BURKINA FASO
UNITE-PROGRES-JUSTICE

MINISTERE DES ENSEIGNEMENTS SECONDAIRE ET SUPERIEUR

UNIVERSITE POLYTECHNIQUE DE BOBO-DIOULASSO

ECOLE SUPERIEURE D'INFORMATIQUE

E.S.I

MEMOIRE DE FIN DE CYCLE

en vue de l'obtention du

DIPLOME D'INGENIEUR DE CONCEPTION EN INFORMATIQUE

THEME :

MISE EN PLACE D'UN SYSTEME REPARTI DE GESTION BUDGETAIRE ET COMPTABLE DU CENTRE MURAZ

Présenté par :

TRAORE Satêhin Michel-Romuald

Maître de stage : Mme Joëlle OUATTARA/COMPAORE

Directeur de mémoire : Dr Loé SANOU

JUILLET 2010

N°: .....-2010/CICI3

TABLE DES MATIERES

Page

DEDICACE

A mon frère, TRAORE Sy Emmanuel, décédé le 30 Mai 2010 (paix à son âme !).

REMERCIEMENTS

- A l'***Ecole Supérieure d'Informatique*** (ESI), pour ces deux (02) ans de formation.
- A ***M. Ibrahim BALLO***, ***Mme Joëlle OUATTARA/COMPAORE*** et ***Dr Loé SANOU***, pour leur encadrement et leur disponibilité durant le stage.
- Au ***Directeur Général du Centre MURAZ***, pour le stage qu'il nous a octroyé.
- A tous les ***agents de la DAF***, ***du Contrôle financier*** et ***de l'Agence comptable***, pour leur disponibilité.
- Au reste du ***personnel du Centre MURAZ***, pour son accueil.
- A toute ***ma famille***, pour leur soutien inlassable.

LISTE DES ABREVIATIONS

Sigle	Signification
AC	Agence Comptable
API	Application Programming Interface
CF	Contrôle Financier
CORBA	Common Object Request Broker Architecture
DAF	Direction de l'Administration et des Finances
DCOM	Distributed Component Object Model
JDBC	Java DataBase Connectivity
JPA	Java Persistence API
LS	Liaison Spécialisée
OMG	Object Management Group
RMI	Remote Methode Invocation
SGBCoM	Système de Gestion Budgétaire et Comptable de MURAZ
UML	Unified Modeling Language
WiFi	Wireless Fidelity

TABLE DES FIGURES

LISTE DES TABLEAUX

RESUME

Ce document fait le point du projet d'informatisation du système d'information (SI) budgétaire et comptable du Centre MURAZ. Ce dernier a vu la mise en œuvre du processus de développement, dénommé *cycle en V*.

A travers la première phase de ce processus — l'analyse des besoins —, le SI budgétaire et comptable actuel a été évalué. Cette évaluation a relevé deux logiciels qui gèrent séparément la comptabilité et le budget. Aussi, elle a abouti à la proposition d'un système réparti qui prend en compte les points forts et les points de dysfonctionnements de l'existant. Ce système répond, également, aux objectifs du projet tels que la réutilisation, l'extensibilité et l'échange des données entre les services à travers un unique système informatique.

Un système réparti est un ensemble de processus communiquant et répartis sur un réseau d'ordinateurs, et est déployé à l'aide d'un middleware. Le système que nous proposons est dénommé SGBCoM (Système de Gestion Budgétaire et Comptable de MURAZ) et est développé a l'aide du middleware CORBA. Ce dernier est l'un des middlewares, de notre point de vue, les plus matures de nos jours et offre des fonctionnalités qui permettent de découper une application en couches. Ces couches obtenues sont indépendantes, extensibles et réutilisables.

CORBA met en œuvre une architecture client-serveur où il est possible de distinguer des objets distants qui offrent des services à d'autres objets qualifiés de clients. Pour spécifier l'ensemble des services (ou interface) d'un objet distant, la norme CORBA offre un langage neutre — IDL. Cette spécification est implantée par un langage de programmation. Parmi les langages de programmation prévus pur cela, Java a été retenu. La programmation avec ce dernier a vu la mise en œuvre des technologies telles que JPA et a permis de réaliser l'application à 50%.

INTRODUCTION GENERALE

Le recours à un système de traitement automatique de l'information par les entreprises est devenu, depuis quelques années, une solution incontournable pour ces dernières. Automatiser les traitements peut aider à gagner en rapidité, fiabilité et traçabilité. L'informatisation d'une structure peut donc conduire, si elle est bien menée, à une meilleure productivité.

C'est ainsi que le Centre MURAZ a lancé un projet d'informatisation budgétaire et comptable. Ce projet a pour thème « ***Mise en place d'un système réparti de gestion budgétaire et comptable*** ». Un système réparti est un ensemble de processus répartis communiquant à travers un réseau de machines le plus souvent hétérogènes, et coopérant à la résolution d'un problème commun. Ce type d'application est développé et déployé à l'aide d'une couche logiciel, dénommée *middleware*, qui est l'intermédiaire entre l'application et le système d'exploitation. Une application répartie peut être facilement extensible et réutilisable si elle est mise au point à l'aide d'un middleware de qualité.

PROBLEMATIQUE

Il est à noter que le Centre MURAZ a déjà un logiciel de gestion budgétaire et un autre pour la comptabilité. Pourquoi donc mener un projet d'informatisation de la gestion du budget et de la comptabilité ?

Selon les utilisateurs, le logiciel budgétaire a permis de gagner largement en rapidité par rapport au système précédent qui était manuel. Cependant, ils observent certaines insuffisances possibles du logiciel : lenteur dans l'exécution de certaines tâches et impossibilité d'accès en écriture multiutilisateurs.

Par ailleurs, l'échange de données entre le *Service de suivi budgétaire* et *l'Agence comptable* n'est pas automatisé. L'Agence comptable partage les informations sur la trésorerie du Centre MURAZ en les imprimant sur papier. Les deux logiciels ne sont pas en liaison pour le moment.

OBJECTIFS ET RESULTATS ATTENDUS

Le projet d'informatisation a donc pour but de fournir un système informatique efficace de gestion budgétaire et comptable. Cette informatisation doit prendre en compte les atouts et les insuffisances de la solution existante, et les préoccupations des utilisateurs (agents) concernés. De ce fait, elle doit :

- offrir une application qui tourne selon l'architecture client/serveur ;
- permettre à plusieurs utilisateurs d'accéder à la base de données simultanément ;
- prévoir l'exploitation des données par d'autres applications ;
- prendre en compte l'existence de plusieurs budgets (global et par projet).

ORGANISATION DU DOCUMENT

Pour atteindre les objectifs ci-dessus, une étude a été menée auprès des utilisateurs afin de dégager une solution informatique. Ce document, qui constitue notre mémoire de fin de cycle, fait le compte rendu de cette étude et des résultats obtenus. Il est organisé en cinq (05) chapitres.

Le premier — *Contexte de stage* — présente la structure d'accueil, la méthode de développement, les acteurs du projet et les moyens mis à notre disposition.

Quant aux quatre (04) autres chapitres, ils concernent les phases de la méthode de développement adoptée. Ces chapitres, qui font donc le point de l'étude technique, sont :

- L'*analyse des besoins* qui relate l'étude du système d'information (SI) actuel et les différentes propositions faites pour son amélioration. Ce chapitre présente la solution retenue pour l'implantation du nouveau SI.

- La *spécification* qui étudie les différentes fonctionnalités que le SI futur pourrait avoir.

- La *conception* regroupe les deux phases de la méthode de développement : la *conception architecturale* et la *conception détaillée*. Ce chapitre est donc subdivisé en deux sous-parties. La première spécifie les différentes couches du système futur ; la seconde montre l'étude détaillée de ces couches.

- La *réalisation* fait le point sur notre processus de développement et le déploiement de l'application.

CHAPITRE 1 :

CONTEXTE DU STAGE

INTRODUCTION

Pour mener à bien un projet, il faut une bonne organisation du travail et situer clairement le rôle des différents acteurs dudit projet. Le premier chapitre présente d'abord le Centre MURAZ, avant d'aborder la méthode de développement. A partir de cette dernière, un planning est établi. Enfin, aux derniers points de ce chapitre, les acteurs du projet et les moyens mis à notre disposition sont présentés.

I- PRESENTATION DU CENTRE MURAZ

Le Centre MURAZ est un centre de recherche biomédicale situé à Bobo-Dioulasso (Burkina Faso). Il jouit du statut d'établissement public de santé (EPS) par décret N° 2006-448/PRES/PM/MS/MFB du 14 septembre 2006. Le Centre MURAZ est sous la tutelle technique du Ministère de la Santé, et sous la tutelle financière du Ministère de l'Economie et des Finances.

I.1- Missions

Trois missions essentielles incombent au Centre MURAZ actuellement :

- **La recherche** : elle est organisée autour de huit axes qui sont :
  - le développement de techniques alternatives de diagnostic et d'évaluation des traitements ;
  - les essais thérapeutiques ;
  - la résistance aux médicaments ;
  - la santé maternelle et infantile ;
  - la socio anthropologie, économie de la santé et systèmes de santé ;
  - le développement de stratégies de lutte contre les maladies ;
  - l'entomologie ;
  - la recherche fondamentale.

- **La formation** : elle consiste en une participation effective à la formation du personnel de santé à travers des stages pour paramédicaux, doctorants (mémoires, thèses), post-doctorants.

- **L'expertise** : elle consiste en un renforcement des capacités des compétences nationales (en priorité) de sorte à disposer sur le plan technique d'experts de haut niveau.

I.2- Organisation

Le Centre MURAZ est organisé en Unités de Recherches (UR) et en Unités Techniques (UT), en départements d'appui à la recherche et en départements administratifs, tous directement rattachés à la direction générale. La *figure 1* présente l'organigramme du Centre MURAZ :

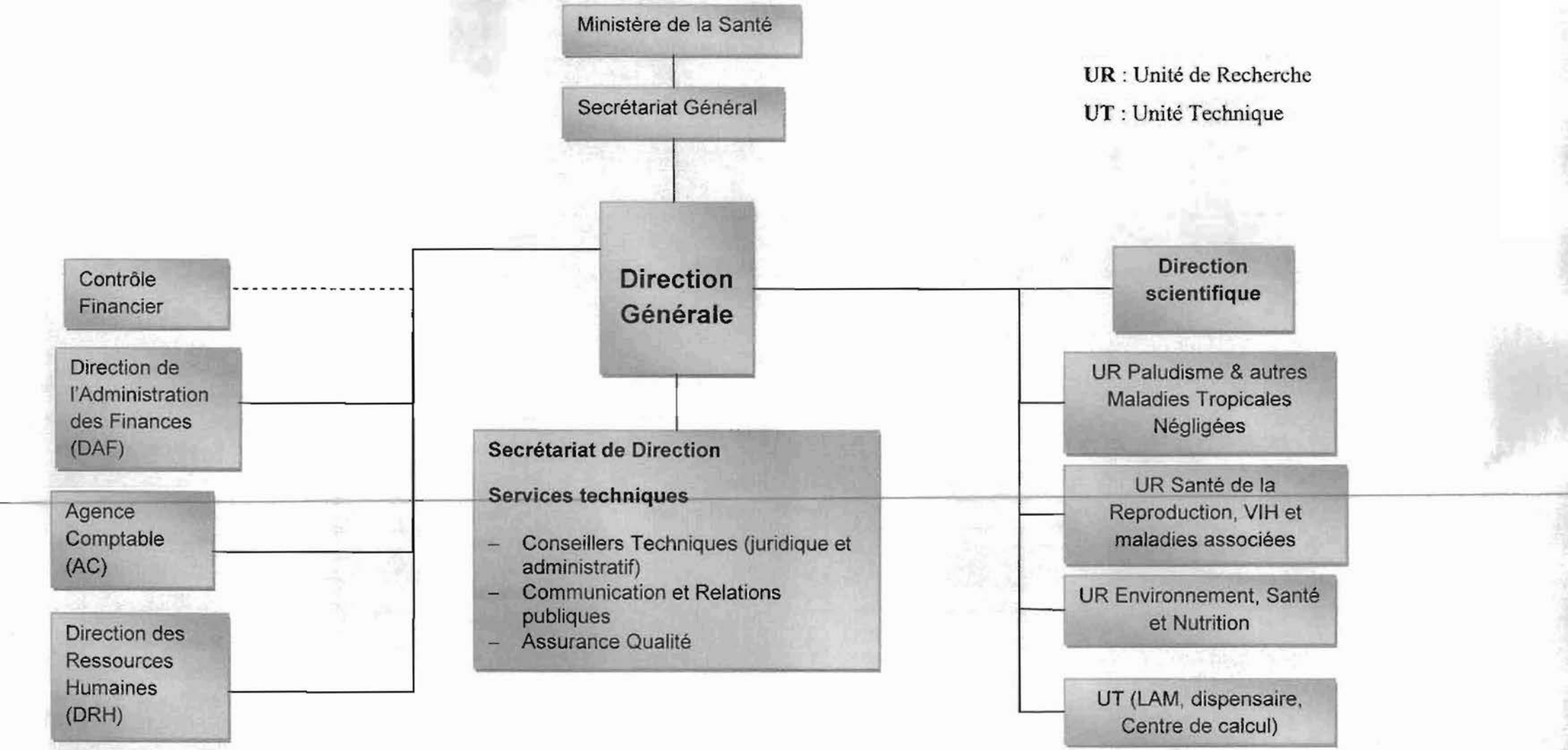


Figure 1 : Organigramme du Centre MURAZ.

II- METHODE DE DEVELOPPEMENT

Une méthode de développement peut être vue comme étant la fusion d'une démarche (phases de développement) et d'un ensemble de formalisme (langage de modélisation). Dans cette partie, nous présentons la démarche (ou cycle) de développement et le langage de modélisation.

II.1- Cycle de développement

Le cycle de développement désigne toutes les étapes de construction d'un logiciel, de sa conception à sa disparition. L'objectif d'un tel découpage est de permettre de définir des jalons intermédiaires qui permettent la validation du développement logiciel et la vérification du processus de développement.

Il existe plusieurs cycles de développement parmi lesquels on distingue le *cycle V*. Ce dernier part du principe que les procédures de vérification de la conformité du logiciel aux spécifications doivent être élaborées dès les phases de conception. Ceci permet à chaque étape de définir non seulement les fonctions, mais également les critères de validation. La cohérence entre les deux éléments permet de vérifier en continu que le projet progresse vers un produit répondant aux besoins initiaux.

Le modèle du *cycle en V* est adapté aux projets de taille et de complexité moyenne. Il permet d'identifier et d'anticiper très tôt les éventuelles évolutions des besoins. C'est un modèle avantageux pour une maîtrise d'œuvre, rassurant pour une maîtrise d'ouvrage qui, cependant, doit s'engager significativement.

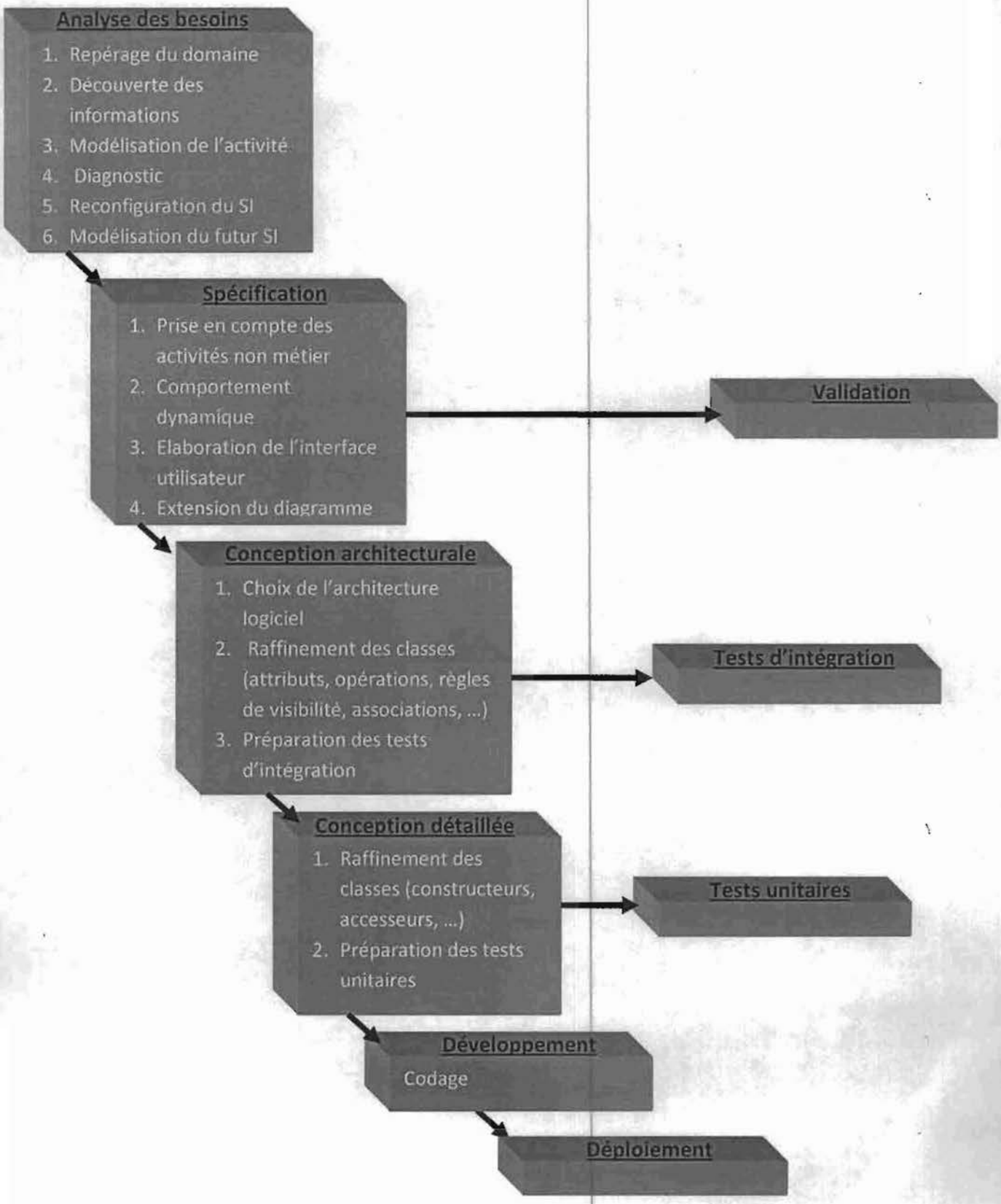


Figure 2 : Cycle en V détaillant les différentes phases.

Comme le montre la figure précédente, le Cycle en V comprend les phases suivantes :

- **Analyse des besoins** : il consiste au recueil et la formalisation des besoins de l'entreprise et de l'ensemble des contraintes.

- **Spécification** : il s'agit de comprendre le ou les problèmes que le futur système logiciel devrait résoudre, et de fournir une base pour répondre aux questions concernant les propriétés spécifiques du problème et du système.

- **Conception architecturale** : il s'agit de l'élaboration des spécifications de l'architecture générale du logiciel. Cette dernière décrit l'organisation générale du produit informatique, sa subdivision en modules et en couches.

- **Conception détaillée** : il consiste à définir précisément chaque sous-ensemble décrit précédemment.

- **Développement** : il concerne la traduction dans un langage de programmation des fonctionnalités définies lors des phases de conception.

- **Tests unitaires** : il permet de vérifier individuellement que chaque sous-ensemble du logiciel est implémenté conformément aux spécifications.

- **Intégration** : l'objectif est de s'assurer de l'interfaçage des différents éléments (modules) du logiciel. Elle fait l'objet de tests d'intégration consignés dans un document.

- **Validation** : il concerne la vérification de la conformité du logiciel aux spécifications initiales.

- **Déploiement** : c'est la mise en place de l'application à travers les différentes machines.

Pour mettre en œuvre les étapes d'un cycle de vie, un langage de modélisation est utilisé pour décrire le système d'information.

II.2- Langage de modélisation

Il existe plusieurs langages de modélisation parmi lesquels on peut citer UML — *Unified Modeling Language*. Ce dernier est un langage standard de modélisation objet conçu pour l'écriture de plan d'élaboration de logiciel. Il fusionne les concepts issus de trois méthodes objets de référence : OMT, BOOCH et OOSE.

C'est un langage élaboré pour visualiser, construire et documenter les artefacts d'un système logiciel. Il utilise treize (13) diagrammes pour cela. Avec ce langage, aucune démarche n'est imposée pour l'analyse du système d'information.

De plus, il conserve tous les atouts du paradigme objet à savoir :

- la stabilité de la modélisation par rapport aux entités du monde réel ;
- la réutilisabilité des objets dans différents modules ;
- l'allègement des tâches de maintenance.

Dans le cadre d'un projet tous les diagrammes ne sont pas nécessaires. Pour ce projet, seulement les diagrammes cités ci-dessous sont employés.

- ***Diagramme de cas d'utilisation***

Le diagramme de cas d'utilisation représente la structure des fonctionnalités nécessaires pour les utilisateurs du système.

Un cas d'utilisation (use case) représente *un ensemble de séquences d'actions* réalisées par le système et produisant un résultat observable intéressant pour un acteur particulier. Chaque cas d'utilisation correspond à une fonction métier du système, selon le point de vue d'un de ses acteurs. Un cas d'utilisation est représenté graphiquement par une ellipse.

Un acteur, au sens UML, représente le rôle d'une entité externe (utilisateur humain ou non) interagissant avec le système.

- ***Diagramme de séquence***

Le diagramme de séquence permet de représenter les collaborations entre acteurs et/ou objets selon un point de vue temporel. Le temps s'écoule du haut vers le bas. La collaboration entre objet se fait par l'envoi de message qui est matérialisé par une flèche. Ce diagramme sert à illustrer les scénarios des cas d'utilisation.

- ***Diagramme d'activité***

UML permet de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation, à l'aide de diagrammes d'activités. Une *activité* représente une exécution d'un mécanisme, un déroulement d'instructions séquentielles. Le passage d'une activité vers une autre est matérialisé par une *transition*. Chaque transition est déclenchée par la fin d'une activité et provoque le début immédiat d'une autre activité.

- ***Diagramme de classe***

La modélisation objet consiste à créer une représentation abstraite, sous forme d'*objets*, d'entités ayant une existence matérielle (arbre, personne, téléphone, ...) ou bien virtuelle (sécurité sociale, compte bancaire, ...). Un objet est donc une représentation abstraite des caractéristiques d'une entité par les *attributs* et de son comportement par les *méthodes*.

Une collection d'objets semblables constitue une *classe*. Cette dernière agit comme un modèle décrivant le comportement d'ensembles d'objets semblables.

Une classe se représente avec UML sous forme d'un rectangle divisé en trois sections : le premier contient le nom donné à la classe ; le deuxième les attributs ; et le dernier les opérations.

- ***Diagramme d'états-transitions***

Un *diagramme d'états-transitions* représente graphiquement le comportement d'un classificateur (composant ou classe). Il montre les changements d'état du classificateur suite à des événements qui permettent la transition d'un état à l'autre.

III- ACTEURS DU PROJET ET MOYENS MIS A DISPOSITION

Un acteur est une personne qui fournit des informations sur le SI ou participe à la réalisation des grandes lignes du projet. Ceux de notre projet sont :

- **Le groupe de pilotage** qui a essentiellement pour rôle de prendre les décisions relatives aux objectifs visés. Il fixe les orientations générales et détermine les moyens à mettre en place pour la réalisation du projet. Il est composé de :
  - M. Ibrahim BALLO, informaticien au centre MURAZ ;
  - Mme Jöelle OUATTARA/COMPAORE, informaticien au centre MURAZ ;
  - Dr. Loé SANOU, enseignant chercheur à l'ESI.
- **Le groupe de projet** étant chargé de mener l'étude du projet de la phase d'analyse des besoins jusqu'à la réalisation. Il est composé d'une seule personne : M. Satêhin Michel-Romuald TRAORE (étudiant en troisième année de CICI).
- **Le groupe des utilisateurs** regroupant l'ensemble des agents de la DAF, du Contrôle financier et de l'Agence comptable. Ces agents ont fourni toutes les informations nécessaires pour comprendre leur système d'information.

Pour la réussite du projet, nous avions à notre disposition un PC portable connecté permanemment à l'Internet pour les recherches documentaires. Une imprimante était aussi à notre disposition.

CONCLUSION

Le présent chapitre a permis de faire un tour d'horizon de la structure d'accueil, de présenter la méthode de développement et de spécifier le rôle des acteurs du projet. La suite du document est un compte rendu du processus de développement mis en œuvre. Elle commence par la phase d'*analyse des besoins*.

CHAPITRE 2 :

ANALYSE DES BESOINS

INTRODUCTION

L'*analyse des besoins* constitue la première phase de notre processus de développement. Elle permet de comprendre l'existant et de poser les jalons du futur système.

Cette phase commence par définir les limites du système d'information actuel. Après une analyse de ce dernier est faite à travers l'*étude du système informatique* et la *modélisation du WorkFlow*. Cette analyse permet de dresser un diagnostic du système. En réponse aux points faibles relevés par ce diagnostic, des solutions sont proposées.

I- REPERAGE DU DOMAINE

Cette activité permet de délimiter le domaine d'étude. Après des échanges avec les acteurs du projet, nous avons pu déterminer les contours du système d'information de gestion budgétaire et comptable. Cette délimitation est représentée par le diagramme de contexte statique de la *figure 3*. Ce diagramme montre les interactions possibles entre les différents acteurs du système.

Rappelons qu'un acteur, au sens UML, représente le rôle d'une entité externe interagissant avec le système étudié. Un acteur principal est celui qui déclenche l'exécution d'un cas d'utilisation. Ceux du système d'information actuel sont :

- ***Projets*** : ils expriment leurs besoins et demandent la situation de leurs budgets à la DAF.
- ***Gestionnaires budgétaires*** : ils ont en charge le suivi du budget.
- ***Clients*** : ils demandent les services du Centre MURAZ.
- ***Fournisseurs*** : ils offrent des prestations de services et livrent les matériels.
- ***Agence comptable*** : elle encaisse les recettes du Centre MURAZ et paye ses factures.
- ***Contrôle financier*** : contrôle les engagements et les liquidations.
- ***Caissier*** : il encaisse les recettes au niveau des laboratoires et du dispensaire.

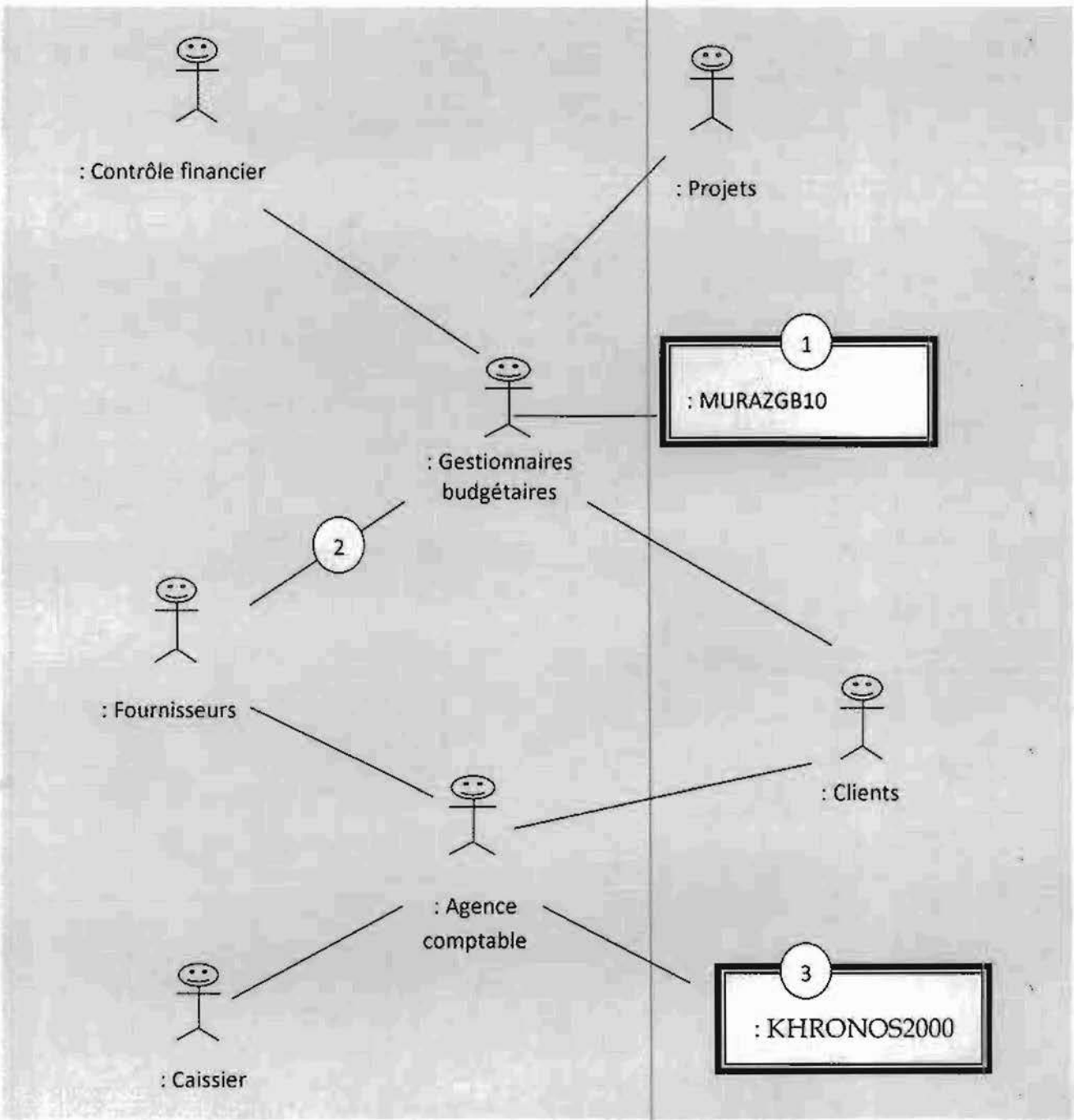


Figure 3 : Diagramme de contexte statique du système d'information.

