République Algérienne et démocratique
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Mouloud Mammeri de Tizi-Ouzou

Faculté de génie Electrique et Informatique
Département Informatique

De fin d'études

En vue de l'obtention du diplôme : **Master II en Informatique**
Option : **Conduite de Projets Informatiques (CPI)**

Thème *:*

De la réalisation d'un système d'informations en utilisant l'approche systémique A la réalisation d'un système d'informations en utilisant l'approche objet

Réalisé par :
M^elle^ ABDEDOU Souhila

Soutenu devant le jury :

M^me^	M. SAAD née BELKADI	**Présidente**
M^me^	K. HOCINI née OUKFIF	**Examinatrice**
M^me^	R. SKENDRAOUI née HADDAOUI	**Examinatrice**
M^me^	R. LAGAB née AOUJIT	**Promotrice**

Année : 2014/ 2015

REMERCIEMENT

Je remercie le bon dieu qui m'a donné le courage et la volonté pour effectuer ce travail.

Je remercie les membres de ma famille qui m'ont encouragé à effectuer ce travail.

Je remercie ma promotrice M^{me} LAGAB pour sa confiance en mes capacités et mes connaissances et son encouragement pour avancer et aller à l'avant.

Je tiens à remercier les membres de jurés d'avoir accepté de juger mon travail, pour leur sympathie. Ainsi que leurs critiques objectives.

Je remercie tous ceux qui m'ont encouragée de prés ou de loin à effectuer ce travail.

Sommaire

Table des figures

Partie (I) :

Partie (II) :

Chapitre (I) : Architecture Client Serveur

Chapitre (II) : Environnement javaEE

Chapitre IV : Conception & Réalisation

Introduction générale:

Le système d'information (SI) est un ensemble d'éléments (personnel, matériel, logiciel...) permettant d'acquérir, traiter, mémoriser et communiquer des informations.

L'objectif principal d'un système d'information (SI) consiste à restituer l'information à la personne concernée sous une forme appropriée et au moment opportun.

Il est généralement spontané dans les entreprises de taille réduite, mais il fait l'objet d'une attention toute particulière dans les grandes entreprises. En effet, son rôle a grandi du fait d'un environnement changeant, de l'émergence de très grandes entreprises internationales et du développement des applications et de la capacité des traitements informatiques.

Le système d'information est aujourd'hui reconnu comme une composante hautement stratégique pour toute organisation (entreprise privée, organisme publique, etc.) car il lui apporte des avantages concurrentiels durables (excellence opérationnelle, meilleure qualité de service, augmentation de chiffre d'affaire, meilleure connaissance/relation avec ses clients et fournisseurs, meilleure prise de décision), ou tout simplement des capacités de survie dans un monde globalisé, complexe, instable et hautement concurrentiel. Le SI est alors considéré comme le centre nerveux des entreprises.

De ce fait, la gestion des systèmes d'informations et leur conception représentent la préoccupation majeure des entreprises.

Plusieurs approches de conception des systèmes d'informations se succédaient, on cite

L'approche cartésienne (fonctionnelle) : préconise d'analyser et de concevoir des systèmes d'informations en se centrant sur ses fonctions.

L'analyse et la conception du système d'informations débutent par l'identification du SI à une fonction globale de gestion. La fonctionne globale est décomposée jusqu'à arriver à une fonction élémentaire (raffinement successif de la fonction globale).

Les méthodes cartésiennes mettent l'accent sur la modélisation des processus fondée sur une technique d'analyse qui consiste à éclater une fonction globalement conçue en processus spécifiques.

La décomposition cherche également à mettre en évidence les interrelations entre processus au moyen de flux de données.

Parmi les méthodes basées sur cette approche la méthode SADT (Structured Analysis and Design Technique).

L'approche systémique : fait la dichotomie entre les données et les traitements. La Méthode Merise suit cette approche.

L'approche objet : s'articule sur le concept d'objet qui englobe les données et les traitements qui lui sont propres. Différentes méthodes suivent cette approche ; OOSE, OMT ainsi que le langage de modélisation UML.

Pour réaliser ce travail, deux approches ont été utilisée

L'approche systémique dans le cadre de la méthode Merise.

L'approche objet dans le cadre du langage de modélisation UML2.0

Ainsi le mémoire est devisé en deux parties

Partie I : qui représente la réalisation d'un SI en utilisant l'approche systémique.

Partie II : qui décrit la réalisation d'un SI en utilisant l'approche objet.

Parti I :
Réalisation d'un SI en utilisant l'approche systémique.

Introduction :

Avant l'apparition de l'ordinateur, toutes les taches et procédures administratives étaient manuelles.
L'employé est amené alors à effectuer des taches répétitives et complexes ce qui l'induit dans la plupart des cas à l'erreur.
Ainsi l'apparition de l'ordinateur a apporté les avantages suivants :

- Décharger l'employé des taches complexes et répétitives.
- Considéré comme un outil d'aide à la décision : en fournissant un maximum d'informations en un temps très court, grâce à la vitesse de sélection dans la base d'informations.

Cependant dans la plupart des cas, l'ordinateur n'est pas utilisé d'une façon optimale ; il est considéré comme un simple outil pour effectuer des traitements de textes avec le logiciel Word et des simples calculs en utilisant le logiciel Excel. Et ainsi la recherche d'information reste manuelle dans les fichiers résultants.
En plus de la lenteur de recherche d'informations dans les fichiers, l'utilisateur de cette méthode pourra être amené à créer de nouveaux fichiers qui contiendront peut-être des informations déjà présentes dans d'autres fichiers.
Or que dans le domaine des technologies les limites présentes dans l'approche des fichiers ont été rectifiées par l'approche des bases de données.
Le recours aux bases de données est une alternative au procédé classique de stockage de données. Les données sont placées dans des fichiers manipulés par le système de gestion de bases de données. Il facilite le partage des informations, permet le contrôle automatique de la cohérence et de la redondance des informations, la limitation de l'accès aux informations et la production plus aisée des informations synthétiques à partir des renseignements bruts.

Ainsi la complexité des opérations (recherche, ajout, modification, suppression) effectuées sur le contenu de la base de données est cachée par le SGBD.

Le travail réalisé dans cette première partie consiste à faire le passage de l'utilisation de l'approche des fichiers vers l'utilisation l'approche de bases de données.
Dans cette partie, est présentée la méthode Merise, la description de la solution proposée et la critique de la solution proposée.

I. Système d'information :

1. Système :

Un système est un ensemble d'éléments matériels ou immatériels (hommes, machines, règles…) en interaction, organisés en fonction d'un objectif à atteindre et transformant un ensemble d'éléments reçus en entrée en un ensemble d'éléments en sortie.
De ce fait l'organisation est un système qui un objectif social ou économique ou administratif.

2. Différents systèmes de l'organisation :

L'organisation peut être décomposée en trois sous systèmes :
Sous système de pilotage : pilote et contrôle l'ensemble des sous systèmes de l'organisation en prenant des décisions. C'est le sous système qui fixe les objectifs ainsi que les méthodes et les moyens pour les atteindre.
Sous système opérant : c'est la partie qui réalise l'objectif de l'organisation ;
Transforme en actions, les décisions prises par le système de pilotage.
Exécute les ordres et directives émises par le système de pilotage.
Sous système d'information :
Le système opérant réalise les activités de l'organisation en se basant sur un ensemble d'informations et règles fournies par le système d'information.
Et c'est le système d'informations qui fournit les informations en fonction desquelles, le système de pilotage fixe ses objectifs et adapte la stratégie de l'organisation pour les atteindre.
En effet le système d'information a pour rôle de traiter et véhiculer l'information pour les deux autres systèmes (opérant et pilotage).

2. Fonctions du système d'information :

Pour prendre en charge les informations de l'organisation, le système d'informations procède à :
Leur collecte : collecte les informations provenant des systèmes pilotage et opérant.
Leur mémorisation : le système d'information doit garder trace de toutes les informations collectées.
Leur traitement : les informations stockées par le système d'information subissent des traitements dans le but de produire d'autres informations sous forme de résultats.
Leur diffusion : pour assurer une bonne coordination entre les systèmes de l'organisation et par conséquent un bon fonctionnement de celle celle-ci, l'information doit circuler d'un système à un autre ainsi que de l'intérieur de l'organisation vers l'environnement extérieur et inversement.

3. Aspect statique et aspect dynamique du SI :

Aspect statique : consiste en la fonction de mémorisation.
Aspect dynamique : les fonctions de collecte, de traitement et de transmission sont les fonctions dynamiques du système d'informations.

4. Le SI formel et SI informel

SI formel : ses règles de fonctionnement et l'ensemble des informations qu'il manipule sont clairement définies et justifiées par des documents officiels.
SI informel : constitué d'un ensemble d'informations et de procédures non recensées par le système d'information formel.

5. SI Automatisable : est la partie du système d'informations qui concerne les actions programmables.
Les Actions programmables sont des actions qui quand elles agissent sur les mêmes entrées auront les mêmes résultats. Or que dans le cas des informations non programmables, la connaissance des entrées ne suffit pas pour déterminer les sorties ; elles rentrent en jeu les décisions. Dans ce dernier cas, la machine ne peut pas décider à la place de l'homme.

II. démarche utilisée :

II.1. Description du domaine d'étude.

Notre domaine d'étude concerne une direction qui a pour rôle le contrôle des réalisations de différents établissements par rapport aux prévisions. Dans le but de calculer l'écart entre les prévisions et les réalisations, après avoir réalisé le bilan des réalisations.

II.2. cycle de vie utilisé.

a.Définition :

Cycle de vie: Le « **cycle de vie d'un logiciel** » (en anglais *software life cycle*), désigne toutes les étapes du développement d'un logiciel, de sa conception à sa disparition. Dans le but de vérifier la conformité du logiciel avec les besoins exprimés. Et la **vérification** du processus de développement, c'est-à-dire l'adéquation des méthodes mises en œuvre.

Le cycle de vie permet de détecter les erreurs au plus tôt et ainsi de maîtriser la qualité du logiciel, les délais de sa réalisation et les coûts associés.

b.Etapes :

Le cycle de vie du logiciel comprend généralement au minimum les activités suivantes :

Définition des objectifs, consistant à définir la finalité du projet et son inscription dans une stratégie globale.

Analyse des besoins et faisabilité, c'est-à-dire l'expression, le recueil et la formalisation des besoins du demandeur (le client) et de l'ensemble des contraintes.

Conception générale. Il s'agit de l'élaboration des spécifications de l'architecture générale du logiciel.

Conception détaillée, consistant à définir précisément chaque sous-ensemble du logiciel.

Codage (Implémentation ou programmation), soit la traduction dans un langage de programmation des fonctionnalités définies lors de phases de conception.

Tests unitaires, permettant de vérifier individuellement que chaque sous-ensemble du logiciel est implémentée conformément aux spécifications.

Intégration, dont 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.

Documentation, visant à produire les informations nécessaires pour l'utilisation du logiciel et pour des développements ultérieurs.

Mise en production,

Maintenance, comprenant toutes les actions correctives (maintenance corrective) et évolutives (maintenance évolutive) sur le logiciel.

c.Types :

La séquence et la présence de chacune de ces activités dans le cycle de vie dépend du choix d'un modèle de cycle de vie

On distingue essentiellement :

Le modèle en cascade : Définit des phases séquentielles à l'issue de chacune desquelles des documents sont produits pour en vérifier la conformité avant de passer à la suivante.

Le modèle en V : Le modèle de cycle de vie en V part du principe que les procédures de vérification de la conformité du logiciel aux spécifications doivent être élaborées dès les phases de conception.

Pour réaliser la solution, j'ai opté pour le cycle de vie dont les étapes suivantes :

Analyse : qui consiste à analyser la situation existante de notre champ d'étude, ce en étudiant les différents documents utilisés, ainsi que la démarche de travail. Et ce dans le but de comprendre la situation existante afin de décerner les insuffisances et les besoins des utilisateurs.

Conception : concevoir le système adopté en utilisant les informations résultantes de la phase d'analyse. Pour concevoir la solution j'ai utilisé deux modèles de la méthode Merise qui sont le conceptuel des traitements (MCT) et le modèle logique des données (MLD).

Réalisation : qui consiste à réaliser la solution en utilisant un environnement de développement. J'ai utilisé l'environnement Acces2007.

II.3. Méthode Merise : [1][2]

a. Introduction :

La méthode MERISE date de 1978-1979, et fait suite à une consultation nationale lancée en 1977 par le ministère de l'Industrie dans le but de choisir des sociétés de conseil en informatique afin de définir une méthode de conception de systèmes d'information. Les deux principales sociétés ayant mis au point cette méthode sont le CTI (Centre Technique d'Informatique) chargé de gérer le projet, et le CETE (Centre d'Etudes Techniques de l'Equipement) implanté à Aix-en-Provence.
Cette méthode a surtout été utilisée en France, par les SSII de ses membres fondateurs (Sema-Metra, ainsi que par la CGI Informatique) et principalement pour les projets d'envergure, notamment des grandes administrations publiques ou privées.

MERISE est une méthode de conception, de développement et de réalisation de projets informatiques. Le but de cette méthode est d'arriver à concevoir un système d'information. Cette méthode reste adaptée pour la gestion des projets internes aux organisations, se limitant à un domaine précis.
La méthode MERISE est méthode systémique, donc elle est basée sur la séparation des données et des traitements à effectuer en plusieurs modèles conceptuels et physiques.

b. Cycle d'abstraction de conception des systèmes d'information :

La conception du système d'information se fait par étapes, afin d'aboutir à un système d'information fonctionnel reflétant une réalité physique. Il s'agit donc de valider une à une chacune des étapes en prenant en compte les résultats de la phase précédente. D'autre part, les données étant séparées des traitements, il faut vérifier la concordance entre données et traitements afin de vérifier que toutes les données nécessaires aux traitements sont présentes et qu'il n'y a pas de données superflues.

Cette succession d'étapes est appelée cycle d'abstraction pour la conception des systèmes d'information.
L'expression des besoins est une étape consistant à définir ce que l'on attend du système d'information automatisé, il faut pour cela :
faire l'inventaire des éléments nécessaires au système d'information délimiter le système en s'informant auprès des futurs utilisateurs

- Cela va permettre de créer le **MCC** (Modèle conceptuel de la communication) qui définit les flux d'informations à prendre en compte.
- L'étape suivante consiste à mettre au point **les modèles conceptuels ;** le **MCD** (Modèle conceptuel des données) et le **MCT** (Modèle conceptuel des traitements) décrivant les règles et les contraintes à prendre en compte.
- **le Modèle conceptuel des données (ou MCD) :**

schéma représentant la structure du système d'information, du point de vue des données, c'est-à-dire les dépendances ou relations entre les différentes données du système d'information (par exemple : le client, la commande, les produits, etc.).
Le MCD repose sur les notions d'entité et d'association et sur les notions de relations. Le modèle conceptuel des données s'intéresse à décrire la *sémantique* du domaine.

L'entité ou objet :

L'entité est définie comme un *objet de gestion* considéré d'intérêt pour représenter l'activité à modéliser (exemple : entité pays). À son tour, chaque entité (ou **objet**) est porteuse d'une ou plusieurs **propriétés** simples, dites atomiques (exemples : code, nom, capitale, population, superficie) dont l'une, unique et discriminante, est désignée comme **identifiant** (exemple : code).
L'entité représente le concept qui se décline, dans le concret, en occurrences d'individus.
Par construction, le MCD impose que toutes les propriétés d'une entité ont vocation à être renseignées (il n'y a pas de propriété « facultative »).
Le MCD doit, de préférence, ne contenir que le cœur des informations strictement nécessaires pour réaliser les traitements conceptuels (cf. MCT). Les informations calculées (ex : montant taxes comprises d'une facture), déductibles (ex : densité démographique = population / superficie) et *a fortiori* celles liées aux choix d'organisation conçus pour effectuer les traitements (cf. MOT) ne doivent pas y figurer.

L'association ou relation

L'association est un lien sémantique entre entités :
1. entité reliée à elle-même : la relation est dite **réflexive**,
2. entités : la relation est dite **binaire** (ex : une usine 'est implantée' dans un pays),
plus rarement 3 ou plus : **ternaire**, voire de dimension supérieure.
Une association peut également être porteuse d'une ou plusieurs propriétés (ex : 'date d'implantation' d'une usine dans un pays)

- **le Modèle conceptuel des traitements** (ou MCT), schéma représentant les traitements, en réponse aux évènements à traiter (par exemple : la prise en compte de la commande d'un client).

- Niveau logique ou d'organisation :

À ce niveau de préoccupation, les modèles conceptuels sont précisés et font l'objet de choix d'organisation. On construit :

- le **Modèle Logique des Données** (ou MLD), qui reprend le contenu du MCD précédent, mais précise la volumétrie, la structure et l'organisation des données telles qu'elles pourront être implémentées. Par exemple, à ce stade, il est possible de connaître la liste exhaustive des tables qui seront à créer dans une base de données relationnelle.

De ce fait, il est également appelée dérivation du MCD dans un formalisme adapté à une implémentation ultérieure, au niveau physique, sous forme de base de données relationnelle ou réseau, ou autres (ex : simples fichiers).
La transcription d'un MCD en modèle relationnel s'effectue selon quelques règles simples qui consistent d'abord à transformer toute entité en table, avec l'identifiant comme clé primaire, puis à observer les valeurs prises par les cardinalités maximum de chaque association pour représenter celle-ci soit (ex : card. max 1 [1-**1** ou 0-**1**]) par l'ajout d'une **clé étrangère** dans une table existante, soit (ex : card. max n [0-**N** ou 1-**N**]) par la

création d'une nouvelle table dont la clé primaire est obtenue par concaténation de clés étrangères correspondant aux entités liées

- un **Modèle Logique des Traitements** (ou MLT), qui précise les acteurs et les moyens qui seront mis en œuvre. C'est ici que les traitements sont découpés en procédures fonctionnelles (ou PF).

Comme son nom l'indique, l'étude d'organisation s'attache à préciser comment on organise les données de l'entreprise (MLD) et les tâches ou procédures (MLT). Pour autant, les choix techniques d'implémentation, tant pour les données (choix d'un SGBD) que pour les traitements (logiciel, progiciel), ne seront effectués qu'au niveau suivant.

- Le **modèle physique** reflète un choix matériel pour le système d'information.

III. différentes étapes suivies.

III.1 Analyse :

Notre champ d'étude s'articule sur le contrôle des réalisations.
Différentes règles de gestion :
Chaque fin du mois : les différents établissements transmettent l'avancement de leurs travaux via des documents à la direction.
Les employés de la direction qui s'occupent des réalisations procèdent à la saisie des informations selon le document « Réalisation », ce dernier concerne les prévisions et les réalisations de chaque établissement pour chaque commune.
Chaque fin d'année et à la fin de saisie des réalisations, une autre équipe procède à la réalisation du Document Bilan qui met en évidence les réalisations:

- De chaque article d'établissement pour toutes les communes, ces articles sont organisés en lots d'activités.
- Comparaison des réalisations et prévisions à travers les années.

Ces étapes sont modélisées par le Modèle Conceptuel des traitements (MCT) dans la phase de conception.

Les différents documents utilisés :
Document « Réalisation ».
Document « Bilan ».

« Document réalisation »

Etablissement:

Année :

Commune :

Lot d'activité	article	unité	prix	prévision	Réalisations												M A d'
					Jan	Fev	Mar	Avr	Mai	Jun	Jul	Aou	Sep	Oct	Nov	Dec	
Lot d'activité1	Article 1 Article 2																
Montant du lot d'activité 1																	
Lot d'activité2	Article 1 Article 2																
Montant du lot d'activité2																	

Réalisation Globale (Bilan)

Année: 2015 **Etablissement:**

Commune:

Lot d'Activité	Article	Unité	PrixUnitaire	Quantité Réalisée	Montant

= Montant

c.Critique de la situation existante :

- Les employés de la direction qui sont chargés de la réalisation du Bilan ne font que ressaisir les informations présentes dans les différents documents Réalisation, et ce sans pouvoir les exploiter.
- cette ressaisie des informations et non possibilité d'exploiter les informations existantes est due aux limites des fichiers.
- On constate aussi la redondance d'informations la non intégrité d'informations (différente entre la donnée saisie dans le document attachement et bilan) dus aussi à l'utilisation des fichiers.

Non exploitation optimale de l'outil informatique.

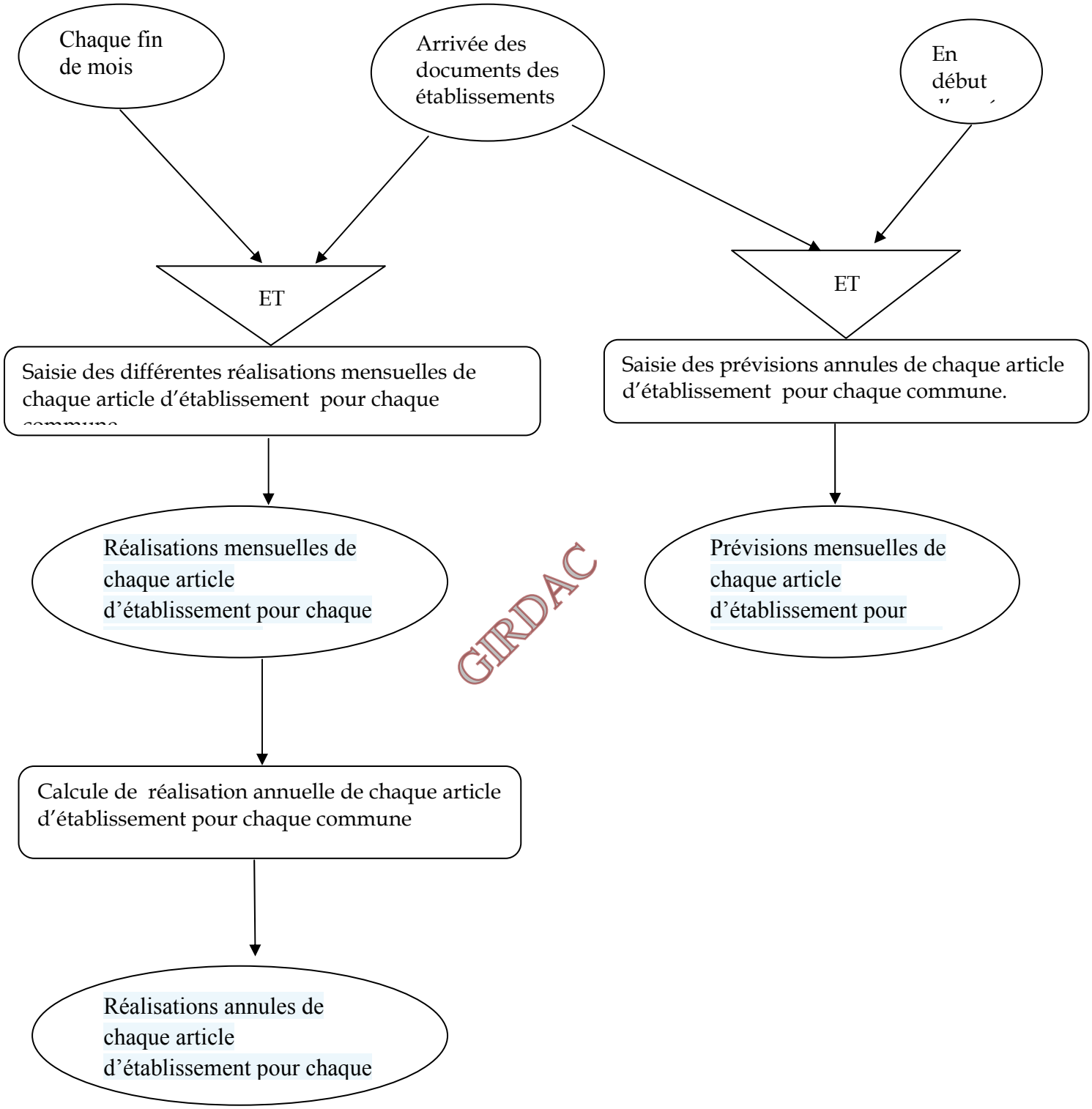
Touts les éléments précédents révèlent l'effort fourni pour la réalisation du bilan alors qu'il peut être réalisé par des programmes informatiques qui vont exploiter les données des réalisations.

III.2 Conception :

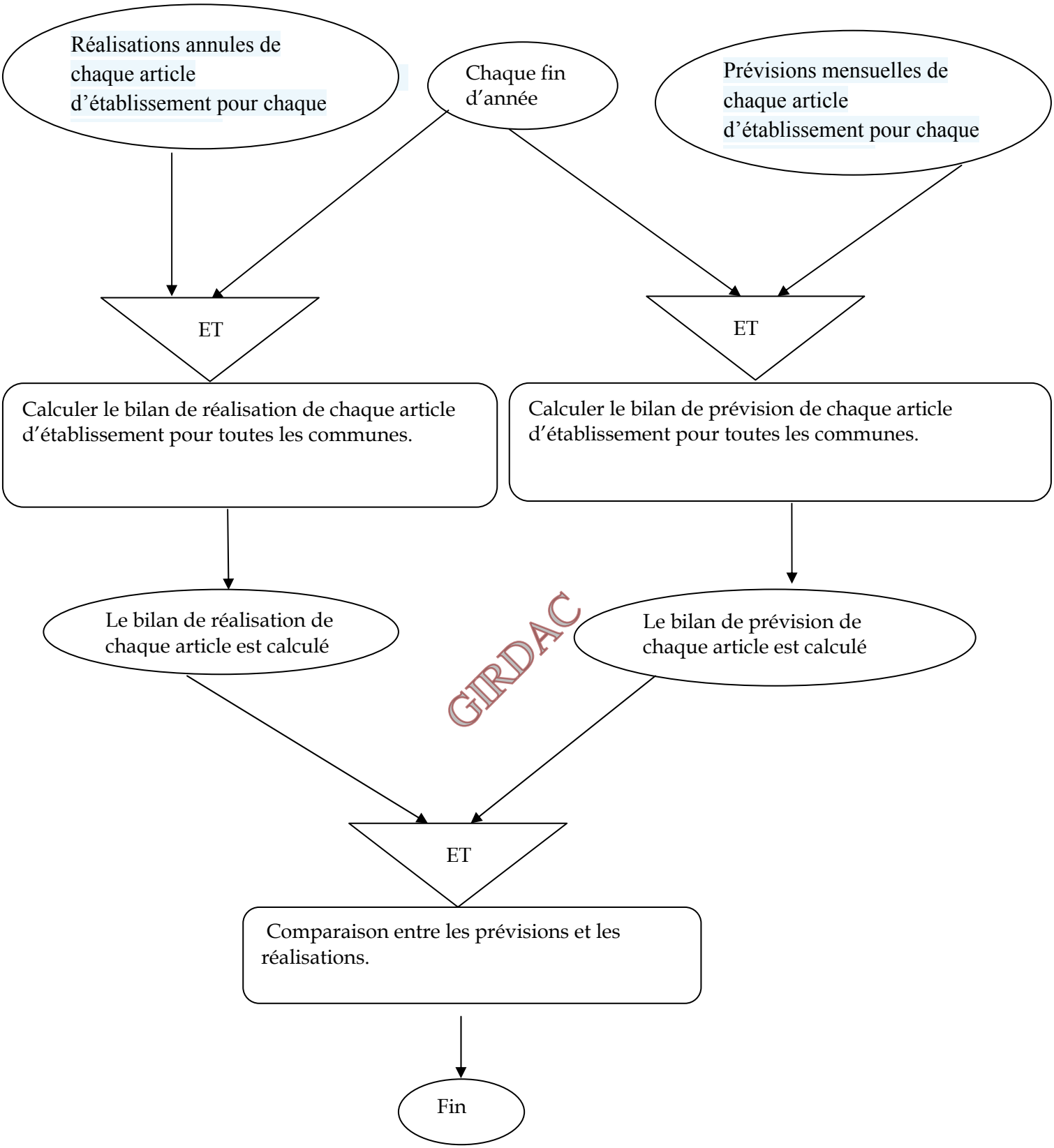
Solution proposée :

Réalisation d'une application comportant une base de données ;
Les documents réalisations sont remplacés par des formulaires de saisie.
Le document bilan par des formulaires d'affichage et les états de sortie.

