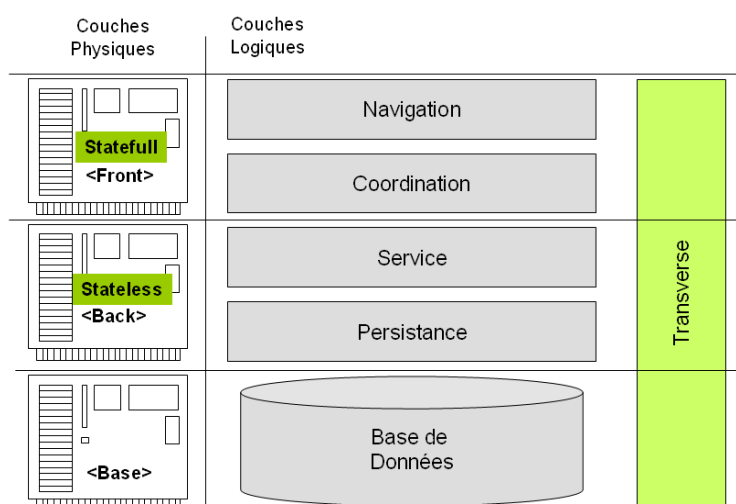


Figure 8 : Principes de la couche transverse

Les composants décrits sont les suivants :

- ✓ La gestion des erreurs (exceptions Java)
- ✓ Les traces applicatives et de débogage
- ✓ L'assemblage et l'initialisation de l'application

2.2.1. La gestion des erreurs (Exceptions)

On distingue deux grandes catégories d'erreurs :

- ✓ Les erreurs attendues : erreurs dues au comportement de l'utilisateur pour lesquelles on soumet un message d'erreur explicite à l'utilisateur pour qu'il change ses saisies ou ses choix et continue sa tâche
- ✓ Les erreurs inattendues : ces erreurs sont dues à des défaillances (soit de l'application : Bug), soit de la plateforme (incident d'exploitation : chute réseau ou base ...), soit une erreur prévue par l'application mais pour laquelle il n'y a pas de solution automatisée (par exemple corruption d'un sceaue)

2.2.1.1. Les erreurs attendues

Elles sont généralement déclenchées lors des contrôles. Elles doivent être gérés par la couche coordination. Pour cela, le contexte utilisateur doit offrir un accès à une liste d'erreurs, que la couche coordination peut enrichir au fur et à mesure des contrôles successifs. En fin de traitement et en cas d'erreurs, la couche navigation retourne la même page à l'utilisateur : cette page sera enrichie de libellés présentant la liste des problèmes détectés, problèmes que l'utilisateur doit résoudre pour continuer son travail.

Dans le cas d'erreurs attendues la couche coordination ne doit pas s'arrêter à la 1^{ère} erreur détectée, elle doit continuer ses contrôles jusqu'au bout si possible afin de signaler toutes les erreurs à l'utilisateur en une fois plutôt qu'au fur et à mesure.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			20/7 7

Ce type d'erreur est géré dans l'architecture applicative grâce à l'exception Java Amalfi2Exception.

2.2.1.2. Les erreurs inattendues

Elles n'ont pas de parades automatiques et ne sont pas liées à la saisie utilisateur, la tâche en cours pour l'utilisateur ne pourra donc pas être terminée. Ces erreurs sont captées par la couche navigation qui présente un écran d'erreur à l'utilisateur avec une description du problème. L'erreur sera aussi inscrite dans un journal applicatif. La procédure de traitement de l'erreur est humaine : l'utilisateur doit appeler son support pour signaler le problème et déclencher une investigation.

Ce type d'erreur est géré dans l'architecture applicative grâce à l'exception Java Amalfi2RuntimeException.

2.2.2. Les traces applicatives et de débogage

Les traces sont divisées en deux groupes de traces :

- ✓ Les traces dites « applicatives » ont pour but d'alimenter un outil de contrôle de l'application et d'alimenter le système de supervision pour gérer l'environnement informatique de l'application
- ✓ Les traces dites « débogue » qui ne concernent que les développeurs

Chaque groupe de traces est paramétré à l'aide d'un fichier de configuration spécifiant les formats des traces et ses flux de sortie.

2.2.3. Les traces applicatives

2.2.3.1. Rapports journaliers

Les informations à tracer ont pour but de contrôler les activités de l'application. Elles ne doivent donc pas excéder une taille raisonnable pour permettre la gestion journalière des informations tracées. Les informations suivantes sont tracées :

- L'appel des services
- Le démarrage d'un processus
- L'exécution d'une action de processus
- L'arrêt d'un processus
- La réactivation d'un processus
- Les données console

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			21/7 7

2.2.3.2. Structure du fichier de traces applicatives

La structure des traces applicatives fait apparaître une partie commune et une partie spécifique à chaque type d'événement tracé. Ce dernier est repéré par un code événement.

2.2.3.2.1. Données communes

La structure de données communes aux traces des différents événements est la suivante :

En-tête, chrono=..., uid=..., evt=..., duree=..., procClick=..., actClick=..., procEnCours=..., actEnCours=..., ...

Le contenu des variables est défini comme suit :

- En-tête : dépendant de la configuration **Log4j** (exemples : nom du thread, nom du serveur, ...),
- chrono: identifiant chronologique et séquentiel (incrémentation par pas de 1) sur un serveur,
- uid : OID de l'utilisateur Amalfi,
↳ Remarque : chrono+uid constitue une clef unique d'événement,
- duree : temps d'exécution de l'événement tracé,
- procClick : processus actif au moment du clic de l'utilisateur,
- actClick : action correspondant au moment du clic de l'utilisateur,
- procEnCours : processus en cours d'exécution au moment de l'événement tracé,
- actEnCours : action en cours d'exécution au moment de l'événement tracé.

2.2.3.2.2. Données spécifiques par événement

2.2.3.2.2.1. L'appel des services

Les appels des services sont repérés par le code événement :

- evt=serviceExecute

Les champs suivants apparaissent également pour ce type d'événement :

- classeService : appel de service du back
- methodeService : méthode appelée dans le service
- param* : description des paramètres d'appel

2.2.3.2.2.2. Le démarrage d'un processus

Le démarrage d'un processus est repéré par le code événement :

- evt=processusDemarre

Les champs suivants apparaissent également pour ce type d'événement :

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			22/7 7

- param* : description des paramètres d'appel

2.2.3.2.2.3.

L'exécution d'une action de

processus

L'exécution d'une action de processus est repérée par le code événement :

- evt=processusExecute

Les champs suivants apparaissent également pour ce type d'événement :

- param* : description des paramètres d'appel

2.2.3.2.2.4.

L'arrêt d'un processus

L'arrêt d'un processus est repéré par le code événement :

- evt=processusArrete

Les champs suivants apparaissent également pour ce type d'événement :

- param* : description des paramètres d'appel

2.2.3.2.2.5.

La réactivation d'un processus

La réactivation d'un processus est repérée par le code événement :

- evt=processusReactive

Les champs suivants apparaissent également pour ce type d'événement :

- param* : description des paramètres d'appel

2.2.3.2.2.6.

Données Console

L'événement de synthèse du clic utilisateur est repéré par le code événement :

- evt=donneesConsole

Le champ suivant apparaît également pour ce type d'événement :

- traceContextuelle

Les informations portées par ce champ sont les suivantes :

- Proc. Action : Processus en cours lors de l'action de l'utilisateur
- Nombre de processus : Nombre de processus dans la pile
- Pile des processus: Pile des processus citée dans l'ordre chronologique des actions utilisateurs Nom de la classe + id unique
- Nombre de transaction: nombre de transactions en cours
- Cache objets métier: NbObjet: nombre d'objets dans le cache de l'utilisateur
- Cache objets métier communs: NbObjet: nombre d'objets dans le cache commun à tous les utilisateurs dans lequel sont présentes les énumérations (exemple : BF, CF, ...)

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			23/7 7

- Nombre de service appelé: nombre de services appelés
- Nombre de erreurs: nombre d'erreurs communiquées à l'utilisateur (RAxxxx, RIxxxx, RCxxxx)
- Codes erreurs: référence des codes erreur communiquées à l'utilisateur (RAxxxx, RIxxxx, RCxxxx)

Remarque : dans ce contexte spécifique, le champ *duree* fournit la durée totale du temps de traitement serveur suite au clic utilisateur.

Exemple :

```
traceContextuelle='Proc. Action =
ProcessusListePersonnesPhysiques-1179993423984-003
DetailObjetLF|Nombre de processus = 3|Pile des processus:|
ProcessusAccueil-1179993373327-001| ProcessusRechercheLF-
1179993382572-002| ProcessusListePersonnesPhysiques-
1179993423984-003|Nombre de transaction 0|Cache objets métier:
NbObjet = 20|Cache objets métier communs: NbObjet =
7605|Nombre de service appelé = 0|Nombre de erreurs = 1|Code
erreurs =RC2000.001'
```

2.2.3.2.3.

Evénements Erreur

2.2.3.2.3.1.

Erreurs attendues

Les erreurs attendues (voir aussi 2.2.1.1) sont tracées dans les événements *donneesConsole* décrits ci-dessus.

Les informations sont fournies par les champs Nombre de erreurs et Code erreurs.

2.2.3.2.3.2.

Erreurs inattendues

Les erreurs inattendues (voir aussi 2.2.1.2) sont tracées dans l'architecture applicative grâce à l'exception Java *Amalfi2RuntimeException*.

2.2.3.2.4.

Mise en œuvre pour la supervision applicative

Le document **CU2932 Supervision applicative** décrit les moyens de suivre le bon fonctionnement des instances applicatives d'AMALFI en direct par le biais d'une console de supervision applicative qui exploite un ensemble d'indicateurs stockés en base de donnée.

Ce sont les événements *donneesConsole* et *erreurs inattendues* de la trace applicative qui fournissent l'intégralité des informations nécessaires à la constitution de ces indicateurs de supervision. En effet, en parallèle de leur écriture dans chacun des journaux

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			24/7 7

applicatifs (un par ear), les événements sont également transmis au moniteur de supervision qui se charge de leur inscription en base de données.

2.2.3.3. Trace du système de supervision

Pour gérer l'environnement informatique de l'application à l'aide des fonctions de diagnostic de TEC, les informations importantes sont tracées et sont utilisées par le système de supervision.

2.2.4. Les traces de déboguage

Les traces de déboguages peuvent être désactivées lors de l'exploitation. En effet, les traces de déboguage peuvent représenter un volume très important dans un environnement de production et pénaliser fortement les performances.

2.2.4.1. Traces techniques

Le développeur peut tracer une quelconque chaîne de caractères selon ses besoins. Il a le droit d'utiliser le niveau DEBUG de l'API de la librairie de trace choisie. Ces traces n'ont un sens que pour le développeur.

2.2.4.2. Traces des erreurs (Exceptions)

L'émission d'exceptions doit être tracée. L'exception émise est passée en paramètre à la fonction de trace des erreurs à l'endroit de sa capture : au plus haut de l'architecture, dans la couche navigation.

2.2.5. Les traces d'audit

Les traces d'audit sont décrites dans le document SECU_DET_LOG "Journaux et traces AMALFI" au chapitre "Traces d'audits".

2.2.6. La console de surveillance applicative

Les traces applicatives ainsi que les erreurs inattendues seront monitorées par une console de surveillance applicative que sera décrite dans un CU. Ce chapitre pose les principes de cette console.

Chaque élément Front ou Back, des applications SC ou SSEE, peut émettre des traces applicatives. Ces traces sont écrites dans un fichier local à la machine hébergeant le composant mais sont aussi envoyé à un service de collecte d'événement.

Il y aura un service de collecte des événements par instance applicative. Ces services utilisent le mode de communication de l'instance applicative pour lesquels ils enregistrent les événements.

Récapitulatif :

Le Front SC enverra ses événements au service de collecte des événements du Back SC. La communication entre le Front et le Back SC sera réalisée à travers un appel d'EJB.

Le Back SC enverra ses événements au service de collecte des événements du Back SC.

Le Front SSEE enverra ses événements au service de collecte des événements du Back SSEE. La communication entre le Front et le Back SSEE sera réalisée à travers un appel de MDB.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			25/7 7

Le Back SSEE enverra ses événements au service de collecte des événements du Back SSEE.

Les services de collecte des événements utilisent des tables de la base AMALFI, dans laquelle ils stockent les événements et des agrégats statistiques élémentaires. Il y a une seule base d'événements : une colonne de la table des événements identifie l'instance applicative ayant envoyé l'événement.

Un utilisateur habilité peut utiliser le module de surveillance, qui permet de visualiser les agrégats statistiques à chaud et de télécharger certains rapports. Il y a un seul module de surveillance, qui se trouve dans le Front SC.

Le schéma ci-dessous propose une synthèse.

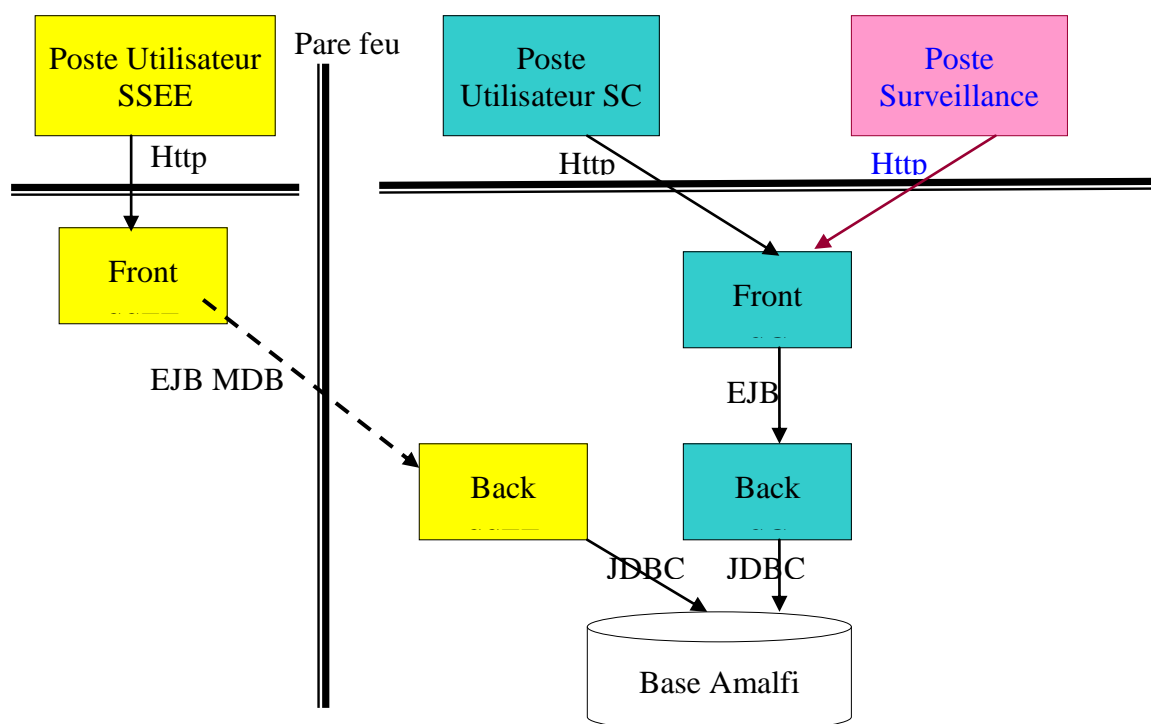


Figure 9 : Architecture logique de la console de surveillance.

2.2.7. Le composant d'assemblage et d'initialisation de l'application

Ce composant adresse la problématique de la mise en place du contexte d'exécution :

- ✓ Dans le cadre des tests unitaires on a un contexte simplifié (front et back dans un serveur)
- ✓ Dans le cadre de la pré-production et production le contexte d'exécution inclut plusieurs serveurs

Ce composant n'est pas utilisé par les composants des autres couches, mais il les initialise et les utilise, c'est pourquoi il se retrouve dans la couche transverse.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			26/7 7

Ce composant est un système sophistiqué qui permet de configurer les tests unitaires simplement en utilisant une ou plusieurs couches de l'architecture applicative. Le composant d'initialisation permet différents modes d'assemblages et d'initialisation de l'application. Les différents modes sont les suivants, ils permettent l'assemblage et l'initialisation :

- ✓ du front avec communication RMI entre front et back (mode utilisé en production pour la communication entre le front SC et le back)
- ✓ du front avec communication JMS entre front et back (mode utilisé en production pour la communication entre le front SSEE et le back)
- ✓ du back (mode utilisé en production pour le back)
- ✓ de l'application totale (front + back) pour les tests unitaires JUnit (mode utilisé en phase de développement)
- ✓ de l'application totale (front + bac) pour les tests de l'application dans un conteneur léger de type Jetty, ... (mode utilisé en phase de développement)

L'initialisation de l'application se fait grâce à un gestionnaire d'initialisation à qui on donne le mode d'assemblage voulu. En fonction de ce mode le gestionnaire d'initialisation va initialiser chaque couche en démarrant le gestionnaire de la couche à initialiser. Chaque gestionnaire de couche gère l'initialisation et le cycle de vie de chaque composant de sa couche.

Ce composant rend l'assemblage et l'initialisation de l'application très souple en fonction de la phase du projet dans laquelle on se trouve. De même il permet un développement et une exécution aisés des tests unitaires tout au long de la phase de développement de l'application.

2.2.8. La sécurité applicative

Le composant de sécurité applicative est un composant transverse :

- Il est initialisé dans le front lors de l'ouverture de la session de l'utilisateur.
- Il est stocké dans la session de l'utilisateur.
- Il est transmis au back lors de l'appel de service.

Le composant de sécurité applicative permet de vérifier que les actions de l'utilisateur sont conformes aux droits de son profil. Les points de vérification sont les suivants :

✓ Dans le Front :

- ✓ Lorsqu'un processus est démarré, la sécurité applicative vérifie les droits d'accès aux processus.
- ✓ Lorsqu'une action (Action Struts) est déclenchée, la sécurité applicative vérifie les droits d'accès à l'action du processus (l'action correspond à une méthode du processus).

✓ Dans le Back :

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			27/7 7

- ✓ Lorsqu'un processus appelle un service, la sécurité applicative vérifie les droits d'accès à ce service.

En fait le composant de sécurité applicative est un point de redirection vers le module fonctionnel qui implémentera les vérifications telles que spécifiées dans les documents de conceptions détaillées.

Les choix fonctionnels sont d'attribuer des profils à un utilisateur et de donner des droits d'accès à des CU (Cas d'Utilisation) pour chaque profil.