(1) : logiciel de gestion budgétaire (voir paragraphe II.2.1 du présent chapitre).

(2) : échange d'informations entre les deux acteurs.

(3) : logiciel de gestion de la comptabilité (voir paragraphe II.2.2 du présent chapitre).

II- ANALYSE DU SYSTEME INFORMATIQUE ACTUEL

L'analyse du système informatique actuel s'inscrit dans l'activité *recueil des informations*. Cette dernière a pour objectif de comprendre les différentes facettes du problème. Elle est menée en parallèle avec l'activité précédente et permet de repérer les grands concepts d'informations du domaine d'étude.

II.1- Architecture des réseaux locaux

Actuellement, au Centre MURAZ, il n'y a pas un réseau unique connectant tous les ordinateurs des différents services. Les différents postes de la DAF partagent les ressources à l'aide d'un réseau local interne ; l'Agence Comptable et le Contrôle Financier qui sont logés dans le même bâtiment, leurs ordinateurs sont reliés par le même réseau.

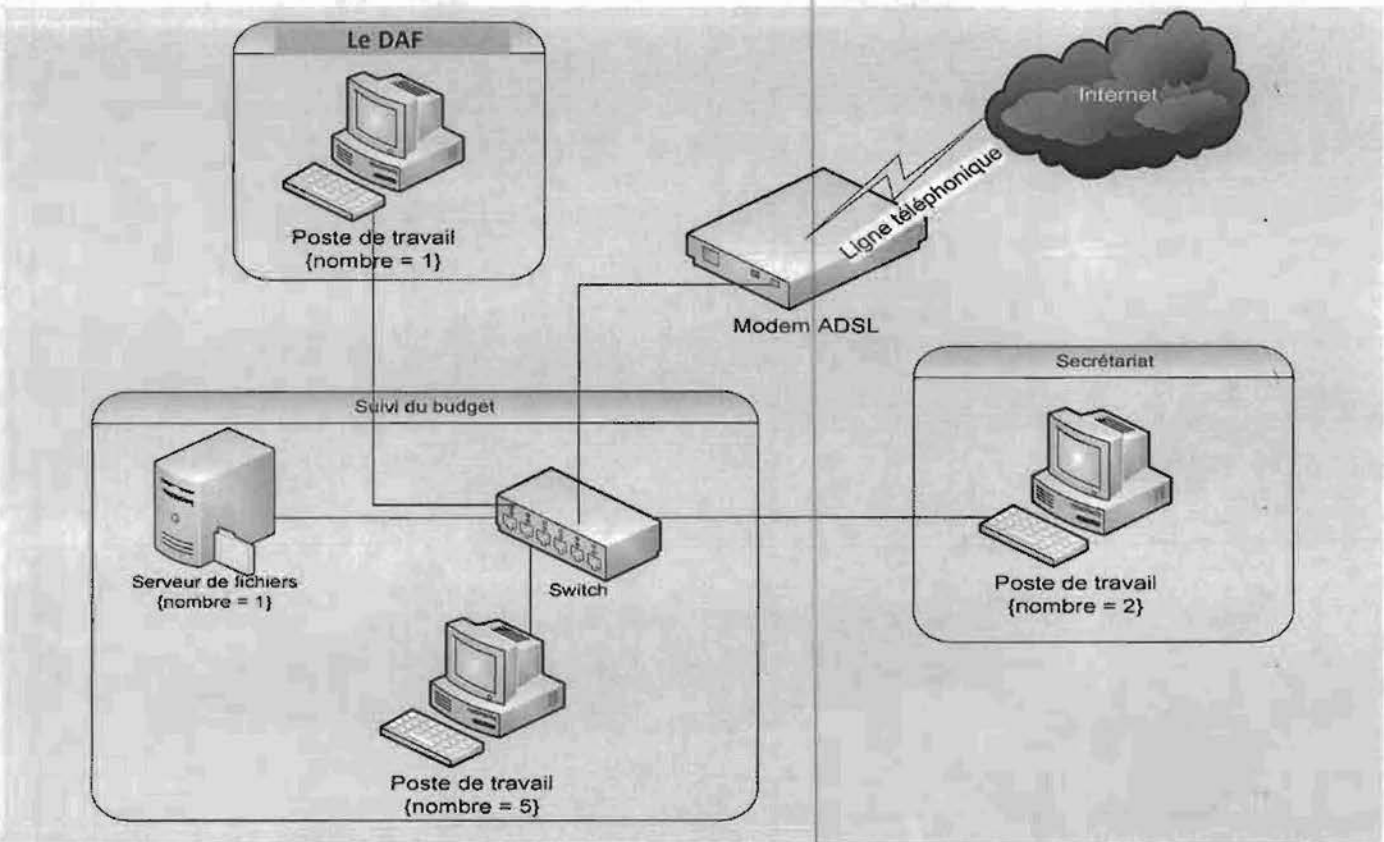


Figure 4 : Architecture réseau de la DAF.

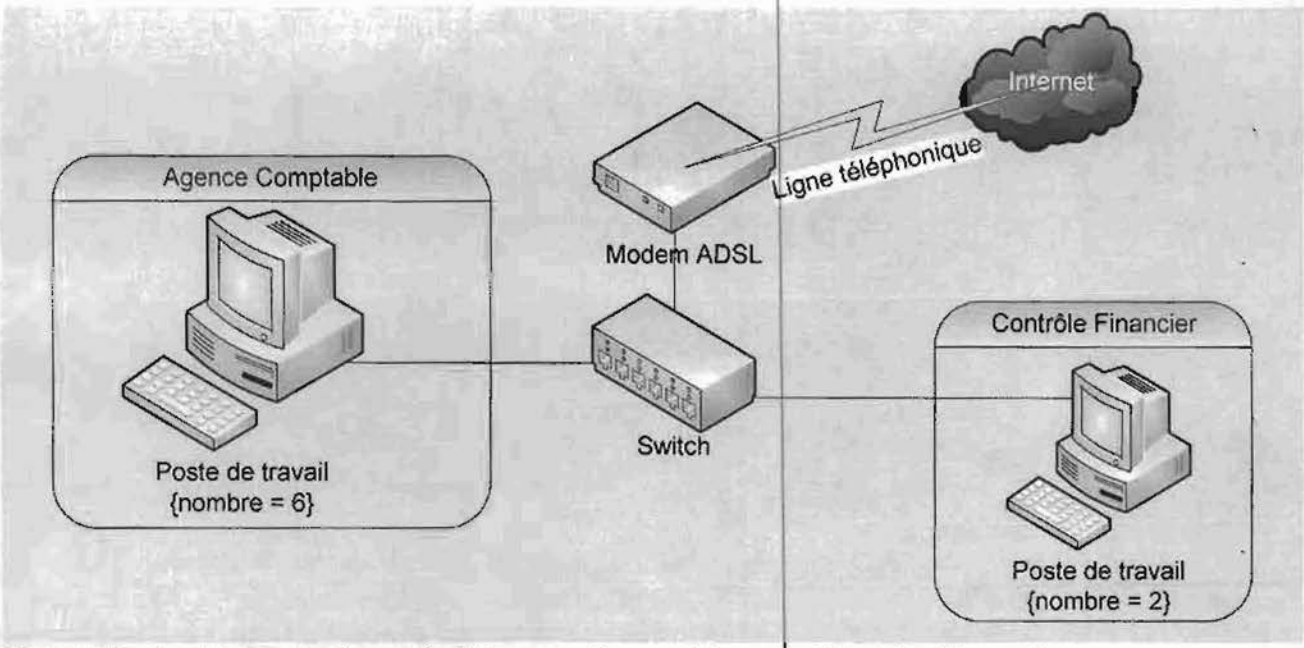


Figure 5 : Architecture réseau de l'Agence Comptable et du Contrôle Financier.

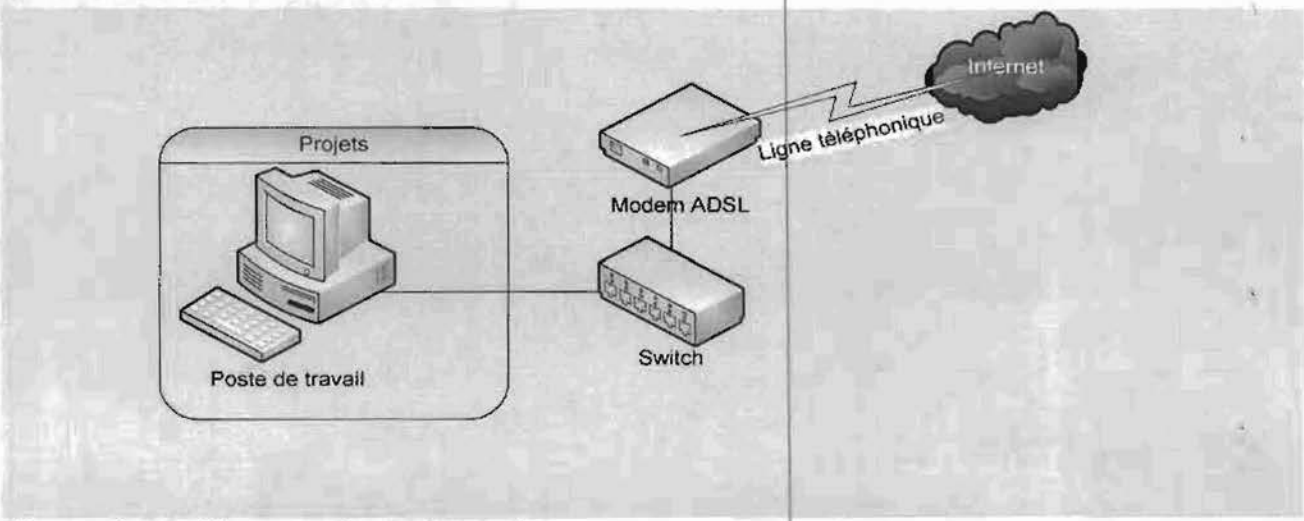


Figure 6 : Architecture réseau des projets.

II.2- Analyse des applications existantes

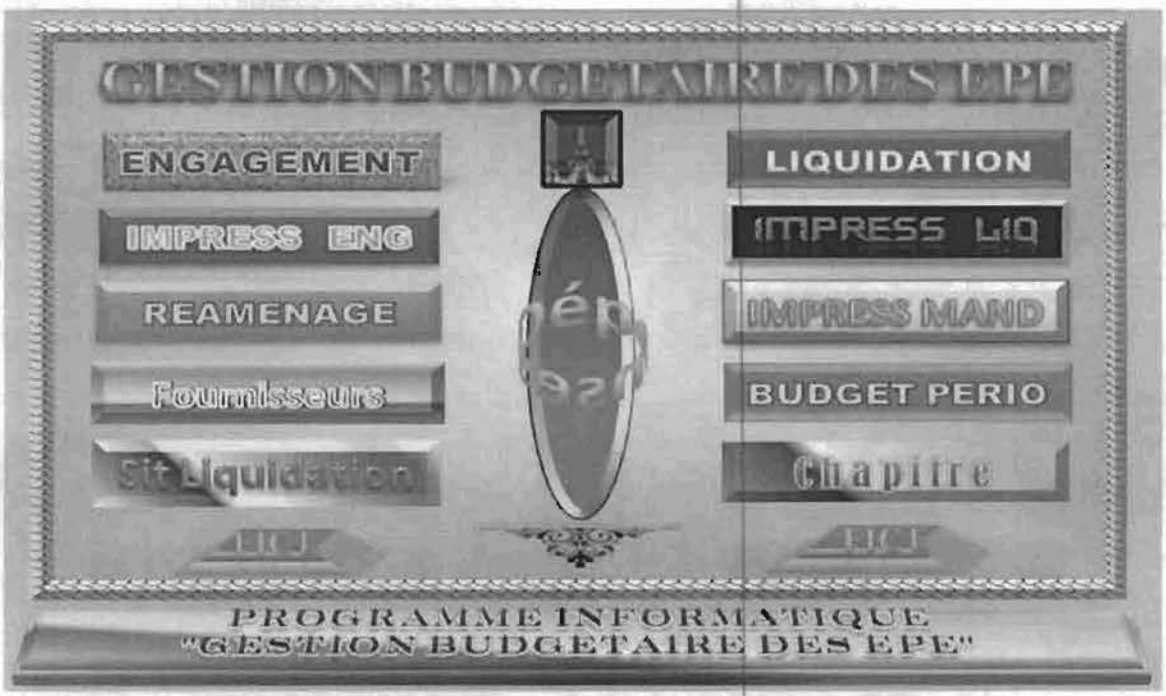
Dans le système d'information actuel, on distingue deux applications informatiques : MURAZGB10 et KRONOS2000.

II.2.1- MURAZGB10

Ce logiciel a été acquis en Mai 2010 afin d'améliorer le système de gestion budgétaire du Centre MURAZ. Ce logiciel a été conçu avec *Microsoft Excel 2007*. En fait, ce logiciel est un classeur de Microsoft Excel 2007 s'appuyant sur les puissances de ce dernier.

Microsoft Excel 2007 est un tableur qui permet de saisir les données et de les analyser à travers les graphismes. Ce tableur peut être vu sous deux angles : l'angle de l'utilisateur, qui se sert du logiciel pour accomplir certaines tâches afin d'obtenir un certain résultat ; l'angle de l'informaticien qui voit en Excel un véritable outil de développement.

- **Ecran d'accueil**



MURAZGB10 offre l'essentiel des fonctionnalités que peut avoir un système de gestion budgétaire. Il permet entre autres de :

- inscrire les différentes lignes budgétaires une fois le budget élaboré ;
- suivre la situation du budget global et celle des projets ;
- établir un engagement ;
- éditer une liquidation ;
- élaborer un mandat de paiement.

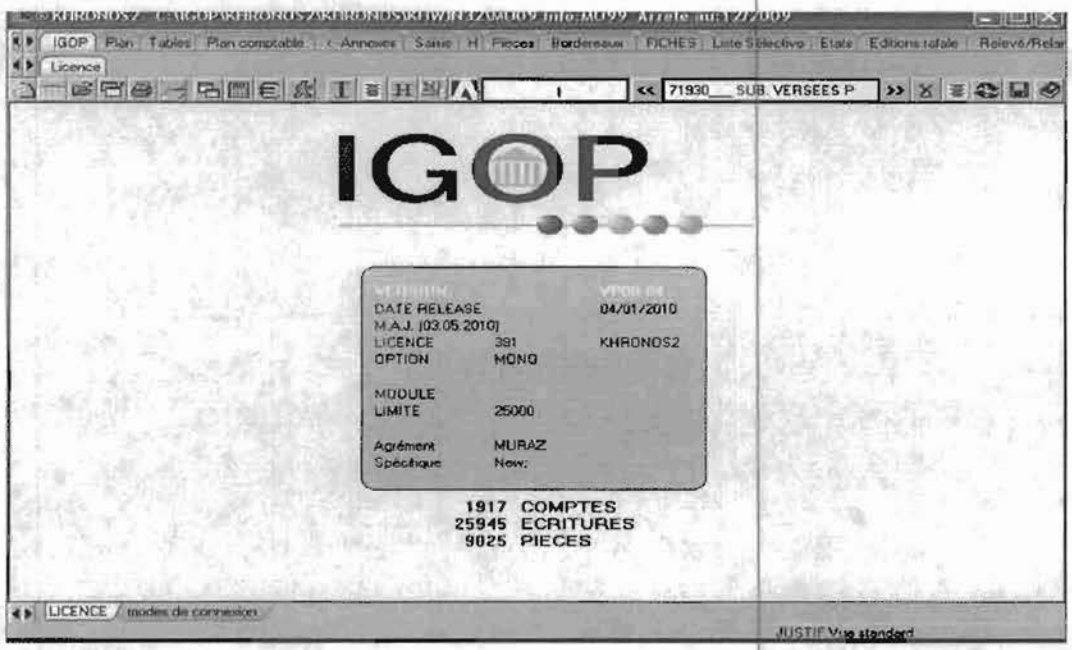
L'écran ci-dessous, qui constitue la page d'accueil, montre toutes les fonctionnalités du logiciel qui peuvent être accédées par simple click.

II.2.2- KHRONOS2000

Ce logiciel est un produit de la société IGOP. Cette dernière est un éditeur de logiciels de gestion et fournisseur de systèmes d'information pour les entreprises et les entrepreneurs. Elle a son siège social à Versailles/France.

KHRONOS est un progiciel de gestion comptable, analytique et budgétaire. Cependant, seulement le *module comptable* est exploité par le Centre MURAZ.

- **Ecran d'accueil**



Cet écran montre l'essentiel des fonctionnalités offertes par ce système. Elles sont organisées en onglets :

- L'onglet *Plan*, pour la création des codes des projets.
- L'onglet *Plan comptable*, pour gérer les différentes rubriques du plan comptable.
- L'onglet *Saisie*, pour enregistrer les dépenses et recettes.

III- MODELISATION WORKFLOW

On appelle *WorkFlow*, traduit littéralement en français *flux de travail*, la modélisation d'un processus métier. Un processus métier représente les interactions sous forme d'échange d'informations entre les acteurs du système.

Menée en parallèle avec les deux (02) activités précédentes, les trois activités permettent de comprendre l'existant. L'activité actuelle décrit davantage le rôle des différents acteurs du domaine, et la façon dont ils interagissent pour atteindre les finalités du domaine. Le diagramme des cas d'utilisation montre globalement ces finalités et leurs liens avec les acteurs.

III.1- Identification des cas d'utilisation

Pour rappel, un cas d'utilisation représente un ensemble de séquences d'actions réalisées par le système et produisant un résultat observable intéressant pour un acteur particulier.

Les interviews et l'analyse des applications ont permis de recenser les cas d'utilisation ci-dessous.

Faire une dépense (*Projets et services*)
Intention : concrétiser une dépense quelconque
Actions : exprimer besoins, engager dépense, liquidation, …

Engager une dépense (*Gestionnaires budgétaires*)
Intention : engager une dépense d'un projet ou de l'administration
Actions : créer un nouvel engagement, imprimer un engagement

Liquider une dépense (*Gestionnaires budgétaires*)
Intention : élaborer une liquidation correspondant à un engagement
Actions : créer une nouvelle liquidation, imprimer une liquidation

Faire un mandat (*Gestionnaires budgétaires*)
Intention : éditer un mandat correspondant à une liquidation
Actions : choisir une liquidation, imprimer un mandat

Demander situation budgétaire (*Projets*)
Intention : connaitre leurs situations budgétaires

Actions : contacter la DAF, interroger la base de données

Etablir une expression des besoins (*Projets*)

Intention : soumettre une expression des besoins à la DAF

Actions : créer une nouvelle expression des besoins

Gérer les dépenses (*Agence comptable*)

Intention : constater la charge et la régler

Actions : débiter et créditer des comptes

Gérer les recettes (*Agence comptable*)

Intention : constater la recette

Actions : débiter et créditer

Gérer les codes des projets (*Agence comptable, Gestionnaire budgétaire*)

Intention : créer et mettre à jour les code des projets

Actions : créer, supprimer et modifier les codes

Gérer le plan budgétaire (*Gestionnaire budgétaire*)

Intention : mettre en place le plan budgétaire

Actions : créer, supprimer et modifier les différentes rubriques du plan.

Gérer le plan comptable (*Agence comptable*)

Intention : mettre en place le plan comptable

Actions : créer, supprimer et modifier les différentes rubriques du plan.

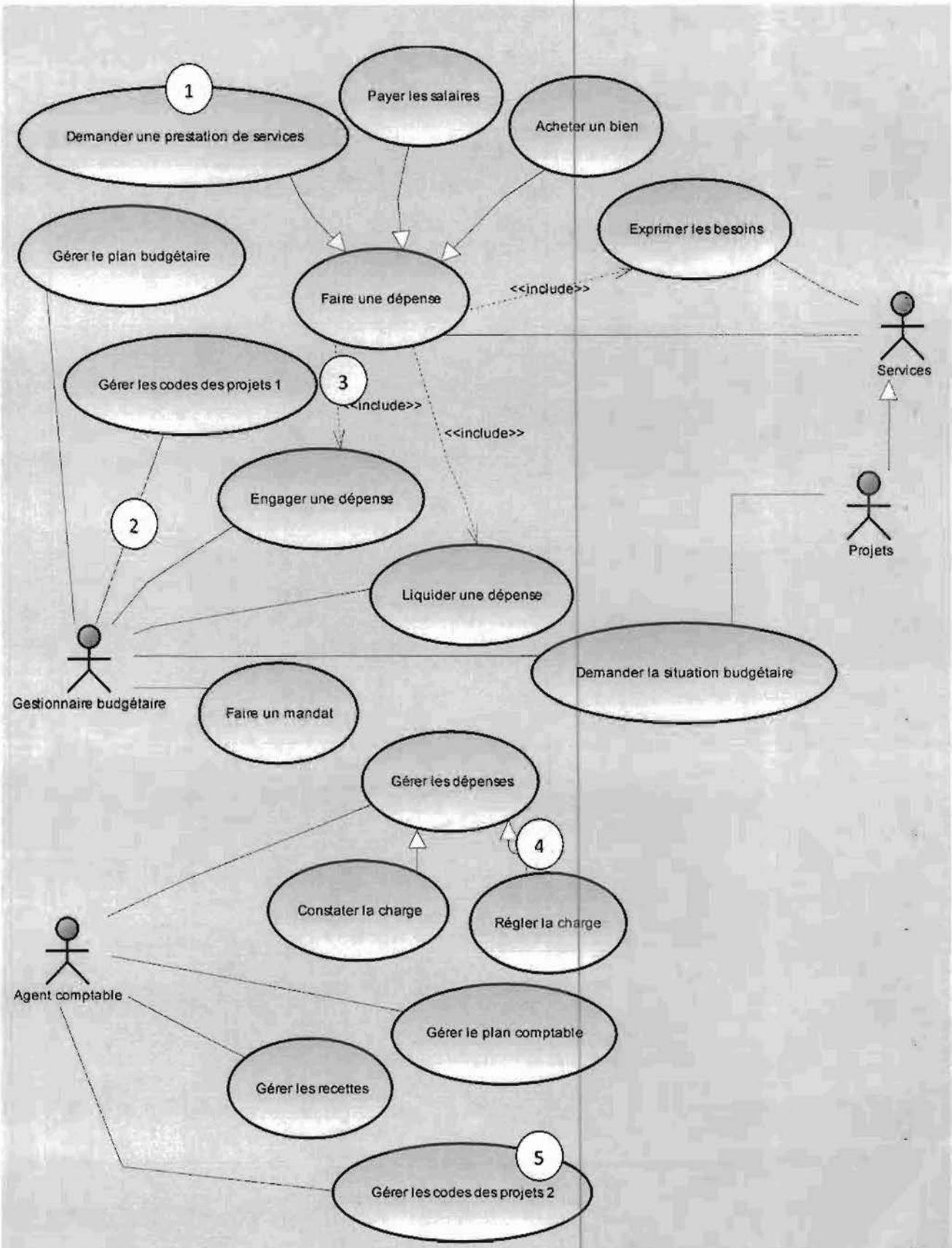


Figure 7 : diagramme de cas d'utilisation de l'existant.

Ce diagramme montre les différents cas d'utilisation du système actuel et leur lien avec les utilisateurs du système.

(1) : correspond à la matérialisation d'un cas d'utilisation ;

(2) : concerne un lien de déclenchement de la réalisation d'un cas d'utilisation par un utilisateur ;

(3) : représente un lien d'inclusion. Pour concrétiser une dépense, il faut impérativement faire un engagement. Idem pour les autres liens de ce type ;

(4) : montre une relation de généralisation/spécification entre le cas "Gérer les dépenses" et le cas "Constater la charge". Ce dernier représente un cas particulier dans la gestion des dépenses. Idem pour les autres relations de ce type ;

(5) : démontre la gestion séparée des codes des projets par les deux logiciels.

III.2- Description détaillée des cas d'utilisation

Pour décrire la dynamique du cas d'utilisation, le plus naturel consiste à recenser toutes les interactions de façon textuelle. Le cas d'utilisation doit par ailleurs avoir un début et une fin clairement identifiés. Il doit préciser quand ont lieu les interactions entre acteurs et système, et quels sont les messages échangés. Il faut également préciser les variantes possibles, telles que les différents cas nominaux, les cas alternatifs, les cas d'erreurs, tout en essayant d'ordonner séquentiellement les descriptions, afin d'améliorer leur lisibilité. Chaque unité de description de séquences d'actions est appelée *enchaînement*.

Les descriptions textuelles sont organisées de la façon suivante :

- **Sommaire d'identification** : inclut titre, but et acteurs.
- **Description des enchainements** : décrit les enchaînements nominaux, les enchaînements alternatifs, les enchaînements d'exception, mais aussi les préconditions, et les postconditions.

Dans le cadre de notre travail, tous les cas d'utilisation ont été décrits de façon textuelle. Cependant, dans cette partie seulement la description du cas d'utilisation "Engager une dépense" est présentée, le reste se trouve dans la partie *Annexe* du document.

Tableau I : Description détaillée du cas d'utilisation "Engager une dépense"

Titre : Engager une dépense
But : établir l'engagement d'une dépense
Acteur (s) : Gestionnaire budgétaire, DG, Contrôle financier
Pré condition (s) : le gestionnaire s'est authentifié
Description des enchainements : Ce cas commence lorsque le gestionnaire demande au système informatique l'établissement d'un engagement. **Scénario nominal** *Enchainement (a) Créer un engagement* Le gestionnaire après avoir accédé à la fenêtre d'engagement, introduit le numéro de la ligne budgétaire. MURAZGB10 affiche le libellé de la ligne et le crédit disponible. Le gestionnaire saisit les autres informations (montant, nature de la dépense, service bénéficiaire, date) et fournit les références des pièces justificatives. *Enchainement (b) Imprimer un engagement* Le gestionnaire précise l'engagement et lance l'impression. *Enchainement (c) Contrôler un engagement* Après avoir imprimé l'engagement, le gestionnaire l'envoie au Contrôle financier. Celui-ci vérifie l'engagement et le renvoie au gestionnaire avec son jugement. **Enchainements alternatifs** *Enchainement (d) Modifier un engagement* Le gestionnaire peut modifier un engagement encours ou non approuvé par le Contrôle financier. *Enchainement (e) Annuler un engagement* Le gestionnaire peut annuler un engagement encours ou non approuvé par le Contrôle financier.

Ce cas se termine lorsque : - l'engagement est approuvé par le Contrôle financier ; - ou annulé par le gestionnaire.
Exception (s) : néant
Post condition (s) : néant

IV- DIAGNOSTIC GENERAL

Le système d'information que nous étudions peut être divisé en deux (02) sous-systèmes : le système de la comptabilité et celui de la gestion budgétaire. Les deux ont chacun un système informatique qui leur permet d'améliorer leur travail quotidien. Ces deux systèmes sont incapables de s'échanger des informations. Pourtant, ils manipulent des informations communes — *engagements, liquidation, mandat, projets…*

Cette analyse de l'existant a permis de comprendre les dysfonctionnements et les atouts du système actuel.

IV.1- Points forts du SI

Les atouts que le système actuel présente sont les suivants :

- Le logiciel de gestion budgétaire, étant un classeur Excel, son environnement est familier aux utilisateurs qui ont déjà une expérience de ce dernier.
- Ce logiciel a des fonctionnalités qui permettent de suivre et d'exécuter le budget. Il a apporté un plus au SI, car il a facilité l'exécution des opérations du budget.
- Il permet d'avoir rapidement la situation budgétaire d'un projet.
- Comme autre point fort, ce logiciel exige un mot de passe pour l'utilisation de ses fonctionnalités.
- KHRONOS2000 permet de garder les traces des dépenses et recettes et facilite la consultation de celles-ci.

- Contrairement au logiciel de budget, le logiciel comptable sépare les données du traitement. Ainsi, les données peuvent être archivées facilement.

IV.2- Carences et dysfonctionnements du SI

L'analyse des carences et dysfonctionnements est la suivante :

- Le logiciel budgétaire observe une lenteur lors d'un calcul. Ce qui est fréquent avec les applications créées avec Excel adoptant le mode de calcul automatique, car il augmente le temps d'exécution.
- Ce logiciel est peu évolutif. Par exemple, un nombre de lignes était prévu pour créer les projets ; actuellement ce nombre est atteint et la DAF est obligé de faire appel au concepteur pour l'ajout d'un nouveau projet. Aussi, aucune fonctionnalité ne permet de créer le budget d'une autre année.
- Ce même logiciel budgétaire ne gère pas efficacement les modifications simultanées multiutilisateurs. Ce problème peut être résolu à une certaine limite en activant la fonctionnalité "*modifications multiutilisateurs*" d'Excel.
- La DAF qui a besoin des informations sur l'état des recettes des projets, ne les a pas en temps réel.
- KHRONOS2000 ne permet pas d'enregistrer le budget ; ce qui ne permet pas le rapprochement entre la situation réelle et les prévisions.
- L'interface de ce logiciel n'est pas facile d'utilisation.
- La maintenance du logiciel comptable est difficile. Aucune représentation du concepteur n'est présente au Burkina Faso.

