Université Polytechnique de Bobo-Dioulasso
(UPB)

Ecole Supérieure d'Informatique
(ESI)

Cycle des Ingénieur de Travaux Informatiques
(CITI)
(Option Analyse et Programmation)
---
01 BP 1091 Bobo-Dioulasso 01
Tél. : (226) 20 98 06 65
Fax : (226) 20 97 27 77

Burkina Faso
Unité – Progrès - Justice

Zongos Consulting & Productions
06 B.P 10048 Ouagadougou 06
Tél.:+226 50 33 25 47
+226 50 30 04 99
Fax:+226 50 33 38 61
Rue de la Palestine
Porte n°403,1er étage.

PROJET DE FIN DE CYCLE

Année académique 2005-2006

Migration vers une architecture 3-tier respectant le modèle MVC de l'application web de gestion des règlements pécuniaires des avocats du Burkina (GESTCARPA)

CONCEPTION DETAILLEE

Groupe de Projet
Abdoulaye ZANGO
Yacouba OUATTARA
Etudiants à l'ESI

Maître de stage
Mme Rasmata COMPAORE
Service Développement ZCP Informatique

Superviseur
Yacouba OUATTARA
Enseignant à l'ESI

DEDICACE

Nous dédions ce modeste travail à :

nos parents, pour leurs soutiens sans faille ;

nos tuteurs, pour leur encouragements ;

nos amis, pour leurs disponibilités ;

toute la communauté de l'Ecole Supérieure d'Informatique.

REMERCIEMENTS

Mme Rasmata COMPAORE
Chef du service formation et développement ZCP Informatique, notre maître de stage pour sa disponibilité, son soutien et son encadrement technique ;

M. Sirima Yaya
M. Ismael TASSEMBEODO
Informaticiens, développeurs à ZCP Informatique, pour leur assistance durant le stage ;

M. Sylvain ZONGO
Directeur de ZCP Informatique, pour nous avoir permis de réaliser notre projet dans son entreprise ;

Toute l'équipe de ZCP Informatique, pour leur disponibilité ;

M. Yacouba OUATTARA
Enseignant à l'ESI, notre superviseur ;

Mme Marie-Claude Viguier-Martinez
Conseiller du Président de l'UPB – Projet RESEAU, pour ses conseils ;

L'Ecole Supérieure d'Informatique (ESI), pour la formation reçue durant ces trois ans ;

Tous ceux qui de manière quelconque ont contribués à la réalisation de ce travail.

Merci à tous.

TABLE DES MATIERES

INTRODUCTION

Le dossier de conception préliminaire du projet nous a permis de mieux cerner le problème posé et de faire des propositions de solutions. Après une analyse critique de chacune d'elles, une solution de mise en oeuvre a été retenue.

La présente phase de conception détaillée sanctionnée par un **dossier de conception détaillée** nous permettra de construire et documenter en détail cette solution afin de lui donner une image "*prêt à coder*".

Le contenu de ce dossier portera essentiellement sur la présentation des langages et outils de mise en oeuvre, la description de la conception des différentes couches logicielles et la présentation des aspects techniques liés aux politiques de secours et de sécurité pour un meilleur fonctionnement du système futur.

I. GENERALITES

1. RAPPELS SUR LE SUJET

a. La CARPA-BF et GESTCARPA

La Caisse Autonome de Règlement Pécuniaire des Avocats du Burkina Faso (CARPA-BF) est une association des avocats du Burkina Faso dont l'objectif principal est la sécurisation des fonds, effets ou valeurs des tiers transitant entre les mains des avocats dans l'exercice de leur profession.

Ces fonds ne sont détenus qu'à titre temporaire par l'avocat qui doit les transmettre à son client ou à l'adversaire, et cela soit au titre d'exécution d'une décision de justice, soit au titre d'une transaction.

La sécurisation de ces fonds, suscitée par différents mouvements (dépôts et retraits) sur les comptes des entités liées à une affaire se fait grâce à un outil logiciel : GESTCARPA.

GESTCARPA est une application web développée par Zongo's Consulting & Production (ZCP), un cabinet de prestations informatiques.

b. Problématique

La réalisation de l'objectif principal pour la CARPA-BF, qui est le renforcement de la sécurisation des fonds des tiers transitant entre les mains des avocats, a été atteinte et des résultats en témoignent[1]. Cependant le fonctionnement très concentré à Ouagadougou du domaine remet en cause certains objectifs. Tous les avocats du Burkina Faso n'ont pas un accès facile aux services de la CARPA-BF.

Par conséquent, il faudra proposer une solution qui permettra de mettre en place une nouvelle organisation orientée vers l'ouverture du système et aussi une solution informatique (logiciel) flexible et robuste basée sur celle existante et qui pourra favoriser les évolutions futures à quelque que niveau que ce soit (présentation, traitement des données, etc.).

[1] Le payement en 2006 des fonds dans l'affaire des 500 militaires contre l'Etat burkinabé a été géré par la CARPA-BF.

2. PRESENTATION DU SUJET

a. Présentation du thème

Le thème soumis à notre étude s'intitule : "**Migration vers une architecture 3-tier respectant le modèle MVC de l'application de gestion des règlements pécuniaires des Avocats du Burkina (GESTCARPA)**".

Comme l'indique le thème, il s'agira pour le groupe de projet de mener une étude sur les conditions de la mise en place d'une solution logicielle orientée objet, réalisant au moins les mêmes fonctionnalités que GESTCARPA et basée sur une architecture 3-tier particulièrement le modèle MVC.

La mise en place de cette solution avec ce modèle permettra tout d'abord une bonne répartition des composants du système afin de favoriser :

- une facilité dans sa maintenance et dans son évolution (ajout de nouvelles fonctionnalités) ;
- une ouverture du système à l'ensemble des utilisateurs du système.

Nous reviendrons plus en détails dans les points suivants sur l'architecture 3-tier et modèle MVC.

b. L'architecture 3-tier

- **Définition et concepts**

L'architecture 3-tier est un modèle logique d'architecture applicative qui vise à séparer très nettement trois couches logicielles au sein d'une même application ou système, à modéliser et présenter cette application comme un empilement de trois couches, étages ou niveaux dont le rôle est clairement défini :

- la **présentation** des données : correspondant à l'affichage, la restitution sur le poste de travail, le dialogue avec l'utilisateur;
- le **traitement** métier des données : correspondant à la mise en œuvre de l'ensemble des règles de gestion et de la logique applicative;

- et enfin l'**accès aux données** persistantes : correspondant aux données qui sont destinées à être conservées sur une durée ou de manière définitive (stockage sur un support physique).

Dans cette approche, les couches communiquent entre elles au travers d'un « modèle d'échange », et chacune d'entre elles propose un ensemble de services rendus. Les services d'une couche sont mis à disposition de la couche supérieure. On s'interdit par conséquent qu'une couche invoque les services d'une couche plus basse que la couche immédiatement inférieure ou plus haute que la couche immédiatement supérieure (chaque niveau ne communique qu'avec ses voisins immédiats).

Le rôle de chacune des couches et leur interface de communication étant bien définis, les fonctionnalités de chacune d'entre elles peuvent évoluer sans induire de changement dans les autres couches. Cependant, une nouvelle fonctionnalité de l'application peut avoir des répercussions dans plusieurs d'entre elles. Il est donc essentiel de définir un modèle d'échange assez souple, pour permettre une maintenance aisée de l'application.

Ce modèle d'architecture 3-tier a pour objectif de répondre aux préoccupations suivantes :

- allégement du poste de travail client (notamment vis à vis des architectures classiques client-serveur de données - typiques des applications dans un contexte Oracle/Unix) ;
- prise en compte de l'hétérogénéité des plates-formes (serveurs, clients, langages, etc.) ;
- introduction de clients dits « légers » (plus liée aux technologies Intranet/HTML qu'au 3-tier proprement dit) ;
- et enfin, meilleure répartition de la charge entre différents serveurs d'application.

Précédemment, dans les architectures client-serveur classiques (c'est le cas de GESTCARPA), les couches présentation et traitement étaient trop souvent imbriquées. Ce qui posait des problèmes à chaque fois que l'on voulait modifier l'IHM[2] du système.

- **Les trois (03) couches**
  - **Couche Présentation (premier niveau)**

Elle correspond à la partie de l'application visible et interactive avec les utilisateurs. On parle alors d'Interaction Homme Machine. En informatique, elle peut être réalisée par une application graphique ou textuelle. Elle peut aussi être représentée en HTML pour être exploitée par un navigateur Web.

La couche présentation relaie les requêtes de l'utilisateur à destination de la couche métier, et en retour lui présente les informations renvoyées par les traitements de cette couche. Il s'agit donc ici d'un assemblage de services métiers et applicatifs offerts par la couche inférieure.

  - **Couche Métier / Business (second niveau)**

Elle correspond à la partie fonctionnelle de l'application, celle qui implémente la « *logique* », et qui décrit les opérations que l'application va appliquer sur les données en fonction des requêtes des utilisateurs, effectuées au travers de la couche présentation. Les différentes règles de gestion et de contrôle du système sont mises en œuvre dans cette couche.

La couche métier offre des services applicatifs et métiers à la couche présentation. Pour fournir ces services, elle s'appuie, le cas échéant, sur les données du système, accessibles au travers des services de la couche inférieure. En retour, elle renvoie à la couche présentation les résultats qu'elle a calculés.

[2] IHM : Interaction Home Machine, étudie la façon dont les humains interagissent avec les ordinateurs ou entre eux à l'aide d'ordinateurs, ainsi que la façon de concevoir des systèmes informatiques qui soient ergonomiques.

Pour exemples, les services métier gérants les sous-comptes affaire enregistrés : les services de recherche d'une affaire, de modification d'une affaire, de suppression d'une affaire, de création d'un sous-compte affaire, de consultation du solde d'un sous-compte affaire,etc.

- **Couche d'accès aux données (troisième niveau)**

Elle consiste en la partie gérant l'accès aux gisements de données du système. Ces données sont pérennes, car destinées à durer dans le temps, de manière plus ou moins longue ou définitive (stockées sur un support physique).

Les données peuvent être stockées indifféremment dans de simples fichiers texte, ou eXtensible Markup Language (XML), ou encore dans une base de données. Quel que soit le support de stockage choisi, l'accès aux données doit être le même. Cette abstraction améliore la maintenance du système.

c. Le modèle MVC

Le modèle MVC (Modèle-Vue-Contrôleur) est étroitement lié à l'origine des langages à objets. De fait, dès le début des années 80, Smalltalk[3] appuyait son organisation IHM (Interface Homme Machine) sur ce modèle. Depuis, le modèle s'est largement répandu et est désormais connu et reconnu comme un modèle de conception très mature.

Le modèle MVC s'appuie essentiellement sur la séparation en deux couches verticales regroupant d'un côté les objets métiers (Modèle) et de l'autre les objets IHM, ces derniers étant eux-mêmes regroupés en objets chargés de l'acquisition d'informations en provenance de l'utilisateur (Contrôleur) et en objets chargés de la restitution d'informations vers l'utilisateur (Vue). Il est recommandé comme modèle pour plate-forme J2EE.

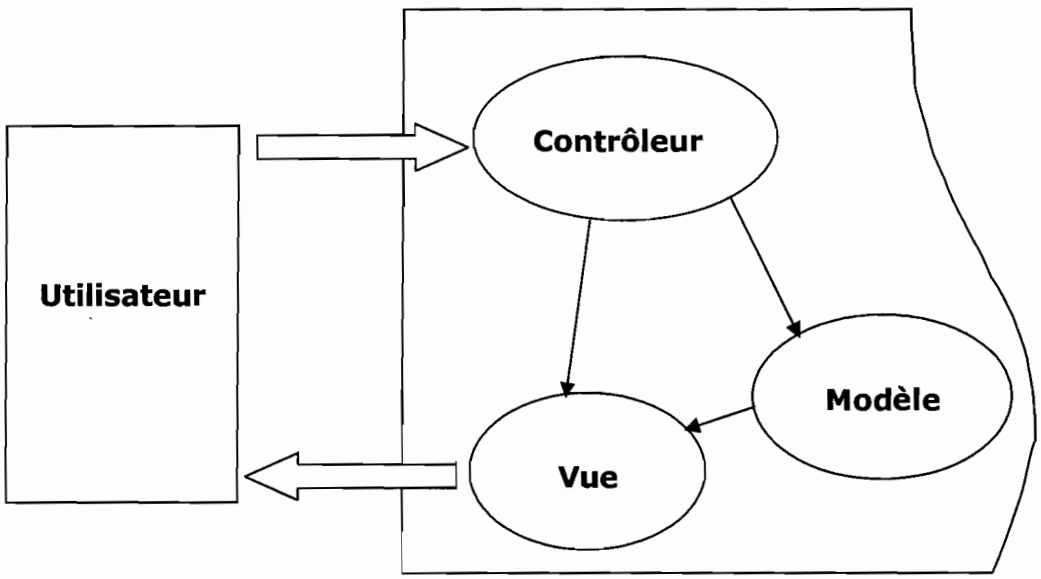


Figure I-2-c.1 : Le modèle MVC

[3] Smalltalk est l'un des premiers langages de programmation orienté objet. Il a été créé en 1972. Il est inspiré par Lisp et Simula. Il a été conçu par Alan Kay, Dan Ingals, Ted Kaehler, Adele Goldberg au Palo Alto Research Center de Xerox.

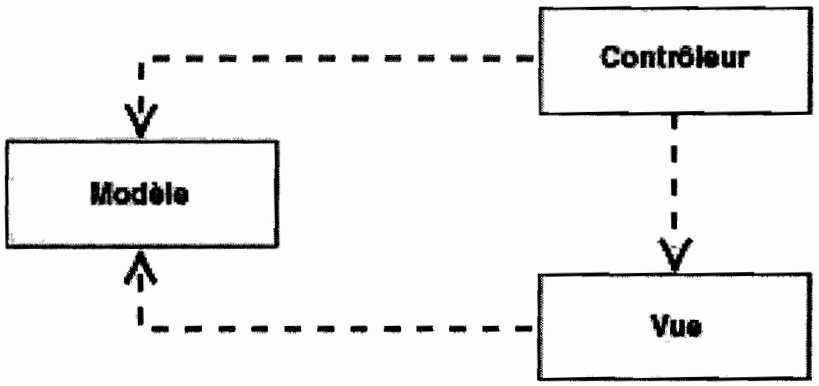
- **Fonctionnement du MVC**

MVC est un modèle de conception qui impose la séparation entre données, traitements et présentation. C'est pour cette raison que l'application est divisée en trois composants fondamentaux: le modèle, la vue et le contrôleur. Chacun de ces composants tient un rôle bien défini.

Ainsi, ce modèle se décompose en trois composants:

- **le Modèle :** décrit les données manipulées par l'application et définit les méthodes d'accès ;
- **la Vue :** définit l'interface utilisateur et la présentation ;
- **le Contrôleur :** prend en charge la gestion des événements de synchronisation pour mettre à jour la vue ou le modèle.

Le schéma suivant présente les relations structurelles entre les trois objets :



Il est important de noter que la vue et le contrôleur dépendent (le sens des flèches en pointillé) tous deux du modèle. En revanche, le modèle ne dépend ni de la vue ni du contrôleur. C'est l'un des principaux avantages de la séparation. Cette séparation des tâches permet de créer et de tester le modèle indépendamment de la représentation visuelle. La séparation entre la vue et le contrôleur est secondaire dans la plupart des applications clientes riches et, en effet, de nombreuses infrastructures d'interface utilisateur implémentent les rôles comme un seul objet.

Au contraire, dans les applications Web, la séparation entre la vue (le navigateur) et le contrôleur (les composants côté serveur gérant les requêtes HTTP) est clairement définie. Le modèle MVC est un modèle de conception fondamental en matière de séparation de la logique de l'interface utilisateur et de la logique métier.

- **Les principaux avantages du MVC**

La plupart des applications Web sont créées à l'aide de langages procéduraux tels que ASP ou PHP ce qui entraîne qu'on est rapidement tenté de mélanger traitements, données et présentation. Alors MVC impose la séparation entre les différentes parties.

La Vue correspond à l'interface avec laquelle l'utilisateur interagit. Auparavant, dans le cas des applications Web, il s'agissait d'une interface HTML et la gestion de toutes ces interfaces dans l'application devenait alors de plus en plus difficile. MVC permet de gérer l'utilisation de différentes vues. Cependant, aucun traitement n'est effectué dans celles-ci. Elles servent uniquement à afficher les données et permettre à l'utilisateur d'agir sur ces données.

Le deuxième composant du MVC, le Modèle, représente les données et les règles métiers. C'est à ce niveau que s'effectuent les traitements. Les bases de données et les objets tels que les EJB[4] en font partie. Les données renvoyées par le modèle sont indépendantes de la présentation, c'est-à-dire que le modèle ne réalise aucune mise en forme. Les données d'un seul modèle peuvent ainsi être affichées dans plusieurs vues. Cette capacité permet de factoriser le code, car le code du modèle n'est écrit qu'une seule fois puis réutilisé par toutes les vues.

De plus, comme la vue est séparée du modèle et qu'il n'y a pas de dépendance directe entre eux, l'interface utilisateur peut afficher simultanément plusieurs vues des mêmes données. Ainsi, plusieurs pages d'une application Web peuvent utiliser les mêmes objets de modèle. L'application Web permettant à l'utilisateur de modifier l'apparence des pages constitue un autre exemple des

[4] EJB : Enterprise Java Beans, composant J2EE pour la gestion des transactions et aussi de la persistance dans la couche M du modèle MVC.

avantages de cette séparation. Ces pages affichent les mêmes données à partir du modèle partagé mais ces données s'affichent de différentes manières. Aussi, la duplication du code est réduite puisqu'on a séparé les données et la logique métier de la présentation.

En outre, comme le modèle est autonome et séparé du contrôleur et de la vue, il est beaucoup plus facile de modifier la couche de données ou les règles métier. Par exemple si on change de bases de données et qu'on passe de MySQL à Oracle, il suffira simplement de revoir le modèle. Le fonctionnement interne des trois parties d'une application MVC est masqué aux autres parties. On peut ainsi élaborer des interfaces bien définies et des composants autonomes.

Un autre avantage important de MVC est bien entendu l'adaptation aux changements. Les exigences en matière d'interface utilisateur ont tendance à changer beaucoup plus rapidement que les règles métiers. Les utilisateurs peuvent préférer utiliser différentes couleurs, polices, disposition à l'écran et plusieurs niveaux de prise en charge des nouveaux périphériques tels que les téléphones mobiles ou les assistants personnels. Comme le modèle ne dépend pas des vues, l'ajout de nouveaux types de vues au système ne l'affecte généralement pas. En conséquence, le changement porte uniquement sur la vue.

Le rôle du Contrôleur est également important puisqu'il interprète les requêtes de l'utilisateur. Ainsi, lorsque l'utilisateur clique sur un lien ou soumet un formulaire HTML, le contrôleur ne produit rien et n'effectue aucun traitement. Il intercepte la requête et détermine uniquement quels modèles et quelles vues doivent êtres associés.

- **Quelques inconvénients**

MVC introduit de nouveaux stades d'indirection ce qui augmente légèrement la complexité de la solution. Il accroît également la nature événementielle du code de l'interface utilisateur. Il est indispensable de consacrer du temps à réfléchir aux interactions entre les différentes parties de l'application. De plus, la stricte séparation entre le modèle et la vue peut parfois rendre plus difficile le débogage. Chaque élément devra être soumis à des tests rigoureux avant son utilisation. La gestion des fichiers est aussi plus fastidieuse. Cela peut sembler évident vu que l'intérêt principal de MVC consiste à séparer les trois couches. Toutefois, la charge de travail supplémentaire induite par la gestion de multiples fichiers ne doit pas être sous-estimée.

Par conséquent, il en résulte que MVC est beaucoup trop complexe pour les petites applications. Une autre contrainte de l'utilisation de MVC est le coût des mises à jour fréquentes. En effet, dissocier le modèle de la vue ne signifie pas que les développeurs du modèle peuvent ignorer la nature de ces vues. Par exemple, si le modèle subit des modifications fréquentes, il peut envahir les vues de demandes de mise à jour. Et de ce fait, certaines vues, telles que les affichages graphiques, peuvent prendre du temps à s'afficher. La vue peut alors être décalée par rapport aux demandes de mise à jour. Il est donc important de penser à la vue lors du codage du modèle. Le modèle pourra, par exemple, regrouper plusieurs requêtes en une seule notification à la vue.

3. OBJECTIF DE LA CONCEPTION DETAILLEE

La phase de conception détaillée du projet est l'ultime phase d'analyse et de conception de notre système d'information, elle précède les phases de codage, test et recette[5]. Après la modélisation des besoins puis l'organisation de la structure de la solution retenue dans la phase de conception préliminaire, la conception détaillée consiste à construire et à documenter précisément les classes, les interfaces (vues), les tables et les méthodes qui constituent le codage de la solution.

Il s'agira de :

- comprendre le rôle d'UML pour la conception détaillée (utilisation des différents diagrammes UML) ;
- appliquer le micro-processus[6] utilisé pour bâtir une conception objet avec UML ;
- construire une solution pour chaque couche: présentation, application, métier, d'accès aux données et leur mode de stockage;
- transformer le modèle objet en modèle relationnel pour le stockage des données sur une base de données relationnelle: PostgreSQL ;
- décrire les aspects techniques que nous préconisons en terme de procédure transitoire, de secours et de sécurisation du nouveau système.

[5]La phase de recette consiste à valider les fonctions du système développé.
[6]Le micro-processus de conception logique concerne la définition des classes à implémenter. C'est donc une activité centrée sur le modèle logique, qui combine les diagrammes UML.

II. LANGAGE ET OUTILS DE REALISATION

1. LE LANGAGE Java

Java, syntaxiquement analogue au C++, est un langage de programmation orienté pur objet qui offre d'énormes potentialités pour le développement d'applications entreprises[7], à travers sa plate-forme J2EE intégrant des API[8] destinés à faciliter cela. Le plus remarquable des atouts de ce langage est le fait qu'il soit interprété indépendamment d'un système d'exploitation. Les applications Java peuvent donc être exécuté sur tous les systèmes d'exploitation équipés de la plate-forme JRE (Java Runtime Environment - Environnement d'exécution Java) constituée d'une JVM (Java Virtual Machine - Machine Virtuelle Java) ; le programme qui interprète le code Java et le converti en code natif.

Ce langage a donc été choisi pour le développement (conception, codage et test) du futur système de gestion des règlements pécuniaires au sein de la CARPA-BF.

2. LA PLATE-FORME J2EE

J2EE (Java 2 Enterprise Edition) est une plate-forme de développement d'applications entreprises multi-tier basée sur des composants standard et modulaires. C'est la version entreprise de la plate-forme "Java 2" qui se compose de l'environnement J2SE[9] ainsi que de nombreuses API et composants destinés à une utilisation côté serveur au sein du système d'information de l'entreprise.

J2EE est une norme qui va spécifier à la fois l'infrastructure de gestion de vos applications et les API des services utilisés pour concevoir ces applications. La plateforme J2EE est essentiellement un environnement fournissant :

- **un environnement d'exécution de J2EE** : un des avantages majeurs de J2EE est de faire abstraction de l'infrastructure d'exécution. En effet, J2EE

[7] Applications entreprises : applications distribuées au sein de l'entreprise qui interagissent par l'intermédiaire d'un réseau.
[8] Une Interface de programmation (en anglais Application Programming Interface ou API) définit la manière dont un composant informatique peut communiquer avec un autre.
[9] J2SE : Java 2 Standard Edition.

spécifie les rôles et les interfaces pour les applications, ainsi que l'environnement d'exécution dans lequel les applications sont déployées. Ceci permet aux développeurs d'application de ne pas avoir à reprogrammer les services d'infrastructure.

- **les API J2EE :** les API sont des outils logiciels, constitués d'un ensemble de modules dotés de fonctions communes, et qui permettent de produire automatiquement des applications Java ou des applets personnalisés.

3. LES API J2EE UTILISEES

a. Les Servlets et les pages JSP

Ce sont des composants réseaux destinés à une exécution orientée question/réponse[10]. La technologie JSP est une extension de la notion de Servlet permettant de simplifier la génération de pages web dynamiques. JSP est un concurrent direct de l'ASP et du PHP. Un Servlet est un composant coté serveur, écrit en Java, dont le rôle est de fournir une trame générale pour l'implémentation de paradigmes "requête / réponse". Ils remplacent les scripts CGI tout en apportant des performances bien supérieures.

b. Interface d'accès aux données : JDBC

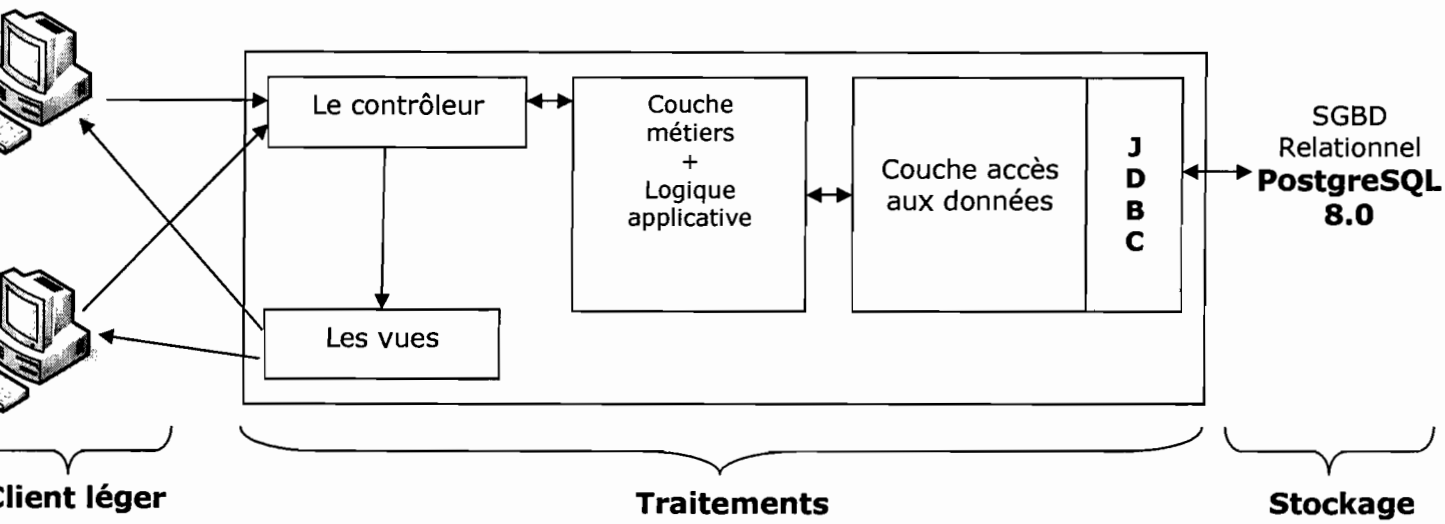
La technologie JDBC (*Java Database Connectivity*) est une API fournie avec Java (depuis sa version 1.1) permettant de se connecter à des bases de données, c'est-à-dire que JDBC constitue un ensemble de classes permettant de développer des applications capables de se connecter à des serveurs de bases de données (SGBD).