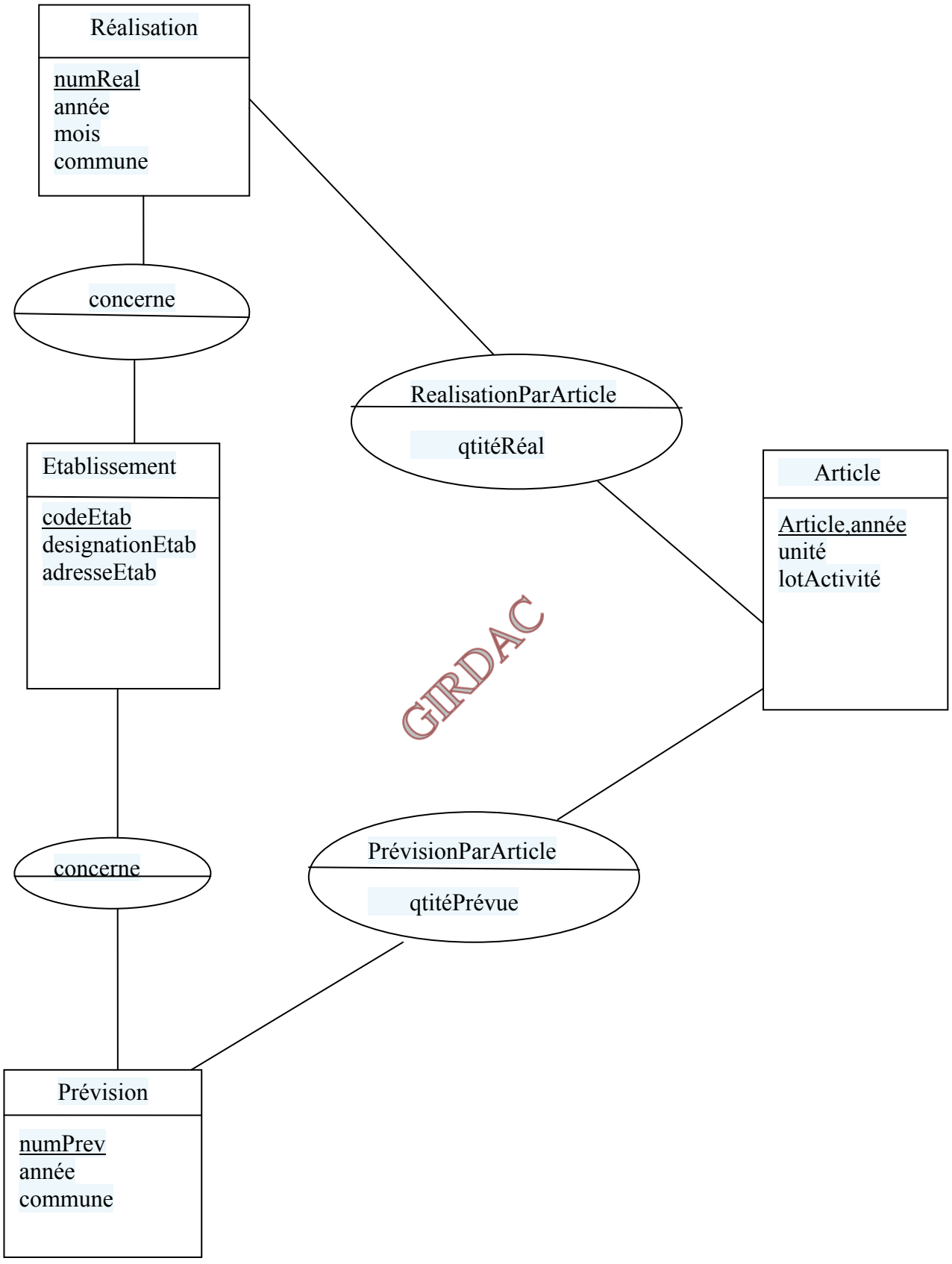
Le modèle conceptuel des traitements (MCT) : on distingue deux processus : processus saisie et processus réalisation bilan



Processus saisie



Processus Bilan



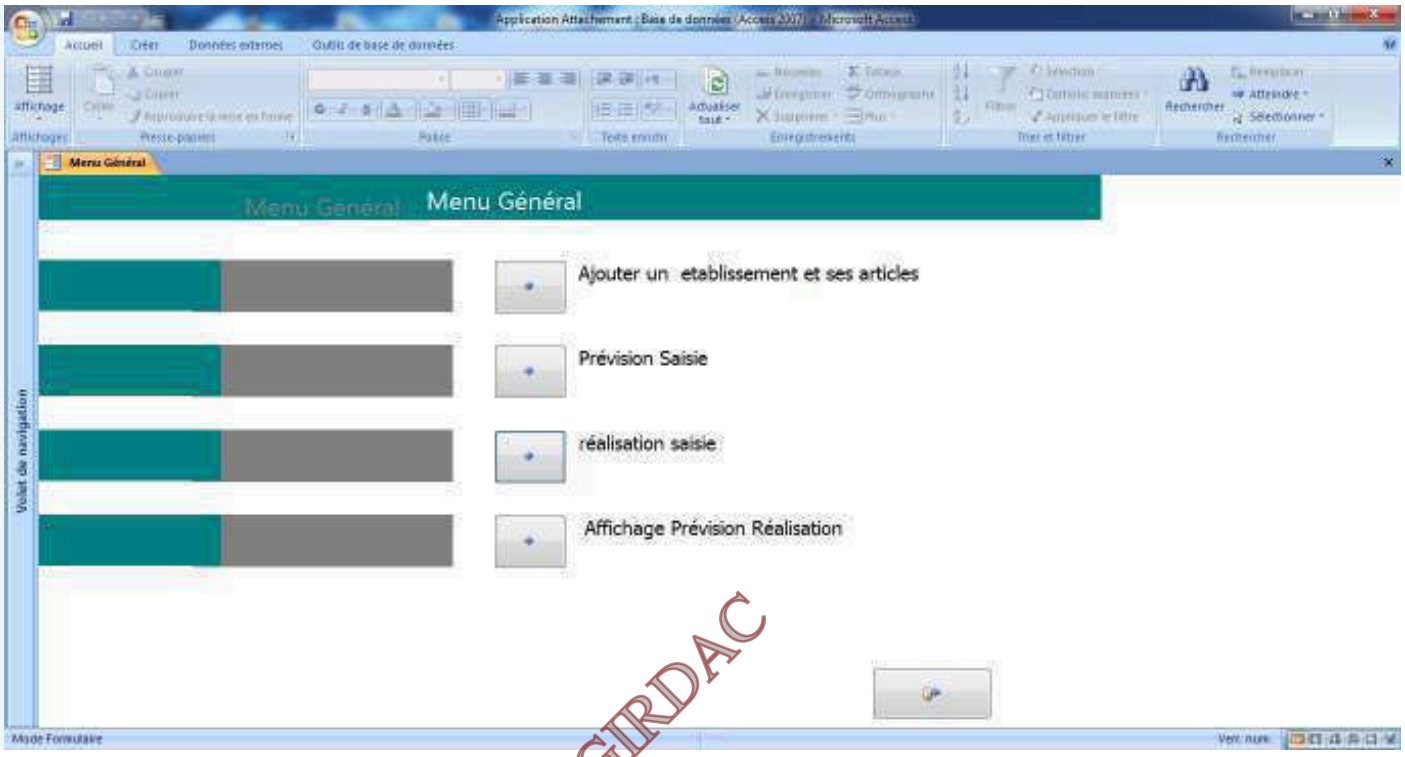
Le Modèle conceptuel des données

Le schéma relationnel :

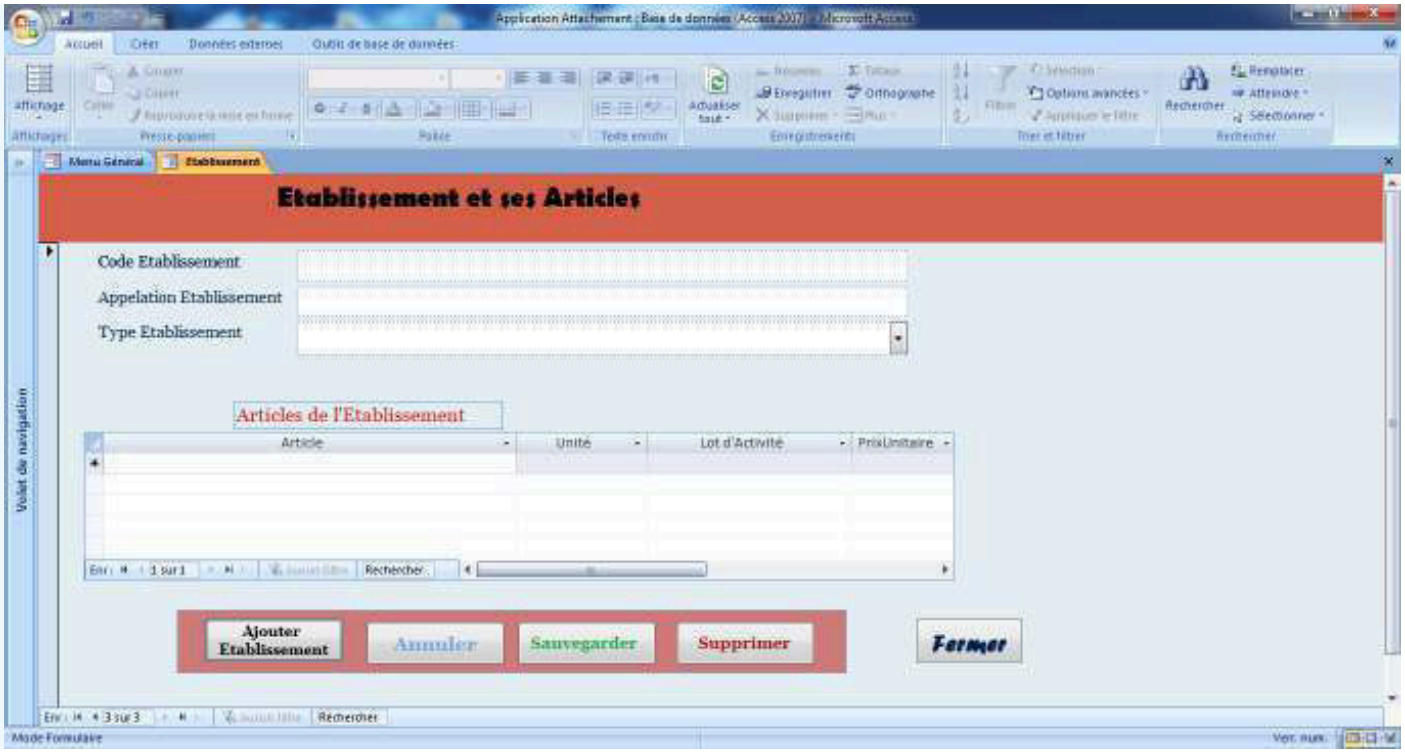
Etablissement (codeEtab, designationEtab, adresseEtab)
Article Etab (article, unité,lotActivite,codeEtab*)
PrixArticle (article*,Annee,prixUnitaire)
Prevision (numPrev,codeEtab, Annee, commune)
PrevisionParArticle (numPrev*,article, QtitéPrevue)
Realisation (numReal,codeEtab,annee,mois, commune)
RealisationParArticle (numReal, article, qtiteReal)

III.3 Réalisation :

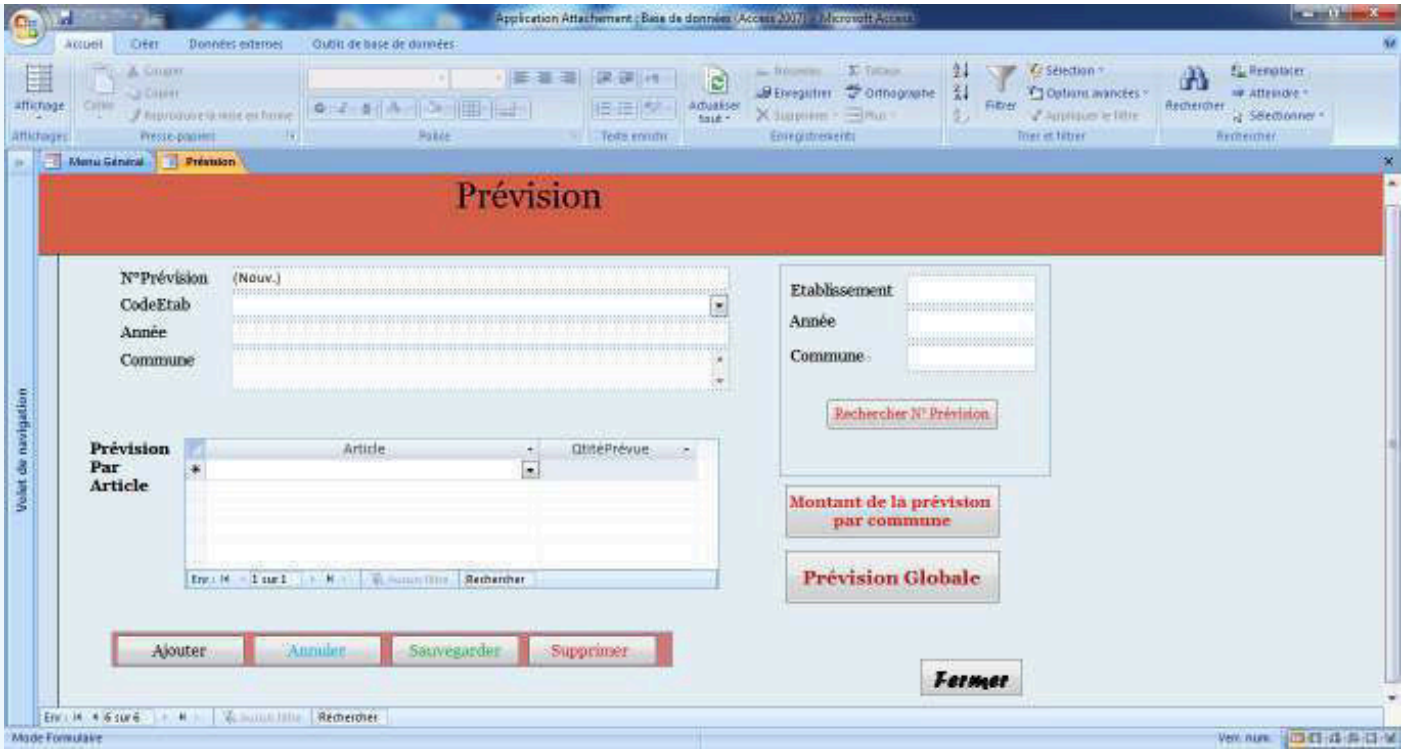
Différents Formulaires de l'application Acces2007 :



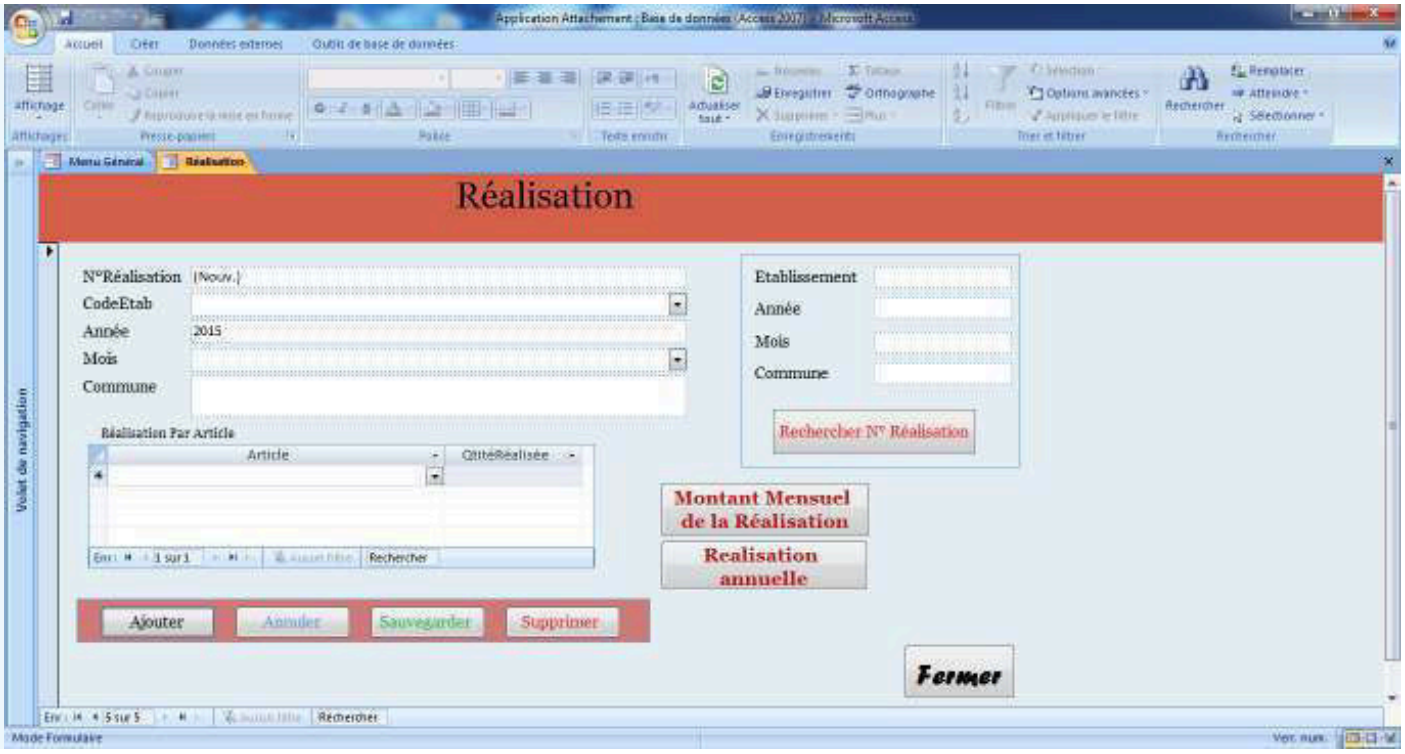
-Menu principal -



Formulaire Etablissement et ses articles



Formulaire prévision



Formulaire réalisation

IV. critique de la solution et brève description d'une nouvelle solution.

IV.1 critique de la solution proposée.

Comme le but initial n'était que de montrer la nécessité de faire le passage de l'utilisation des fichiers à l'utilisation des bases de données associée à une application. Ce but est atteint car le fait de montrer qu'on peut consulter le bilan sans avoir à le réaliser, et ce après avoir saisi que les données des attachements.
Mais la solution présente des insuffisances qui sont :
Utilisation de l'environnement Acces2007 non adéquat aux bases de données volumineuses.

IV.2 proposition d'une nouvelle solution.

Buts envisagés :
Améliorer l'application existante tout en palliant aux différentes lacunes citées précédemment.
Faire le passage de l'approche classique de conception des systèmes d'informations vers l'approche objet.
Utiliser un environnement de développement plus courant et plus riche en termes de fonctionnalité qui est java.
Améliorer ma capacité de conception et de développement d'applications.

Pour se faire on a procédé comme suit :

- Utilisation de l'approche orienté objet au lieu de l'approche classique qui n'est pas pratique pour l'ajout de nouvelles fonctionnalités et la maintenance des programmes.
- Modélisation avec le langage UML.
- Utilisation de la technologie client/serveur et comptes d'utilisateurs.
- Utilisation de langage JEE dans l'environnement Eclipse.
- Utilisation du SGBD MySQL au lieu du SGBD Access.

En effet, les fonctionnalités de la solution restent les mêmes, mais on ajoute les concepts cités précédemment tout en changeant l'environnement de développement.
Les détails de la nouvelle solution sont décrits dans la deuxième partie.

Parti II:
Réalisation d'un SI en utilisant l'approche orienté objet.

Introduction :

Les clients recherchent des applications évolutives, capables de répondre à leurs besoins changeants, de s'adapter à l'évolution constante de leurs processus métiers. Le développeur doit aussi être en mesure de réagir très rapidement, tout en faisant face à des délais de plus en plus serrés.
Dans ce contexte, la clé du succès est de pouvoir délivrer rapidement des applications personnalisées, en ne provoquant qu'un minimum de perturbations pour l'entreprise et son système d'Informations.

Le développement d'applications peut être facilité grâce à l'utilisation :
de la programmation orientée objet (POO) : facilite la conception, la maintenance, et la productivité. La notion de composant logiciel est introduite comme une suite à la notion d'objet. Le composant offre une meilleure structuration de l'application et permet de construire un système par assemblage de briques élémentaires en favorisant la réutilisation de ces briques.
et des design patterns lors de la conception :
Reconnu comme bonne pratique en réponse à un problème de conception d'un logiciel. Il décrit une solution standard, utilisable dans la conception de différents logiciels. Le nom du patron sert de vocabulaire commun entre le concepteur et le programmeur.
La solution apportée dans cette deuxième partie est réalisée en se basant sur la programmation orientée objet dans le cadre du langage de programmation java et la version Entreprise Edition. Tout en utilisant un patron de conception « Modèle Vue Contrôle ».
Cette partie est divisée en quatre chapitres ou :
Le premier chapitre s'articule sur l'architecture client-serveur.
Le deuxième chapitre présente la plate forme JEE ou es mis l'accent sur les patrons de conception utilisés par cet environnement et les différentes technologies utilisées.
Le troisième chapitre présente une brève description du langage de modélisation UML2.0.
Le quatrième chapitre est consacré à la présentation de la conception et la réalisation de la solution qui se base sur la conception de la première partie.

Chapitre I : Architecture client serveur [3][6][7].

I. Architecture client-serveur :

Définition : l'architecture client-serveur est un modèle de fonctionnement logiciel qui peut se réaliser sur tout type d'architecture matérielle (petites ou grosses machines), à partir du moment ou ces architectures peuvent être interconnectées.
On parle de fonctionnement logiciel dans la mesure ou cette architecture est basée sur l'utilisation de deux types de logiciels, à savoir un logiciel serveur et un logiciel client s'exécutant normalement sur deux machines différentes.
Chaque machine hébergeant le logiciel client ne communique qu'avec la machine hébergeant le logiciel serveur (n'a aucun lien avec ses homologues machines clientes) de la manière suivante :
Le client demande un service au serveur via une requête.
Le serveur réalise ce service et renvoie le résultat au client.

II. Différents modèles client-serveur :

En fait les différences essentiellement sont liées aux services assurés par le serveur :
On distingue les différents modèles du client-serveur suivants :

II.1 Client serveur de présentation :

Dans ce cas la présentation des pages affichés par le client est intégralement prise en charge par le serveur. Cette organisation présente l'inconvénient de générer un fort trafic réseau.

II.2 Client serveur de traitement :

Dans ce cas, le serveur effectue des traitements à la demande du client. Il peut s'agir de traitement particulier sur des données, de vérification de formulaires de saisie, du traitement d'alarmes...
Ces traitements peuvent être réalisés par des programmes installé sur des serveurs mais également intégrés dans des bases de données (Triggers, Procédures Stockées), dans se cas, la partie, donnée et traitement sont intégrées.

II.3 Client serveur de données :

Dans ce cas, le serveur assure des tâches de gestion, de stockage et de traitement de données. C'est le cas le plus connu de client-serveur qui est utilisé par tous les grands SGBD :
La base de données avec tous ses outils (maintenance, sauvegarde...) est installée sur un poste serveur.
Tous les traitements sur les données sont effectués sur le serveur qui renvoie les informations demandées (souvent à travers une requête SQL par le client.

III. Différentes architectures client-serveur :

Les variantes de l'architecture sont liées essentiellement aux nombre de niveaux de services présents dans l'architecture. On distingue les trois variantes suivantes :

III.1 Architecture 2 tiers : encore appelée architecture client-serveur de première génération, caractérise les systèmes clients/serveurs dans lesquels le client demande une ressource et le serveur la lui fournit directement. Cela signifie que le serveur ne fait pas appel à une autre application afin de fournir le service. Cette architecture offre des interfaces conviviales aux clients et organisée comme suit :

- L'ensemble des interfaces et toute la logique applicative est mis sur la partie client.
- L'ensemble des données est mis sur le serveur.

L'expérience a démontré qu'il était couteux et contraignant de vouloir faire porter l'ensemble des traitements applicatifs par le poste client. On en arrive aujourd'hui à ce que l'on appelle le client lourd, avec un certain nombre d'inconvénients :

On ne peut pas soulager la charge du poste client. Qui supporte la grande majorité des traitements applicatifs.

Ce type d'architecture est grandement rigidifié par les couts et la complexité de sa maintenance.

Pour résoudre les limitations du client-serveur 2-tiers tout en conservant ses avantages, on a cherché une architecture plus évoluée, facilitant les forts déploiements à moindre coût. La réponse est apportée par les architectures distribuées.

III.2 Architecture 3 tiers :

Cette architecture trois tiers, également appelée client-serveur de deuxième génération ou client-serveur distribué sépare l'application en 3 niveaux de services distincts, conformes au principe précédent.

La solution résiderait donc dans l'utilisation d'un poste client simple communicant avec le serveur par le biais d'un protocole standard. Dans ce but, l'architecture trois tiers applique les principes 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.

De ce fait l'architecture à 3 niveaux, est partagée entre:

- **Le client**

le demandeur de ressources

- **Le serveur d'application**

(appelé aussi *middleware*) le serveur chargé de fournir la ressource mais faisant appel à un autre serveur

- **Le serveur secondaire**

(Généralement un serveur de base de données), fournissant un service au premier serveur.

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 s'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(le serveur d'application peut s'exécuter sur la même machine que le SGBD).

La fiabilité et les performances ainsi que la facilité de traitements de certains traitements se trouvent améliorées par leur centralisation.

Dans l'architecture 3- tiers, le poste client est communément appelé client léger, par opposition au client lourd des architecture deux tiers, il ne prend en charge que la présentation de l'application avec, éventuellement, une partie de logique applicative permettant une vérification immédiate de la saisie et la mise en forme des donnée.

Le serveur de traitement constitue la pierre angulaire de l'architecture et se trouve souvent fortement sollicité. Dans ce type d'architecture, il est difficile de répartir la charge entre client et serveur. On se retrouve confronté aux épineux problèmes de dimensionnement serveur et de gestion de la montée en charge rappelant l'époque des mainframes, de plus, les solutions mises en œuvre sont relativement complexes à maintenir et la gestion des sessions est compliquée. Les contraintes semblent inversées par rapport à celles rencontrées avec l'architecture deux tiers : le client est soulagé, mais le serveur est fortement sollicité.

III.3 Architecture n tiers :
L'architecture n-tiers a été pensée pour pallier aux limitations des architectures des trois tiers et concevoir des applications puissantes et simples à maintenir. Ce type d'architecture permet de distribuer plus librement la logique applicative, ce qui facilite la répartition de la charge entre tous les niveaux.
Cette évolution des architectures des trois tiers met en œuvre une approche objet pour permettre une plus grande souplesse d'implémentation et faciliter la réutilisation.
L'appellation « n-tiers » pourrait faire penser que cette architecture met en œuvre un nombre indéterminé de niveaux de service, alors que ces derniers sont au maximum trois. En fait, l'architecture n-tiers qualifie la distribution d'application entre multiples services et non la multiplication de niveaux de services.
Cette distribution est facilitée par l'utilisation de composants 'métier', spécialisés et indépendants, introduits par les concepts orientés objets. Elle permet de tirer pleinement partie de la notion de composants métiers réutilisables.
La distribution des services applicatifs facilite aussi l'intégration de traitements existants dans les nouvelles applications.
Ces nouveaux concepts sont basés sur la programmation objet ainsi que sur des communications standards entre applications. Ainsi est né le middleware objet **[1]**.

IV. Middleware :
Un middleware désigne toutes les couches logicielles qui permettent aux applications de communiquer à distance. Il représente une intersection entre plusieurs domaines de l'informatique : Systèmes d'exploitation, réseaux, langages de programmation, systèmes d'exploitation répartis. Le middleware est un outil pour le développeur, enfoui dans les applications et qui n'est pas visible à l'utilisateur final. Il est considéré comme un complément de services réseau permettant la réalisation du dialogue client/serveur.

L'objectif principal du Middleware est d'unifier pour les applications, l'accès et la manipulation de l'ensemble des services disponibles sur le réseau, afin de rendre l'utilisation de ces derniers presque transparente.

IV.1 Services offerts par un middleware :
Un Middleware est susceptible de rendre les services suivants :

Conversion : Service utilisé pour la communication entre machines mettant en œuvre des formats de données différents.
Adressage : Permet d'identifier la machine serveur sur laquelle est localisé le service demandé afin d'en déduire le chemin d'accès. Dans la mesure du possible.
Sécurité : Permet de garantir la confidentialité et la sécurité des données à l'aide de mécanismes d'authentification et de cryptage des informations.
Communication : Permet la transmission des messages entre les deux systèmes sans altération. Ce service doit gérer la connexion au serveur, la préparation de l'exécution des requêtes, la récupération des résultats et la déconnexion de l'utilisateur.

Le middleware masque la complexité des applications des échanges inter-applications et permet ainsi d'élever le niveau des API utilisées par les programmes. Sans ce mécanisme, la programmation d'une application client-serveur serait complexe et difficilement évolutive **[1]**.

IV.2 Types de Middleware :
Ils sont de plusieurs types, les principaux étant :
Les middlewares d'accès aux données : Ils mettent en communication les applications avec les différentes sources de données de l'entreprise *exemple : JDBC, ODBC drivers de bases de données relationnelles.*

Les moniteurs transactionnels (par exemple, Tuxedo). Leur fonction est de coordonner des transactions distribuées sur plusieurs machines.

Les Message Oriented Middleware (ou MOM). Il s'agit serveurs qui servent d'intermédiaire dans la communication entre applications Ils peuvent stocker des messages dans une file d'attente et attendre que l'application destinataire soit prête à recevoir le message qui lui est dressé. Les MOM offrent aussi une bonne résistance aux interruptions de services **[3]**.