V- RECONFIGURATION DU SI

Cette activité se base sur le diagnostic précédent pour reconfigurer le SI actuel. C'est une réponse aux points de carences et dysfonctionnement de ce système.

Comme l'analyse de l'existant l'a montré, il est impossible de fédérer les deux logiciels pour permettre les échanges de données. Une solution serait d'acquérir l'autre module (module de gestion budgétaire) de KHRONOS2000. Ainsi, le logiciel tout entier pourrait être déployé en

réseau. Cependant, comme le fait savoir le diagnostic ci-dessus, ce logiciel n'est pas facile d'utilisation et sa maintenance est difficile. Aussi, cette solution ne répond pas à un des objectifs du projet, prévoir l'exploitation des données par d'autres applications.

Il ne reste donc plus qu'à concevoir une autre application qui prend en compte tous les objectifs du projet. Pour mettre en place cette autre application, des propositions de systèmes sont faites. Ces propositions sont précédées d'une étude de technologies qui permet de faire le choix de certaines pour l'atteinte des objectifs.

V.1- Etude des technologies

Ce paragraphe fait l'état des technologies mises en œuvre pour l'atteinte des objectifs. Il étudie les différents types de base de données, puis ceux des architectures client-serveur.

V.1.1- Système de gestion de base de données (SGBD)

En informatique, une base de données est un lot d'informations stockées dans un dispositif informatique. Les technologies existantes permettent d'organiser et de structurer la base de données de manière à pouvoir facilement manipuler le contenu et stocker efficacement de très grandes quantités d'informations.

Le SGBD, qui est une collection de logiciels, est celui qui permet de manipuler et d'organiser les bases de données. Les données sont organisées selon un ***modèle de données***. Il en existe plusieurs : le *modèle hiérarchique*, le *modèle réseau*, le *modèle relationnel* et le *modèle objet*. Très rapidement, les deux (02) premiers, n'étant pas conçus sur des bases solides, ont montré leurs limites et ne sont pratiquement pas utilisés. Nous étudions donc les deux derniers modèles qui sont les plus utilisés de nos jours.

V.1.1.1- Le modèle relationnel

Les concepts mis en œuvre dans le modèle relationnel sont fondés sur une théorie mathématique directement issue de l'algèbre relationnelle, de la théorie des ensembles et de la logique formelle. Cette technologie a vu le jour dans les années 70 avec les travaux d'Edward Codd.

Bien que le modèle relationnel repose sur des principes simples, il permet néanmoins de modéliser des données complexes. Une relation est une structure de données tabulaire. Elle est composée d'attributs ; chaque attribut prend sa valeur dans un domaine. Physiquement, le concept de *relation* est implanté par une *table* dont les colonnes représentent les attributs et les lignes, les occurrences de la relation.

Le SGBD relationnel (SGBDR) est muni d'un langage de requête unique, non procédural et normalisé, SQL, qui permet de manipuler les tables.

V.1.1.2- Le modèle objet

Ci-dessus, nous avons vu qu'un objet est une représentation abstraite des entités du monde réel et qu'une classe est une collection d'objets semblables. En programmation orientée objet, un objet est une instance de classe. Les objets créés par un langage de programmation, lors de l'exécution d'un programme, disparaissaient une fois que celui-ci se terminait. Les bases de données objet, apparues en 1990, permettent de prolonger leur durée de vie. Dans ces bases, les données sont stockées sous forme d'objets.

Les SGBDO (SGBD Objet) bénéficient des vertus du concept objet. Cependant, ces systèmes manquent de normalisations [W7] ; chaque SGBDO propose son propre langage de requête.

V.1.2- Architecture client-serveur

Une application peut être considérée comme ayant une *architecture client-serveur*, si elle a au moins deux programmes distincts — *client* et *serveur* — qui communiquent. Le *client* initie toujours le dialogue en demandant les services du *serveur* qui se contente de répondre seulement aux requêtes du client. L'échange entre les deux processus est rendu possible grâce à un *middleware*. Ce dernier est un ensemble de couches réseau et de services logiciels.

Il existe plusieurs types d'architecture client-serveur :

V.1.2.1- L'architecture 2-tiers

Dans cette architecture, la partie cliente se contente de déléguer la gestion des données à un service spécialisé. Le cas typique de cette architecture est l'application de gestion fonctionnant sous Ms-Windows et exploitant un SGBD centralisé.

Le SGBD centralisé est exécuté le plus souvent sur un serveur dédié. Ce dernier est interrogé en utilisant un langage de requête qui, généralement, est SQL. Le dialogue entre client et serveur se résume donc à l'envoi de requêtes et au retour des données correspondantes aux demandes.

Ce modèle présente de nombreux avantages :

- il permet l'utilisation d'une interface riche ;
- il a permis l'appropriation des applications par les utilisateurs ;
- les middlewares proposés par les fournisseurs de SGBD sont très performants et permettent de tirer profit de l'ensemble des fonctionnalités du serveur de données pour lequel ils ont été conçus.

Cependant, cette première génération de client/serveur présente quelques limites :

- le client est lourd. Il supporte la grande majorité des traitements applicatifs ;
- le client est fortement sollicité, il devient de plus en plus complexe et doit être mis à jour régulièrement pour répondre aux besoins des utilisateurs ;
- cette architecture est difficile à maintenir.

V.1.2.2- L'architecture 3-tiers

L'architecture 2-tiers s'est heurtée à la complexité du "client". La solution se trouverait donc dans l'utilisation d'un client simple communicant avec le serveur. Ainsi, l'architecture 3-tiers a été proposée dont les principes sont les suivants :

- les données sont toujours gérées de façon centralisée ;
- la présentation est toujours prise en charge par le poste client ;
- la logique applicative est prise en charge par un serveur intermédiaire.

Encore appelée client-serveur de deuxième génération ou client-serveur distribué, elle sépare l'application en trois niveaux distincts :

- ***premier niveau*** : l'affichage et les traitements locaux (contrôles de saisie, mise en forme de données...) sont pris en charge par le poste client ;
- ***deuxième niveau*** : les traitements applicatifs globaux sont pris en charge par le service applicatif ;
- ***troisième niveau*** : les services de base de données sont pris en charge par un SGBD.

Tous ces niveaux étant indépendants, ils peuvent être implantés sur des machines différentes, de ce fait :

- le poste client ne supporte plus l'ensemble des traitements, il est moins sollicité et peut être moins évolué, donc moins coûteux ;
- les ressources présentes sur le réseau sont mieux exploitées, puisque les traitements applicatifs peuvent être partagés ou regroupés ;
- la fiabilité et les performances de certains traitements se trouvent améliorées par leur centralisation ;
- il est relativement simple de faire face à une forte montée en charge, en renforçant le service applicatif.

Le cas typique de cette architecture, est l'application Web : le poste client prend la forme d'un simple navigateur Web et le service applicatif est assuré par un serveur Web.

Cependant, le client étant constitué souvent d'un navigateur, rencontre des difficultés au niveau de l'ergonomie. Aussi, le serveur Web constituant l'élément crucial de cette architecture, se trouve souvent sollicité. Il est difficile de répartir la charge entre client et serveur. Une solution serait de répartir la charge du serveur Web.

V.1.2.3- L'architecture n-tiers

Cette architecture est apparue afin de faciliter la répartition de la charge entre les différents niveaux. Elle est une variante de l'architecture précédente.

Cette évolution des architectures trois tiers met en œuvre une approche objet. Ceci, afin d'offrir une plus grande souplesse d'implémentation et faciliter la réutilisation des développements. Ce modèle vise à :

- permettre l'utilisation d'interfaces utilisateurs riches ;
- séparer nettement tous les niveaux de l'application ;
- offrir de grandes capacités d'extension ;
- faciliter la gestion des sessions.

L'architecture n-tiers qualifie la distribution d'application entre de multiples services et non la multiplication des niveaux de service. Cette distribution est facilitée par l'utilisation de composants métiers, spécialisés et indépendants introduits par les concepts orientés objets — langages de programmation et middleware. Elle permet de tirer pleinement partie de la notion de composants métiers réutilisables.

Ces composants rendent un service si possible générique et clairement identifié. Ils sont capables de communiquer entre eux et peuvent donc coopérer en étant implantés sur des machines distinctes.

Pour permettre la répartition d'objets entre machines et l'intégration des systèmes "non objets", il doit être possible d'instaurer une communication entre tous ces éléments. Ainsi, le concept de *middleware objet* est né. Trois des paradigmes populaires de middleware objet sont ***Common Object Request Broker Architecture (CORBA)***, ***Distributed Component Object Model (DCOM)*** et ***Remote Method Invocation (RMI)***.

V.1.2.3.1- CORBA

Depuis 1989, l'*OMG* (*Object Management Group*) s'est activement attaché à définir un bus global de composants répartis — *CORBA*. Ce dernier est un bus à objets. Il définit la forme des composants qui y vivent, ainsi que leur mode d'interopérabilité. Les programmes peuvent

s'appuyer sur un *langage de définition d'interface (IDL)*, qui est un langage neutre, pour définir les frontières des composants. Ces frontières sont des interfaces contractuelles que les composants ont avec leurs clients potentiels.

Ce middleware fournit les caractéristiques suivantes :

- *La liaison avec « tous » les langages de programmation* : cependant, seulement les langages C, C++, SmallTalk, Ada, COBOL, Java... ont été définis officiellement pour cette liaison.
- *La transparence des invocations* : les requêtes aux objets semblent toujours être locales, le bus CORBA se chargeant de les acheminer en utilisant le canal de communication le plus approprié.
- *L'invocation statique et dynamique* : ces deux mécanismes complémentaires permettent de soumettre les requêtes aux objets. En statique, les invocations sont contrôlées à la compilation. En dynamique, les invocations doivent être contrôlées à l'exécution.
- *Un système auto-descriptif* : les interfaces des objets sont connues du bus et sont aussi accessibles par les programmes via le référentiel des interfaces.
- *L'activation automatique et transparente des objets* : les objets sont en mémoire uniquement s'ils sont utilisés par des applications clientes.
- *L'utilisation sur plusieurs plateformes de système d'exploitation.*

V.1.2.3.2- DCOM

DCOM, Modèle Objet de Composants Distribués, tire son origine de la technologie *OLE* de Microsoft (en français, *Intégration d'objets et Lien sur des objets*). Cette dernière a été développée, initialement, dans le but de permettre la programmation d'objets capables d'être insérés dans des applications soit par intégration complète, soit par référence.

Par la suite, OLE a été étendue en introduisant le *Modèle Objet de Composants — COM.* L'objectif de ce dernier est d'assurer la communication inter-applications et l'intégration de documents dans les applications Microsoft.

Cette technologie a, ensuite, été complétée pour permettre la répartition des composants en réseau. Cette version de composants distribués en réseau est *DCOM* et a vu le jour en 1998.

Même si elles utilisent des techniques différentes, les deux technologies, CORBA et DCOM sont similaires. Sauf que, DCOM ne supporte, efficacement, qu'une seule plateforme de système d'exploitation — Windows.

V.1.2.3.3- RMI

RMI est un middleware objet, version Java, permettant de manipuler des objets distants de manière transparente pour l'utilisateur. Ainsi, un serveur permet à un client d'invoquer des méthodes à distance sur un objet qu'il instancie.

RMI reposant sur Java, peut être utilisé sur plusieurs plateformes de système d'exploitation. Aussi, il reste le middleware le plus facile à mettre en œuvre.

Cependant, RMI ne permet de faire communiquer que les applications Java et propose moins de services que les autres.

V.2- Architecture préliminaire de SGBCoM

Le système à mettre en place, dénommé *SGBCoM — Système de Gestion Budgétaire et Comptable de MURAZ* — met en communication les différents services concernés. Il donne à chaque utilisateur les données dont il a besoin. Deux (02) propositions sont faites à cet effet, et tiennent compte de l'intégration des services.

Ces deux (02) propositions peuvent être distinguées selon le mode d'accès des projets au système.

V.2.1- Accès par Internet (AI)

Cette solution offre la possibilité aux projets d'accéder au système à travers l'Internet. Comme il est montré au niveau ci-dessus, presque tous les services dirigeant les projets ont un accès à Internet. Aussi, très prochainement, les machines des trois services — DAF, Contrôle financier, Agence comptable — seront reliés par le même réseau.

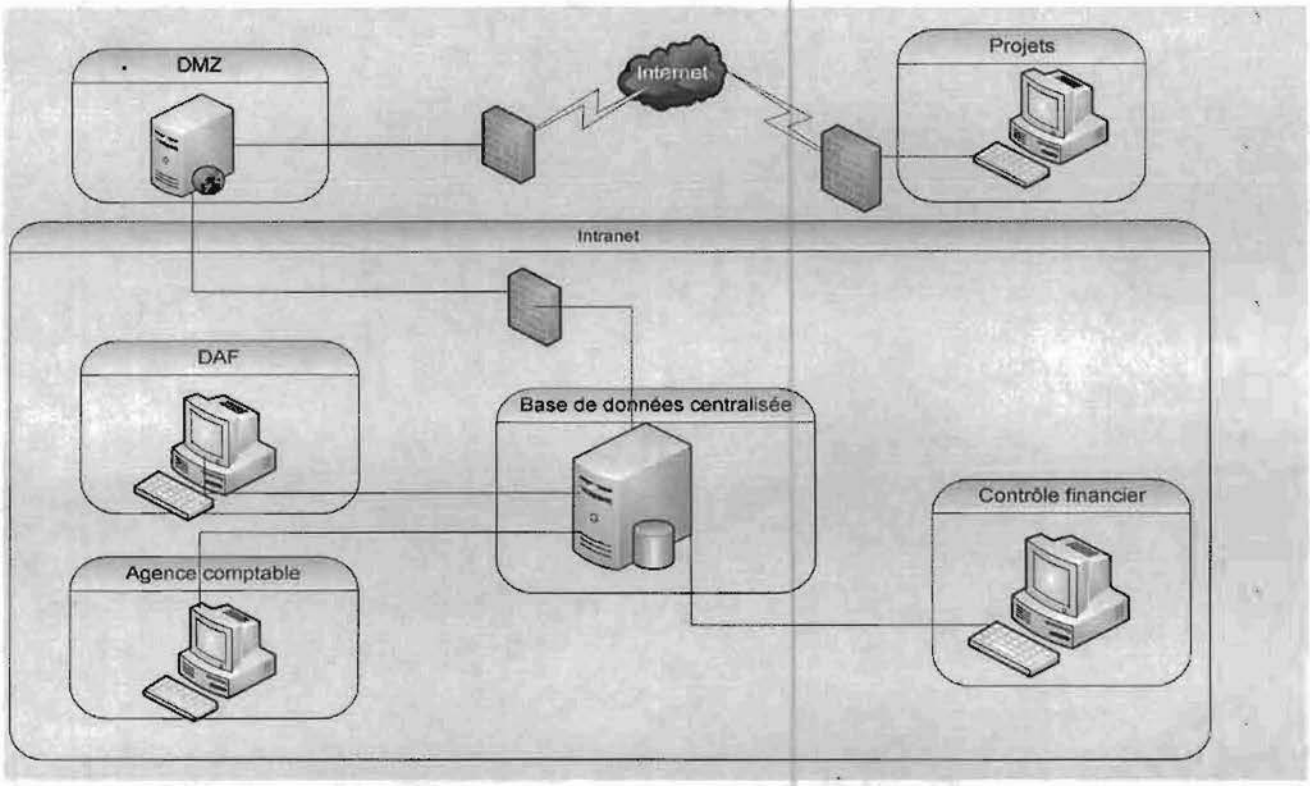


Figure 8 : Architecture AI préliminaire de SGBCOM.

Pour la mise à œuvre, la partie Web sera hébergée au sein du Centre MURAZ. Pour cela, il faut une *liaison spécialisée* (LS). Cette liaison est un abonnement auprès du fournisseur d'accès à Internet (ONATEL-SA) ; il donne droit à une adresse publique.

Tableau II : Redevance mensuelle de la LS et frais d'abonnement [B5].

Débit	Redevance mensuelle (F CFA TTC)	Frais d'accès (F CFA TTC)
2 Mbits	1 416 000	556 960
Total		**1 972 960**

- **Avantages**
  - facile à mettre en œuvre : la plupart des projets ayant un accès à internet, on n'a pas besoin de faire une reconfiguration majeure ;
  - permet l'intégration de tous les services concernés ;

- **Inconvénients**
  - les données étant centralisées en cas de panne du serveur les travaux à la DAF et à l'Agence comptable se trouveront paralysés ;
  - cette architecture est moins sécurisée à cause de la DMZ ;

- sa mise en œuvre est coûteuse à long terme.

V.2.2- Accès par réseau local (AR)

La solution actuelle prévoit la mise en réseau de toutes les machines du Centre MURAZ. Ainsi, les projets pourront accéder au système à travers ce réseau local. Il est prévu un accès des projets par WiFi. Cette liaison sans fil a beaucoup d'avantages par rapport à celle filaire. Elle est facile à mettre œuvre et peut évoluer aisément.

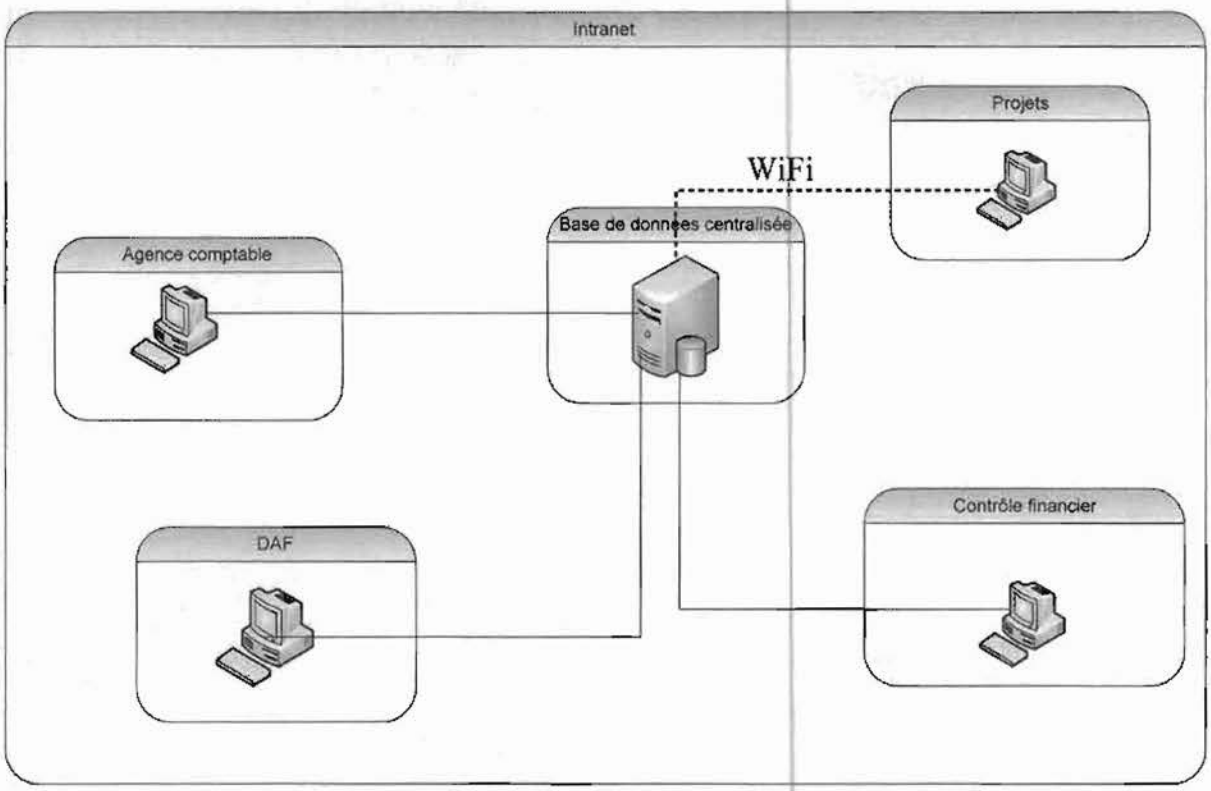


Figure 9 : Architecture AI préliminaire de SGBCOM.

Pour la mise œuvre il faut les logiciels et les matériels qui sont données par les tableaux *III* et *IV* [B5].

Tableau III : Logiciels à acquérir et leurs coûts.

Désignation	Qté	Prix unitaire	Prix total (F CFA HT)
Logiciel Windows 2003 server Entreprise Edition	1	2 234 380	2 234 380
License pour utilisateur Windows 2003 server	1	27 578	27 578
License antivirus Kaspersky 2010	1	38 000	38 000
Total 2			**2 299 958**

Tableau IV : Matériels à acquérir et leurs coûts.

Désignation	Qté	Prix unitaire	Prix total (F CFA HT)
Serveur HP Proliant ML150	1	854 130	854 130
Antenne Omni WiFi Dlink Ant 24-1500	1	950 000	950 000
Modem Routeur wifi Linksys Technologie N	1	225 000	225 000
Points d'accès Linksys ou répéteurs 1	27	70 000	890 000
Rouleau câble RJ45 cat6 blindé	1	135 000	135 000
Carte WiFi	86	17 500	1 505 000
Câble de descente type CF200-NL500Hms coaxial (15m+connectique)	2	85 000	170 000
Total 1			**5 879 130**

Qté = Quantité.

- **Avantages**
  - sécurité des données plus élevées si toutes les dispositions sont prises pour la mise en place du réseau, aucune donnée n'est vue de l'extérieur ;
  - cette solution permettra de doter le Centre MURAZ d'un réseau unique où toutes les machines pourront communiquer ;
  - coût de mise en œuvre moins élevé à long terme ;
  - ce réseau sera une ouverture pour la mise en place d'un intranet.

- **Inconvénients**

L'inconvénient majeur de cette solution est l'administration du réseau. Cette dernière sera complexe à cause de la grandeur du réseau.

V.2.3- Choix de la solution

Parmi les deux propositions précédentes, AR présente beaucoup d'avantages. Aussi, Elle répond non seulement aux objectifs du projet, mais également aux préoccupations des utilisateurs.

Pour la réalisation de cette solution, les technologies suivantes ont été adoptées :

- POSTGRESQL qui est un SGBD basé sur le modèle relationnel. Ce système est très riche fonctionnellement et facile à acquérir. Aussi, il gère de façon efficace les bases de données de taille moyenne.

- CORBA qui est un middleware objet permettant la séparation nette des couches d'une application. Parmi les middlewares objets présentés ci-dessus, il est le plus mature et apporte beaucoup de possibilités.

CONCLUSION

Cette analyse des besoins a permis, à travers les interviews et l'étude des logiciels existants, de comprendre les préoccupations des utilisateurs et le système actuel. Les fonctionnalités de ce dernier ont été regroupées en cas d'utilisation, puis décrites de façon détaillée. C'est cette description détaillée des cas d'utilisation qui a permis de dresser un diagnostic. Ce bilan a révélé deux (02) systèmes informatiques, utilisés par deux (02) services différents, incapables de s'échanger des informations ; pourtant manipulant des données communes.

Il a donc fallu proposer des solutions afin de permettre l'échange des données entre les différents services. Deux propositions ont ainsi été faites et une a été retenue. La solution retenue prévoit la construction d'une nouvelle application qui prend en compte les préoccupations des utilisateurs et l'intégration des services concernés. Il s'agit maintenant d'étudier les fonctionnalités de cette application à la phase suivante.

CHAPITRE 3 :

SPECIFICATION

INTRODUCTION

Cette phase, qui constitue à étudier les fonctionnalités de la solution retenue, commence par la délimitation du contour du futur système et le recueil des cas d'utilisation. La description de ces cas donne le diagramme de classes métier qui est structuré après en *catégories*.

I- CAPTURE DES BESOINS FONCTIONNELS

Cette activité représente un point de vue fonctionnel de l'architecture système. En collaboration avec les utilisateurs, les besoins fonctionnels ont pu être recensés pour le système futur.

I.1- Modélisation du contour de SGBCoM

Il s'agit de repérer les contours de SGBCoM en déterminant les acteurs potentiels qui interagiront avec le système. Après avoir identifié les acteurs principaux, un diagramme de contexte statique est ensuite utilisé pour montrer les interactions possibles entre ces acteurs et le système.

I.1.1- Identification des acteurs

Rappelons qu'un acteur incarne le rôle d'une entité externe interagissant avec le système. Les acteurs principaux interagissant avec SGBCoM sont :

- ***Projets*** : ils établissent leurs expressions des besoins et consultent leurs situations budgétaires.
- ***Gestionnaires budgétaires*** : ils ont en charge le suivi et l'exécution du budget.
- ***Contrôle financier*** : il contrôle les opérations des agents budgétaires.
- ***Agence comptable*** : elle enregistre les recettes et les dépenses.
- ***Administrateur*** : il gère les utilisateurs du système.

I.1.2- Diagramme de contexte statique

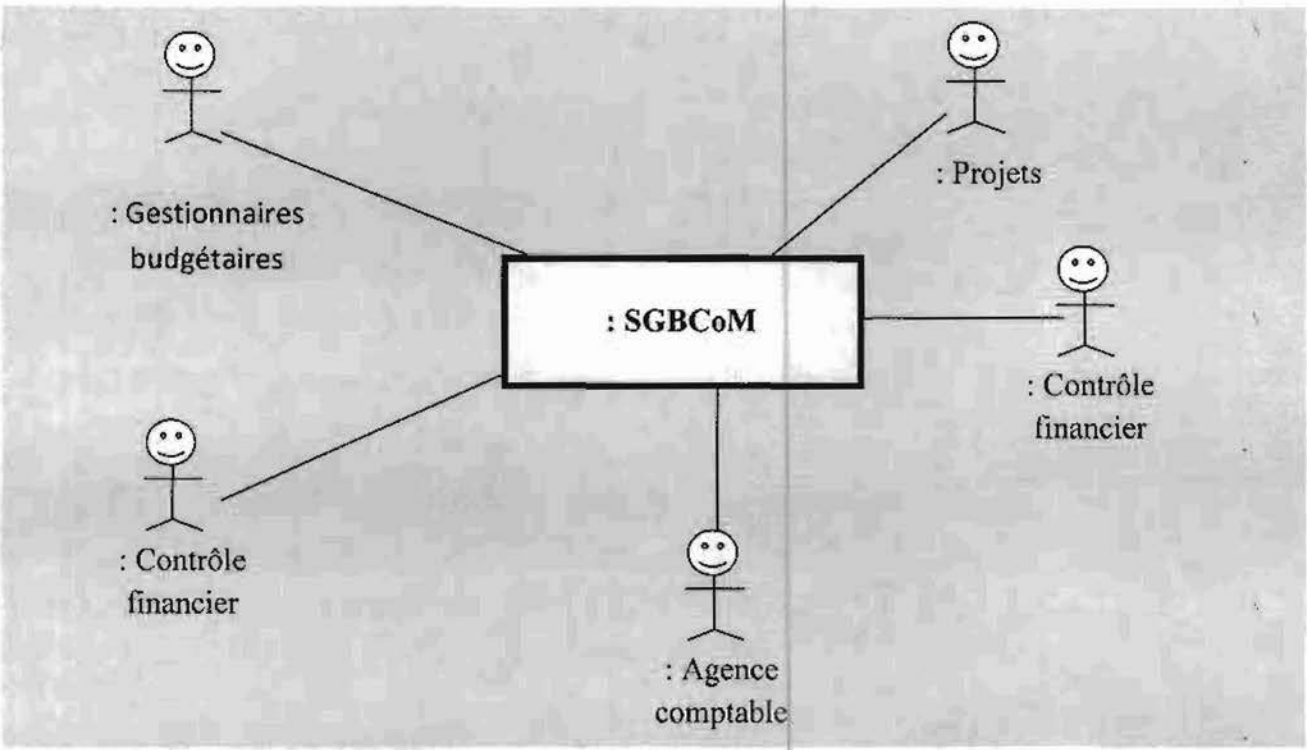


Figure 10 : Diagramme de contexte statique de SGBCoM.

Ce diagramme montre que tous les utilisateurs accèdent à un et un seul système informatique contrairement au système actuel.

I.2- Identification des cas d'utilisation

Pour rappel, un cas d'utilisation représente un ensemble de séquences d'actions réalisées par le système et produisant un résultat observable intéressant pour un acteur particulier.

En se basant sur l'analyse des besoins, les cas d'utilisation ci-dessous ont pu être dénombrés.

Engager une dépense (*Gestionnaires budgétaires*)

Intention : engager une dépense d'un projet ou de l'administration.

Actions : créer un nouvel engagement, modifier ou annuler un engagement.

Faire une liquidation (*Gestionnaires budgétaires*)

Intention : élaborer une liquidation correspondant à un engagement.

Actions : créer une nouvelle liquidation, modifier ou annuler une liquidation.

Etablir un mandat (*Gestionnaires budgétaires*)

Intention : éditer un mandat correspondant à une liquidation.

Actions : imprimer un mandat ou annuler l'impression.

Réaménager le budget d'un projet (*Gestionnaires budgétaires*)

Intention : accorder plus de crédit à une ligne budgétaire d'un projet.

Actions : vérifier le crédit disponible des autres lignes, débiter les lignes sélectionnées.

Mettre à jour les codes des projets (*Gestionnaires budgétaires*)

Intention : mise à jour des paramètres d'un projet.

Actions : créer un nouveau projet, supprimer un projet ou modifier les paramètres d'un projet.

Consulter situation budgétaire (*Projets, Gestionnaires budgétaires*)

Intention : connaitre la situation budgétaire du projet.

Actions : choisir le projet.