L'API JDBC a été développée de telle façon à permettre à un programme de se connecter à n'importe quelle base de données en utilisant la même syntaxe, c'est-à-dire que l'API JDBC est indépendante du SGBD.

[10] Questions du client et réponse du serveur

De plus, JDBC bénéficie des avantages de Java, dont la portabilité du code, ce qui lui vaut en plus d'être indépendant de la base de données d'être indépendant de la plate-forme sur laquelle elle s'exécute.

- **JDBC dans notre architecture J2EE (MVC)**



- **Utilisation de JDBC**

Les accès à une base de données avec l'API JDBC, doivent être effectués en sept (07) étapes consécutives.

1. ***Chargement d'un pilote JDBC***
2. ***Définition de l'url de connexion.***
3. ***Établissement de la connexion.***
4. ***Création d'une requête.***
5. ***Exécution de la requête.***
6. ***Traitement des résultats.***
7. ***Fermeture de la connexion.***

4. LE SERVEUR D'APPLICATION APACHE TOMCAT 5.X

Apache Tomcat est un serveur d'application libre qui agit comme un conteneur[11] de Servlet J2EE. Issu du projet Jakarta, Tomcat est désormais un projet principal de la fondation Apache.

Tomcat implémente les spécifications des Servlets et des JSP de Sun Microsystems. Il inclut des outils pour la configuration et la gestion, mais peut également être configuré en éditant des fichiers de configuration XML. Il supporte à ce titre l'ensemble des classes définies par Sun. Tomcat est totalement gratuit et généralement utilisé en couplage avec un serveur web Apache, il est aussi considéré comme un serveur HTTP.

Écrit en Java, il nécessite, pour pouvoir fonctionner, la présence d'une machine virtuelle Java, et plus précisément du SDK (Sun Developpement Kit complet). Ceci implique que Tomcat est totalement portable et peut être mis en oeuvre sur des systèmes radicalement différents, tels que Linux ou Windows.

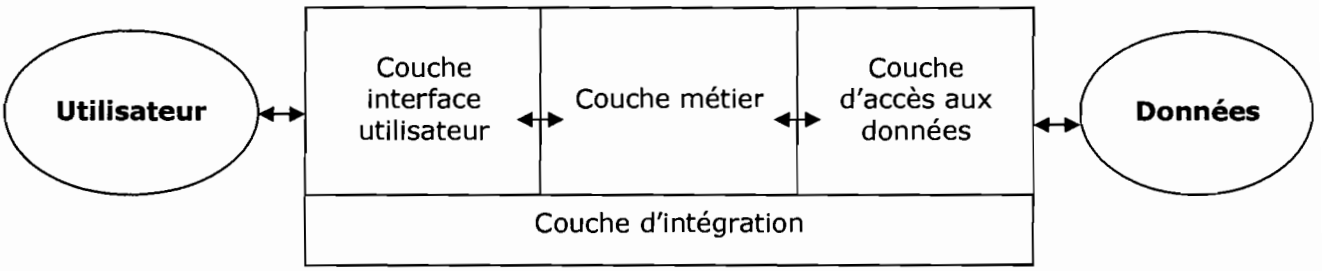
Nous utiliserons la version 5.0 de Tomcat comme serveur d'application pour ce projet. Cette version implémente :

- les spécifications Servlet 2.4 et JSP 2.0 ;
- ramasse-miettes[12] réduit, performances améliorées et stabilité ;
- wrappers[13] natifs Windows et Unix ;
- parsing JSP plus rapide par le compilateur de JSP Jasper.

[11] Environnement d'exécution des Servlets.
[12] Un ramasse-miettes, ou récupérateur de mémoire, ou glaneur de cellules (en anglais *garbage collector*, abrégé en GC) est un sous-système informatique de gestion automatique de la mémoire.
[13] Classe d'application programme « enveloppant » l'exécution d'un autre programme, pour lui préparer un environnement particulier

III. DEVELOPPEMENT MVC EN Java

Dans le développement logiciel en général et dans le développement web en particulier, on cherche à séparer nettement les couches présentation, traitement et accès aux données. L'architecture d'une application web est souvent la suivante :



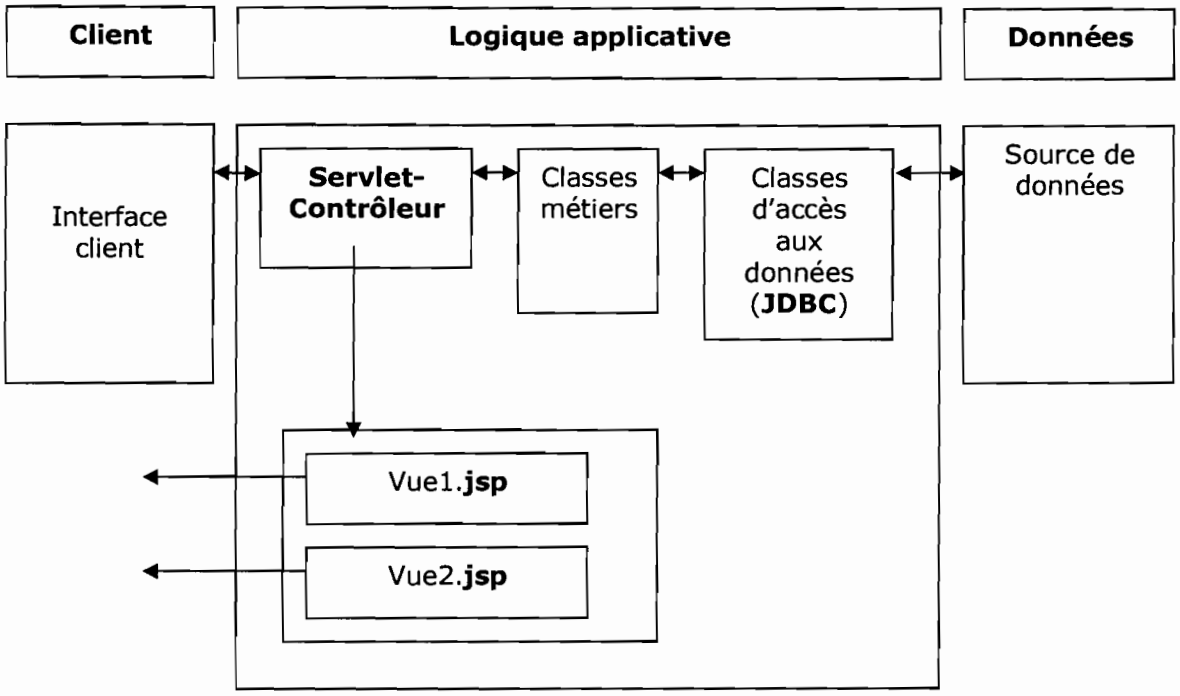
Une telle architecture, appelée 3-tier ou à 3 niveaux cherche à respecter le modèle **MVC** (Model View Controller).

L'architecture 3-tier sépare, tout comme MVC, l'application en trois parties bien distinctes :

- L'interface utilisateur : La partie présentation de l'application.
- La couche métier : La couche métier qui s'occupe du traitement de l'information.
- La couche d'accès aux données : La partie accès et stockage des données

Pour qu'une application 3-tier respecte le modèle MVC il faut lui ajouter une couche de contrôle entre l'interface utilisateur et la couche métier.

Pour l'architecture web J2EE du futur système, nous détaillons le schéma à trois couches de la façon suivante :



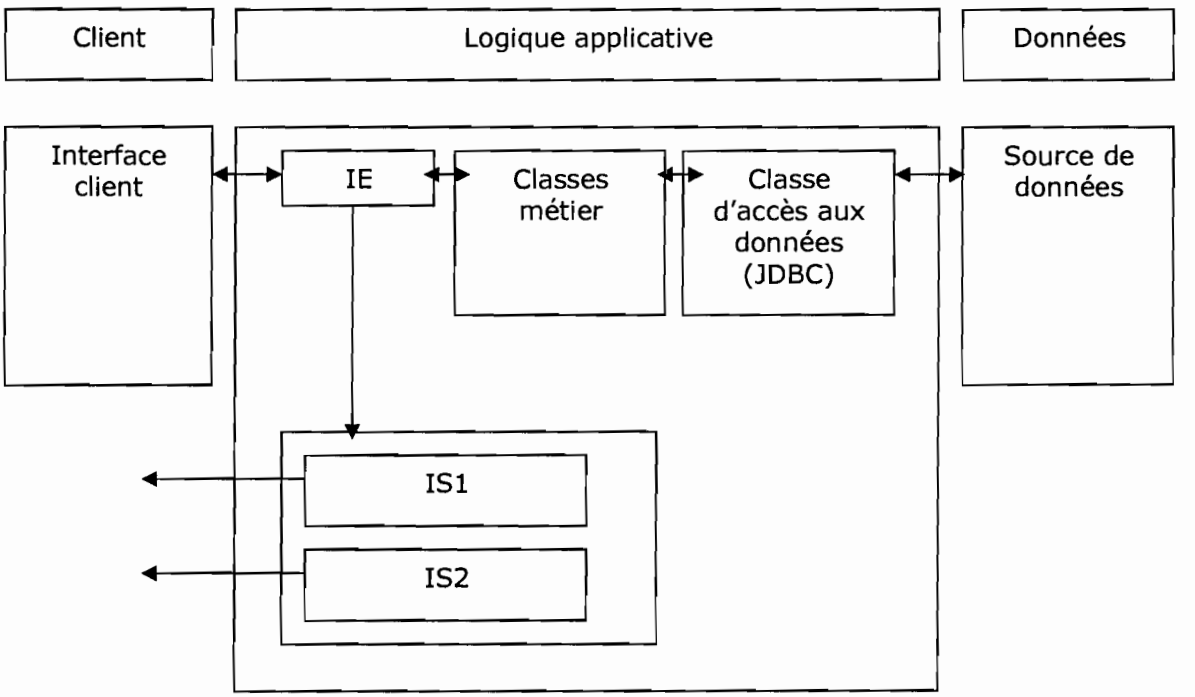
De manière analogue, la répartition en fonction du modèle MVC est la suivante :

- **Modèle** (M) : les classes métiers, les classes d'accès aux données et la source de données ;
- **Les vues** (V) : les pages JSP ;
- **Les contrôleurs** (C) : la Servlet-Contrôleur de traitement des requêtes clientes.

L'**interface utilisateur** est ici un navigateur web mais cela pourrait être également une application autonome qui, via le réseau, enverrait des requêtes HTTP au service web et mettrait en forme les résultats que celui-ci lui envoie. La **logique applicative** est constituée des scripts traitant les demandes de l'utilisateur, ici la Servlet-Contrôleur Java. La **source de données**, pour notre cas sera une base de données relationnelle.

Nous maintiendrons une grande indépendance entre ces trois entités afin que si l'une d'elles change, les deux autres n'aient pas à changer ou peu.

On mettra la logique métier de l'application dans des classes Java séparées de la Servlet-Contrôleur qui contrôle le dialogue *demandes-réponses* ou *questions-réponses*. Ainsi le bloc [Logique applicative] ci-dessus sera constitué des éléments suivants :



Dans le bloc [Logique Applicative], on pourra distinguer :

- le bloc [IE=Interface d'Entrée] qui est la porte d'entrée de l'application. Elle est la même quelque soit le type de client.
- le bloc [Classes métiers] qui regroupe des classes Java est nécessaire à la logique de l'application. Elles sont indépendantes du client.
- le bloc des générateurs des pages réponses (pages JSP) [IS1 IS2 ... IS=Interfaces de Sortie]. Chaque générateur est chargé de mettre en forme les résultats fournis par la logique applicative le client léger : le navigateur web.

Ce modèle assure une bonne indépendance vis à vis du client léger. Que le client change ou qu'on veuille faire évoluer sa façon de présenter les résultats, ce sont les générateurs de sortie [IS] qu'il faudra créer ou adapter.

Dans une application web, l'indépendance entre la couche présentation et la couche traitement peut être améliorée par l'utilisation de feuilles de style. Celles-ci gouvernent la présentation d'une page Web au sein d'un navigateur. Pour changer cette présentation, il suffit de changer la feuille de style associée. Il n'y a pas à toucher à la logique de traitement.

Dans le diagramme ci-dessus, les classes métiers passent par des classes intermédiaires pour accéder aux données (des classes utilisant l'API JDBC). Ainsi les classes métiers forment le noyau dur de l'application insensible aux changements de présentation ou d'accès aux données.

IV. CONCEPTION DETAILLEE DU FUTUR SYSTEME

1. MODELISATION DU FUTUR SYSTEME

a. Diagramme de cas d'utilisation

NB : Pour des besoins de lisibilité du diagramme et faciliter sa bonne compréhension, nous n'avons pas représenté le cas d'utilisation « **Authentification** » sur le digramme.

Le diagramme ici

b. Diagrammes de séquence système

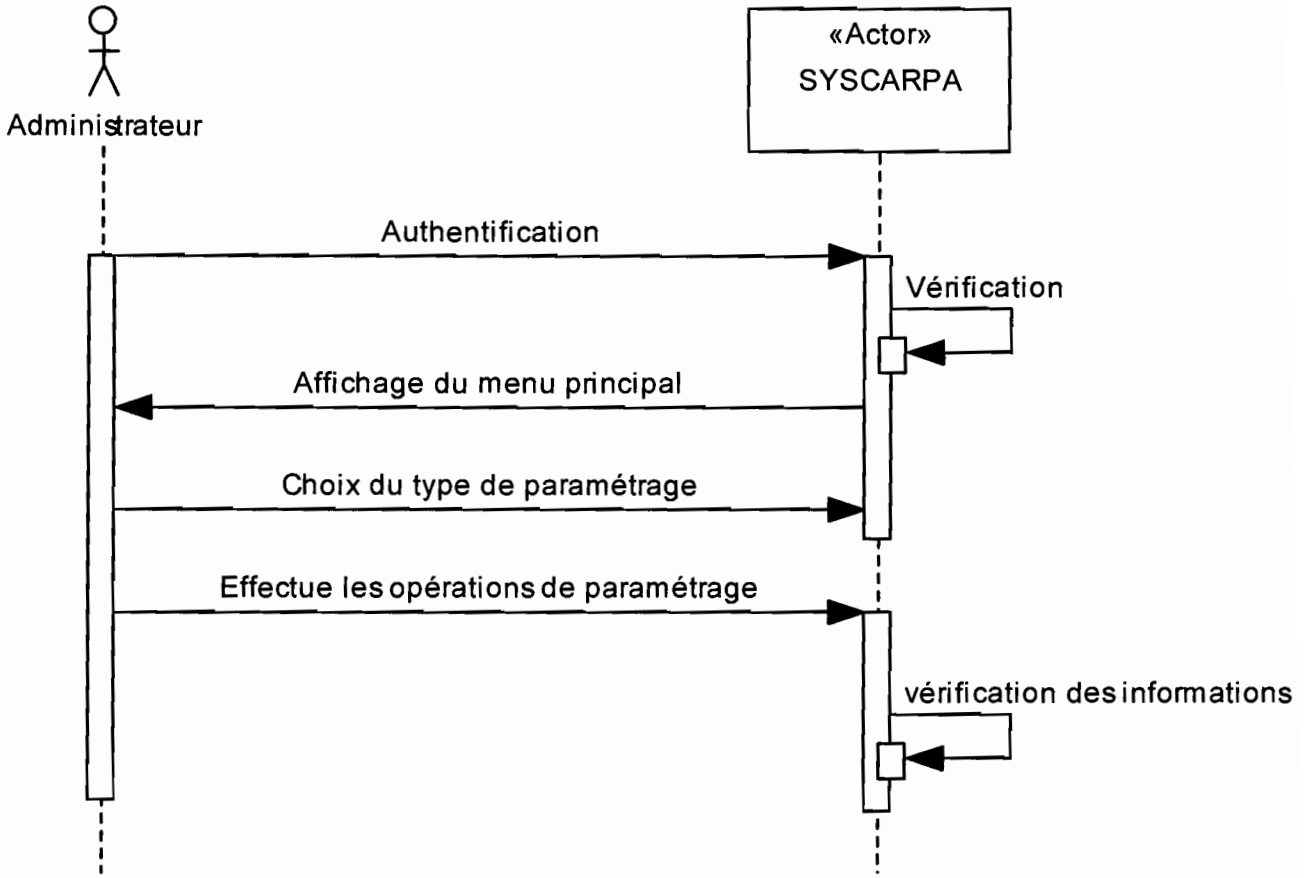


Diagramme séquence 1 : illustrant le cas d'utilisation « **Paramétrage** »

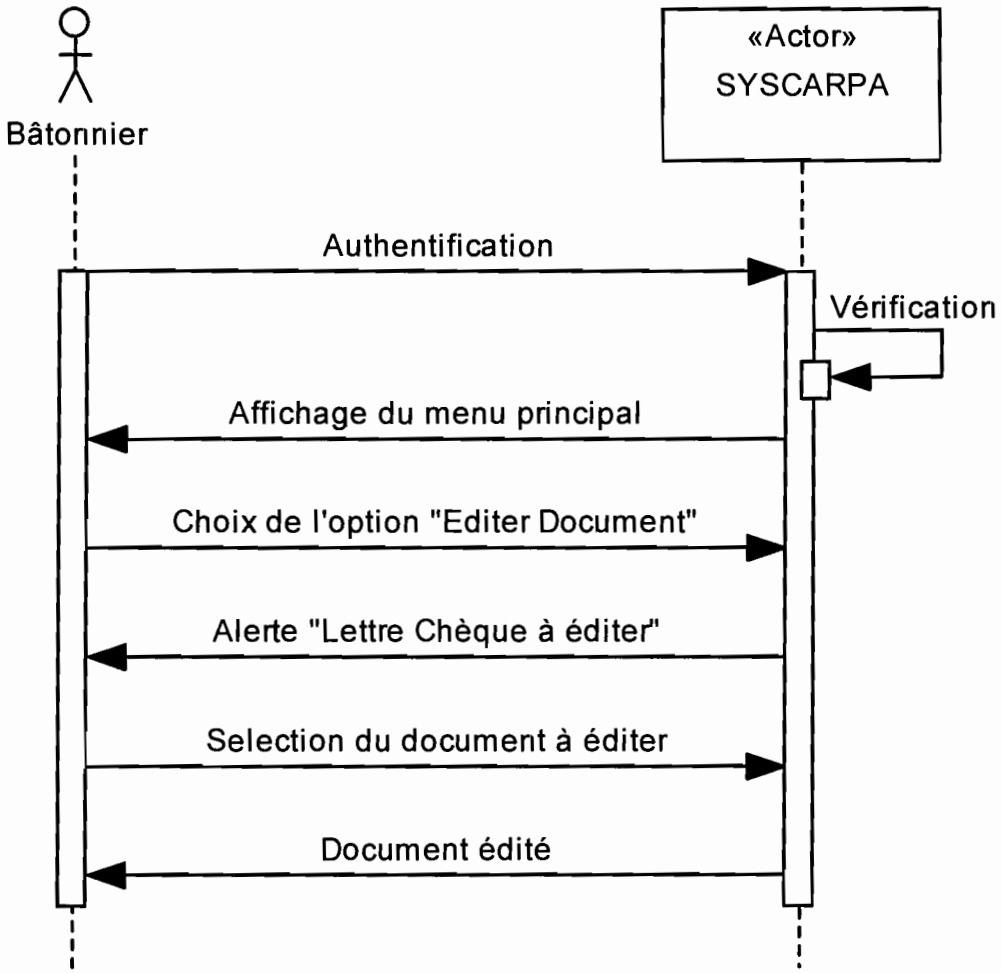


Diagramme séquence 2 : illustrant le cas d'utilisation « **Éditer document** »

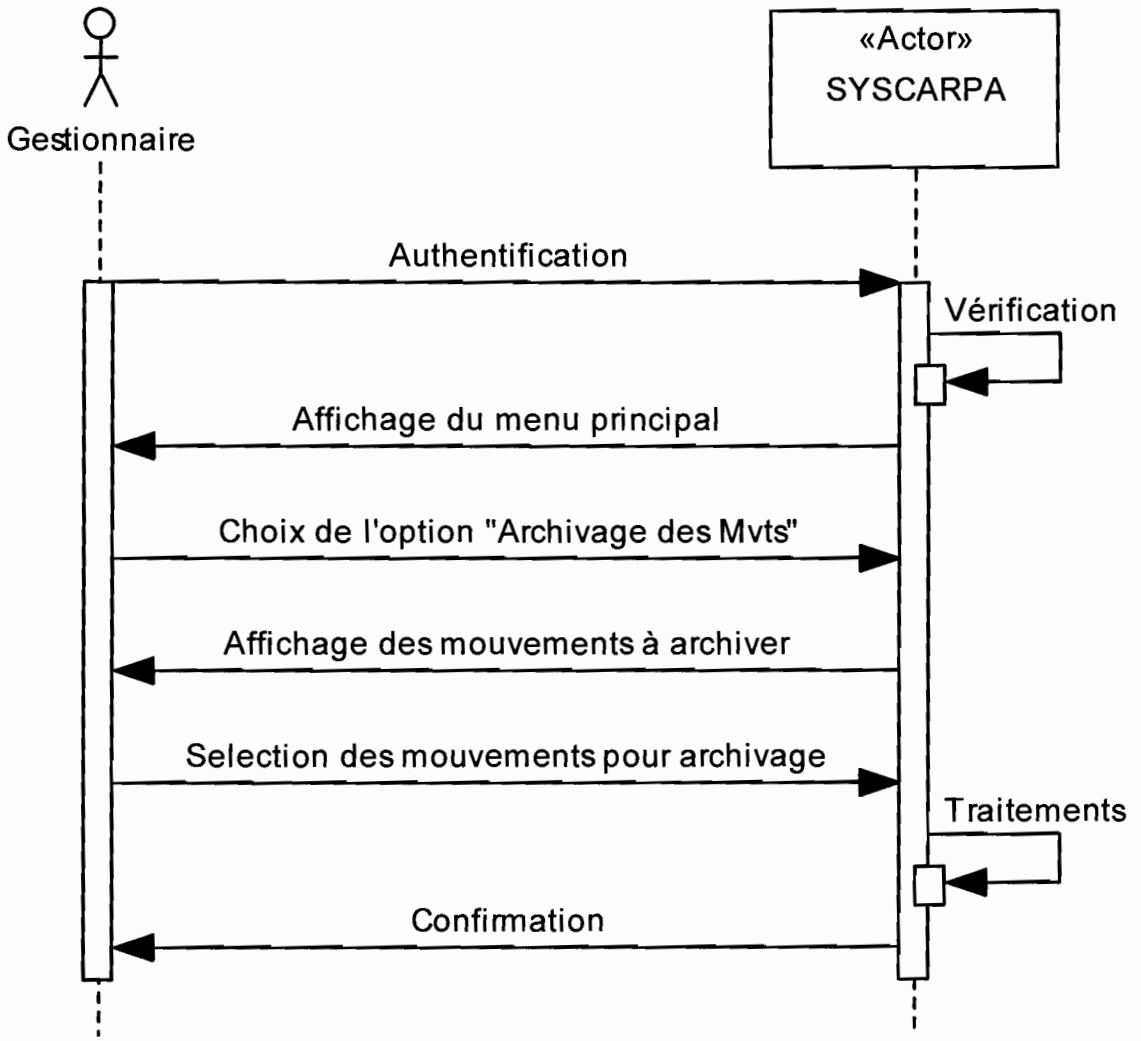


Diagramme séquence 3 : illustrant le CU « **Archivage des Mvts** »