Middleware objet dit ***Object Request Broker*** **(Cobra, DCOM, ou RMI):** constitué d'un ensemble d'une série de mécanismes permettant à un ensemble de programmes d'inter opérer de façon transparente. Les services offerts par les applications serveurs sont présentés aux clients sous la forme d'objets. La localisation et les mécanismes mis en œuvre pour cette interaction sont cachés par le middleware **[1]**.

La communication entre objets gomme la différence entre ce qui est local ou distant. Les appels de méthodes d'objets à objet sont traités par un mécanisme se chargeant d'aiguiller les messages vers les objets (locaux ou distants).

V. Différents types de serveurs :

Profitant des développements technologiques (système d'exploitation multitâches pour les postes de travail, évolution des protocoles réseau) et d'autres touchant les architectures de traitement et de stockage des informations, La complexité du réseau évolue aussi d'une structure n clients vers 1 serveur en une structure n clients vers p serveurs, chacun des serveurs étant censé jouer un rôle particulier.
Les experts parlent d'architecture distribuée et éditent des normes pour assurer l'interopérabilité (DCE de l'OSF, en clair Distributed Computing Architecture de l'Open Software Fondation).
Les serveurs peuvent être dédiés à plusieurs tâches spécialisées : **[2]**

- **Serveurs de fichiers**

Les serveurs de fichiers : Dans le cas d'un *serveur de fichiers*, le client requiert des enregistrements de fichiers en émettant des requêtes sur un réseau en direction d'un serveur
fichiers (il s'agit d'une forme très primitive de serveur de données, qui nécessite de nombreux échanges de messages sur le réseau avant d'obtenir l'information demandée).

- **Serveurs de bases de données**

Concernant le *serveur de bases données*, le client émet des requêtes SQL sous forme de messages en direction du serveur. Le résultat de chaque requête SQL est renvoyé sur le réseau. Le code qui traite la requête ainsi que les données résident sur la même machine. Le serveur utilise sa propre capacité de traitement pour rechercher les données demandées au lieu de transmettre tous les articles au client et de laisser ce dernier les traiter, comme c'est le cas du serveur de fichiers.

- **Serveurs de transactions**

Dans ce modèle, le client invoque des procédures (ou services) résidantes sur le serveur qui comporte un moteur de base de données SQL. Ces procédures distantes du serveur exécutent un ensemble d'instructions SQL. L'´echange sur le réseau consiste en un seul message de requête/réponse (contrairement à l'approche serveur de bases de données pour laquelle il existe un message de requête/réponse pour chaque instruction SQL). Le succès ou l'échec de l'opération concerne l'ensemble des instructions SQL.
Cet ensemble est appelé transaction.
Avec un serveur de transactions, l'application client/serveur nécessite qu'on écrive du code pour les deux composants client et serveur. Le coté client comprend généralement une interface graphique utilisateur (GUI). Le coté serveur contient l'ensemble des transactions SQL sur une base de données.

- **Serveurs d'applications objet**

Dans le cas d'un serveur d'objets, l'application client/serveur est écrite sous forme d'un jeu d'objets communicants. Les objets client communiquent avec des objets serveur au moyen d'un courtier d'objet ou

ORB (Objet Request Brocker). Le client invoque une méthode sur un objet distant. L'ORB localise une instance de la classe, appelle la méthode demandée et renvoie les résultats à l'objet client. Les serveurs d'objets doivent assurer le traitement de la simultanéité et du partage.

- **Serveurs groupeware**

Le groupeware s'intéresse à la gestion d'informations semi-structurées telle que le texte, l'image, le courrier ou la messagerie. Ces systèmes client/serveur mettent les individus en contact direct les uns avec les autres.
Serveurs Web

- **Le World Wide Web** est le modèle client/serveur qui consiste en des clients, légers, portables et "universels" qui communiquent avec de très gros serveurs.

Dans la concrétisation la plus simple, un serveur Web renvoie des documents lorsque le client les demande par leur nom.
Les clients et les serveurs communiquent par un protocole RPC (Remote Procedure Call) appelé HTTP**[3].**

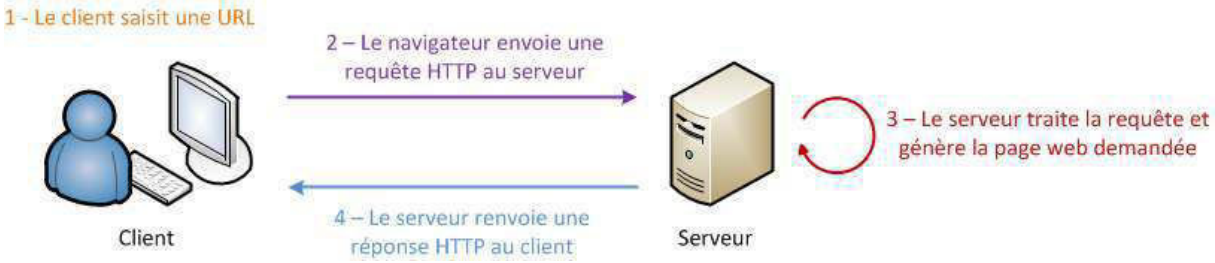
VI. Web et http :

VI.1 Différence entre internet et le web :

Avant tout, il ne faut pas confondre l'internet et le web :
L'internet est le réseau, le support physique de l'information. Pour faire simple, c'est un ensemble de machines, de câbles et d'éléments réseau en tout genre éparpillés sur la surface du globe.
Le web constitue une partie seulement du contenu accessible sur l'internet. Vous connaissez et utilisez d'autres contenus, comme le courrier électronique ou encore la messagerie instantanée.

Un site web est un ensemble constitué de pages web (elles-mêmes faites de fichiers HTML, CSS, Javascript, etc.). Lorsqu'on développe puis publie un site web, on met en réalité en ligne du contenu sur internet. On distingue deux types de sites :

Les sites internet statiques : ce sont des sites dont le contenu est « fixe », il n'est modifiable que par le propriétaire du site. Ils sont réalisés à l'aide des technologies HTML, CSS et Javascript uniquement.
Les sites internet dynamiques : ce sont des sites dont le contenu est « dynamique », parce que le propriétaire n'est plus le seul à pouvoir le faire changer ! En plus des langages précédemment cités, ils font intervenir d'autres technologies : Java EE est l'une d'entre elles.
Lorsqu'un utilisateur consulte un site, La communication entre le client et le serveur s'effectue, et elle est régit par des règles bien définies : le **protocole HTTP** (voir la figure suivante). Cet echange est constitué des étapes suivantes :
L'utilisateur saisit une URL dans la barre d'adresses de son navigateur.
Le navigateur envoie alors une **requête HTTP** au serveur pour lui demander la page correspondante.
Le serveur reçoit cette requête, l'interprète et génère alors une page web qu'il va renvoyer au client par le biais d'une **réponse http**.
le navigateur reçoit, via cette réponse, la page web finale, qu'il affiche alors à l'utilisateur.



-Echange dynamique HTTP client / serveur –

VI.2 Le protocole http :
La communication entre le client et le serveur s'effectue via le protocole http.
Le client utilise essentiellement trois méthodes du protocole http (GET, POST, HEADER) pour demander des pages web au serveur via une URL, le serveur retourne la ressource demandée tout en l'accompagnant d'autres informations diverses (comme la longueur des données envoyées, la date d'envoi) dans les entêtes ou headers.
GET : la transmission des données en utilisant la requête GET s'effectue au travers des paramètres directement placés après l'URL, ou de cookies placés dans les entêtes de la requête.
comme la taille d'une URL est limitée, **on ne peut pas utiliser cette méthode pour envoyer des données volumineuses au serveur**, par exemple un fichier.
il est uniquement possible de **récupérer ou de lire** des informations, sans que cela ait un quelconque impact sur la ressource demandée : ainsi, une requête GET est censée pouvoir être répétée indéfiniment sans risques pour la ressource concernée.
POST : La taille du corps du message d'une requête POST n'est pas limitée. C'est cette requête qui est utilisée pour transmettre des données volumineuses ou de tailles variables.
cette méthode doit être utilisée pour réaliser les opérations qui ont un effet sur la ressource, et qui ne peuvent par conséquent pas être répétées sans l'autorisation explicite de l'utilisateur.
Affichage d'un message d'alerte de la part du navigateur prévenant qu'un rafraîchissement de la page entraînera un renvoi des informations et demandant confirmation avant de renvoyer à nouveau la requête, suite à l'actualisation de la page.
HEADER : le serveur envoie seulement les informations concernant la ressource demandée dites entêtes HTTP, sans envoyer cette dernière. Il est ainsi possible par exemple de vérifier la validité d'une URL ou de vérifier si le contenu d'une page a changé ou non.

A retenir que :
Les données sont échangées entre le client et le serveur via le protocole HTTP .
Le client ne comprend que les langages de présentation de l'information, en d'autres termes les technologies HTML, CSS et Javascript.
les pages sont générées sur le serveur de manière dynamique, à partir du code source du site qui constitué des langages capables de traiter les informations.

Il existe plusieurs technologies capables de traiter les informations sur le serveur. Java EE est l'une d'entre elles, mais il en existe d'autres : PHP, .NET, Django et Ruby on Rails, pour ne citer que les principales. Toutes offrent sensiblement les mêmes possibilités, mais toutes utilisent un langage et un environnement bien à elles.
Dans le chapitre suivant serait détaillée une de ces technologies capables de traiter les informations sur le serveur qui est la technologie JEE.

Ce chapitre est consacré au modèle client /serveur comme l'application à réaliser est de type client/serveur.
Ce chapitre est débuté par la définition de l'architecture client/serveur.
Ensuite sont présentés

- *les différents modèles du client-serveur qui sont distingués par les services assurés par le serveur.*
- *les différentes variantes de l'architecture qui dépendent du nombre de niveaux de service (2 et3 tiers) et la distribution des services (architecture n-tiers).*
- *Middleware dont le rôle est d'unifier, pour les applications, l'accès et la manipulation de l'ensemble des services disponibles sur le réseau, afin de rendre l'utilisation de ces derniers presque transparente.ses services et les types de middlewares.*
- *Différents types de serveurs.*
- *Et enfin le serveur web et le protocole http.*

Chapitre II : plate forme JEE.[4]

Introduction :

Java EE est une extension de la plate-forme standard Java SE, principalement destinée au développement d'applications web.

Le *Java Enterprise Edition*, comme son nom l'indique, a été créé pour le développement d'applications d'entreprises. Ses spécifications ont été pensées afin, notamment, de faciliter le travail en équipe sur un même projet : l'application est découpée en couches, et le serveur sur lequel tourne l'application est lui-même découpé en plusieurs niveaux.

Car en entreprise on pourrait être amené à :

- travailler à plusieurs contributeurs sur un même projet ou une même application (travail en équipe) ;
- maintenir et corriger une application que l'on n'a pas créée soi-même ;
- faire évoluer une application que l'on n'a pas créée soi-même.

Pour toutes ces raisons, il est nécessaire d'adopter une architecture plus ou moins standard, que tout développeur peut reconnaître, c'est-à-dire dans laquelle tout développeur sait se repérer.

Un des patrons de conception répondait à ces besoins qui est MVC (Modèle – Vue – Contrôleur).

La création d'une application web avec Java EE s'effectue généralement à l'aide d'un Environnement de Développement Intégré (IDE en Anglais).

I. Environnement de développement :

a. Définition

C'est un logiciel destiné à faciliter grandement le développement dans son ensemble. Il est caractérisé par :

- L'intégration des outils nécessaires au développement et au déploiement d'une application.
- paramétrage aisé et centralisé des composants d'une application.
- multiples moyens de visualisation de l'architecture d'une application.
- génération automatique de portions de code.
- assistance à la volée lors de l'écriture du code.
- outils de débogage…

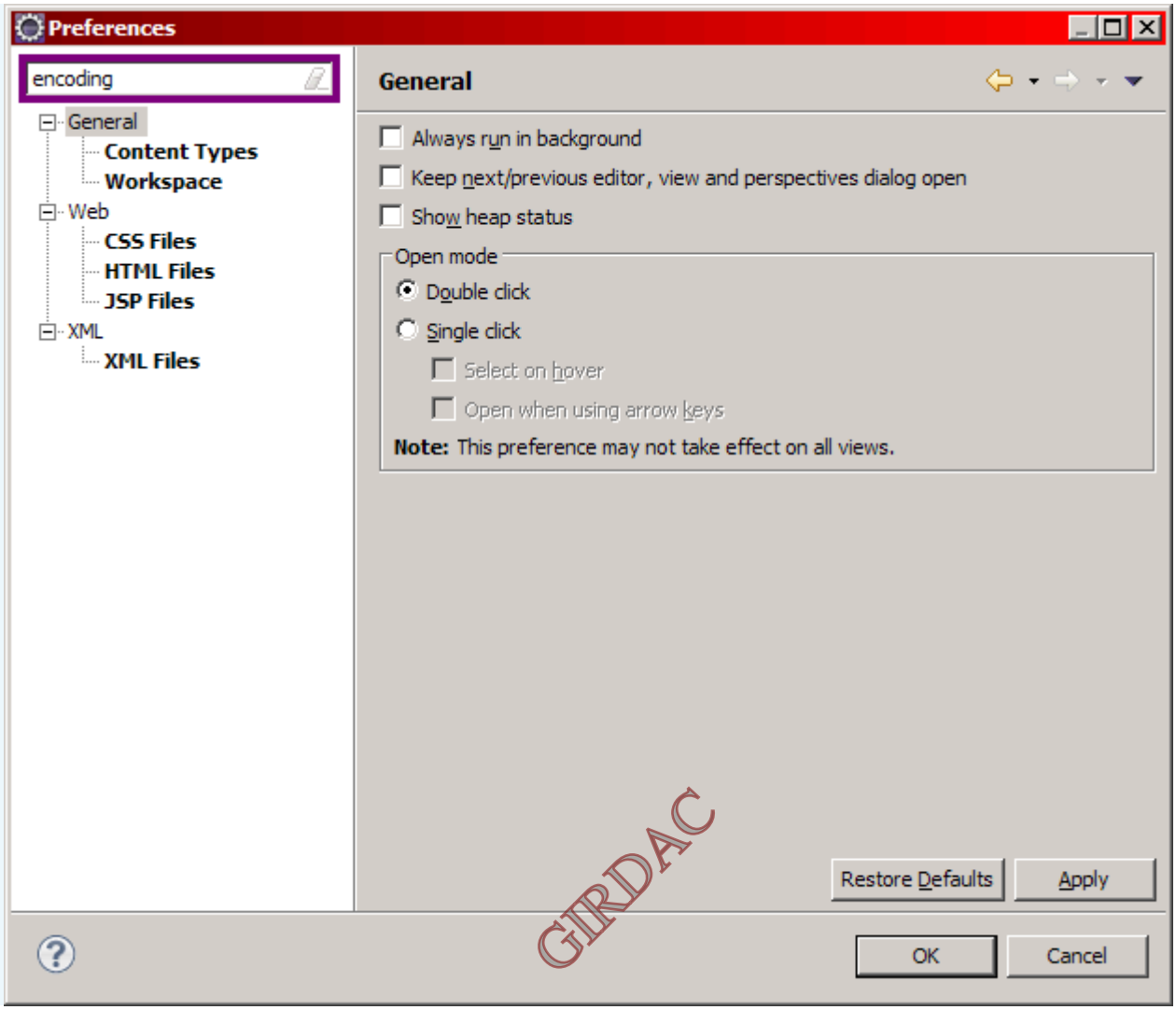
b. IDE Eclipse : Eclipse est un IDE gratuit, libre et multiplateforme ; il existe une version Eclipse pour java standard, une version pour java EE et bien d'autres versions pour d'autres environnements.

b.1 Paramétrage de l'IDE Eclipse :

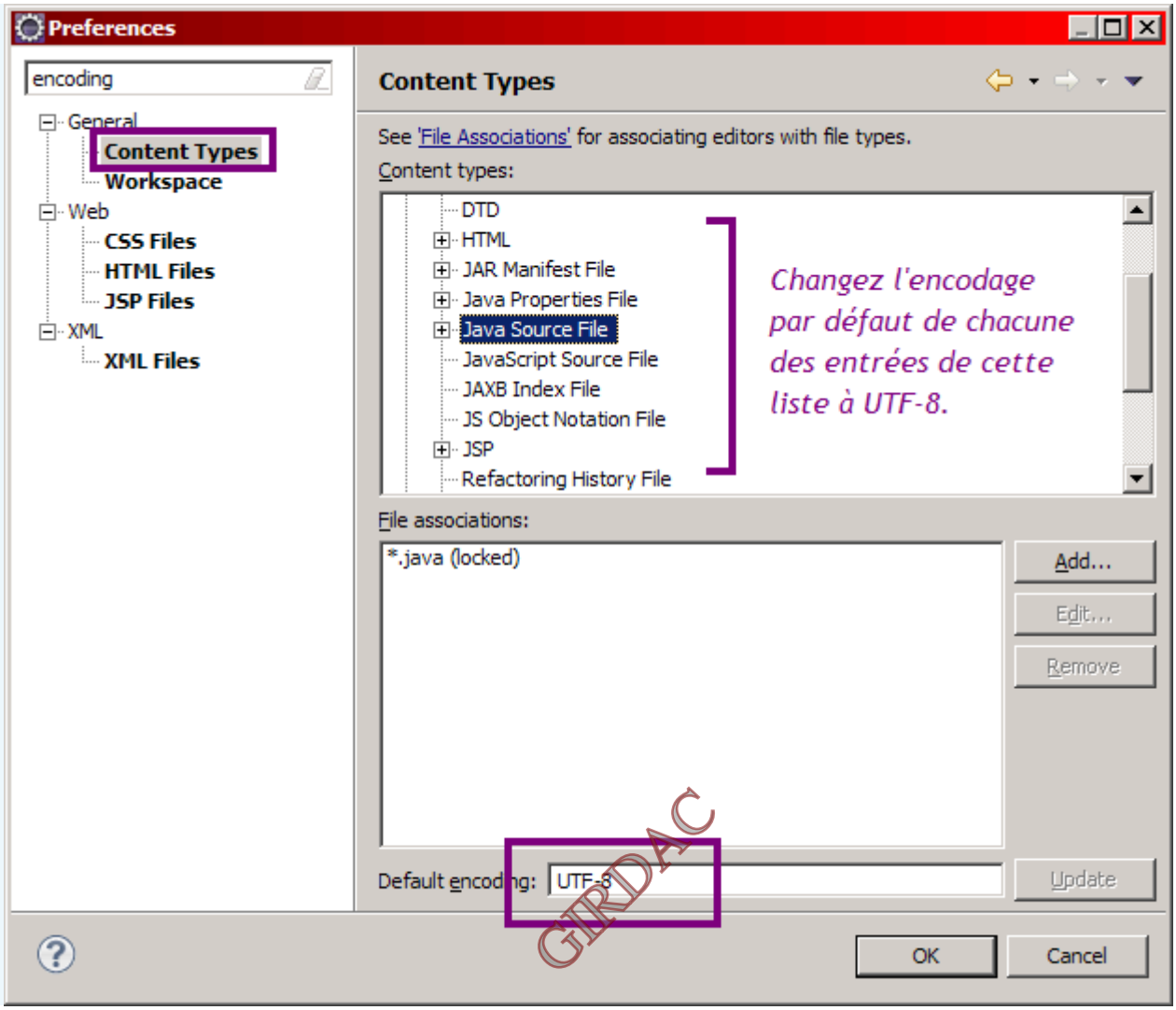
Il est préférable d'installer Eclipse dans un répertoire situé directement à la racine du disque dur, dont le titre ne contient ni espace ni caractères spéciaux. Ensuite effectuer les configurations suivantes :

b.1.1 Modification de l'encodage par défaut :

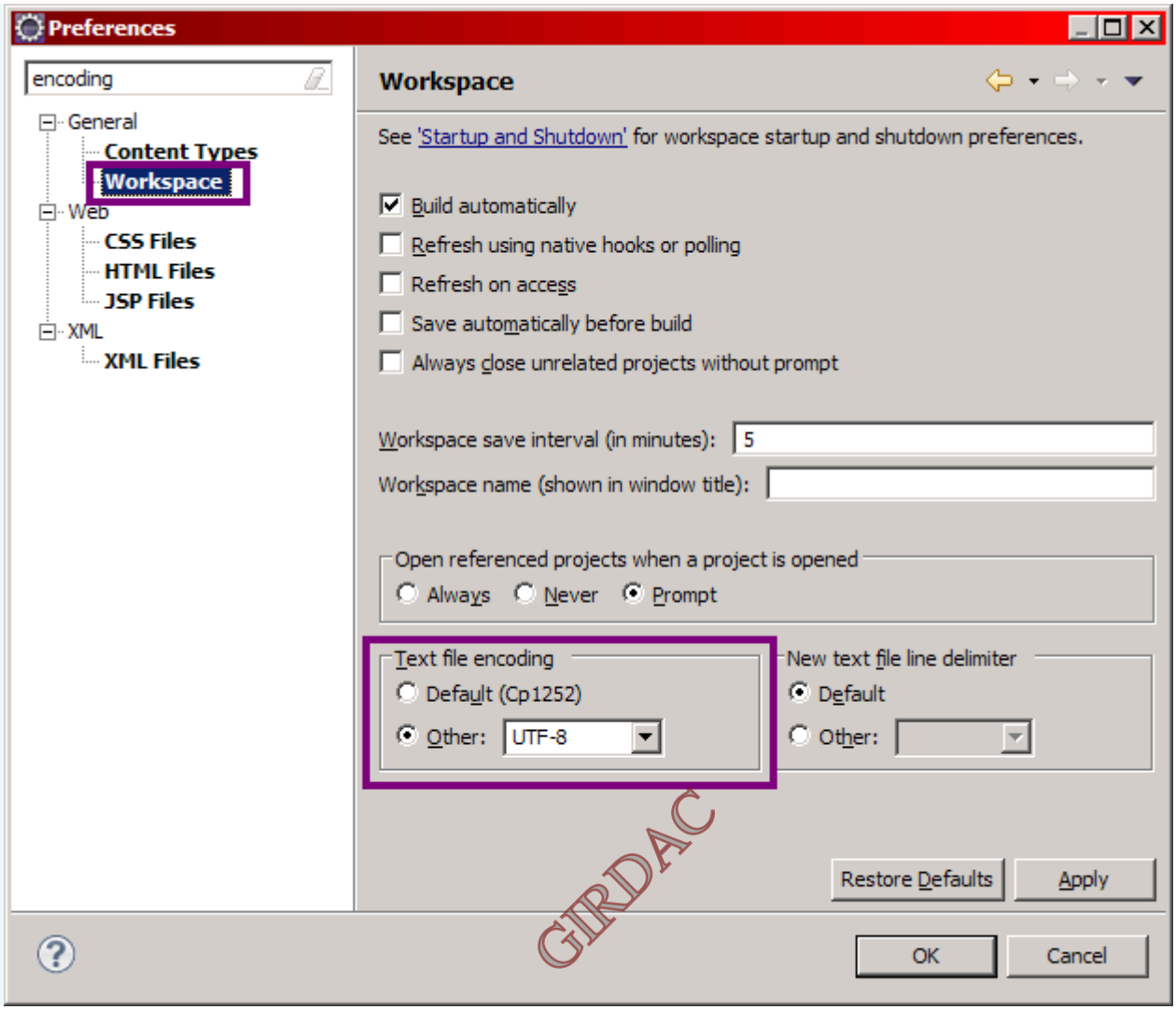
dans la barre des menus supérieur, cliquer sur Window, puis Préférences. Aller vers encoding dans le volet à gauche, et dans chaque section qui apparait, changer la valeur par défaut par la valeur UTF-8. Validez pour finir en cliquant sur **Ok**