Etablir une expression des besoins (*Projets*)

Intention : éditer leurs expressions des besoins.

Actions : créer une nouvelle expression des besoins, modifier ou annuler une expression des besoins.

Gérer un budget (*Gestionnaires budgétaires*)

Intention : constituer un budget.

Actions : créer un budget avec ses lignes, supprimer et modifier les lignes.

Gérer le plan comptable (*Agence comptable*)

Intention : mettre en place le plan comptable.

Actions : créer les différentes rubriques du plan, supprimer et modifier les rubriques.

Gérer le plan budgétaire (*Gestionnaires budgétaires*)

Intention : mettre en place le plan budgétaire.

Actions : créer les différentes rubriques du plan, supprimer et modifier les rubriques.

Gérer les comptes auxiliaires

Intention : créer des comptes clients et fournisseur.

Actions : ajouter un compte, supprimer et modifier un compte.

Gérer les dépenses (*Agence comptable*)

Intention : enregistrer les dépenses.

Actions : débiter et créditer des comptes.

Gérer les profils utilisateur (*Administrateur*)

Intention : octroyer des droits à un utilisateur.

Actions : créer un nouvel utilisateur, modifier le profil d'un utilisateur ou supprimer un utilisateur.

S'authentifier (*tous les utilisateurs*)

Intention : accéder au système ou utiliser une fonctionnalité de ce dernier.

Actions : saisir le login et mot de passe.

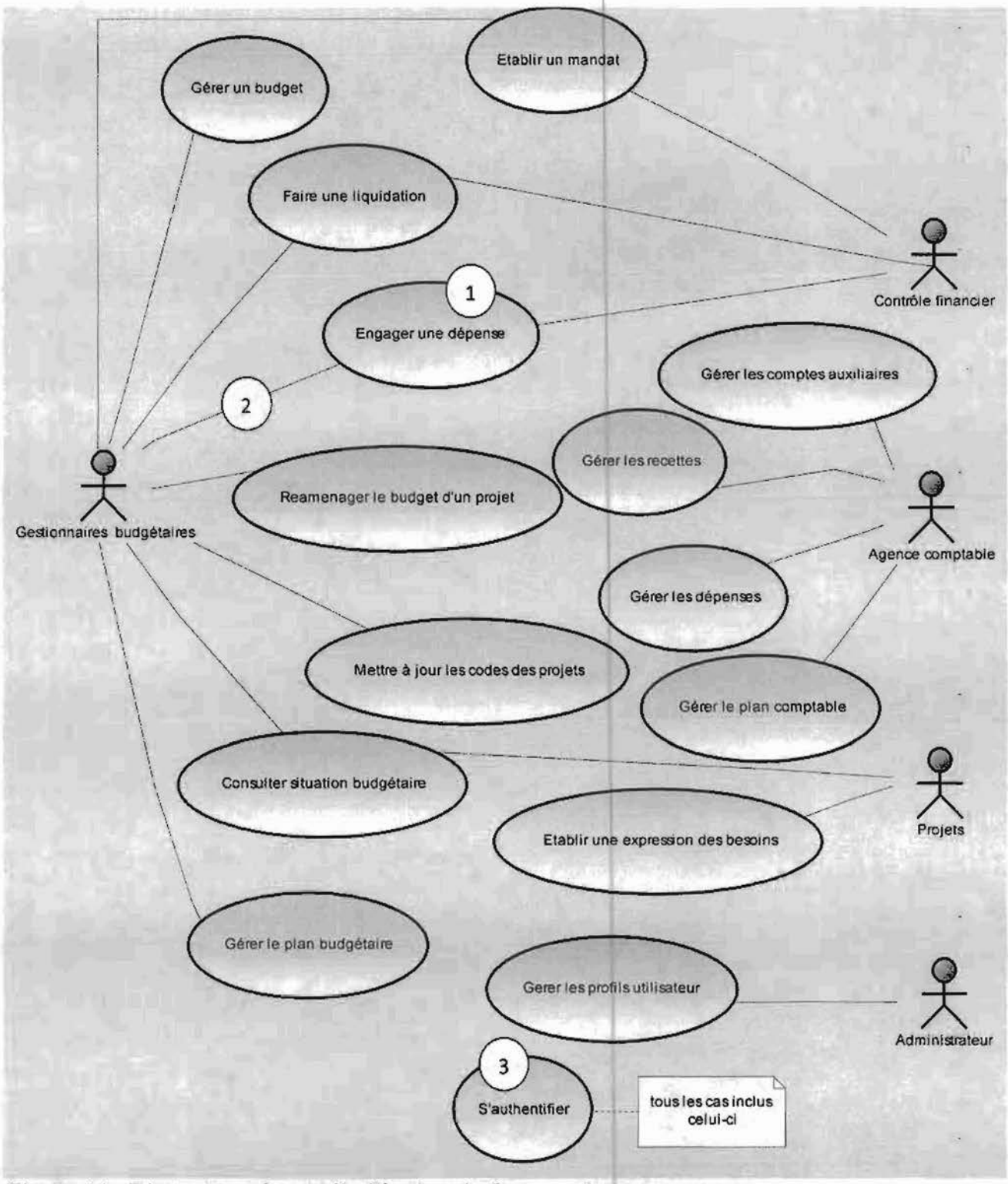


Figure 11 : Diagramme de cas d'utilisation du futur système.

Ce diagramme montre les différents cas d'utilisation de SGBCoM et leur lien avec les utilisateurs du système.

(1) : correspond à la matérialisation d'un cas d'utilisation ;

(2) : concerne les liens de participation à la réalisation des cas d'utilisation par les utilisateurs ;

(3) : représente le cas d'utilisation que tous le reste doit exécuter dans leur réalisation.

Ce diagramme est l'amélioration de celui du système actuel ; des cas ont émergé et d'autres ont été supprimés :

- "Gérer les profils utilisateur" est apparu afin de gérer l'accès au système. Cette gestion est faite par la création de comptes utilisateurs avec des droits spécifiques.
- "Mettre à jour les codes" ce cas montre la gestion des codes en un seul endroit.

I.3- Description détaillée des cas d'utilisation

Chaque cas d'utilisation, dans cette partie, est décrit de façon détaillée. Ces descriptions montrent l'intention de l'acteur lorsqu'il utilise le système et les séquences d'actions principales qu'il est susceptible d'effectuer. Ces définitions servent à fixer les idées et n'ont pas pour but de spécifier un fonctionnement complet et irréversible.

Dans notre travail, nous avons décrit tous les cas d'utilisations énumérés ci-dessus, cependant dans le cadre de ce document seulement le cas "Engager une dépense" est présenté.

Tableau V : Description textuelle du cas d'utilisation "Engager une dépense".

Titre : Engager une dépense
But : établir un engagement pour une expression
Acteur (s) : Gestionnaires budgétaires, Contrôleur financier
Précondition (s) : - le gestionnaire s'est authentifié ; - une expression a été envoyée.
Description des enchainements : Ce cas commence lorsque le Gestionnaire demande au système l'édition d'un nouvel engagement.

Enchainements nominaux

Enchainement (a) Créer un engagement

Le gestionnaire après avoir demandé la création d'un nouvel engagement, le système le crée en lui attribuant un numéro. Le gestionnaire indique la date puis le numéro de l'expression des besoins. Le système affiche les paramètres de cette expression des besoins (ligne budgétaire, montant, ...).

Si l'expression n'a pas été validée par le contrôle financier alors exécuter :

[Exception 1 : expressionNonValidée]

Si le montant de l'engagement est supérieur au crédit disponible alors il faut exécuter :

[Exception 2 : créditInsuffisant]

Enchainement (b) Fournir les références des pièces justificatives

Le gestionnaire indique au fur et à mesure les références (date, montant, ...) de toutes les pièces justificatives.

Enchainements (c) Valider uniquement un engagement en construction

Le gestionnaire valide l'engagement en construction. Le système enregistre les données dans la base de données.

Enchainements (d) Imprimer un engagement en construction

Le gestionnaire valide l'engagement en construction. Le système enregistre les données et imprime l'engagement.

Enchainements (e) Imprimer un engagement validé

Le gestionnaire indique l'engagement et valide. Le système imprime l'engagement.

Enchainements (f) Contrôler un engagement

Le contrôleur financier précise l'engagement. Le système détermine les informations correspondantes. Le contrôle vérifie l'engagement et introduit son jugement (engagement approuvé ou non approuvé).

Enchainements alternatifs

Enchainements (g) Modifier un engagement en construction

Le gestionnaire peut modifier au fur et à mesure les informations sur l'engagement.

Enchainements (h) Modifier un engagement

Le gestionnaire peut modifier un engagement qui n'a pas été encore approuvé par le contrôle financier ou un engagement dont la liquidation n'a pas commencé. La modification d'un engagement approuvé par le contrôle financier, le ramène à un état non agréé.

Enchainements (i) Annuler un engagement

Le gestionnaire peut annuler un engagement en construction ou un engagement validé dont la liquidation n'a pas commencé. Dans ce dernier cas, il indique l'engagement et valide. Le système le met dans un état *annulé*. Tout engagement annulé est détruit après un temps défini par l'administrateur.

Ce cas se termine lorsque :

- le gestionnaire obtient l'engagement approuvé ;
- ou le gestionnaire annule l'engagement.

Exception (s) :
[Exception 1 : expressionNonValidée], [Exception 2 : créditInsuffisant]

Le système informe le gestionnaire par un message.

Postcondition (s) : le crédit disponible est suffisant

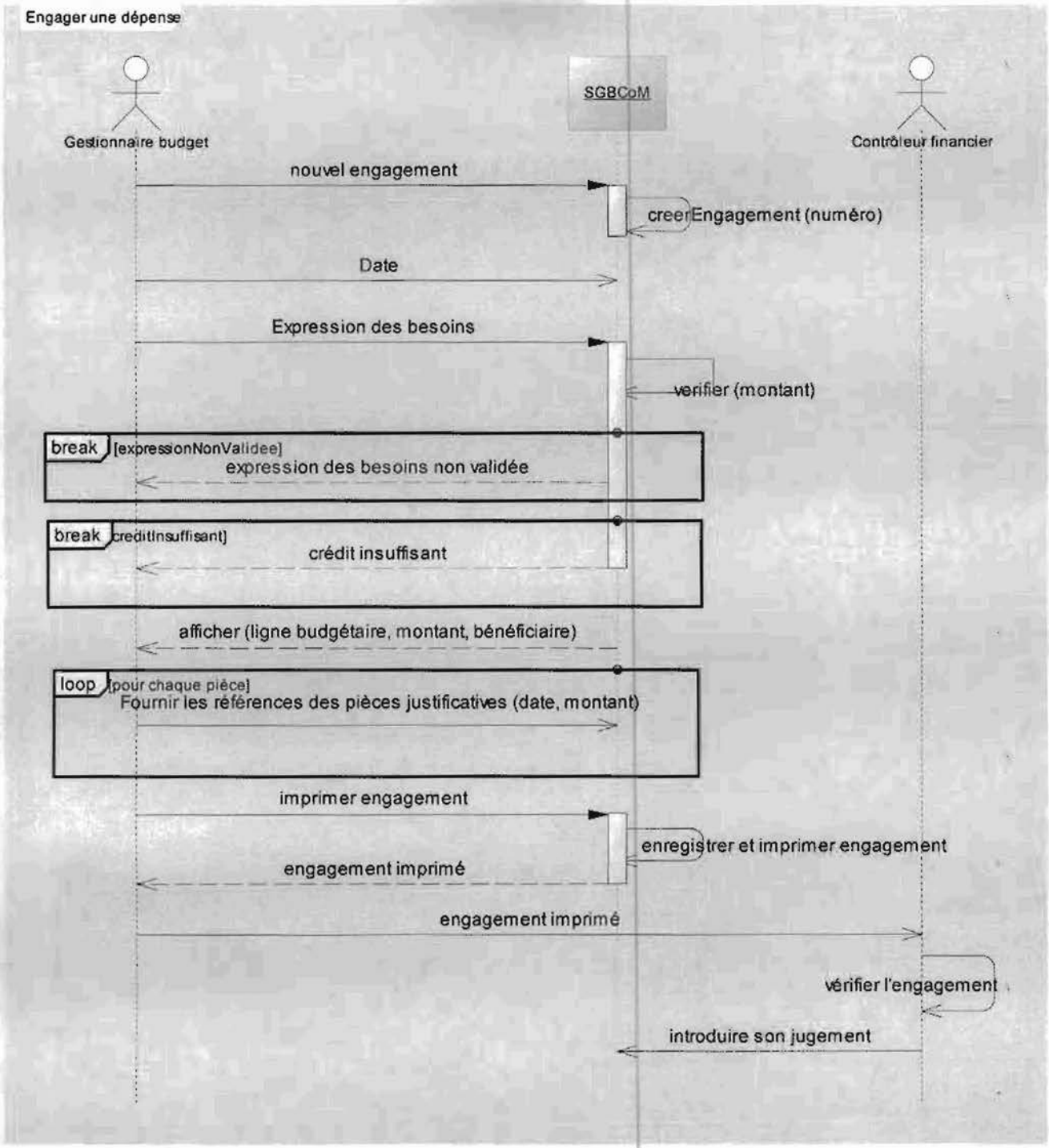


Figure 12 : Diagramme de séquence du scénario nominal du cas "Engager une dépense".

II- DEVELOPPEMENT DU MODELE DE CLASSES D'ANALYSE

Le modèle de classes d'analyse ou "métier" s'exprime par les diagrammes composés de classes et de relations entre ces dernières. Il s'agit de représenter les classes clés du domaine et d'exprimer les liens entre celles-ci en utilisant seulement les associations. Pour enrichir ces classes, celles justifiant un cycle de vie des objets significatifs sont décrits par un diagramme d'états-transitions.

II.1- Identification des classes

La technique utilisée pour identifier les classes est la suivante :

- La recherche des noms communs importants dans les descriptions textuelles des cas d'utilisation.
- La vérification des propriétés "objet" de chaque concept — identité, propriétés, comportement —, puis définir ses responsabilités.

Après avoir appliqué cette technique, les classes suivantes ont pu être identifiées :

Classe : Engagement		
Attributs		
Désignation	**Description**	**Type**
numeroEng	Représente le numéro attribué à un engagement	Alphanumérique
dateEng	Date de la création de l'engagement	Date
creditDispo	Représente le crédit disponible au moment de l'engagement	Numérique
montantEng	Le montant de la dépense à engager	Numérique
encours	L'état d'un engagement qui n'a pas, encore, été contrôlé	Booléen
Rejete	Un engagement jugé non valide par le contrôle financier	Booléen
annule	L'état d'un engagement qui a été annulé	Booléen
Liquide	Vrai si l'engagement a été liquidé totalement	Booléen

Classe : Liquidation **Rôle :** concerne les propriétés d'une liquidation		
Attributs		
Désignation	**Description**	**Type**
numeroLiquid	Correspond au numéro attribué à une liquidation	Alphanumérique
dateLiquid	Date d'établissement d'une liquidation	Date
montantLiquid	Le montant à liquider	Numérique
encours	Vrai si la liquidation n'a pas encore été contrôlée	Booléen
rejetee	Une liquidation jugée non valide par le contrôle financier	
Mandatee	L'état d'une liquidation dont le mandat de paiement a été tiré	Booléen

Classe : Mandat		
Attributs		
Désignation	**Description**	**Type**
numeroMandat	Le numéro du mandat de paiement	Alphanumérique
dateMandat	La date d'édition du mandat	Date
encours	Un mandat encours de payement	Booléen
paye	Un mandat réglé	Booléen

Classe : Contrôle		
Attributs		
Désignation	**Description**	**Type**
numeroControle		Alphanumérique
dateControle	La date à laquelle le contrôle a été effectuée	Date
Rejet	Vrai si le document n'est pas approuvé par le contrôle financier	Booléen
Commentaire	Les motifs de rejet ou de validité	Alphanumérique

Classe : Réaménagement		
Attributs		
Désignation	**Description**	**Type**
numeroReamenag	Le numéro de réaménagement	Alphanumérique
dateReamenag	La date à laquelle le réaménagement a été effectué	Date
annule	L'état d'un réaménagement annulé	Booléen
impute	La ligne bénéficiaire a été imputée depuis le réaménagement	Booléen

Classe : Budget		
Attributs		
Désignation	**Description**	**Type**
debut	Début de la période d'exécution du budget	Date
Fin	Fin de la période d'exécution du budget	Date
libelle	Désignation du budget	Alphanumérique

Classe : LignesBudgetaires		
Attributs		
Désignation	**Description**	**Type**
dotationInitiale	Prévision de départ des dépenses pour la ligne	Numérique
dotationFinale	Prévision après réaménagement du budget	Numérique
creditDispo	Dotation finale moins engagement total sur la ligne	Numérique

Classe : CompteUtilisateur		
Attributs		
Désignation	**Description**	**Type**
Login	Nom de connexion de l'utilisateur	Alphanumérique
motPasse	Mot de passe	Alphanumérique
profil	Les types de profils : *Administrateur* : tous les droits *DAF* : Gestion du budget *CF* : contrôle d'un engagement, d'une liquidation… *AC* : comptabilité *Projets* : consultation et expression des besoins *Services* : expression des besoins	Alphanumérique
Actif	Vrai pour un utilisateur qui peut accéder au système	Booléen

Classe : Pièces justificatives (ci-après PJ)		
Attributs		
Désignation	**Description**	**Type**
datePJ	Date de délivrance la pièce	Date
libelle	Désignation de la pièce	Alphanumérique
montantPJ	Montant de la pièce	Numérique

Classe : ModeReglement		
Attributs		
Désignation	**Description**	**Type**
modePayement	Représente les différents modes de paiement (espèce,	Alphanumérique

Classe : Titre		
Attributs		
Désignation	**Description**	**Type**
codeTitre	Code du titre	Alphanumérique
libelleTitre	Libellé du titre	Alphanumérique

Classe : Section		
Attributs		
Désignation	**Description**	**Type**
codeSection	Code section	Alphanumérique
libelleSection	Libellé section	Alphanumérique

Classe : Chapitre		
Attributs		
Désignation	**Description**	**Type**
codeChapitre	Code chapitre	Alphanumérique
libelleChapitre	Libellé	Alphanumérique

Classe : Article		
Attributs		
Désignation	**Description**	**Type**
codeArticle	Code article	Alphanumérique
libelleArticle	Libellé article	Alphanumérique

Classe : Paragraphe		
Attributs		
Désignation	**Description**	**Type**
codeParagraphe	Code paragraphe	Alphanumérique
libelleParagraphe	Libellé paragraphe	Alphanumérique

Classe : Rubrique		
Attributs		
Désignation	**Description**	**Type**
codeRubrique	Code rubrique	Alphanumérique
libelleRubrique	Libellé rubrique	Alphanumérique

	chèque, virement bancaire)	
domiciliation	Nom de la banque	Alphanumérique
numeroCompte	Numéro de compte bancaire	Alphanumérique

Classe : Projet		
Attributs		
Désignation	**Description**	**Type**
codeProjet	Code attribué à un projet dans le cadre du budget	Alphanumérique
libelleProjet	L'appellation d'un projet	Alphanumérique

Classe : ExpressionBesoin		
Attributs		
Désignation	**Description**	**Type**
numeroExpression	Numéro de l'expression des besoins	Alphanumérique
dateExpression	Date de création de l'expression des besoins	date

Classe : LignesExpressions		
Attributs		
Désignation	**Description**	**Type**
references	Référence d'une ligne de l'expression des besoins (par exemple un produit)	Alphanumérique
detailBesoins	Désignation de la ligne	Alphanumérique
quantite	Le nombre de produit	Numérique
prixUnitaire	Le prix unitaire du produit	Numérique

Classe : Agent		
Attributs		
Désignation	**Description**	**Type**
matricule	Le numéro matricule de l'agent	Alphanumérique
nom	Nom de l'agent	Alphanumérique
prénom	Le ou les prénom(s) de l'agent	Alphanumérique

Classe : Service		
Attributs		
Désignation	**Description**	**Type**
codeService	Code identifiant un service	Alphanumérique
désignation	Le nom du service	Alphanumérique

Classe : Fournisseur		
Attributs		
Désignation	**Description**	**Type**
idFournisseur	Le numéro matricule de l'agent	numérique
nom	Nom du fournisseur	Alphanumérique
prénom	Le ou les prénom(s) du fournisseur	Alphanumérique
raisonSocial	Raison sociale de la société fournisseur	
typeFournisseur	Personne morale ou physique	

Classe : CompteAuxiliaire		
Attributs		
Désignation	**Description**	**Type**
codeCompte	Le code du compte	Alphanumérique
libellé	Désignation du compte	Alphanumérique

Classe : Constatation		
Attributs		
Désignation	**Description**	**Type**
matricule	Le numéro matricule de l'agent	Alphanumérique
nom	Nom de l'agent	Alphanumérique
prénom	Le ou les prénom(s) de l'agent	Alphanumérique

Classe : Constatation		
Attributs		
Désignation	**Description**	**Type**
matricule	Le numéro matricule de l'agent	Alphanumérique
nom	Nom de l'agent	Alphanumérique
prénom	Le ou les prénom(s) de l'agent	Alphanumérique

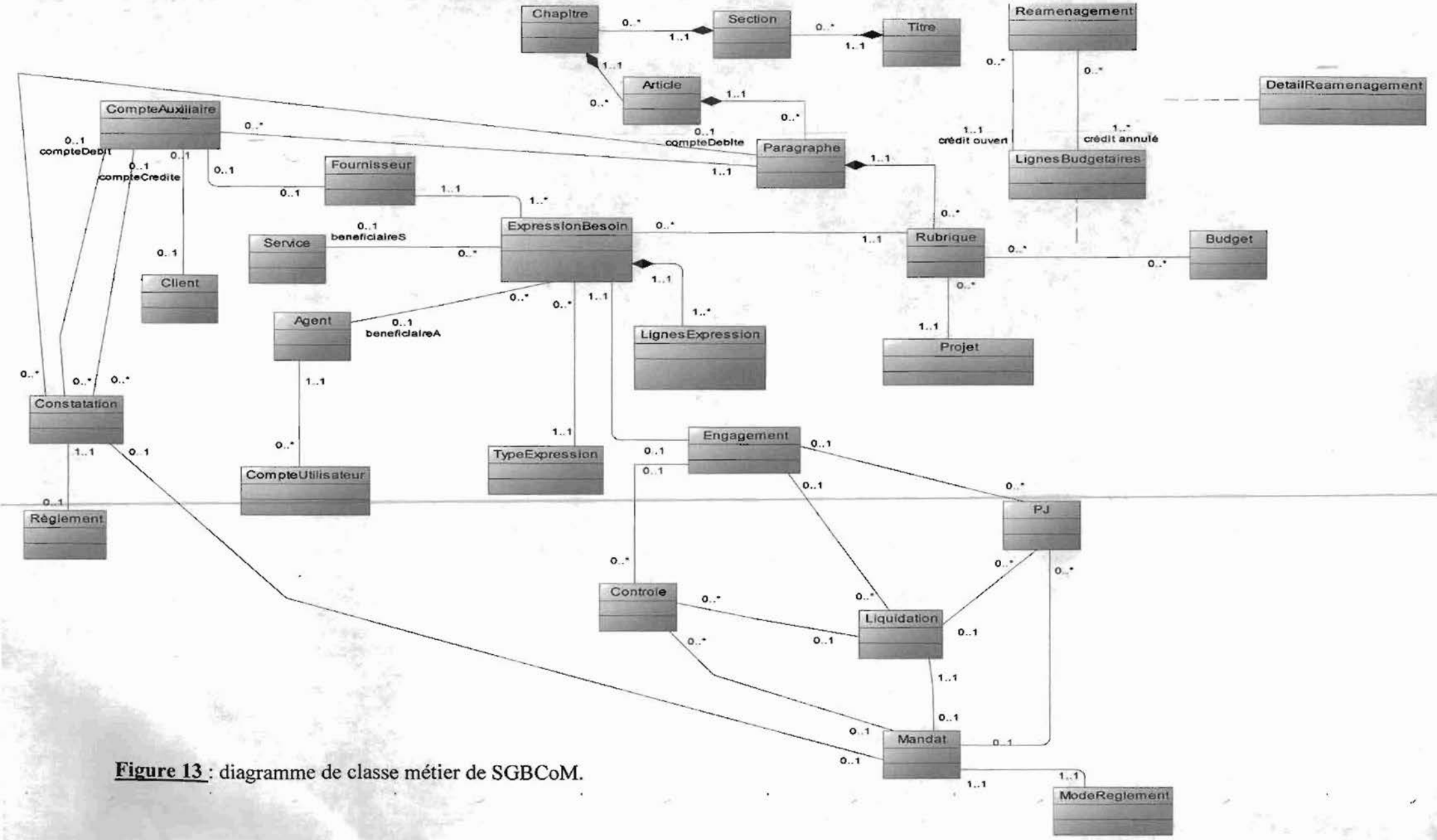


Figure 13 : diagramme de classe métier de SGBCoM.

II.2- Organisation des classes

L'organisation vise à regrouper les classes en catégorie. Une catégorie consiste en un regroupement logique de classes à forte cohérence interne et faible couplage externe.

On a d'abord regroupé les classes par catégorie, avant de montrer les liens entre ces catégories.

Tableau VI : Catégories de SGBCoM.

Catégories	Commentaires
Plan comptable et budgétaire	Regroupe tous les éléments pour la mise en place du plan comptable et budgétaire.
Gestion budget	Englobe tous les objets permettant de créer le budget et de l'exécuter.
Expressions des besoins	Modélise tous les éléments associés à l'établissement des expressions de besoins.
Profils utilisateurs	Représente tous les objets nécessaires à la gestion des droits d'accès au système.
Comptabilité	

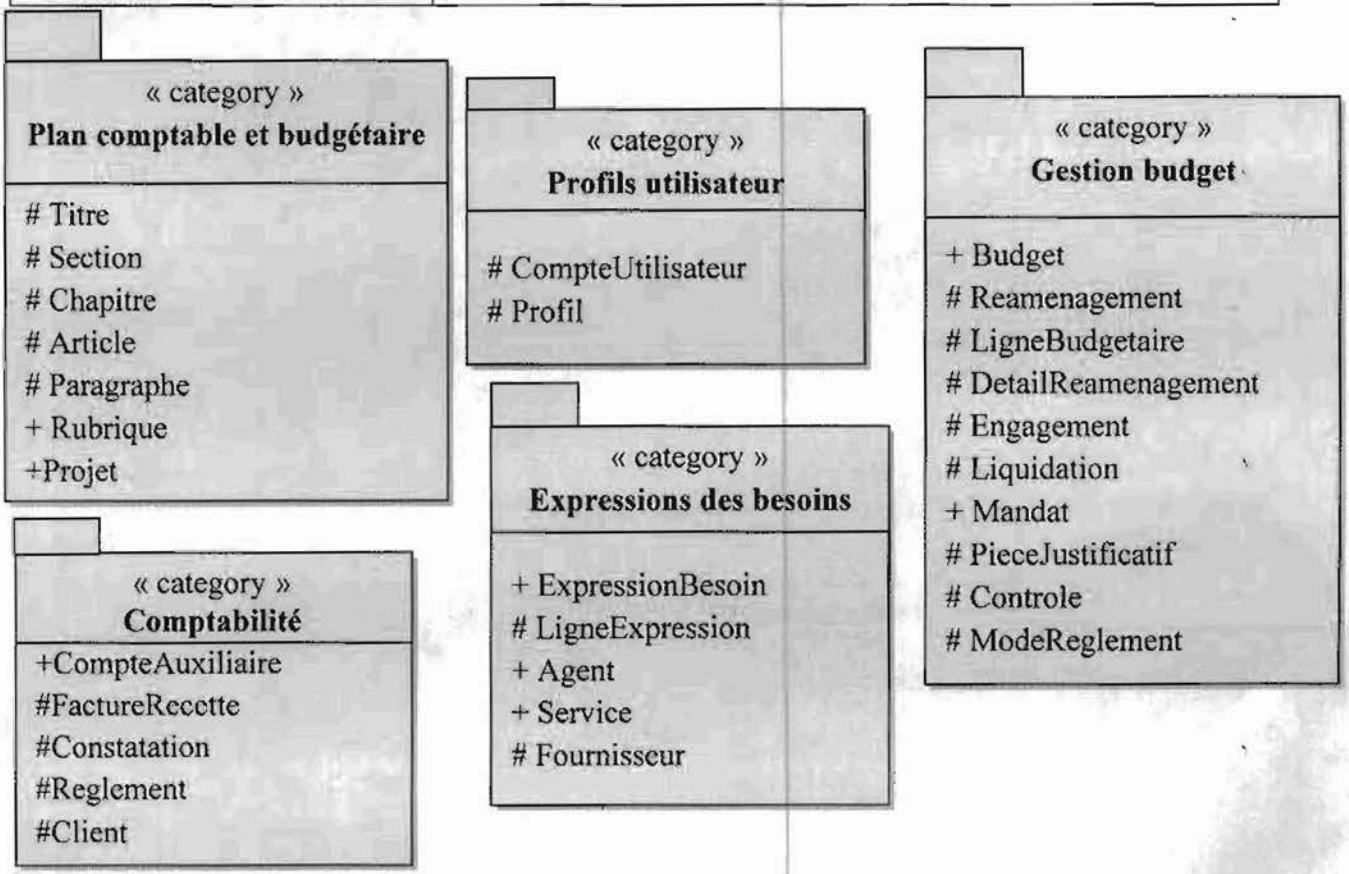


Figure 14 : Les catégories et leurs classes.

: l'objet est accessible seulement à l'intérieure de sa catégorie.