- Dans le cadre de sa session, l'utilisateur n'a qu'un profil.
- Chaque processus est associé à un et un seul CU.
- Chaque service est associé à un et un seul CU.

La vérification de droits d'accès à un processus consiste à vérifier si le CU du processus est autorisé pour le profil de l'utilisateur.

La vérification de droits d'accès à un service consiste à vérifier si le CU du service est autorisé pour le profil de l'utilisateur.

Aucune vérification de niveau action processus n'est demandée par la spécification.

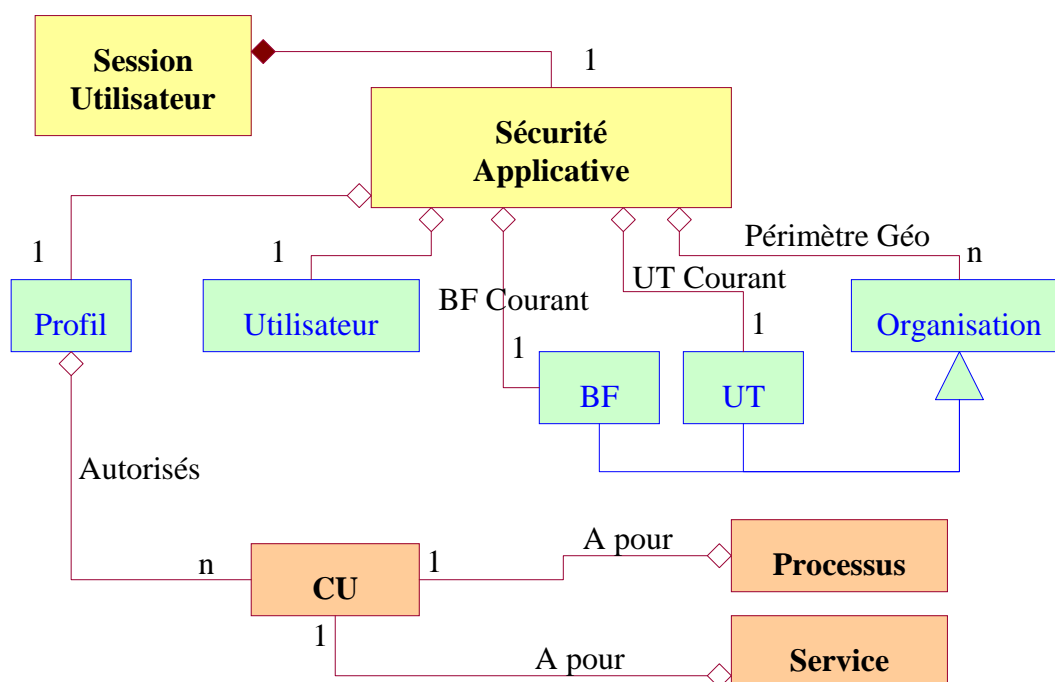


Figure 10 : Principes de la sécurité applicative (Structure de données)

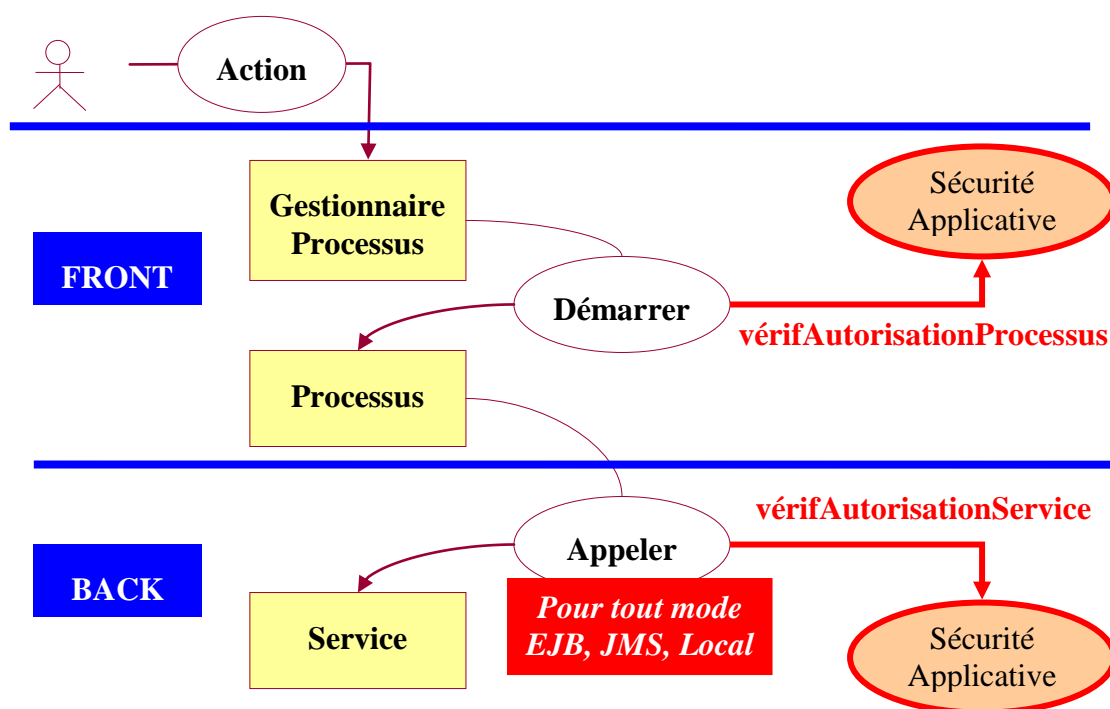


Figure 11 : Principes de la sécurité applicative (Dynamique)

2.3. Principes et définitions du Front

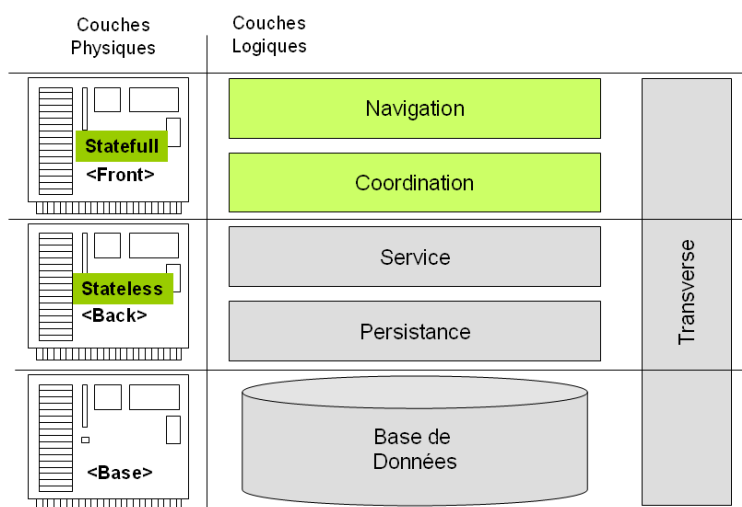


Figure 12 : Principes du Front

Cette partie présente les principes des couches Navigation et Coordination de la zone Front.

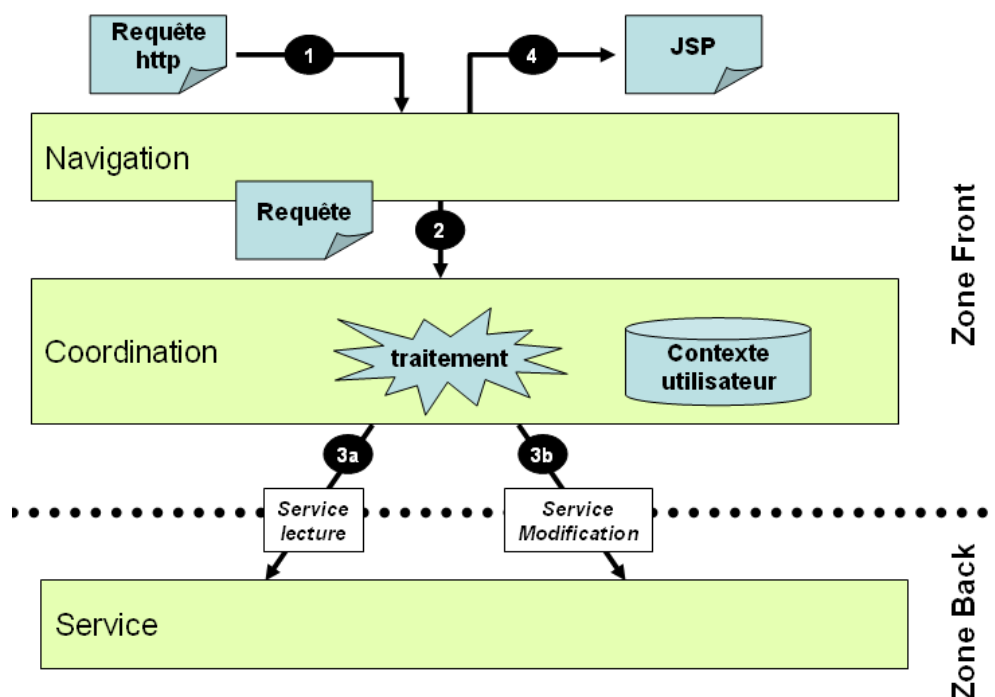


Figure 13 : Vue générale des couches Navigation et Coordination

La couche navigation a pour objectifs principaux :

- ✓ la réception des requêtes http en provenance du navigateur de l'utilisateur (1)
- ✓ le transfert de la requête utilisateur à la couche coordination (2)
- ✓ l'envoi des pages JSP correspondant à la réponse du traitement (4)

La couche coordination a pour objectif principal :

- ✓ le traitement de la requête utilisateur (3a/3b). Pour cela elle contient un contexte des informations en cours d'utilisation dans la session de l'utilisateur, et elle a la possibilité d'appeler la couche service (zone back), pour obtenir des données (lectures) ou faire des créations ou des mises à jours de la base de données

D'un point de vue technique la couche navigation encapsule tous les aspects J2EE, alors que la couche coordination est constituée de java simple (JSDK).

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			30/7 7

2.3.1. Principes et définitions de la couche navigation

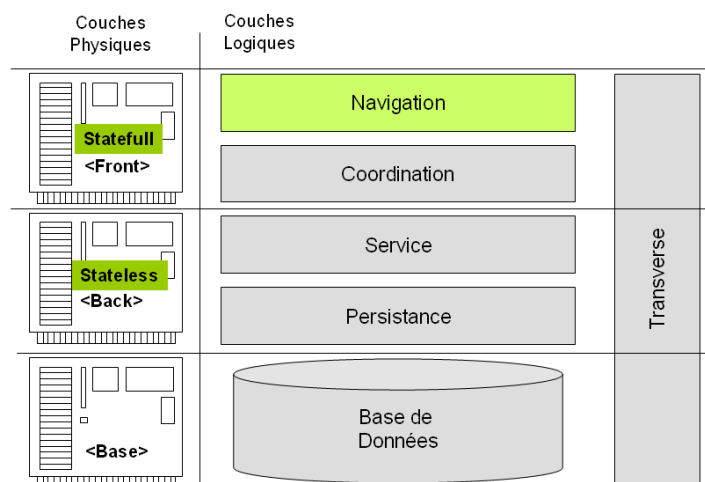


Figure 14 : Principes de la couche navigation

Rappel : la couche navigation est en charge :

- ✓ De la réception des requêtes http de l'utilisateur
- ✓ Du déclenchement du traitement correspondant à la requête (le traitement est fourni par la couche coordination)
- ✓ De l'envoi de la page correspondant à la réponse vers l'utilisateur

2.3.1.1. Struts

La couche navigation s'appuie sur le Framework Struts pour mettre en œuvre les éléments J2EE dans un cadre MVC.

2.3.1.2. Classes Action et Formulaire Struts

Afin d'améliorer la lisibilité et la quantité de code à produire, Struts est mis en œuvre à travers une classe action Struts générique et une classe formulaire Struts générique : c'est-à-dire que en dehors des cas particuliers, tous les enchaînements s'appuient sur la même classe action et les JSP utilisent la même classe formulaire.

2.3.1.3. Une classe Action Struts mais plusieurs actions Struts

Bien qu'il y ait une seule classe action Struts, le fichier de configuration de Struts contient plusieurs actions : ce sont des actions avec des noms différents, mais utilisant la même classe.

2.3.1.4. Les processus métiers (vus de la navigation)

Les écrans (JSP) sont réunis en groupes logiques du point de vue fonctionnel. Le niveau de regroupement peut être le CU (Cas d'Utilisation) ou un regroupement plus fin que le CU. Ce regroupement correspond à une liste de traitements et de JSP. Comme convenu, les traitements seront implémentés dans la couche coordination. Il y aura une classe dite **Processus Métier** dans la couche coordination pour implémenter les traitements regroupés.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			31/7 7

Un processus métier représente donc l'ensemble des traitements correspondant à un groupe d'écrans.

2.3.1.5. Le couple Action Struts / Processus Métier

Comme vu précédemment, il y a une unique classe d'action Struts mais il y a plusieurs actions Struts (avec des noms différents) utilisant cette classe Action unique. En fait, on a un couple action Struts / Processus métier, pour chaque processus métier on a une action Struts nommée qui regroupe (dans ses « forward ») les écrans utilisés par le processus métier.

2.3.1.6. La session utilisateur (vue de la navigation)

Tout utilisateur utilise l'application dans le cadre d'une session. Cette session est une classe java de la couche coordination placée dans la session http. Cette session utilisateur n'est créée que si l'utilisateur est authentifié. La session utilisateur contient toutes les informations du contexte de travail de l'utilisateur : un cache de composants métiers (données en cours d'utilisation), un accès au processus métier en cours d'exécution.

L'initialisation de la session utilisateur déclenche le lancement d'un processus métier par défaut : en général ce processus est en charge de la page d'accueil.

2.3.1.7. De l'action Struts vers le Processus Métier

L'action Struts générique se contente de transmettre au processus métier en cours d'exécution les paramètres d'appel. Ces paramètres comprennent toujours un paramètre « action Amalfi » désignant le traitement attendu. Cette étape permet de transformer la requête http (paramètres d'appel) en simple dictionnaire (Map) afin de garantir l'indépendance de la couche coordination vis-à-vis de J2EE.

2.3.1.8. La clé de la navigation est l'état du Processus métier

Le processus métier appelé par l'action Struts exécute le traitement correspondant au paramètre « action Amalfi ». Les processus métiers ont un état qui est mis à jour lors de l'exécution des traitements. En fait l'état du processus est l'information qui permet à l'action de choisir le « forward » vers la page réponse.

Remarque :

A des fins d'audit et de débogage l'« action Amalfi » est rajoutée dans l'URL sous forme d'un paramètre préfixé par « __ ». Ainsi pour « l'action Amalfi » « signer » sur le ProcessusDecisionDuJuge, l'URL est « ProcessusDecisionDuJuge.do ?__signer ».

Ce paramètre n'a pas vocation à être utilisé dans l'application. Son seul rôle est de permettre d'analyser plus efficacement les fichiers journaux des reverse proxy frontaux.

2.3.1.9. Un mécanisme de pile pour les processus métiers

Un processus métier peut sous-traiter une partie des traitements à un autre processus métier. Par exemple, le processus courant (X) démarre un nouveau processus (Y) qui devient le nouveau processus courant, c'est donc son état (de Y) qui définira la navigation et c'est lui (Y) qui recevra les actions. En fait les processus métiers en cours s'empilent, le processus courant est celui au-dessus de la pile. Lorsque le processus courant se termine, il est retiré de la pile, le nouveau processus en dessus de pile devient le processus courant.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			32/7 7

2.3.1.10. Enchaînement d'écrans

Comme vu précédemment, l'action Struts appelle le processus métier courant qui exécute le traitement attendu. Ce traitement peut entraîner le lancement d'un nouveau processus. A la fin du traitement, l'action Struts reprend la main et 2 cas se présentent :

1. Le processus courant est toujours le même et l'action Struts en cours déclenche le « forward » vers la page correspondant à l'état du processus courant.
2. Le processus courant n'est plus le même (l'ancien processus courant s'est terminé ou a lancé un nouveau processus), dans ce cas l'action en cours « forward » l'action correspondante au nouveau processus courant ; l'action ainsi activée s'occupera de faire le « forward » de la page correspondante à l'état de ce nouveau processus courant.

2.3.1.11. Le formulaire Struts

Comme vu précédemment, il y a une unique classe de Formulaire, qui est associée à la classe Action unique. Cette classe formulaire Struts contient un attribut qui est une référence sur le processus courant.

Ainsi grâce à la notation pointée permise par les taglibs Struts, il est possible depuis une JSP d'accéder au processus en cours via le formulaire.

Le processus peut avoir des accesseurs sur les différentes données métiers manipulées : par exemple la personne, le droit, l'immeuble, ... La page JSP peut ainsi accéder aux informations du processus, par exemple : processus.personne.nom permet d'accéder au nom de la personne du processus (si le processus à des accesseurs sur personne).

2.3.1.12. Construction du formulaire HTML dans la JSP

Comme décrit précédemment, la JSP peut accéder aux informations du processus grâce au formulaire. Il reste à traiter la prise en charge de la mise en forme des données (date, entier, etc.). Le tag texte de Struts est enrichi de manière à appeler le convertisseur de l'architecture applicative. Ce convertisseur est lui-même configuré pour prendre en charge les différents formats (date ...).

Les données affichées dans les pages suivent donc le formatage voulu de manière transparente.

2.3.1.13. Prise en compte du formulaire reçu dans la requête.

Lorsqu'un formulaire est soumis via la requête http, il faut le prendre en compte en mettant à jour les objets correspondants.

Pour mémoire, comme le formulaire pointe sur le processus en cours, les objets à mettre à jour sont ceux du processus.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			33/7 7

Afin d'éviter une prise en compte manuelle et fastidieuse du formulaire, le Request Processor de Struts est enrichi pour mettre à jour automatiquement les données du processus courant avec les données reçues grâce au formulaire, charge au processus de vérifier a posteriori si les affectations faites sont acceptables. Le Request Processor ainsi enrichi s'appuie aussi sur le convertisseur de l'architecture applicative pour affecter les dates à partir de données au format texte.

Cependant, afin d'éviter des affectations intempestives (par exemple suite à une utilisation d'un bouton annuler : pas d'affectation), le processus est appelé avant la mise à jour automatique, afin qu'il puisse vérifier les données et émettre un éventuel veto.

2.3.1.14. Contrôle de la navigation hors actions applicatives (boutons navigateur)

Les navigateurs Web proposent des boutons de navigation en arrière puis en avant (si l'on a fait marche arrière). Ces boutons peuvent être supprimés mais il existe toujours des menus contextuels ou des raccourcis permettant à l'utilisateur d'exécuter ces actions.