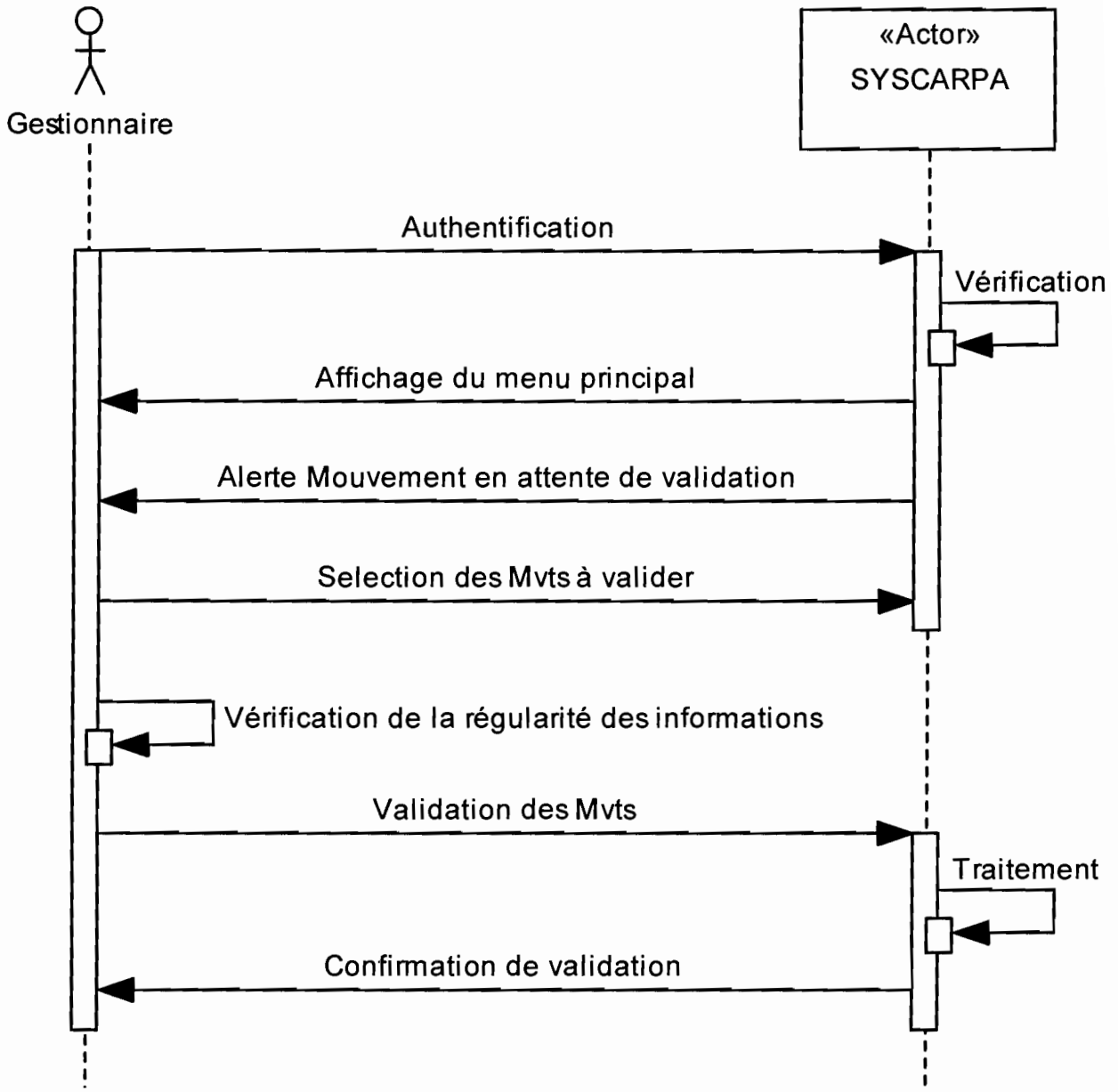


Diagramme séquence 4 : illustrant le CU « **Validation** »

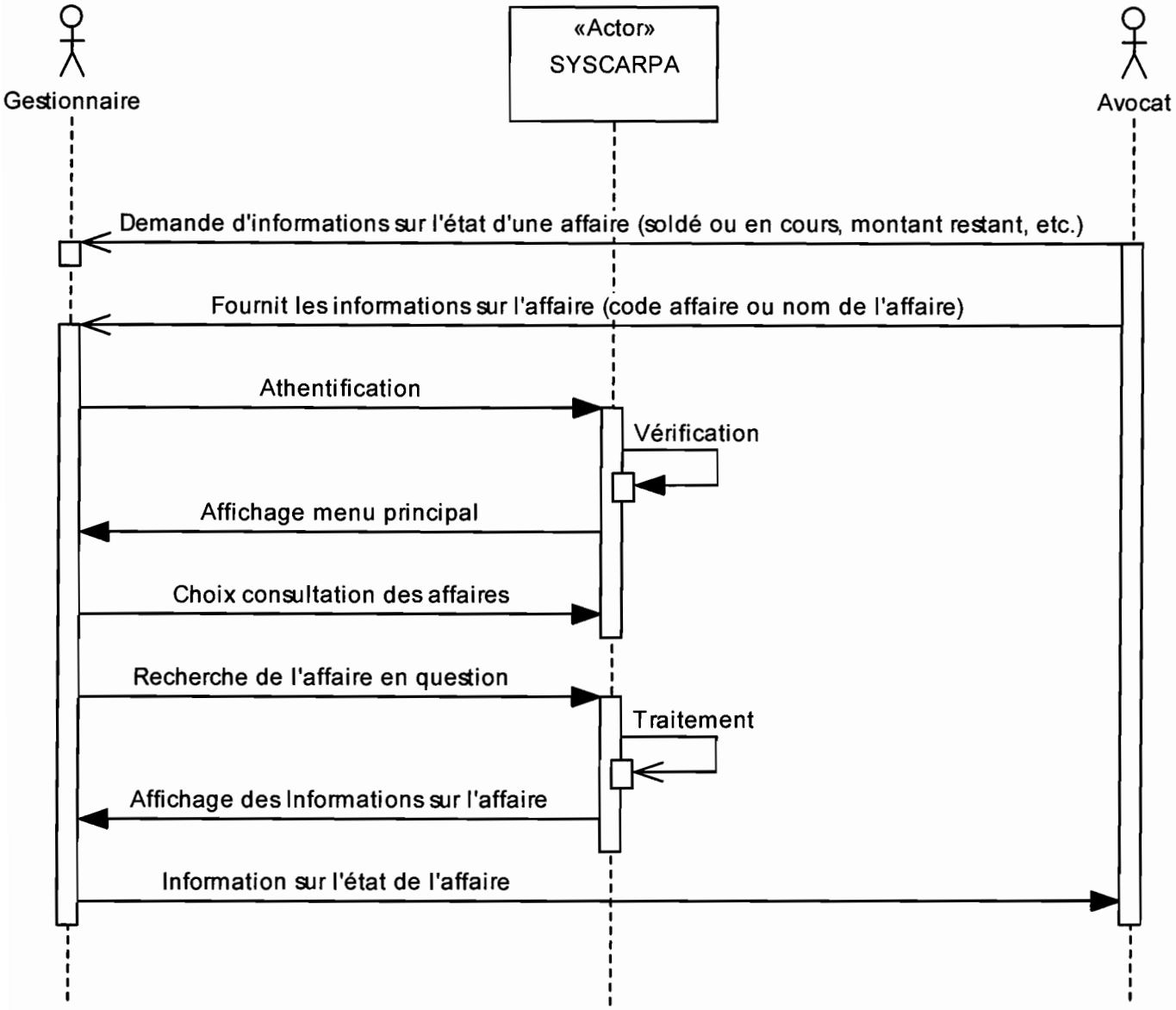


Diagramme séquence 5 : illustrant le CU «**Consultation état d'une affaire** » par un avocat qui n'a pas accès direct à SYSCARPA.

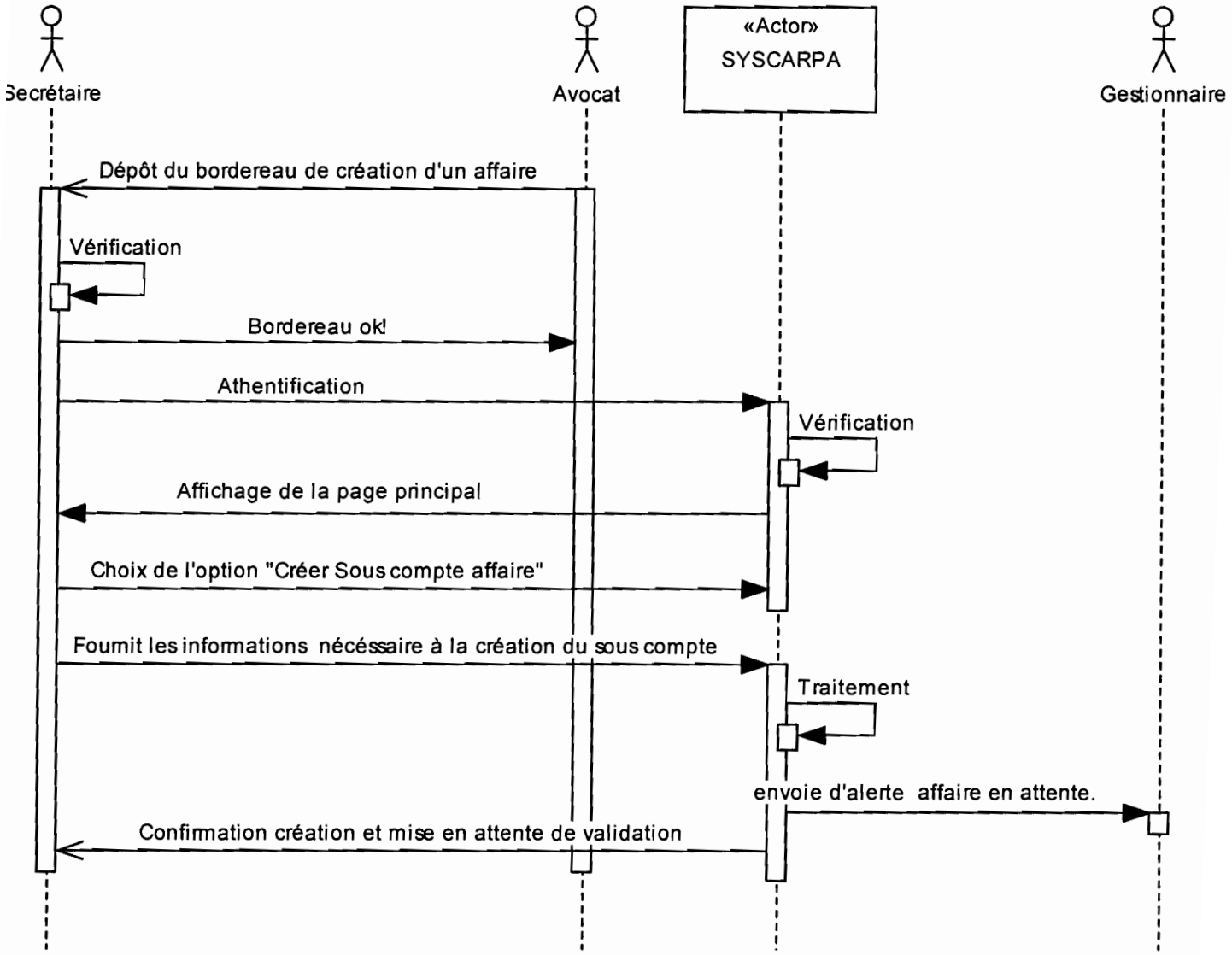


Diagramme séquence 6 : illustrant le CU «Création d'un sous-compte affaire »

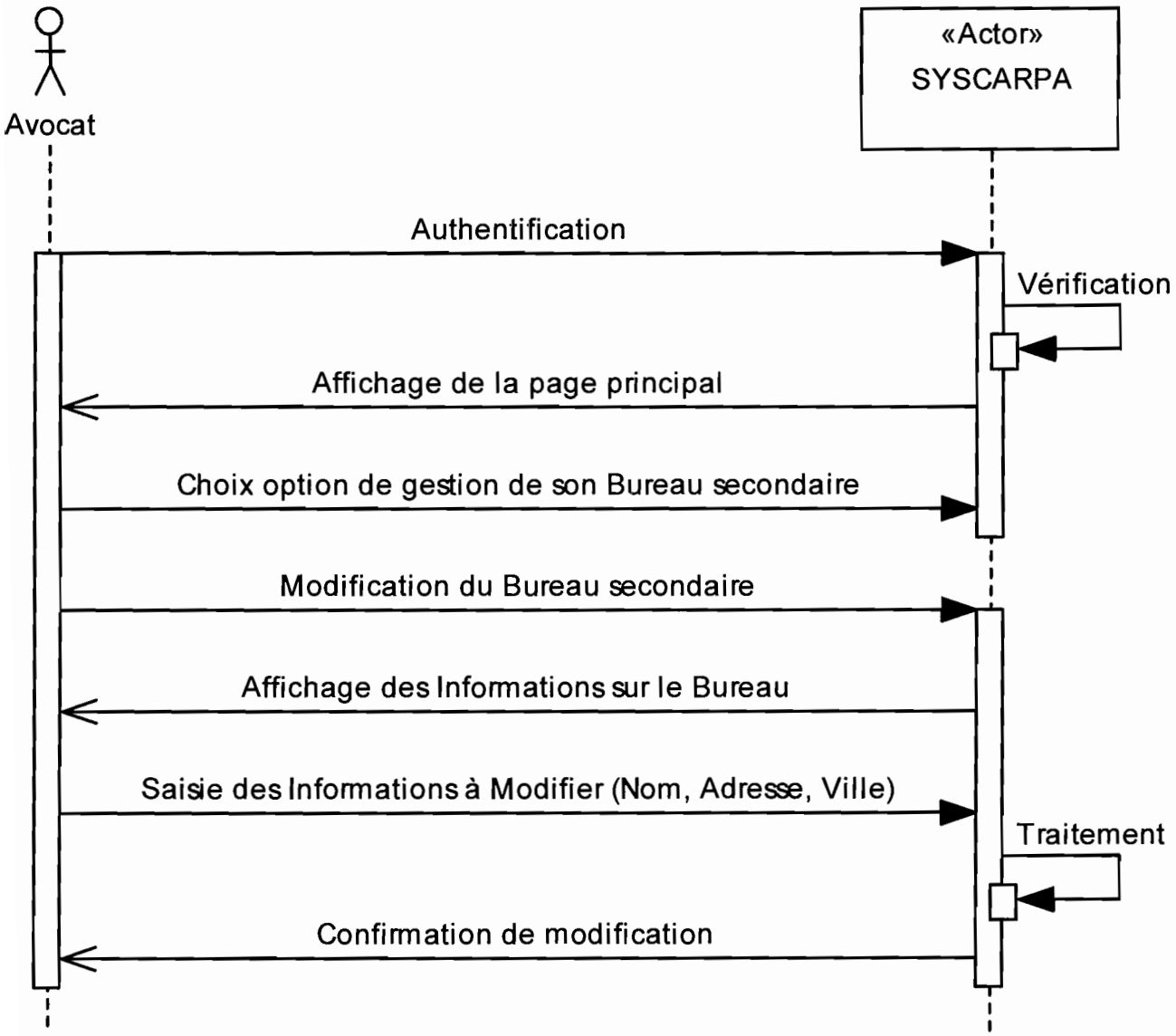


Diagramme séquence 7 : illustrant le CU « Gestion Bureaux secondaires »