+ : l'objet est accessible par une autre catégorie.

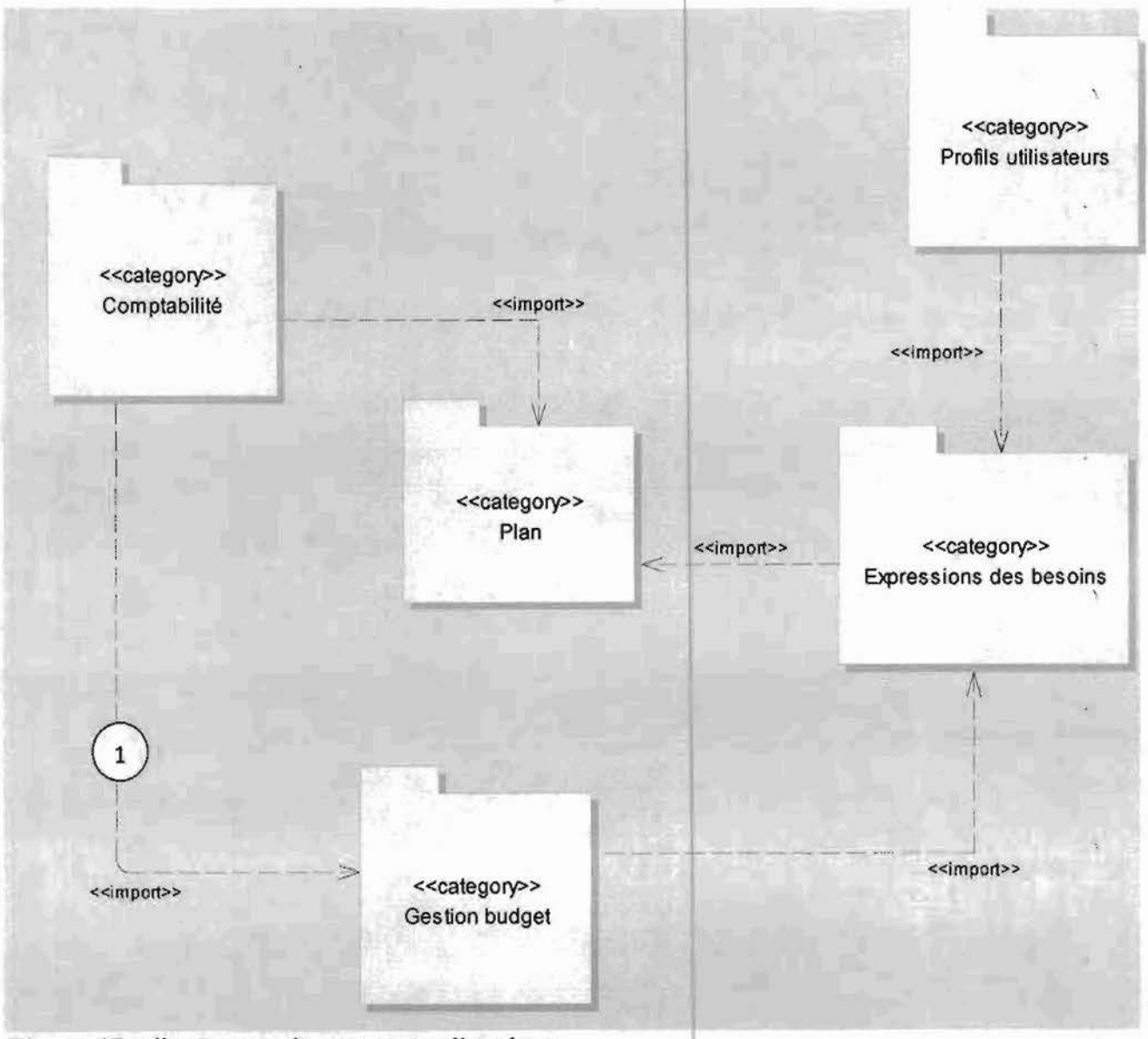


Figure 15 : diagramme de paquetage d'analyse.

(1) : Ceci indique que certains objets de la catégorie "Comptabilité" ont besoin d'autres objets de la catégorie "Gestion budget" pour s'exécuter.

II.3- Développement du cycle de vie des objets

Il s'agit de comprendre le cycle de vie d'un objet au fil de ses interactions avec les autres instances de classes, dans tous les cas possibles. Ceci est représenté par un diagramme d'états-transitions.

Il ne s'agit pas de faire le diagramme d'états-transitions pour toutes les classes, mais pour ce qui représente au moins trois états.

Etude de cas : classe "*Engagement*"

- **Identification des états**

  - **En création** : de la création d'une instance de la classe *Engagement* jusqu'au déclanchement de l'évènement validation.
  - **Validé** : la création de l'engagement s'est bien passé.
  - **Annulé** : tout engagement dont la liquidation n'a pas commencé, peut être annulé. Après un temps (ici 48 heurs) qui peut être défini par l'administrateur, l'engagement est supprimé totalement de la base.
  - **En contrôle** : l'état d'un engagement en cours de contrôle.
  - **Rejeté** : un engagement qui a été jugé non conforme aux règles.
  - **Approuvé** : un egagement jugé conforme aux règles.
  - **Liquidation partielle** : le montant engagé n'a pas été totalement liquidé.
  - **Liquidé** : montant engagé totalement liquidé.

- **Diagramme d'états-transitions**

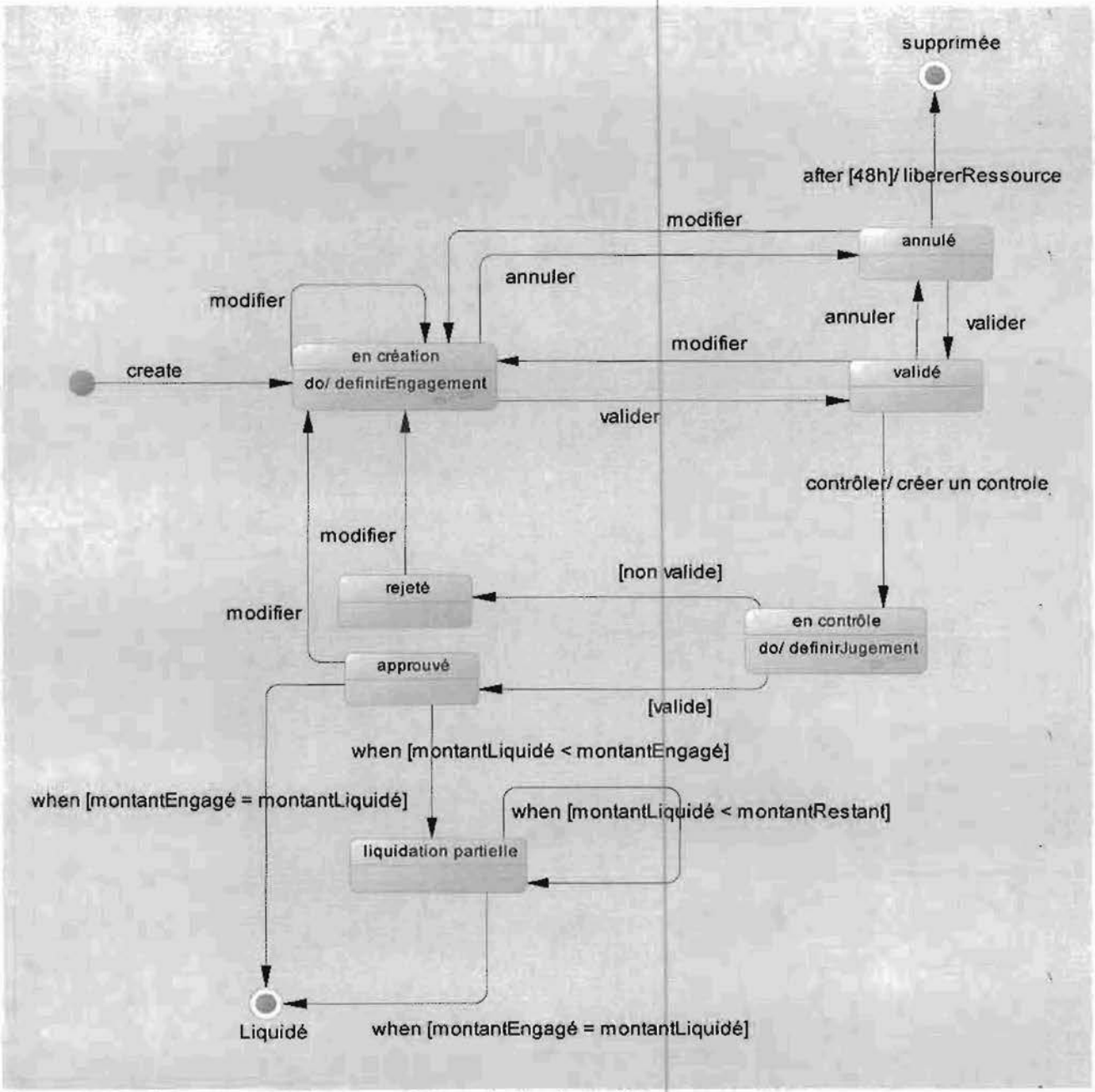


Figure 16 : Diagramme d'états-transitions de la classe "Engagement".

A l'issue de cette étude de cas les attributs suivants ont été ajoutés à la classe "Engagement" : En-cours, Rejeté, Liquidé et Annulé.

Etude de cas : classe "*Liquidation*"

- **Identification des états**

  - **En création** : de la création d'une instance de la classe *Liquidation* jusqu'au déclanchement de l'évènement validation.
  - **Validée** : la création de la liquidation s'est bien passée.
  - **Annulée** : toute liquidation dont le mandat de paiement n'a pas été établi, peut être annulé. Après un temps, la liquidation est supprimée totalement de la base.
  - **En contrôle** : l'état d'une liquidation en cours de contrôle.
  - **Rejeté** : une liquidation qui a été jugée non conforme aux règles.
  - **Approuvée** : une liquidation jugée conforme aux règles.
  - **Mandatée** : le mandat est établi.

- **Diagramme d'états-transitions**

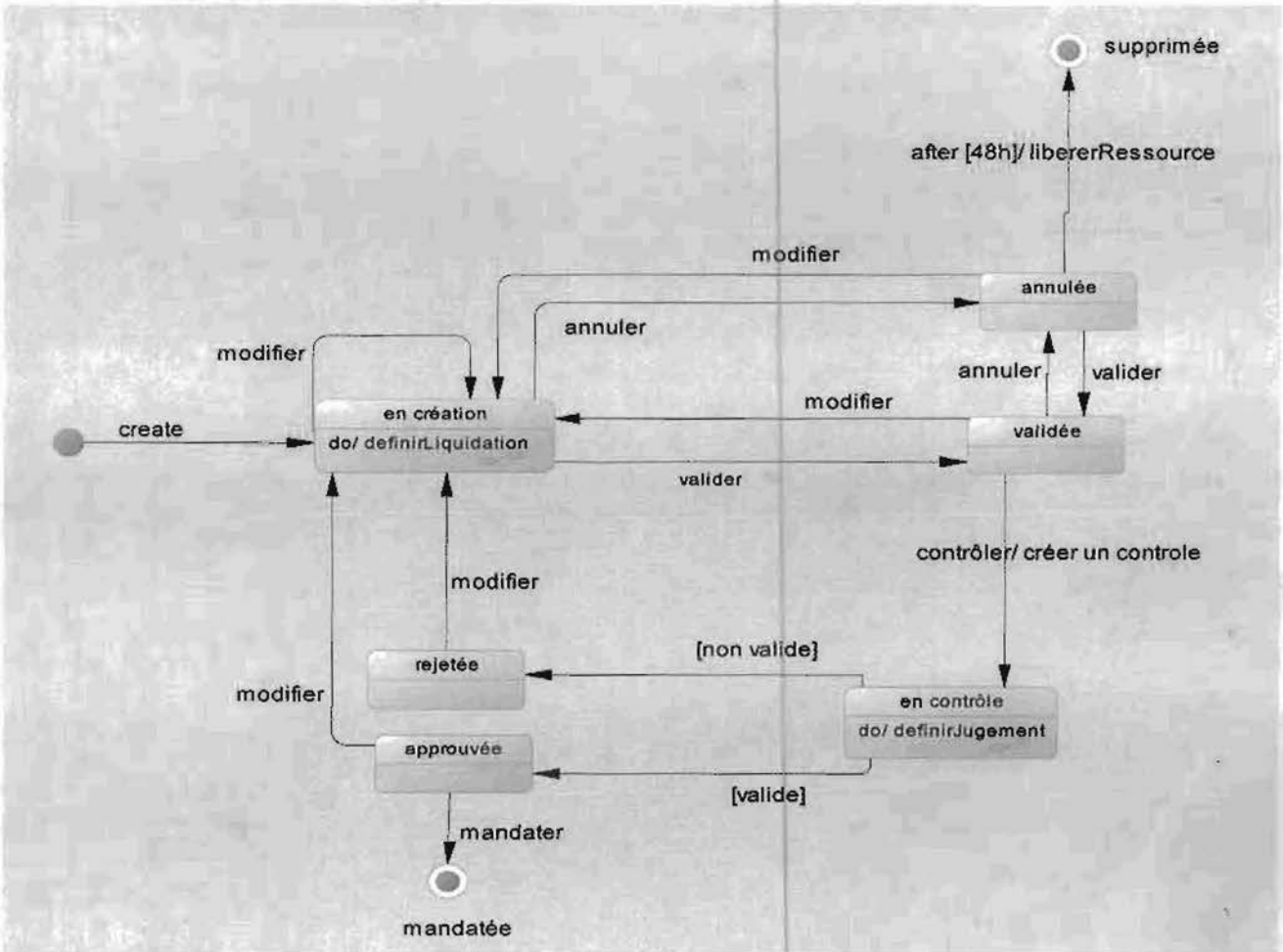


Figure 17 : diagramme d'états-transitions de la classe "Liquidation".

Le diagramme d'états-transitions pour la classe "Liquidation" a permis d'ajouter les attributs suivants à celle-ci : En-cours, Rejetée, Mandatée et Annulée.

II.4- Spécification des maquettes de cas d'utilisation

Cette partie pose les jalons de l'IHM de SGBCOM. Les maquettes ont été faites pour chaque cas d'utilisation et laisser à l'appréciation des utilisateurs. Ci-dessous, nous présentons deux (02) de ces maquettes.

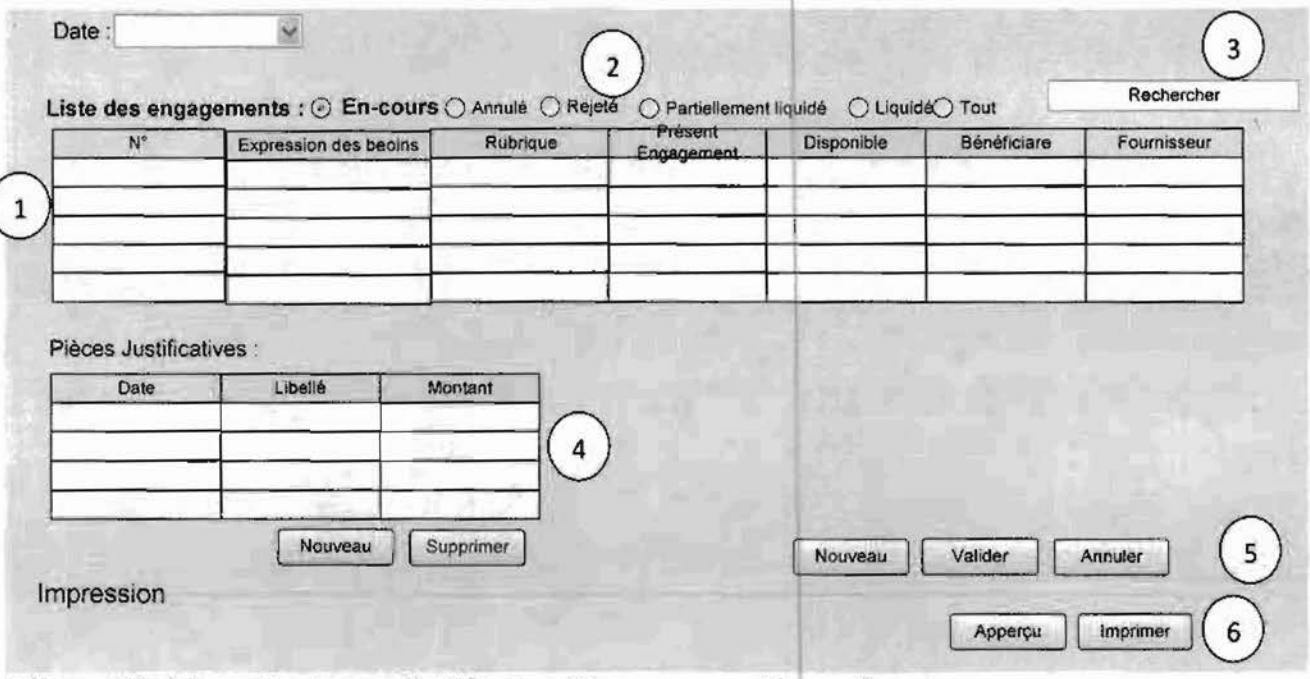


Figure 18 : Maquette du cas d'utilisation "Engager une dépense".

(1) : représente la liste des engagements. Cette liste peut être affichée selon certains critères (2) qui représentent les états des engagements.

(3) : on peut chercher un engagement en introduisant son numéro ou sa date de création.

(4) : représente les pièces justifiant l'engagement.

(5) : concerne les actions qu'on peut appliquer à un engagement. "Nouveau", pour ajouter un nouvel engagement ; "Valider", pour enregistrer les paramètres d'un engagement ; "Annuler", pour annuler un engagement choisi dans la liste.

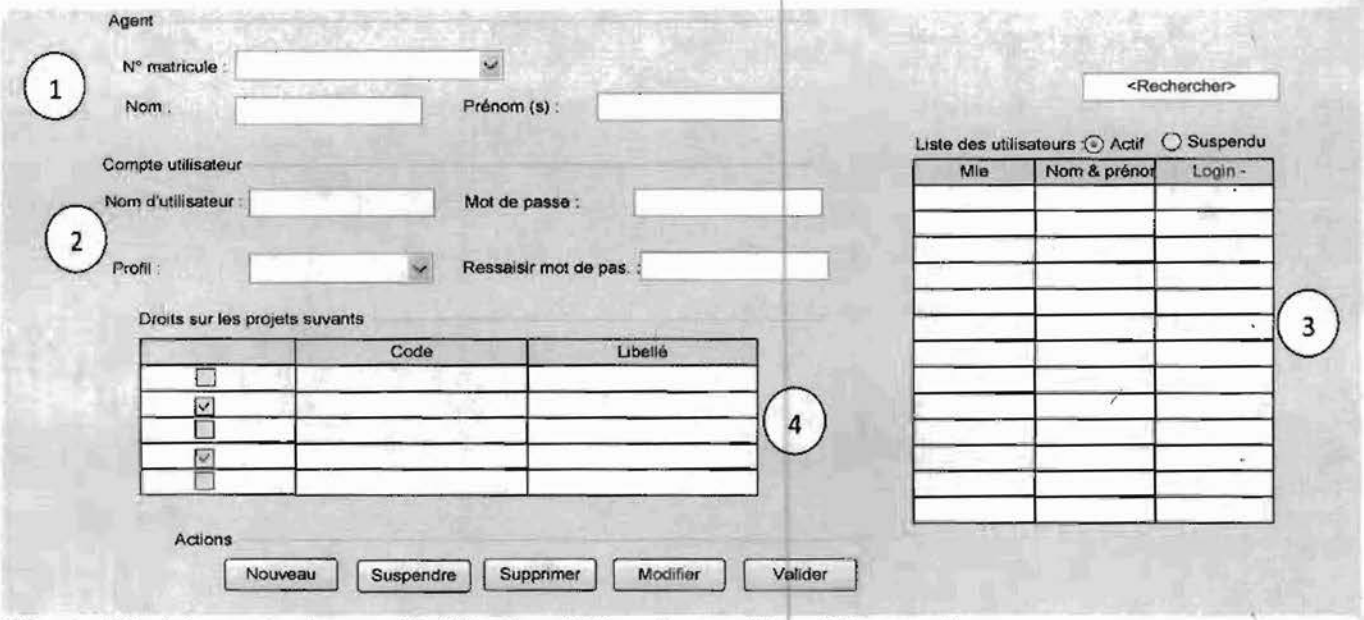


Figure 19 : Maquette du cas d'utilisation "Gérer les profils utilisateurs".

(1) : les paramètres de l'agent propriétaire du compte utilisateur.

(2) : les paramètres du compte.

(3) : la liste des comptes utilisateurs. Cette liste peut être affichée selon l'état des comptes. Pour modifier, supprimer ou suspendre un compte, l'utilisateur le choisit dans cette liste.

(4) : les projets dont l'utilisateur a le droit de faire des opérations sur leurs lignes budgétaires.

CONCLUSION

Cette phase de spécification nous a permis de recenser les fonctionnalités de SGBCoM à travers les cas d'utilisation. Les différentes descriptions de ces cas d'utilisation ont abouti à un diagramme de classes métier du système futur. Les maquettes des cas d'utilisation ont aussi été construites et présentées aux utilisateurs, afin d'être en adéquation avec leurs aspirations. Ces modèles obtenus sont enrichis dans la partie conception pour les adapter à la programmation.

CHAPITRE 4 :
CONCEPTION

INTRODUCTION

La spécification a permis d'avoir un modèle d'analyse, ce modèle est repris en *conception* dans le but de faciliter le passage à la programmation. Cette phase de *conception* est subdivisée en deux grandes parties : la *conception architecturale* et la *conception détaillée*.

I- CONCEPTION ARCHITECTURALE

La *conception architecturale* consiste à la spécification des différentes couches de SGBCoM. Cette partie présente d'abord la technologie CORBA, puis la répartition en couches du système. Après, une découverte des méthodes est faite afin d'enrichir le diagramme de classes de la phase de *spécification*.

I.1- Présentation de CORBA

CORBA est une norme, un ensemble de spécifications et de recommandations rédigées par un groupe de travail nommé OMG (*Object Management Group*), auquel appartiennent la plupart des acteurs informatiques majeurs.

Ce groupe de travail a été créé en 1989 par huit sociétés : 3Com, American Airlines, Canon, Data General, Hewlett-Packard, Philips Telecommunications, Sun Microsystems et Unisys. Le rôle de ce consortium est de standardiser et de promouvoir sa vision de l'architecture distribuée. Les standards réalisés permettent de développer des applications réparties entre plusieurs ordinateurs, totalement interopérables en termes de plates-formes matérielles, de systèmes d'exploitation ou encore de langages de développement.

I.1.1- Modèle objet client-serveur

Le bus CORBA propose un modèle orienté objet client-serveur d'abstraction et de coopération entre les applications réparties. Chaque application peut exporter certaines de ses fonctionnalités (*services*) sous la forme d'objets CORBA : c'est la composante d'abstraction de ce modèle. Les

interactions entre les applications sont alors matérialisées par des invocations à distance des méthodes des objets : c'est la partie coopération. La notion client-serveur intervient uniquement lors de l'utilisation d'un objet : l'application implantant l'objet est le serveur, l'application utilisant l'objet est le client.

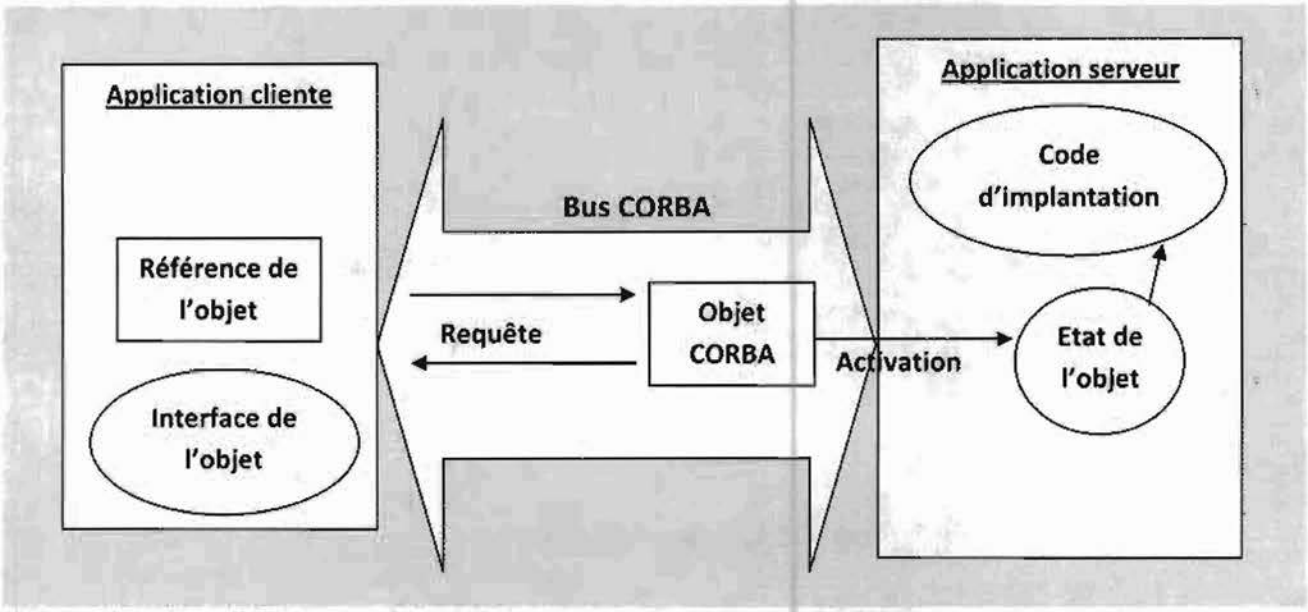


Figure 20 : Les différentes notions intervenant dans le modèle client-serveur CORBA.

- L'**application cliente** est un programme qui invoque les méthodes des objets à travers le bus CORBA.
- La **référence de l'objet** est une structure désignant l'objet CORBA et contenant l'information nécessaire pour le localiser sur le bus.
- L'**interface de l'objet** est le type abstrait de l'objet CORBA définissant ses opérations et attributs. Celle-ci se définit par l'intermédiaire du langage OMG-IDL.
- La **requête** est le mécanisme d'invocation d'une opération ou de l'objet.
- Le **bus CORBA** achemine les requêtes de l'application cliente vers l'objet en masquant tous les problèmes d'hétérogénéité (langages, systèmes d'exploitation, matériels, réseaux).
- L'**objet CORBA** est le composant logiciel cible. C'est une entité virtuelle gérée par le bus CORBA.
- L'**activation** est le processus d'association d'un objet d'implantation à un objet CORBA.
- L'objet d'implantation est l'entité codant l'objet CORBA à un instant donné et gérant un **état de l'objet** temporaire.

- Le **code d'implantation** regroupe les traitements associés à l'implantation des opérations de l'objet CORBA. Cela peut être par exemple une classe Java.
- L'**application serveur** est la structure d'accueil des objets d'implantation et des exécutions des opérations.

I.1.2- OMG-IDL (Interface Definition Language)

Ci-dessus, nous avons démontré que pour offrir ses services une application serveur utilise un objet CORBA. Cet objet est défini à l'aide du langage IDL, qui est un langage de définition d'interface orienté objet. Ce langage sert à définir, sous forme de contrats IDL, la coopération entre le serveur et les clients. Il permet de séparer l'interface de l'implantation des objets et de masquer les divers problèmes liés à l'interopérabilité et l'hétérogénéité. Un contrat IDL spécifie les types manipulés par un ensemble d'applications réparties et les types de données échangés entre les objets. Il isole ainsi les clients et les serveurs de l'infrastructure logicielle et matérielle, et les mettent en relation à travers le bus CORBA.

L'IDL ne permet de définir que l'interface de l'objet. Pour implémenter cette interface, il faut traduire cette définition d'IDL dans un langage cible (ou de programmation). Dans le paragraphe suivant, nous verrons le langage de programmation pour SGBCoM qui est Java.

I.2- Langage d'implantation

OMG a défini la liaison avec plusieurs langages de programmation parmi lesquels, on a Java.

Ce dernier est un langage de programmation orienté objet créé par la société *Sun Microsystems*. Les logiciels écrits avec Java sont facilement portables sur plusieurs systèmes d'exploitation tels que UNIX et Windows. Cette portabilité est assurée par la machine virtuelle.

Sun fournit un grand nombre d'API (*interface de programmation*) qui permettent l'utilisation de Java de façon très diversifiée. Parmi ces API, on distingue :

- *JPA* (Java Persistence API) qui est une spécification de mapping objet-relationnel ;

- *JDBC* (Java DataBase Connectivity) qui est un middleware permettant d'accéder à une base de données relationnelle.

I.3- Architecture globale de SGBCoM

Les parties précédentes ont permis de se fixer sur les technologies à utiliser :

- PostgreSQL comme SGDBR ;
- CORBA pour le middleware client/serveur ;
- Java comme langage de développement.

Au regard des technologies à utiliser, l'architecture logicielle suivante peut être proposées.