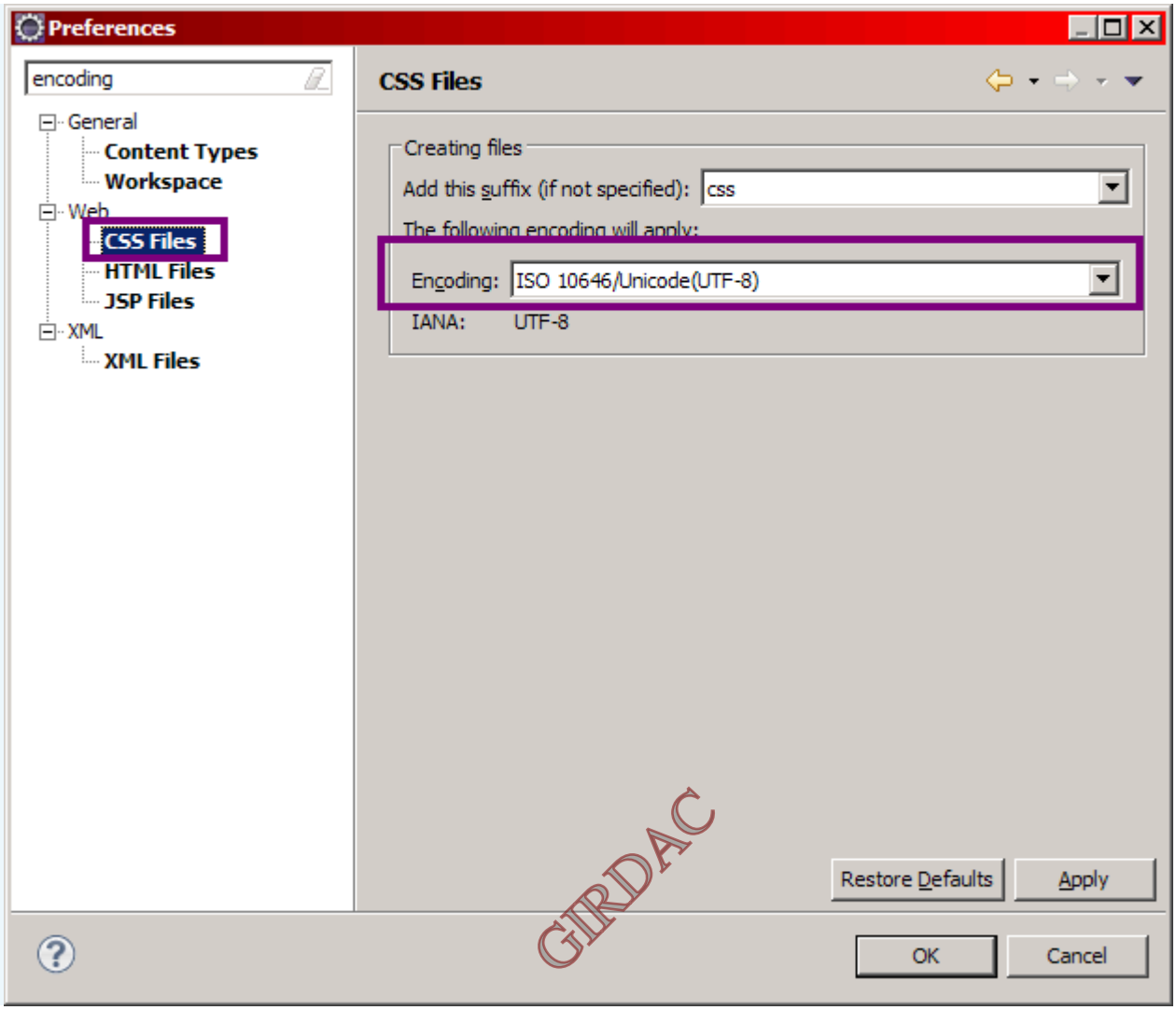
Modification de l'encodage par défaut 1



Modification de l'encodage par défaut 2



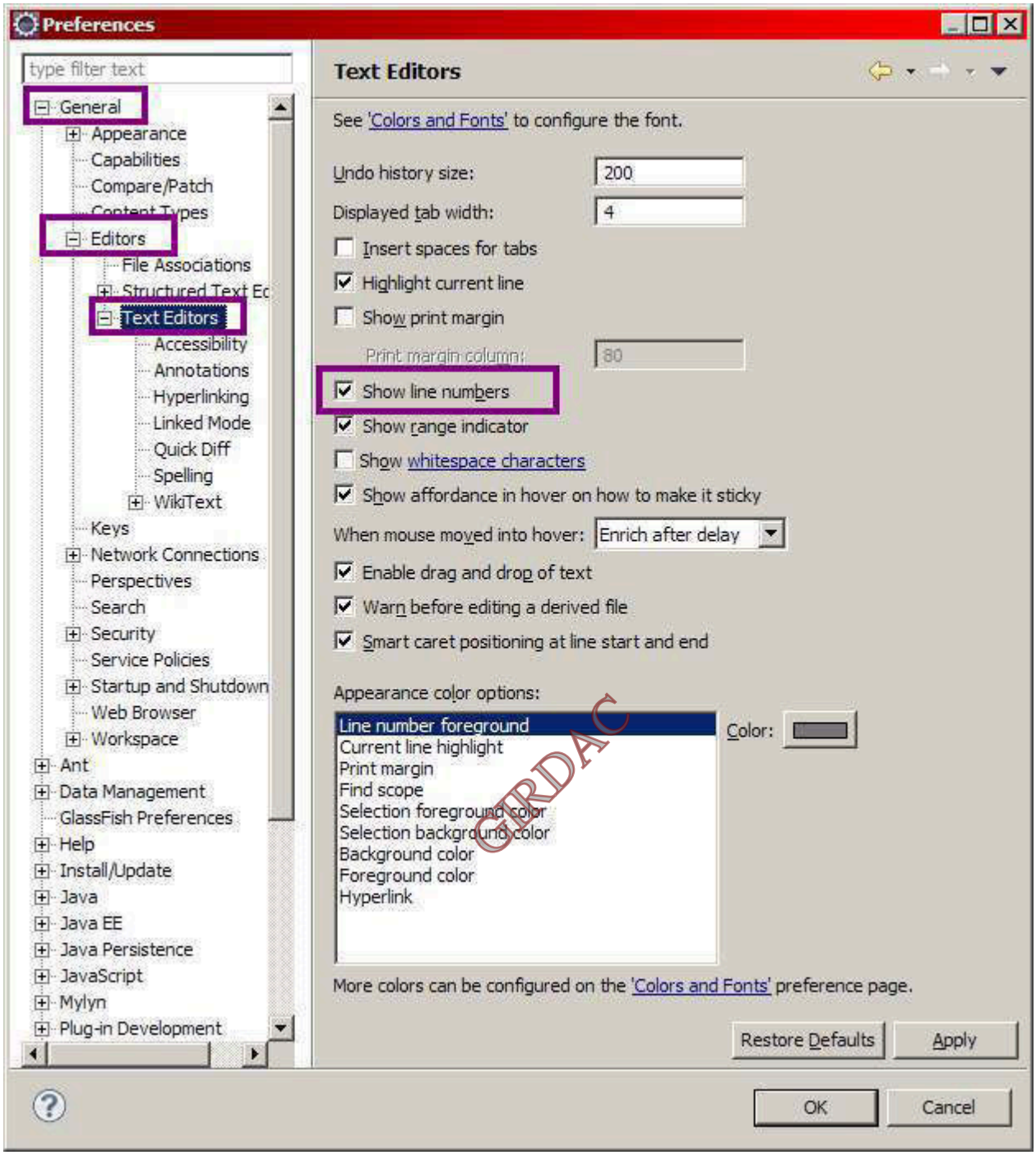
Modification de l'encodage par défaut 3



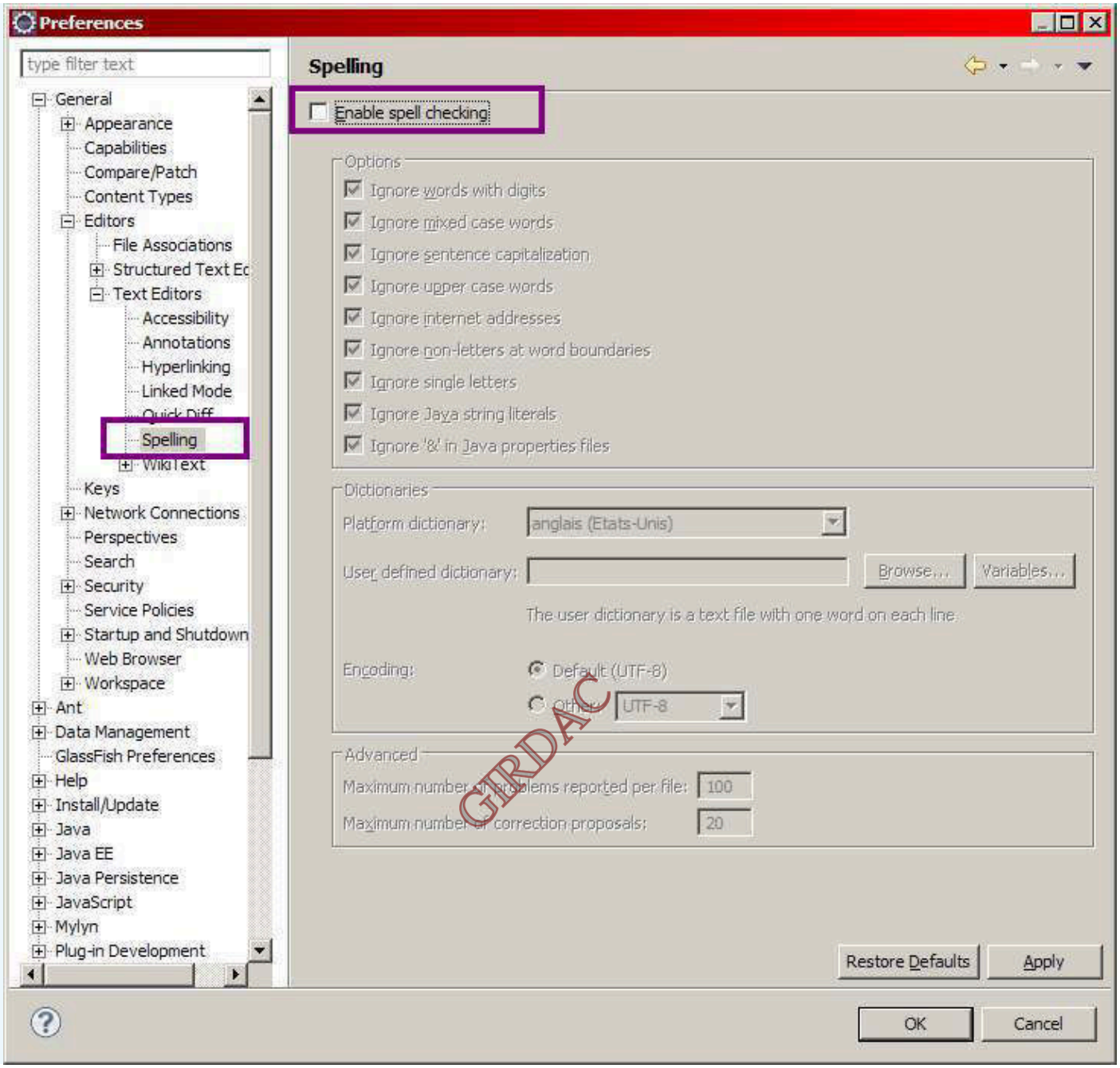
Modification de l'encodage par défaut 4

b.1.2 Désactivation de la vérification de l'orthographe :

Dans le menu Window > Preferences, puis dans le volet de gauche cliquer sur General > Editors > Text Editors, et dans le volet de droite cocher la case "Show line numbers". Dans le volet de gauche, cliquer sur le sous-menu Spelling, et dans le nouveau volet de droite qui apparaît, décocher la case "Enable spell checking". Valider r en cliquant sur **Ok** .



Désactivation de la vérification de l'orthographe1



Désactivation de la vérification de l'orthographe2

Pour faire fonctionner une application web Java EE, nous avons besoin de mettre en place un **serveur d'applications**.

c.Le serveur Tomcat : c'est un serveur léger, gratuit, libre, multiplateforme.

Tout comme Eclipse, Il est préférable de l'installer dans un répertoire situé directement à la racine du disque dur, dont le titre ne contient ni espace ni caractères spéciaux.

Le répertoire d'installation de Tomcat contient différents dossiers, essentiellement :

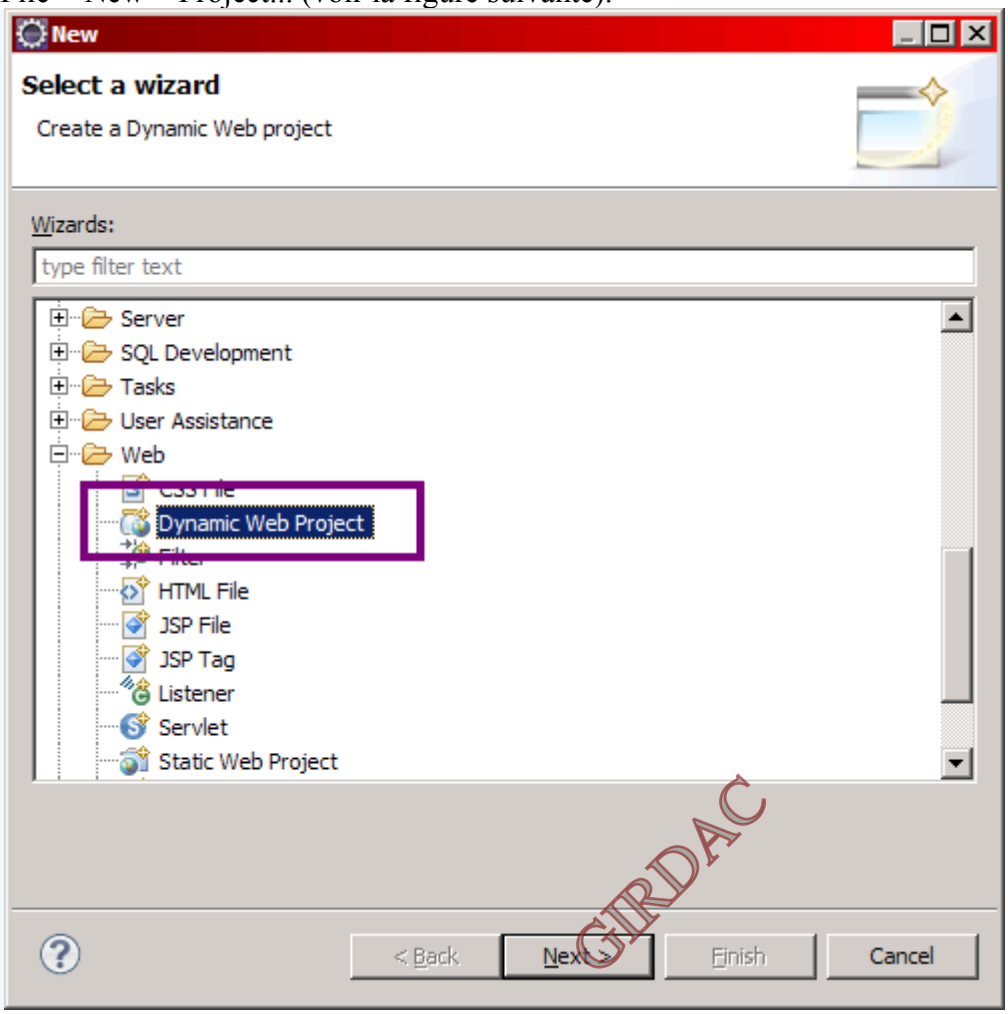
Webapps : ou sont stockées par défaut les applications.

Conf : qui contient les fichiers suivants :

server.xml : contient les éléments de configuration du serveur.
context.xml : contient les directives communes à toutes les applications web déployées sur le serveur.
tomcat-users.xml : contient entre autres l'identifiant et le mot de passe permettant d'accéder à l'interface d'administration de votre serveur Tomcat.

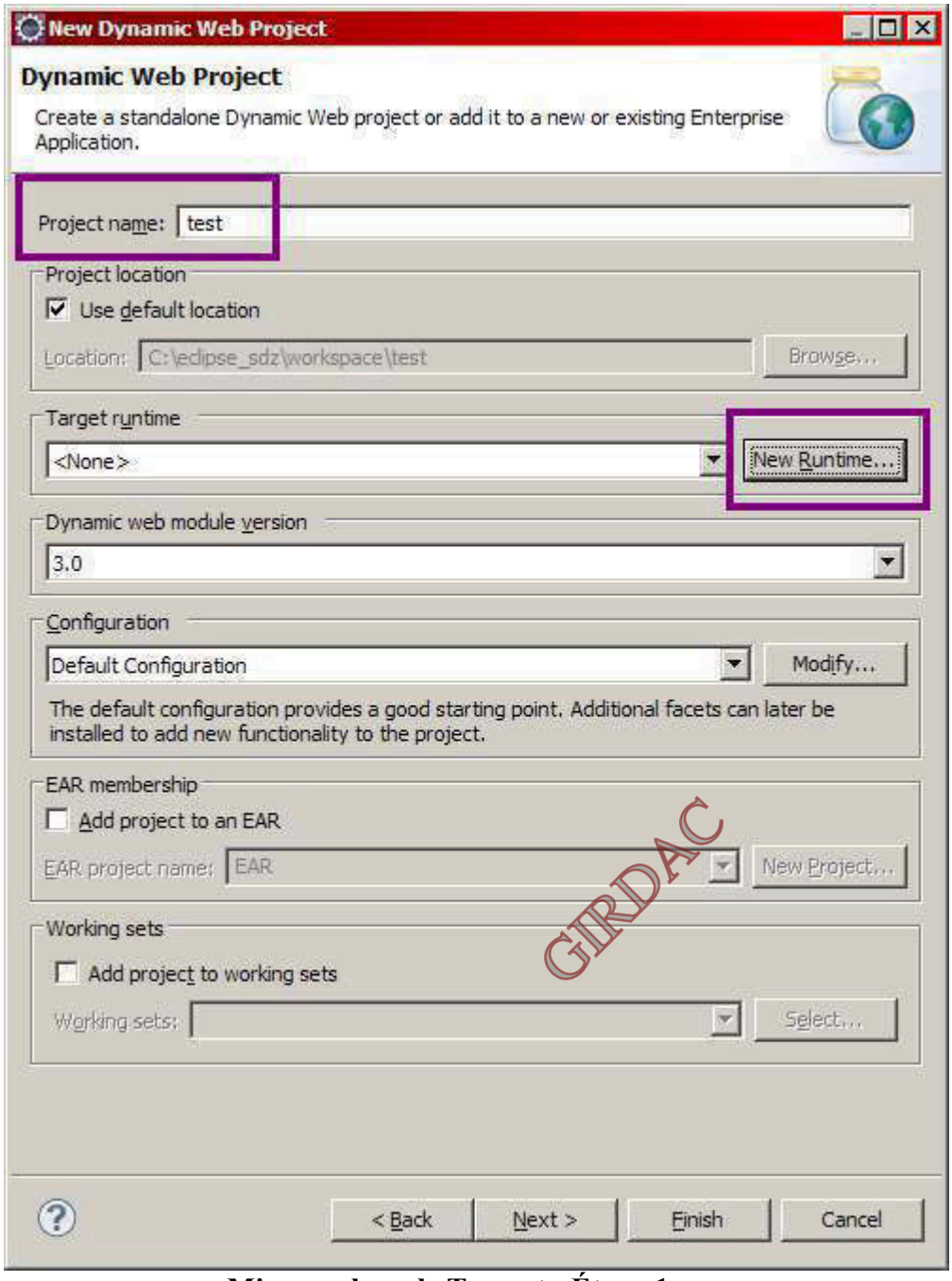
- **Les modifications sur ces dossiers s'effectuent indirectement via l'IDE Eclipse.**

Création du projet web avec Eclipse :
File > New > Project... (voir la figure suivante).



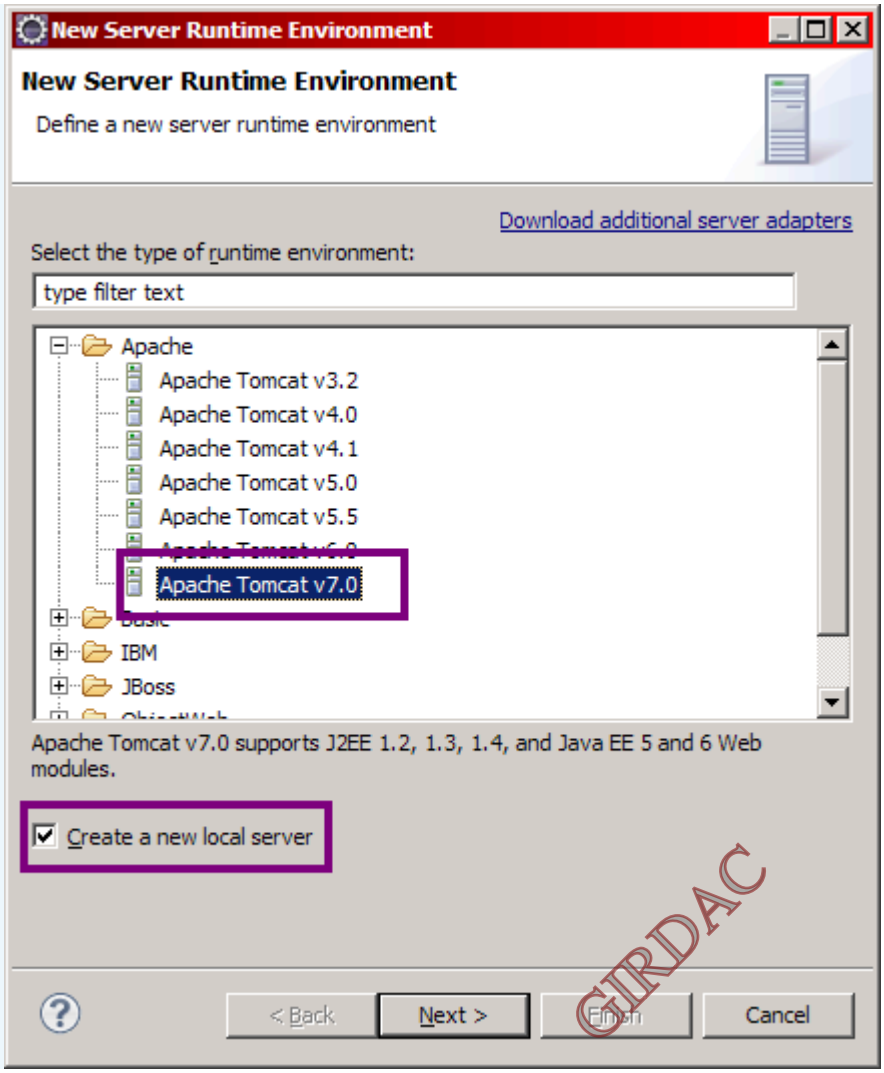
Nouveau projet web sous Eclipse

Sélectionnez alors **Dynamic Web Project** comme le montre l'image ci-dessus, puis cliquez sur Next >.
J'appelle ici mon projet **test**. Remarquez ensuite à la figure suivante le passage concernant le serveur.



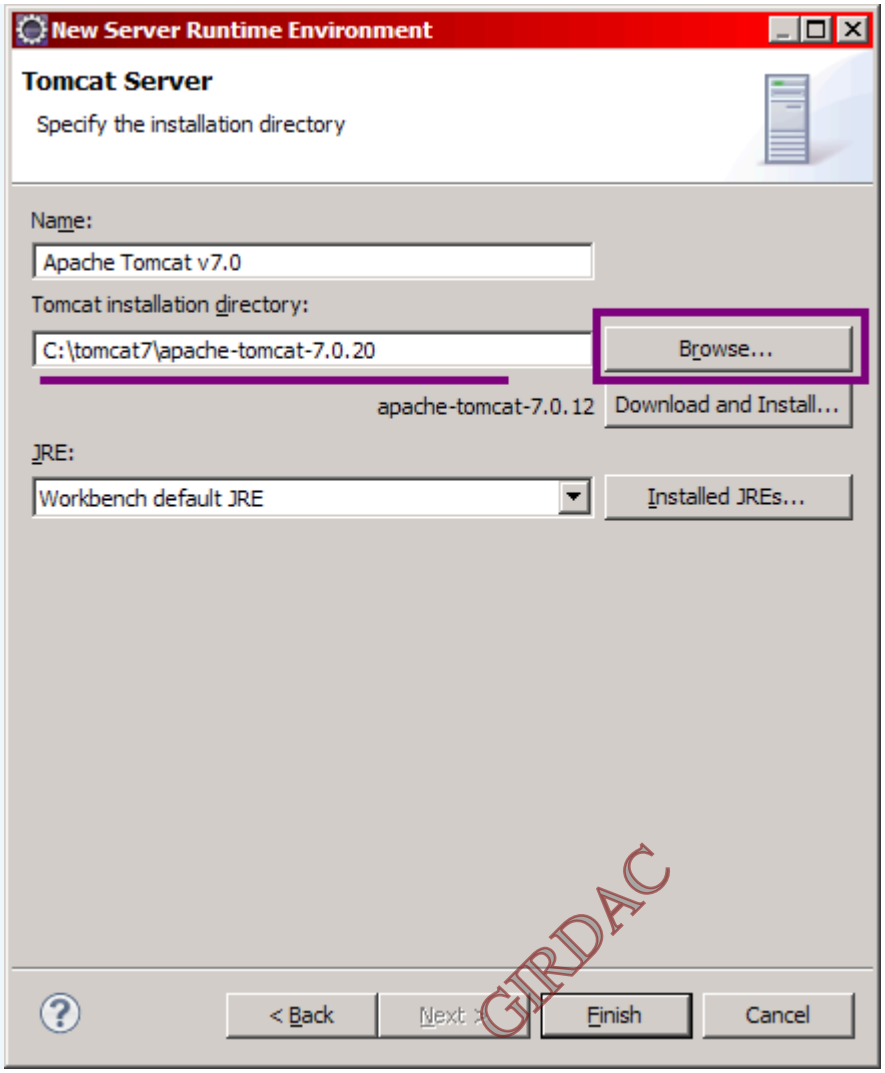
Mise en place de Tomcat - Étape 1

Cliquez sur le bouton New Runtime... et sélectionnez alors Apache Tomcat 7.0 dans la liste des possibilités, comme indiqué à la figure suivante.



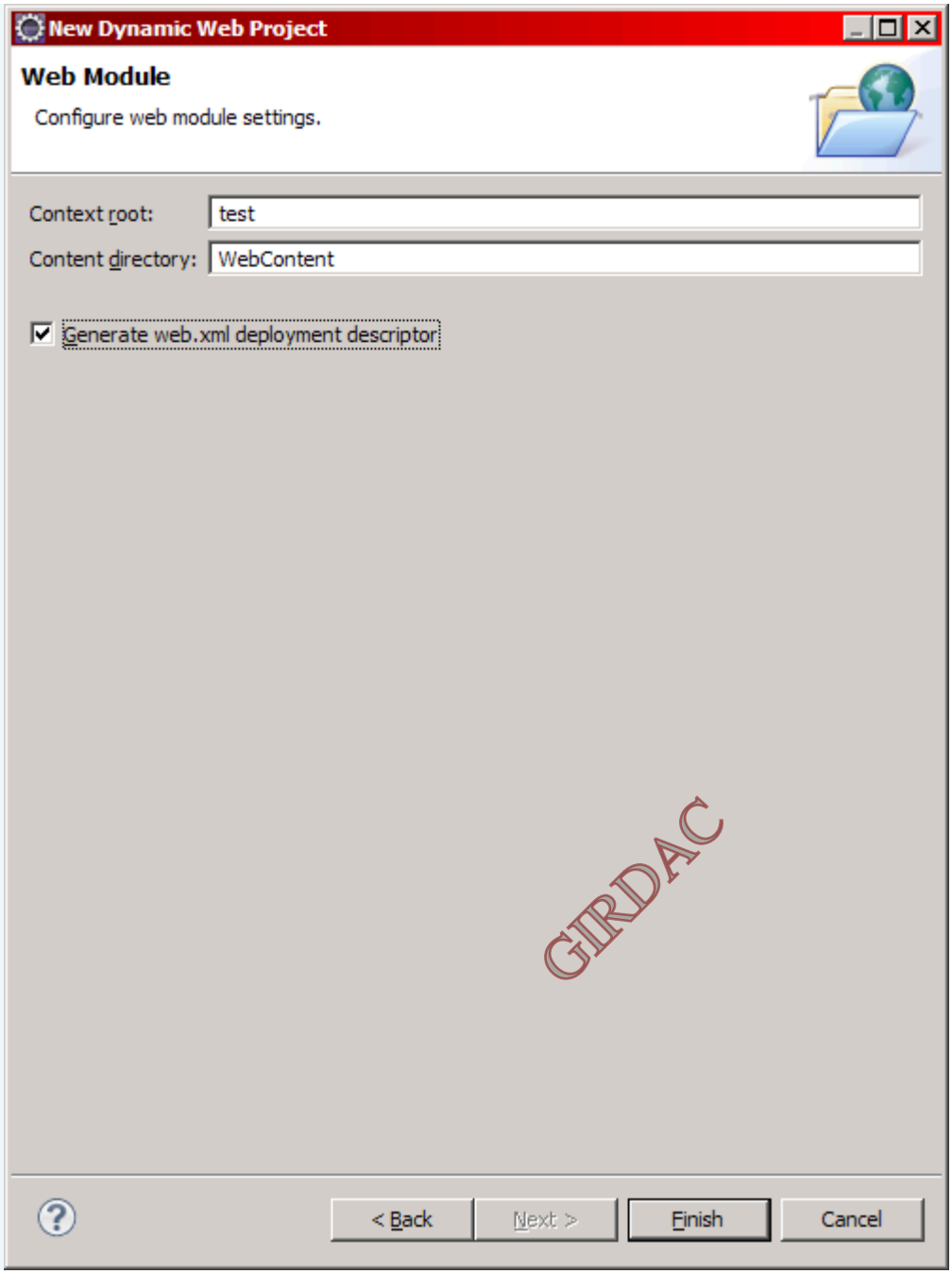
Mise en place de Tomcat - Étape 2

Cochez la case comme indiqué ci-dessus, ce qui signifie que nous allons en plus du projet créer localement une nouvelle instance d'un serveur, instance que nous utiliserons par la suite pour déployer notre application. Cliquez ensuite sur Next > et remplissez correctement les informations relatives à votre installation de Tomcat en allant chercher le répertoire d'installation de Tomcat sur votre poste. Les champs devraient alors ressembler à ceux de la figure suivante, le répertoire d'installation et le numéro de version de Tomcat 7 pouvant être différents chez vous selon ce que vous avez choisi et installé.



Mise en place de Tomcat - Étape 3

Validez alors en cliquant sur Finish, puis cliquez deux fois sur Next >, jusqu'à obtenir cette fenêtre (voir la figure suivante).



Mise en place de Tomcat - Étape 4

Il est déjà mentionné que le serveur Tomcat contient le fichier **context.xml** associé à toutes les applications. comme il est possible de spécifier un contexte propre à chaque *webapp*. Ces applications web sont empiriquement contenues dans le dossier… **webapps** de *Tomcat Home*.
Or que le projet à créer depuis Eclipse se trouve dans le répertoire workspace Eclipse : il n'est pas du tout dans le répertoire *webapps* de Tomcat.

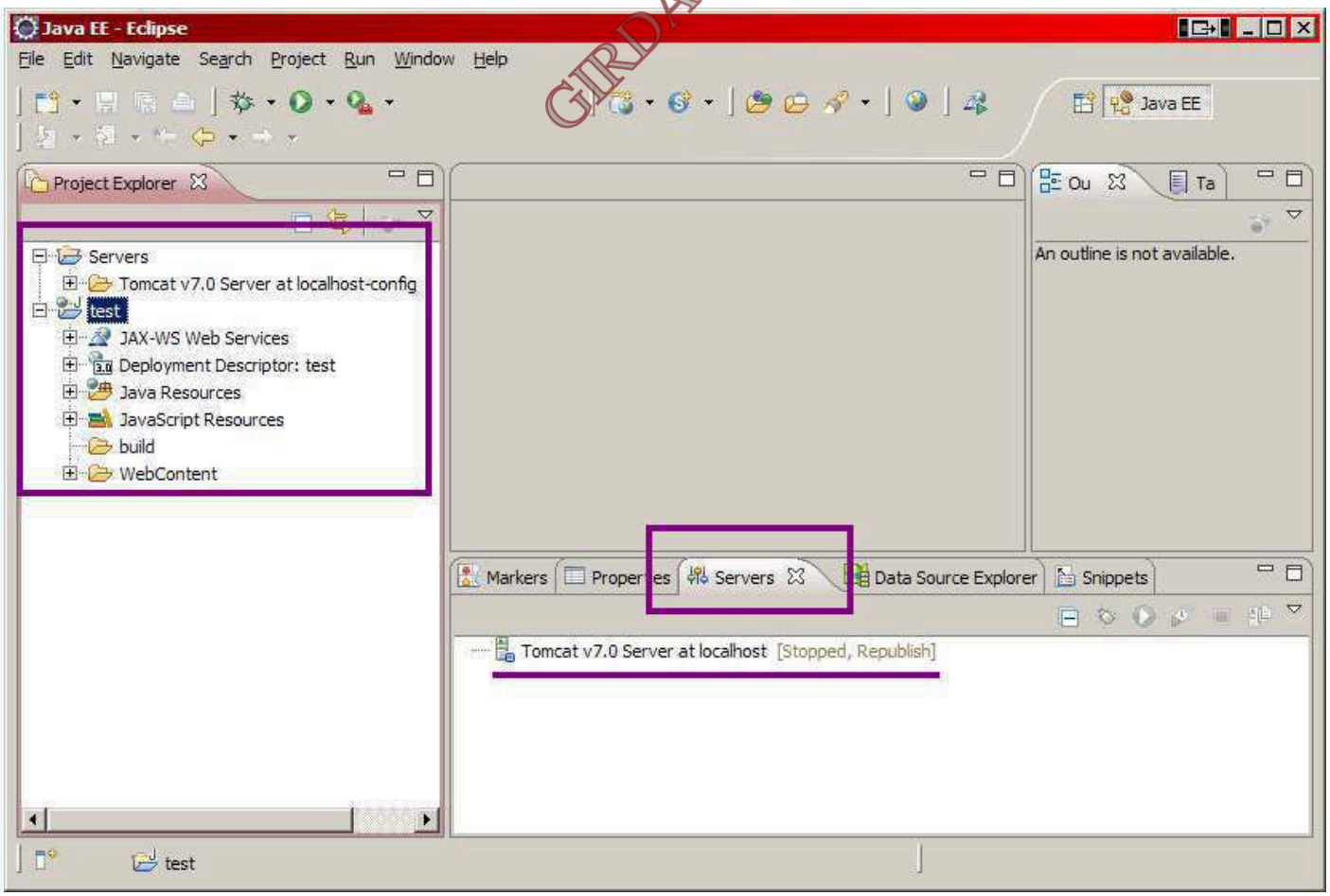
Pour que le serveur prenne en compte l'application, Plusieurs solutions s'offrent :

- créer un répertoire du même nom que notre projet sous Eclipse, directement dans le dossier *webapps* de Tomcat, et y copier-coller nos fichiers, et ce à chaque modification de code ou configuration effectuées.
- créer un nouveau projet depuis Eclipse, en utilisant directement le répertoire *webapps* de votre *Tomcat Home* comme *workspace* Eclipse.
- modifier le **server.xml** ou le **context.xml** de votre Tomcat, afin qu'il sache où chercher.
- utiliser les propriétés d'un projet web dynamique sous Eclipse.