Ce type d'action est très perturbateur pour l'application puisqu'il s'agit d'actions qui ne sont pas proposées par l'application et qui la plupart du temps se déroulent de façon autonome à l'intérieur du navigateur.

Le principe mis en œuvre pour contrôler cette navigation est le suivant :

- ✓ Toutes les pages proposées par l'application auront un timestamp. Dans son contexte (http session) l'application mémorise le dernier timestamp envoyé à l'utilisateur
- ✓ Toutes les requêtes de l'application envoient le timestamp de la page qui propose le lien
- ✓ Ainsi l'application sait si les requêtes qu'elle reçoit, proviennent de la dernière page proposée
- ✓ Si une requête reçue ne provient pas de la dernière page transmise par le serveur (le timestamp reçu par le serveur n'est pas le dernier timestamp envoyé par celui-ci), l'application retourne la dernière page envoyée par le serveur avec son timestamp pour resynchroniser l'utilisateur avec la situation attendue par l'application.

En procédant ainsi, on permet à l'utilisateur de naviguer avec les boutons du navigateur, mais s'il déclenche une action qui n'est pas dans la dernière page envoyée, l'application ne traite pas la requête et resynchronise l'utilisateur. Pour garantir que les requêtes émises par les pages contiendront bien le timestamp, les tags de liens et de « submit » de formulaires seront surchargés.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			34/7 7

2.3.1.15. Les exceptions (inattendues) vues par l'action Struts

Dans un cas normal (sans erreurs inattendues), l'action générique n'est pas interrompue par des exceptions. Par conséquent, si une exception arrive à l'action générique, c'est que des problèmes indépendants des actions utilisateurs sont survenus.

Dans ce cas, l'exception sera tracée dans le journal de l'application et un écran d'erreur fatale sera envoyé à l'utilisateur. Plus de détails sont fournies dans le paragraphe de gestion des erreurs.

2.3.1.16. Synthèse pour la couche navigation

2.3.1.16.1.

Les transitions Page - Action -

Processus

Le schéma ci-après propose une synthèse de la navigation entre 2 processus métiers et plusieurs pages. Le processus X contenant les traitements Xa, Xb, Xc, est associé à l'action X qui permet de naviguer entre les pages X1, X2, X3. Le processus Y plus simple contenant un seul traitement Ya, est associé à l'action Y qui permet d'accéder la page Y1. Les flèches numérotées représentent une séquence d'exécution durant laquelle le processus X fait un appel de sous-traitance au processus Y.

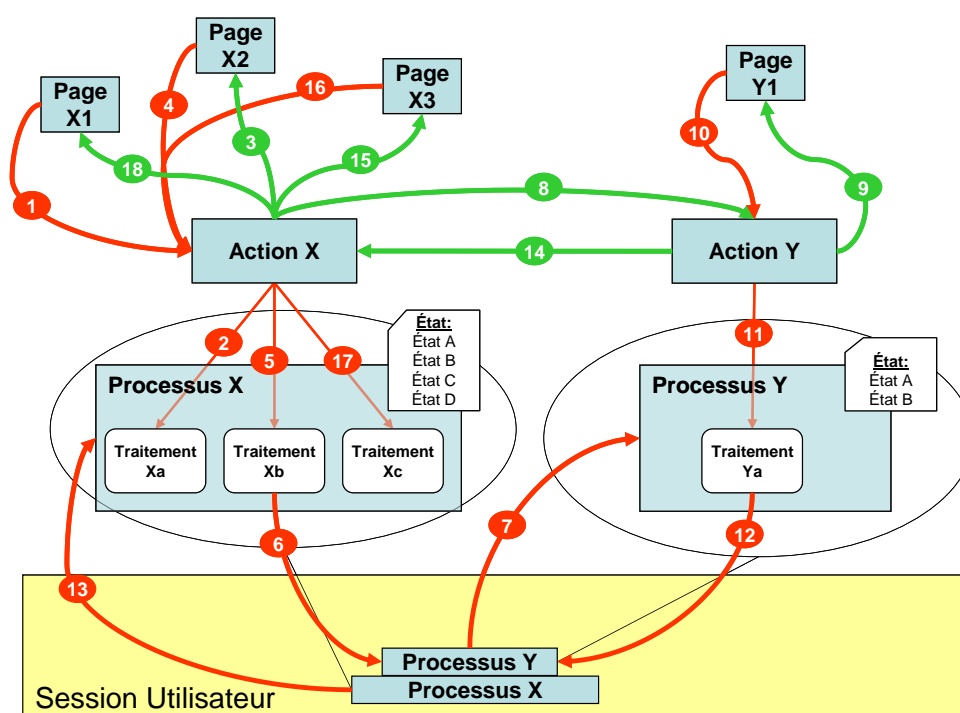


Figure 15 : Exemple de transition Page - Action - Processus

1. Un clic sur la page X1 déclenche un appel à l'action Struts X.
2. L'action X transforme les paramètres d'appel en Map (non J2EE) et transmet l'appel au processus X qui exécute le traitement Xa.
3. Après l'exécution du traitement, l'action X voit que le processus en cours est toujours le processus X, elle lit donc l'état du processus X et déduit qu'il faut faire un forward vers la page X2.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			35/7 7

4. Un clic sur la page X2 déclenche un appel à l'action Struts X.
5. L'action X transforme les paramètres d'appel en Map (non J2EE) et transmet l'appel au processus X qui exécute le traitement Xb.
6. Le traitement Xb déclenche une sous-traitance vers le processus Y. Le processus Y est donc créé et mis en dessus de pile.
7. Le processus Y qui vient d'être démarré est initialisé, ce qui lui permet de mettre à jour son état.
8. Après l'exécution du traitement l'action X voit que le processus en cours n'est plus le processus X mais le processus Y, elle déduit qu'il faut faire un forward vers l'action Y.
9. L'action Y voit que le processus en cours est le processus Y, elle lit donc l'état du processus Y et déduit qu'il faut faire un forward vers la page Y1, qui est ensuite transmise au navigateur.
10. Un clic sur la page Y1 déclenche un appel à l'action Struts Y.
11. L'action Y transforme les paramètres d'appel en Map (non J2EE) et transmet l'appel au processus Y qui exécute le traitement Ya.
12. Le traitement Ya décide que le processus Y est terminé, le processus Y est donc retiré de la pile des processus.
13. Le processus X redevient le processus courant, il est donc réactivé, ce qui lui permet de mettre à jour son état.
14. Après l'exécution du traitement l'action Y voit que le processus en cours n'est plus le processus Y mais le processus X, elle déduit qu'il faut faire un forward vers l'action X.
15. L'action X voit que le processus en cours est le processus X, elle lit donc l'état du processus X et déduit qu'il faut faire un forward vers la page X3, qui est transmise au navigateur.
16. Un clic sur la page X3 déclenche un appel à l'action Struts X.
17. L'action X transforme les paramètres d'appel en Map (non J2EE) et transmet l'appel au processus X qui exécute le traitement Xc.
18. Après l'exécution du traitement l'action X voit que le processus en cours est toujours le processus X, elle lit donc l'état du processus X et déduit qu'il faut faire un forward vers la page X1.

..... et ainsi de suite.....

2.3.1.16.2.

Les interactions Page - Formulaire

Données du Processus

Le schéma ci-après montre :

- ✓ Comment la page JSP accède les informations via le formulaire pour construire les formulaires HTML

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			36/7 7

- ✓ Comment suite à un « submit » de formulaire les données saisies sont prises en compte

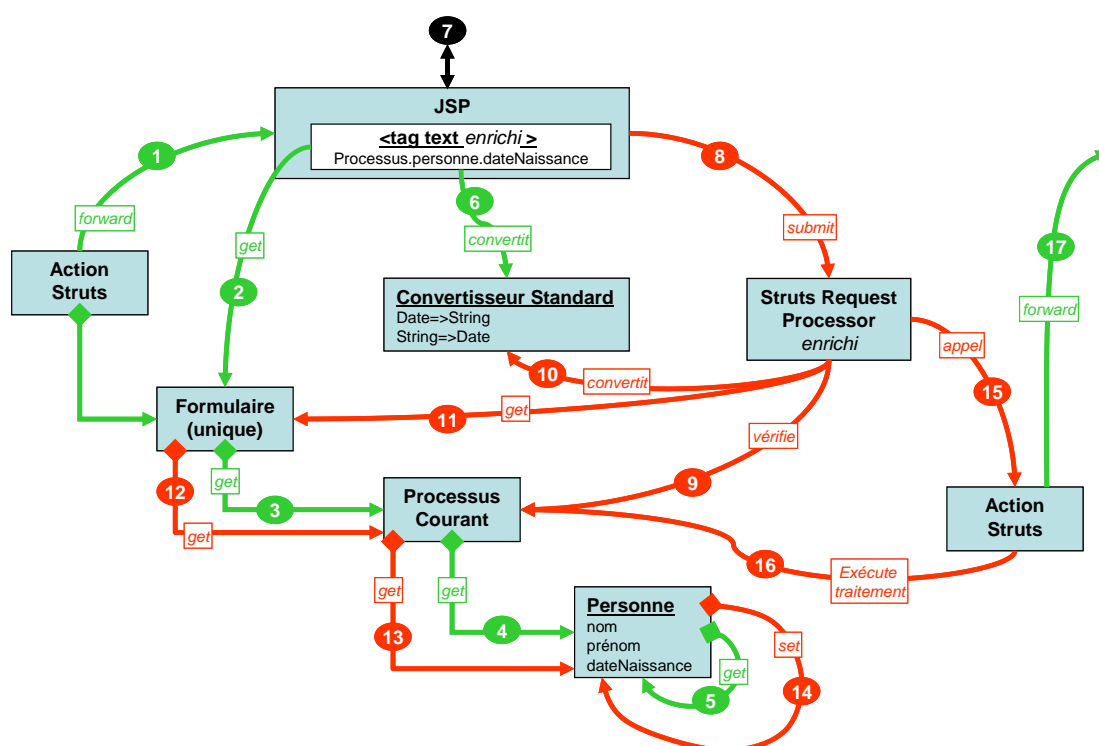


Figure 16 : Exemple d'interaction Page - Formulaire

1. Comme vu précédemment l'action « forward » une page JSP.
2. La page JSP contient un formulaire, qui est construit via le tag texte de Struts. Le tag accède aux données via le formulaire de l'action Struts (formulaire porté par une instance de l'unique classe formulaire).
3. Le formulaire sert de passerelle pour atteindre le processus courant.
4. Le processus courant sert de passerelle pour atteindre les données en cours d'utilisation : par exemple la personne
5. La personne est un bean qui possède des accesseurs sur ses champs.
6. Le tag a récupéré l'information, il appelle le convertisseur standard pour obtenir la donnée en format texte.
7. L'utilisateur reçoit la page et fait des saisies avant de valider.
8. La requête issue du « submit » du formulaire arrive dans le Request Processor de Struts (classe enrichie).
9. Avant prise en compte des données soumises le processus en cours est appelé pour qu'il vérifie les données reçues et émette un veto si nécessaire.
10. Le Request processor convertit les données du format texte vers le format réel avant de faire les affectations.
11. Les données reçues correspondent à la date de naissance de la personne du processus, une suite de « get » est donc déclenchée via le formulaire.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			37/7 7

12. Accès au processus courant du formulaire.
13. Accès à la personne du processus.
14. Mise à jour de la date de naissance de la personne.
15. Une fois les données reçues propagées, le Request Processor Struts continue son travail habituel en appelant l'action correspondante.
16. On se retrouve dans la situation du schéma précédent sur la navigation, l'action invoque le processus courant.
17. Après que le processus courant ait exécuté le traitement, l'action fait le « forward » correspondant à l'état du processus.

..... et ainsi de suite.....

2.3.2. Principes et définitions de la couche coordination

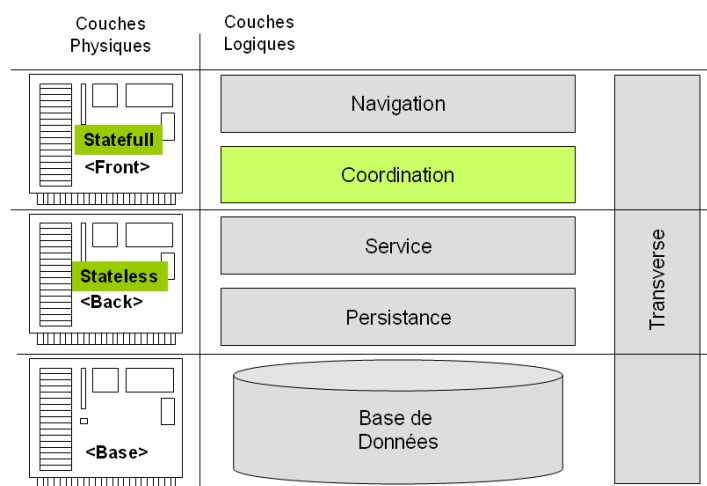


Figure 17 : Principes de la couche coordination

Rappel : La couche coordination est en charge :

- ✓ De la gestion de la session utilisateur
- ✓ Des traitements qui doivent satisfaire les requêtes http des utilisateurs
- ✓ De la gestion du cache de données sur lesquelles l'utilisateur est en train de travailler
- ✓ De la sécurité applicative : contrôle de la conformité des requêtes utilisateurs avec son profil

2.3.2.1. Le contexte Utilisateur

Le contexte utilisateur représente l'ensemble des informations utiles dans le cadre du travail en cours réalisé par l'utilisateur. Cela comprend :

- ✓ Des informations sur son profil (ses droits)

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			38/7 7

- ✓ Un cache de données en cours d'utilisation par les processus métiers en cours d'exécution
- ✓ La pile des processus métiers en cours d'exécution (tel que décrite précédemment)

Le contexte utilisateur est donc un objet Java simple (non J2EE), il sera placé par la couche navigation dans la session http lors de l'initialisation de celle-ci.

2.3.2.2. Les processus métiers

Comme expliqué précédemment dans la couche navigation, un processus métier regroupe les traitements nécessaires pour satisfaire les requêtes utilisateurs venant d'un groupe d'écrans. En fait, le regroupement des écrans sera proche de, ou plus fin que, la notion de CU (cas d'utilisation), le processus métier contiendra donc les traitements pour les actions faites dans le cadre d'un CU (ou plus fin).

Les processus métiers sont donc très proches des conceptions générales et détaillées du comportement fonctionnel. Comme vu précédemment, chaque processus a un état et cet état est utilisé pour déterminer la page à proposer à l'utilisateur.

L'organisation en pile des processus métiers permet la mise en place d'une navigation simplifiée. Grâce à la pile un processus métier n'a pas besoin de connaître le processus appelant pour lui rendre la main : lorsque le processus métier démarre, il est mis en dessus de pile, lorsqu'il se termine, il est retiré de la pile, le processus en dessus de pile quel qu'il soit, devient alors de nouveau processus courant.

2.3.2.3. Le cache et les données métiers

2.3.2.3.1. Les données métiers

Classiquement, les données sont stockées dans une base de données relationnelle (DB2) et elles sont représentées par un modèle de classes Java afin d'être manipulées dans les processus et services métiers.

Un mapping Objet/Relationnel définit la correspondance entre le modèle de base de données et le modèle de classe Java.

Comme les données métiers sont manipulées par les services (couche service du « back »), mais aussi par les processus métiers (couche coordination du « front ») elles font partie de la couche transverse.

Quasiment n'importe quelle classe de type Java Bean peut devenir une classe d'objets métier, l'exigence principale est que la classe sache fournir un identifiant métier (en fait la clé unique de l'objet dans le monde Amalfi).

L'architecture mise en œuvre ici a une spécificité dans la structuration des relations entre objets métiers (référence 0..1 ou liste). En effet les objets métiers ne sont pas liés directement, afin de simplifier le travail du ramasse miettes Java, mais surtout afin de rendre le chargement des objets métiers indépendant les uns des autres, les liens entre objets sont établis grâce aux identifiants, par exemple :

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			39/7 7

- si un objet droit a un lien 1 vers un objet personne, on ne place pas directement un attribut de type personne dans le droit, mais un identifiant de personne. Ainsi, le droit pourra être chargé indépendamment de la personne
- si un objet copropriété a un lien 1..n vers un objet lot, on ne place pas directement une collection de lots dans la copropriété, mais une liste d'identifiant de lots. Ainsi, la copropriété pourra être chargé indépendamment des lots.

La possibilité de chargement indépendant des objets est indispensable pour bénéficier pleinement de l'efficacité du cache des objets métiers et avoir ainsi une véritable économie d'allers-retours entre le « front » et le « back », mais aussi pour avoir une garantie de mono instanciation : un objet donnée, par ex la personne P2006COL00023 version x, ne doit être présent qu'une fois dans un contexte utilisateur, d'une part pour des raisons d'économie de mémoire, mais surtout pour des raisons de cohérence (si on avait plusieurs objets présents, ils n'auraient probablement pas tous le même contenu, lequel serait le « bon » ?)

2.3.2.3.2.

Le cache

Le cache contient les objets métier en cours d'utilisation par les processus métier en cours d'exécution. Il contient les objets en cours de modification, mais aussi les objets lus.

Il y a plusieurs raisons principales d'avoir un cache de données métiers :

- ✓ Il faut conserver les données en cours de modification jusqu'à la fin de la transaction (même si la saisie se fait à travers plusieurs écrans : plusieurs écrans de saisies avec un bouton « valider » global pour toutes les saisies de tous les écrans)
- ✓ Il faut économiser les appels de services en lectures qui causent des chutes de performance en saturant les couches services et d'accès aux données du serveur « back » ainsi que la base de données
- ✓ Il faut garantir la mono instanciation des objets métiers dans un contexte utilisateur (cf. paragraphe précédent).

2.3.2.3.3.

Les transactions

Les changements dans le cache des objets métiers sont gérés dans un cadre transactionnel respectant les principes d'acidités (ACID) des transactions :

- Atomicité, la transaction est prise en compte globalement (tout ou rien)
- Consistance, si la transaction est acceptée c'est que toute les données quelle contient sont valides.
- Isolation, les données en cours de modification dans une transaction ne sont pas visibles des autres transactions.
- Durabilité, les données validées dans le système sont conservés dans un état stable.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			40/7 7

Les transactions misent en œuvre au niveau du cache sont des transactions mémoires (pas dans la base). Ces transactions peuvent être imbriquées (possibilité de démarrer une transaction à l'intérieur d'une transaction).

Les transactions du cache de données métier sont pilotées par les processus métier qui les démarrent et les valident ou les annulent.