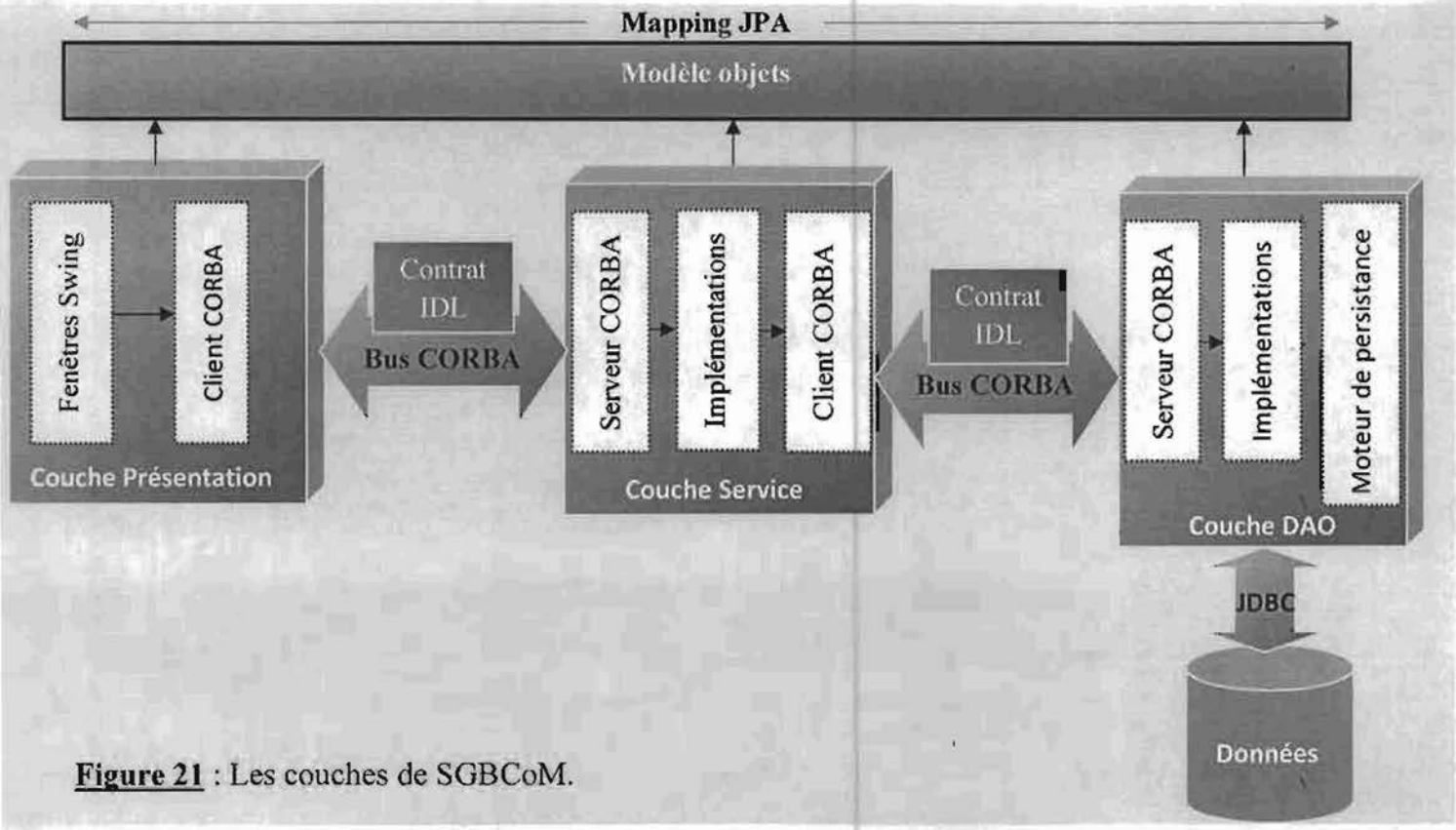


Figure 21 : Les couches de SGBCoM.

La mise en œuvre d'une telle architecture en CORBA se fait par l'utilisation des *contrats IDL*. Entre deux couches un contrat IDL est défini ; la couche supérieure, par l'intermédiaire du bus CORBA, invoque les méthodes implémentées sur la couche inférieure.

I.3.1- Couche modèle objets

Cette couche, comme son nom l'indique, représente un modèle d'objets et non un ensemble de services. Les éléments de cette couche sont tous de simples objets composés de champs, de *getters* et de *setters*. Les getters et setters sont des méthodes permettant, respectivement, d'accéder à la valeur d'un attribut et de la modifier.

Ce sont les objets de cette couche qui sont persistés en base de données par un moteur de persistance. Pour se faire, ces objets sont mappés par *JPA*. Dans notre cas, chaque objet persisté est l'image d'une ligne de la base de données relationnelle. Le moteur de persistance pour cette architecture est *Hibernate*.

I.3.2- Couche DAO (Data Access Object)

La *couche d'accès aux objets* est en charge de la gestion des relations avec la source de données relationnelle. Elle offre, à travers le *serveur CORBA*, des services d'accès, de recherche, de création, de mise à jour et de suppression de données. Cette couche s'appuie sur une interface standard : *JPA*, qui permet d'encapsuler l'implémentation réelle du moteur de persistance.

I.3.3- Couche Présentation

Cette couche est composée d'une partie visible, les *fenêtres Swing*, qui permet à l'utilisateur d'interagir avec le système. Elle comprend aussi un *client CORBA* qui a pour rôle d'invoquer les méthodes de la *couche Service*.

I.3.4- Couche Service

La *couche Service* implémente l'ensemble de la *logique métier* de l'application, indifféremment de la source de données utilisées et de la présentation. Elle s'appuie sur les couches DAO et modèle objets pour effectuer des opérations sur des objets persistés et leur appliquer ensuite des traitements métier.

I.3.5- Couche Données

Cette dernière contient les données sauvegardées physiquement sur le support de stockage de manière relationnelle. Les objets métiers sont transformés en données relationnelles par le moteur de persistance. Les données ainsi transformées sont envoyées au SGBD par l'intermédiaire du middleware JDBC.

I.3.6- Framework JPA/Hibernate

Hibernate est un ORM (Object Relational Mapping), un outil qui fait le pont entre le monde relationnel des bases de données et celui des objets manipulés par Java. Le développeur de la couche *DAO* ne voit plus la couche *JDBC* ni les tables de la base de données dont il veut exploiter le contenu. Il ne voit que l'image objet de la base de données, image objet fournie par Hibernate.

Devant le succès des produits ORM, Sun le créateur de Java, a décidé de standardiser une couche ORM via une spécification appelée *JPA* (Java Persistence API) apparue en même temps que *Java 5*. Cette spécification a été implémentée par Hibernate. La couche *DAO* dialogue maintenant avec la spécification JPA, un ensemble d'interfaces. Ce dernier rend l'utilisation d'Hibernate beaucoup plus facile. Le développeur y gagne également en standardisation.

I.4- Développement dynamique

Il s'agit dans cette partie d'enrichir le diagramme de classes métier établi au niveau de la phase de *spécification*. A cette même phase, un scénario représente une séquence d'interaction entre le système et ses acteurs. Maintenant que l'architecture logicielle est établie, le système est remplacé, dans cette partie, par une collaboration d'objets. Cet éclatement du système en collaboration d'objets, permet de découvrir d'autres objets différents de l'étape de la spécification. Aussi, il sert à trouver les méthodes de classes.

Pour modéliser les interactions entre les instances de classes, nous disposons de deux diagrammes : le diagramme de communication et le diagramme de séquence. Contrairement au

premier, le second permet de représenter plusieurs scénarios d'un cas d'utilisation et d'appréhender clairement la chronologie d'envoi des messages.

Afin de mettre en œuvre plusieurs scénarios d'un cas d'utilisation établis en phase de description textuelle, le diagramme de séquence est approprié.

I.4.1- Découverte des méthodes

Une méthode est un *service* qu'un objet met à la disposition des autres pour communiquer avec lui. Afin de mettre en lumière ces *services*, des scénarios sélectionnés, parmi les enchainements d'un cas d'utilisation de la phase de spécification, sont décrits par le diagramme de séquence. Un scénario décrit une exécution particulière d'un cas d'utilisation. On peut distinguer plusieurs types de scénarios :

- **Nominaux** : ils réalisent les post conditions du cas d'utilisation, d'une façon naturelle et fréquente ;
- **Alternatifs** : ils remplissent les post conditions du cas d'utilisation, mais en empruntant des voies détournées ou rares ;
- **D'exception** : ne réalisent pas les post conditions du cas d'utilisation.

Etude de cas : scénarios de "*Engager une dépense*" (ED)

- **Choix des scénarios**
  - Scénarios nominaux
    - ✓ ED_N1 : création d'un engagement (création d'un nouvel engagement à sa validation).
    - ✓ ED_N2 : contrôler un engagement.
    - ✓ ED_N3 : annulation d'un engagement.
  - Scénario d'exception
    - ✓ ED_E1 : non validation de la création d'un engagement pour cause d'expression des besoins non validée.
    - ✓ ED_E2 : non validation de la création d'un engagement pour cause d'expression déjà engagée.

✓ ED_E3 : non validation de l'annulation d'un engagement du fait que sa liquidation a été entamée.

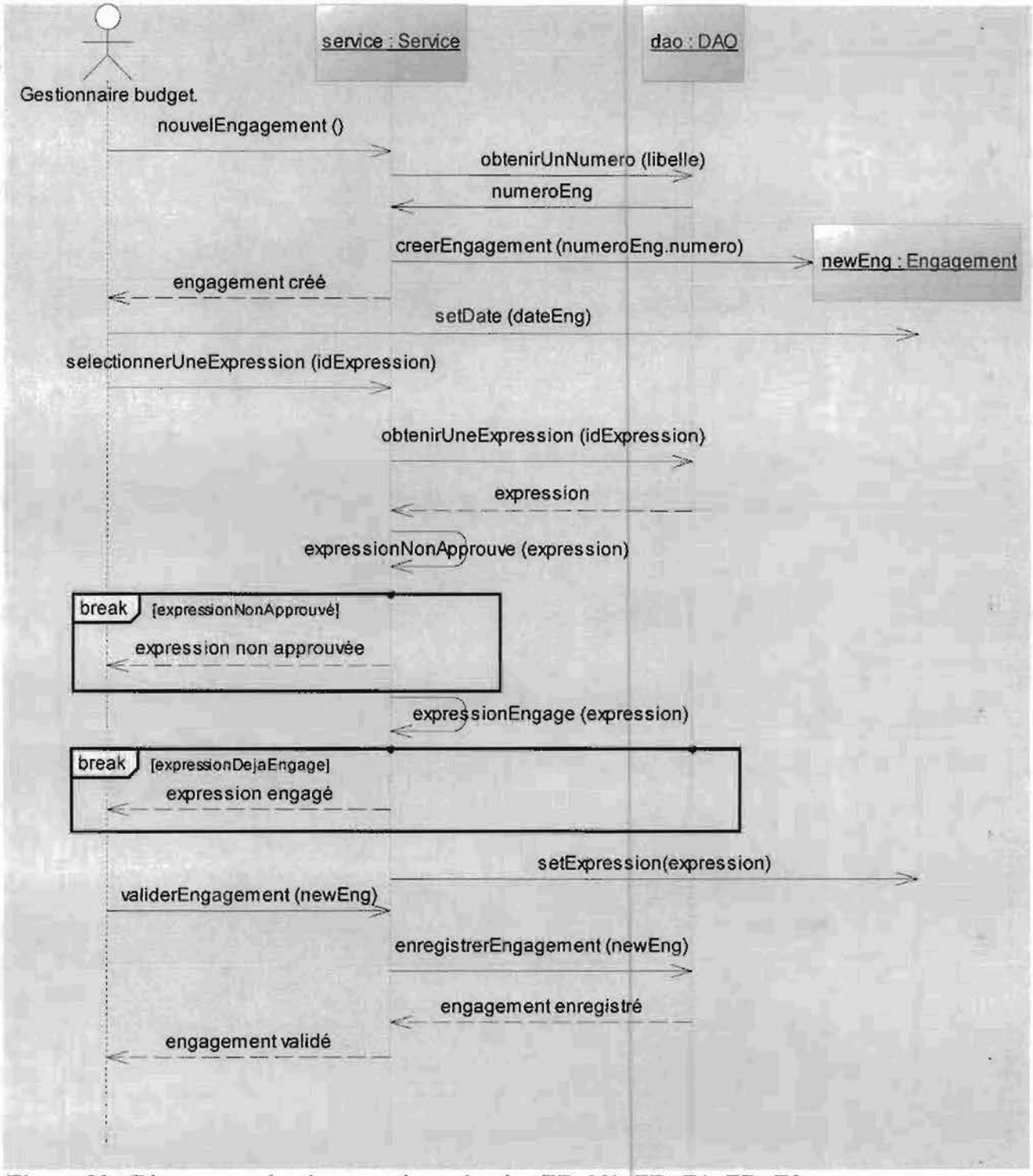


Figure 22 : Diagramme de séquence des scénarios ED_N1, ED_E1, ED_E2

Etude de cas : scénarios de "*Mettre à jour les codes des projets*" (MP)

- **Choix des scénarios**
  - Scénarios nominaux
    - ✓ MP_N1 : création d'un projet.
    - ✓ MP_N2 : suppression d'un projet.
  - Scénario d'exception
    - ✓ MP_E1 : non validation de la création pour cause : code déjà attribué à un autre projet.
    - ✓ MP_E2 : non validation de la création pour cause : doublon de libellé.

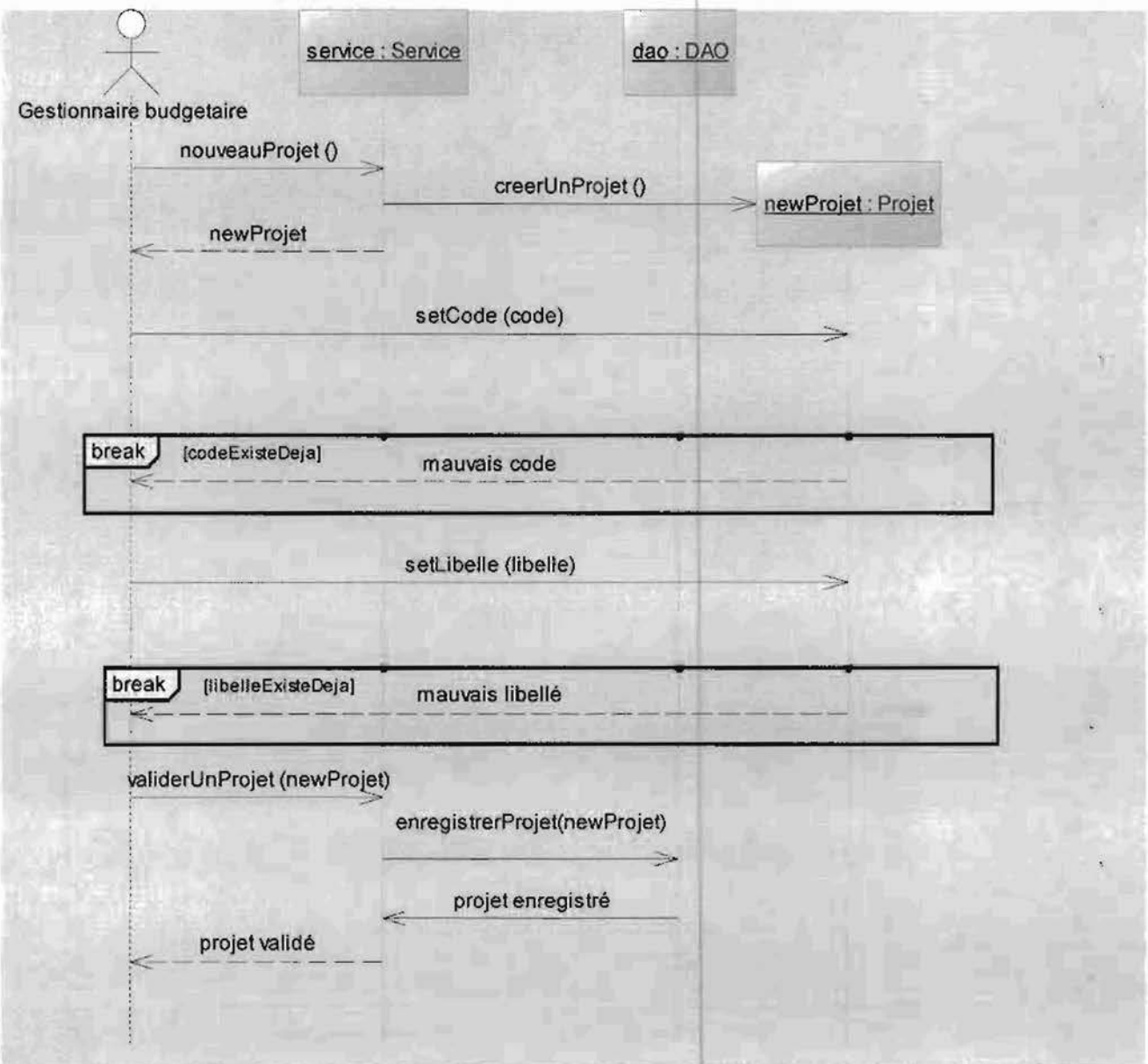


Figure 23 : Diagramme de séquence des scénarios MP_N1, MP_E1, MP_E2

Etude de cas : scénarios de "*Gérer les profils utilisateurs*" (GP)

- **Choix des scénarios**
  - Scénarios nominaux
    - ✓ GP_N1 : création d'un utilisateur.
    - ✓ GP_N2 : suspension d'un utilisateur.
    - ✓ GP_N3 : suppression d'un utilisateur.
  - Scénario d'exception
    - ✓ GP_E1 : non validation de la création pour cause : login déjà utilisé.

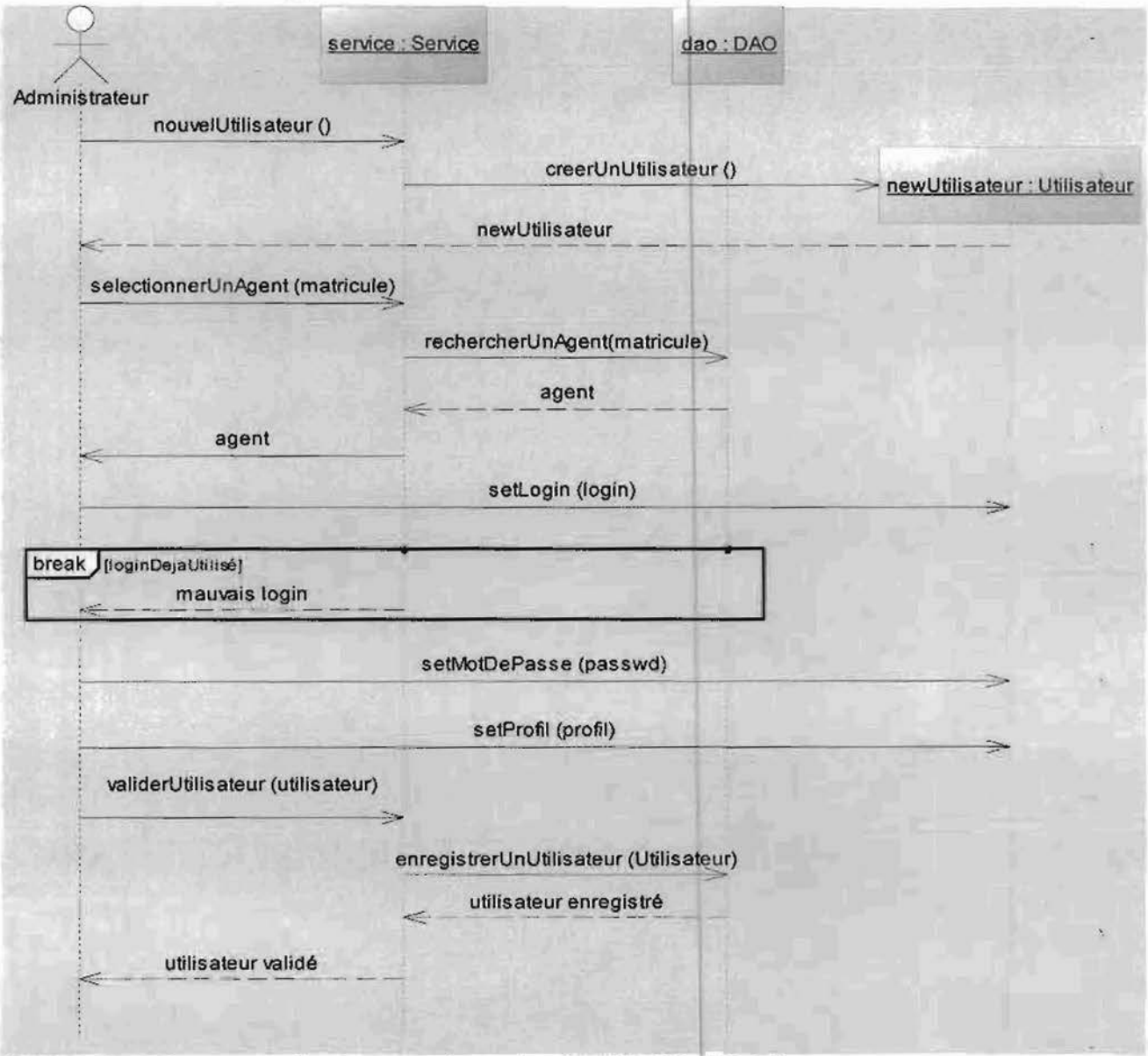


Figure 24 : Diagramme de séquence des scénarios GP_N1, GP_E1

Etude de cas : scénarios de "*Consulter la situation budgétaire*" (CB)

- **Choix des scénarios**
  - Scénarios nominaux
    - ✓ CB_N1 : consulter situation d'un projet.
  - Scénario d'exception : néant.

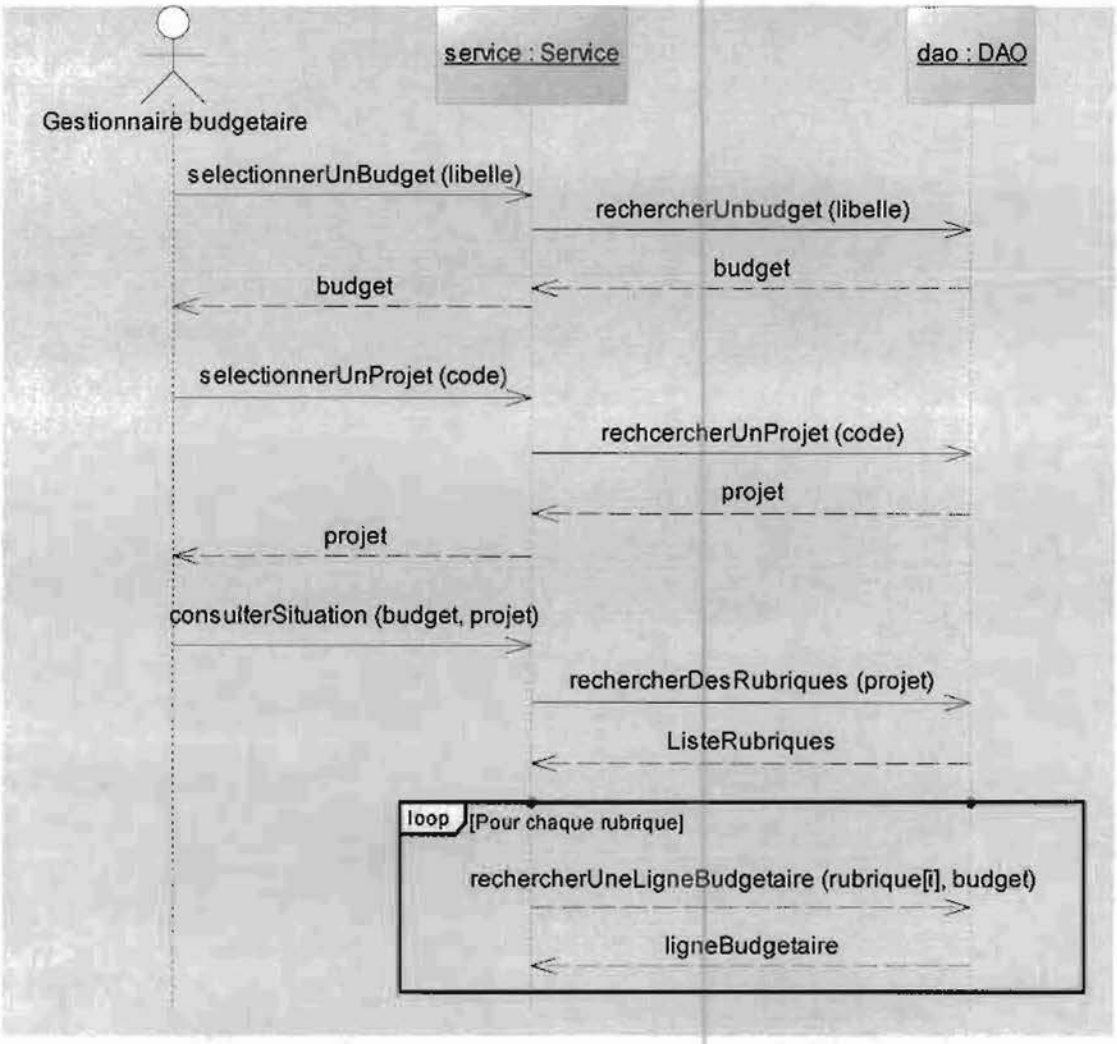


Figure 25 : Diagramme de séquence du scénario CB_N1.

Dans presque tous les diagrammes présentés ci-dessus, des objets génériques ont émergé :

- Service (ci-après ServiceMetier) : qui représente toutes les méthodes métier des objets de la *couche modèle objets*.
- DAO : regroupant les méthodes d'accès aux objets.

Après la description des scénarios, les méthodes qui peuvent être retenues pour chaque *classe* de la *couche modèle objets*, sont les *getters* et *setters*.

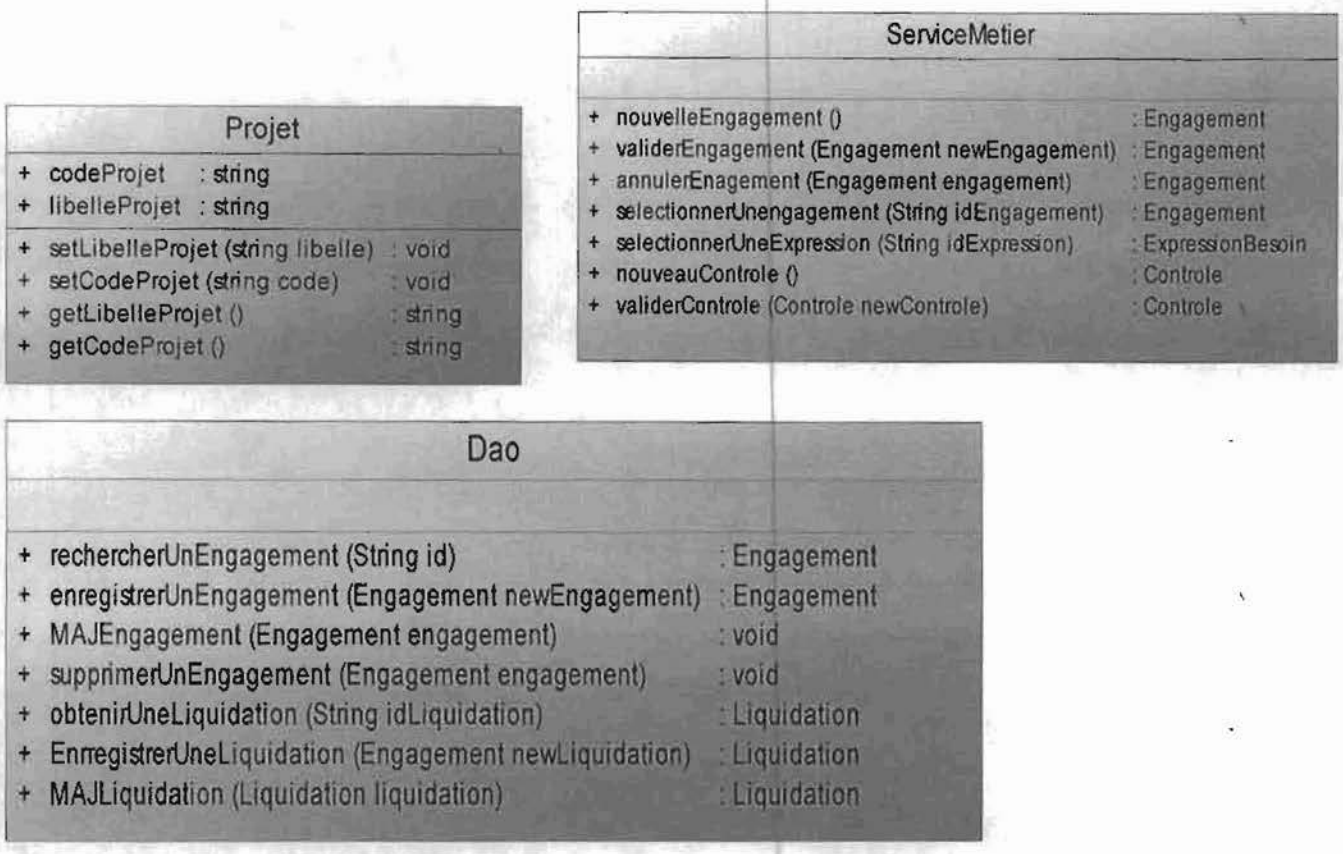


Figure 26 : Les objets avec quelques unes de leurs méthodes.

I.4.2- Réorganisation des classes

Après avoir définie les couches de SGBCoM, les objets de ce système peuvent être restructurés en catégories selon ces mêmes couches de la manière suivante :

Tableau VII : Réorganisation des classes de SGBCoM.