La solution utilisée comme le montre la dernière fenêtre est la quatrième ; Conserver le nom du projet sous Eclipse comme contexte de déploiement sur le serveur Tomcat ("*Context root*" sur l'image précédente), afin de rester cohérent.

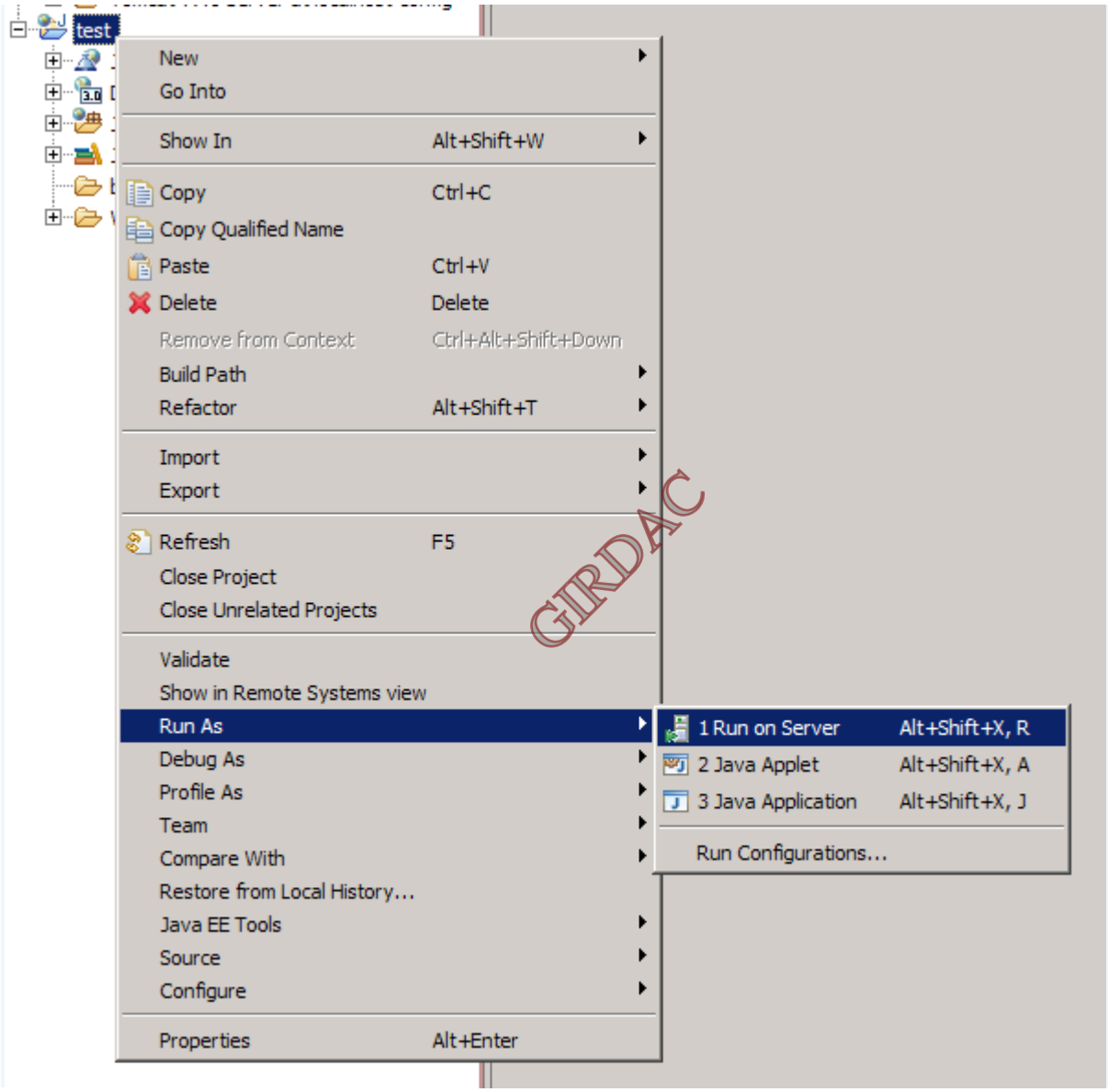
Ainsi, toute modification sur les pages et classes sera ainsi automatiquement prise en compte par le serveur Tomcat, qui s'occupe de recharger le contexte à chaque modification sauvegardée, lorsque le serveur sera lancé.

Voici à la figure suivante ce à quoi doit ressembler la fenêtre Eclipse.



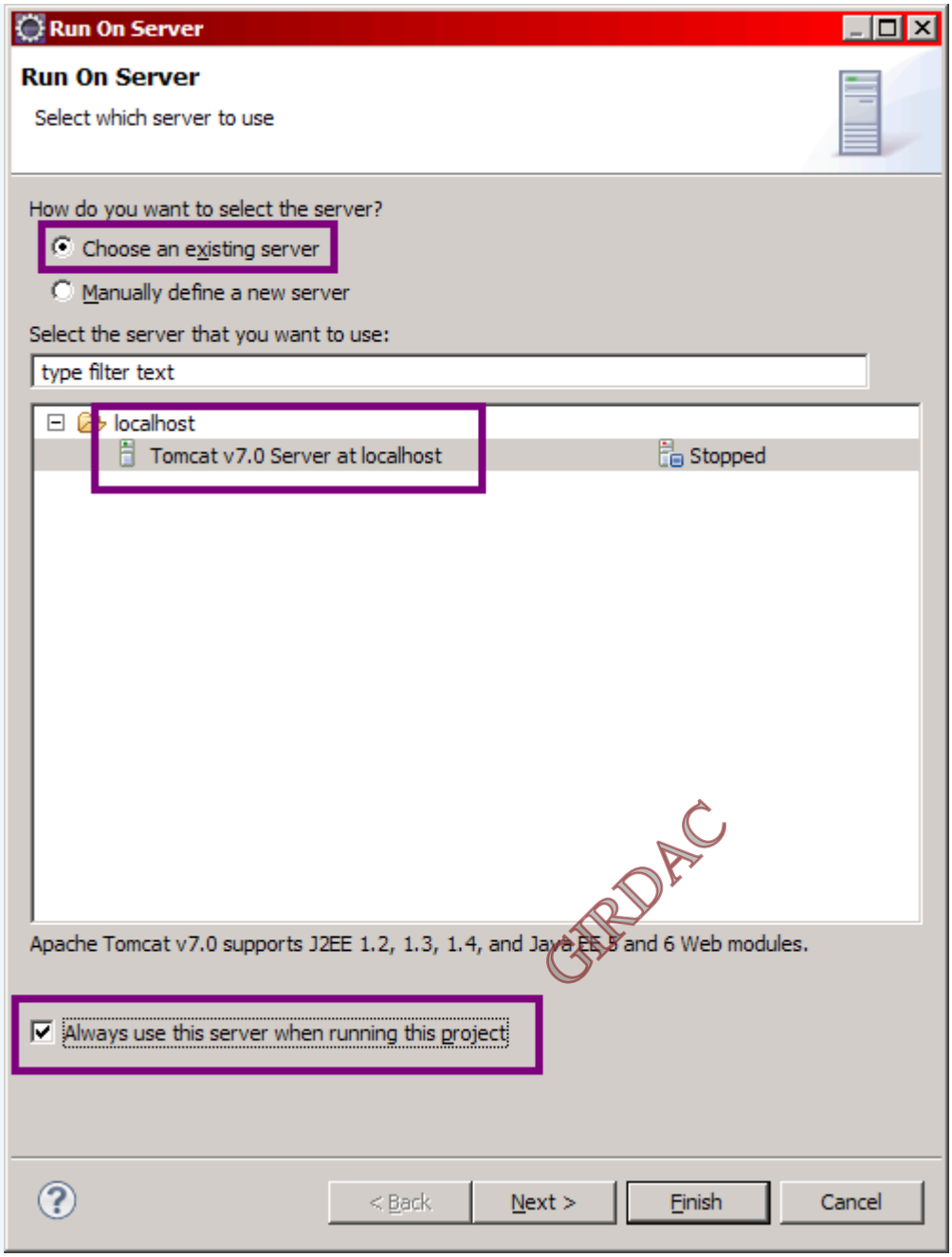
Mise en place de Tomcat - Étape 5

on note l'apparition d'une entrée **Tomcat v7.0** dans l'onglet Servers, et de l'arborescence de votre projet test dans le volet de gauche.
Faites maintenant un clic droit sur le titre de votre projet dans l'arborescence Eclipse, et suivez Run As > Run on Server, comme indiqué à la figure suivante.



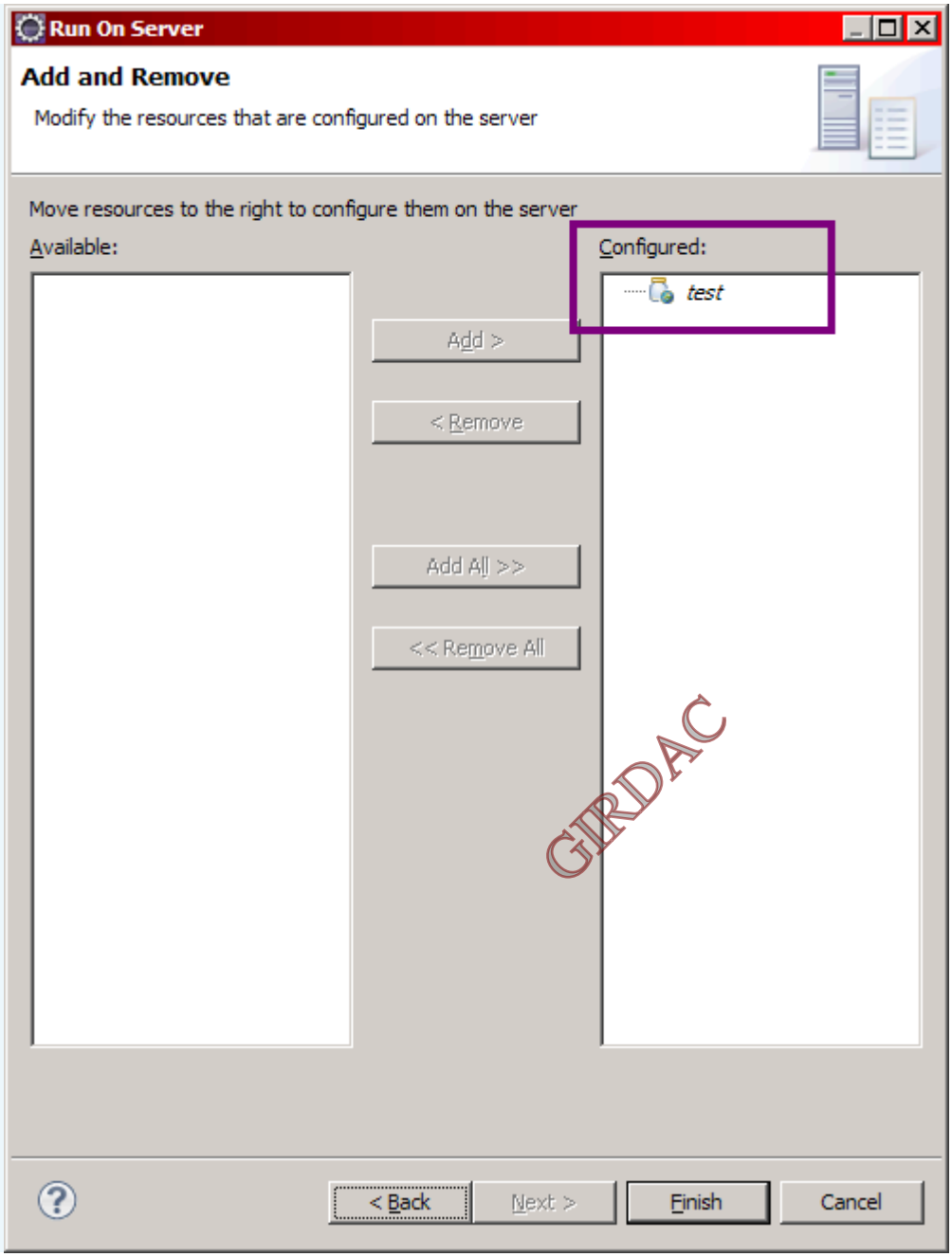
Mise en place de Tomcat - Étape 6

Dans la fenêtre qui s'ouvre, sélectionner le serveur Tomcat mis en place lors de la création du projet web, et préciser que l'on souhaite associer par défaut notre projet à ce serveur.



Mise en place de Tomcat - Étape 7

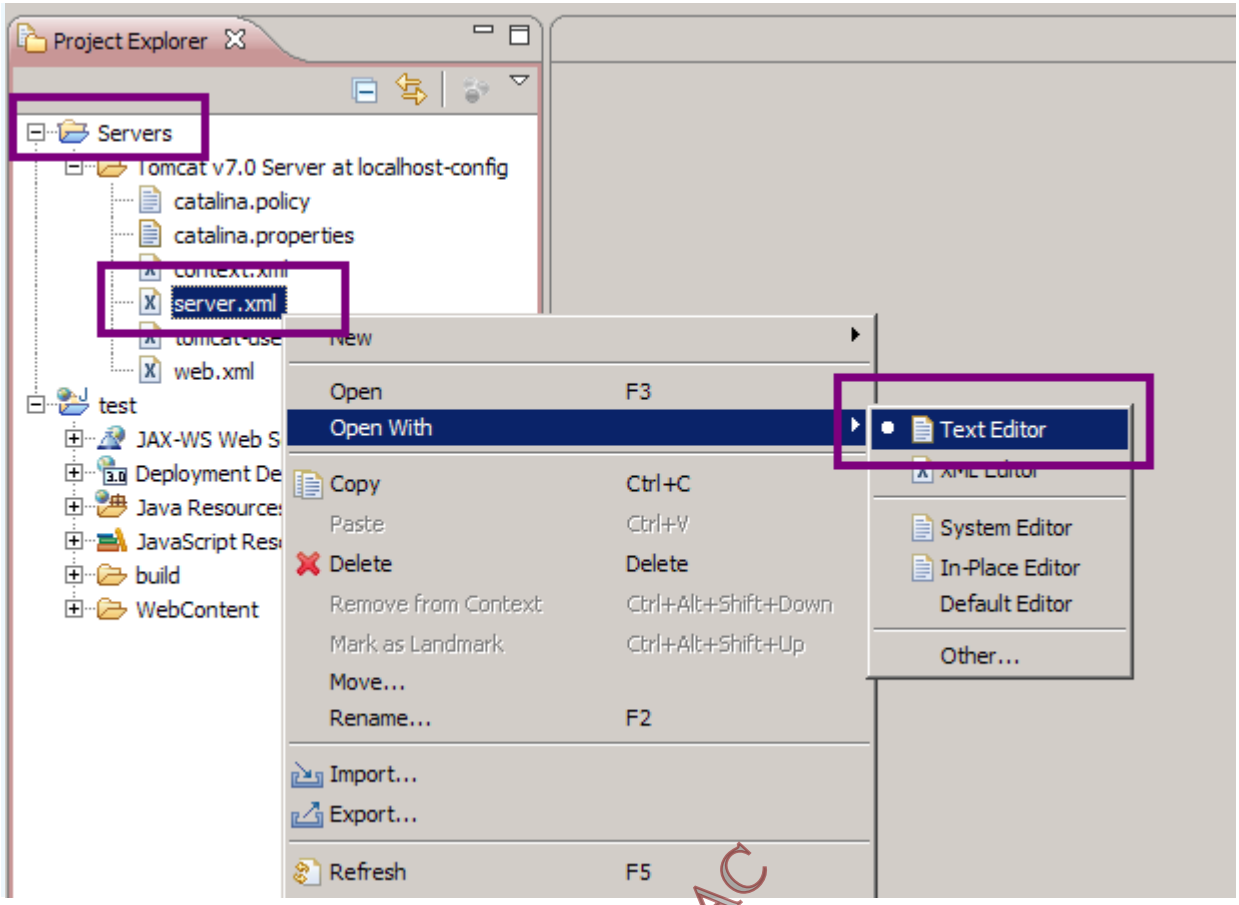
Cliquer sur Next >, puis vérifier que le nouveau projet est bien pris en compte par le serveur (voir la figure suivante).



Mise en place de Tomcat - Étape 8

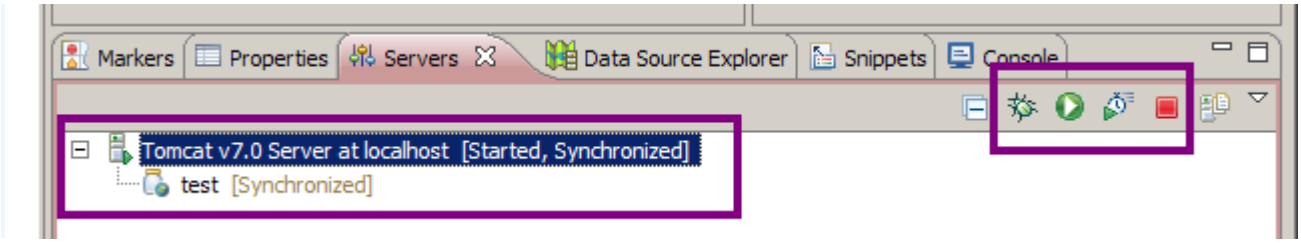
Valider enfin en cliquant sur Finish, et voilà la mise en place de votre projet et de son serveur **terminée**.

une section est ajoutée dans le fichier **server.xml** de l'instance de Tomcat crée, qui est maintenant accessible depuis le dossier **Servers** de l'arborescence Eclipse, comme le montre la figure suivante.



Mise en place de Tomcat - Étape 9

Dorénavant, pour piloter le serveur Tomcat il suffira de se rendre dans l'onglet **Servers** en bas de la fenêtre Eclipse, et d'utiliser un des boutons selon le besoin (redémarrage, arrêt, debug), comme indiqué à la figure suivante.

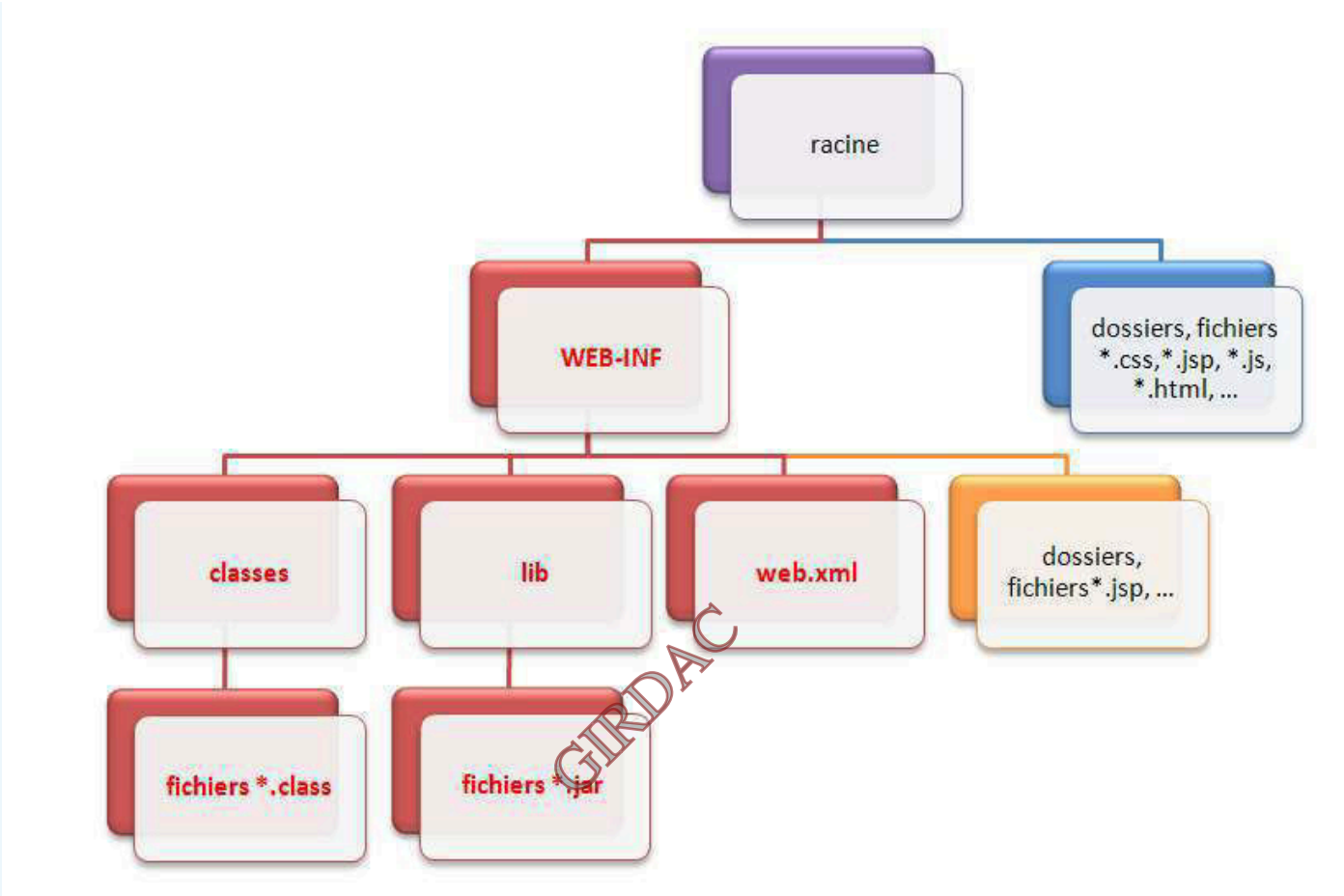


Mise en place de Tomcat - Étape 10

Une fois correctement paramétré un serveur Tomcat depuis Eclipse, c'est possible de réutiliser la même instance de Tomcat en y déployant plusieurs applications web différentes.

d.Structure d'une application javaEE :

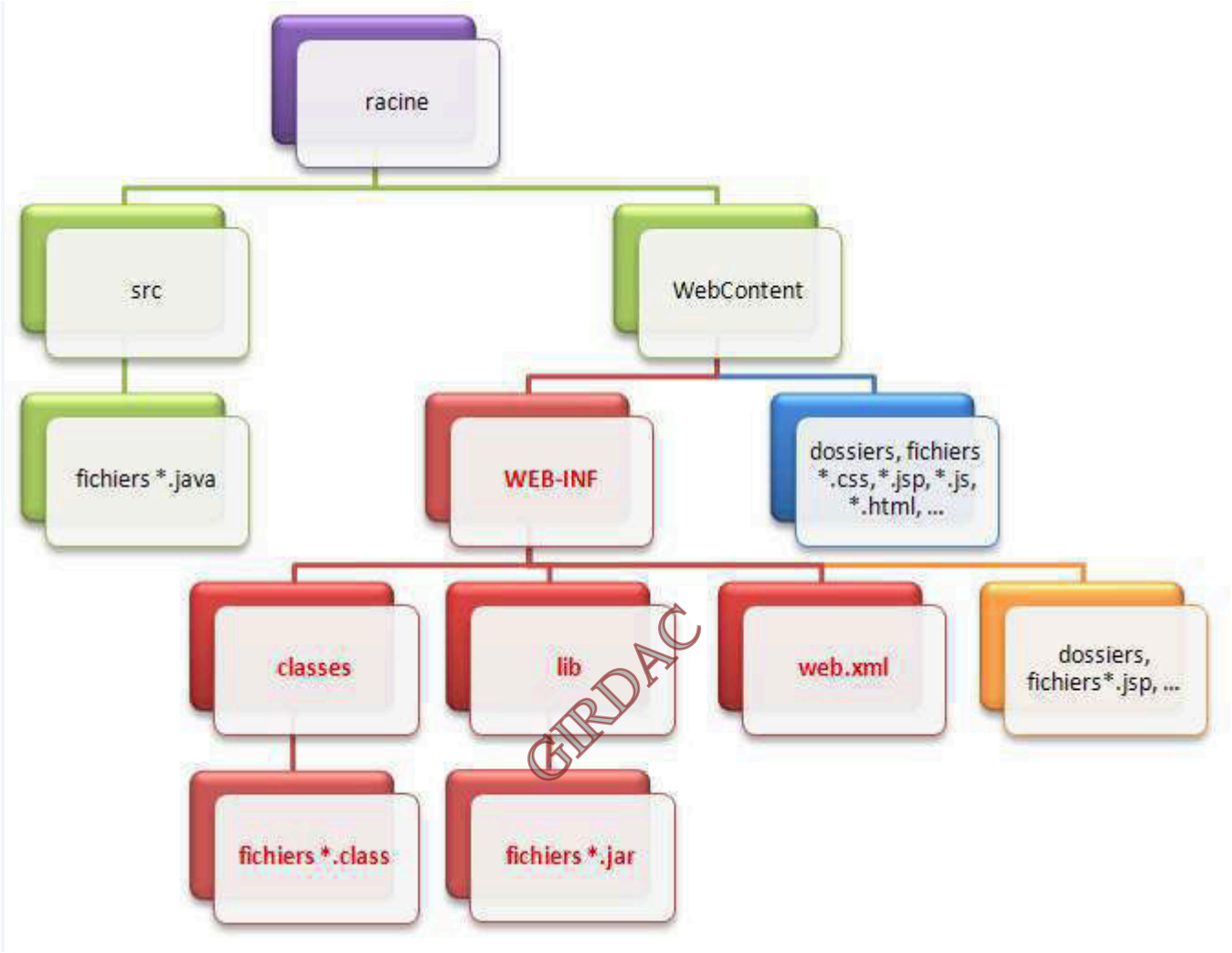
Toute application web Java EE doit respecter une structure de dossier standard, qui est définie dans les spécifications de la plate-forme.



Structure des fichiers d'une application web JSP/Servlet

- La racine de l'application, est le dossier qui porte le nom du projet et qui contient l'intégralité des dossiers et fichiers de l'application.
- Le dossier nommé **WEB-INF** est un dossier spécial. Il doit obligatoirement exister et être placé juste sous la racine de l'application. Il doit à son tour obligatoirement contenir :
- le fichier de configuration de l'application (web.xml).
- un dossier nommé **classes**, qui contient à son tour les classes compilées (fichiers .class).
- un dossier nommé **lib**, qui contient à son tour les bibliothèques nécessaires au projet (archives .jar).
- Les fichiers et dossiers persos placés directement sous la racine, en bleu sur le schéma, sont publics et donc accessibles directement par le client via leurs URL. **(*)**

e.Structure d'une application web sous Eclipse :
Eclipse adapte certaines caractéristiques comme le montre le schéma suivant à son mode de fonctionnement :



Structure des fichiers d'une application web sous Eclipse

Eclipse déplace la structure standard de l'application vers un dossier nommé **Web Content**, et ajoute sous la racine un dossier **src** qui contient le code source des classes (les fichiers .java).

Eclipse ajoute également sous la racine quelques fichiers de configuration qui lui permettront de gérer correctement l'application.

- le dossier **Web Content** n'existe légitimement qu'au sein d'Eclipse. Si vous développez sans IDE, ce répertoire ne doit pas exister et votre application doit impérativement suivre la structure standard présentée précédemment.

Pour cette même raison, si on souhaite utiliser votre application en dehors de l'IDE, il faudra obligatoirement utiliser l'outil d'export proposé par Eclipse. Réaliser un simple copier-coller des dossiers ne fonctionnera pas en dehors d'Eclipse.