c. Diagramme d'activités

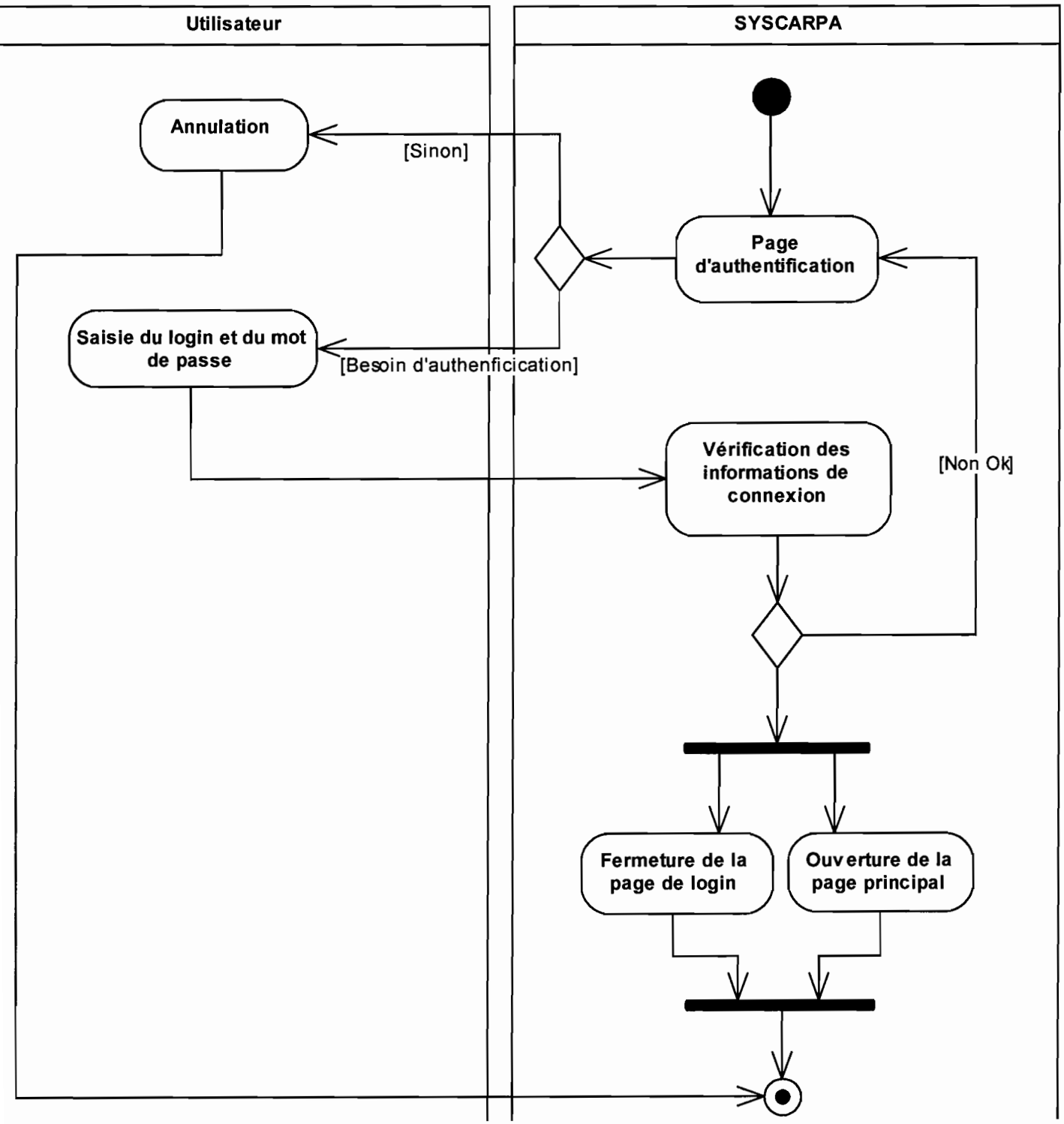


Diagramme d'activité 1 : CU **Authentification**

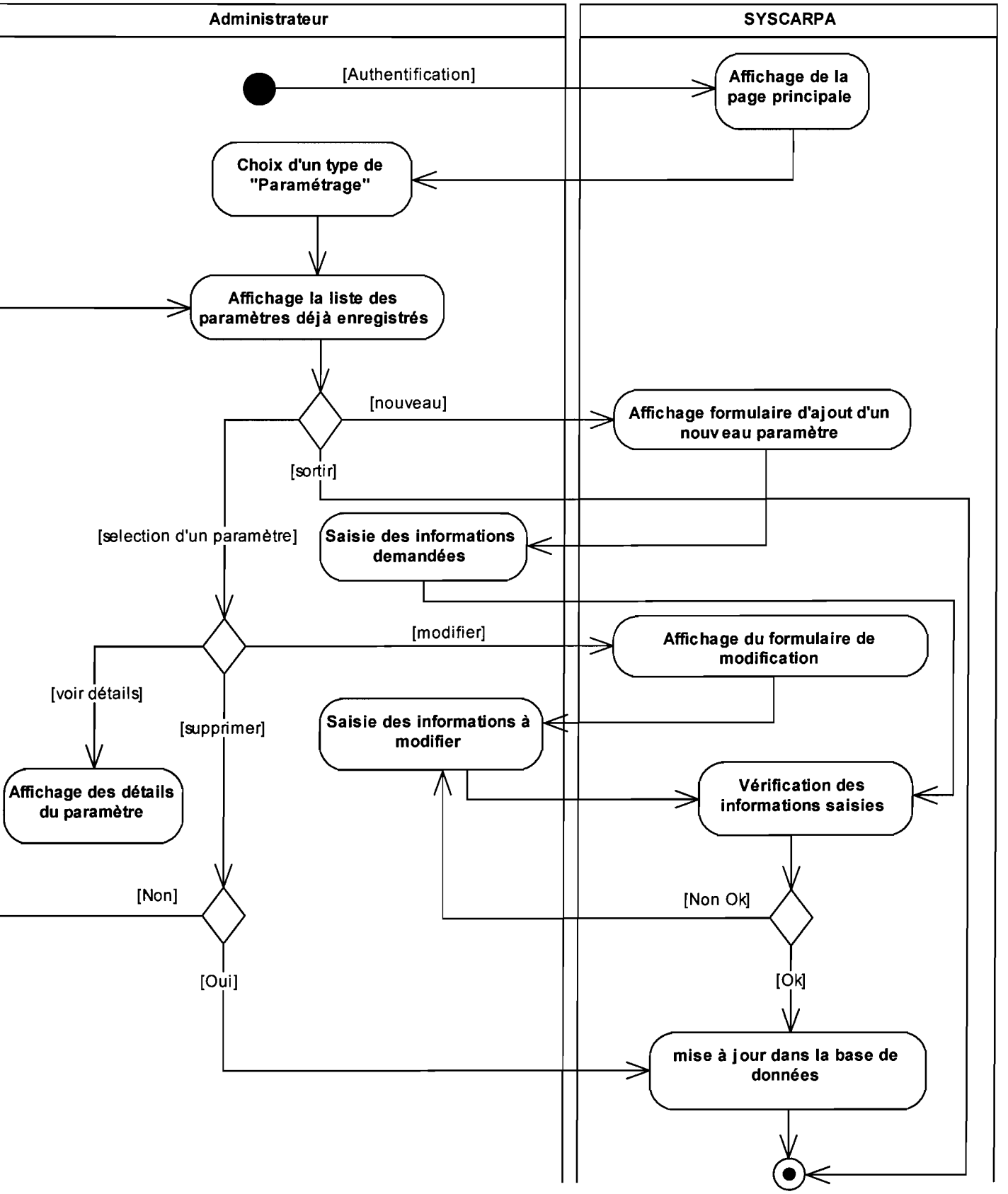


Diagramme d'activité 2 : CU **Paramétrage**

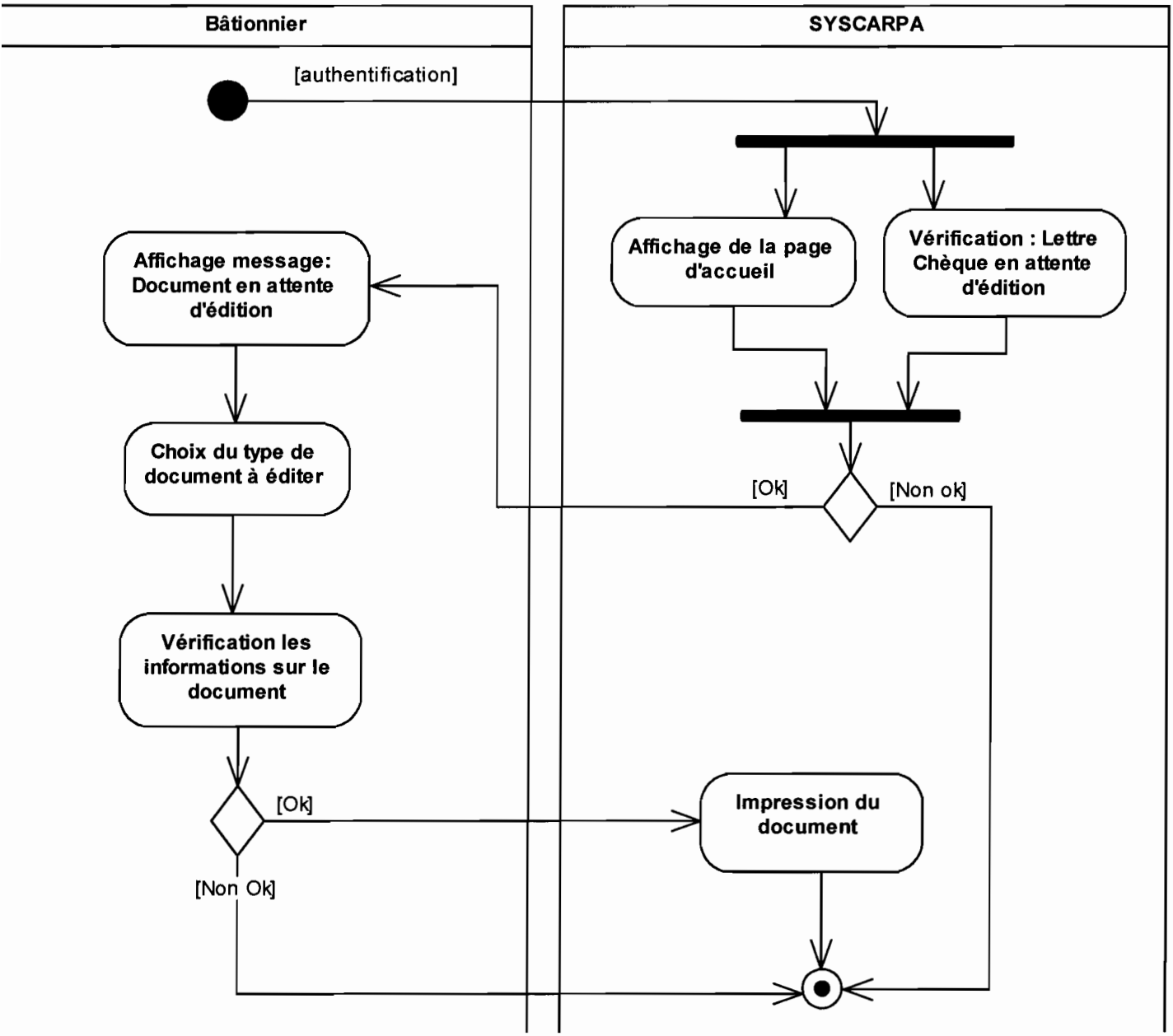


Diagramme d'activité 3 : CU **Édition document**

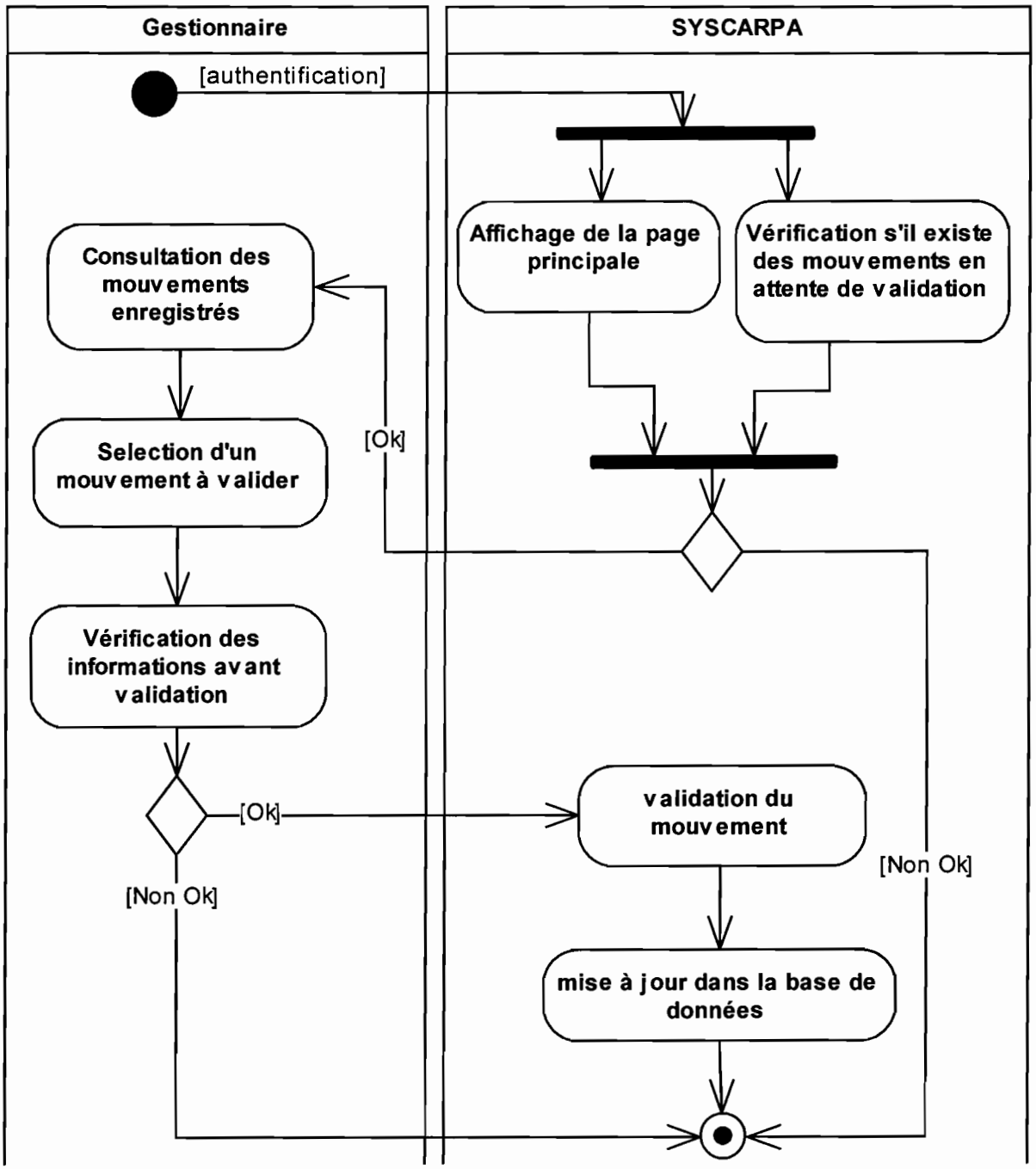


Diagramme d'activité 4 : CU **Validation**

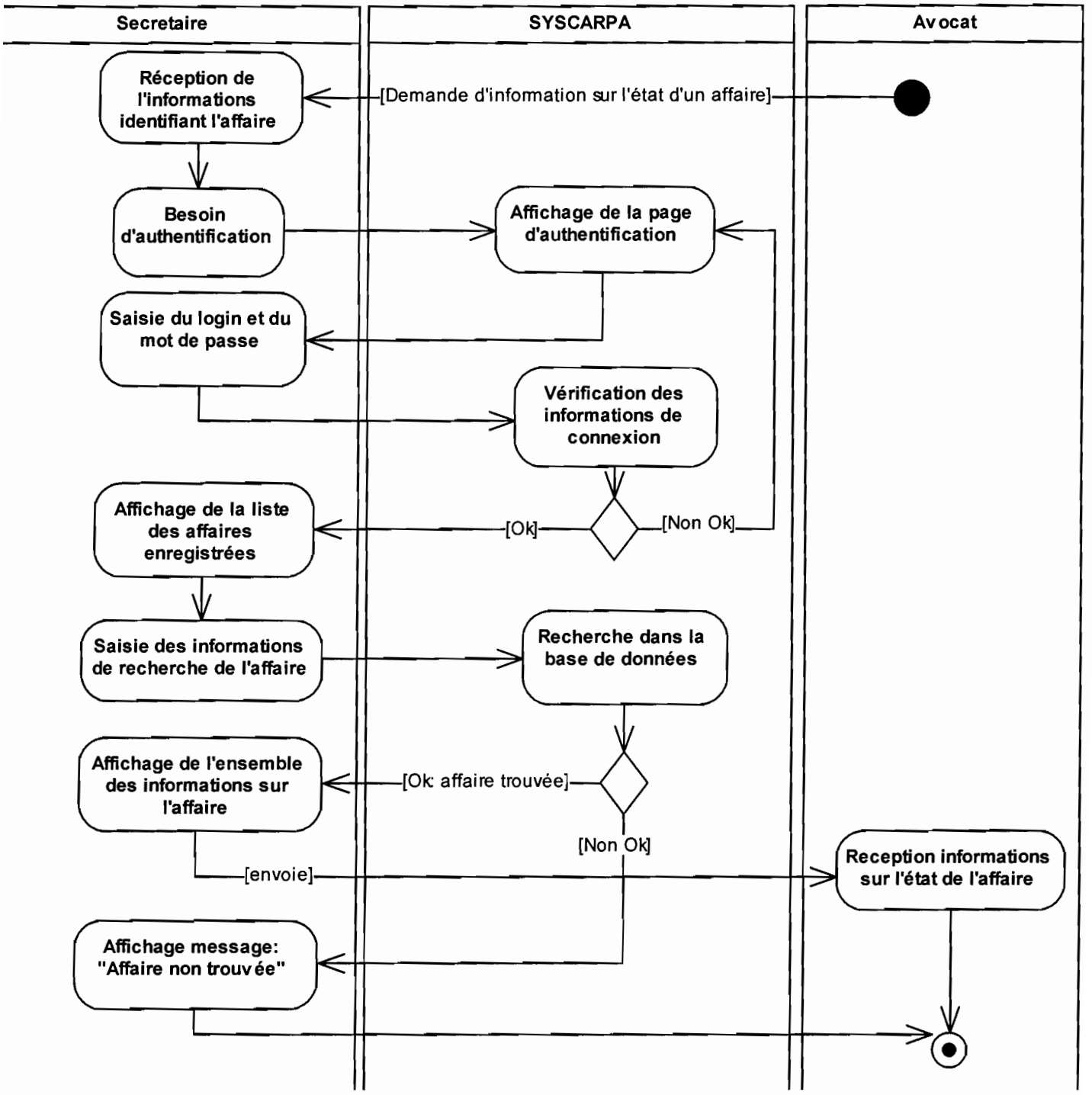


Diagramme d'activité 5 : CU **Consultation état affaire**

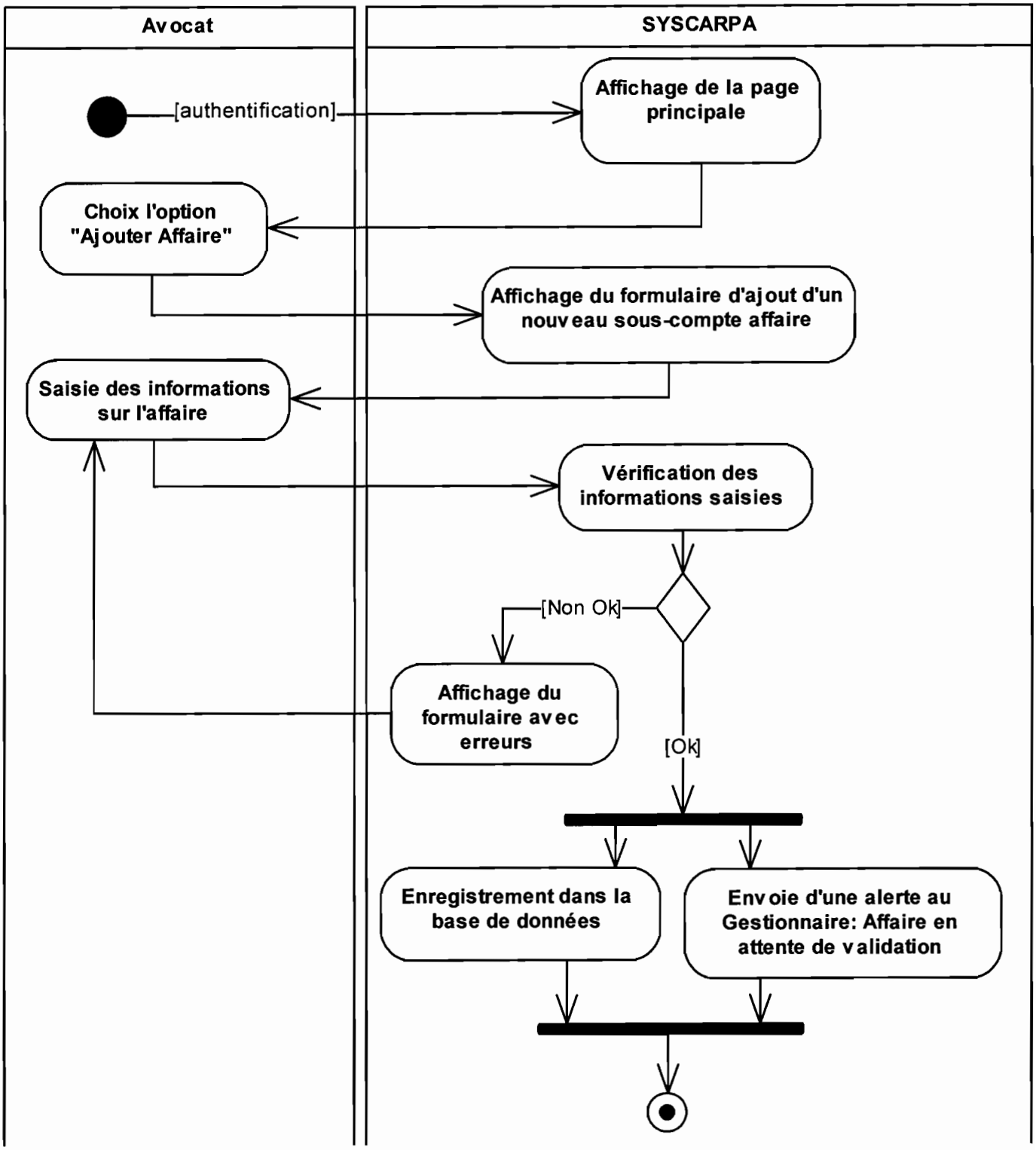


Diagramme 6 : CU **Création un sous-compte affaire**

d. Les classes métiers détaillées

Les classes qui proviennent de l'analyse lors de l'étude préliminaire ne sont pas toujours conformes aux possibilités d'implémentation, il est alors nécessaire d'y ajouter de nouveaux concepts pour prendre en charge des responsabilités purement techniques.

Dans certains cas, une analyse orientée objet est réalisée dans un codage non objet, alors la transformation des classes en codage est alors particulièrement importante pour conserver la trace du passage de l'analyse au code. Java offre bien entendu une transformation beaucoup plus directe lorsque les classes sont conçues de manière détaillée. La conception détaillée des classes consistera donc à expliciter les concepts d'attribut, de méthode et de visibilité, de propriété, de multiplicité, de type des ces attributs et méthodes.

- **Les Classes**
  - un concept dans le système **:** personne, place, chose, concept, événement, écran, ou rapport, etc. ;
  - descripteur d'un ensemble d'objets ayant une structure, un comportement et des relations similaires.
  - un module à partir duquel des objets sont créés.
  - Les blocs constitutifs d'une application POO (Programmation Orientée Objet).

- **Les attributs**
  - Information relative à un objet stockée par l'objet ou au moins temporairement maintenue.
  - Syntaxe :
    visibilité nom [**multiplicité**] : Type = val initiale **{propriétés}**

- **Les méthodes :** les opérations qu'une classe sait réaliser.
  - Syntaxe :
    visibilité nom(liste de paramètres) : type de retour {**propriétés**}

- **Visibilité:**
  - **+ (public)** : accessible à tout objet du système ;
  - **# (protected)** : accessible à toute instance de la classe et ses sous-classes (classes filles) ;
  - **- (private)** : accessible à toute instance de la classe.

- **Multiplicité:**
  - Valeur par défaut : [1..1]
  - Exemple :
    - coul[3] : Couleur
    - point[2..*] : Point
    - adr: Adresse

- **Propriété:** {nom_propriété}
  - **addOnly :** pour un attribut dont la multiplicité est supérieure à 1, on peut ajouter des valeurs, mais on ne peut modifier ou effacer une valeur une fois qu'elle a été ajoutée, pour un lien, on peut ajouter des nouveaux liens depuis un objet du côté opposé de l'association donnée;
  - **changeable :** pour un attribut, on peut ajouter, modifier et effacer des valeurs sans restriction, pour un lien, on peut ajouter, modifier et effacer des liens entre les objets connectés par l'association donnée sans restriction;
  - **frozen** : pour un attribut, on ne peut ajouter, modifier ou effacer des valeurs, pour un lien, on peut ajouter des nouveaux liens d'un objet en cours de création de l'autre côté de l'association donnée, mais on ne peut modifier ou effacer un lien une fois qu'il a été ajouté ;
  - **readOnly** : attribut ou méthode en lecture seule.

- **Description des classes**

Stéréotype : **entity**					
CLASSE : **Cabinet**, les cabinets intervenants dans le système					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
privée	CodeCabinet	1	String	Code du Cabinet (unique)	frozen
public	NomCabinet	1..*	String	Nom du cabinet	changeable
public	ArdCabinet	1..*	String	Adresse postale du cabinet	changeable
public	NumRue	1..*	Integer	Numéro de rue du Cabinet	changeable
public	TelCabinet	1..*	String	N° de téléphone du Cabinet	changeable
public	FaxCabinet	1..*	String	N° de Fax du Cabinet	changeable
public	RegistreCom	1	Double	N° du registre de commerce	changeable
public	DateRegistre	1	Date	Date d'enregistrement du registre de commerce	changeable
public	NumIDFiscale	1	Double	N° d'identification fiscale	changeable
public	DateCreat	1	Date	Date d'enregistrement du cabinet à la CARPA-BF	changeable
public	FormJuridiq	1..*	String	La forme juridique du cabinet	changeable
public	Ville	1..*	Ville	La ville où se trouve le cabinet	readOnly
public	Carpa_Cab	1	Carpa	La CARPA du cabinet	readOnly

Méthodes

Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	Cabinet	Cabinet	NomCabinet,...	Le constructeur	constructor
public	Booléen	AjouterCab	void	Ajouter un cabinet	update
public	Booléen	ModifierCab	CodeCabinet,...	Modifier un cabinet	update
public	Double	CalcSoldCab	CodeCabinet	Calculer le solde du cabinet	query
public	void	ConsultCab	CodeCabinet	Voir détails Cabinet	query
public	void	ListerCab	void	Consulter la liste des cabinets enregistrés	query
public	Cabinet	RechercherCab	CodeCab ou NomCab	Rechercher un cabinet	query
public	Booléen	EditerCab	CodeCab	Éditer les informations d'un cabinet	query

Stéréotype : **entity**					
CLASSE : **Personne,** classe généralisant toutes les entités humaines du système					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
privée	CodePersonne	1	String	Identifiant de la Personne	frozen
public	NomPersonne	1	String	Nom	changeable
public	PrenPersonne	1	String	Prénom(s)	changeable
public	Adresse	1..*	String	Adresse postale	changeable
public	Telephone	1..*	String	Téléphone	changeable
public	Civilite	1..3	String	Civilité	changeable
public	email	1..*	String	Adresse email	changeable
public	Banque	1..*	Banque	La Banque des clients et avocats	readOnly

Méthodes					
Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	Personne	Personne	CodePersonne,...	Le constructeur	constructor
public	String	GetCodePers	void	Retourne l'identifiant	query

Stéréotype : **entity**					
CLASSE : **Personnel**, les acteurs internes de la CARPA-BF					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
public	Fonction	1	String	Fonction assumée au sein de la CARPA-BF	changeable
public	Personne	1	Personne	Informations complémentaires	readOnly
privée	DroitAcces	1	ComptUser	Droits d'accès au système	readOnly

Méthodes					
Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	Personnel	Personnel	Fonction,...	Le constructeur d'objets	constructor
public	Booléen	AjouterPersonnel	void	Retourne l'identifiant	update
public	Booléen	ModifierPersonnel	CodePersonne,...	Modifier un personnel	update
public	Booléen	SupprPersonnel	CodePersonne	Supprimer personnel	update
public	ComptUser	GetAccess	void	Obtenir les accès	query

Stéréotype : **entity**					
CLASSE : **Client**, les informations sur les clients (client ou partie adverse) dans une affaire					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
public	TypeClient	1..2	String	Type client : client ou adverse	frozen
public	Personne	1	Personne	Informations complémentaires	readOnly

Méthodes					
Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	Client	Client	TypeClient,...	Le constructeur	constructor
public	Booléen	AjouterClient	void	Ajouter Client	update
public	Booléen	ModifierClient	CodePersonne	Modifier un Client	update

Stéréotype : **entity**					
CLASSE : **BuroSec**, les informations sur les bureaux secondaires					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
public	CodeBuro	1	String	Identifiant Bureau secondaire	frozen
public	NumPorte	1..*	Integer	N° de Porte du bureau	changeable
public	AdressBuro	1..*	String	Adresse postale	changeable
public	NumRue	1..*	Integer	N° de rue	changeable
public	TelephoneBuro	1..*	String	Téléphone	changeable
public	FaxBuro	1..*	String	Fax	changeable
public	AvocatProprio	1..*	Avocat	Propriétaire du bureau	readOnly
public	Ville_Buro	1..*	Ville	La ville où se trouve le Bureau	readOnly

Méthodes					
Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	BuroSec	BuroSec	CodeBuro,...	Le constructeur	constructor
public	Booléen	AjouterBuro	void	Ajouter un bureau	update
public	Booléen	ModifierBuro	CodeBuro	Modifier un Bureau	update
public	BuroSec	ConsultBuro	CodeBuro	Voir détail bureau	query
public	void	ListerBuro	void	Lister les bureaux	query
public	BuroSec	RechercherBuro	CodeBuro ou CodeAvocat	Rechercher un bureau	query

Stéréotype : **entity**

CLASSE : **Avocat**, les informations d'identification d'un avocat

Attributs

Visibilité	Nom	Multiplicité	Type	Description	Propriété
privée	SitMat	1	String	Situation matrimoniale	changeable
privée	DateSerment	1	Date	Date de prestation de serment	frozen
privée	LieuSerment	1	String	Lieu de prestation du serment	frozen
public	NumCase	1..*	Integer	N° de case du dossier	changeable
privée	DateNaiss	1	String	Date de naissance	changeable
privée	LieuNaiss	1	String	Lieu de naissance	changeable
privée	Nationalité	1..*	String	Nationalité	changeable
privée	Pere	1	String	Le nom du père de l'Avocat	changeable
privée	Mere	1	String	Le nom de la mère	changeable
privée	NombreEnfs	1..*	Integer	Nombre d'enfants	changeable
public	Personne	1	Personne	Informations complémentaires	readOnly
privée	DroitAcces	1	ComptUser	Droits d'accès au système	readOnly

Méthodes

Visibilité	Type	Nom	Paramètres	Description	Propriété
public	Avocat	Avocat	SitMat,...	Le constructeur d'objets	constructor
public	Booléen	AjouterAvocat	void	Ajouter un Avocat	update
public	Booléen	ModifierAvocat	CodePersonne,...	Modifier informations Avocat	update
public	Avocat	ConsultAvocat	CodeCabinet	Voir détails Avocats	query
public	void	ListerAvocat	void	Consulter la liste des Avocats enregistrés	query
public	Avocat	RechercherAv	CodeCab, NomCab	Rechercher un Avocat	query
public	Booléen	EditerInfoAv	CodeCab	Éditer les informations d'un Avocats	query
public	String	GetCodeAv	void	Retourner le code de l'Avocat	query
public	ComptUser	GetAccess	void	Obtenir les accès au système	query

Stéréotype : **entity**					
CLASSE : **Pays**, les pays d'origine des Avocats					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
privée	CodePays	1	Integer	Identifiant du Pays	frozen
public	NomPays	1	String	Nom du pays	changeable
public	CodeAvocat	1	Avocat	Informations complémentaires	readOnly

Méthodes					
Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	Pays	Pays	CodePays,...	Le constructeur	constructor
public	Booléen	AjouterPays	void	Ajouter Pays	update
public	Booléen	ModifierPays	CodePays,...	Modifier un Pays	update
public	Booléen	SuppPays	CodePays	Supprimer un pays	update
public	Pays	ConsultPays	CodePays	Voir détails d'un pays	query
public	void	ListerPays	void	Lister tous les pays	query
public	Pays	RechercherPays	CodePays	Rechercher un pays	query

Stéréotype : **entity**					
CLASSE : **Ville**, les villes où sont situés les cabinets, bureaux secondaires, etc.					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
privée	CodeVille	1	Integer	Identifiant d'une ville	frozen
public	NomVille	1..*	String	Nom de la ville	changeable
public	PaysVille	1	Pays	Le pays de la ville	readOnly

Méthodes					
Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	Ville	Ville	CodeVille,...	Le constructeur	constructor
public	Booléen	AjouterClient	void	Ajouter Client	update
public	Booléen	ModifierClient	CodeVille,...	Modifier un Client	update
public	Booléen	SuppVille	CodeVille	Supprimer une ville	query
public	Ville	ConsultVille	CodeVille	Voir détails d'un pays	query
public	void	ListerVille	void	Lister toutes les villes	query

Stéréotype : **entity**

CLASSE : **Langue**, langue parlée par un Avocat

Attributs

Visibilité	Nom	Multiplicité	Type	Description	Propriété
privée	CodeLang	1	Integer	Identifiant langue	frozen
public	NomVille	1..*	String	Nom de la ville	changeable
public	Avocat_parlant	1..*	Avocat	Avocat parlant	readOnly

Méthodes

Visibilité	Type	Nom	Paramètres	Description	Propriété
public	Langue	Langue	CodeLang,...	Le constructeur d'objets	constructor
public	Booléen	AjouterLang	void	Ajouter langue	update
public	Booléen	ModifierLang	CodeLang,...	Modifier une langue	update
public	Booléen	SupprLang	CodeLang	Supprimer une langue	query
public	Langue	ConsultLang	CodeLang	Voir détails langue	query
public	void	ListerLang	void	Lister toutes les langue	query

Stéréotype : **entity**

CLASSE : **CompteSpec**, comptes spéciaux de la CARPA-BF pour des transactions temporaires

Attributs

Visibilité	Nom	Multiplicité	Type	Description	Propriété
privée	NumCompt	1	String	Numéro du compte	frozen
public	NomCompt	1	String	Libellé du compte	changeable
privée	SoldeComptSpec	1..*	Double	Solde du compte	changeable
public	CarpaProprio	1	Carpa	Propriétaire du compte	readOnly

Méthodes

Visibilité	Type	Nom	Paramètres	Description	Propriété
public	CompteSpec	CompteSpec	NumCompt,...	Le constructeur d'objets	constructor
public	Booléen	SetSolde	MontantSolde	Ajouter langue	update
public	Booléen	ModifierLang	NumCompt,...	Modifier une langue	update
public	CompteSpec	ConsultCompt	NumCompt	Supprimer une langue	query

Stéréotype : **entity**					
CLASSE : **Carpa**, informations sur la/les CARPA-BF					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
privée	NumCompt	1	String	Numéro du compte	frozen
public	NomCarpa	1	String	Libellé de la Carpa	changeable
privé	SigleCarpa	1	string	Sigle de la Carpa	changeable
privée	SoldeBanque	1..*	Double	Solde de la banque	readOnly
Méthodes					
Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	Carpa	Carpa	void	Le constructeur	constructor
public	Booléen	ModifInfoCompt	void	Ajouter langue	update
public	Booléen	EditerRIB	void	Modifier une langue	update
public	Double	CalcSoldGenral	void	Supprimer une langue	query
public	Carpa	ConsltInfoCpt	void	Consulter le Compte	update
public	Double	GetSoldeBanq	void	Obtenir Solde CARPA	query

Stéréotype : **entity**					
CLASSE : **TypeMvt**, la nature du mouvement effectué					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
privée	CodeTypMvt	1	String	Code du mouvement	frozen
public	LibelleMvt	1..7	String	Libellé	changeable
Privé	NatureMvt	1..7	string	Nature	changeable
Méthodes					
Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	TypeMvt	TypeMvt	CodeTYpMvt,...	Le constructeur	constructor
public	Booléen	AjouterType	void	Ajouter	update
public	Booléen	ModifierType	CodeTypMvt,...	Modifier	update
public	Booléen	SupprType	CodeTypMvt	Supprimer	update
public	TypeMvt	ConslterType	CodeTypMvt	Consulter	query
public	void	ListerType	void	Lister tous les types	query

Stéréotype : **entity**					
CLASSE : **Banque,** Les informations des différents banques					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
privée	CodeBanq	1	String	Code bancaire	frozen
public	NomBanq	1	String	Nom de la banque	changeable
public	Swift	1	String	Le Swift	changeable
public	AdrBanq	1..*	String	Adresse postale	changeable
public	TelBanq	1..*	String	Téléphone	changeable
public	CarpaBanq	1	Carpa	La Carpa cliente	readOnly

Méthodes					
Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	Banque	Banque	CodeBanq,...	Le constructeur	constructor
public	Booléen	AjouterBanq	void	Ajouter	update
public	Booléen	ModifierBanq	CodeBanq,...	Modifier	update
public	Booléen	SupprBanq	CodeBanq	Supprimer	update
public	Banque	ConsIterBanq	CodeBanq	Consulter	query
public	void	ListerBanq	void	Lister toutes les banques	query

téréotype : **entity**					
LASSE : **CategorieAff**, les catégories d'affaire traitée					
Attributs					
isibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
ublic	CodeCat	1	String	Code de la catégorie	frozen
ublic	LibelleCat	1..*	String	Libellé	changeable

Méthodes					
isibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
ublic	CategorieAff	CategorieAff	CodeCat,...	Le constructeur	constructor
ublic	Booléen	AjouterCat	void	Ajouter	update
ublic	Booléen	ModifierCat	CodeCat,...	Modifier	update
ublic	CategorieAff	ConsIterCat	CodeCat	Consulter	query
ublic	void	ListerCat	void	Lister toutes les catégories	query

Stéréotype : **entity**					
CLASSE : **Affaire**, les affaires traitées par la CARPA					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
privée	SComptAff	1	String	Sous compte affaire	frozen
public	NomAff	1	String	Libellé	changeable
public	DateAff	1	Date	Date de création	changeable
public	EtatAff	1..2	String	Etat de l'affaire [en cours, soldée]	changeable
privée	SoldeAff	1..*	Double	Solde global	readOnly
public	NatureAff	1..*	CategorieAff	Catégorie de l'affaire	readOnly

Méthodes					
Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	Affaire	Affaire	SComptAff,...	Le constructeur	constructor
public	Booléen	AjouterAff	void	Ajouter	update
public	Booléen	ModifierAff	SComptAff	Modifier	update
public	Booléen	SupprAff	SComptAff	Supprimer	update
public	Affaire	ConslterAff	SComptAff	Consulter	query
public	void	ListerAff	void	Lister toutes les catégories	query
public	Affaire	RecherchAff	SComptAff	Rechercher une affaire	query
public	Booléen	EditerAff	SComptAff	Editer une affaire	query
public	Booléen	SolderAff	SComptAff	Solder une affaire	update
public	Double	GetSoldeAff	void	Obtenir solde de l'affaire	query
public	String	GetSsCompt	void	Obtenir sous compte	query

Stéréotype : **entity**					
CLASSE : **ComptUser**, les comptes des utilisateurs du système					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
privée	IDCompte	1	Integer	Identifiant	frozen
privée	Login	1	String	Login	changeable
privée	Password	1..*	String	Mot de passe	changeable
public	Etat	1..2	String	Etat de connexion[14]	changeable
privée	DateInscript	1	Date	Date d'inscription	frozen
public	Description	1	String	Description	changeable
public	Statut	1..*	String	Statut d'utilisateur	changeable

Méthodes					
Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	ComptUser	ComptUser	IDCompte,...	Le constructeur	constructor
public	Booléen	CreerCompt	void	Créer un compte	update
public	Booléen	ModifCompt	IDCompte,...	Modifier un compte	update
public	Booléen	SupprCompt	IDCompte	Supprimer un compte	update
public	ComptUser	ConsltCompt	IDCompte	Consulter un compte	query
public	void	ListerCompt	void	Lister tous les comptes	query
public	Booléen	ActiverCompt	IDCompte	Activer un compte	update
public	Booléen	DesactivCpt	IDCompte	Désactiver un compte	update
public	String	GetLogin	void	Accéder au login	query
public	String	GetPasswd	void	Accéder au mot de passe	query
public	Integer	GetIDCompt	void	Accéder à l'identifiant	query

[14] Actif, Inactif ou Suspendu.