Catégories	Commentaires
Model	Regroupe toutes les sous-catégories définies en phase de spécification.
Présentation	Regroupe les objets de fenêtres (IHM)
Service	Contient l'objet ServiceMetier
Dao	Contient l'objet Dao

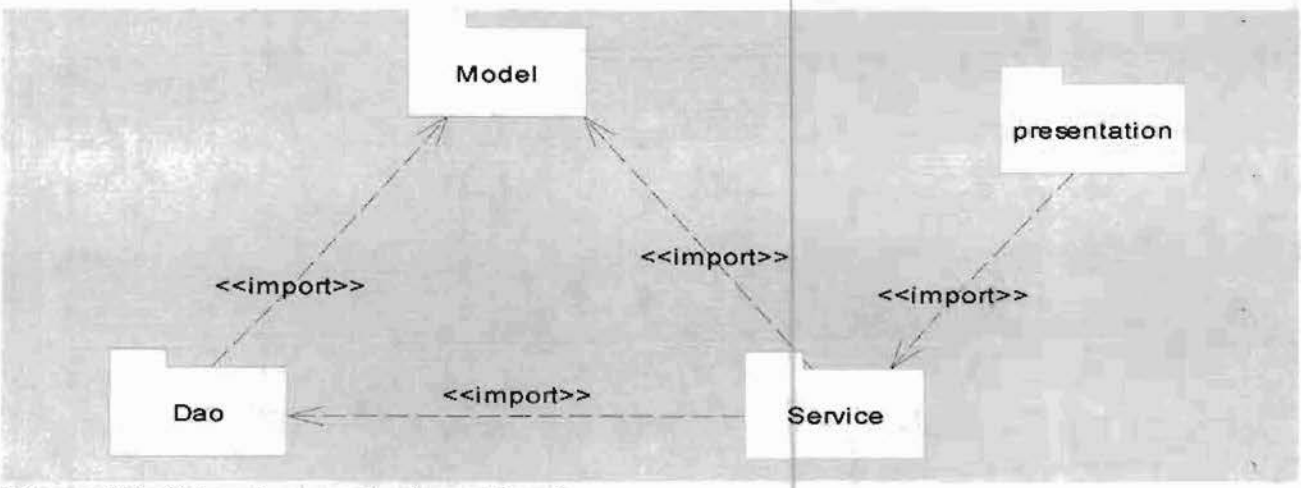


Figure 27 : Dépendance entre les catégories.

Ce diagramme montre les liens de dépendance entre les différentes catégories.

II- CONCEPTION DETAILLEE

A cette phase de notre cycle de développement, les classes, les interfaces, les tables et les méthodes sont construites et documentées de manière précise pour le codage de la solution. Pour se faire, les équivalences sont mises en lumière entre les technologies de développement et le langage de modélisation.

II.1- Conception des contrats OMG-IDL

La construction d'une application en CORBA commence par la définition des contrats d'IDL. A la différence de la couche de données et d'une partie de la couche de présentation, les objets des autres couches sont d'abord décrits en IDL avant d'être traduits en Java.

Cette partie prépare donc les objets des couches *Modèle objets*, *Service* et *DAO*, afin qu'ils puissent être traduits en IDL. Cette traduction est faite à l'aide de l'outil de modélisation, *Power AMC*.

II.1.1- Conception de la couche Modèle objets

Dans la phase précédente, cette couche est définie comme étant composée de simples objets Java constitués de champs, de getters et setters.

La phase de spécification a permis d'avoir un diagramme de classes métier (ou analyse). Cependant, les concepts utilisés par ce diagramme diffèrent de ceux de l'IDL. Pour pouvoir réaliser la génération de code en IDL, il faut donc utiliser les concepts de ce dernier.

II.1.1.1- Conception des classes

En IDL, le concept « *struct* » permet de représenter des "objets" qui pourront être traduits en des simples classes Java. Pour concevoir ce concept en UML, il suffit de créer une classe avec tous ses attributs et de lui attribuer le stéréotype « *CORBAStruct* ».

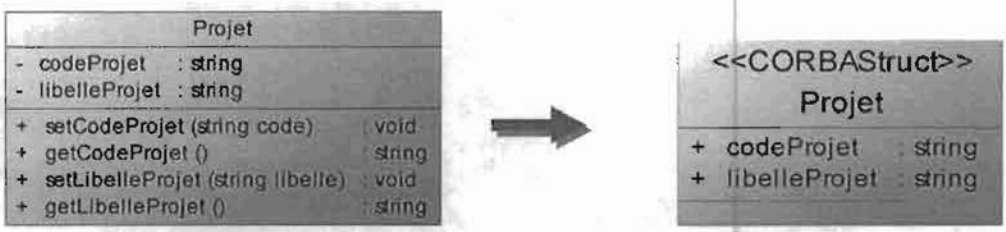


Figure 28 : Conception d'un objet IDL.

II.1.1.2- Conception des associations

L'association est un concept inconnu du langage OMG-IDL. Elle se transforme en un champ ou une séquence de champs suivant sa multiplicité.

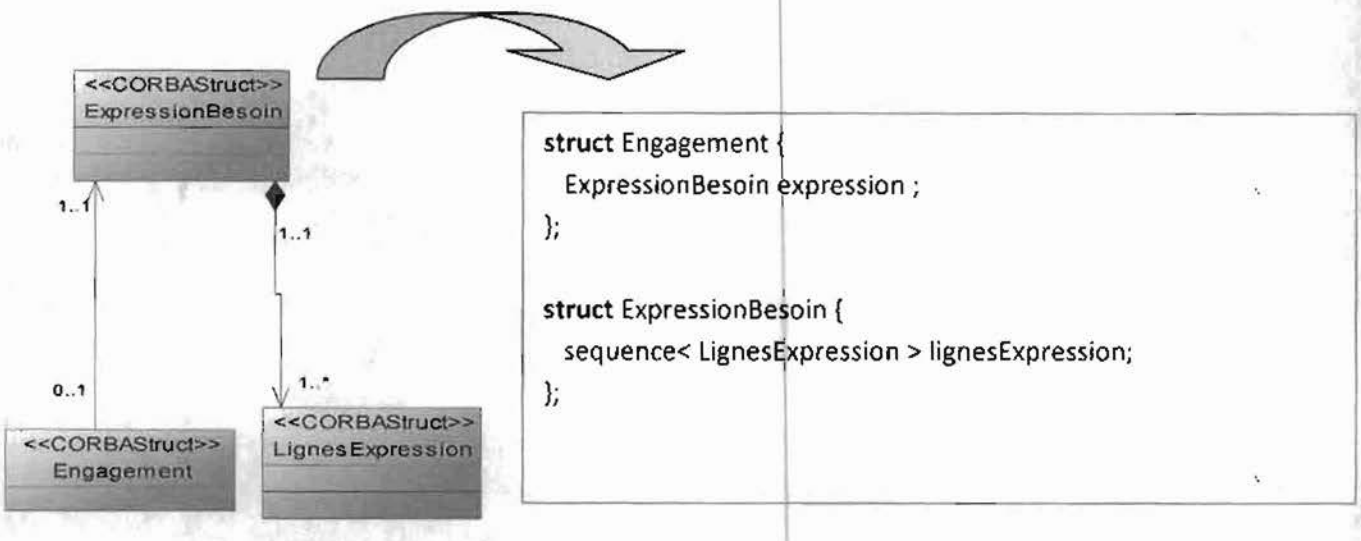


Figure 29 : Traduction des associations en IDL.

Le tableau VIII montre les règles de passage pour la conception des associations en IDL.

Tableau VIII : Quelques règles de passage des associations d'analyse en conception IDL

UML	Conception IDL
Classe B — 0..* — 1..1 — Classe A	<<CORBAStruct>> Classe A — 0..* — 1..1 — <<CORBAStruct>> Classe B
Classe B — 1..1 — 0..* — Classe A	<<CORBAStruct>> Classe A — 1..1 — 0..* — <<CORBAStruct>> Classe B
Classe B — 0..1 — 1..1 — Classe A	<<CORBAStruct>> Classe A — 0..1 — 1..1 — <<CORBAStruct>> Classe B
Classe B — 0..* — 0..* — Classe A ; Classe C	<<CORBAStruct>> Classe A ; <<CORBAStruct>> Classe B ; <<CORBAStruct>> Classe C (1..1, 0..*, 0..*, 1..1)

II.1.1.3- Conception des attributs

La conception des attributs consiste principalement à définir le type des attributs identifiés. A la différence du type *Date* identifié en phase de spécification, tous les autres types peuvent être référencés en OMG-IDL (présentation des types en *Annexe*). Il s'agit donc de définir une technique pour représenter ce type.

Ce dernier pourra être représenté par un type *string*. Une fois la projection du code IDL faitè en Java, les techniques de passage d'une donnée de type *Date* à celle de type *String* pourront être utilisés.

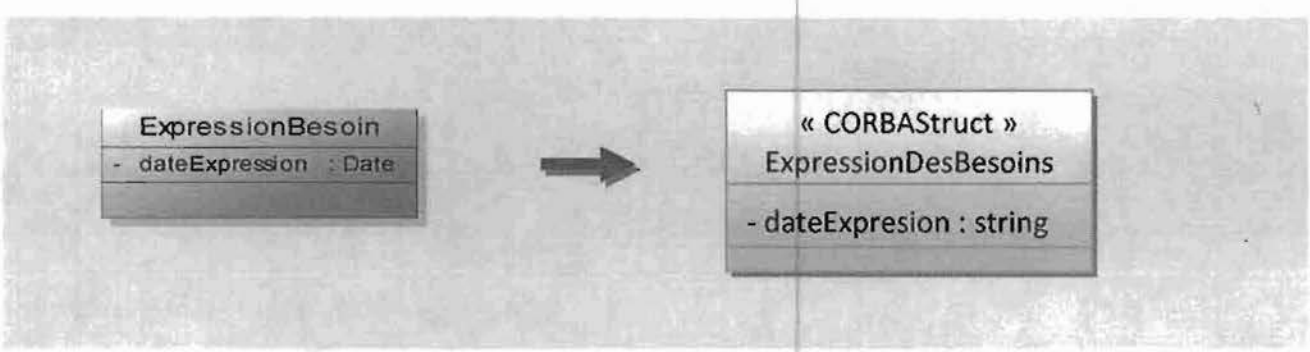


Figure 30 : Conception d'un attribut de type Date en OMG-IDL.

II.1.2- Conception de la couche Service et de la couche DAO

Ces deux (02) couches sont composées chacune d'un objet qui offre des services à leur couche immédiatement supérieure : couche Service pour *DAO et* couche présentation pour *Service*.

En CORBA, c'est le concept d'*interface* qui permet de spécifier un contrat entre un objet offrant des services et ses clients. Les objets *ServiceMetier* et *DAO* peuvent être représentés par une classe UML munie du stéréotype « *CORBAInterface* ».

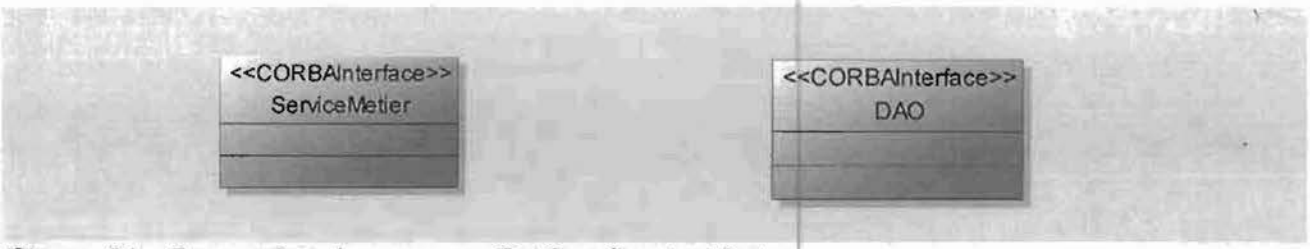


Figure 31 : Conception des contrats *DAO* et *ServiceMetier*.

II.2- Conception d'un serveur et d'un client CORBA

La communication entre les deux (02) objets, serveur et client, met en œuvre plusieurs d'autres objets représentés dans la *figure 32*.

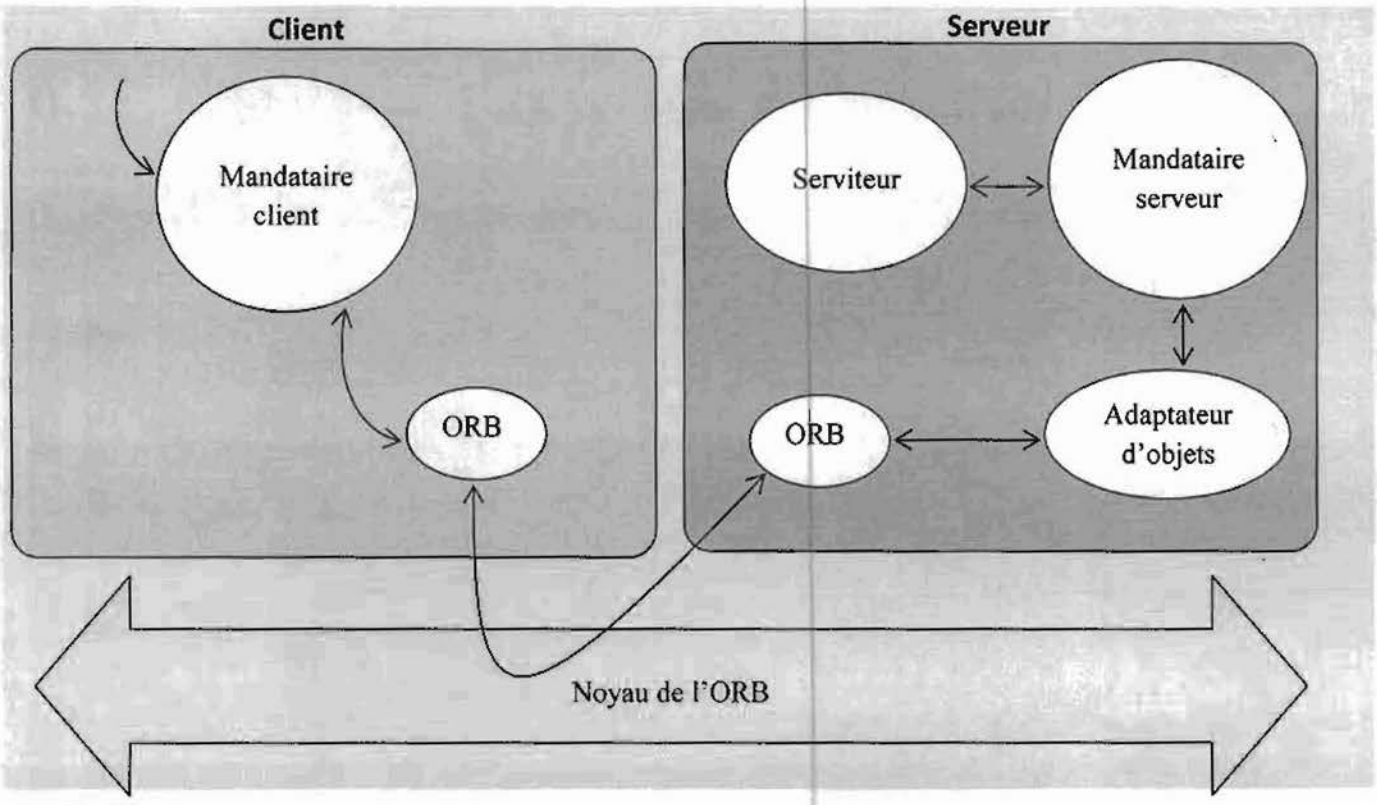


Figure 32 : Les objets impliqués dans une requête CORBA.

- **L'ORB (Object Request Broker)**

L'ORB est celui qui négocie les requêtes de l'objet client auprès du serveur. Il a pour rôle de :

- assurer la localisation du serveur qui héberge l'objet ;
- assurer l'activation du serveur (si besoin) ;
- attribuer la partie serveur de la référence de l'objet ;
- assurer l'acheminement de la requête vers le serveur, puis vers l'adaptateur d'objets.

- **L'adaptateur d'objets**

L'adaptateur d'objets est le mécanisme qui permet à l'implémentation d'un objet CORBA d'accéder aux services d'un ORB. Il fournit un environnement complet pour qu'une application serveur s'exécute. Il a pour rôle de :

- assurer la localisation du serviteur ;
- assurer l'activation du serviteur ;
- attribuer la partie locale de la référence d'objet ;
- transmettre la requête au mandataire du serveur.

On distingue plusieurs types d'adaptateurs tels que **POA** *(Portable Object Adapter)* qui est utilisé dans les cas où l'implantation de l'objet CORBA est assurée par des objets fournis par un langage de programmation.

- **Mandataire client**

Il est chargé d'acheminer les requêtes de l'utilisateur vers l'ORB.

- **Mandataire serveur**

Il est chargé de la liaison entre le serviteur et l'adaptateur d'objet.

- **Serviteur**

Une instance de la classe définie pour implémenter un objet CORBA est appelée *serviteur*. Ce dernier est relié à un adaptateur d'objets. Dans la *figure 33* ci-dessous, *DaoImpl* représente la classe implémentant l'objet CORBA.

L'adaptateur d'objet et l'ORB sont fournis par la norme CORBA. Puis, à part le serviteur tous les autres objets sont générés après traduction de l'IDL en Java. Pour un objet CORBA Dao, si l'adaptateur est un POA, la compilation en IDL donne les objets principaux suivants en Java :

- Dao : mandataire client ;
- DaoPOA : mandataire serviteur ;
- DaoOperations : interface des opérations de l'objet Dao.

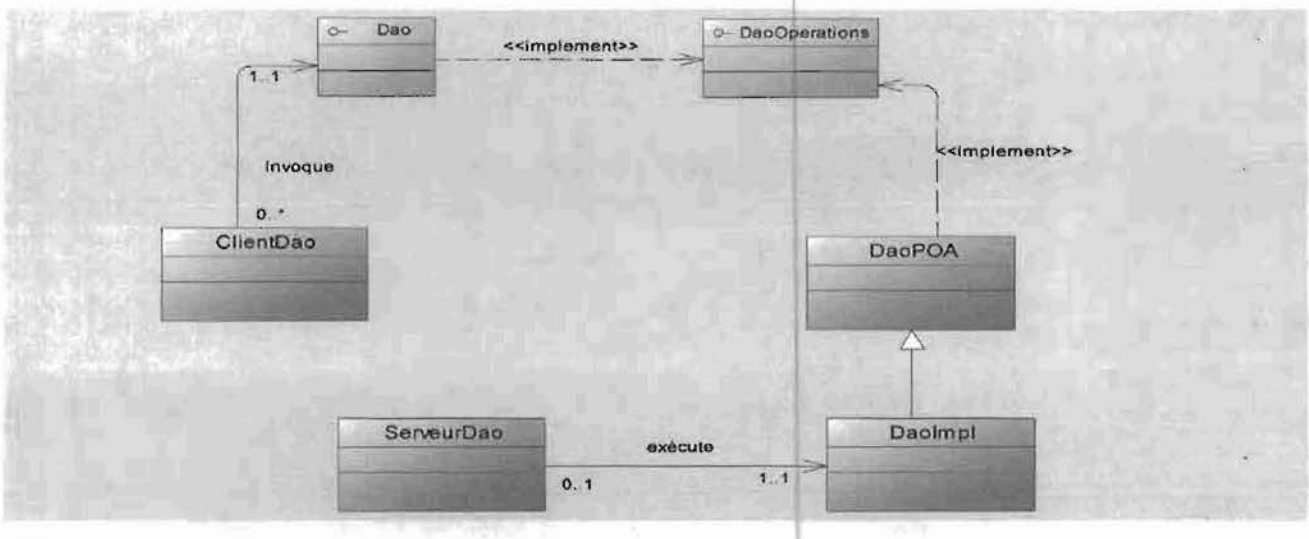


Figure 33 : Diagramme de classes mode POA.

Le développement d'un serveur et d'un client suit généralement les même processus donnés dans le *tableau IX.*

Tableau IX : Processus de développement d'un client et d'un serveur.

Client	Serveur
1- initialiser le bus CORBA ; 2- obtenir les références des objets utilisés par l'application ; 3- appliquer des traitements sur les objets obtenus.	1- initialiser le bus CORBA ; 2- initialiser l'adaptateur d'objet ; 3- créer une implémentation de l'objet ; 4- enregistrer cette implémentation auprès de l'adaptateur d'objet ; 5- diffuser la référence de l'objet d'implémentation aux applications clientes ; 6- se mettre en attente des requêtes d'objets.

II.3- Conception de la couche Données

Pour rappel, cette couche contient les données stockées physiquement sur le support de stockage de manière relationnelle. La modélisation étant objet, un passage du modèle objet au modèle relationnel s'impose. Il s'agit de trouver les équivalences entre les deux modèles pour effectuer le rapprochement.

II.3.1- Rapprochement modèle objet et modèle relationnel

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 (T-uplet) et son OID (identifiant de l'objet) sert de clé primaire. Le *tableau X* montre le rapprochement entre les deux mondes.

Tableau X : Equivalences entre concepts objets et relationnel

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

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 supplémentaire. Cette dernière stocke des couples de clés étrangères provenant de chacune des deux classes de l'association.

II.3.2- Extrait du modèle physique de données

Un modèle physique de données montre la structure de déploiement des données sur le support de stockage. A partir du diagramme de classes de la phase de *spécification*, Power AMC a généré le modèle physique de données de SGBCoM. Pour se faire, il a mis en œuvre les règles ci-dessus. C'est ce modèle qui est utilisé pour générer la base de données du système.

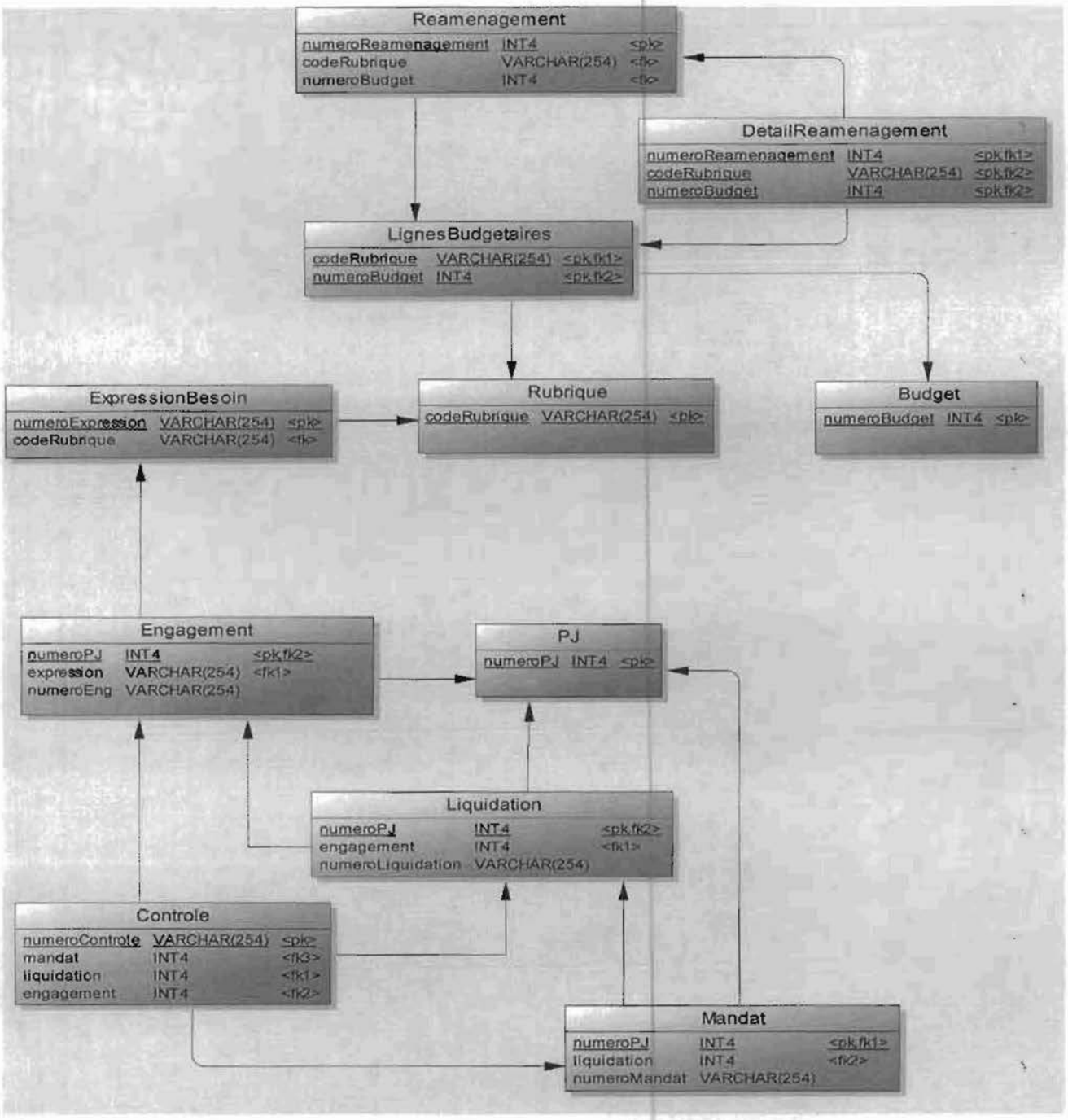


Figure 34 : Extrait du modèle physique de données de SGBCoM.

II.4- Mapping objet-relationnel

Rappelons que le mappage objet-relationnel permet de transformer une instance d'une classe en un enregistrement d'une base de données. Ce mappage est fait sur les objets de la couche *Modèle objets*.

Dans notre conception IDL ces objets sont représentés par le concept *struct*. Traduit en Java, ceux-ci donnent des classes de type *final* avec des attributs de visibilité *public*.

Pour réaliser le mapping, les objets doivent être sérialisés. Pour qu'une classe puisse être sérialisée elle ne doit pas être munie du type *final*. Donc, ce type sera enlevé devant les classes devant la couche *Modèle objets*. Aussi, les attributs auront la visibilité *protected*.

La configuration du mappage peut se faire de deux (02) manières : par les annotations et les fichiers XML. La première option permet de gagner en temps et n'oblige pas de comprendre XML. Le *tableau XI* donne quelques exemples d'annotations.

Tableau XI : Les annotations JPA pour le mappage objet-relationnel.

Annotations	Description
@Entity	Marque la classe pour la persistance de ses objets.
@Table	Définit la table qui sera utilisée lors de la persistance.
@Id	Définit la clé primaire.
@Column	Spécifie la colonne qui sera mise en correspondance avec le champ de la classe.
@OneToOne	Définit le champ de la relation 1-1.
@ManyToOne	Définit le champ de la relation N-1.
@OneToMany	Définit le champ de la relation 1-N.
@ManyToMany	Définit le champ de la relation N-M.

II.5- Conception de la couche Présentation

La couche de présentation concerne en gros la partie IHM de SGBCoM. Ce dernier est conçu à l'aide des techniques de fenêtrage.

A part la fenêtre principale, toutes les autres ont été abordées en phase de *spécification* avec les maquettes. Il reste donc à définir cette fenêtre. Elle est organisée de la manière suivante :

- pour faciliter l'accès aux opérations courantes (*engagement*, *liquidation*, *mandat*...), elles sont structurées en onglets sur cette fenêtre ;
- les opérations de paramétrage sont accessibles à travers le menu *Fichier*.

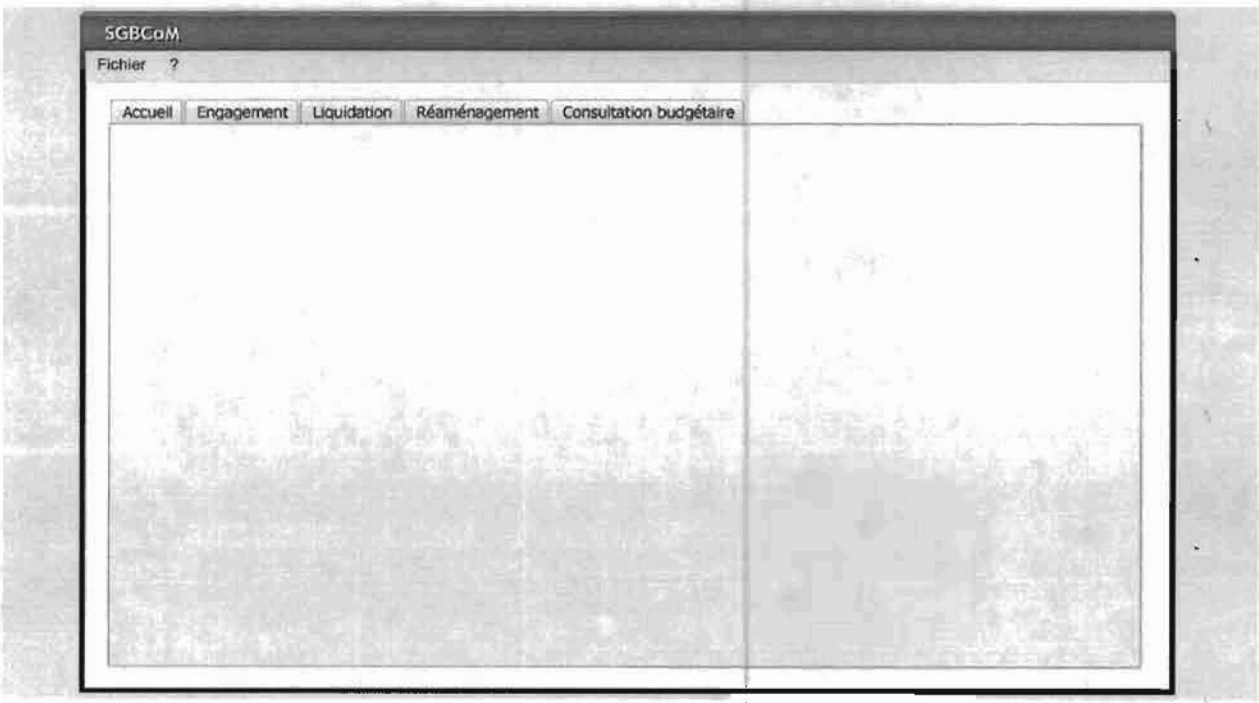


Figure 35 : Maquette de la fenêtre principale.