II. Patrons de conception :

II.1 Définition : En anglais *design pattern*, un modèle de conception (ou encore patron de conception) est une simple **bonne pratique**, qui répond à un problème de conception d'une application. C'est en quelque sorte une ligne de conduite qui permet de décrire les grandes lignes d'une solution. De tels modèles sont issus de l'expérience des concepteurs et développeurs d'applications.
En d'autres termes un patron de conception est une solution générique d'implémentation répondant à un problème spécifique.
Par leur aspect générique, ils sont considérés comme des microarchitectures qui visent à réduire la complexité, à promouvoir la réutilisation et à fournir un vocabulaire commun aux concepteurs.
La description d'un patron de conception suit un formalisme fixe :
Nom : le nom significatif du patron.
Description du problème à résoudre ;
Description de la solution : les éléments de la solution, avec leurs relations. La solution est appelée patron de conception ;
Conséquences : résultats issus de la solution.

II.2 Types de patrons de conception :
Il existe différents ensembles de patrons de conception, créés par différents auteurs.
Les plus connus sont ceux du « Gang of Four » (ou GoF : Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides) décrits dans leur livre « Design Patterns -- Elements of Reusable Object-Oriented Software » en 1995. Les patrons de conception tirent leur origine des travaux de l'architecte Christopher Alexander dans les années 70.
ces patrons sont classés leur rôle et leur domaine d'application. On distingué alors les patrons créateurs, structurels et comportementaux.

Les patrons créateurs concernent la création de classes ou d'objets.
Les patrons structurels s'intéressent à la composition d'objets ou classes pour réaliser de nouvelles fonctionnalités.
Les patrons comportementaux concernent les interactions entre classes et l'affection des responsabilités
Les patrons GRASP sont des patrons créés par Craig Larman qui décrivent des règles pour affecter les responsabilités aux classes d'un programme orienté objets pendant la conception, en liaison avec la méthode de conception BCE (pour « Boundary Control Entity » - en français MVC « Modèle Vue Contrôleur »).
Les patrons d'entreprise (*Enterprise Design Pattern*) créés par Martin Fowler, décrivent des solutions à des problèmes courants dans les applications professionnelles. Par exemple, des patrons de couplage entre un modèle objet et une base de données relationnelle.
D'**autres patrons** créés par divers auteurs existent et décrivent des solutions à des problèmes différents de ceux vus précédemment.

II.3 Le patron MVC :

II.3.1Description du MVC :
Il découpe littéralement l'application en couches distinctes, et de ce fait impacte très fortement l'organisation du code ! Voici dans les grandes lignes ce qu'impose MVC :

- Tout ce qui concerne le traitement, le stockage et la mise à jour des données de l'application doit être contenu dans la couche nommée "Modèle" (le M de MVC) ;
- Tout ce qui concerne l'interaction avec l'utilisateur et la présentation des données (mise en forme, affichage) doit être contenu dans la couche nommée "Vue" (le V de MVC) ;
- tout ce qui concerne le contrôle des actions de l'utilisateur et des données doit être contenu dans la couche nommée "Contrôle" (le C de MVC).

II.3.2Application du MVC par java EE

En pratique ces éléments sont concrétisés par les éléments java EE suivants :

Modèle : des traitements et des données.

Ce bloc contient donc des objets Java d'une part, qui peuvent contenir des attributs (données) et des méthodes (traitements) qui leur sont propres, et un système capable de stocker des données d'autre part.

Vue : des pages JSP

Une page JSP permet l'écriture de pages en langage "client" comme HTML, CSS, Javascript, XML, etc.). Elle permet au concepteur de la page d'appeler de manière transparente des portions de code Java, via des balises et expressions ressemblant fortement aux balises de présentation HTML.

Contrôleur : des servlets

Une servlet est un objet qui permet d'intercepter les requêtes faites par un client, et qui peut personnaliser une réponse en conséquence. Il fournit pour cela des méthodes permettant de scruter les requêtes **HTTP**.

La servlet joue le rôle d'aiguilleur entre la couche modèle et la couche vue ; il intercepte une requête issue d'un client, appelle éventuellement des traitements effectués par le modèle, et ordonne en retour à la vue d'afficher le résultat au client.

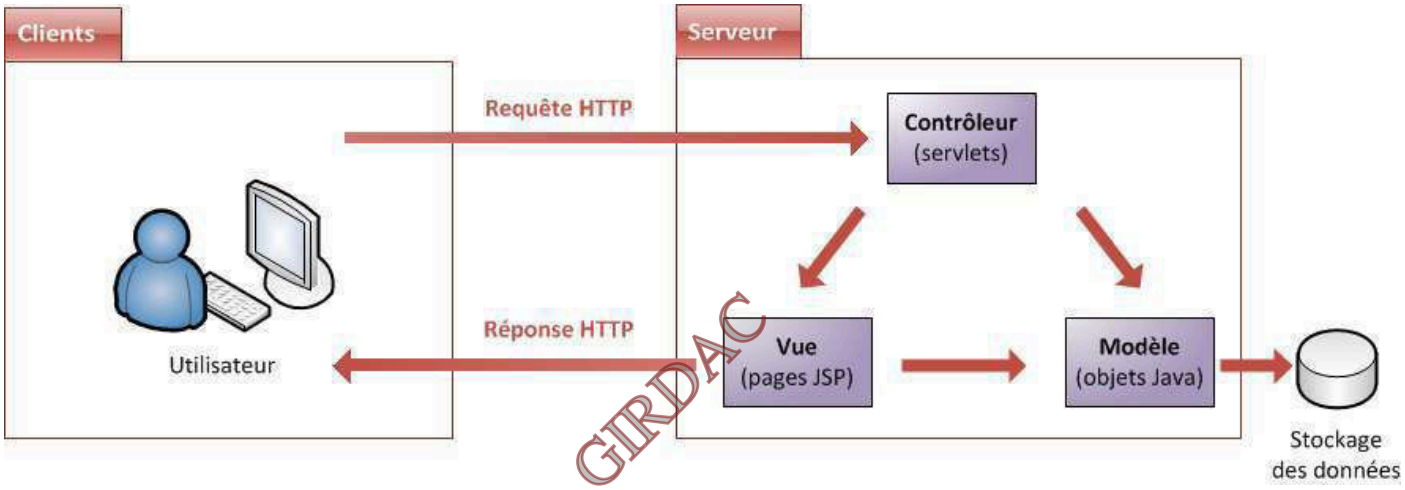


Schéma1 : l'application du modèle MVC par la plate forme java EE.

II.3.3Différentes couches du modèle MVC :

A.La couche modèle :

a.Communication directe entre les objets métiers de la couche Modèle du MVC et le système de stockage :

Nous remarquons que la couche Modèle comprend les données et les traitements. Ce qui implique la liaison forte entre le code responsable des traitements métier au code responsable de stockage des données.

Schéma2 : communication directe entre les objets métiers de la couche Modèle avec le système de stockage.

Cette manière de codage engendre les problèmes suivant :

- il est impossible de mettre en place des tests unitaires :
- impossible de tester le code métier de l'application sans faire intervenir le stockage (BDD, etc.) ;
- impossible de ne tester que le code relatif au stockage des données, obligation de lancer le code métier.
- il est impossible de changer de mode de stockage. Que ce soit vers un autre SGBD, voire vers un système complètement différent d'une base de données, cela impliquerait une réécriture complète de tout le modèle, car le code métier est mêlé avec et dépendant du code assurant le stockage.

Sans oublier la difficulté de maintenance.
Sachant qu'écrire un code orienté objet et bien organisé est une excellente pratique.
Afin de mieux organiser cette couche, il faut séparer les données et les traitements.

b.Isoler le stockage des données :

L'idée est qu'au lieu de faire communiquer directement les objets métier avec la base de données, ou le système de fichiers, ou les *webservices*, ceux-ci vont communiquer avec une autre couche. Et c'est cette couche qui va ensuite de son côté communiquer avec le système de stockage.
Nous souhaitons en effet littéralement encapsuler ce code dans une couche de laquelle aucune information concernant le mode de stockage utilisé ne s'échappe. En d'autres termes, notre objectif est de cacher la manière dont sont stockées les données au reste de l'application.
C'est ce qui conduit à la facilitation de changement du système de stockage ; en ne portant des modifications que sur cette couche.
java EE utilise un autre patron de conception qui agit sur cette couche (Modèle), il s'agit du pattern DAO (Data Acces Objects).

c.Le modèle DAO : permet la distinction entre les données à accéder et la manière dont sont stockées ces données.
En l'appliquant sur la couche Modèle, résulte deux sous- couche :

- une couche gérant les traitements métier appliqués aux données, souvent nommée couche **service ou métier.**
- une couche gérant le stockage des données, logiquement nommée **couche de données**.
  Il s'agit là des opérations classiques de stockage : la création, la lecture, la modification et la suppression. Ces quatre tâches basiques sont souvent raccourcies à l'anglaise en ***CRUD***.

il s'agit de faire en sorte que le code basé sur JDBC soit bien à l'abri dans des implémentations de DAO, et que nos objets métier n'aient connaissance que des **interfaces** qui les décrivent.

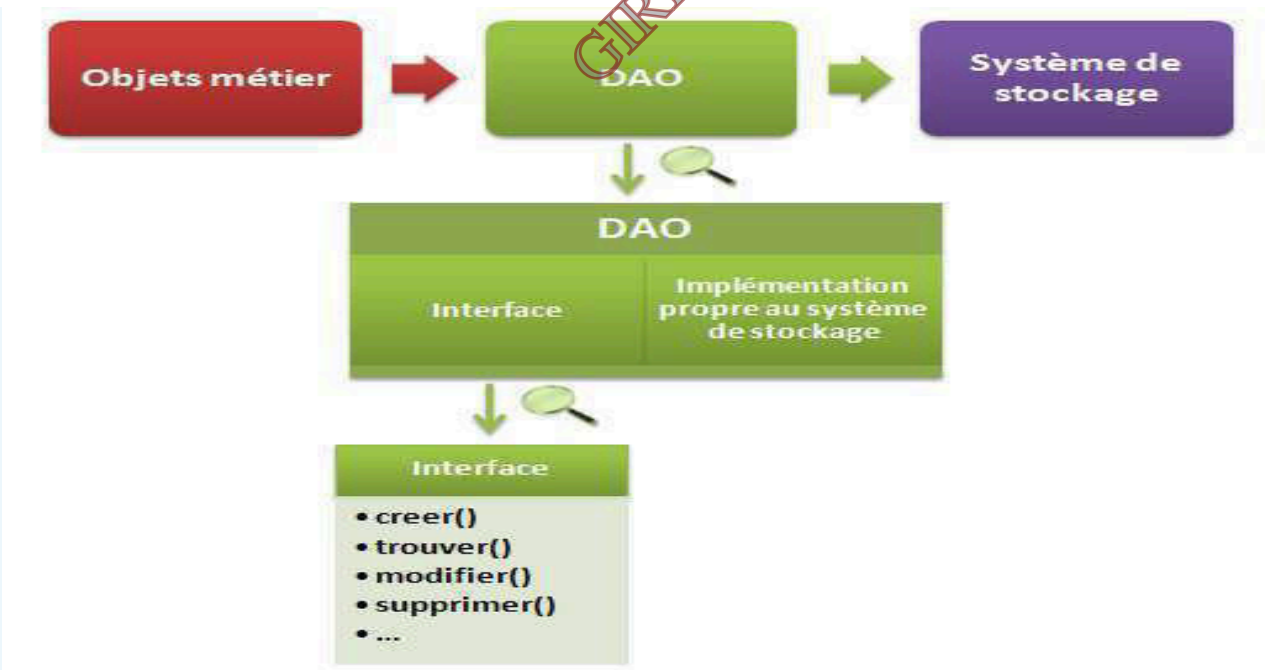


Schéma3 : l'application du pattern DAO sur la couche Modèle.

- En résumé java EE mis en pratique deux patrons de conceptions ; voir le patron de conception MVC qui découpe l'application en trio Modèle-Vue-Contrôle.

Et le patron de conception DAO qui intervient dans la couche Modèle du MVC pour encapsuler la manière dont elles sont stockées les données.et ainsi apporte de la souplesse lors du changement du système de stockage.

d.Le JavaBean : une autre composante de la couche modèle est le JavaBean.

d.1 Définition :

Le **JavaBean**. Souvent raccourci en "bean", un JavaBean désigne tout simplement un composant réutilisable. Il est construit selon certains standards, définis dans les spécifications de la plate-forme et du langage Java eux-mêmes : un bean n'a donc rien de spécifique au Java EE.

d.2 Objectifs :

Voici un récapitulatif des principaux concepts mis en jeu :

Les propriétés : un bean est conçu pour être **paramétrable**. On appelle "propriétés" les champs non publics présents dans un bean. Qu'elles soient de type primitif ou objets, les propriétés permettent de paramétrer le bean, en y stockant des données.

La sérialisation : un bean est conçu pour pouvoir être **persistant**. La sérialisation est un processus qui permet de sauvegarder l'état d'un bean, et donne ainsi la possibilité de le restaurer par la suite. Ce mécanisme permet une persistance des données, voire de l'application elle-même.

La réutilisation : un bean est un composant conçu pour être **réutilisable**. Ne contenant que des données ou du code métier, un tel composant n'a en effet pas de lien direct avec la couche de présentation, et peut également être distant de la couche d'accès aux données (avec modèle de conception DAO). C'est cette indépendance qui lui donne ce caractère réutilisable.

L'introspection : un bean est conçu pour être **paramétrable de manière dynamique**. L'introspection est un processus qui permet de connaître le contenu d'un composant (attributs, méthodes et événements) de manière dynamique, sans disposer de son code source. C'est ce processus, couplé à certaines règles de normalisation, qui rend possible une découverte et un paramétrage dynamique du bean.

d.3 Structure :

Un bean :

doit être une **classe publique** ;

doit avoir au moins **un constructeur par défaut, public et sans paramètres**.

peut implémenter l'interface Serializable, il devient ainsi persistant et son état peut être sauvegardé ;

ne doit pas avoir de champs publics ;

peut définir **des propriétés (des champs non publics), qui doivent être accessibles via des méthodes publiques *getter* et *setter***, suivant des **règles de nommage**.

```
public class MonBean{
private String proprieteNumero1;
private int proprieteNumero2;
public String getProprieteNumero1() {
   return this.proprieteNumero1;
   }
public int getProprieteNumero2() {
   return this.proprieteNumero2;
   }
public void setProprieteNumero1( String proprieteNumero1 ) {
          this.proprieteNumero1 = proprieteNumero1;        }
public void setProprieteNumero2( int proprieteNumero2 ) {
      this.proprieteNumero2 = proprieteNumero2;}
   }
```

d.4 Mise en place :

Cet objet doit être placé dans le répertoire des sources "src" de notre projet web.

Clique droit de la souris sur le dossier→ src →New→ Class
Rendre un bean accessible à notre application :
L'application ne peut pas se baser sur ces fichiers sources, elle ne comprend que les classes compilées.
il faut que les classes compilées à partir des fichiers sources soient placées dans un dossier "classes".
Le dossier classes doit être placé sous le répertoire **/WEB-INF**.

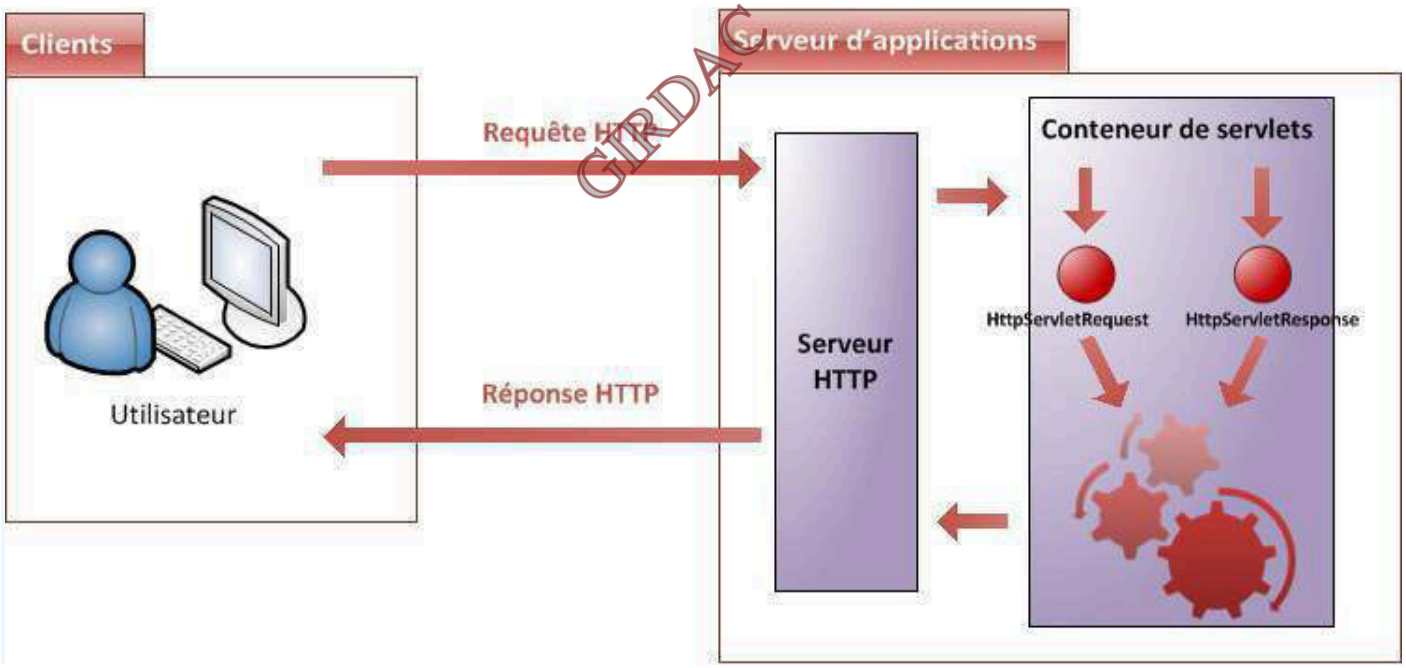
- L'environnement Eclipse, envoi par défaut les classes compilées dans un dossier nommé « build ».

Pour changer ce comportement, il faut modifier le **build path** :
Clic droit sur le dossier du projet – sélectionner Build path – configurer Build path
Sélectionner l'onglet Source – Default output folder ; mettre le chemin approprié
(nomprojet/WebContent/Web-INF/classes.

d.5 Mise en service dans notre application :
Pour mettre en service le bean, il faut l'instancier dans la servlet et l'enregistrer comme un attribut de requête, ensuite le récupérer dans la JSP pour affichage.

B. la couche contrôle (La Servlet) :

Nous savons qu'une requête http part du client au serveur http. Le serveur http (serveur web) a pour rôle d'écouter les requêtes http sur le port 80. Et lorsqu'elle lui parvient, il l'a transmet au conteneur de servlets, également nommé **conteneur web (dans notre cas Tomcat). Ce dernier** les transmet à l'application (plus précisément aux servlets) après avoir créer les deux objets HttpServletRequest et HttpServletResponse.



Traitements des requêtes et réponses http dans le serveur d'application

a.Définition d'une servlet : c'est une simple classe Java, qui a la particularité de **permettre le traitement de requêtes et la personnalisation de réponses.** Dans la très grande majorité des cas une servlet n'est rien d'autre qu'une classe capable de recevoir une requête HTTP envoyée depuis le navigateur de l'utilisateur, et de lui renvoyer une réponse HTTP.
b.Création d'une servlet : pour créer une servlet, on doit l'hériter de l'une des classes de base qui implémentent l'interface servlet.
Comme nous nous intéressons aux requêtes http, nous allons hériter nos servlets d'une classe de base **HttpServlet** après avoir fait l'import : javax.servlet.http.HttpServlet.

La classe HttpServlet : est une classe abstraite, et elle propose des méthodes java nécessaires au traitement des différentes requêtes et réponses HTTP et qui sont :
doGet() pour gérer la méthode GET ;
doPost() pour gérer la méthode POST ;
doHead() pour gérer la méthode HEAD.
En plus de ces méthodes, la **méthode service ()** qui se charge de lire l'objet HttpServletRequest et de distribuer la requête HTTP à la méthode doXXX() correspondante.
une servlet **doit implémenter** au moins une des méthodes doXXX(), afin d'être capable de traiter une requête entrante.

```
package exemple;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class Test extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse respons
e) throws ServletException, IOException{
}   }
```


c.Mise en œuvre d'une servlet :
Lorsqu'on met en place une page HTML statique, on accède directement à la page en question via une URL directe pointant vers le fichier depuis notre navigateur.
La servlet qui est un fichier java, on doit configurer le fait que notre servlet va être associée à une URL, dans le fichier texte **web.xml de l'application.**
La mise en place d'une servlet se déroule en deux étapes :

- **déclarer la servlet : pour que le serveur puisse la reconnaitre.** Pour ce faire, il faut ajouter la section suivante au fichier web.xml :

```
<servlet>
        <servlet-name>nomServlet</servlet-name>
        <servlet-class> chemin de la classe de la servlet dans l'application </servlet-class>
</servlet>
```

- ***lui faire correspondre une URL*** **(**Mapping de la servlet) : afin qu'elle soit joignable par le client.

```
<servlet-mapping>
        <servlet-name> nomServlet </servlet-name>
        <url-pattern>/urlRelative</url-pattern>
</servlet-mapping>
```

URL patern : c'est un modèle, et pas nécessairement d'une URL fixe. Ainsi, on peut choisir de rendre notre servlet responsable du traitement des requêtes issues d'une seule URL, ou bien d'un groupe d'URL.
De même, il est tout à fait possible de déclarer plusieurs sections <servlet-mapping> pour une même section <servlet> dans le fichier **web.xml**.
URL relative : l'URL ou le pattern que vous renseignez dans le champ <url-pattern> sont basés sur **le contexte de l'application**.

- il est impératif de définir une servlet avant de spécifier son mapping.

Puisque ce sont elles qui prennent en charge les requêtes entrantes, **les servlets vont être les points d'entrée de notre application web, c'est par elles que tout va passer (Contrôle)**.

Après avoir découvrir le « C » du MVC, passons au composant du «V » du MVC toujours, qui est le JavaBean .

C. La couche vue (La jsp) :

Le modèle MVC nous conseille de placer tout ce qui touche à l'affichage final (texte, mise en forme, etc.) dans une couche à part : la vue. Et comme nous avons mentionné précédemment que dans l'environnement javaEE, la technologie utilisée pour réaliser une vue est **la page JSP**.

a.Description d'une page JSP :

C'est un document qui, à première vue, ressemble beaucoup à une page HTML, mais qui en réalité en diffère par plusieurs aspects :

- extension d'une telle page devient **.jsp** et non plus .html.
- une telle page peut contenir des balises HTML, mais également des **balises JSP** qui appellent de manière transparente du code Java.
- contrairement à une page HTML statique directement renvoyée au client, **une page JSP est exécutée côté serveur**, et génère alors une page renvoyée au client.

il est possible de générer n'importe quel type de format avec une page JSP : du HTML donc, mais tout aussi bien du CSS, du XML, du texte brut, etc.

la technologie JSP consiste en une véritable abstraction de la technologie servlet : cela signifie concrètement que **les JSP permettent au développeur de faire du Java sans avoir à écrire de code Java ;** vous pouvez voir le code JSP écrit par le développeur comme une succession de raccourcis en tous genres qui, dans les coulisses, appellent en réalité des portions de code Java toutes prêtes.

- les servlets résultant de la traduction des JSP dans une application n'ont pour rôle que de permettre la manipulation des requêtes et réponses HTTP. En aucun cas elles n'interviennent dans la couche de contrôle, elles agissent de manière transparente et font bien partie de la vue : ce sont simplement des traductions en un langage que comprend le serveur (le Java !) des vues présentes dans l'application (de simples fichiers textes contenant de la syntaxe JSP).

b.Garder le contrôle :

Il est possible de visualiser le résultat en appelant directement la page JSP depuis son URL. Mais dans une application Java EE il ne faut jamais procéder ainsi ; MVC nous recommande en effet de passer par un contrôleur (qui est la servlet) pour afficher le contenu des pages JSP. Nous allons donc tâcher de **toujours associer une servlet à une vue**.

Nous allons donc systématiquement créer une servlet lorsque nous créerons une page JSP. Pour garder ainsi le contrôle, en vous assurant qu'une vue ne sera jamais appelée par le client sans être passée à travers une servlet. **la servlet est le point d'entrée de l'application.**

Malgré l'association de la JSP à la servlet, elle peut toujours être appelée depuis une url. Ce qui dépasse la règle : la servlet est le point d'entrée d'une application.

Pour remédier à ce problème, il faut déplacer la page JSP dans le répertoire **/WEB-INF**. nous savons que ce dossier a une particularité : il cache automatiquement les ressources qu'il contient.

En d'autres termes, une page présente sous ce répertoire n'est plus accessible directement par une URL côté client. Il devient alors **nécessaire** de passer par une servlet côté serveur pour donner l'accès à cette page.

Pour associer notre servlet à notre vue. Cette opération est réalisée depuis la servlet, ce qui est logique puisque c'est elle qui décide d'appeler la vue, et ce via la méthode doGet().

```
...
public void doGet( HttpServletRequest request, HttpServletResponse response ) throws ServletException,
IOException {
        this.getServletContext().getRequestDispatcher( "/WEB-INF/test.jsp" ).forward( request, response );
}
```

Nous pouvons aussi transmettre des données depuis la servlet à la JSP.

c.Technologie JSP :
La technologie JSP est constitué de balises, des directives et actions JSP ainsi que des expressions EL, et la JSTL qui constitue la puissance des JSP.
c.1 Les balises :
Commentaires : entre <%-- et --%>
Balises de déclaration : <%! et %>
Balises de scriptlet (script + servlet): <% et %> sert à inclure du code Java.
Balises d'expression : Elle retourne le contenu d'une chaîne <%= " " %>.

c.2 Les directives :
Les directives JSP permettent :
- D'importer un package.
- D'inclure d'autres pages JSP.
- D'inclure des bibliothèques de balises.
- De définir des propriétés et informations relatives à une page JSP.

Elles sont toujours comprises entre les balises <%@ et %>, et sont à placer **en tête de page JSP**, hormis la directive d'inclusion de page qui peut être placée n'importe où.
Rôle : contrôlent comment le conteneur de servlets va gérer la JSP.

Les différentes directives sont :
Directive taglib : qui permet d'inclure une bibliothèque.
exp : <%@ taglib uri="maTagLib.tld" prefix="tagExemple" %>.
Directive page : définit des informations relatives à la page JSP.
Exemple importer des classes java : <%@ page import="java.util.List, java.util.Date" %>

D'autres options sont utilisables via cette directive, comme le **contentType** ou l'activation de la session.

Directive include : permet d'inclure le contenu d'un fichier dans un autre.
<%@ include file="uneAutreJSP.jsp" %>
Cette directive est utile pour découper une page web en plusieurs fragments.

l'inclusion est réalisée au moment de la compilation ; par conséquent, si le code du fichier est changé par la suite, les répercussions sur la page l'incluant n'auront lieu qu'après une nouvelle compilation.

Action standard include : <jsp:include page="test_inc.jsp" />
permet d'inclure du contenu de manière "dynamique".
Le contenu sera ici chargé à l'exécution, et non à la compilation.

Pour inclure un même *header* et un même *footer* dans toutes les pages de l'application ou du site web, il est préférable de spécifier directement ces portions communes dans le fichier **web.xml** du projet.

c.3 La portée des objets (visibilité ou scope): Un concept important intervient dans la gestion des objets par la technologie JSP : **la portée des objets**. Souvent appelée visibilité, ou *scope* en anglais, elle définit tout simplement leur **durée de vie**.

Il existe au total quatre portées différentes dans une application :

Page (jsp seulement) : les objets dans cette portée sont uniquement accessibles dans la page JSP en question.

Requête : les objets dans cette portée sont uniquement accessibles durant l'existence de la requête en cours.

Session : les objets dans cette portée sont accessibles durant l'existence de la session en cours.

Application : les objets dans cette portée sont accessibles durant toute l'existence de l'application.

c.4 Les actions standards :
Cette technologie est ancienne et n'est utilisée que dans des cas spécifiques. En effet la plupart des actions standards sont remplacées par les expressions EL.

useBean : permet d'utiliser un bean ou de le créer s'il n'existe pas.
<jsp: useBean id="exempleBean" class="com.beans.ExempleBean" scope="request" />
Cette action récupère un bean de type ExempleBean et nommé "exempleBean" dans la portée requête s'il existe, ou en crée un sinon.

getProperty : permet d'obtenir une des propriétés du bean qu'on a utilisé dans la page.
<jsp: useBean id=" exempleBean» class=" com.beans.ExempleBean» />
<jsp:getProperty name=" exempleBean" property="prenom" />

setProperty : permet de modifier une propriété du bean récupéré. Il existe quatre façons pour cela.
L'action suivante associe une valeur à la propriété 'prenom' du bean '=" exempleBean ' :
<jsp: setProperty name=" exempleBean" property="prenom" value="monPrenom" />

2. L'action suivante associe directement la valeur récupérée depuis le paramètre de la requête nommé ici 'prenomexempleBean ' à la propriété 'prenom' :

<jsp: setProperty name=" exempleBean" property="prenom" param="prenomexempleBean "/>

L'action suivante associe directement la valeur récupérée
depuis le paramètre de la requête nommé ici 'prenom' à la propriété de même nom :

<jsp: setProperty name=" exempleBean" property="prenom" />

L'action suivante associe automatiquement la valeur récupérée depuis chaque paramètre de la requête à la propriété de même nom :

```
<jsp: setProperty name=" exempleBean" property="*" />
```

Forward : permet d'effectuer une redirection vers une autre page.
L'action de forwarding *est ainsi limitée aux pages présentes dans le contexte de la servlet ou de la JSP utilisée :*

```
<jsp:forward page="/page.jsp" />
```

Lorsqu'on utilise le *forwarding*, **le code présent après cette balise dans la page n'est pas exécuté.**

c.5 Les expressions EL :

Les bases de l'*Expression Language*, que l'on raccourcit très souvent EL permettent une utilisation optimale des JSP.