Une transaction est en fait un journal (mémoire) des changements effectués depuis le début. Valider la transaction, c'est envoyer tous les changements dans la liste des changements de la transaction englobante. Annuler la transaction, c'est défaire les changements. La validation de la transaction de plus haut niveau (qui n'a pas de transaction englobante), consiste à mettre à jour les informations de la base de données, pour cela le cache appelle le service de mise à jour correspondant au processus en cours d'exécution.

Classiquement, chaque processus démarre une transaction lorsqu'il commence et la valide ou l'annule lorsqu'il se termine. Si le processus est un processus sous-traitant, la validation de celui-ci fera entrer les changements dans la transaction du processus appelant. La validation ou l'annulation de la transaction de plus haut niveau se termine par la suppression du journal des changements.

2.3.2.4. Synthèse de la couche coordination

2.3.2.4.1. *Exemple de comportement de la pile de processus métiers*

Le diagramme ci-après décrit le comportement de la pile de processus et les interactions avec les actions Struts (Classe Action Générique) et les pages JSP. Le scénario décrit suit les étapes A à H du point de vue de l'utilisateur :

Etape A : Validation de la page de connexion (nom d'utilisateur / mot de passe)

Etape B : Choix du menu d'accès au module personne dans la page d'accueil

Etape C : Choix du menu « recherche de personne »

Etape D : Saisie des critères de recherche et lancement de la recherche

Etape E : Choix dans la liste résultat d'une personne et demande de consultation

Etape F : Sur la page de synthèse de la personne demande d'accès à la page de détail

Etape G : Fermeture de la consultation personne pour retour au résultat de recherche

Etape H : Choix dans la liste résultat d'une autre personne et demande de consultation

.....Etc....

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			41/7 7

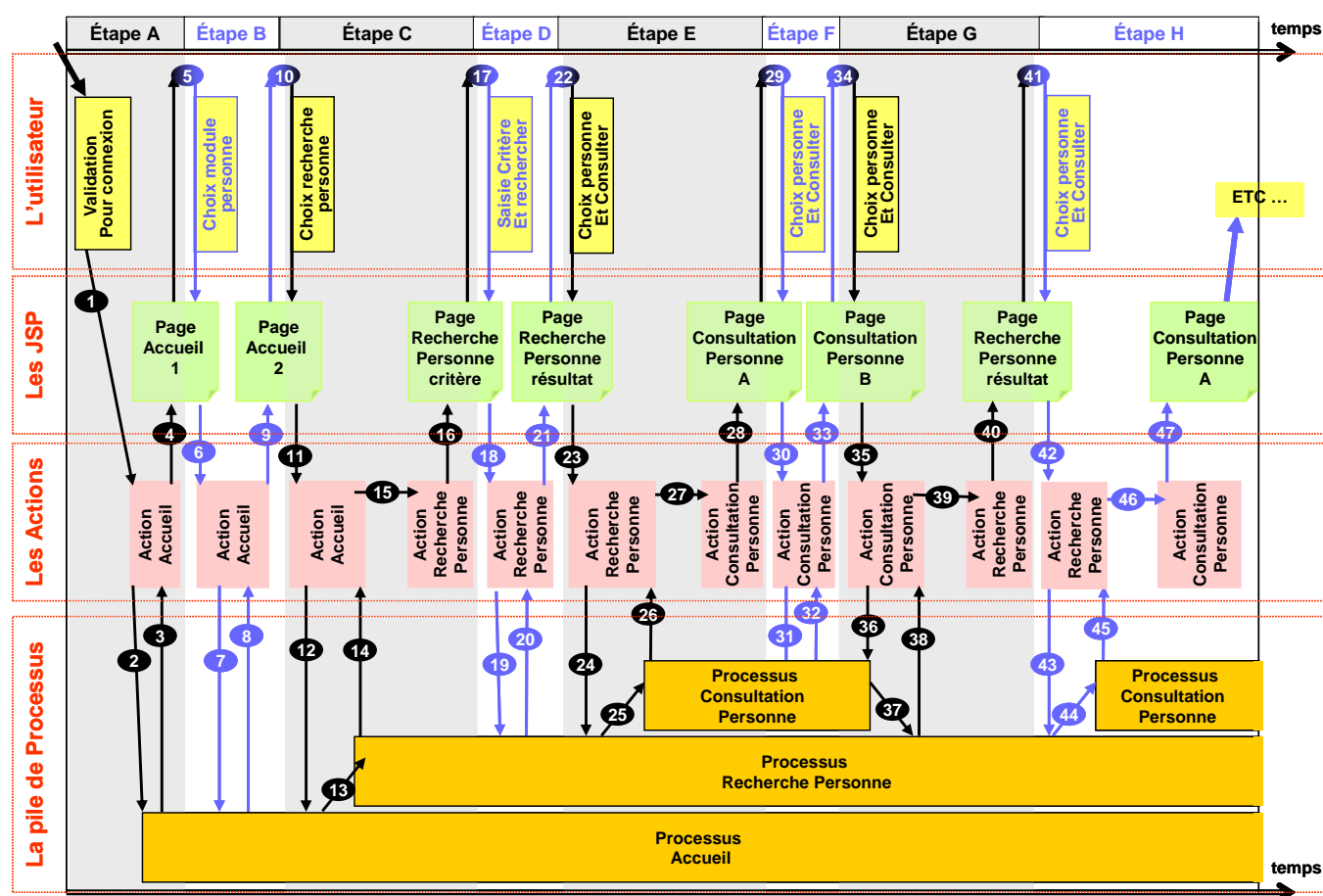


Figure 18 : Comportement de la pile Processus et interactions Processus-Action-JSP

2.3.3. Principes et définitions de la partie présentation

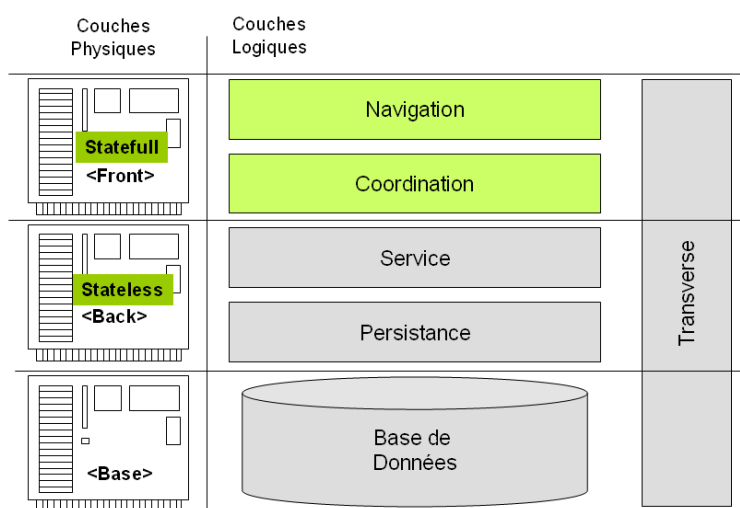


Figure 19 : Principes de la présentation

Ci-dessous une vue générale des couches navigation et coordination :

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			42/7 7

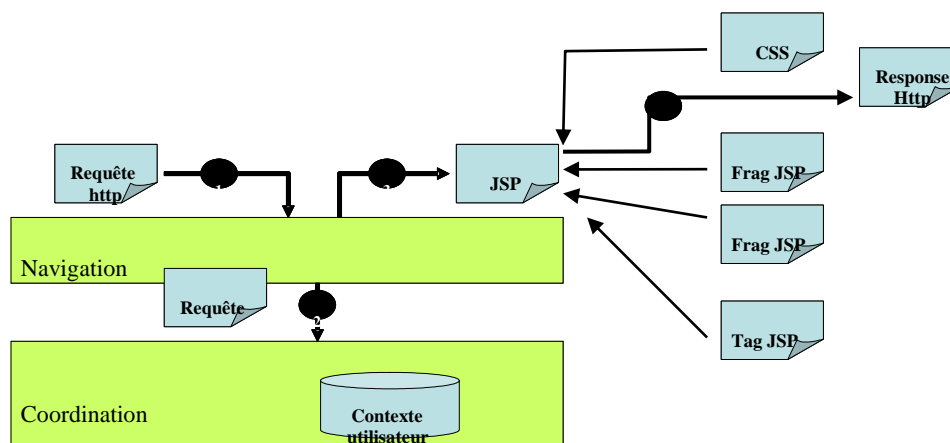


Figure 20 : Vue générale des couches navigations et présentation du point de vue de la présentation des informations

La couche présentation a pour objectifs principaux :

- ✓ La présentation des informations à l'utilisateur
- ✓ La génération de liens et d'informations pour l'envoi des événements utilisateurs à la couche navigation (sous forme de requêtes HTTP)
- ✓ L'édition des états imprimés

L'ensemble de ces objectifs doit être atteint en respectant les règles d'ergonomie, de présentation et d'accessibilité définies pour AMALFI V2. D'un point de vue technique la couche *Présentation* est gérée exclusivement avec des pages JSP.

2.3.3.1. Ergonomie, Charte graphique et Accessibilité

Les règles d'ergonomie sont définies dans la version applicable du document :

- ✓ CD_Ergo_Document_Ergonomie_vx.y.doc

La charte graphique AMALFI V2 est définie dans la version applicable du document

- ✓ CD_Ergo_Charte_Graphique vx.y.doc

L'application AMALFI V2 doit suivre les règles d'accessibilité WCAG. Ces règles sont définies par le W3C. Les règles d'accessibilité sur lesquelles nous nous appuyons sont les règles de priorité 2.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			43/7 7

- ✓ <http://www.w3.org/TR/WAI-WEBCONTENT/#wc-priority-2>

Voici les principales règles :

- ✓ Fournir des alternatives équivalentes au contenu auditif et visuel
- ✓ Ne pas s'en remettre uniquement aux couleurs
- ✓ Utiliser le balisage et les feuilles de style, et cela de façon appropriée
- ✓ Clarifier l'utilisation du langage naturel
- ✓ Créer des tableaux qui se transforment de façon élégante
- ✓ S'assurer que les pages qui contiennent de nouvelles technologies se transforment de façon élégante
- ✓ Assurer à l'utilisateur le contrôle des changements du contenu lorsque ce dernier varie dans le temps
- ✓ Assurer un accès direct aux interfaces utilisateur intégrées

2.3.3.2. Présentation des données à l'utilisateur

Nous retrouverons dans l'application des types de pages qui seront réutilisables. Ils ont tous la même structure. Ils comportent un entête (incluant le menu de l'application), un corps de page (c'est ce qui est différents entre chaque type) et un pied de page.

Voici la liste de ces différents types :

- ✓ Page de recherche
 - Page de recherche standard
 - Page de recherche avec périmètre de recherche
- ✓ Page de détails d'un objet
- ✓ Page de saisie d'un Acte de Gestion
 - Création/Modification d'un acte de gestion
 - Synthèse d'un Acte de Gestion
- ✓ Les pages d'aide à la sélection
- ✓ Les affichages de liste des requêtes

2.3.3.3. Édition des états imprimés

Il existe 2 sortes d'impressions :

- les impressions dites « génériques », qui permettent d'avoir sur papier un contenu très proche de l'écran.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			44/7 7

- Les impressions dites « spécifiques », qui produisent un document pdf imprimable suivant un modèle spécifié.

2.3.3.3.1.

Les impressions génériques

La gestion de l'impression des pages HTML se fera à l'aide d'un fichier CSS propre qui ne sera utilisé que pour les impressions. L'impression reprendra la totalité des données affichées à l'écran à l'exception des parties qui seront défini dans ce fichier CSS.

Ces parties sont :

- ✓ l'entête de page (qui contient le menu)
- ✓ les boutons de navigations
- ✓ les zones de saisie
- ✓ le pied de page

2.3.3.3.2.

Les impressions spécifiques

Le principe général est conservé, une action d'un processus demande la production d'un document pdf pour impression, dans ce cas :

- au clic de l'utilisateur l'action part vers le serveur et le processus fait la collecte de toutes les données nécessaires au document (accès base, recherche etc.).
- en retour le navigateur de l'utilisateur reçoit l'ordre d'ouvrir une nouvelle fenêtre dont l'url correspond au pdf.
- Cette url, pointe sur un servlet de génération de pdf, qui lance le générateur du document à proprement parler, ce générateur interroge le processus en cours pour obtenir les informations à mettre dans le document.

La plupart des documents sont produits à la demande de l'utilisateur et ne sont pas stockés sur le serveur.

Quelques documents nécessitent une sauvegarde particulière dans la GED, par exemple les ordonnances d'inscription. Dans ce cas ils sont transmis au serveur de GED via l'interface du client toujours.

Le moteur de génération PDF utilisé est la librairie libre iText.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			45/7 7

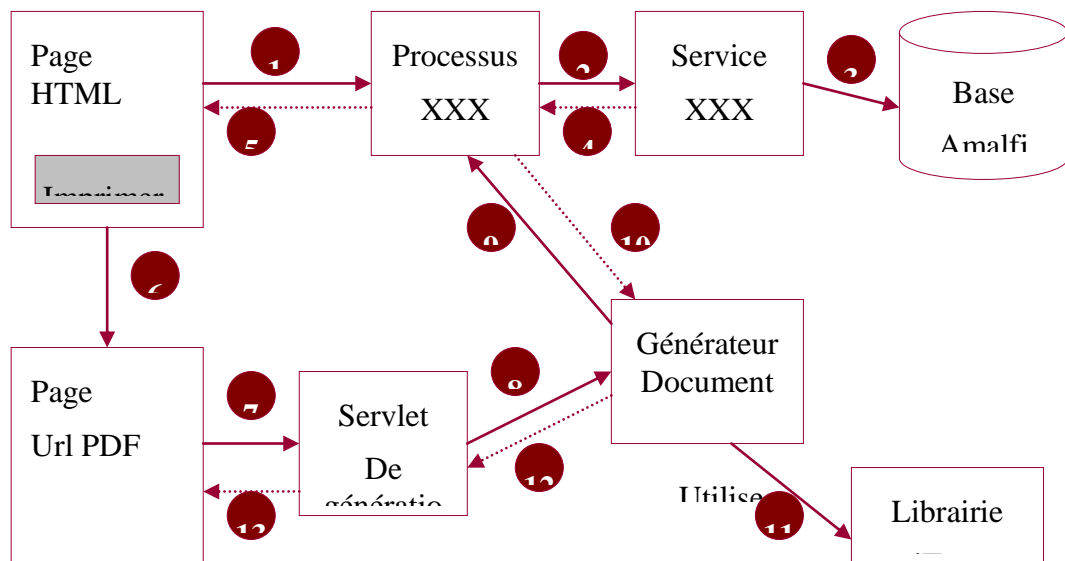


Figure 21 : Diagramme de synthèse de la construction d'un PDF pour impression.

2.4. Principes et définitions de l'accès aux services

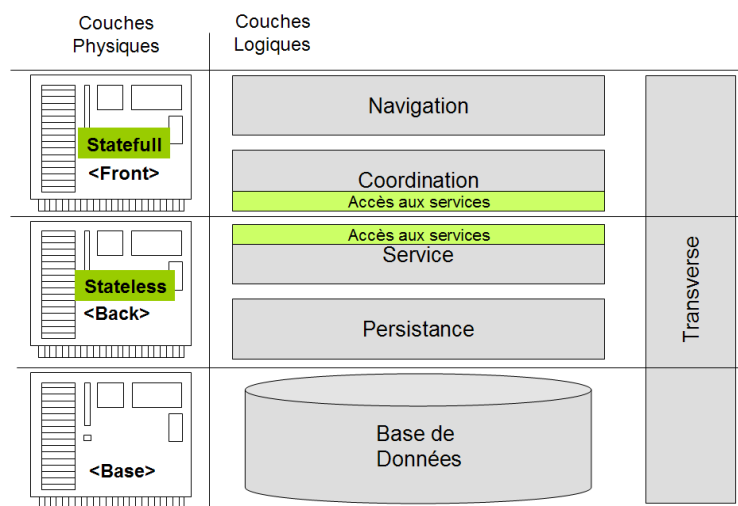


Figure 22 : Principes de l'accès aux services

Comme décrit sur le schéma ci-dessus la couche d'accès aux services est un composant dont on retrouve des parties aussi bien dans le Front (couche Coordination) que dans le back (couche Service), une partie des éléments du composant se retrouve donc dans la couche transverse (non représenté sur le schéma).

La couche d'accès aux services a pour objectifs principaux :

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			46/7 7

- ✓ de permettre la communication simple de données structurées entre les couches coordination et service
- ✓ de permettre un accès simple et centralisé aux services de la couche service par la couche coordination à l'aide d'un gestionnaire de services
- ✓ de permettre de communiquer en différents modes entre les couches (ce qui est transparent pour le client du service) : en mode distant (à l'aide d'un EJB ou d'un MDB suivant que l'appel doit être synchrone ou asynchrone) ou en mode local (les couches coordination et service sont dans la même JVM et l'appel aux services ne passe pas par un middleware, ce qui permet un test complet sans utiliser un conteneur J2EE)
- ✓ de permettre de configurer la communication entre le front et le back
 - le mode d'appel local est utilisé lors de la phase des développements pour permettre un test plus léger et simple de l'application (tests hors conteneur sous forme de JUnit, ou test conteneur « léger » type Jetty pour les parties Web : jsp, Struts, ...)
 - le mode distant sera lui mis en œuvre durant la phase d'homologation afin de coller au mode opérationnel d'exploitation, puis durant les phases de pré-production et de production
- ✓ d'offrir une gestion simple des EJB avec en mode distant un EJB unique qui se charge de router l'appel de service vers la bonne implémentation du service demandée

D'un point de vue technique seul le mode distant (EJB ou MDB) contient des aspects J2EE, alors que le mode local est constitué de java simple (JSDK) et peut donc se tester hors conteneur.