Stéréotype : **entity**					
CLASSE : **Mouvement**, les mouvements effectués sur les affaires					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
privée	IDMvt	1	Double	Identifiant	frozen
public	DateCreation	1	Date	Date de création	changeable
public	PartieBenef	1..*	String	Partie bénéficiaire	changeable
public	Devise	1..2	String	Devis du montant	changeable
public	DateOperation	1..*	Date	Date opération	changeable
public	MontantMvt	1..*	Double	Montant du mouvement	changeable
public	LettrEdited	1..2	Booléen	Témoin d'édition de lettre	changeable
public	MvtAnnuler	1..2	Booléen	Témoin d'annulation	changeable
public	MvtRapprch	1..2	Booléen	Témoin de rapprochement	changeable
public	NbLCEdited	1..*	Integer	Nombre de LC[15] édité	changeable
public	NbLREdited	1..*	Integer	Nombre de LR[16] édité	changeable
public	NbVir	1..*	Integer	Nombre de Virement	changeable
public	NatureMvt	1..7	TypeMvt	Nature du mouvement	readOnly
public	AffaireSubit	1	Affaire	Affaire concernée	readOnly

Méthodes

Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	Mouvement	Mouvement	IDMvt,...	Le constructeur	constructor
public	Booléen	EnregistrerMvt	void	Enregistrer un mouvement	update
public	Booléen	ModifierMvt	IDMvt,...	Modifier un mouvement	update
public	Booléen	AnnulerMvt	IDMvt	Annuler un mouvement	update
public	Booléen	RapprochMvt	IDMvt	Rapprocher un mouvement	update
public	Mouvement	ConsulterMvt	IDMvt	Voir détails mouvement	query
public	void	ListerMvt	void	Listes les mouvements	query
public	Booléen	EditerMvt	IDMvt,...	Editer Mvt	query
public	Double	GetIDMvt	void	Accéder à l'identifiant	query

[15] LC : Lettre chèque
[16] LR : Lettre de relance

téréotype : **entity**					
LASSE : **Sauvegarde**[17], les affaires sauvegardées					
Attributs					
isibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
rivée	IDSauvegarde	1	Integer	Identifiant	frozen
rivée	DateSauvegarde	1	Date	Date de sauvegarde	changeable
rivée	NomSauvegarde	1..*	String	Nom de sauvegarde	changeable
ublic	CheminFicher	1..2	String	Chemin de sauvegarde	changeable
ublic	AffaireSaved	1	Affaire	Affaire concernée	readOnly

Méthodes					
isibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
ublic	Sauvegarde	Sauvegarde	IDSauvegarde,...	Le constructeur	constructor
ublic	Booléen	Sauvegarder	void	Créer un compte	update
ublic	Booléen	Restauration	IDSauvegarde	Modifier un compte	update
ublic	Booléen	ListerSauvegarde	IDSauvegarde	Supprimer un compte	query

[17] Lorsque la sauvegarde d'une affaire est effectuée, l'ensemble des mouvements qui lui sont rattachés est retiré de la liste des mouvements.

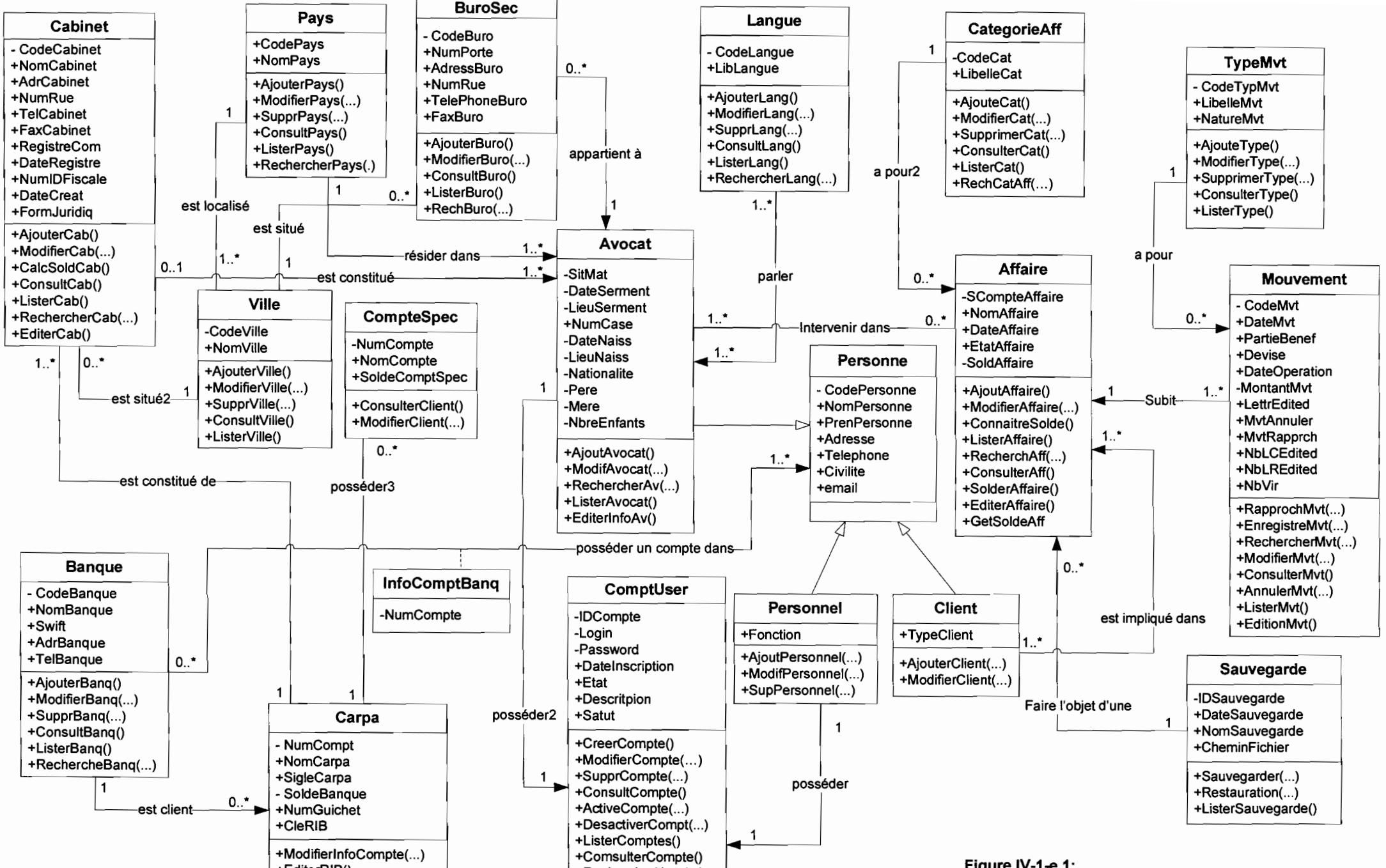


Figure IV-1-e.1:

Diagramme de Classe détaillé du Système futur

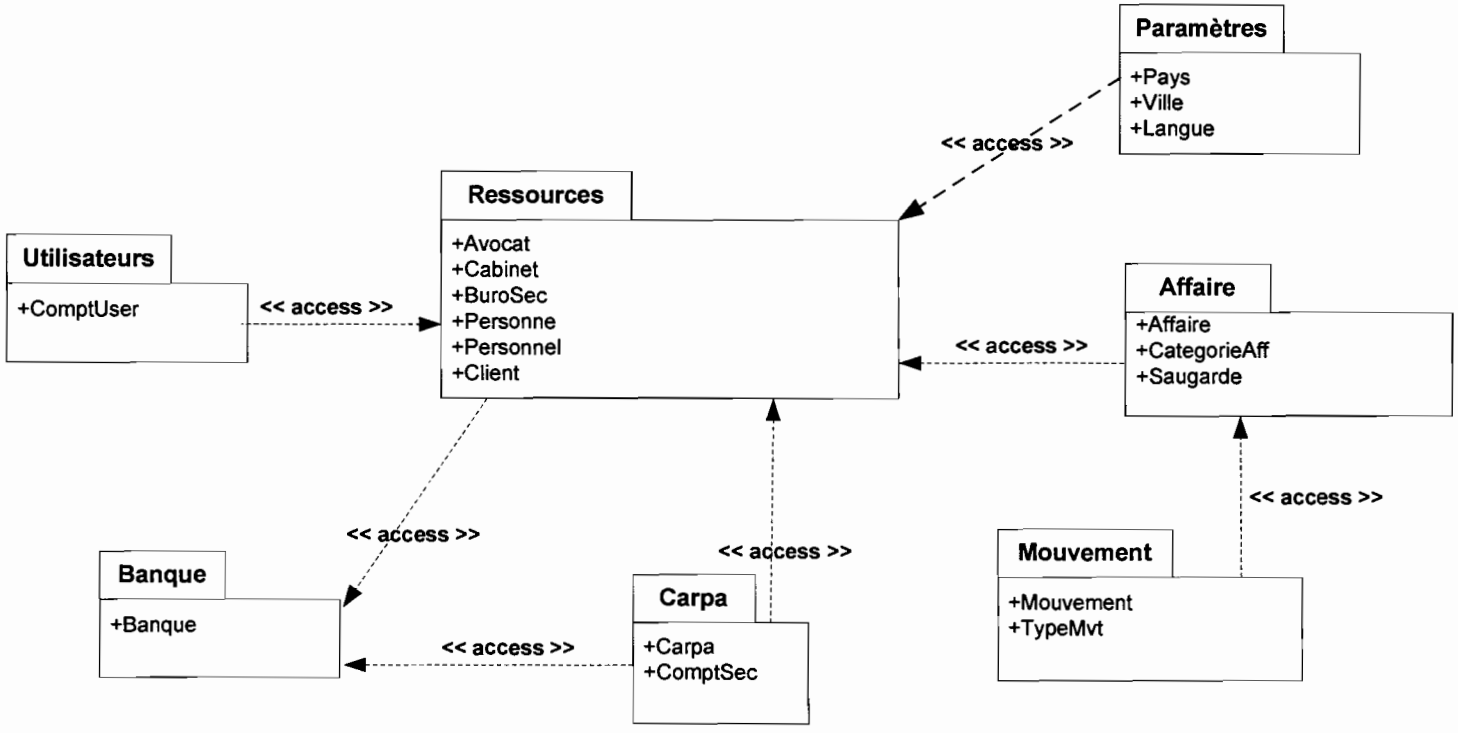


Figure IV-1-f:1

Diagramme de paquetage de classe du système futur

2. CONCEPTION DE LA COUCHE DE PRESENTATION

La couche de présentation ou IHM (Interaction Homme Machine) se limite à la partie visible d'une application. La couche de présentation du futur système sera uniquement composée de pages JSP équipées de mécanismes réflexes JavaScript.

Dans cet environnement, l'utilisateur est face à trois grandes familles de concepts :

- les pages JSP et leur contenu qu'il voit, redimensionne, bouge et réduit, font partie des concepts visuels ;
- les actions qu'il peut déclencher et les changements d'aspects, font partie du comportement de la présentation ;
- les flux de données qu'il transmet via des listes de choix, des champs d'édition font partie de l'échange d'informations avec l'application.

Concevoir ou documenter une couche de présentation revient à passer en revue ces trois aspects : le visuel, le comportemental et le fonctionnel. Grâce aux environnements de développement actuels, la conception d'une IHM s'effectue le plus souvent conjointement avec sa construction visuelle. Cette capacité est liée à l'existence d'éléments standard qui peuvent se composer à souhait pour former l'application de notre choix. Une IHM web se caractérise également par la notion de page qui représente un élément insécable de présentation. Cette notion présente en effet une unité visuelle, comportementale et fonctionnelle, caractéristique très utile pour organiser le modèle UML de conception. Par ailleurs, une page correspond à une vue de la couche applicative. Dans le cas de notre projet, la classe contrôleurs (Servlet) de la couches applicative sera chargée d'établir la jonction entre les classes métiers et les vues (pages JSP).

- **Enumération des vues (les pages JSP)**

L'utilisation d'une application web est rendue possible par ses interfaces d'envoi de requêtes clientes et de visualisation des réponses. L'énumération des vues d'IHM (Interface Homme Machine) permet de se faire un idée des pages JSP (vues) composants la couche de présentation.

Le tableau suivant donne la liste complète des vues du système futur :

Pages				Description des pages
MainPage[18]	**PgLogin**			Page d'authentification
	PgAdminUser	PgModifUser		Page de modification d'un compte utilisateur
		PgAjoutUser		Page de création d'un compte utilisateur
		PgListUser		Page de visualisation des comptes crées
		PgConsultUser		Page de consultation des informations d'un compte utilisateur
	PgMouvement	PgRechMvt		Page de rechercher d'un mouvement
		PgListMvt		Page de visualisation des mouvements enregistrés
		PgAjoutMvt		Page d'enregistrement d'un mouvement
		PgConsultMvt		Page de consultation d'un mouvement
		PgEditBrouillard		Page d'édition des brouillards
		PgRapprchBanq		Page de rapprochement bancaire
		PgMvtNonRapprch		Page des mouvements non rapprochés
	PgParam	PgParamCatAff	PgAjoutCatAff	Page d'ajout d'un type d'affaire
			PgModifCatAff	Page de modification d'une catégorie d'affaire
			PgListCatAff	Page de visualisation des types d'affaire ajoutée

[18] La page principale de l'application, page d'accueil

			PgRechCatAff	Page de recherche d'une catégorie d'affaire
		PgParamTypMvt	PgAjoutTypMvt	Page d'ajout d'un type de mouvement
			PgModifTypMvt	Page de modification d'un type de mouvement
			PgListTypMvt	Page de visualisation des types de mouvements ajoutés
			PgRechTypMvt	Page de rechercher d'un type de mouvement
		PgParamVille	PgAjoutVille	Page d'ajout d'une ville
			PgModifVille	Page de modification d'une ville
			PgListVille	Page de visualisation des villes ajoutées
			PgRechVille	Page de recherche d'une ville
		PgParamLang	PgAjoutLang	Page d'ajout d'une langue
			PgModifLang	Page de modification d'une langue
			PgListLang	Page de visualisation des langues ajoutées
			PgRechLang	Page de recherche d'une langue
		PgParamBanq	PgAjoutBanq	Page d'ajout d'une banque
			PgModifBanq	Page de modification d'une banque
			PgListBanq	Page de visualisation des banques ajoutées
			PgRechBanq	Page de recherche d'une banque
		PgParamCptSpec	PgAjoutCptSpec	Page d'ajout d'un compte spécial

			PgModifCptSpec	Page de modification d'un compte spécial
			PgListCptSpec	Page de visualisation des comptes spéciaux ajoutés
			PgRechCptSpec	Page de recherche d'un compte spécial
	PgCompte	PgComptCARPA	PgConsultRIB	Page de consultation du RIB
			PgModifCompt	Page de modification du compte CARPA
			PgEditEtatAff	Page d'édition des états des affaires ajoutées
		PgAvocat	PgAjoutAvocat	Page d'ajout d'un avocat
			PgModifAvocat	Page de modification d'un avocat
			PgListAvocat	Page de visualisation des avocats ajoutés
			PgConsultAvocat	Page de consultation des informations sur un avocat
			PgRechAvocat	Page de recherche d'un avocat
		PgBuroSec	PgAjoutBuroSec	Page d'ajout d'un bureau secondaire
			PgModifBuroSec	Page de modification d'un bureau secondaire
			PgListBuroSec	Page de visualisation des bureaux secondaires ajoutés
			PgConsultBuroSec	Page de consultation des informations d'un bureau secondaire
			PgRechBuroSec	Page de recherche d'un bureau secondaire
		PgAffaire	PgAjoutAff	Page d'ajout d'un sous compte affaire
			PgModifAff	Page de modification d'un sous compte affaire
			PgListAff	Page de visualisation des sous compte affaire
			PgConsultAff	Page de consultation de l'état d'un sous compte affaire
			PgRechAff	Page de recherche d'un sous compte affaire

	PgEdit	PgEditBord	Page d'édition d'un bordereau
		PgEditLettre	Page d'édition d'une lettre chèque ou de relance
		PgEditHisto	Page d'édition d'historique d'affaire ou de mouvement

Tableau IV-2.1: Liste descriptive des vues d'IHM

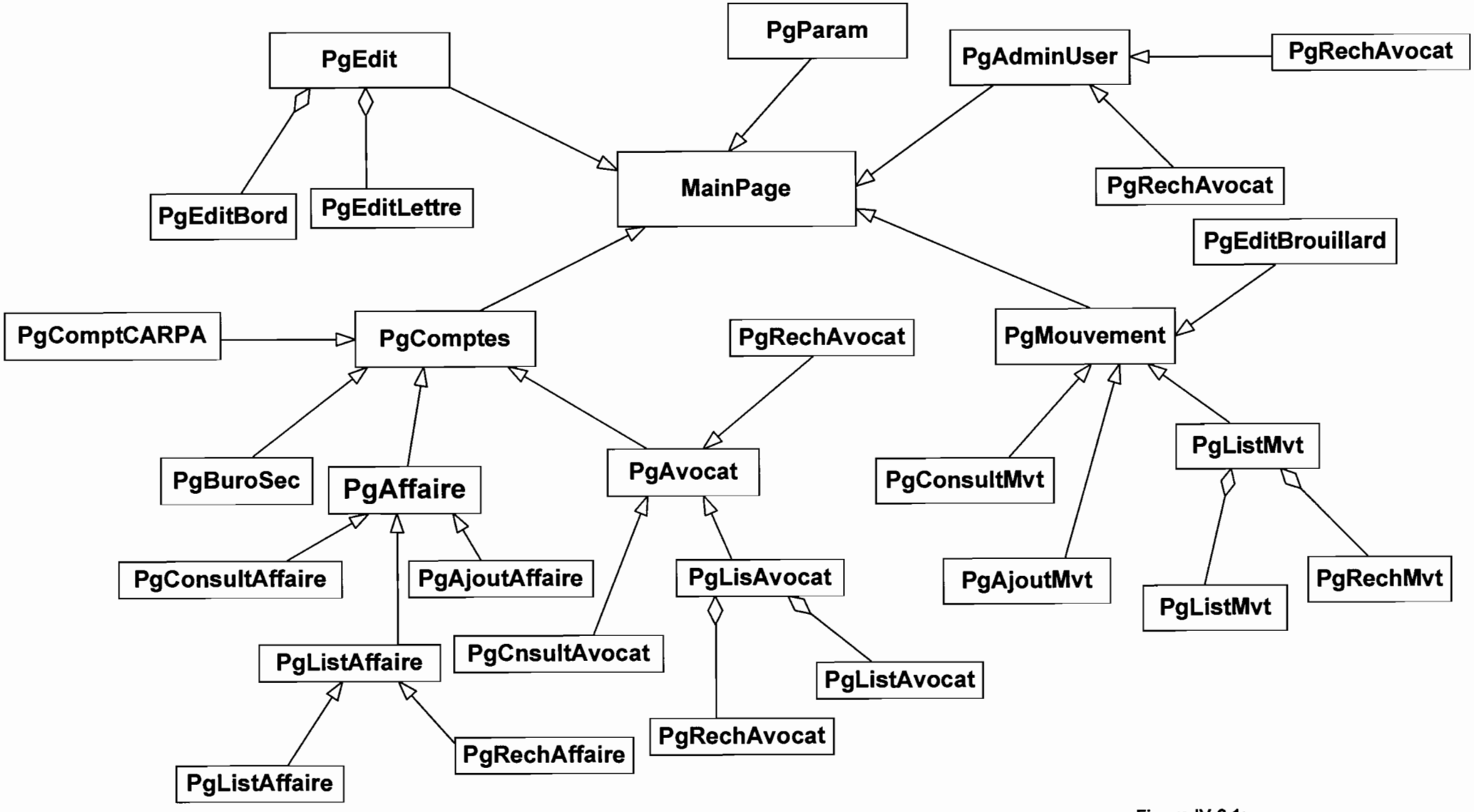


Figure IV-2.1:
Une vue de l'organisation des vues (IHM) du système futur

- **Description du comportement des vues**

Cette étape de notre conception consiste à définir les comportements des pages JSP. Une modélisation UML s'impose alors pour concevoir la dynamique des IHM. En effet, à partir d'une page, un utilisateur déclenche des actions ou contrôle des activités. À chaque instant, la vue doit refléter ce qu'il a le droit ou non de faire, en désactivant des boutons, des champs, des menus ou tout autre élément visuel de contrôle. La plupart des vues ont en fait un comportement de machine à états ; le développement d'un diagramme d'états est alors très utile à la conception de leur comportement.

Le diagramme d'état suivant décrit les différents comportements de la page « PgMouvement » lorsqu'une action est déclenchée par un utilisateur.

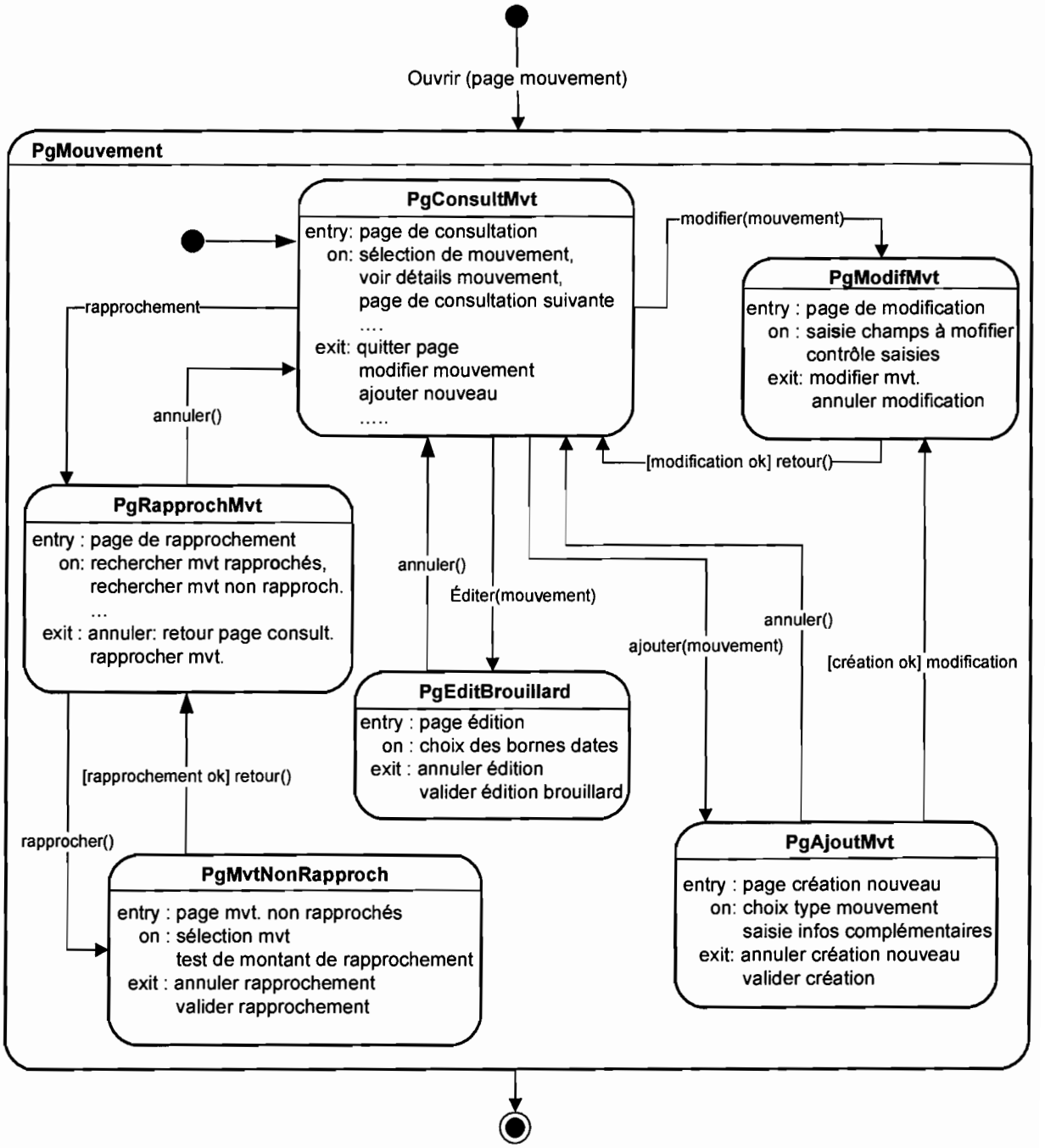


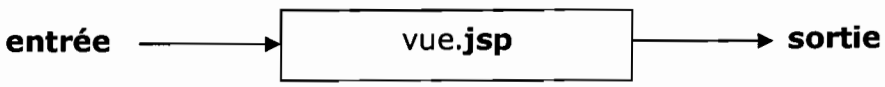
Figure IV-2.2 : État de la page Mouvement

- **La conception des pages JSP**

Les vues de la couche de présentation sont essentiellement composées de pages JSP, par conséquent on définira pour chaque vue, une page JSP et pour chacune d'elle :

- on dessinera l'aspect de la page
- on déterminera quelles sont les parties dynamiques de celle-ci :
  - les informations à destination de l'utilisateur qui devront être fournies par la Servlet-Contrôleur en paramètres à la vue JSP;
  - les données de saisie qui devront être transmises à la Servlet-Contrôleur pour traitement. Celles-ci devront faire partie d'un formulaire HTML.