Cette maquette montre l'organisation de la fenêtre principale du système. Les onglets représentent les opérations effectuées fréquemment. Selon le profil de l'utilisateur certains onglets sont activés ou désactivés. Par exemple, un utilisateur avec le profil *DAF* verra les onglets "Engagement", "Liquidation", "Réaménagement" et "Consultation budgétaire" activés. Cependant, un utilisateur avec le profil *Projets* ne verra que l'onglet "Consultation budgétaire" activé. L'onglet "Accueil" est activé quelque soit le profil.

CONCLUSION

La conception qui est subdivisée en deux parties, conception architecturale et détaillée, a permis au niveau de sa première partie de spécifier les couches de SGBCoM. La seconde partie, quant à elle, a étudié en détail ces différentes couches et donné ainsi un produit prêt à être construit.

CHAPITRE 5 :
REALISATION

INTRODUCTION

La phase précédente nous a donné une solution prête à être codée. Il s'agit donc à cette phase de la construction effective de cette solution. Les paragraphes suivants présentent d'abord le processus de développement de l'application avant d'aborder le déploiement de celle-ci.

I- DEVELOPPEMENT

Le développement de SGBCoM consiste à la programmation avec le langage Java. Le processus de notre développement passe par la génération du code IDL par l'outil ***Power AWC***, la compilation de ce code en Java et la programmation.

I.1- Génération de code avec *Power AMC*

PowerAMC est une solution de modélisation et de gestion de métadonnées, destinée aux architectures de données, aux architectures d'informations et aux architectures d'entreprise. Cet outil permet la modélisation en UML et la génération de code dans le langage souhaité (IDL CORBA, Java...).

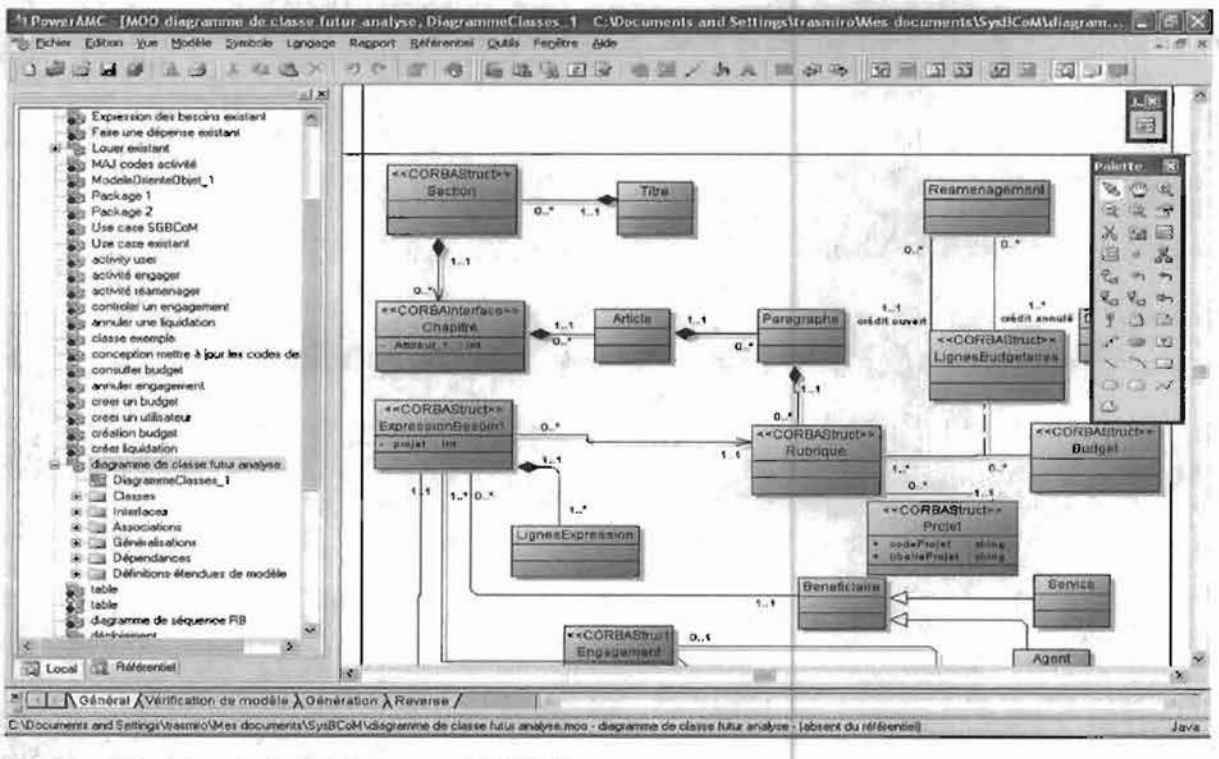


Figure 36 : Vue de l'outil Power AMC 15.

I.2- Compilation IDL CORBA

Le produit spécifiant la norme CORBA, appelé encore l'*ORB CORBA*, que nous avons utilisé est celui intégré à Java. Son compilateur (*idlJ*) traduit uniquement le code IDL CORBA en Java. La compilation se passe en ligne de commande.

Figure 37 : Compilation du fichier IDL en ligne de commande.

I.3- Programmation en Java

Pour la programmation en Java nous avons utilisé ***NetBeans 6.5***. C'est un *environnement de développement intégré (EDI)*, placé en open source par Sun. En plus de Java, NetBeans supporte également différents autres langages comme Python, C, C++, JavaScript... Il comprend toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, éditeur graphique d'interfaces et de pages Web).

Conçu en Java, NetBeans est disponible sous Windows, Linux, Solaris et Mac OS X. Un environnement *Java Development Kit (JDK)* est requis pour les développements en Java.

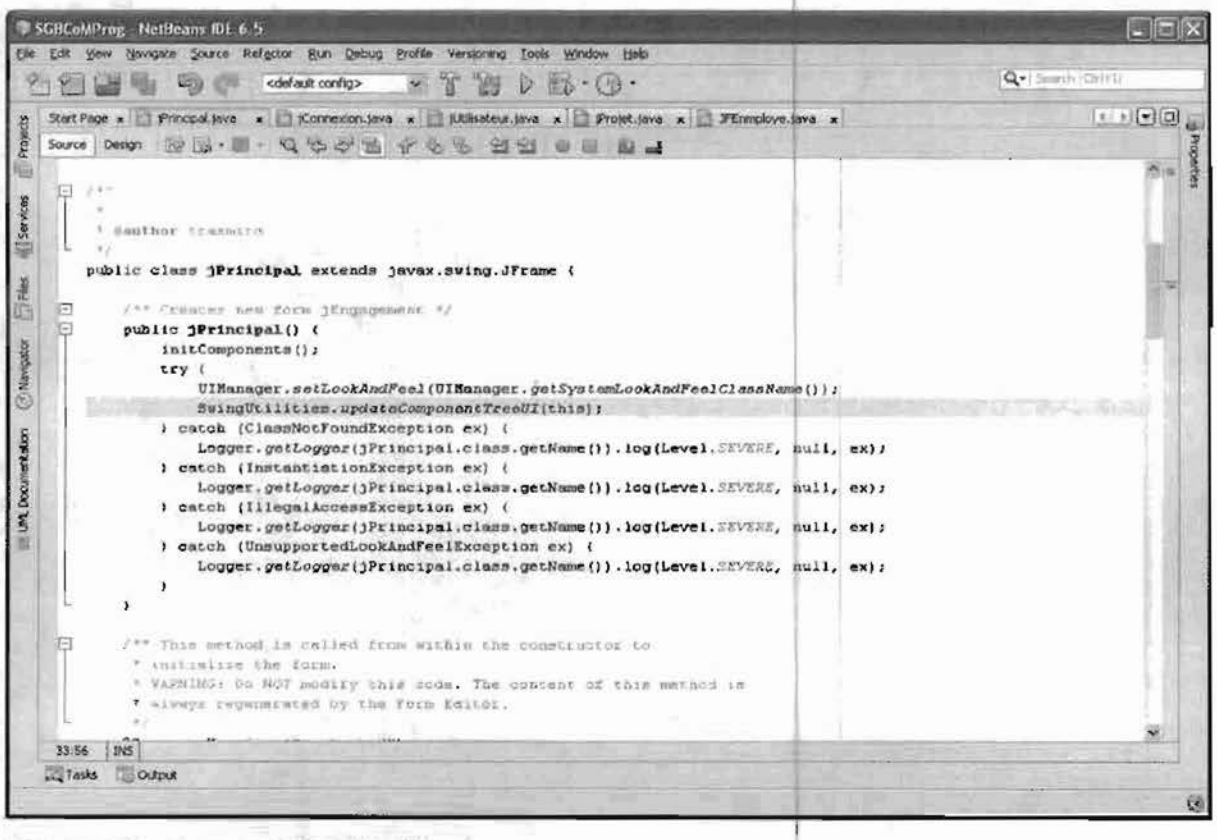


Figure 38 : Vue de l'EDI NetBeans.

II- DEPLOIEMENT

Le déploiement est la diffusion des composants d'une application sur des machines du réseau. Après avoir passé en revue la configuration des différentes couches de *SGBCoM*, un diagramme de déploiement montre la distribution des composants.

II.1- Configuration d'un client et serveur CORBA

Le bus CORBA offre un environnement orienté objet et client/serveur : l'application client peut invoquer des objets gérés (serviteurs) par l'application serveur. Pour cela, le client doit posséder une référence du serviteur afin d'accéder à ses services. Cette référence est l'*IOR* (Interoperable Object Reference).

Le serveur après avoir créé un serviteur, lui attribue un IOR par l'intermédiaire de l'adaptateur d'objets. Dans notre cas, cette référence est stockée dans un fichier texte. Le client, pour invoquer une méthode du serveur, accède à cet IOR pour instancier la souche client.

Serveur

```
//Création du serviteur
DaoImplement daoImpl = new DaoImplement();

//creation de l'IOR
org.omg.CORBA.Object obj = poa.servant_to_reference( daoImpl );

//creation du fichier de stockage
PrintWriter pw = new PrintWriter( new FileWriter( "serveurdao" ));

// Publication de l'ior
pw.println( orb.object_to_string( obj ));
```

Client

```
File f = new File ("serveurdao");
BufferedReader br = new BufferedReader( new FileReader( f ));

 // lire l'ior
org.omg.CORBA.Object obj = orb.string_to_object( br.readLine() );
br.close();

//active la souche
Metier metier = MetierHelper.narrow (obj);
```

Figure 39 : La diffusion et la lecture d'un IOR.

II.2- Configuration de PostgreSQL

La configuration du SGBD est faite à partir d'un fichier XML. Cette configuration consiste à configurer la couche *JDBC*.

```
1- <?xml version="1.0" encoding="UTF-8"?>
2- <persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence">
3-
4- <persistence-unit name="sgbcom" transaction-type="RESOURCE_LOCAL">
5-                <!-- provider -->
6-                <provider>org.hibernate.ejb.HibernatePersistence</provider>
7-                <properties>
8-                <!-- Classes persistantes -->
9-                <property name="hibernate.archive.autodetection" value="class, hbm" />
10-
11-               <! -- connexion JDBC -->
12-               <property name="hibernate.connection.driver_class" value="org.postgresql.Driver" />
13-               <property name="hibernate.connection.url" value="jdbc:postgresql:sgbcom" />
14-               <property name="hibernate.connection.username" value="sgbcom" />
15-               <property name="hibernate.connection.password" value="sgbcom" />
16-      </persistence-unit>
17- </persistence>
```

Figure 40 : La configuration de PostgreSQL.

Les lignes 12-15 servent à configurer JDBC :

- ligne 12 : la classe du pilote JDBC de PostgreSQL ;
- ligne 13 : l'url de la base de données utilisée ;
- lignes 14, 15 : l'utilisateur de la connexion et son mot de passe.

II.3- Exécution de SGBCoM

Pour exécuter un programme écrit en Java, il faut une machine virtuelle Java. Donc les machines d'accueil des programmes de SGBCoM doivent être munies de cette machine virtuelle.

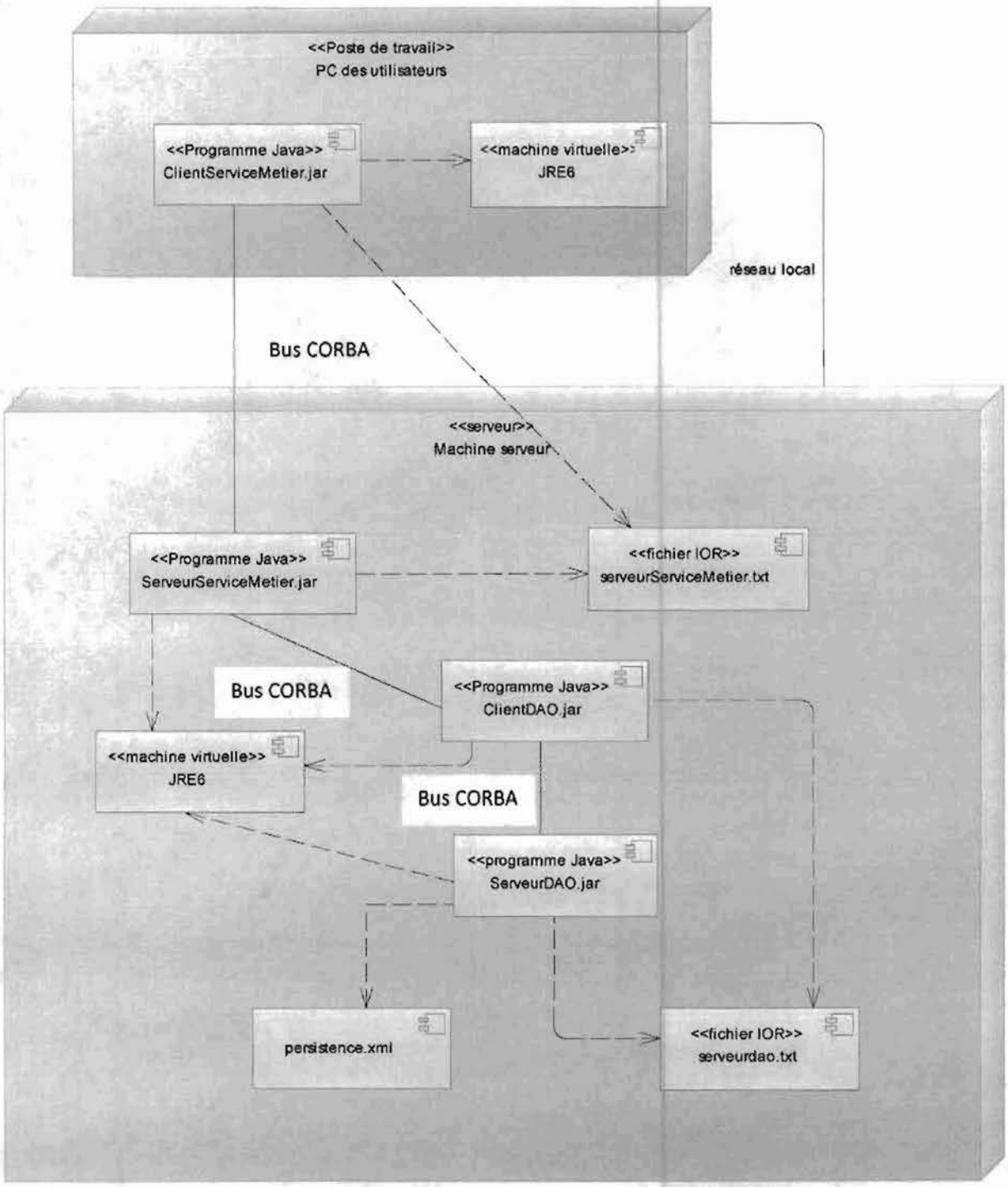


Figure 41 : Le diagramme de déploiement de SGBCoM.

CONCLUSION GENERALE

Suite à une expression des besoins formulée par les utilisateurs, une étude a été menée dans le but de parfaire le système d'information budgétaire et comptable du Centre MURAZ.

Cette étude a commencé par l'analyse du système actuel qui a permis de faire le diagnostic de l'existant. Ce système est composé de deux logiciels qui s'occupent de la gestion budgétaire et comptable du Centre MURAZ. Ces deux systèmes informatiques sont incapables d'échanger des informations. Pourtant, ils utilisent des données communes. Des solutions ont donc été proposées afin de gérer le budget et la comptabilité par une seule application. La solution retenue a été étudiée en détail de la phase de spécification jusqu'à celle de la conception détaillée. Cette dernière a permis d'adapter la solution au codage.

La programmation qui s'est faite en Java a mis en œuvre les technologies telles que CORBA et JPA/Hibernate.

Comme montré ci-dessus, CORBA est un "middleware objet" prônant la séparation nette des différentes couches d'une application. Les couches ainsi séparées sont réparties entre les différentes machines du réseau et CORBA assure la communication entre elles. De nombreuses autres solutions "middlewares objet", telles que RMI et DCOM, existent pour la construction d'applications réparties. Cependant, CORBA se distingue car elle offre :

- *Une solution ouverte et évolutive* : choisir CORBA revient à ne pas s'enfermer dans une solution propriétaire évoluant difficilement. L'OMG réagit très vite aux demandes du marché.
- *L'interopérabilité entre composants hétérogènes.*
- *Le libre choix des technologies d'implantation.*

Toutefois, lorsque l'on utilise CORBA pour bâtir une application répartie, les mécanismes propriétaires qui nous lient à un fournisseur sont à éviter.

Pour la gestion des données, un SGBD du modèle relationnel était convenable pour les objectifs du projet. Les SGBD de ce modèle sont dotés d'un langage de requête universel, SQL, qui laisse une ouverture pour l'utilisation des données par d'autres applications. C'est ainsi que PostgreSQL qui est un SGBDR qui gère bien les bases de données, comme le nôtre, a été retenue. Utilisant un SGBDR et un langage de programmation objet, il a donc fallu utiliser un outil de mappage objet-relationnel — JPA/Hibernate ; ceci, afin de faciliter l'accès aux données.

Cette étude a abouti à la construction d'une application répartie avec ces technologies qui est encours. Nous attendons terminer l'application qui s'en suivra avec des tests. Une fois les tests terminés, des formations seront faites pour initier les utilisateurs à la nouvelle application. Ensuite, l'application sera mise en place et suivie afin de s'assurer qu'elle répond totalement aux besoins des utilisateurs.

BIBLIOGRAPHIE

[B1] Pascal Rocques, & Franck Vallée, *UML 2 en action, de l'analyse des besoins à la conception J2EE*, 3e édition, 2004.

[B2] Kazi A. B. & Rostane Z., *Suivie des enseignements du LMD par application de la méthode 2TUP, Mémoire d'ingéniorat, Université Abou Bekr Belkaid de Tlemcen, Tlemcen(Algérie)*, 2007.

[B3] *UML, La Notation unifiée de modélisation objet, de Java aux EJB*, Michel Lai, 2e édition, 2000.

[B4] *CORBA, des concepts à la pratique*, J.-M. Geib, C. Gransart et P. Merle, 2e édition, 1999.

[B5] *Etude d'un plan de connexion internet par WiFi du Centre MURAZ, mémoire d'ingéniorat de travaux en technologie des réseaux et systèmes informatiques, ISIG-BOBO*, Yacouba DJIBO, 2009.

[B6] *Objets réparti, guide se survie*, Robert Orfali, Dan Harkey et Jeri Edwards, 1996.

[B7] Rémi LEBLOND, *Vers une architecture n-tiers, Oral probatoire*, soutenu le 02/12/1999.

WEBOGRAPHIE

[W1] *Comparatif des SGBD*, http://fadace.developpez.com/sgbdcmp/, 5 octobre 2010.

[W2] *FAQ sur DCOM/OLE*, http://windows.developpez.com/dcom/t1.html, 7 octobre 2010.

[W3] *Part du Marché des SGBD*, http://www.lemagit.fr/article/oracle-idc-sgbd-sql/677/1/marche-mondial-des-sgbd-oracle-toujours-1/, 20 octobre 2010.

[W4] *Part du Marché des SGBD*, http://fr.wikipedia.org/wiki/Syst%C3%A8me_de_gestion_de_base_de_donn%C3%A9es#Le_march.C3.A9, 20 octobre 2010.

[W5] *Tutoriel persistance Java 5*, ftp://ftp-developpez.com/tahe/fichiers-archive/jpa.pdf, 20 novembre 2010.

[W6] *Tutoriel persistance Java 5*, http://loic-frering.developpez.com/tutoriels/java/hibernate-jpa-spring-tapestry/, 20 novembre 2010.

[W7] *Modèle objet-relationnel*, http://deptinfo.unice.fr/~grin/mescours/minfo/modpersobj/supports/Objet-Relationnel6.pdf, 24 décembre 2010.

[W8] http://www.onatel.bf/internet/ls.htm, 17 décembre 2010.

[W9] *Les annotations de JPA*, http://www.jpox.org/docs/1_1/jpa_annotations.html, 20 décembre 2010.

ANNEXE

A- DESCRIPTION TEXTUELLE DES CAS D'UTILISATION DU SYSTEME ACTUEL

Cette partie présente la description textuelle des cas d'utilisation du système actuel.

Tableau XII : description détaillée du cas d'utilisation "Exprimer les besoins".

Titre : Exprimer les besoins
But : établir une fiche d'expression des besoins
Acteur (s) : services ou projets
Pré condition (s) : néant
Description des enchainements : Ce cas commence lorsque le service a besoin d'acheter un matériel, ou d'une prestation de service, ou de faire toute autre dépense. **Enchainements nominaux** *Enchainement (a) Créer une expression des besoins* Les projets créent une expression des besoins en remplissant une fiche. *Enchainement (b) Soumettre une expression des besoins* Après avoir rempli la fiche d'expression des besoins, on l'envoie au Contrôle financier. Ce dernier après avoir donné son approbation, transmet la fiche au DG puis la DAF. Ce cas prend fin lorsque la fiche d'expression des besoins arrive à la DAF
Exception (s) : néant
Post condition (s) : néant

Tableau XIII : description détaillée du cas d'utilisation "Faire une dépense"

Titre : Faire une dépense
But : concrétiser une dépense d'un service
Acteur (s) : services, Gestionnaire budgétaire, Fournisseurs et Agence comptable
Pré condition (s) : néant
Description des enchainements : Ce cas commence avec l'expression des besoins faite par un projet. **Enchainements nominaux** *Enchainement (a) Etablir expression des besoins* Inclusion du cas d'utilisation *"Exprimer les besoins"* *Enchainement (b) Faire un engagement de dépense* Inclusion du cas d'utilisation *"Engager une dépense"* *Enchainement (c) Liquider une dépense* Inclusion du cas d'utilisation *"Liquider une dépense"* *Enchainement (d) Faire un mandat* Inclusion du cas d'utilisation *"Etablir le mandat d'une liquidation"* Ce cas prend fin lorsque la dépense a été totalement liquidée.
Exception (s) : néant
Post condition (s) : néant

Tableau XIV : description détaillée du cas d'utilisation "Liquider une dépense".

Titre : Liquider une dépense
But : éditer une liquidation avec le logiciel budgétaire
Acteur (s) : Gestionnaire budgétaire
Pré condition (s) : - le gestionnaire s'est authentifié ; - l'engagement a été déjà établi.

Description des enchainements :

Ce cas commence lorsque le gestionnaire demande l'établissement d'une liquidation.

Enchainement nominaux

Enchainement (a) créer une liquidation

Le gestionnaire saisit le numéro d'engagement. Le logiciel budgétaire affiche les autres informations en rapport avec l'engagement. Ensuite, le gestionnaire saisit le montant à liquider. Le logiciel calcule le reste à liquider et le montant total des liquidations de l'année pour la ligne budgétaire. Pour finir, le gestionnaire entre les références des pièces justificatives.

Enchainement (b) imprimer une liquidation

Le gestionnaire précise la liquidation et procède à son impression.

Enchainement (c) contrôler une liquidation

Après avoir imprimée la liquidation, le gestionnaire l'envoie au Contrôle financier. Celui-ci vérifie la liquidation et la renvoie au gestionnaire avec son jugement.

Enchainement alternatifs

Enchainement (d) Modifier une liquidation

Le gestionnaire peut modifier une liquidation en-cours ou non approuvée par le Contrôle financier.

Enchainement (e) annuler un engagement

Le gestionnaire peut annuler une liquidation en-cours ou non approuvée par le Contrôle financier.

Ce cas se termine lorsque :

- la liquidation est approuvée par le Contrôle financier ;
- ou annulée par le gestionnaire.

Exception (s) : néant

Post condition (s) : néant

Tableau XV : description détaillée du cas d'utilisation "Faire un mandat".

Titre : Faire un mandat
But : imprimer le mandat concernant une liquidation
Acteur (s) : Gestionnaire budgétaire
Pré condition (s) : le gestionnaire s'est authentifié, la liquidation a été déjà enregistrée
Description des enchainements : Ce cas débute lorsque le gestionnaire demande à imprimer un mandat. **Enchainements nominaux** *Enchainement (a) Choisir une liquidation* Le gestionnaire après avoir accédé à l'écran d'impression d'un mandat, introduit le numéro de la liquidation. *Enchainement (b) Imprimer un mandat* Ensuite le gestionnaire procède à l'impression du mandat.
Exception (s) : néant
Post condition (s) : néant

Tableau XVI : description détaillée du cas d'utilisation "Demander situation budgétaire".

Titre : Demander situation budgétaire
But : donner la situation budgétaire d'un projet
Acteur (s) : Projets, Gestionnaire budgétaire
Pré condition (s) : néant
Description des enchainements : Ce cas a comme point initial lorsque le projet veut connaitre sa situation budgétaire. **Enchainements nominaux** *Enchainement (a) Contacter le gestionnaire* Le projet contacte le gestionnaire. *Enchainement (b) Consulter situation budgétaire*

Le gestionnaire accède à la partie "Situation budgétaire par projet". Il précise le projet et le logiciel affiche la situation budgétaire du projet. Ce cas prend fin lorsque le gestionnaire communique au projet sa situation budgétaire.
Exception (s) : néant
Post condition (s) : néant

Tableau XVII : description détaillée du cas d'utilisation "Gérer les dépenses".

Titre : Gérer les dépenses
But : constater la charge en enregistrant un mandat et la régler une fois le payement effectué
Acteur (s) : Agence comptable
Pré condition (s) : l'acteur s'est authentifié
Description des enchainements : Ce cas commence dès lors que l'utilisateur demande à constater la charge. **Enchainements nominaux** *Enchainement (a) Choisir le type d'opération* Après avoir choisi l'onglet "Saisie", puis le sous-onglet "Choix", l'utilisateur fait le choix de l'opération (dépense ou recette). *Enchainement (b) Constater la charge* L'utilisateur entre les références du mandat et débite le compte du projet supportant la charge. Après, il crédit le compte du fournisseur *Enchainement (c) Régler la charge* L'utilisateur introduit les références du mandat et crédite le compte du projet supportant la charge. Ensuite, il débite le compte du fournisseur Ce cas prend fin lorsque le gestionnaire communique au projet sa situation budgétaire.
Exception (s) : néant
Post condition (s) : néant

Tableau XVIII : description détaillée du cas d'utilisation

Titre : Gérer les recettes
But : constater la charge en enregistrant un mandat et la régler une fois le payement effectué
Acteur (s) : Agence comptable
Pré condition (s) : l'acteur s'est authentifié
Description des enchainements : Ce cas commence dès lors que l'utilisateur demande à constater la charge. **Enchainements nominaux** *Enchainement (a) Choisir le type d'opération* Après avoir choisi l'onglet "Saisie", puis le sous-onglet "Choix", l'utilisateur fait le choix de l'opération (dépense ou recette). *Enchainement (b) Constater la recette* L'utilisateur entre les références de la facture et débite le compte du client, puis crédite le compte bénéficiant de cette recette. *Enchainement (c) Régler la facture* L'utilisateur introduit les références de la facture. Après il débite le compte de la banque et crédite le compte client. Ce cas prend fin lorsque le gestionnaire communique au projet sa situation budgétaire.
Exception (s) : néant
Post condition (s) : néant

B- REGLES DE PROJECTION EN JAVA

Ci-dessous dans le *tableau XXV*, les règles de projection de l'IDL en JAVA, et dans le *tableau XXVI*, les règles de passage de paramètres en Java.

Tableau XIX : règles de projection en Java.

IDL	JAVA
module	package
interface	interface
exception utilisateur	extends org.omg.CORBA.UserException
exception CORBA	extends org.omg.CORBA.SystemException
const	public static final
short ; unsigned short	short
long ; unsigned long	int
long long ; unsigned long long	long
float	float
double	double
char	char
string	String
boolean	boolean
octet	octet byte
Object	org.omg.CORBA.Object
any	org.omg.corba.Any

Tableau XX : passage de paramètres en Java.

Type IDL	in	inout	out	return
short	short	ShortHolder	ShortHolder	short
long	int	IntHolder	IntHolder	int
double	double	DoubleHolder	DoubleHolder	double
Object	Object	ObjectHolder	ObjectHolder	Object