2.4.1. Vue générale de l'accès aux services en EJB (Site Central)

Ce mode est utilisé par l'application Amalfi v2 site Central. Les utilisateurs accèdent donc à travers le RPVJ.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			47/7 7

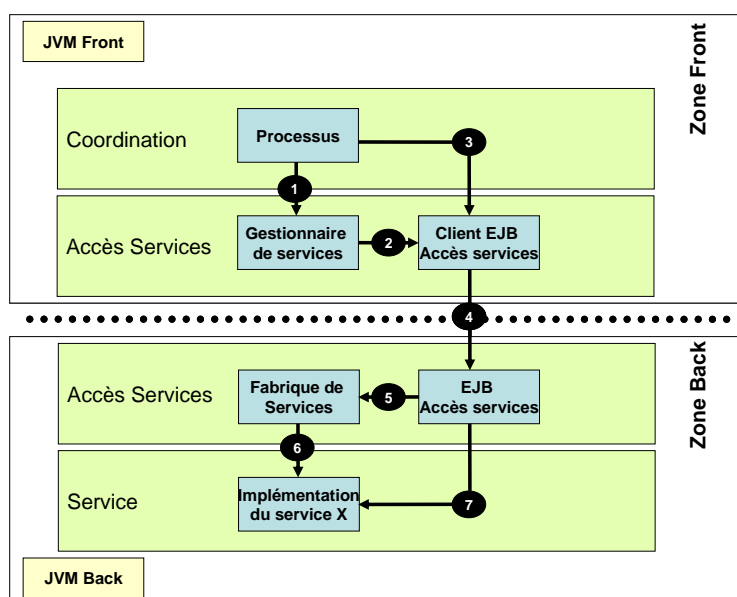


Figure 23 : Vue générale de la couche d'accès aux services en mode distant synchrone (EJB)

- ✓ le processus demande au gestionnaire de services une implémentation de l'interface du service X (1)
- ✓ le gestionnaire de services renvoie au processus un « client de l'EJB Accès services » qui implémente l'interface du service X (2)
- ✓ le processus peut appeler la méthode métier voulue sur le client de l'EJB (3)
- ✓ le client de l'EJB Accès services va déléguer à l'EJB le soin d'appeler la méthode, pour cela le client lui passe le nom de l'interface du service X, le nom de la méthode et les arguments (4)
- ✓ l'EJB demande à la fabrique de services métiers une implémentation de l'interface du service X (5)
- ✓ la fabrique donne à l'EJB une référence de l'implémentation du service X (6)
- ✓ l'EJB peut appeler la méthode demandée sur l'implémentation du service X et renvoyer les résultats au client de l'EJB qui lui renvoie les résultats au processus (7)

2.4.2. Vue générale de l'accès aux services en mode local (Test)

Ce mode est utilisé pour les tests en développement, en effet la possibilité de fonctionner hors conteneur J2EE permet d'exécuter des tests avec JUnit.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			48/7 7

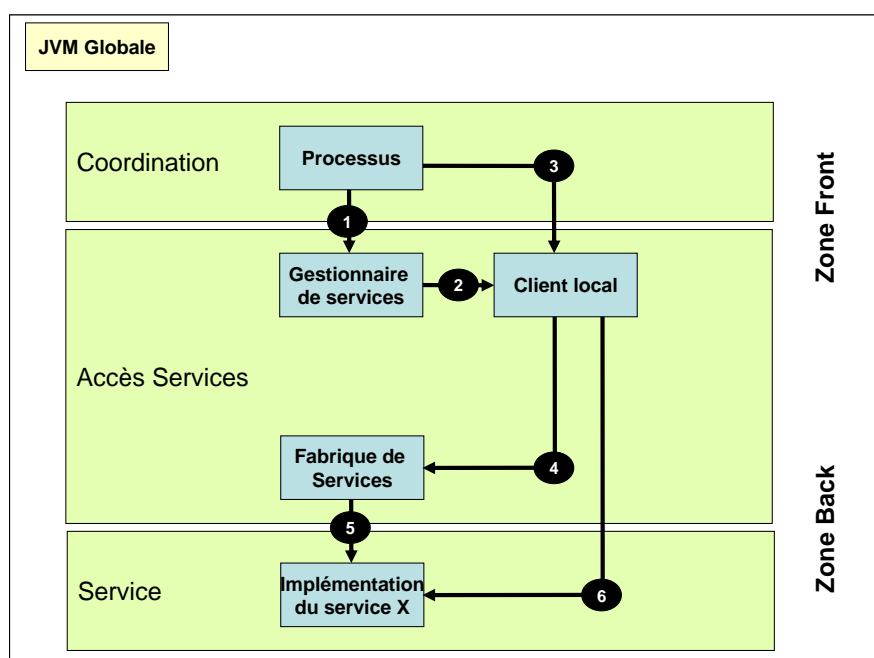


Figure 24 : Vue générale de la couche d'accès aux services en mode local

- ✓ contrairement au cas précédent l'ensemble des traitements s'effectue dans une seule JVM
- ✓ le processus demande au gestionnaire de services une implémentation de l'interface du service X (1)
- ✓ le gestionnaire de services renvoie au processus un « client local générique » qui implémente l'interface du service X (2)
- ✓ le processus peut appeler la méthode métier voulue sur le client local (3)
- ✓ le client local demande à la fabrique de services métiers une implémentation de l'interface du service X (4)
- ✓ la fabrique donne au client local une référence de l'implémentation du service X (5)
- ✓ le client local peut appeler la méthode demandée sur l'implémentation du service X et renvoyer les résultats au processus (6)

2.4.3. Vue générale de l'accès aux services en MDB (SSEE)

Ce mode est utilisé par l'application Amalfi v2 SSEE. Les utilisateurs accèdent donc à l'application à travers le web.

Remarque : Comme les exigences, et par conséquent la solution retenue, sortent un peu des sentiers battus, la description est plus précise que les autres.

2.4.3.1. Rappel des choix et contraintes pour le SSEE

Le SSEE est la partie de l'application permettant aux personnes externes (principalement les notaires et autres habilités loi 1904, mais aussi les banques et le grand public) d'accéder aux données du Livre Foncier. Il doit donc être ouvert sur l'Internet.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			49/7 7

Comme l'application SC (Site Central) du Livre Foncier, la partie applicative du SSEE se décompose en 2 EAR :

- un EAR dit « Front »
- un EAR dit « Back ».

Le front contient les écrans et la session utilisateur avec toutes ses données contextuelles.

Le back contient les services d'accès aux données.

Choix : Afin d'avoir une synergie temps réel entre le SC et le SSEE, il a été décidé que la base de données seraient mutualisées.

Le SSEE étant ouvert sur l'Internet, il faut être particulièrement vigilant à la sécurité, d'autant que SC et SSEE partagent la même base de données.

Contrainte : Il a donc été décidé que la partie front du SSEE serait séparée du SC (front / back / base de données) par un pare-feu entièrement bloqué dans le sens SSEE vers SC.

Une reformulation de cette contrainte est : l'EAR front du SSEE ne peut pas appeler que le BACK SSEE. Par contre les différentes parties du SC peuvent faire des appels au SSEE.

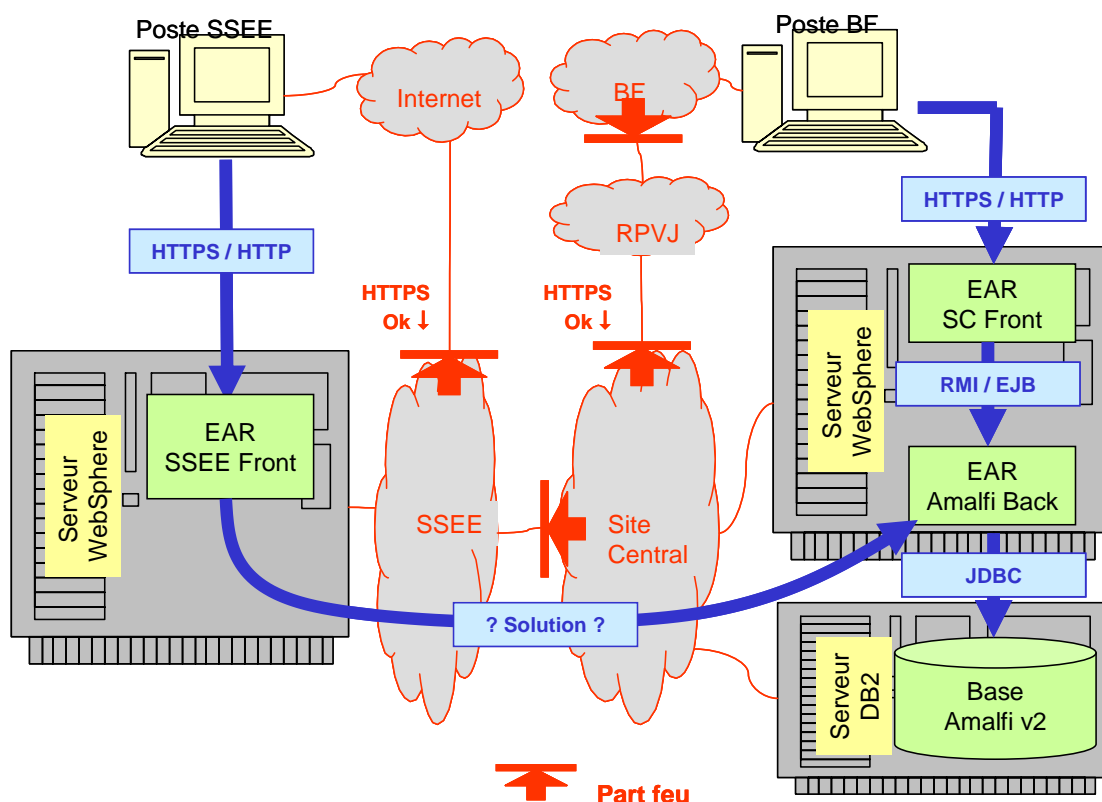


Figure 1: Choix et Contrainte d'architecture SSEE

La communication par EJB choisie pour le SC ne convient donc plus, il faut mettre en place une solution de communication spécifique entre SSEE et SC.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			50/7 7

2.4.3.2. Vue logique de la solution de communication pour le SSEE

La conséquence des contraintes précédentes est que la communication Front vers Back du SSEE doit être à l'initiative du Back.

La solution s'appuie sur la mise en place d'un système de communication pseudo synchrone.

Côté Front SSEE, on place un gestionnaire de file d'attente avec 2 files.

- Une file pour les requêtes.
- Une file pour les réponses.

L'appel à un service fonctionne comme suit :

- Le Front SSEE dépose dans la file des requêtes une demande d'exécution de services. Cette demande est en fait un message contenant toutes les informations nécessaires à l'appel du service.
- Le Front SSEE se met ensuite en attente d'une réponse à sa requête dans la file des réponses. Le message de requête a un identifiant unique, la réponse reprendra cet identifiant. Remarque : le SSEE se met en attente pour une durée limitée, si la réponse n'arrive pas dans un délai raisonnable une erreur sera déclenchée.
- Le Back contient un « Listener » qui s'active lorsque des messages sont déposés dans la file des requêtes. Ce « Listener » interprète le message de la requête pour déclencher l'appel au service demandé après avoir vérifié les autorisations de l'appelant.
- Une fois, le service exécuté, le Back place un message de réponse dans la file des réponses placée du côté SSEE.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			51/7 7

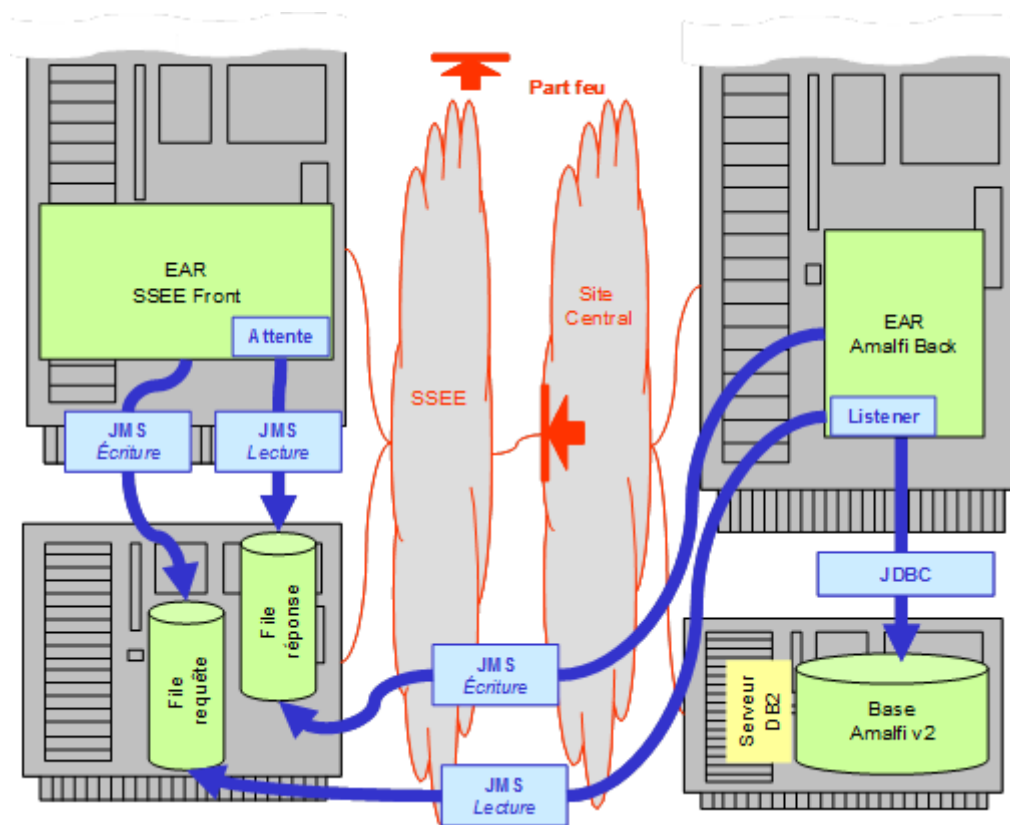


Figure 2: SSEE La solution de communication

2.4.3.3. Vue technique de la solution de communication pour le SSEE

2.4.3.3.1.

Diagramme de classes

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			52/7 7

Comme pour l'EJB Session d'accès service du « SC », l'EJB MDB s'initialise par la méthode ejbCreate.

2.4.3.3.2.

Diagrammes de collaboration

Les diagrammes de collaborations suivant montrent les phases d'initialisation, d'envoi et de réception de messages.

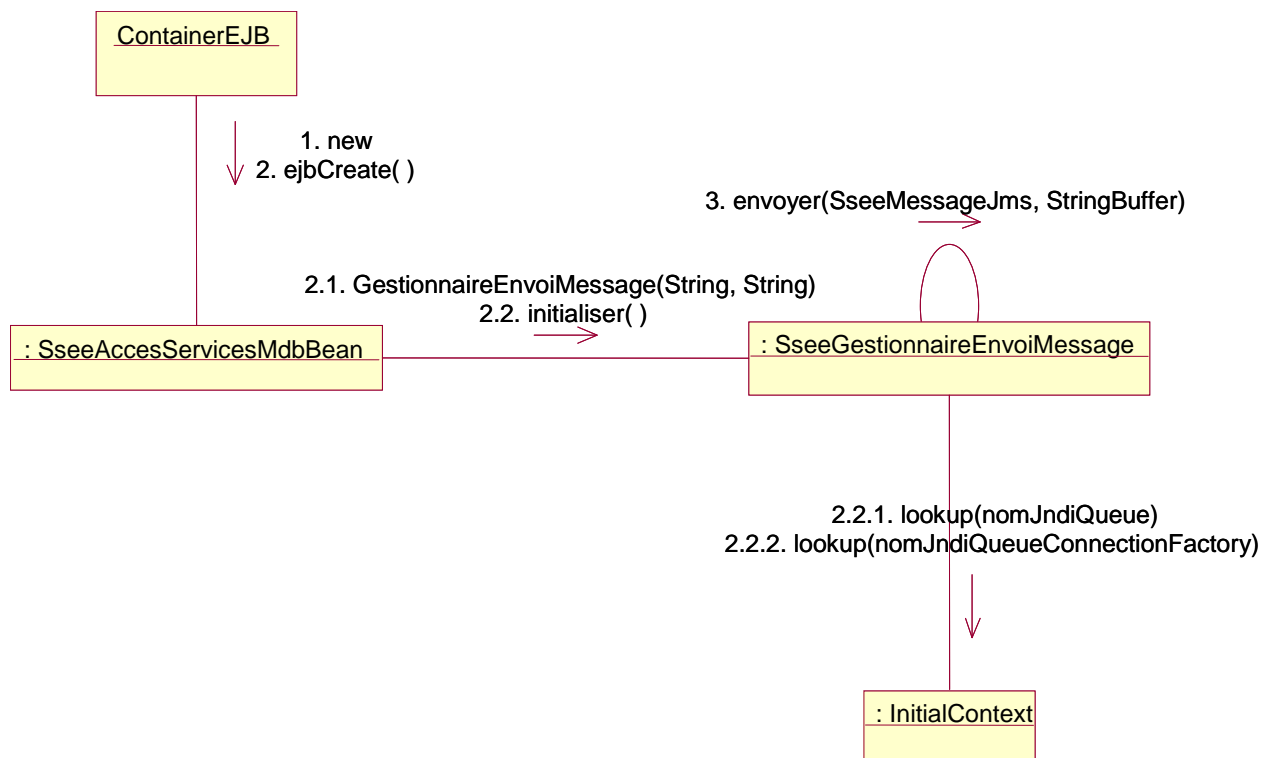


Figure 4: Diagramme de collaboration d'initialisation du « back » SSEE

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			54/7 7

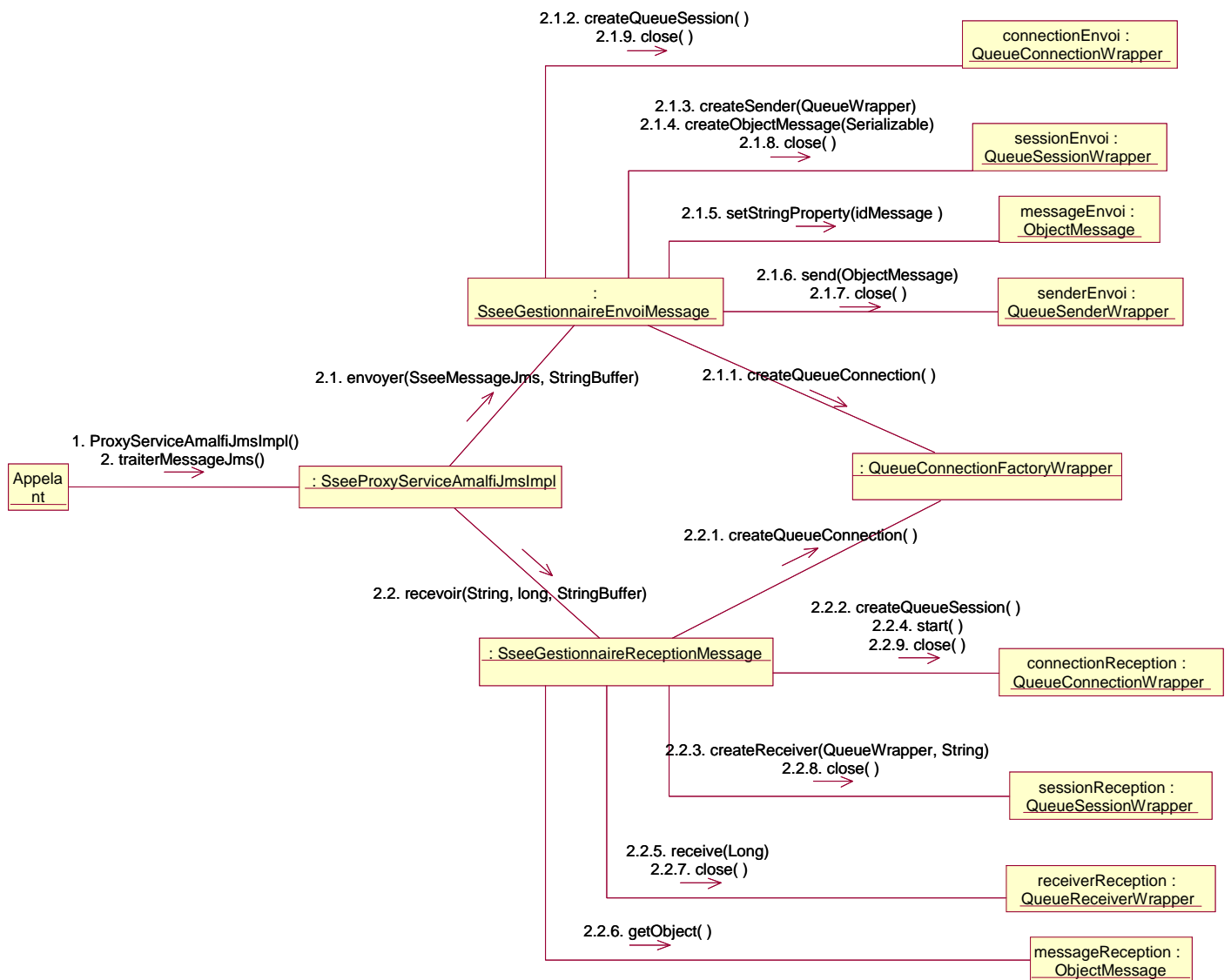


Figure 5: Diagramme de collaboration d'appel de service par le front SSE

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			55/7 7



Figure 6: Diagramme de collaboration d'exécution de service par le « back » SSEE

2.4.4. La partie Front de la couche d'accès aux services

2.4.4.1. Le gestionnaire de services

Le gestionnaire de services est le point central d'accès aux services pour les processus de la couche coordination.