Le schéma de chaque vue donnera ce qui suit :



- les **entrées** sont les données que devra fournir la Servlet à la page JSP soit dans la requête (request) ou la session (session).
- les **sorties** sont les données que devra fournir la page JSP à la Servlet. Elles font partie des formulaires HTML et la Servlet-Contrôleur les récupèrera par une opération du type **request.getparameter(...)**.

Le code Java/JSP de chaque vue a la forme suivante :

```
<%@ page ... %> // importations de classes le plus souvent
<%!
// variables d'instance de la page JSP (=globales)
// nécessaire que si la page JSP a des méthodes partageant des variables (rare)
...
%>
<%
// récupération des données envoyées par la Servlet
// soit dans la requête (request) soit dans la session (session)
...
%>
<html>
...
// on cherchera à minimiser ici le code Java
...
</Html>
```

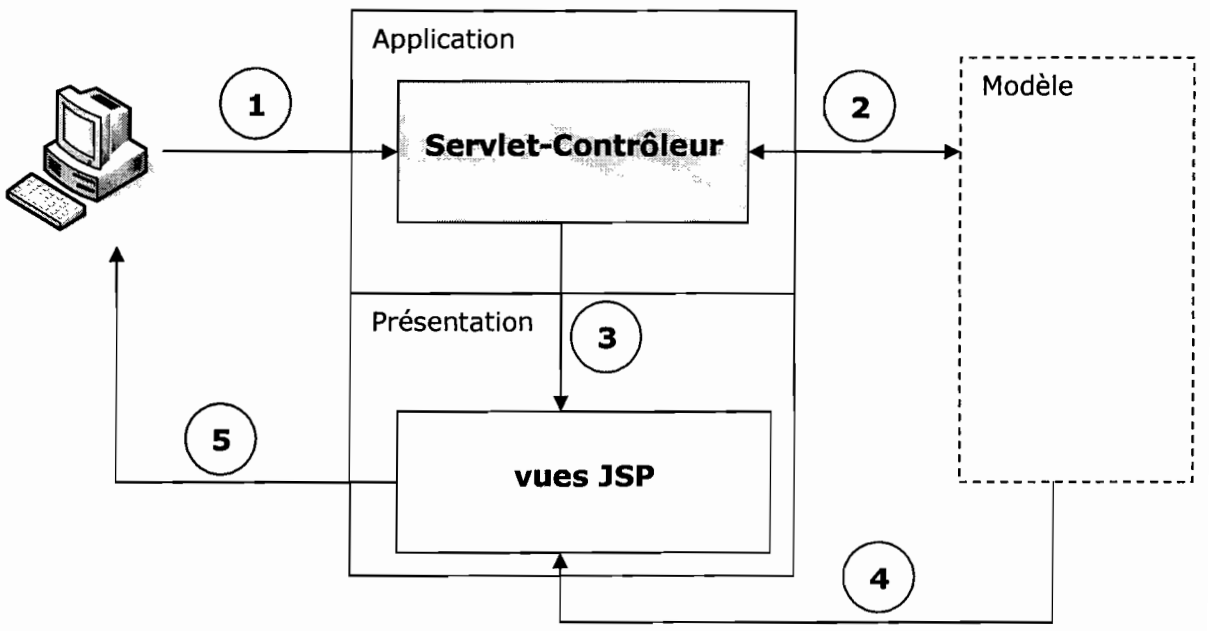
3. CONCEPTION DE LA COUCHE APPLICATIVE

Le rôle de la couche applicative consiste à piloter les processus d'interactions entre l'utilisateur et le système. Cette notion peut paraître abstraite, mais il s'agit généralement de mettre en oeuvre toutes les règles nécessaires au maintien d'une application cohérente et à l'optimisation des échanges client/serveur et/ou des requêtes HTTP.

La conception de la couche applicative du système futur consistera à concevoir la Servlet-Contrôleur qui constitue le point d'entré de notre application et aussi le moteur de traitement des requêtes clientes.

La Servlet-Contrôleur est le composant de notre architecture qui reçoit les requêtes (actions, demandes) des clients, les traite et les transmet aux composants chargés de traiter les données. Elle les dirige ensuite vers des composants appropriés responsables des vues : les pages JSP. Une Servlet est conçue pour recevoir les requêtes des clients et leur retourner une réponse, ce qui est précisément le rôle du contrôleur. Les requêtes clientes sont en fait des sollicitations des clients sous forme d'actions directement liées aux services offerts par le système, nous faisons donc allusion aux différents cas d'utilisation de notre système d'information.

La figure suivante donne illustration du rôle joué par le contrôleur :



1. **Le client envoie les requêtes à la Servlet-Contrôleur**
2. **La Servlet-Contrôleur modifie les données dans le modèle**
3. **La Servlet-Contrôleur transmet les requêtes aux pages JSP pour affichage**
4. Les JSP lisent les données dans le modèle
5. Les JSP envoient l'affichage aux clients

Les différentes requêtes des clients adressées à la Servlet-Contrôleur contiendront un paramètre appelé [**action**] précisant l'action demandée par le client. Le tableau suivant donne la liste des actions possibles traitées par le contrôleur concernant les requêtes en direction de la classe métier « Affaire ».

action	signification	action du contrôleur	réponses possibles
action=**ListAffaire**	Le client veut la liste des affaires enregistrées	demande la liste des affaires enregistrées à la couche métier	-PgListAffaire -PgErreur
action=**ConsultAffaire**	Le client demande à voir les informations sur une affaire affichée dans PgListAvocat	demande une affaire de la couche métier	-PgConsultAffaire -PgErreur
action=**modifierAffaire**	Le client demande à modifier les informations sur une affaire	demande la mise à jour des informations sur une affaire à la couche métier	-PgModifAffaire -PgConfirmation -PgErreur
action=**rechercherAff**	Le client demande à retrouver une affaire enregistrée	demande une affaire à la couche métier dont la référence est fournie	-PgRechAffaire -PgConsultAffaire -PgErreur
action=**editAffaire**	Le client demande à éditer une affaire	Demande les informations sur une affaire à la couche métier pour édition	-PgConsultAffaire -PgEditAffaire -PgErreur
action=**ajoutAffaire**	Le client demande à ajouter un sous compte affaire	Demande d'ajout d'un sous compte affaire à la couche métier	-PgAjoutAffaire -PgConfirmation -PgErreur

- **La Servlet-Contrôleur :**

Cette Servlet Java est un composant implémentant l'interface *javax.servlet.Servlet*[19]. Son invocation est la conséquence de la requête du client, dirigé vers cette Servlet. Bien que cela ne soit pas obligatoire, généralement les Servlets sont associées à l'environnement Web et aux requêtes HTTP. Le serveur Web reçoit une demande adressée à cette Servlet sous la forme d'une requête HTTP. Il transmet la requête à la Servlet-Contrôleur, puis renvoie la réponse fournie par elle au client. La Servlet reçoit également les paramètres de la requête envoyée par le client. Elle peut alors effectuer toutes les opérations nécessaires pour construire la réponse avant de renvoyer celle-ci à travers la vue (page JSP) concernée.

Le code de la Servlet-Contrôleur a deux méthodes distinctes dont l'une est facultative :

- la méthode **init()**, facultative, qui sert à :
  - récupérer les paramètres de configuration de l'application dans le fichier web.xml (fichier de configuration de l'application web) de celle-ci ;
  - éventuellement créer des instances de classes métier qu'elle sera amenée à utiliser ensuite ;
  - gérer une éventuelle liste d'erreurs d'initialisation qui sera renvoyée aux futurs utilisateurs de l'application. Cette gestion d'erreurs peut aller jusqu'à l'envoi d'un message à l'administrateur de l'application afin de le prévenir d'un dysfonctionnement.
- la méthode **doGet()** ou **doPost()** selon la façon dont la Servlet-Contrôleur reçoit ses paramètres de ses clients. La Servlet peut commencer par lire la valeur de ce paramètre puis déléguer le traitement de la requête à une méthode privée interne chargée de

[19] Pour créer une Servlet il est indispensable de mettre en oeuvre l'interface *javax.servlet.Servlet* en général héritant des classes javax.servlet.GenericServlet ou javax.servlet.http.HttpServlet et permettant au conteneur d'assurer le cycle de vie de la Servlet.

traiter ce type de requête. Lors de l'écriture du code de la Servlet-Contrôleur, on évitera au maximum d'y intégrer du code métiers afin de pérenniser cette notion de séparation entre les couches de l'application et de permettre au chef d'équipe (contrôleur) de mieux jouer son rôle de réception des demandes clientes pour les faire exécuter par des personnes appropriés (classes métiers).

- **Illustration code de la Servlet-Contrôleur:**

```
public class Servlet-Contrôleur extends HttpServlet {

//Méthode GET
public void doGet (HttpServletRequest request, HttpServletResponse   response)
   throws ServletException,IOException {
      String action = req.getParameter("action");
      if("ConsultAffaire".equals(action)) {
       //traitement de la Servlet vers le modèle et les pages JSP
      }
        else if ("ListerAffaire".equals(action)) {
       //traitement de la Servlet vers le modèle et les page JSP
        }
       //autres listes d'actions pour le tratement des requêtes clients

//Méthode POST
public void doPost (HttpServletRequest request, HttpServletResponse response)
 throws IOException, ServletException {
// on passe la main au GET
doGet (request, response);
}
}
```

4. CONCEPTION DE LA COUCHE METIER

La couche métier dans l'architecture J2EE du futur système est constituée de composants qui gèrent la logique métier principale de l'application. Elle propose les interfaces nécessaires aux composants de services sous-jacents.

Cette couche propose des services d'accès à ces objets à travers des méthodes de création, recherche, modification, suppression, etc. Ces méthodes contiennent les règles de gestion associées aux différentes opérations.

La couche métier contient essentiellement trois types de composants :

- des objets métier qui sont des représentations de concepts métiers sous la forme d'objets métier (simples classes comportant des attributs et des méthodes de manipulation);
- des fabriques d'objets métier dont le rôle est de gérer le cycle de vie des objets métier (création, recherche, modification, destruction, constructeur, etc.) ; on implémente une fabrique par objet métier ;
- des classes implémentant les comportements extrinsèques à plusieurs objets métier : règle de manipulation portant sur un ensemble d'objets, règle d'intégrité entre plusieurs objets, etc.

La conception de la couche métiers revient donc à concevoir les composants de cette couche.

La conception des deux premiers composants de la couche métier consiste à concevoir les classes métiers structurants de la CARPA-BF avec leurs méthodes de manipulation et d'accès aux différents services. Ces deux premiers composants sont composés des différentes classes détaillées conçues un peu plus haut (*page 37*).

La conception des classes implémentant les comportements extrinsèques consiste en la conception des classes mettant en oeuvre des règles d'intégrité entre des objets, et aussi des classes permettant la mise en œuvre de certains fonctionnalités du système, à travers des classes de stéréotype « utility ou bondary », telles le cryptage des mots de passe, la gestion des impressions, la génération automatique des sous comptes, etc.

- **Description de ces classes**

Stéréotype : **entity**					
CLASSE : **InfoComptBanq,** les informations sur un compte bancaire					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
privée	NumCompt	1	String	Numéro du compte bancaire	frozen
public	LaBanque	1	Banque	La banque	readOnly
public	Proprio	1	Personne	Propriétaire du compte	readOnly
Méthodes					
Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	InfoComptBanq	InfoComptBanq	NumCompt,...	Le constructeur	constructor
public	String	GetNumCompt	void	Retourne NumCompt	query

Stéréotype : **entity**					
CLASSE : **Parle,** intégrité entre la classe Avocat et Langue					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
public	Parlant	1	Avocat	L'Avocat parlant	readOnly
public	Lang_Parle	1	Langue	La langue parlée	readOnly
Méthodes					
Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	Parle	Parle	Parlant, Lang_Parle	Le constructeur	constructor

Stéréotype : **entity**					
CLASSE : **AvocatAff,** intégrité entre la classe Avocat et Affaire					
Attributs					
Visibilité	**Nom**	**Multiplicité**	**Type**	**Description**	**Propriété**
public	LAvocat	1	Avocat	L'Avocat intervenant	readOnly
public	LAffaire	1	Affaire	L'affaire concernée	readOnly
Méthodes					
Visibilité	**Type**	**Nom**	**Paramètres**	**Description**	**Propriété**
public	AvocatAff	AvocatAff	LAvocat, LAffaire	Le constructeur	constructor

Stéréotype : **entity**

CLASSE : **ClientAff,** intégrité entre la classe Client et Affaire

Attributs

Visibilité	Nom	Multiplicité	Type	Description	Propriété
public	LeClient	1	Client	Le client intervenant	readOnly
public	LAffaire	1	Affaire	L'affaire concernée	readOnly

Méthodes

Visibilité	Type	Nom	Paramètres	Description	Propriété
public	ClientAff	ClientAff	LAvocat, LAffaire	Le constructeur	constructor

Stéréotype : **entity**

CLASSE : **BuroAvocat,** intégrité entre la classe BuroSec et Avocat

Attributs

Visibilité	Nom	Multiplicité	Type	Description	Propriété
public	LAvocat	1	Avocat	L'Avocat membre ou propriétaire	readOnly
public	LeBuro	1	BuroSec	Le bureau secondaire	readOnly

Méthodes

Visibilité	Type	Nom	Paramètres	Description	Propriété
public	BuroAvocat	BuroAvocat	LAvocat, LeBuro	Le constructeur	constructor

Stéréotype : **utility**

CLASSE : **Fonction,** collection de méthodes

Méthodes

Visibilité	Type	Nom	Paramètres	Description
ublic	String	CryptDecryptPwd	Password	Cryptage décryptage Password
ublic	Booléen	verifFormat	entrée saisie	Vérifier cohérence d'un format
ublic	Booléen	IsJourFerie	Date	Vérifier si un jour est férié
ublic	Booléen	verifBornDate	Dates	Vérifier les bornes des dates
ublic	void	SendAlerte	void	Envoyer une alerte
ublic	String	GeneratSCompt	Type de compte	Générer un sous compte
ublic	void	SendLog	Infos sur le log	Envoyer une information log
--	---	---	---	---

5. CONCEPTION DE LA COUCHE D'ACCES AUX DONNEES

La couche d'accès aux données est une couche très stratégique de l'architecture J2EE de notre futur système. Elle a pour rôle d'assurer la persistance des objets métier grâce à l'API JDBC, le standard d'accès aux données dans la plate-forme J2EE.

La conception de cette couche pour notre système consiste à concevoir les classes de stéréotypes "boundary", classes frontières, permettant d'assurer le stockage des objets métiers vers une base de données.

Ces classes sont en fait des classes implémentant la logique d'accès aux données à travers des collections méthodes pour insérer, sélectionner, mettre à jour les objets métiers du système dans la base de données.

Pour un objet métier, classes persistantes, nous allons implémenter une classe frontière, en utilisant les classes de l'API JDBC, permettant de rendre persistant cet objet métiers. Ce travail sera répété pour chaque objet métier.

La mise en œuvre de cette couche commence tout d'abord par l'implémentation de deux classes, l'une permettant la déclaration des pilotes de la base de données et l'autre pour la connexion à la base de données, cela pour faciliter leur réutilisation, selon les différentes étapes pour l'accès à une base de données avec l'API JDBC.

Les extraits de code suivants donnent une illustration de ces classes :

```
import java.sql.*;

public class DeclarePilote {
  public boolean driverDeclared() {
    try {

       Class.forName ("pilote JDBC du SGBDR utilisé");
       return true;
       }
    catch (Exception E) {
       return false;
       }
    }
}
```

Illustration de la classe **DeclarePilote** avec L'API JDBC

```
import java.sql.*;

public class ConnectToDB {

  public boolean Connect() {
    try {
      String url = "l'URL de Connexion à la base de données";
      Connection conn = DriverManager.getConnection (url, "user", "password");
      return true
   }
    catch (SQLException E)
    {
      return false;
    }
  }
}
```

Illustration de la classe **ConnectToDB** avec L'API JDBC

NB : Toutes les classes frontières permettant de gérer la persistance des objets métiers devront intégrer dans leur code des instances des classes précédentes.

Le code suivant donne une illustration de la classe **Avocat_Data**, permettant d'assurer la persistance de l'objet métier Avocat.

```
import java.io.*;
import java.sql.*;

public class  Avocat_Data {

 //méthode pour créer un Avocat
  public boolean StoreAvocat (tous les paramètres d'un avocat){

    if (DeclarePilote.driverDeclared()){

       if (ConnectToDB.Connect()){

          .....
          //requête d'ajout d'un avocat dans le DB..
          .....
          return true;
      }
      else
        return false;
  }
  else
  {
     return false;

  }//fin de store to DB

 //méthode de mise à jour d'un Avocat
 public boolean UpdateAvocat(les paramètres d'un avocat à modifier){
    ......
 }

 //méthode de suppression d'un avocat
 public boolean DeleteAvocat(l'id de l'avocat){
    ......
 }
 ......
}
```

Illustration de la classe **Avocat_Data**

6. GESTION DE LA PERSISTANCE

La gestion de la persistance objet est une problématique complexe, qui est en passe de devenir l'un des volets stratégiques des architectures logicielles. Elle traite de la manière dont on peut sauvegarder et restaurer l'état des objets manipulés au sein des applications, dans le but de prolonger leur durée de vie au-delà de la durée de vie d'une session applicative.

- **Différents modes de stockage des objets**

La réalisation du stockage des instances varie suivant le mode de stockage retenu. Dans tous les cas, la réalisation d'un modèle objet facilite la maintenance des données stockées. Il existe aujourd'hui plusieurs modes de stockage possibles.

- Le système de fichiers est actuellement le moyen le plus rudimentaire de stockage;
- La base de données relationnelle ou SGBDR est un moyen déjà plus sophistiqué. Il existe aujourd'hui une large gamme de SGBDR répondant à des besoins de volume, de distribution et d'exploitation différents. Le SGBDR permet d'administrer les données et d'y accéder par des requêtes complexes. C'est la technique la plus répandue.
- La base de données objet ou SGBDO constitue la méthode la plus élaborée de toutes;
- La base de données XML ou SGBDX est un concept émergeant qui répond au besoin croissant de stocker des documents XML sans risque d'altération de ces derniers;

Bien qu'il existe des systèmes dédiés à la persistance objet (bases de données objet), nous avons choisi de gérer la persistance de nos données avec le SGBD relationnel PostgreSQL 8.0, pour trois raisons principales :

- Prendre en compte la procédure transitoire avec l'existant;
- La popularité des SGBDR;

- Et utiliser la possibilité de tirer parti de fonctionnalités sophistiquées: les SGBDR allient performance et fiabilité, à travers notamment des fonctionnalités de recherche puissantes (langage de requêtage, indexation) et le support de la concurrence d'accès (transactions).

- **Le passage du modèle objet au modèle relationnel**

L'utilisation d'un SGBDR impose un changement de représentation entre la structure des classes et la structure des données relationnelles. Les deux structures ayant des analogies, les équivalences exprimées au *tableau IV-6.1* sont utilisées pour en réaliser le rapprochement.

Une classe définit une structure de données à laquelle souscrivent des instances; elle correspond donc à une table du modèle relationnel : chaque attribut donne lieu à une colonne, chaque instance stocke ses données dans une ligne (*n-uplets*) et son OID (Object Identifier) sert de clé primaire où il est défini.

Certains attributs de type complexe ne correspondent à aucun des types de SQL; on rencontre fréquemment ce cas pour les attributs représentant une structure de données. Un type complexe peut être conçu :

- soit avec plusieurs colonnes, chacune correspondant à un champ de la structure ;
- soit avec une table spécifique dotée d'une clé étrangère pour relier les instances aux valeurs de leur attribut complexe.

Modèle Objet	Modèle relationnel
Classe	Table
Attribut de type simple	Colonne
Attribut de type composé	Colonnes ou clé étrangère
Instance	n-uplets
OID	Clé primaire
Association	Clé étrangère ou Table de liens
Héritage	Clé primaire identique sur plusieurs tables

Tableau IV-6.1 : Équivalences entre les concepts objets et relationnels

Le diagramme suivant (figure IV-6.1) illustre la conception du stockage de la classe Mouvement dans la table **TblMouvement** correspondante. UML définit spécifiquement un stéréotype « **table**» pour représenter la table d'un schéma relationnel. Nous avons ajouté le stéréotype « **join** » pour exprimer les jointures que définissent les clés étrangères entre tables.

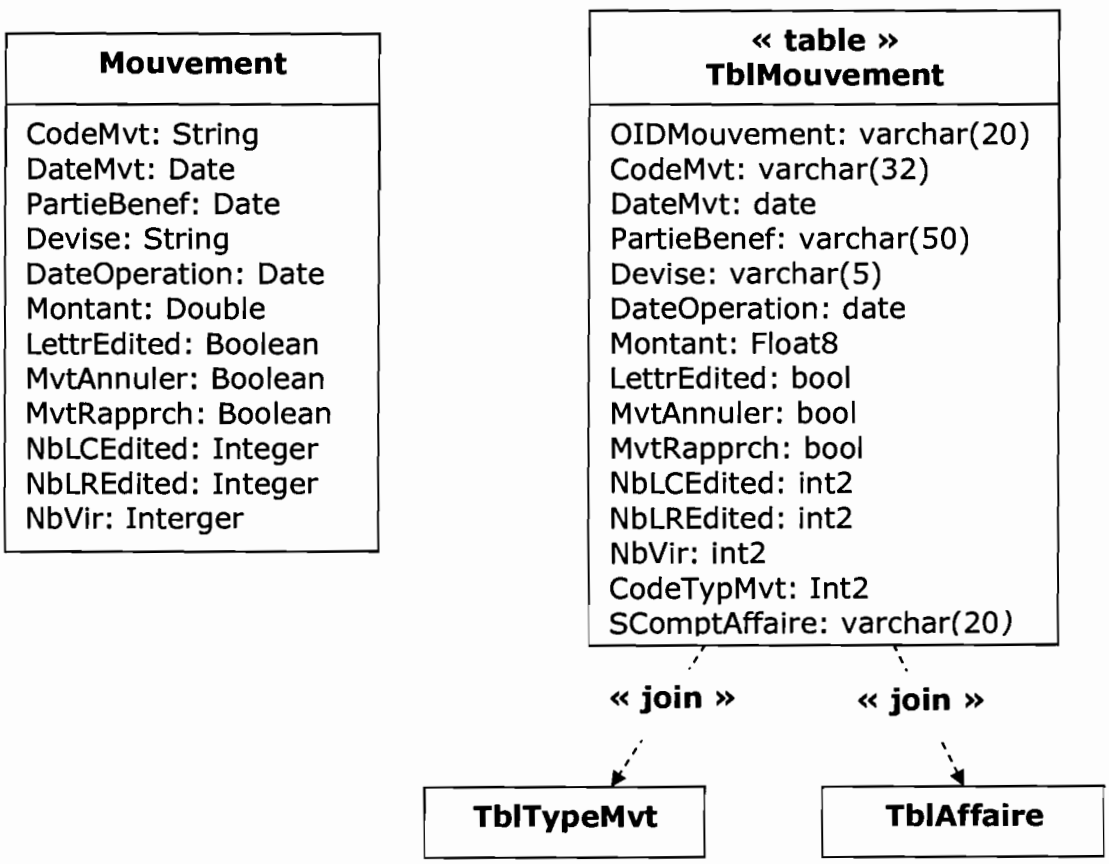


Figure IV-6.1:
Conception du stockage de la classe Mouvement avec une table relationnelle

Il est à noter que le schéma relationnel ne permet pas de différencier les associations des agrégations et des compositions. Quel qu'en soit le type, les relations correspondent en effet à une clé étrangère lorsque la multiplicité le permet. Une association multiple, plusieurs à plusieurs, nécessite en revanche la définition d'une table de liens supplémentaires. Cette dernière stocke des couples de clés étrangères provenant de chacune des deux classes de l'association.

Le diagramme de la figure IV-6.2 schématise la conception des associations, simple et multiple, de la classe **Avocat**.