Syntaxe : ${expression}

Ce qui est situé entre les accolades va être interprété.

Ces expressions permettent la réalisation des tests et peuvent inclure différents opérateurs :

opérateurs arithmétiques, applicables à des nombres : +, -, *, /, % ;
opérateurs logiques, applicables à des booléens : &&, ||, ! ;
opérateurs relationnels, basés sur l'utilisation des méthodes equals () et compareTo () des objets comparés : == ou eq, != ou ne, < ou lt, > ou gt, <= ou le, >= ou ge.


```
<!-- Compare les caractères 'a' et 'b'. Le caractère 'a' étant bien situé avant le caractère 'b' dans l'alphabet ASCII, cette EL affiche true. -->
    ${ 'a' < 'b' } <br />
<!-- Compare les chaînes 'hip' et 'hit'. Puisque 'p' < 't', cette EL affiche false. -->
```

on peut utiliser également des *simple quottes* (apostrophes) dans une expression EL.

Deux autres types de test sont fréquemment utilisés au sein des expressions EL :

les conditions ternaires, de la forme : test ? si oui : sinon ;
les vérifications si vide ou null, grâce à l'opérateur empty.

```
${'a'>'b' ?'oui' :'non'} < !—Le résultat de la comparaison vaut false, non est affiché -->
${empty 'test' ?'vide' :'non vide'} < ! – La chaine testée n'est pas vide, non vide est affiché -->
```

- sachez enfin que la valeur retournée par une expression EL positionnée dans un texte ou un contenu statique sera insérée à l'endroit même où est située l'expression :

La ligne suivante :

```
<p>12 est inférieur à 8 : ${12 lt 8 }.</p>
```

Sera rendue ainsi après interprétation de l'expression, 12 n'étant pas inférieur à 8 :
12 est inférieur à 8 : false.

La manipulation d'objets :
Les beans :
les EL permettent d'accéder à une propriété du bean :

${exempleBean.prenom}

Au lieu d'accéder par une action standard :

<jsp:getProperty name="exempleBean" property="prenom" />

les expressions EL sont protégées contre un éventuel retour null :

L'expression EL suivante n'affiche rien si la propriété "prenom" n'a pas été initialisée, et n'affiche rien si l'objet "coyote" n'a pas été initialisé :

${exempleBean.preno}

les collections : les EL permettent d'accéder aux collections. Cela inclut donc tous les objets de type java.util.List,java.util.Set, java.util.Map, etc.

```
<!-- Les quatre syntaxes suivantes retournent le deuxième élément de la liste de légumes  -->
        ${maList.get(1) }<br />
        ${maList [1]}<br />
        ${maList['1']}<br />
        ${maList["1"]}<br />
```


l'indice du premier élément d'une collection est 0.

tableaux : les trois syntaxes suivantes sont équivalentes :

```
 ${tab[2] }<br />
${tab['2'] }<br />
${tab["2"] }<br />
```

Sachez par ailleurs qu'il est impossible d'utiliser directement l'opérateur *point* pour accéder à un élément d'une liste ou d'un tableau. Les syntaxes ${entiers.1} ou ${animaux.2} enverront une exception à l'exécution

la map :

Les quatre syntaxes suivantes retournent la valeur associée à la clé "cle" de la Map maMa

```
${ maMap.cle }<br />
${ maMap.get("cle") }<br />
${ maMap['cle'] }<br />
${ maMap["cle"] }<br />
```

Désactiver l'évaluation des expressions EL :

il est possible de désactiver l'évaluation des expressions EL :

grâce à la directive **page : <%@ page isELIgnored ="true" %>**

Avec le fichier web.xml : désactiver l'évaluation des expressions EL sur tout un ensemble de pages JSP dans une application grâce à l'option <el-ignored> du fichier web.xml.

Elle se présente dans une section de cette forme :

```
<jsp-config>
   <jsp-property-group>
      <url-pattern>*.jsp</url-pattern>
      <el-ignored>true</el-ignored>
   </jsp-property-group>
</jsp-config>
```

Comportement par défaut : la valeur de l'option **isELIgnored** dépend de la version de l'API servlet utilisée par l' application :

- si la version est supérieure ou égale à 2.4, alors les expressions EL seront **évaluées par défaut**.
- si la version est inférieure à 2.4, alors il est possible que les expressions EL soient **ignorées par défaut.**

Comment savoir si une application permet d'utiliser les expressions EL ou non ?:
il faut s'assurer que l'application est déclarée correctement pour utiliser cette version.
Cela se passe au niveau de la balise <web-app> du fichier **web.xml**.


```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
 xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
 version="3.0">
  ...
/web-app>
```

c.6 La technologie JSTL :
La JSTL est une bibliothèque, une collection regroupant des balises implémentant des fonctionnalités à des fins générales, communes aux applications web. Citons par exemple la mise en place de boucles, de tests conditionnels, le formatage des données ou encore la manipulation de données XML.
Son objectif est de permettre au développeur d'éviter l'utilisation de code Java dans les pages JSP, et ainsi de respecter au mieux le découpage en couches recommandé par le modèle MVC.
En apparence, ces balises ressemblent aux balises JSP.

Versions de la JSTL :
La JSTL a fait l'objet de plusieurs versions :
JSTL 1.0 pour la plate-forme J2EE 3, et un conteneur JSP 1.2 *(ex: Tomcat 4)*.
JSTL 1.1 pour la plate-forme J2EE 4, et un conteneur JSP 2.0 *(ex: Tomcat 5.5)*.

JSTL 1.2, qui est partie intégrante de la plate-forme Java EE 6, avec un conteneur JSP 2.1 ou 3.0 *(ex: Tomcat 6 et 7)*.
Les différences entre ces versions résident principalement dans le conteneur JSP nécessaire.
Le conteneur JSP 1.2 sur lequel est basée la JSTL 1.0 ne gérait pas les expressions EL.
La version 1.1 est basée sur le conteneur JSP 2.0, qui intègre nativement un interpréteur d'expressions EL.
la JSTL 1.2, apporte des EL "unifiées", ainsi qu'une meilleure intégration dans le framework JSF.

Configuration de la JSTL :
Pour pouvoir utiliser la technologie JSTL, il faudrait au préalable ajouter la librairie **jstl-1.2.jar** au dossier lib situé dans le dossier WEB-INF du projet.
Ensuite ajouter la directive suivante à la page JSP :
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
Cette directive devra **être présente sur chacune des pages** utilisant les balises JSTL.
Bibliothèques de la JSTL :
La JSTL contient cinq bibliothèques :
fmt : destinée au formatage et au parsage des données. Permet notamment la localisation de l'affichage ;
fn : destinée au traitement de chaînes de caractères.
sql : destinée à l'interaction avec une base de données.
xml : destinée au traitement de fichiers et données XML.
Core : que nous détaillons ci-après.

Bibliothèque Core :
La bibliothèque *Core*, offre des balises pour les principales actions nécessaires dans la couche présentation d'une application web.
Affichage d'une expression :
La balise utilisée pour l'affichage est <c:out value="" />
<c:out value="test" <%-- Affiche test --%>
c:out value="${ 'a' < 'b' }" <%-- Affiche true --%>
L'utilité qu'apporte cette balise par rapport à l'affichage direct avec des expressions régulières est qu'elle permet d'échapper automatiquement les caractères spéciaux de nos textes et rendus d'expressions.
Il faut prendre l'habitude d'utiliser ce tag JSTL lorsque vous affichez des variables, notamment lorsqu'elles sont récupérées depuis le navigateur, c'est-à-dire lorsqu'elles sont saisies par l'utilisateur.
<input type="text" name="donnee" value="<c:out value="${donneeSaisieParUnUtilisateur}"/>" />
les données récupérées depuis un formulaire sont potentiellement dangereuses, puisqu'elles permettent **des attaques de type XSS ou d'injection de code**.
L'utilisation du tag <c:out> permet d'échapper les caractères spéciaux responsables de cette faille, et ainsi de prévenir tout risque à ce niveau.

Gestion d'une variable

Création :
La balise utilisée pour la création d'une variable est <c:set>.
<c:set var="message" value="Chaine" scope="request" />
Cette balise met l'expression "Chaine" dans l'attribut "message" de la requête.

Pour récupérer cette valeur pour affichage, on procède comme suit :
<c:out value="${requestScope.message}" />

Modification
La modification d'une variable s'effectue de la même manière que sa création.

Le code suivant créera une variable nommée "maVariable" si elle n'existe pas déjà, et initialisera son contenu à "12".
<c:set var="maVariable" value="12" />
Il est également possible d'initialiser une variable en utilisant le corps de la balise, plutôt qu'en utilisant l'attribut **value**.
c:set var="maVariable"> 12 c:set
Il est possible d'imbriquer d'autres balises dans le corps de cette balise, et pas seulement d'utiliser de simples chaînes de caractères ou expressions.
c:set var="locale" scope="session">
c:out value="${param.lang}" default="FR"/>
c:set
Ce code permet initialiser la valeur d'une variable de session depuis une valeur lue dans un paramètre de l'URL.
Modification des propriétés d'un objet :
Il est possible de définir ou modifier une valeur particulière lorsqu'on travaille sur certains types d'objets comme les propriétés d'un bean par exemple.
Pour cela, on ajoute deux attributs :
target : contient le nom de l'objet dont la propriété sera modifiée.
property : contient le nom de la propriété qui sera modifiée.
<c:set target="${monBean}" property="prenom" value="monPrenom"/>
Ce code Permet de modifier la propriété prenom du bean monBean.
<c:set target="${ monBean }" property="prenom" value="${null}" />
Passer à nul la propriété prenom du bean monBean.
lorsque l'objet traité n'est pas un bean mais une simple Map, cette action a pour effet de directement supprimer l'entrée de la Map concernée.

Suppression :
La balise dédiée à cette tâche, avec pour seul attribut requis **var** est :
c:remove var="maVariable" scope="session" />
Supprime la variable ma variable de la session.

Les conditions :

Les condition simple :
permet de tester une seule expression, et correspond au bloc if() du langage Java. Le seul attribut obligatoire est **test**.
c:if test="${ 12 > 7 }" var="maVariable" scope="session">
Ce test est vrai.
c:if
Le corps de la balise ne sera affiché dans la page finale que si la condition est vraie, à savoir si l'expression contenue dans l'attribut **test** renvoie **true**. Le résultat du test conditionnel sera stocké dans la variable et dans le *scope* défini, et sinon dans le *scope* page par défaut.
Des conditions multiples
La seconde méthode fournie par la JSTL est utile pour traiter les conditions mutuellement exclusives, équivalentes en Java à une suite de if () / else if () ou au bloc switch (). Elle est en réalité constituée de plusieurs balises :
c:choose
<c:when test="${expression}">Action ou texte.</c:when>
...
<c:otherwise>Autre action ou texte.</c:otherwise>
c:choose

La balise <c:choose> peut contenir qu'une ou plusieurs balises <c:when> et une ou zéro balise <c:otherwise>.
La balise <c:when> n'existe qu'à l'intérieur d'une balise <c:choose> ; équivalente au mot-clé **case** en Java, dans un bloc switch().Tout comme la balise <c:if>, elle doit obligatoirement se voir définir un attribut **test** contenant la condition. À l'intérieur d'un même bloc<c:choose>, un seul <c:when> verra son corps évalué, les conditions étant mutuellement exclusives.
La balise <c:otherwise> ne peut également exister qu'à l'intérieur d'une balise <c:choose>, et après la dernière balise <c:when>. Elle est l'équivalent du mot-clé **default** en Java, dans un bloc switch().
Avec la JSTL, deux choix vous sont offerts, en fonction du type d'élément que vous souhaitez parcourir avec votre boucle : <c:forEach> pour parcourir une collection, et<c:forTokens> pour parcourir une chaîne de caractères.

```
  c:forEach var="i" begin=" " end=" " step=" "
……
   c:forEach
```

L'attribut var permet d'accéder au compteur.

Itération sur une collection :

On ajoute un attribut **items** à la boucle simple qui indiquera la collection à parcourir.

```
<c:forEach items="${maListe}" var=" ">
…….
</c:forEach >
```

les différentes collections sur lesquelles il est possible d'itérer avec la boucle <c:forEach> de la bibliothèque *Core* :
java.util.Collection.
java.util.Map.
java.util.Iterator.
java.util.Enumeration.
Array d'objets ou de types primitifs.
(Chaînes de caractères séparées par des séparateurs définis).

L'attribut varStatus :
Le **varStatus** permet de définir un ensemble de propriétés définissant l'état courant d'une itération
Exemple de propriétés :
First : booléen précisant si l'itération courante est la première.
Count : compteur d'itération (commence à 1).
Current : élément courant de la collection parcourue.

Itération sur une chaîne de caractères :
Une variante de la boucle <c:forEach> existe, spécialement dédiée aux chaînes de caractères.
<c :forTokens/>.

```
<c:forTokens var="sousChaine" items="chaine séparée avec séparateurs" delims=";,+"
  ${sousChaine}   br/>
    c:forTokens
```

L'attribut **delims :** spécifie quels sont les caractères qui serviront de séparateurs dans la chaîne que la boucle parcourra.

Les liens : voici la syntaxe.

```
<c:url value="test.jsp" var="lien"
```

L'attribut **value** contient logiquement l'adresse, et l'attribut **var** permet comme pour les tags vus auparavant de stocker le résultat dans une variable.
Les trois fonctionnalités associées à la balise :

ajouter le nom du contexte aux URL absolues : sans cette propriété, il serait nécessaire d'écrire en dur le contexte de l'application dans les URL lors du développement, et de toutes les modifier si le contexte est changé par la suite.
réécrire l'adresse pour la gestion des sessions (si les cookies sont désactivés ou absents, par exemple) pour intégrer l'identifiant de session en question.
encoder les noms et contenus des paramètres de l'URL : les caractères spéciaux qu'ils contiennent éventuellement vont être transformés en leurs codes HTML respectifs.
Ainsi, une url générée de cette manière :

```
  c:url value="/monSiteWeb/test.jsp">
 <c:param name="param1" value="123"/>
 <c:param name="param2" value=" des données contenant des c@r#ct%res bi&a**es "/>
   c:url
```

Sera rendu comme suit si le cookie est absent :

/Test/monSiteWeb/countZeros.jsp;jsessionid=A6B57CE08012FB431D?nbZeros=123&param2=des+donn%e9es+contenant+des+c%40r%23ct%25res+bi%26a**es

On remarque :

- l'ajout automatique du contexte en début de l'URL absolue.
- l'encodage automatique des paramètres.
- l'ajout automatique de l'identifiant de session avant les paramètres de la requête.

Dans ce chapitre est présentée la configuration de l'IDE Eclipse et les étapes de création d'une application web dynamique qui inclut la mise en place du serveur Tomcat pour exécuter l'application concernée.
Pour bien débuter avec l'environnement Eclipse, la structure globale d'une application web dynamique sous Eclipse est mise en évidence.
Comme l'architecture d'une application web dynamique javaEE est basée sur le patron de conception MVC, une vue globale sur les patrons de conception est faite.
Enfin la présentation de chacune des couches du modèle MVC ainsi que les technologies de chaque couche.

Dans ce qui suit sera présentée la réalisation de l'application en suivant le modèle MVC après avoir présenter la conception de la solution faite dans le cadre du langage de modélisation orienté objet UML2.0 Qui fera l'objet du chapitre suivant.

Chapitre III : Le langage UML.

Depuis des temps très anciens, les hommes ont cherché un langage à la fois universel et synthétique, et leurs recherches les ont amenés à découvrir des images, des symboles qui expriment, en les réduisant à l'essentiel, les réalités les plus riches et les plus complexes.
Les images, les symboles parlent, ils ont un langage.
O.M. Aïvanhov

I. Apparition d'UML :

Pour pouvoir suivre l'évolution des systèmes d'information de plus en plus complexes de nouvelles méthodes ont vu le jour. Les méthodes classiques tel que MERISE ont dû s'adapter en y introduisant les concepts liés à la démarche objet.
La modélisation objet consiste à créer une représentation informatique des éléments du monde réel, sans se préoccuper de l'implémentation, ce qui signifie indépendamment d'un langage de programmation. Il s'agit donc de déterminer les objets présents et d'isoler leurs données et les fonctions qui les utilisent. Pour cela, des méthodes de conception et de développement orientées objet ont été mises au point. Entre 1970 et 1990, de nombreux analystes ont mis au point des approches orientées objets, si bien qu'en 1994 il existait plus de 50 méthodes objet. Toutefois seules 3 méthodes ont véritablement émergées :

La Méthode OMT De Rumbaugh
La Méthode BOOCH'93 de Booch
La Méthode OOSE de Jacobson

Devant le grand nombre de méthodes orientées objet, l'OMG1 (Object Management Group) a procédé à la définition d'une notation standard utilisable dans le développement des applications informatiques orientées objet. C'est ainsi qu'est apparu UML (UnifiedModelingLanguage) langage de modélisation unifié en 1994, où Rumbaugh et Booch (rejoints en 1995 par Jacobson) ont unis leurs efforts pour mettre au point cette méthode, qui permet de définir une notation standard en incorporant les avantages de chacune des méthodes précédentes (ainsi que celles d'autres analystes). Les trois auteurs James Rumbaugh, Ivar Jacobson et Grady Booch sont les pionniers du langage UML.

II. Présentation d'UML :

Le langage UML (UnifiedModelingLanguage) langage de modélisation unifié est un langage de modélisation visuel conçu pour analyser et concevoir les systèmes d'information, en utilisant les techniques orientés objet.

UML n'est pas une méthode de conception (*i.e.* c'est une description normative des étapes de la modélisation) car il est constitué uniquement d'un ensemble de diagrammes de modélisation et il ne comporte pas une méthodologie complète de développement : ses auteurs ont en effet estimé qu'il n'était pas opportun de définir une méthode en raison de la diversité des cas particuliers. Ils ont préféré se borner à définir un langage graphique qui permet de représenter et de communiquer les divers aspects d'un système d'information ainsi que des textes qui expliquent les contenus des graphes.

UML n'est pas un langage de programmation, il peut être exploité pour rédiger des programmes, ce n'est pas un langage au vrais sens du terme, car il n'a pas encore atteint la rigueur syntaxique et sémantiques des langages de programmation (ne possède pas les avantages et sémantique d'un langage de programmation).

UML est aujourd'hui l'outil incontournable dans les projets de développement des applications informatiques.

III. Concepts de l'approche objet :

III.1. Objet
Un objet est une entité du monde réel (ou du monde virtuel pour les objets immatériels) qui a une identité qui lui est propre et qui permet de le distinguer d'un autre objet, des états significatifs qui représentent des valeurs à un instant donné au niveau de ses attributs et un comportement ou ensemble d'opération qu'il peut effectuer en fonction des messages reçus des autres objets. On dira que tout objet est une instance de classe.

III.2. Classe
Une classe d'objets décrit un groupe d'objets ayant des attributs similaires, des opérations (comportement) identiques, des relations communes avec les autres objets. Une classe permet de créer une abstraction des objets porteurs d'informations qui leur sont propres, les objets eux appartiennent à une classe qui contient les généralités les caractérisant. Une classe peut avoir un attribut considéré comme identifiant.

III.3. Attribut
Un attribut est une valeur de données détenue par les objets d'une classe. Nom et page sont les attributs de la classe Livre.

III.4. Opérations et Méthodes
Les opérations sont des fonctions applicables aux objets d'une classe. La méthode est l'implémentation de l'opération pour une classe. En effet l'opération <Imprimer> de la classe livre est implémentée avec un code diffèrent selon qu'il s'agit d'imprimer en ASCII ou en binaire.

III.5. Association
L'association, est une simple connexion bidirectionnelle représentant l'abstraction des liens qui relient des objets du monde réel ayant une même structure, une même sémantique. Elle n'est pas porteuse comme dans d'autres méthodes plus classiques d'une véritable représentation graphique (exemple, Merise).

III.6. Lien
Un lien est une relation physique ou conceptuelle entre deux objets (ou plus). De même que les objets sont instances de classes, les liens sont instances des relations entre les classes.

III.7. Agrégation
L'agrégation est une relation « composé-composant » ou « partie de » dans laquelle les objets représentant les composants d'une chose sont associés à un objet représentant l'assemblage entier.

III.8. Généralisation
La généralisation de classe consiste à factoriser dans une classe appelée super-classe les attributs et/ou les opérations des classes considérées.

III.9. Spécialisation
C'est l'inverse d'une généralisation, puisqu'elle consiste à créer à partir d'une classe, plusieurs classes spécialisées.

III.10. Polymorphisme
Le polymorphisme permet à une opération définie dans une classe à s'exécuter différemment dans une sous-classe.

III.11. Encapsulation
Les composants développés sous forme de « briques » ont leurs informations internes occultées. Ce sont des « boîtes noires ». Une interface permet de les connecter à l'application.

III.12. Persistance
La persistance permet à une propriété donnée à un objet de continuer à exister après la fin de l'exécution du programme qui l'a créé.

IV. Présentation générale des différents diagrammes d'UML :
Un diagramme UML est une représentation graphique, qui s'intéresse à un aspect bien précis du modèle ; c'est une perspective du modèle. Les diagrammes permettent d'inspecter un modèle selon différentes perspectives.

Pour modéliser un système complexe, un seul diagramme ne suffit pas, c'est la raison pour laquelle UML comprend neufs types de diagrammes (versions antérieures à UML 2) essentiels qui sont les plus utilisés. Combinés, les différents diagrammes UML offrent une vue presque complète du système en question.

UML dans sa version 2 propose treize diagrammes qui peuvent être utilisés dans la description d'un système. Ces diagrammes sont regroupés dans deux grands ensembles : les diagrammes structurels et les diagrammes de comportement.

IV.1 Les diagrammes structurels : Ces diagrammes, au nombre de six, ont vocation à représenter l'aspect statique d'un système (classes, objets, composants…).

Diagramme **de classe :** Ce diagramme représente la description statique du système en intégrant dans chaque classe la partie dédiée aux données et celle consacrée aux traitements. C'est le diagramme pivot de l'ensemble de la modélisation d'un système.

Diagramme d'objet : Le diagramme d'objet permet la représentation d'instances des classes et des liens entre instances.

Diagramme de composant (modifié dans UML 2) : Ce diagramme représente les différents constituants du logiciel au niveau de l'implémentation d'un système.

Diagramme de déploiement (modifié dans UML 2) : Ce diagramme décrit l'architecture technique d'un système avec une vue centrée sur la répartition des composants dans la configuration d'exploitation.

Diagramme de paquetage (nouveau dans UML 2) : Ce diagramme donne une vue d'ensemble du système structuré en paquetage. Chaque paquetage représente un ensemble homogène d'éléments du système (classes, composants…).

Diagramme de structure composite (nouveau dans UML 2) : Ce diagramme permet de décrire la structure interne d'un ensemble complexe composé par exemple de classes ou d'objets et de composants techniques. Ce diagramme met aussi l'accent sur les liens entre les sous-ensembles qui collaborent.

IV.2 Les diagrammes de comportement : Ces diagrammes représentent la partie dynamique d'un système réagissant aux événements et permettant de produire les résultats attendus par les utilisateurs. Sept diagrammes sont proposés par UML :

Diagramme des cas d'utilisation : Ce diagramme est destiné à représenter les besoins des utilisateurs par rapport au système. Il constitue un des diagrammes les plus structurants dans l'analyse d'un système.

Diagramme d'état-transition (machine d'état) : Ce diagramme montre les différents états des objets en réaction aux événements.

Diagramme d'activités (modifié dans UML 2) : Ce diagramme donne une vision des enchaînements des activités propres à une opération ou à un cas d'utilisation. Il permet aussi de représenter les flots de contrôle et les flots de données.

Diagramme de séquence (modifié dans UML 2) : Ce diagramme permet de décrire les scénarios de chaque cas d'utilisation en mettant l'accent sur la chronologie des opérations en interaction avec les objets.

Diagramme de communication (anciennement appelé collaboration) : Ce diagramme est une autre représentation des scénarios des cas d'utilisation qui met plus l'accent sur les objets et les messages échangés.

Diagramme global d'interaction (nouveau dans UML 2) : Ce diagramme fournit une vue générale des interactions décrites dans le diagramme de séquence et des flots de contrôle décrits dans le diagramme d'activités.

Diagramme de temps (nouveau dans UML 2) : Ce diagramme permet de représenter les états et les interactions d'objets dans un contexte où le temps a une forte influence sur le comportement du système à gérer.

Chapitre IV : Conception&Réalisation
I. Conception

La solution proposée dans cette partie consiste à améliorer la solution proposée dans la première partie en utilisant la plate forme JEE avec l'IDE Eclipse en respectant le MVC.

La solution a gardé l'organisation des taches existante et ce en créant un compte pour chaque type de tache et en ajoutant un compte administrateur de l'application.
Ainsi :
L'employé chargé des réalisations peut accéder aux formulaires de saisie des différentes informations (concernant les établissements, les articles, les réalisations), et ainsi joue le rôle de saisie (agent de saisie).
L'employé chargé du bilan aura accès à l'affichage du bilan.
L'administrateur de l'application qui a pour rôle la gestion des comptes d'utilisateurs (ajout, suppression)

Les détails de la solution sont expliqués dans ce qui suit.

a. **Le diagramme de cas d'utilisation :**

a.1Identification des acteurs :
Un acteur est un rôle joué par une personne physique ou un autre systéme.il représente « un type d'utilisateurs ».

Dans le but de garder la répartition des taches (l'agent de saisie, l'employé chargé du bilan) nous avons désigné les acteurs suivants pour notre système:
Administrateur.
Agent de saisie.
Chargé du bilan.

a.2Identification des cas d'utilisation :
Ils décrivent le comportement d'un système du point de vue de ses utilisateurs et les interactions entre les utilisateurs d'un système et le système lui même dans notre cas d'utilisations pour les acteurs définis précédemment sont :

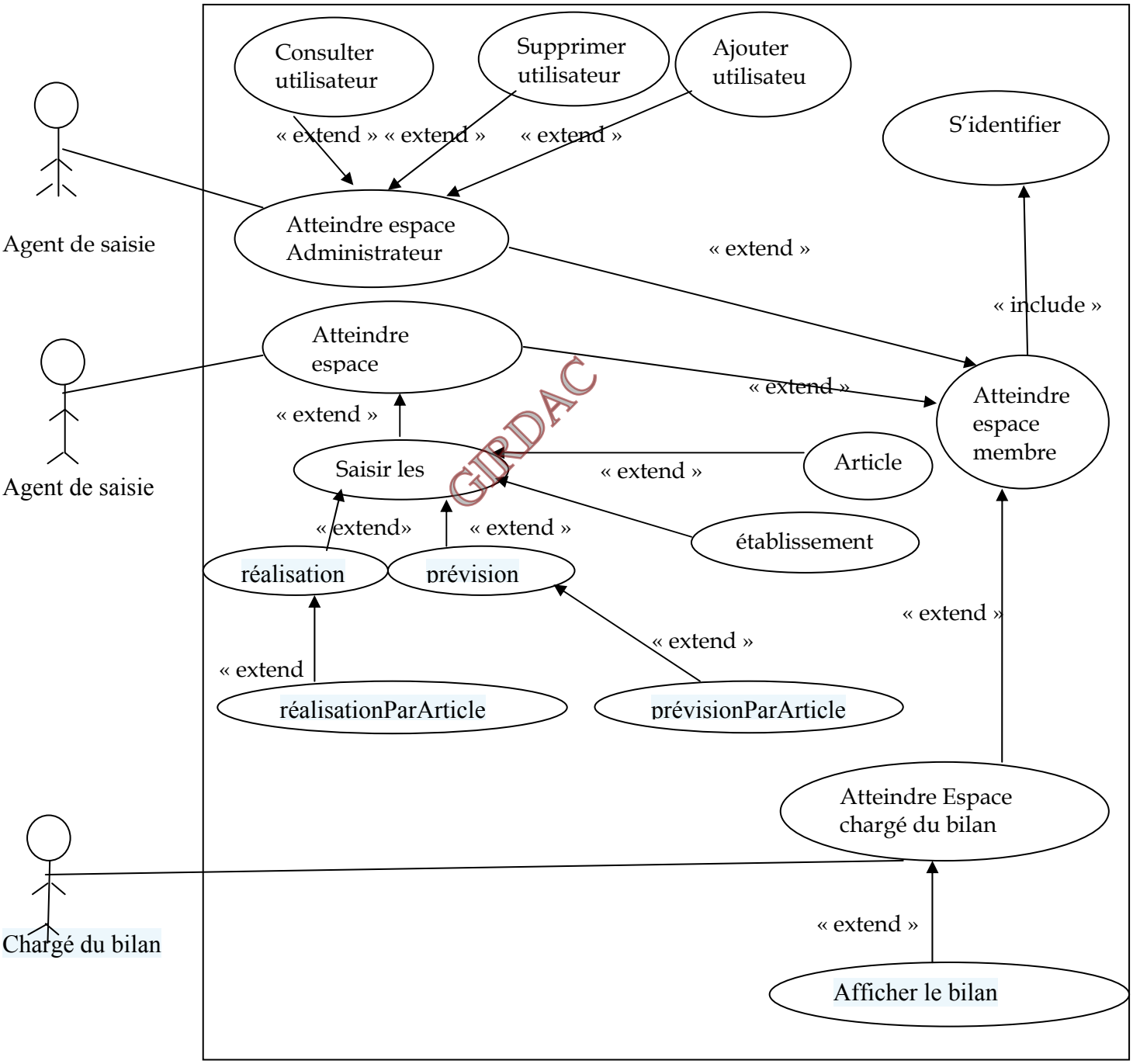
Cas d'utilisation relatif a l'administrateur :

S'identifier
Changer le mot de passe
Ajouter un agent de saisie
Consulter un agent de saisie
Ajouter un chargé du bilan
Consulter un chargé du bilan
Supprimer un agent de saisie
Supprimer un chargé du bilan

Cas d'utilisation relatif à l'agent de saisie :
S'identifier
Changer le mot de passe
Ajouter un établissement
Ajouter un article d'établissement
Saisir les informations relatives à chaque article d'établissement de chaque commune.