Le processus client doit spécifier au gestionnaire l'interface métier dont il veut une implémentation (cette interface devra étendre l'interface globale des services Amalfi). Le gestionnaire de services fournit une implémentation ou renvoie une exception si cette implémentation n'existe pas.

Ainsi, le processus client récupère une implémentation de l'interface de service et peut l'utiliser sans se préoccuper du mode de communication (local ou distant).

Lors des tests en développement c'est une communication « locale » qui se fera entre la couche coordination et la couche service. Ainsi, le test des processus et des services sera possible avec des tests JUnit hors d'un conteneur J2EE.

Pour récupérer l'implémentation de l'interface demandée par le processus, le gestionnaire s'adresse à une fabrique qui donne une implémentation suivant le mode de communication configuré : local, EJB, MDB, ...

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			56/7 7

2.4.4.2. La fabrique des proxys de services

Le rôle de cette fabrique est de retourner au gestionnaire de services une implémentation de l'interface demandée par le processus suivant le protocole de communication choisi : local, EJB, MDB, ... Pour cela la fabrique fournit un représentant (le proxy) du service.

2.4.4.3. Le proxy de service local

C'est le proxy de service renvoyé par la fabrique des proxys de services en mode local (1 JVM). Il permet de tester les processus et services hors d'un conteneur J2EE, les appels aux services peuvent donc être testés au sein d'un test JUnit simple. Ce proxy utilise la fabrique des services métiers pour obtenir une implémentation réelle de l'interface de service.

2.4.4.4. Le proxy de service EJB

C'est le proxy de service renvoyé par la fabrique des proxys de services en mode EJB. Il appelle l'EJB Accès services du back qui exécute la méthode demandée. Ce proxy utilise les concepts J2EE (EJB, nommage JNDI), il ne peut donc être utilisé et testé que dans un conteneur J2EE. Ce proxy est utilisé en production pour la communication entre le front SC et le back.

2.4.4.5. Le proxy de service JMS

C'est le proxy de service renvoyé par la fabrique des proxys de services en mode MDB. Il appelle le MDB générique qui demande une exécution asynchrone de la méthode demandée (la réponse est attendue dans une queue JMS, il s'agit donc d'un mode pseudo synchrone). Ce proxy utilise les concepts J2EE (EJB, nommage JNDI, JMS), il ne peut donc être utilisé et testé que dans un conteneur J2EE. Ce proxy est utilisé en production pour la communication entre le front SSEE et le back. L'utilisation d'un mécanisme asynchrone permet d'avoir une communication SSEE ↔ SC à l'initiative du SC exclusivement.

2.4.5. La couche back d'accès aux services

2.4.5.1. L'EJB Accès services (MDB pour SSEE, Session pour SC)

Cet EJB unique est le seul point d'entrée du back pour appeler les services de manière distante et synchrone. Sa tâche est d'appeler la méthode voulue sur une implémentation de l'interface de service spécifiée par le processus. Pour trouver cette implémentation de l'interface de service il s'adresse à la fabrique des services métiers.

Cet EJB Accès services est :

- un EJB session Stateless dans le cadre de la communication « SC »,
- un EJB MDB dans le cadre de la communication « SSEE ».

Ils seront testés durant les phases d'homologation et de pré-production (les tests de développements utilisent le mode local).

Avant de lancer l'exécution du service, l'EJB (Session ou MDB) fait appel à la sécurité applicative pour vérifier l'autorisation d'accès au service pour l'utilisateur appelant.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			57/7 7

2.4.5.2. La fabrique des services métiers

Cette fabrique est utilisée soit par l'EJB Accès services (mode EJB), soit par le proxy de service (mode local). Elle délivre une instance qui implémente l'interface demandée.

2.4.6. Schéma de synthèse des principes et définitions

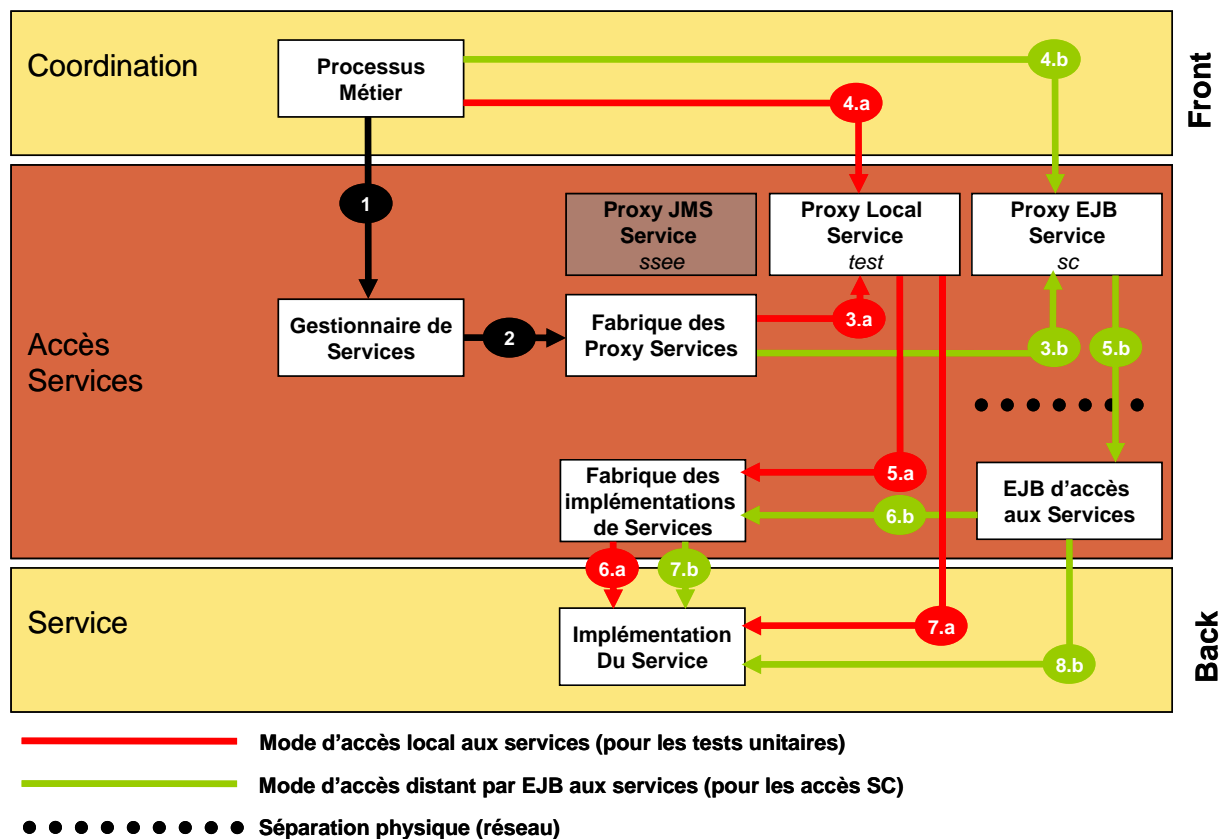


Figure 25 : Synthèse des principes et définitions de la couche d'accès aux services

2.5. Principes et définitions du Back

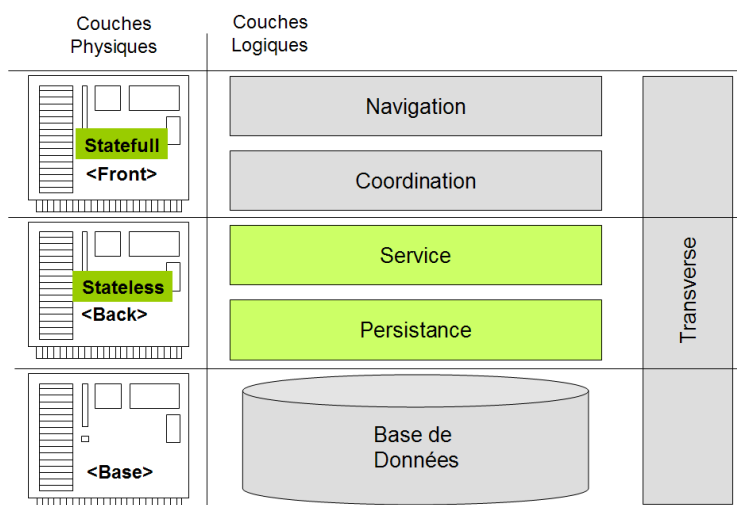


Figure 26 : Principes du Back

Cette partie présente les principes des couches Service et Persistence de la zone Back.

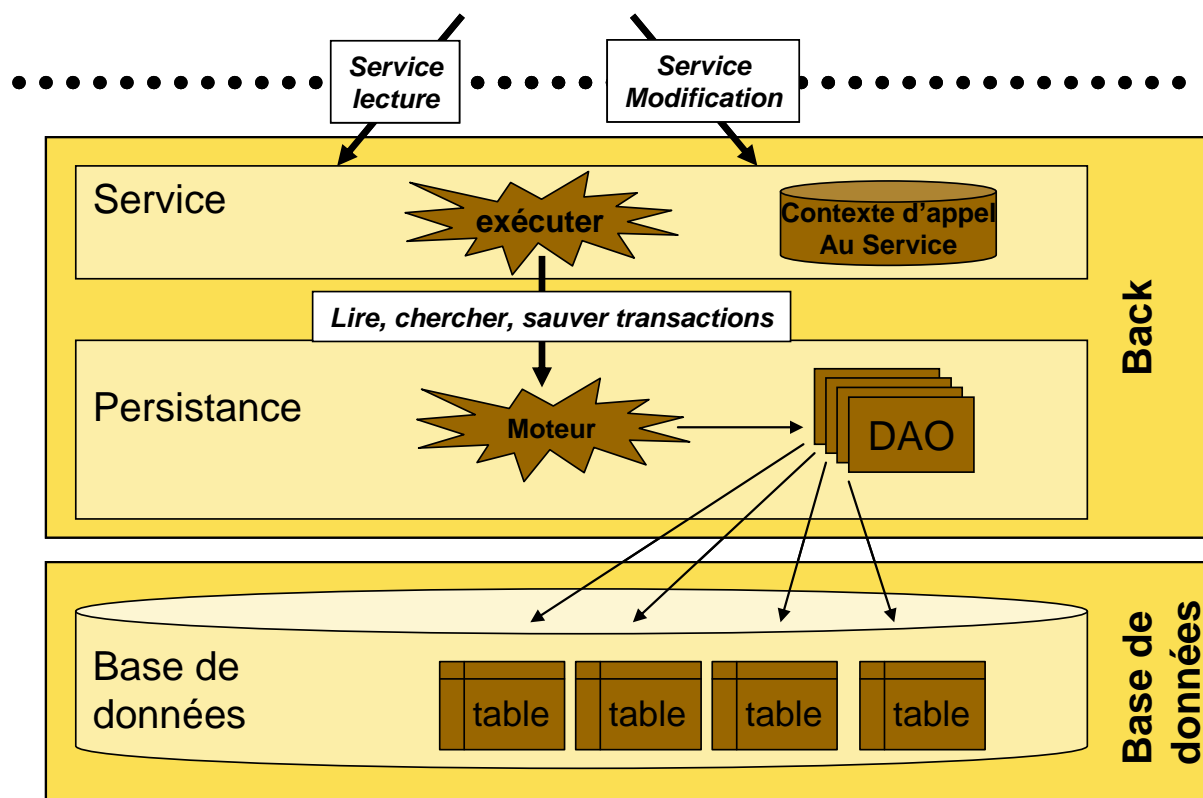


Figure 27 : Vue générale des couches Service et Persistence

La couche service a pour objectif principal :

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			59/7 7

- ✓ De proposer au Front des services (soit de lecture soit de modification) implémentant des contrôles ainsi que des règles de gestion. De même que dans la couche coordination une gestion transactionnelle ainsi qu'un cache d'objets métiers ont été mis en place. Elle utilise la couche persistance pour tous les accès base dont elle a besoin.

La couche persistance a pour but de :

- ✓ Permettre de lire et sauvegarder en base les informations véhiculées dans les objets métiers
- ✓ Permettre l'identification unique des objets métiers (gestion des identifiants)
- ✓ Garantir la cohérence des données en base même lors d'accès simultanés d'utilisateurs différents sur une même information
- ✓ Transcrire au niveau de la base de données la notion d'unité d'œuvre des processus métier : gestion des transactions
- ✓ Permettre de faire des recherches multicritères sur les objets métiers

2.5.1. Principes et définitions de la couche service

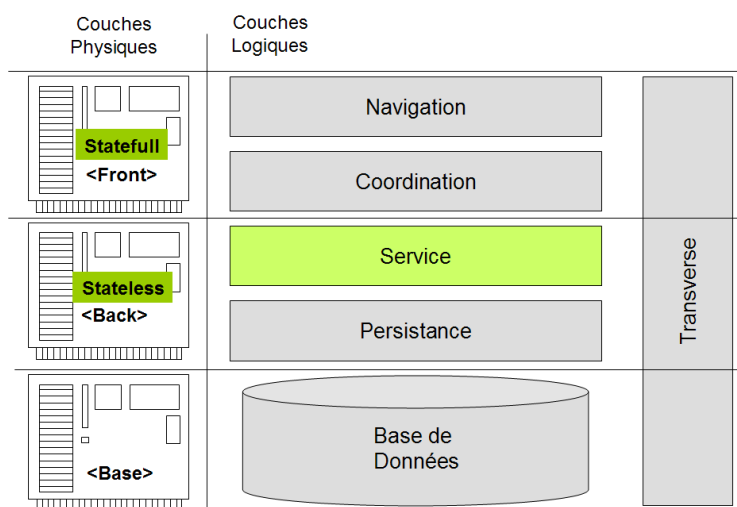


Figure 28 : Principes de la couche service

Rappel : La couche Service est en charge :

- ✓ Des traitements qui doivent satisfaire les demandes de services de la couche Coordination
- ✓ De la gestion du cache de données sur lesquels le service est en train de travailler
- ✓ De la sécurité applicative : contrôle de la conformité des demandes de services avec le profil de l'utilisateur

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			60/7 7

2.5.1.1. Le contexte d'appel de service

Chaque service propose des méthodes qui peuvent être appelées par la couche Coordination. Ces méthodes doivent toutes avoir en premier paramètre le contexte d'appel de service. Ce composant transite entre le front et le back et contient la sécurité applicative de l'utilisateur déclenchant l'appel de service ainsi que le périmètre sur lequel se fait l'appel de service : soit tous les objets du Livre Foncier (objets publiés + objets temporaires) soit uniquement les objets publiés.

2.5.1.2. Le contexte du service

Malgré le fait que la couche service soit stateless, chaque service possède son contexte qui n'est valable que durant la vie du service : lors d'un nouvel appel de service, un nouveau contexte sera créé. Le contexte du service représente l'ensemble des informations utiles dans le cadre du travail en cours réalisé par le service. Cela comprend :

- ✓ Des informations sur le profil de l'utilisateur (ses droits) : la sécurité applicative
- ✓ Un cache de données en cours d'utilisation par le service en cours d'exécution
- ✓ Un gestionnaire de transactions qui assure l'intégrité transactionnelle

Le contexte utilisateur est donc un objet Java simple (non J2EE), il sera placé par la couche service lors de l'appel de service.

2.5.1.3. Le cache et les données métiers

Le cache et les objets métier dans le « back » jouent le même rôle de la même façon que dans le « front », la seule différence se fait au niveau de la validation des transactions de plus haut niveau (sans transaction englobante). Au lieu d'appeler le service de persistance correspondant au processus en cours, la validation de la transaction de plus haut niveau appelle directement la couche persistance : c'est ici que commence la véritable transaction avec la base de données.

2.5.2. Principes et définitions de la couche persistance

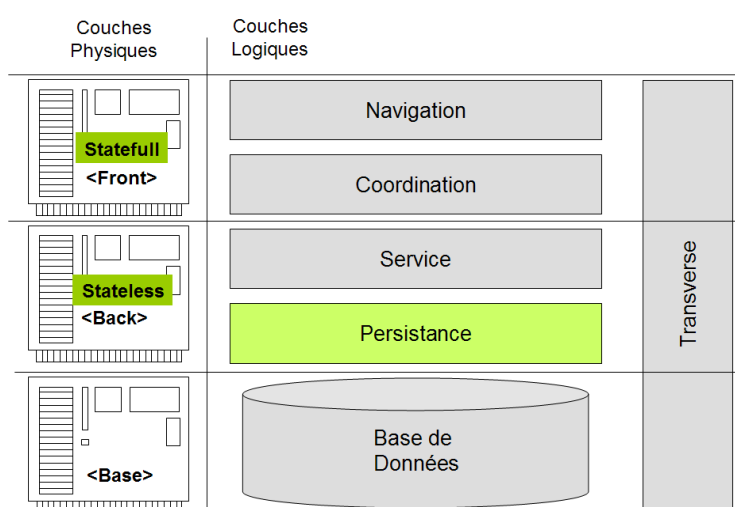


Figure 29 : Principes de la couche persistance

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			61/7 7

2.5.2.1. Principes structurants

2.5.2.1.1.