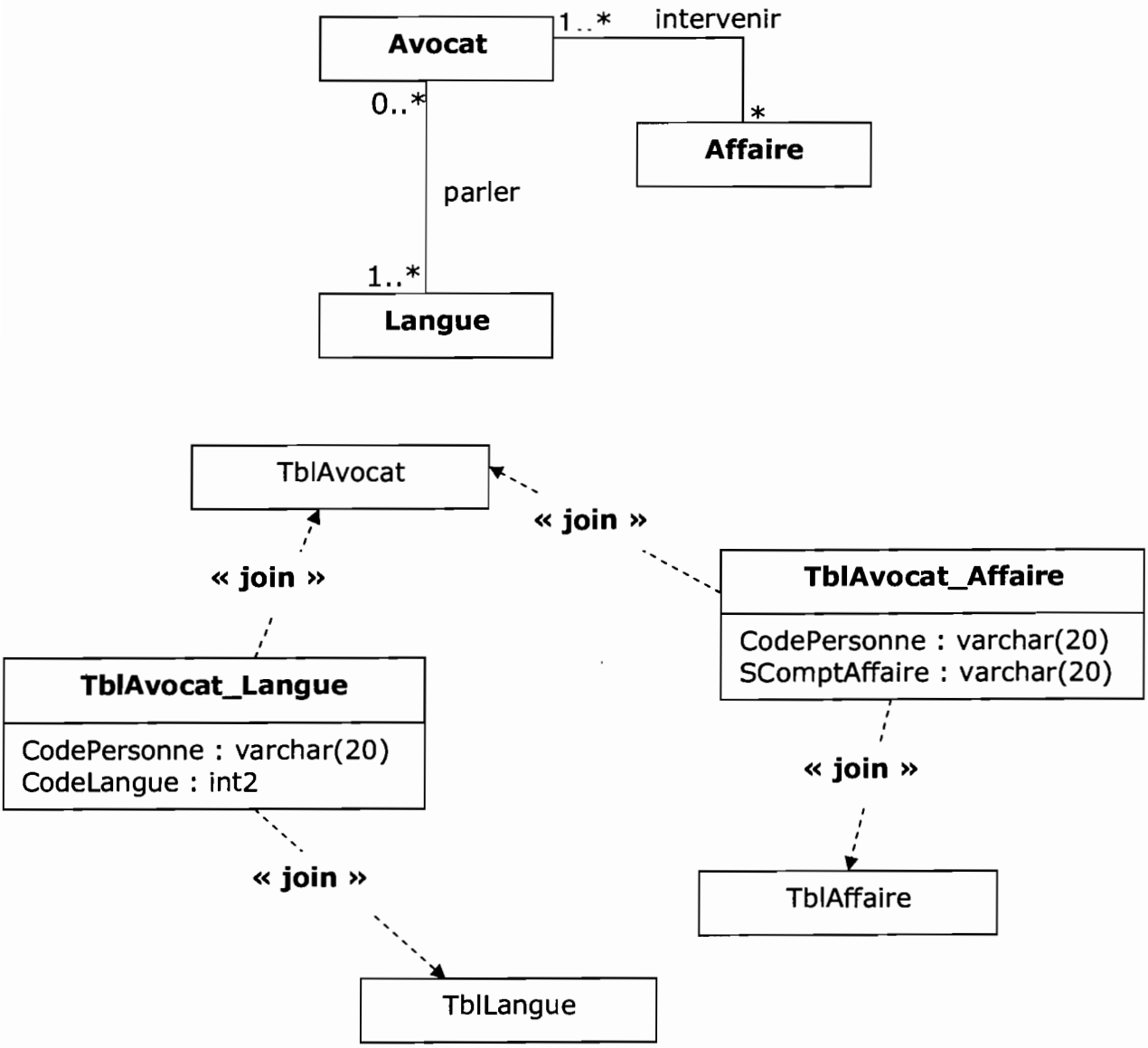


Figure IV-6.2: Illustration d'une table servant à stocker une association multiple

La relation d'héritage se définit par le partage du même OID entre les tables provenant d'une même hiérarchie de classes.

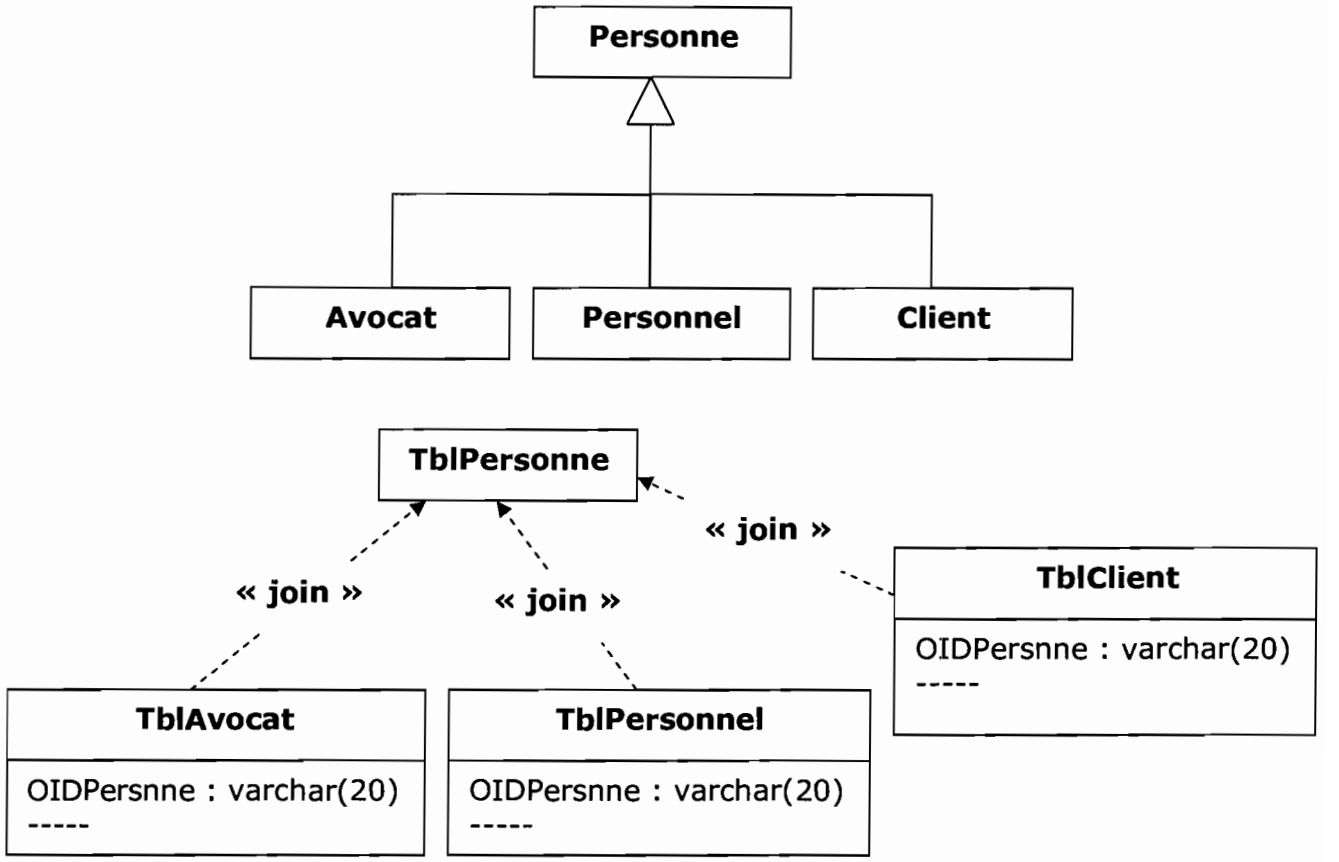


Figure IV-6.3: Illustration du stockage d'une relation de généralisation

7. DIAGRAMME DE SEQUENCE OBJET

Le diagramme de séquence objet, plus détaillé, montre les interactions entre les différents objets du système, à la différence du diagramme de séquence système qui montre les interactions entre acteurs et le système. Ce diagramme permet de présenter l'algorithmique permettant de mettre en oeuvre les différents services rendus par le système.

Nous présenterons ici le diagramme de séquence objet du scénario nominal de quelques cas d'utilisation du système futur.

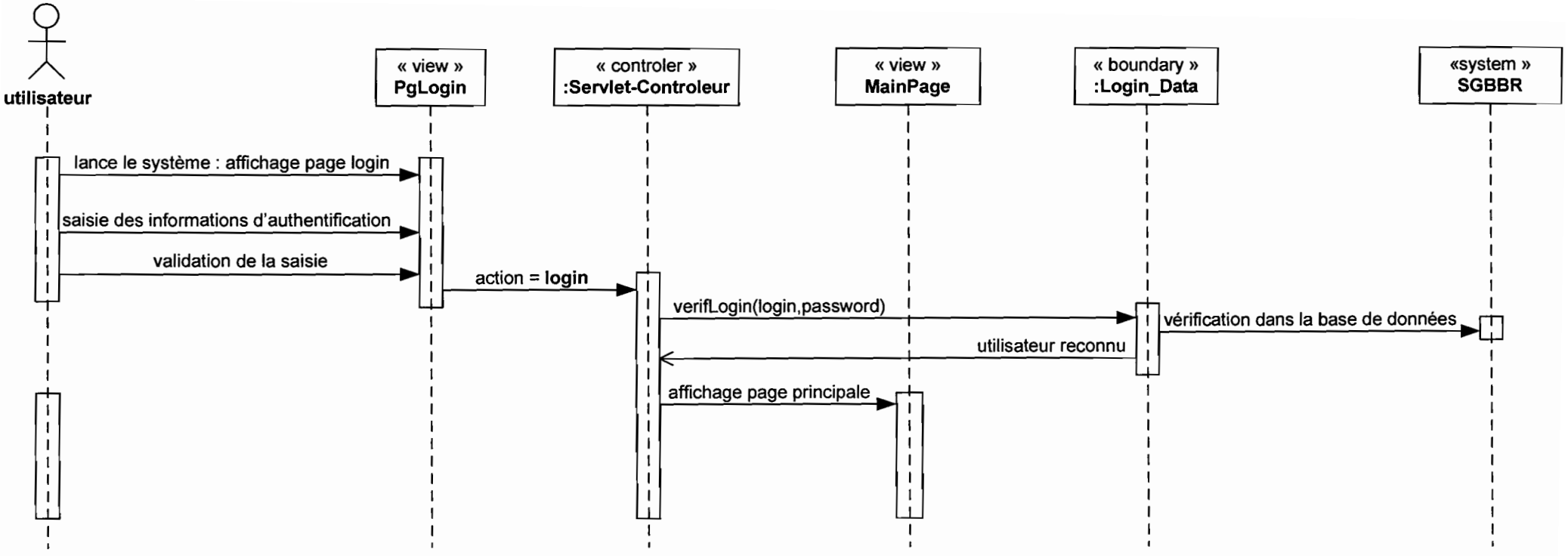


Diagramme de séquence objet 1 : cas d'utilisation **Authentification**

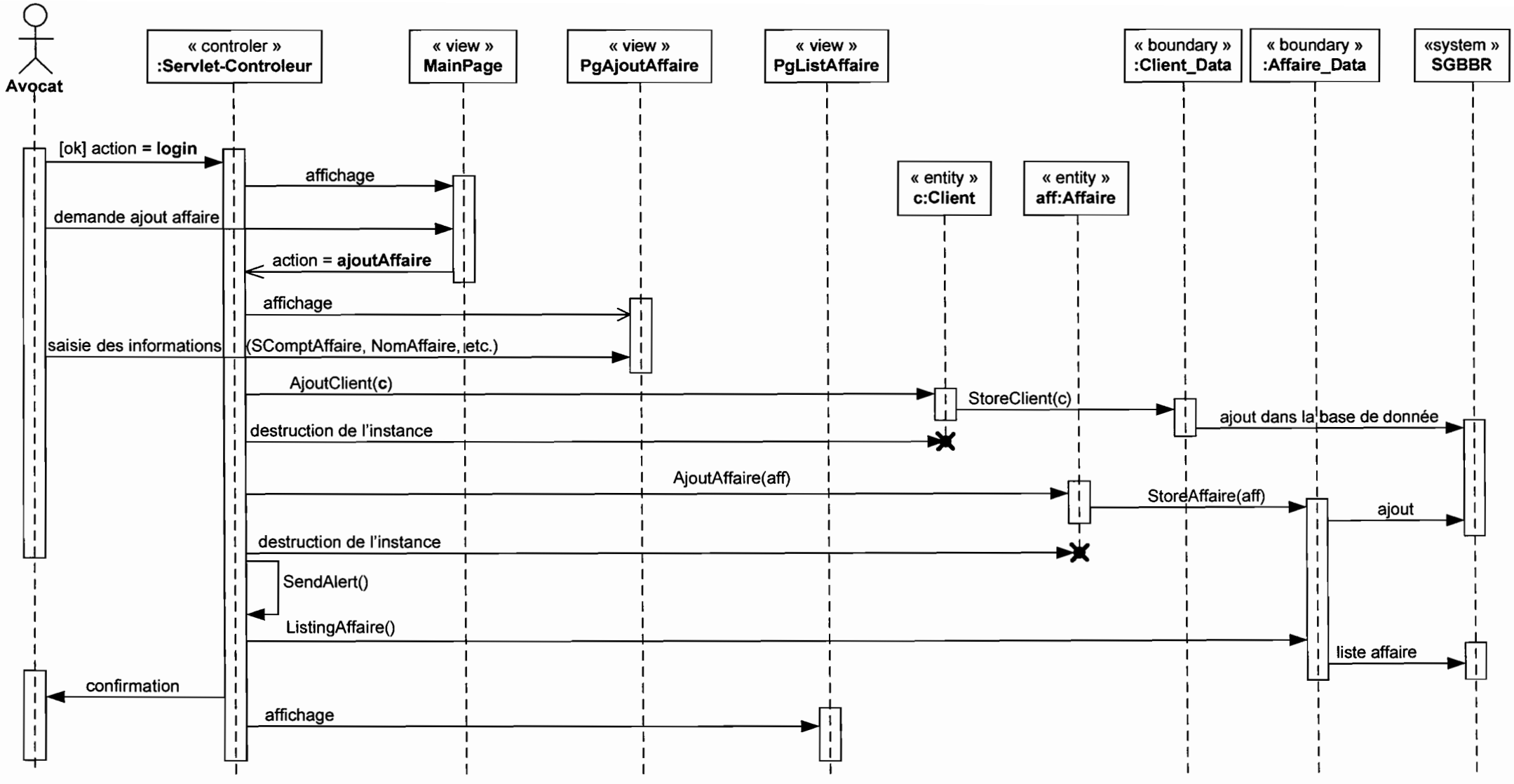


Diagramme de séquence objet 2 : cas d'utilisation **Ajout sous-compte affaire** par un Avocat

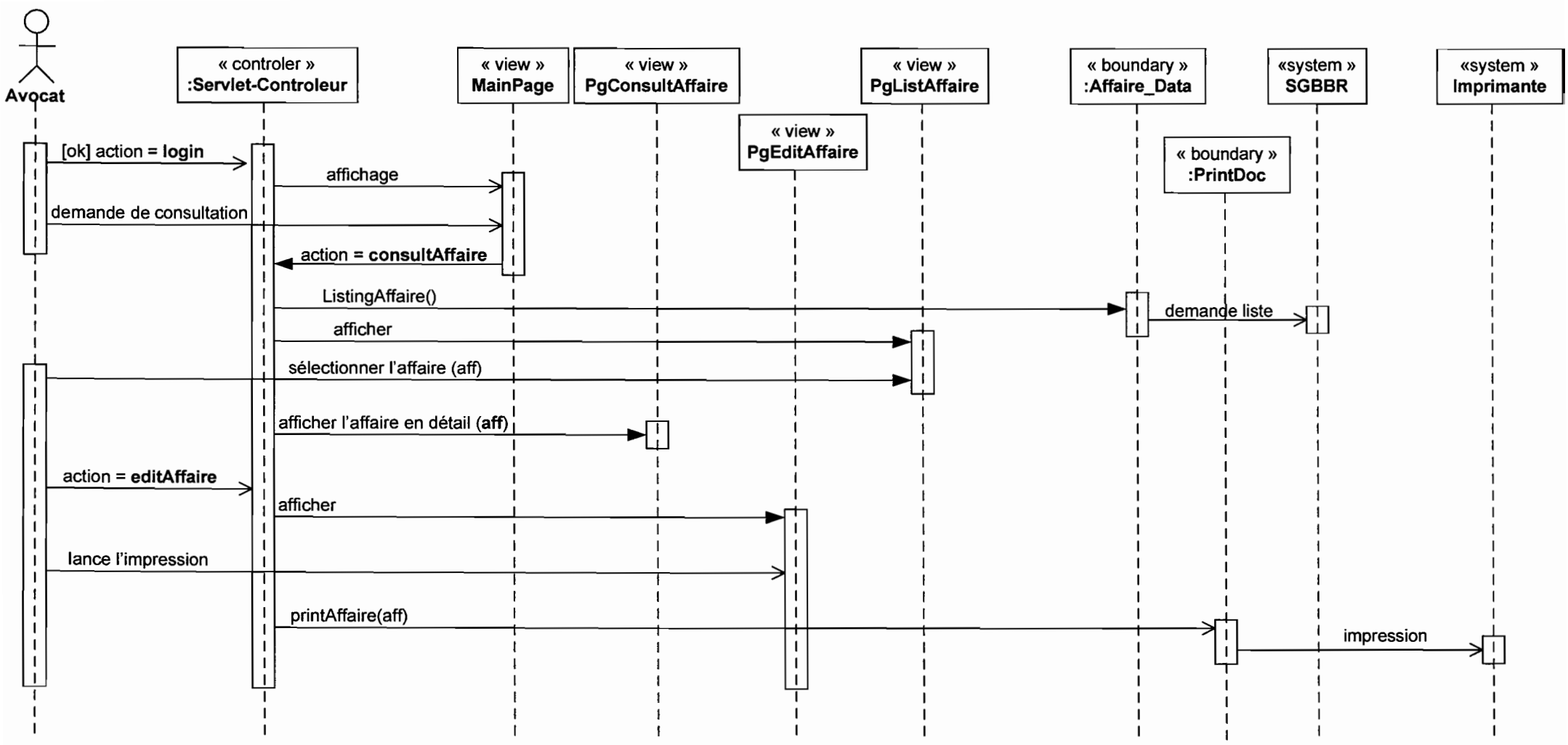


Diagramme de séquence objet 3 : cas d'utilisation **Consultation état d'un affaire** par un Avocat

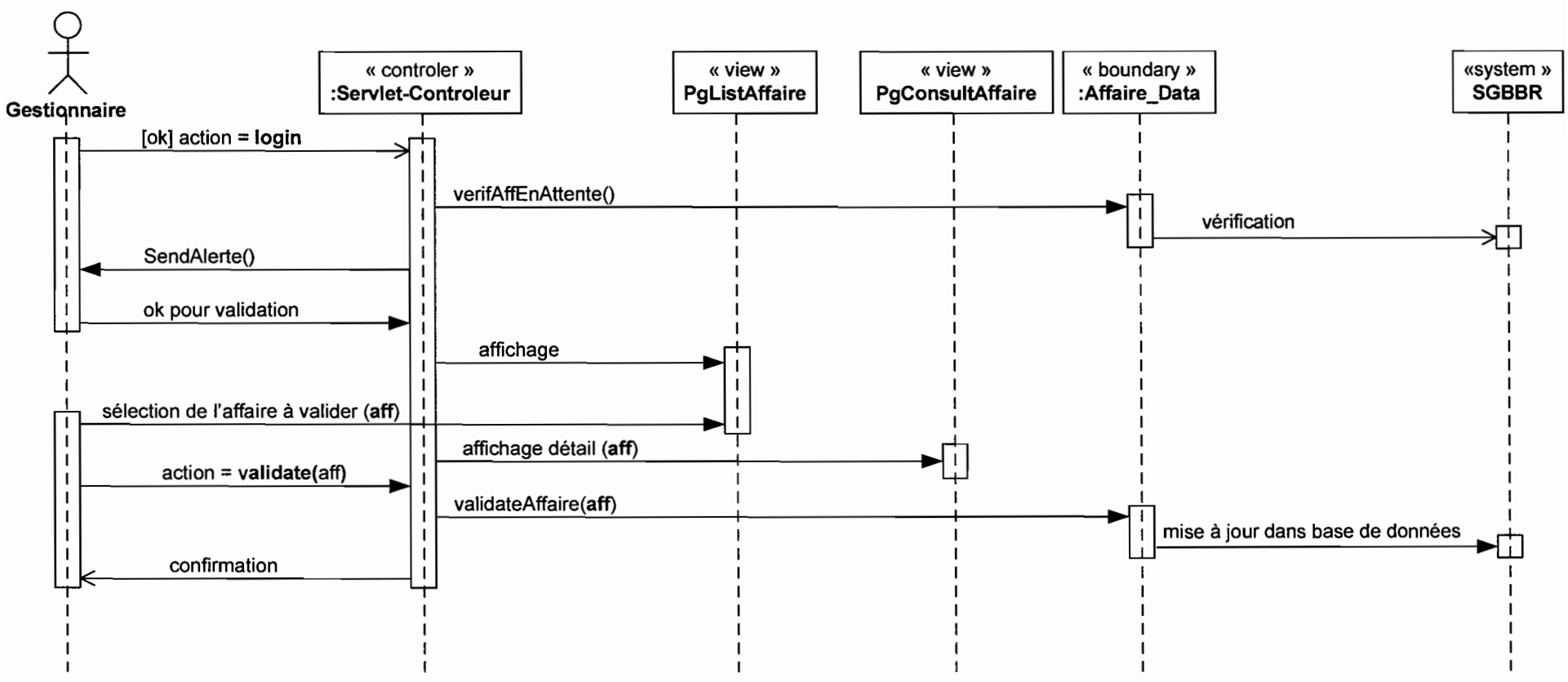


Diagramme de séquence objet 4 : cas d'utilisation **Validation** par le Gestionnaire

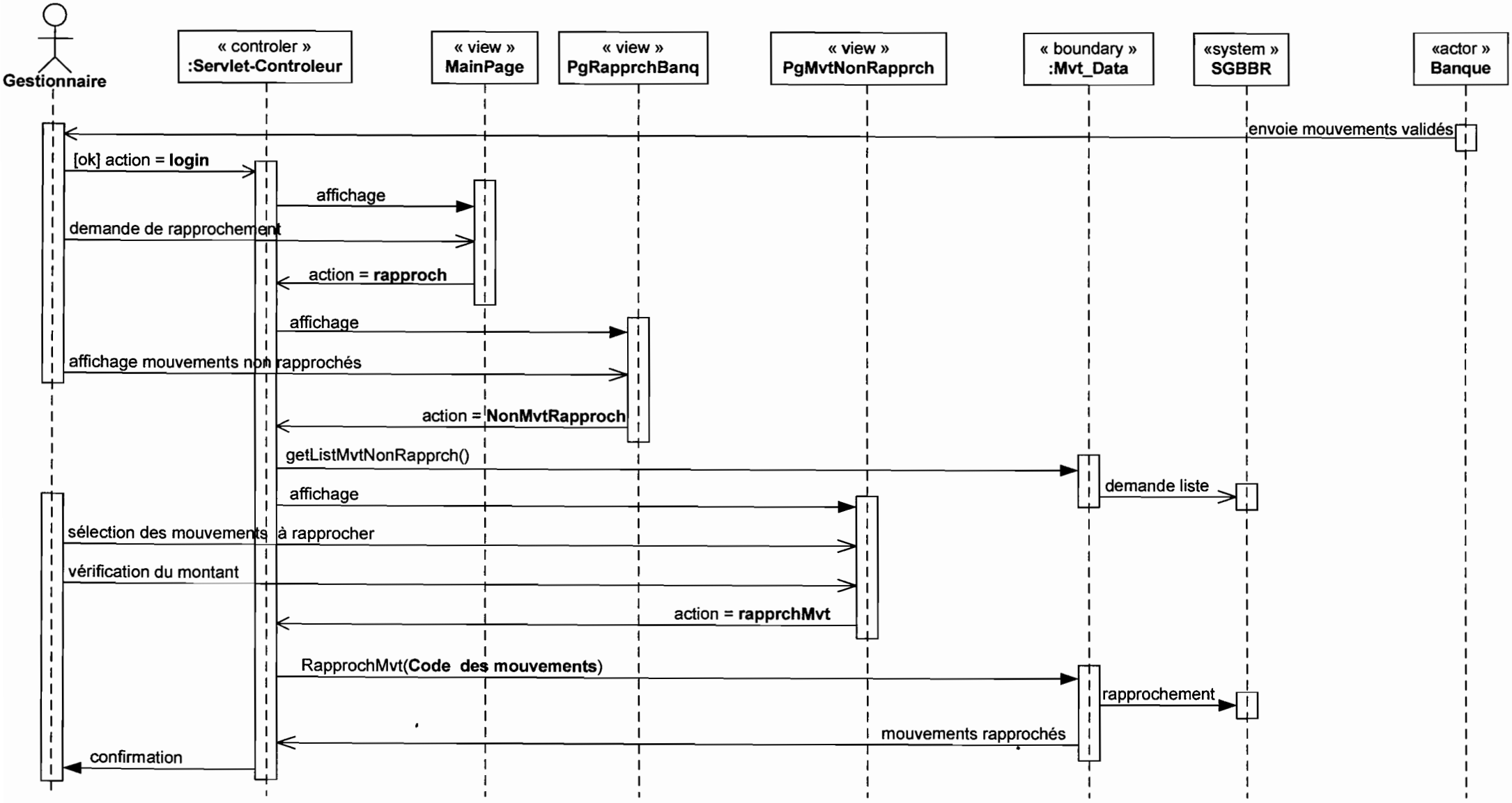


Diagramme de séquence objet 5 : cas d'utilisation **Rapprochement Mvt.** par le Gestionnaire

V. RAPPELS SUR LES DETAILS TECHNIQUES DU FUTUR SYSTEME

1. ARCHITECTURE RESEAU DU SYSTEME FUTUR

→*(Voir Annexe page 93).*

2. BESOINS MATERIELS ET LOGICIELS

Désignation	**Caractéristiques**	**Qté**	**Prix unitaire (FCFA)**	**Montant (FCFA)**
Serveur	Serveur HP ProLiant ML310 G2 : Processeur Intel® Pentium® 4 à 3,20 GHz (3,20 GHz), Disque dur SCSI Ultra320 72 Go enfichable à chaud, 10 000 tr/min, format 1", 1 Go de RAM en standard Lecteur/graveur DVD et RW HP 16X max demi-hauteur	01	1 230 000	1 230 000
Ordinateur (clients)	Pentium IV & III	04	0	Existant
Ordinateur (serveur Firewall)	Pentium III, 800 MHz, 10 Go Disque Dur, 512 RAM.	01	350 000	350 000
Routeur	"Cisco 1721" Routeur externe, Cisco IOS, protocole SNMP, 32 Mo de mémoire vive, Ethernet 10 /100 (Cat SmartNet 2)	01	930 000	930 000
Carte Réseau	D-Link DFE-530TX Carte réseau PCI 10/100	03	12 500	37 500
Imprimante	HP LaserJet 1200	03	0	Existant
Onduleur	APC Back-UPS RS - 800VA	01	165200	165200
Switch	D-Link DES-1008D - Switch 8 Ports 10/100	01	115 000	115 000

Antivirus	ClamWin Antivirus Version 0.88.3.1	01	Gratuit	Gratuit
Système d'exploitation serveur	Debian GNU/Linux 3.1 - Sarge	01	Gratuit	Gratuit
Firewall	IPCop v1.4.10	01	Gratuit	Gratuit
SGBDRO	PostgreSQL 8.1	01	Gratuit	Gratuit
IDE de développement	Eclipse-SDK 3.1.2 pour Windows	01	Gratuit	Gratuit
Serveur d'application (container de Servlets)	Jakarta Tomcat 5.0.28 incluant le serveur web Apache	01	Gratuit	Gratuit
Liaison spécialisée	LS-Fasonet de 128 kbits/s à usage communautaire non commercial	01	326 000	326 000
Nom de domaine	Un nom de domaine **.bf** pour la visibilité de l'application depuis l'extérieure.	01	30 000	30 000
TOTAL				**3 183 700**

Tableau V-3.1: Récapitulation des besoins matériels et logiciels

Sources
Serveur : http://h41306.www4.hp.com, Switch et Cartes Réseaux : http://www.ldlc.com, Système d'exploitation : http://www.debian.org, Onduleur : http://hugotech.bf, Antivirus : http://fr.clamwin.com/, SGDBR : http://www.mysql.com, IDE : http://www.eclipse.org/, Serveur d'application : http://tomcat.apache.org/, Routeur CISCO : http://www.wstore.fr,
La Liaison Spécialisée : http://www.onatel.bf.

3. EVALUATION DU COUT DE REALISATION

a. Coût de développement

NB : Application des formules de calcul en mode organique de la méthode COCOMO simple.

Nombre de lignes de code : 5000 lignes.

Le calcul nous donne :

$HM = 2.4 (5000 / 1000)^{1.05} = 13$ Homme / Mois

$TDEV = 2.5 (13)^{0.38} = 6.62$ mois

Coût Total = 13 * 200.000 = **2 600 000** FCFA

b. Coût de la formation des utilisateurs

	Prix de l'horaire (FCFA)	**Nombre d'heures par utilisateur**	**Nombre d'utilisateurs**	**Montant (FCFA)**
Utilisateur CARPA	3000	10	4	120 000
Avocats	1000	5	150	750 000
Total				**870 000**

Tableau V-3-b.1 : Coût de la formation du système futur

c. Coût d'installation du réseau local

NB : Pour l'installation de ce réseau nous ferons appelle à un techniciens en réseaux informatiques (un Ingénieur de Travaux Informatique).

Désignation	Montant (FCFA)
Matériels d'interconnexion (câbles UTP RJ45, Connecteur RJ45, prises, coffret Switch, etc.)	210 000
Installation et configuration du réseau local	90 000
Configuration du des serveurs [20]	400 000
Total	**700 000**

Tableau V-3-b.2 : Coût d'installation du réseau local du système futur

d. Coût total de réalisation

Désignation	Prix (FCFA)
Coût Matériel et logiciel à acquérir	**3 183 700**
Coût de développement	2 600 000
Coût de formation	870 000
Coût d'installation du réseau local	700 000
Coût total	**7 353 700**

Tableau V-3-b.3 : Coût total de réalisation du système futur

[20] La configuration des serveurs comprend : la configuration des services web sécurisés (protocole https), de résolution de domaine, la configuration du serveur Firewall et la mise en place de la base de données.

VI. PROCEDURE TRANSITOIRE

La procédure transitoire est un ensemble de tâches consécutives à exécuter pour passer du système actuel au futur système. Ces tâches correspondent tout d'abord à la mise en place du nouveau système et son fonctionnement parallèle avec l'existant pendant une période déterminée, afin de déceler d'éventuelles erreurs et de s'assurer de son efficacité en terme des services rendus.

Pour permettre la continuité des services de la CARPA-BF lors de l'installation du nouveau système, nous préconisons un isolement du système actuel GESTCARPA qui fonctionne en monoposte et ne nécessitant pas un réseau.

Par la suite après l'installation du réseau local et de sa connexion à Internet (ligne spécialisée) pour permettre l'ouverture du système d'information, nous procéderons à la configuration des serveurs.

Enfin nous préconisons une installation des deux systèmes (ancien et nouveau) sur le nouveau serveur pour favoriser leur fonctionnement en parallèle pour une période de six (06) mois au cours de laquelle il sera demandé aux utilisateurs de faire des comptes-rendus hebdomadaires sur le comportement du nouveau système. Ces comptes-rendus permettront au groupe de projet de suivre l'application afin de pallier les éventuelles erreurs.

La fin de la procédure transitoire est prévue au terme des six (06) mois de fonctionnement en parallèle des deux systèmes (ancien et nouveau) avec une satisfaction à 90% par le groupe d'utilisateur.

VII. POLITIQUE DE SECURITE

1. SECURISATION DES CONNEXIONS DISTANTES AU SERVEUR

Les activités de la CARPA-BF se résumant essentiellement à la sécurisation des fonds des tiers transitant entre les mains des avocats sont gérées à 90% par un système informatique. Par conséquent, rendre accessible ce système sur Internet pose un problème de sécurité des données échangées à travers ce réseau. Il est donc important pour nous de prévoir une politique de sécurité vis-à-vis des transactions effectuées par les utilisateurs depuis l'extérieur. Cette sécurisation des transferts commence tout d'abord par la sécurisation des connexions distantes au serveur hébergeant le nouveau système.

Pour cela, nous proposons lors de la configuration du serveur, d'intégrer le moteur de Servlet Apache-Tomcat et le serveur web Apache-HTTP afin de protéger les transferts de données par une connexion sécurisée : HTTPS.

HTTPS[21] (avec S pour secured, soit « sécurisé ») est la variante de HTTP basée sur les protocoles SSL[22]. Il permet au visiteur de vérifier l'identité du site auquel il accède grâce à un certificat d'authentification. Il permet également de chiffrer la communication. Il est utilisé pour les transactions sécurisées sur Internet. Ce protocole est inclus dans pratiquement tous les navigateurs.

Cela assurera trois choses :

- **Confidentialité** des données: Il est impossible d'espionner les informations échangées.
- **Intégrité** des données: Il est impossible de truquer les informations échangées.
- **Authentification**: Il permet de s'assurer de l'identité du programme, de la personne avec laquelle le système communique.

[21] HTTPS : HTTP+SSL, web sécurisé.
[22] SSL (Secure Socket Layer) est un complément à TCP/IP et permet (potentiellement) de sécuriser n'importe quel protocole ou programme utilisant TCP/IP.

2. PROTECTION CONTRE LES VIRUS INFORMATIQUES

L'une des plus grandes sources de danger d'un système informatique et des données qu'il renferme sont les virus informatiques. Ces petits programmes informatiques s'auto-reproduisent et se répandent sur le réseau à travers les supports amovibles comme les clés USB, les CDROM etc. Cela peut avoir pour cause la déstabilisation du système et la destruction des données. Avec la connexion Internet ce risque d'infection va considérablement augmenter.

En vue d'éviter tous mauvais désagrément et de renforcer les mesures de sécurité nous préconisons d'installer un anti-virus sur tous les postes de travail du réseau local de la CARPA et sur les serveurs.

3. POLITIQUE DE SAUVEGARDE DES DONNEES

La politique de sauvegarde est la stratégie mise en place pour sécuriser les données contenues dans un système informatique. Cette mise en sécurité des données est une précaution pour prévenir l'intégrité du système en cas de perte de données.

La CARPA-BF, de par ses activités, peut être considérée comme une « *institution financière* », pour cela nous suggérons de réaliser la sauvegarde des données issues des transactions ou opérations effectuées par les utilisateurs du système sur un support physique (CD, disque magnétique, etc.).

Concernant le mode de sauvegarde nous préconisons une sauvegarde incrémentale tous les soirs et une sauvegarde complète toutes les fins de semaine de la base de données du système. La mise en oeuvre de ces deux méthodes de sauvegarde permettra de garantir pleinement l'intégrité des données.

VIII. PROCEDURES DE SECOURS

1. PANNE D'ELECTRICITE

Les serveurs d'application, de Firewall et la borne Internet sont munis d'un onduleur afin de prévenir des coupures brusques ou pannes d'électricité. Cette mesure nous permettra de garantir une disponibilité temporaire du serveur durant le temps d'autonomie des onduleurs.

Une coupure d'électricité à la CARPA-BF entraînera une indisponibilité partielle du système car les utilisateurs internes ne pourront pas interagir directement avec le système, mais grâce aux onduleurs, les utilisateurs externes pourront utiliser le système durant le temps d'autonomie. Les utilisateurs internes (Secrétaire, Gestionnaire, etc.) effectueront leurs tâches en mode manuelle avant le rétablissement de l'électricité. Ce fonctionnement en mode manuel consistera uniquement à réceptionner les bordereaux de création et autres traitements ne nécessitant pas directement le système SYSCARPA.

2. INDISPONIBILITE GENERALE DU SYSTEME

a. Indisponibilité due à une panne du serveur

Une panne du serveur d'application est synonyme d'indisponibilité totale du système informatique de la CARPA-BF. Dans ce cas nous préconisons dont un fonctionnement en mode manuel jusqu'à ce que le problème soit résolu.

b. Indisponibilité due à une panne du serveur Firewall

Une panne du serveur Firewall, qui est le cordon ombilical du réseau de la CARPA-BF, entraîne automatiquement une fermeture des accès de l'intérieur comme de l'extérieur au système, donc une indisponibilité de ce dernier. Les mesures à prendre sont identiques à celles du cas précédent.

3. PANNE DE LA BORNE INTERNET

Les utilisateurs externes (les avocats) ont accès au système grâce à la borne Internet. Par conséquent une panne de cette borne Internet a pour conséquence l'inaccessibilité du système en externe. Les avocats pourront donc avoir accès aux services de la CARPA-BF en utilisant le réseau local de la CARPA-BF.

4. PANNE DU SWITCH DU RESEAU LOCAL

Le Switch du réseau local permet aux utilisateurs internes d'avoir accès à Internet et au serveur d'application afin d'effectuer leurs traitements. Par conséquent une panne de ce dernier isole ces utilisateurs.

Nous préconisons donc un fonctionnement en mode manuel pendant le temps que durera la panne.

CONCLUSION

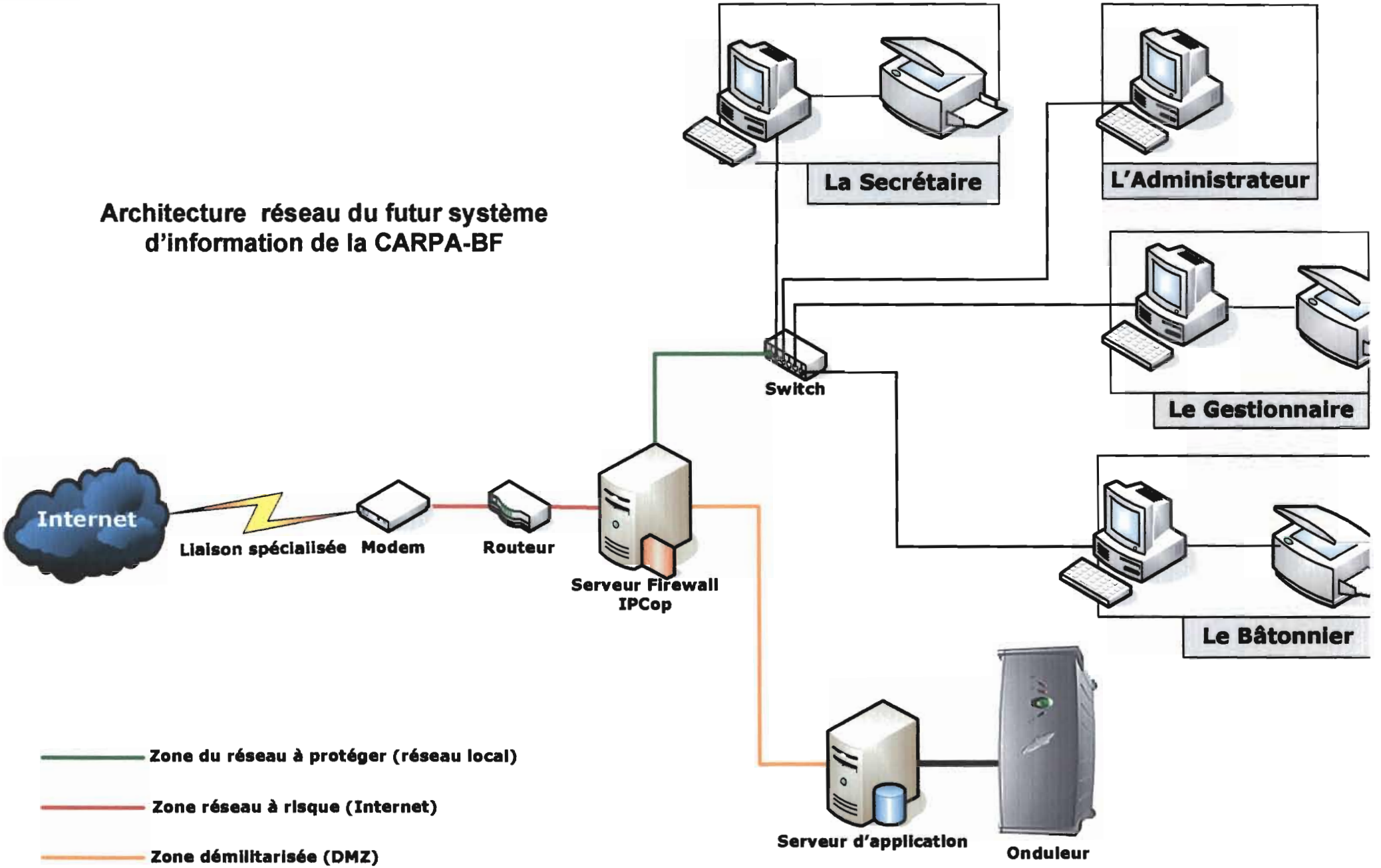
L'étude qui nous a été demandée à ZCP Informatique, durant ces trois mois, sur la "migration vers une architecture 3-tier (modèle MVC) de l'application de gestion des règlements pécuniaires des Avocats du Burkina (GESTCARPA)" a été réalisée en parfaite harmonie avec le groupe de pilotage. Ce dernier rapport marque aussi la fin de cette étude, dont l'objectif aura été l'analyse et la conception d'un nouveau système d'information pour la CARPA-BF.

La conception détaillée du système d'information nous a permis de concevoir en détail les nouvelles orientations du système d'information, elle constitue ainsi une base solide pour sa réalisation dans les phases suivantes du processus unifié 2TUP qui nous sert de feuille de route.

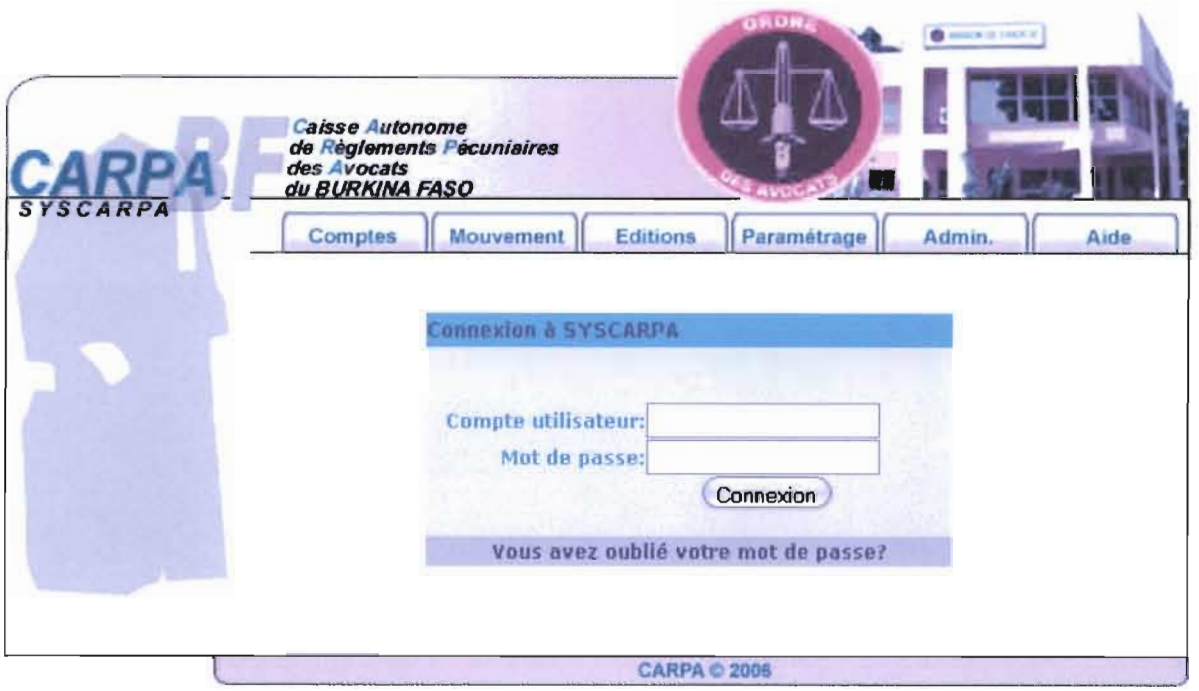
Les diagrammes UML produits, les classes détaillées présentées, les différentes vues explicitées, etc. serviront de briques pour la réalisation de la future application entreprise de gestion des règlement pécuniaires des avocats du Burkina et respectant les spécifications de la plate-forme J2EE.

ANNEXES

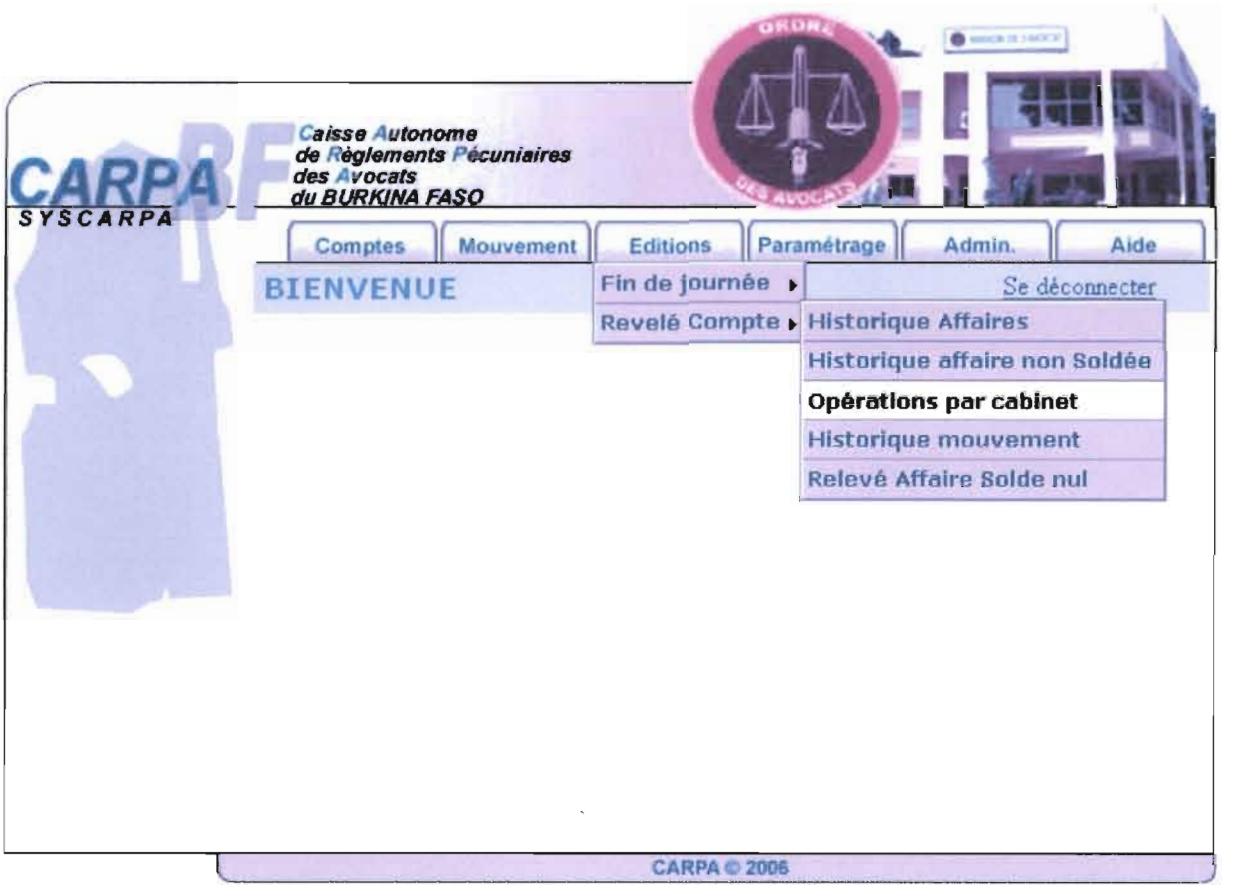
Architecture réseau du futur système d'information de la CARPA-BF



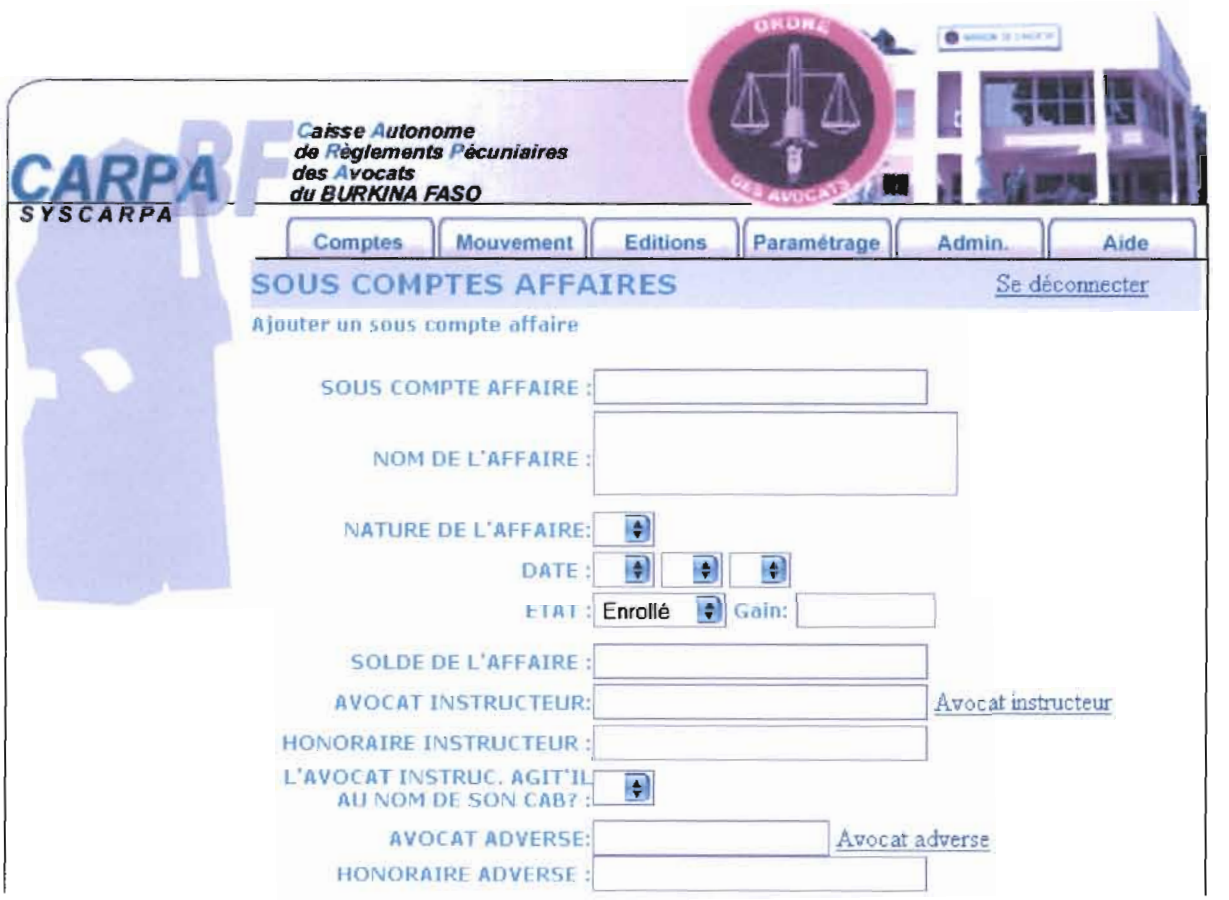
QUELQUES MAQUETTE DU FUTUR SYSTEME



Maquette 1 : la vue d'authentification : **PgLogin**



Maquette 3 : la vue principale du système : **MainPage**



Maquette 3: la vue d'ajout d'un sous compte affaire : **PgAjoutAffaire**

Maquette 4: la vue d'ajout d'un sous compte Avocat : **PgAjoutAvocat**

Maquette 5: la vue d'enregistrement d'un mouvement : **PgAjoutMvt**