Classification des objets

On classe les objets en 4 grandes familles. Ce classement est fait par rapport aux besoins spécifiques de l'accès aux données. Les deux concepts qui président à cette classification sont en particulier les suivants :

- ✓ Nécessité ou non de manipuler une version temporaire de l'objet.
- ✓ Nécessité ou non de versionner un objet
- ✓ Nécessité de référencer une version

Type d'objet	Description	Version temporaire	Versionnement	Exemple
Objets LF	Les objets du Livre Foncier	OUI	OUI	Immeuble, Droit, Personne, Charge, Servitude, Mention
Objets RD	Les objets du registre des dépôts	NON	NON	Requête
Objets RF	Les objets du référentiel	NON	OUI	Circonscription foncière, Bureau foncier, toutes les énumérations
Objets Partie LF	Les composants d'un objet Lf, indissociables	OUI	OUI	Règlement d'une copropriété

2.5.2.1.2.

Concept de version

Lorsqu'un objet est modifié, il peut être nécessaire de conserver son état précédent. On dira alors qu'on a deux versions (ou plus) d'un même objet. Une version donnée d'un objet a une date de début (celle de sa création) et une date de fin.

- ✓ La dernière version ou encore version courante d'un objet a une date de fin nulle
- ✓ Une version d'un objet antérieure à la dernière version a nécessairement sa date de fin non nulle et égale à la date de début de la version qui le suit.

Lorsqu'un objet est versionnable on distingue :

- ✓ L'entité qui représente l'objet indépendamment de sa version

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			62/7 7

- ✓ La ou les versions de l'entité

La date de début d'une version est la date de création technique de la version. Elle n'a pas de rôle fonctionnel. Elle est unique pour une version donnée d'un objet Amalfi. Par abus de langage, on pourra appeler version d'un objet la valeur de la date de début pour une version donnée.

L'identifiant de l'entité est une chaîne de caractère dépendant du type de l'objet (par exemple le n° Amalfi dans le cas des objets LF, un code d'énumération dans le cas des objets du référentiels)

- ✓ Les versions d'un objet ont un identifiant composite qui comporte l'identifiant et la date de début

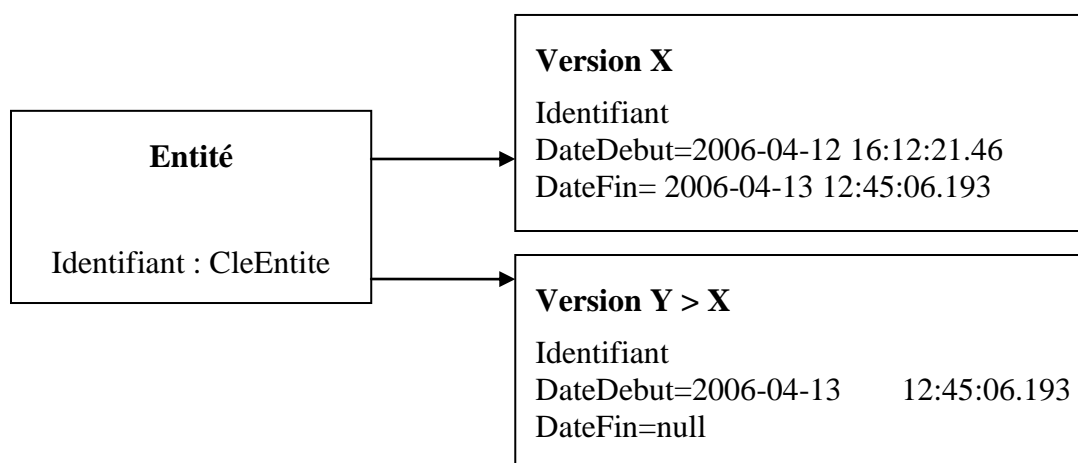


Figure 30 : Le concept de version

2.5.2.1.3.

Référence entre Objets

- ✓ Lorsqu'un objet référence un objet non versionnable, ce référencement se fait via l'identifiant de l'objet non versionnable
- ✓ Lorsqu'un objet référence un objet versionnable, ce référencement peut se faire indépendamment ou non de la version
 - Lorsque l'objet référence une version, ce référencement se fait via l'identifiant de la version elle-même
 - Lorsque la référence doit être indépendante de la version, ce référencement se fait via l'identifiant de l'entité

Dans ce dernier cas (indépendance de la version), l'intégrité de la base de données devant être respectée de manière forte, cette dernière contrainte implique que lorsqu'un objet est versionnable, on distingue la table des versions proprement dite de la table des entités. La

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			63/7 7

table des entités s'appelle un index. C'est une table dont la clef primaire est l'identifiant de l'entité.

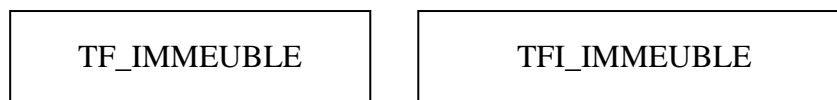


Figure 31 : Table des versions et Index des entités

2.5.2.1.4. *Référence vers un objet dont le numéro Amalfi est momentanément inconnu*

On peut se trouver dans une situation où l'on persiste un objet en référençant un autre avant que ce dernier ne soit lui-même persisté. En effet, en présence d'une grappe d'objets, on ne peut garantir qu'il est possible de trouver un ordre de persistance pertinent.

Pour cette raison une référence vers un objet peut être nulle en base de donnée, même si cela est impossible d'un point de vue fonctionnel. Cette tolérance suppose que le mécanisme qui persistera un groupe d'objets le fasse en 2 passes :

- ✓ La première passe persiste tous les objets proprement dits en ignorant les références
- ✓ La seconde passe met à jours les attributs dans les objets

Ce mécanisme ne s'applique pas aux références vers les items d'énumération, car ceux-ci, faisant partie du référentiel, sont réputés exister avant tous les autres objets.

2.5.2.1.5. *Concept d'objet temporaire*

Certains objets ont un cycle de vie qui peut avoir deux étapes : la création, dite à l'état temporaire, puis la publication. Typiquement, un objet à l'état temporaire est un objet créé en saisie, modifiable directement (sans qu'il soit besoin de le versionner). Un objet peut avoir deux états : temporaire ou publié

- ✓ Un objet temporaire peut référencer d'autres objets temporaires ou non
- ✓ Un objet publié ne peut référencer que d'autres objets publiés

En principe il peut exister des objets temporaires versionnables et des objets temporaires non versionnables. Dans la pratique, le concept d'objet temporaire ne s'applique qu'à des objets qui sont aussi versionnables, c'est à dire les objets du Livre Foncier.

En dehors du fait qu'il est temporaire ou non, un objet a les mêmes attributs. En revanche, il y a une différence fonctionnelle profonde entre un objet temporaire et un objet publié. On sait en particulier qu'un objet publié ne peut être en principe ni modifié, ni supprimé. Le respect

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			64/7 7

de ces contraintes fortes doit pouvoir se faire, le cas échéant, au niveau de la base de donnée. C'est pourquoi, la table dans laquelle on stocke un objet qui peut être temporaire est représentée physiquement par deux tables : la table des objets temporaires et celle des objets publiés.

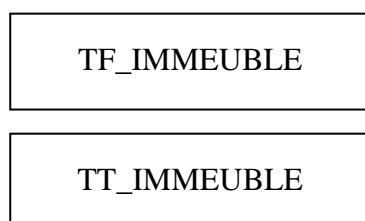


Figure 32 : Concept d'objet temporaire

2.5.2.1.6.

Objet versionnable et temporaire

Lorsque les deux concepts précédents s'appliquent, il est donc nécessaire de prévoir quatre tables.

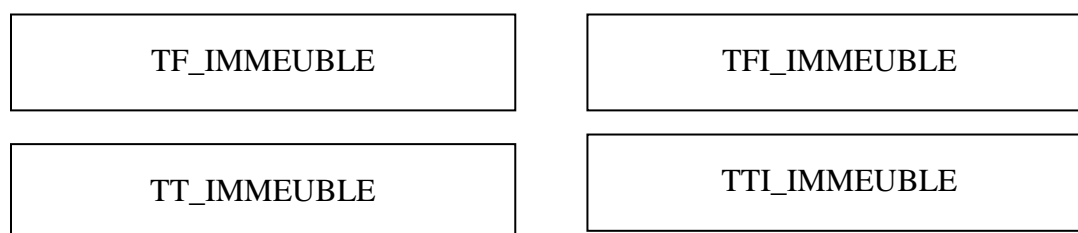


Figure 33 : Les objets versionnables et temporaires

2.5.2.1.7.

Concurrence d'accès

Pour interdire la modification d'un même objet par deux utilisateurs, on utilisera la technique du lock optimiste : un numéro de modification est stocké avec chaque objet. Ce numéro de modification est chargé chaque fois qu'un objet est lu en base. Lors de la modification de l'objet :

- ✓ d'une part on incrémente le numéro de modification
- ✓ d'autre part on modifie les données si et seulement si le numéro de modification n'a pas changé depuis la lecture. Plus précisément une clause de type WHERE d'égalité sur le numéro de modification est intégrée à l'ordre de mise à jour

Ainsi un deuxième utilisateur ne pourra pas modifier un objet sans avoir pris en compte les modifications intermédiaires.

En cas de concurrence d'accès, l'erreur est tracée dans les logs applicatif et indique l'objet à l'origine de la concurrence.

2.5.2.1.8.

Objets du référentiel

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			65/7 7

Les objets du référentiel sont classés par famille ou encore énumération (les circonscriptions foncières, les bureaux fonciers, les civilités, ...).

Un élément d'une énumération donnée est un item de l'énumération. Il a un code qui est unique pour l'énumération considérée. Un item d'énumération est versionnable.

2.5.2.1.9. *Référence d'un item vers des items d'autres énumérations*

Un item d'énumération peut en référencer jusqu'à deux autres items. Par exemple :

- ✓ Un bureau foncier ou une commune référencent un département
- ✓ Un bureau foncier référence un tribunal d'instance

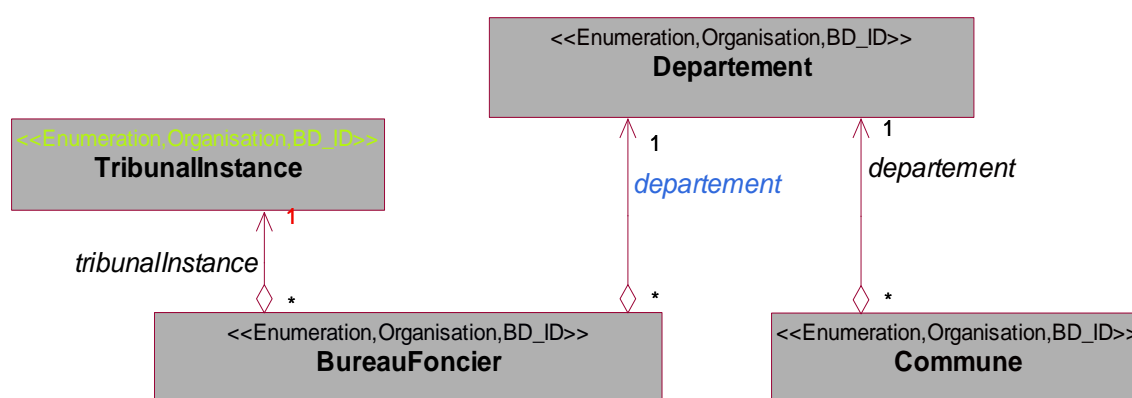


Figure 34 : Référence d'un item vers des items d'autres énumérations

2.5.2.1.10. *Collection d'items référençant un item donné*

Lorsqu'un item est référencé par d'autres items, ce dernier item donne l'accès à la collection de ces items le référençant. Cette collection est construite en mémoire uniquement. Ainsi, dans l'exemple précédent, on a :

- ✓ Pour un item de Tribunal d'instance, la collection des bureaux fonciers de ce tribunal
- ✓ Pour un item département, la collection des bureaux fonciers de ce département ainsi que la collection des communes de ce département

2.5.2.2. Organisation des données dans la base

2.5.2.2.1. *Objets Lf*

Pour chaque objet Lf, on trouvera au moins les éléments suivants en base de données.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			66/7 7

- ✓ La table des versions publiées : les données de cette table sont fournies par l'application (tables TF_*)
- ✓ La table des versions temporaires : les données de cette table sont fournies par l'application (tables TT_*)
- ✓ L'index des entités publiées : c'est une table technique mise à jour par un déclencheur (trigger) associé à la table des versions publiées (tables TFI_*)
- ✓ L'index des entités temporaires : c'est une table technique mise à jour par un déclencheur (trigger) associé aux deux tables des versions (tables TTI_*)
- ✓ Dans le cas où il est nécessaire de regrouper plusieurs tables (exemples : regroupement des personnes simples et des personnes agrégats, ainsi que le regroupement de tous les objets Lf), on définit des tables Master qui regroupent ces versions. La liste complète est la suivante :
 - TFM_OBJETLF: contient toutes les versions de tous les objets LF publiés
 - TFM_PERSONNE: contient toutes les versions de toutes les objets LF de type Personne publiés
 - TTM_OBJETLF : contient toutes les versions de tous les objets LF (publiés et temporaires)
 - TTM_PERSONNE: contient toutes les versions de toutes les objets LF de type Personne (publiés et temporaires)

2.5.2.2.2.

Utilisation d'un discriminant

Dans certains cas, on souhaite stocker dans la base de données des objets de classes différentes les uns des autres mais qui héritent d'une même classe mère. Par exemple, les objets Parcelle, Lot et Volume sont tous stockés dans la table des Immeubles. Une colonne particulière, DISCR, sert à identifier les objets les uns des autres. Cet attribut est un attribut technique. Il sert lors du chargement ou lors de la recherche d'un immeuble au sens large. Cette colonne n'est pas rappelée ultérieurement dans ce document.

2.5.2.2.3.

Discriminant d'un objet référencé

Lorsqu'un objet métier référence un autre objet métier pour lequel un discriminant est nécessaire, la valeur du discriminant est incluse dans la référence. Ainsi on pourra construire l'identifiant d'un objet référencé en trouvant les informations dans la table de l'objet référençant. La classe métier de l'objet référencé sera la vraie classe et non une classe abstraite intermédiaire.

2.5.2.2.4.

Timestamp et User technique

Chaque enregistrement dans la base est marqué des informations techniques suivantes :

- ✓ TEC_TS : timestamp de la dernière modification

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			67/7 7

- ✓ TEC_UID : identifiant de l'utilisateur ayant opéré la dernière modification

Ces colonnes techniques ne sont pas rappelées ultérieurement dans le document.

2.5.2.2.5. *Table des versions publiées*

La table des versions publiées d'un objet Lf contient les colonnes suivantes :

- ✓ NUM_LOCK, qui contient le numéro de modification de l'objet, et sur lequel s'appuie le mécanisme de lock optimiste
- ✓ NUM_AMALFI qui contient le numéro Amalfi de l'objet LF
- ✓ DATE_DEBUT, timestamp qui contient la date technique de création de la version
- ✓ DATE_FIN, timestamp qui contient la date de fin de validité d'une version. Cette colonne est utilisée pour la consultation de l'historique
- ✓ Plusieurs colonnes communes à tous les objets Lf, typiquement :
 - DATE_DEPOT (colonne fonctionnelle)
 - DATE_DEPOT_MAJ (colonne fonctionnelle)
 - DATE_PUBLICATION (colonne fonctionnelle)
 - DATE_PUBLICATION_MAJ (colonne fonctionnelle)
 - IS_RADIE (colonne fonctionnelle)
 - COMPLEMENT (colonne fonctionnelle)
- ✓ Plusieurs colonnes correspondant aux données spécifiques de l'objet Lf considéré

La table des versions publiées a pour clef primaire le couple (Numéro Amalfi, Date de début).

TF_DROIT		
NUM_LOCK	SMALLINT	
<u>NUM_AMALFI</u>	<u>VARCHAR(15)</u>	<pk>
<u>DATE_DEBUT</u>	<u>TIMESTAMP</u>	<pk>
DATE_FIN	TIMESTAMP	
DATE_DEPOT	TIMESTAMP	
DATE_DEPOT_MAJ	TIMESTAMP	
DATE_PUBLICATION	TIMESTAMP	
DATE_PUBLICATION_MAJ	TIMESTAMP	
COMPLEMENT	VARCHAR(200)	
IS_RADIE	CHARACTER(1)	
ATTRIBUT_SPECIFIQUE_1	<Undefined>	
ATTRIBUT_SPECIFIQUE_2	<Undefined>	
...	<Undefined>	

Figure 35 : Exemple de table de versions publiées

2.5.2.2.6. *Table des versions temporaires*

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			68/7 7

La table des versions temporaires d'un objet LF a exactement le même contenu que la table des versions publiées de cet objet Lf.

TT_DROIT		
NUM_LOCK	SMALLINT	
<u>NUM_AMALFI</u>	<u>VARCHAR(15)</u>	<pk>
<u>DATE_DEBUT</u>	<u>TIMESTAMP</u>	<pk>
DATE_FIN	TIMESTAMP	
DATE_DEPOT	TIMESTAMP	
DATE_DEPOT_MAJ	TIMESTAMP	
DATE_PUBLICATION	TIMESTAMP	
DATE_PUBLICATION_MAJ	TIMESTAMP	
COMPLEMENT	VARCHAR(200)	
IS_RADIE	CHARACTER(1)	
ATTRIBUT_SPECIFIQUE_1	<Undefined>	
ATTRIBUT_SPECIFIQUE_2	<Undefined>	
...	<Undefined>	

Figure 36 : Exemple de table de versions temporaires

2.5.2.2.7.

Table Index des entités publiées

L'index des entités publiées d'un objet contient les deux colonnes suivantes :

- ✓ NUM_AMALFI qui contient le numéro Amalfi de l'objet LF
- ✓ DATE_DEBUT, timestamp qui contient la date technique de création de la version

La clef primaire est le numéro Amalfi.

TFI_DROIT		
<u>NUM_AMALFI</u>	<u>VARCHAR(15)</u>	<pk>
<u>DATE_DEBUT</u>	<u>TIMESTAMP</u>	

Figure 37 : Exemple de table index d'entités publiées

L'insertion dans l'index des entités publiées se fait via un déclencheur associé à l'insertion d'une nouvelle donnée dans la table des versions publiées. Pour chaque nouvelle version, on réalise :

- ✓ La mise à jour de la date début dans l'index des entités publiées s'il existe déjà une entité pour le numéro Amalfi de la nouvelle version
- ✓ La création de l'entité dans l'index, si elle n'existe pas encore, par insertion du numéro Amalfi et de la date début de la nouvelle version

2.5.2.2.8.

Table Index des entités temporaires

L'index des entités temporaires d'un objet Lf a exactement le même contenu que l'index des entités publiées de cet objet Lf.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			69/7 7

TTL_DROIT		
<u>NUM_AMALFI</u>	<u>VARCHAR(15)</u>	<pk>
<u>DATE_DEBUT</u>	<u>TIMESTAMP</u>	

Figure 38 : Exemple de table index d'entités temporaires

Cette table accueille toutes les entités d'un objet Lf, publiées ou temporaires¹. L'insertion dans l'index des entités temporaires se fait via deux déclencheurs :

- ✓ un déclencheur² associé à l'insertion d'une nouvelle donnée dans la table des versions publiées
- ✓ Un déclencheur associé à l'insertion d'une nouvelle donnée dans la table des versions temporaires

Chacun de ces deux déclencheurs mettent à jour la date de début de l'entité si celle-ci existe déjà.

2.5.2.2.9.

Master de toutes les versions

Ces masters ont pour objectifs principaux :

- ✓ de permettre l'accès à une version donnée d'un objet Lf, indépendamment de la connaissance de son type (recherche d'une personne quelconque dans le master des personnes, recherche d'un objet Lf quelconque dans le master des objets Lf)
- ✓ de permettre l'accès à la dernière version d'un objet Lf, indépendamment du fait qu'elle soit publiée ou non et indépendamment du type de l'objet

Chaque master a pour colonnes :

- ✓ DTDEB, timestamp qui contient la date technique de création de la version
- ✓ NUM_AMALFI, qui contient le numéro Amalfi de l'objet LF
- ✓ DTFIN, timestamp qui contient la date d'invalidation de la version
- ✓ NUMLOCK pour le lock optimiste

Une fois la version identifiée, la connaissance du discriminant permet de trouver la table des versions correspondant à l'objet et qui contient les attributs spécifiques.

¹ Son nom est donc ambiguë puis qu'elle contient des objets non temporaires.

² Il peut s'agir du même déclencheur que celui utilisé pour la table des masters publiés.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			70/7 7

2.5.2.2.10.

Division d'une table en sous tables

Dans certains cas, on souhaite répartir les différentes sous-classes d'une classe principale dans des tables différentes. Par exemple, les personnes simples (Personne Physique, Société, ...) seront dans une première table, tandis que les personnes « multiples » seront dans une seconde table.

Dans ce modèle, on aura :

- ✓ La table des versions de personnes simples publiées
- ✓ La table des versions de personnes simples temporaires
- ✓ La table des versions de personnes agrégats publiées
- ✓ La table des versions de personnes agrégats temporaires
- ✓ L'index des entités personnes simples publiées
- ✓ L'index des entités personnes simples temporaires
- ✓ L'index des entités personnes agrégats publiées
- ✓ L'index des entités personnes agrégats temporaires
- ✓ Le master des personnes publiées
- ✓ Le master des personnes temporaires (incluant les personnes publiées)

De même on trouvera :

- ✓ Un déclencheur sur chacune des 2 tables de versions de personnes publiées qui alimente l'index des entités publiées
- ✓ Un déclencheur sur chacune des 4 tables de versions qui alimente l'index des entités temporaires

2.5.2.2.11.

Objets du référentiel

Pour chaque famille d'objets du référentiel, c'est-à-dire chaque énumération, on trouvera au moins les éléments suivants en base de données :

- ✓ La table des versions de l'énumération (TE_*). Les données de cette table sont fournies par l'application
- ✓ L'index des entités de l'énumération (TEI_*). C'est une table technique mise à jour par un déclencheur (trigger) associé à la table des versions de l'énumération

2.5.2.2.12.

Table des versions

La table des versions publiées d'une énumération contient les colonnes suivantes :

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			71/7 7

- NUM_LOCK, qui contient le numéro de modification de l'objet, et sur lequel s'appuie le mécanisme de lock optimiste
- CODE, code unique pour un item donné
- DATE_DEBUT, timestamp qui contient la date technique de création de la version de l'item
- DATE_FIN, timestamp qui contient la date de fin de validité d'une version d'un item. Cette colonne est utilisée pour la consultation de l'historique
- LIBELLE_COURT
- LIBELLE_LONG
- REF_PARENT1, optionnel, qui contient une référence vers un autre énuméré
- REF_PARENT2, optionnel, qui contient une référence vers un autre énuméré
- ORDRE, un entier pour classer les items de l'énumération lorsque plusieurs d'entre eux sont chargés par une même requête

Dans le cas où une énumération en référence une autre, c'est-à-dire lorsqu'une version d'item en référence un autre, ce référencement se fait indépendamment de la version, c'est-à-dire uniquement via le code de l'item référencé. Il existe forcément une contrainte de type clef étrangère entre la table des versions de l'énumération vers la table des masters référencés.

TE_CIVILITE		
NUM_LOCK	SMALLINT	
<u>CODE</u>	<u>VARCHAR(15)</u>	<pk>
<u>DATE_DEBUT</u>	<u>TIMESTAMP</u>	<pk>
DATE_FIN	TIMESTAMP	
LIBELLE_COURT	VARCHAR(24)	
LIBELLE_LONG	VARCHAR(128)	
ORDRE	INTEGER	

Figure 39 : Exemple d'énumération sans référence

TE_BUREAU_FONCIER		
NUM_LOCK	SMALLINT	
CODE	VARCHAR(15)	
DATE_DEBUT	TIMESTAMP	
DATE_FIN	TIMESTAMP	
LIBELLE_COURT	VARCHAR(24)	
LIBELLE_LONG	VARCHAR(128)	
ORDRE	INTEGER	
REF_Tribunal_Instance	VARCHAR(15)	
REF_Département	VARCHAR(15)	

Figure 40 : Exemple d'énumération avec référence(s)

Il y a dans ce cas deux contraintes de type clef étrangère sur la table TE_BUREAU_FONCIER :

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			72/7 7

- ✓ La référence vers le tribunal d'instance est un code clef de la table des masters de tribunaux d'instance
- ✓ La référence vers le département est un code clef de la table des masters des départements

2.5.2.2.13.

Objets du registre des dépôts

Les objets du registre des dépôts sont des objets auxquels ne s'appliquent ni le concept de versionnement, ni le concept d'objet temporaire. Parmi les principaux, on peut citer ActeGestion ou Requete. Pour chacun d'eux, il existe une et une seule table. Exemple : la table TR_REQUETE pour les objets Requete. Les tables du registre des dépôts contiennent au moins les colonnes suivantes :

- ✓ OID : clef de l'objet dans la table
- ✓ DISCR : discriminant des différents types d'objet dans la table

En général les liens des objets du registre des dépôts vers le livre foncier se font vers des versions. Typiquement, un acte de gestion référence des versions précises d'objet Lf.

2.5.2.3. Module d'accès à la persistance

Le module d'accès à la persistance fournit et implémente les méthodes d'accès à la persistance. Ce module comporte :

- ✓ Le composant proprement dit d'accès, qui définit les méthodes d'accès
- ✓ Les composants permettant l'implémentation, à savoir :
 - Le gestionnaire de persistance
 - Les daos

2.5.2.3.1.

Les méthodes d'accès à la persistance

Ces méthodes accomplissent les fonctions suivantes :

- ✓ Allocation d'une clef pour un nouvel objet métier
- ✓ Chargement par sa clef d'un objet métier
- ✓ Persistance de un ou plusieurs objets métier. Persister signifie ici insérer, mettre à jour ou supprimer un objet
- ✓ Recherche d'objets métiers en fonction de critères donnés

Ces méthodes d'accès à la persistance s'appuient, le cas échéant sur le périmètre de persistance et la sécurité applicative. Le périmètre de persistance indique s'il faut considérer :

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			73/7 7

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP- ste	IBM puis ASTEK			74/7 7

Comme toutes les classes métiers, elles ont un DAO pour l'accès base. Ici, le DAO est un peu spécial dans la mesure où il utilise l'API ICM (IBM Content Manager) plutôt que JDBC. Lors de la lecture dans la GED, c'est la sécurité Applicative qui donne le BF de localisation de l'utilisateur.

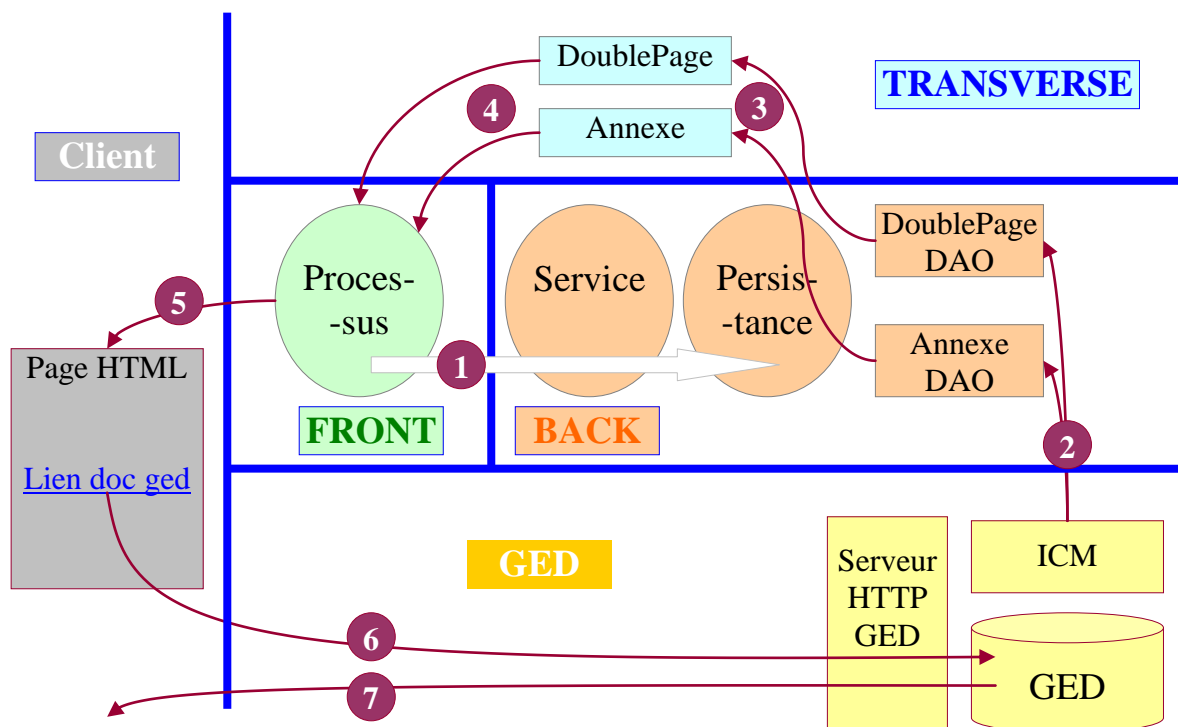


Figure 41 : synthèse simplifiée de l'accès GED pour le SC.

Dans l'environnement SSEE, ce principe ne fonctionne pas, en effet l'URL fournie pour le lien vers le document de la GED pointe vers un serveur qui se trouve du côté SC qui rend le document inaccessible depuis le poste client.

Le principe de communication SSEE à base de file d'attente JMS est donc réutilisé. :

- ✓ L'URL originale du document GED est obtenue de la même façon que dans le SC.
- ✓ L'URL placé dans la page de l'utilisateur est l'URL d'un servlet du Front SSEE.
- ✓ Ce servlet poste dans une file requête cette URL,
- ✓ Le servlet se met en attente d'un message sur une file réponse.
- ✓ Un EJB MDB est activé par le message dans la file requête, il télécharge le document de la GED.
- ✓ L'EJB MDB positionne une table d'octet représentant le document dans une file réponse.
- ✓ Le servlet prend le document dans la file réponse et l'envoi au poste client.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			75/7 7

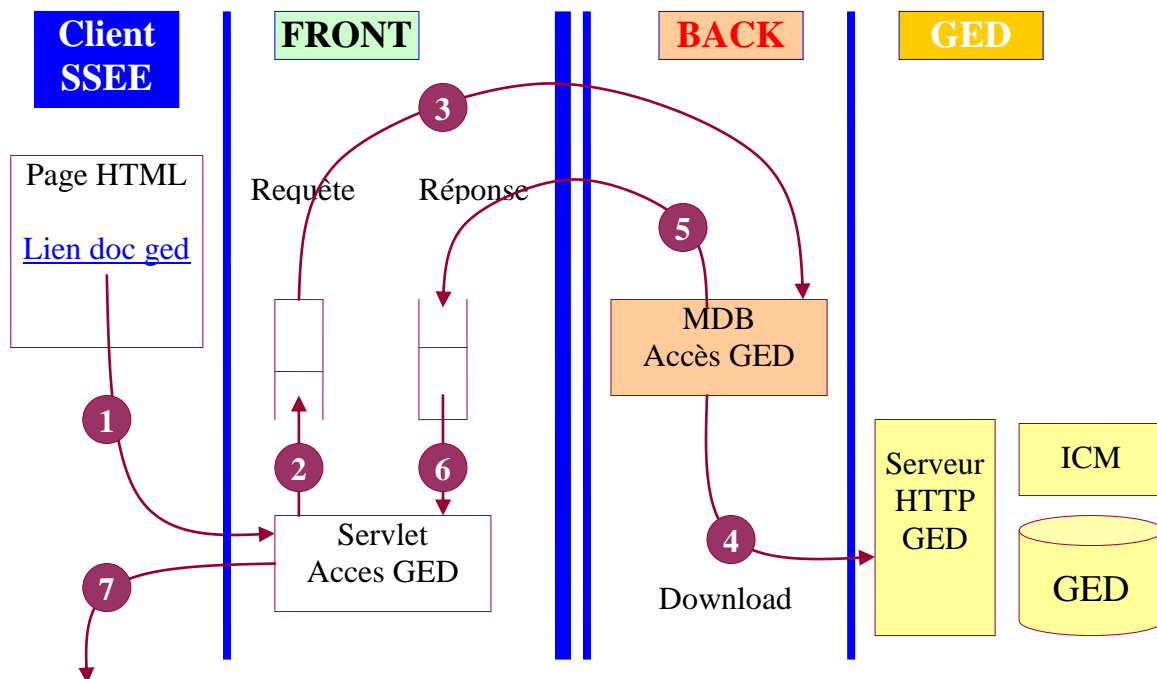


Figure 42 : synthèse simplifié de l'accès GED pour le SSEE.

2.7. Le contrôle des performances

Reprise du document d'architecture générale.

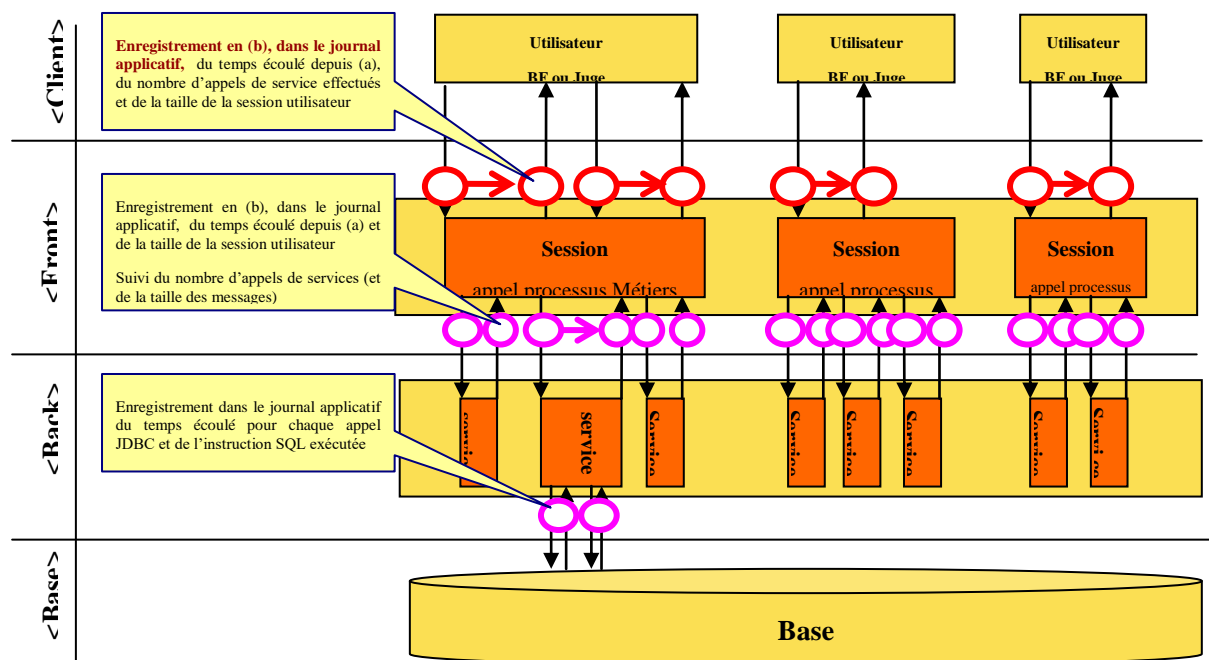


Figure 43 : Le contrôle des performances

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			76/7 7

Le contrôle des performances est une des problématiques que l'équipe de conception de l'architecture applicative d'Amalfi V2 a voulu prendre en compte tout au long des développements applicatifs (tout comme les tests unitaires).

Plusieurs indicatifs sont tracés tout au long de la vie de l'application comme :

- ✓ le temps écoulé pour un appel de service
- ✓ le nombre d'appels de services effectués dans un processus et de la taille des messages
- ✓ la taille de la session
- ✓ le temps de chaque appel JDBC

Tous ces indicateurs doivent permettre de garder la maîtrise sur les performances durant les développements afin de ne pas avoir trop de surprises durant les tests en pré-production et surtout en production.

Référence	Auteur			Page
TMA-8.1.2.2.1 - ARCHI_APP-ste	IBM puis ASTEK			77/7 7