Cas d'utilisation relatif au chargé du bilan :
S'identifier
Changer le mot de passe
Afficher le bilan

a.3Diagramme global de cas d'utilisation



a.4 Description du cas d'utilisation « s'identifier » :
Titre : « s'identifier »
Acteur : Utilisateur (Administrateur, Agent de saisie, Chargé du bilan).
Scénario :

1. Exécuter application
2. La page de connexion s'affiche.
3. Saisir le user, mot de passe, profile.
4. La servlet Connexion (qui est un objet) appelle l'objet ConnexionForm.
5. l'objet ConnexionForm effectue une vérification sur le type du profile et retourne le résultat à l'objet Connexion.
6. L'objet Connexion donne accès à l'espace membre correspondant au profile retourné.

Enchainement alternatif :
Si le profile n'est pas saisi alors retourner à la page connexion.

a.5 Description du cas d'utilisation « changer mot de passe » :
Titre : « changer mot de passe »
Acteur : utilisateur
Scénario :

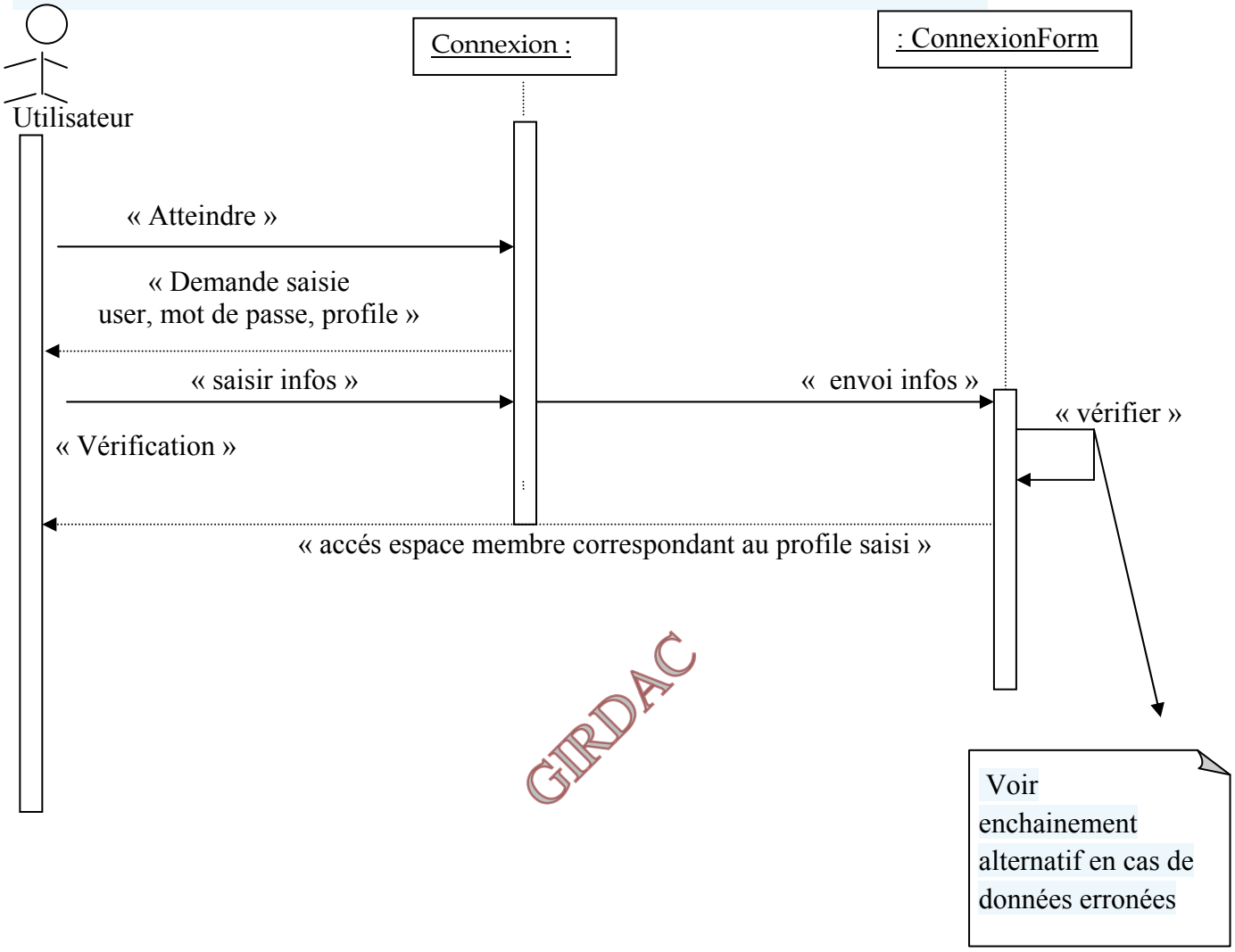
1. Atteindre formulaire changement du mot de passe.
2. Demande de remplir les informations nécessaires (user, ancien mot de passe, nouveau mot de passe, confirmation du nouveau mot de passe).
3. Le système vérifie si l'ancien mot de passe correspond au user.
4. Le système vérifie si le mot de passe et sa confirmation se correspondent.

Enchainement alternatif :

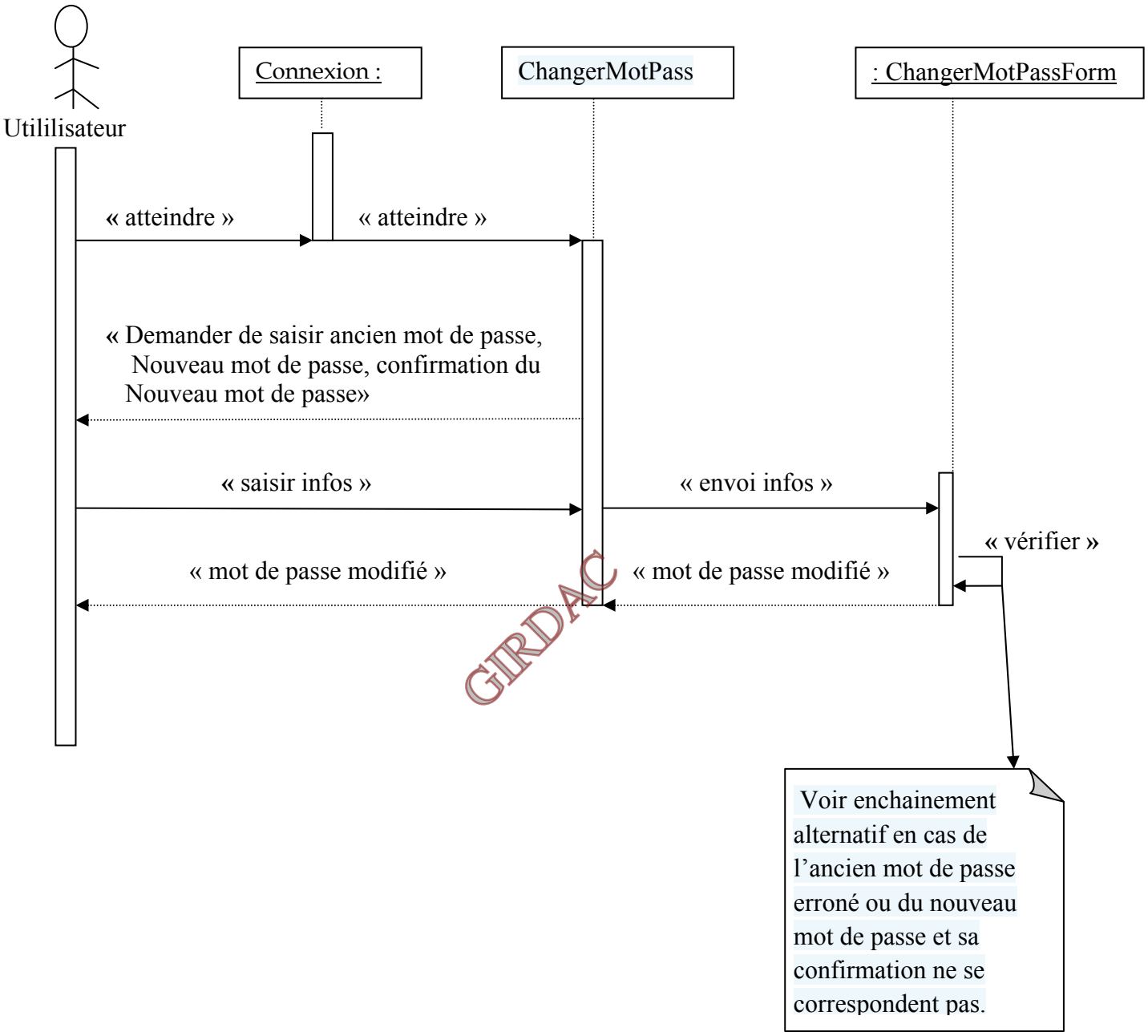
- Si l'ancien mot de passe erroné afficher message «ancien mot de passe erroné, êtes vous sur du user ? ». et renvoi au formulaire changement du mot de passe
- Si le nouveau mot de passe et sa confirmation ne sont pas les mêmes, afficher message « les deux mots de passe rentrés ne sont pas les mêmes » et renvoi au formulaire changement du mot de passe.

b. Diagramme de séquence :

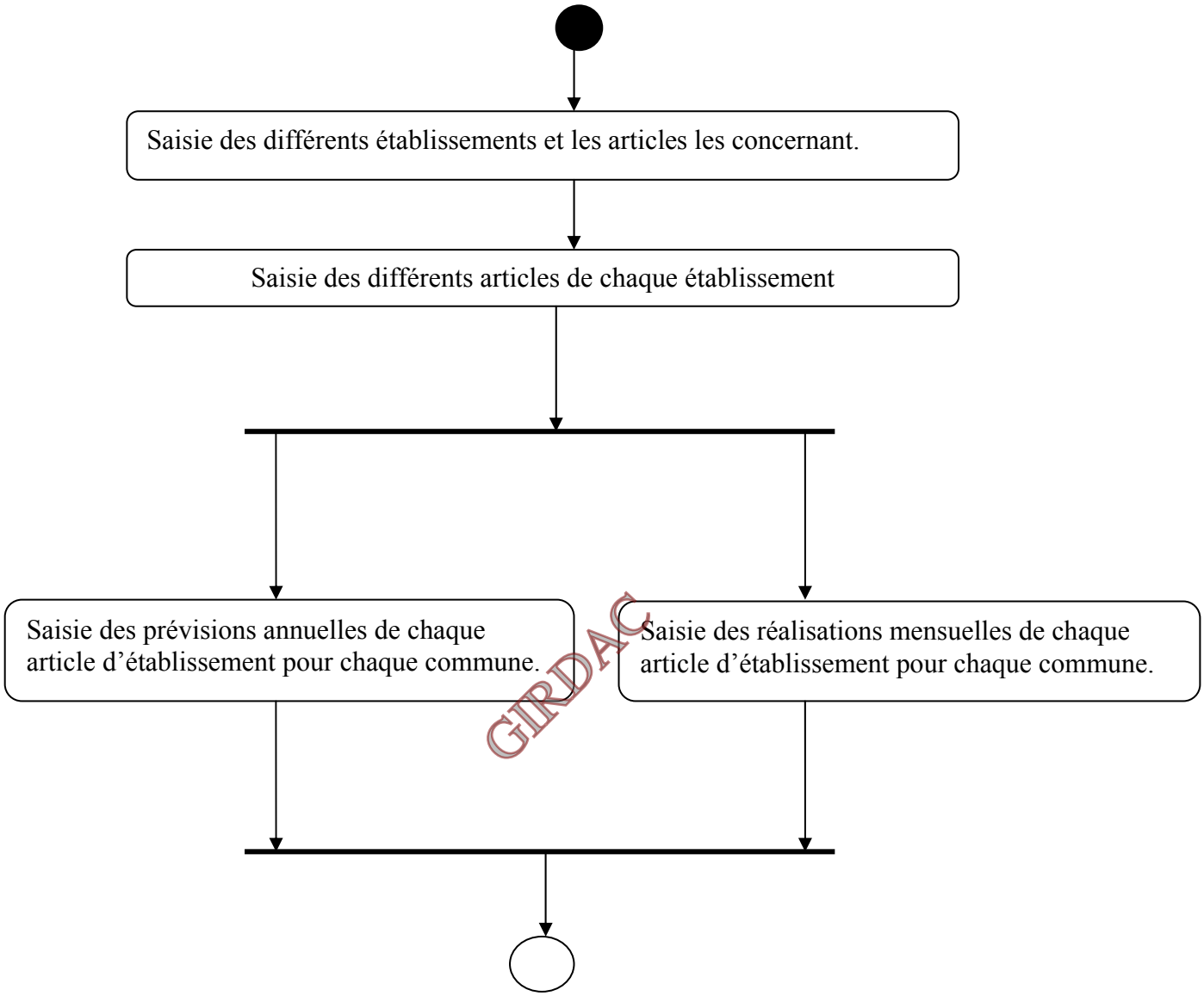
b.1 Diagramme de séquence du cas d'utilisation « s'identifier ».



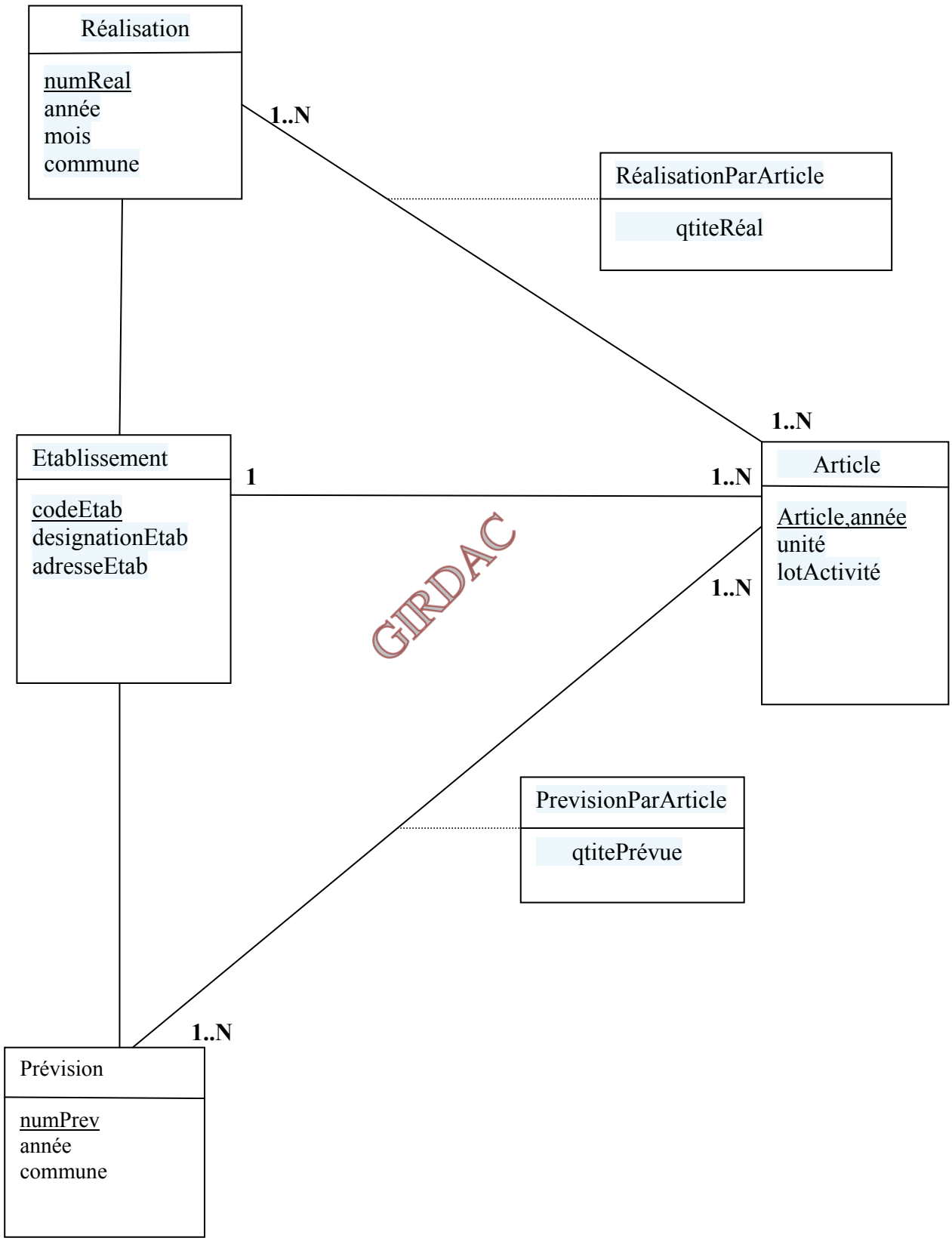
b.2 Diagramme de séquence du cas d'utilisation « changer mot de passe » :



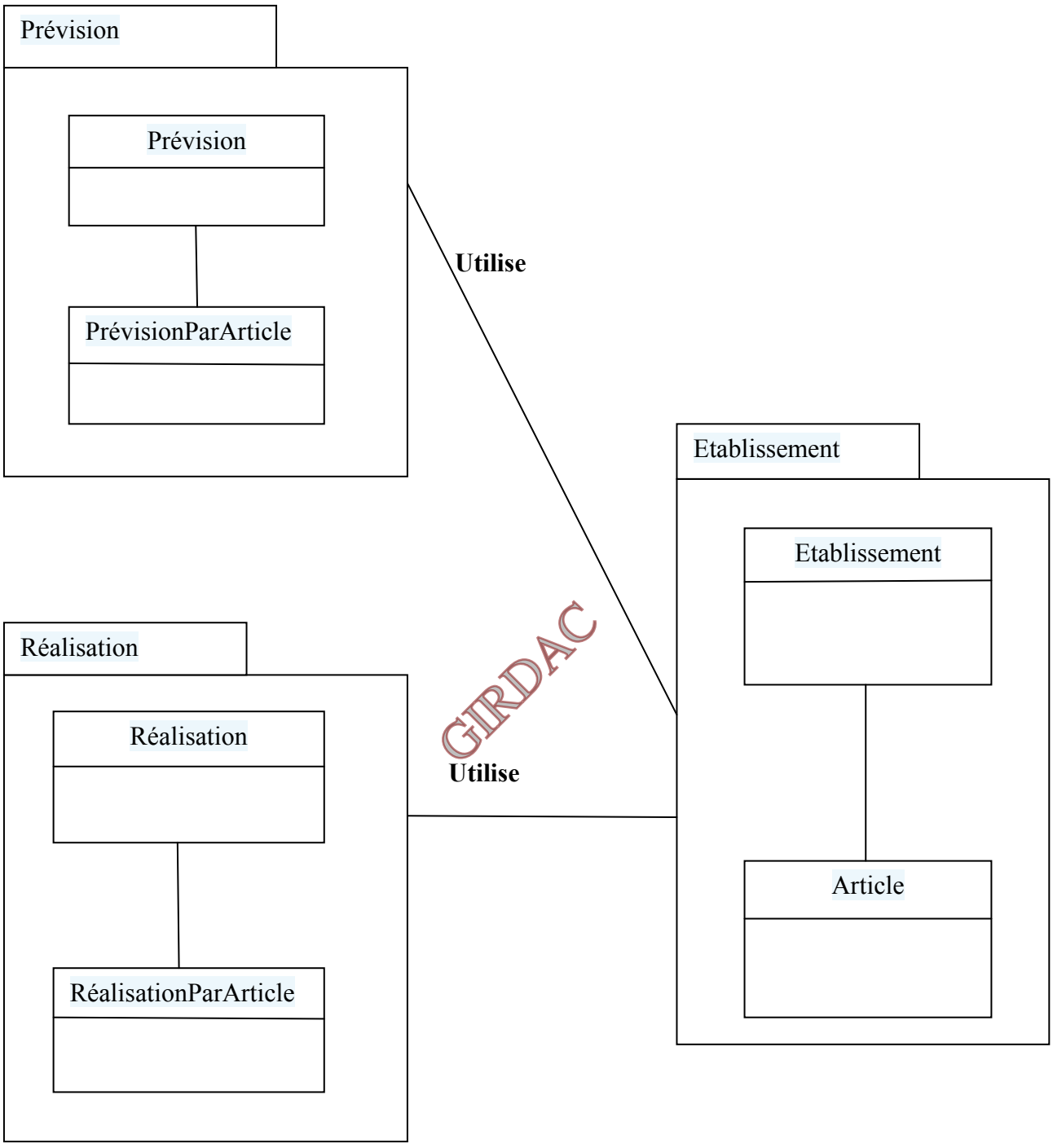
c. Diagramme d'activité relatif à l'agent de saisie :



d. Diagramme de classes



Diagrammes de packages pour les classes :



II. Réalisation :

Introduction

Dans le but de comprendre le principe, j'ai procédé à la réalisation en se basant sur le MVC, sans utiliser aucun Framework.

La solution selon le modèle MVC est constituée du trio Modèle, Vue, Contrôle

La vue : se sont les différentes pages JSP.

Le Contrôle est constitué des différentes servlets.

Le modèle : constitué des objets métiers qui comportent l'implémentation des différents traitements et des différents beans qui permettent de stocker les données.

Pour mieux organiser le code, il est nécessaire de mettre chaque type d'objet dans un package à part. Ainsi on met les servlets dans un package, les JSP dans un package, les beans dans un package, les objets métier dans un package. Cette organisation est illustrée dans le diagramme de paquetage suivant :

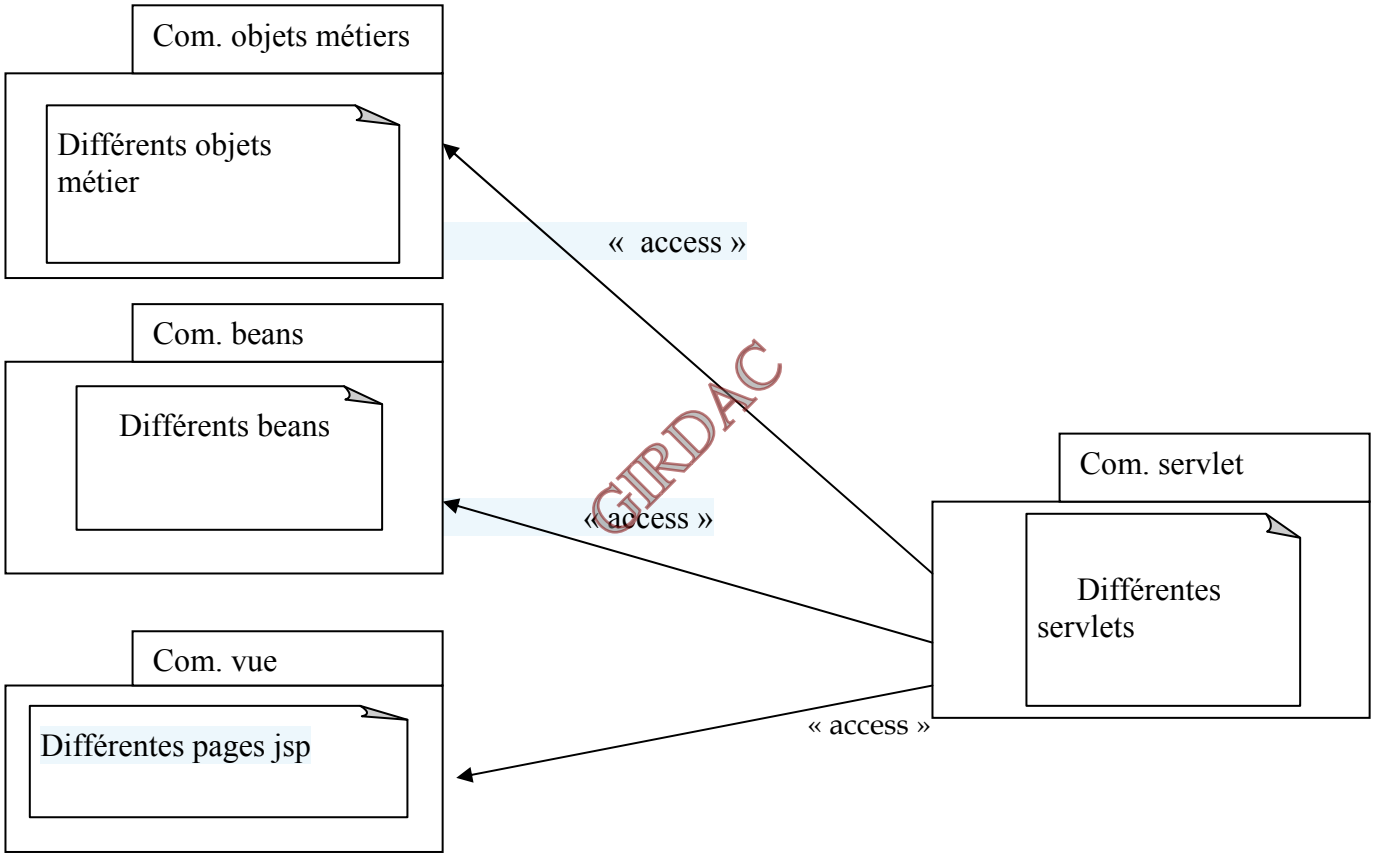


Diagramme de paquetage correspondant à l'organisation des composants de l'application

1.Composants de l'application :

1.a Modèle : qui englobe les traitements

Mappage des données avec des objets java :

Consiste à faire correspondre les attributs d'une fiche du système de stockage (BDD) avec les attributs d'un objet (objet Java en ce qui nous concerne et plus précisément les beans).

La construction des beans se fait en respectant les règles suivantes :

A chaque table correspond un bean.

La clé étrangère dans une table se transforme en un attribut de type objet dans un bean.

Les beans correspondant à notre réalité sont :

```
public class Etablissement {
        private String codeEtab;
        private String nomEtab;
        private String adresseEtab;


        public String getCodeEtab(){
                return codeEtab;
        }

        public void setCodeEtab(String codeEtab){
                this.codeEtab = codeEtab;
        }

        public String getNomEtab(){
                return nomEtab;
        }

        public void setNomEtab(String nomEtab) {
                this.nomEtab = nomEtab;
        }

        public String getAdresseEtab(){
                return adresseEtab;
        }

        public void setAdresseEtab(String
adresseEtab){
            this.adresseEtab = adresseEtab;


}
}
```

```
public class Article {
        private String nomArt;
        private String unite;
        private String lotAct;
        private Etablissement etablissement;

        public String getNomArt(){
                return codeEtab;
        }
public void   set (String nomArt ){
                this.nomArt = nomArt ;
        }

public String getUnite(){
                return unite;
        }

public void   setUnite (String unite){
                this.unite = unite;
        }
public String getLotAct(){
                return lotAct;
        }

public void   setLotAct (String codeEtab){
                this.lotAct = lotAct ;
        }

public String getEtablissement(){
                return etablissement;
        }

public void   setEtablissement (String
etablissement){
                this.etablissement = etablissement;
        }
```

```
public class Utilisateur {
        private Long id;
        private String nom;
        private String prenom;
        private String user;
        private String motDEpass;
        private String profile;

public Long getId() {
    return id;
}
  public void setId( Long id )
{ this.id = id;
 }
public String getNom(){
return nom;
}
   public void setNom (String nom){
        this.nom = nom;
   }
     public String getPrenom(){
      return prenom;
}
   public void setPrenom(String prenom){
        this.prenom = prenom;
   }
      public String getUser(){
        return user;
   }
      public void setUser (String user){
        this.user= user;
   }
      public String getMotDEpass(){
        return motDEpass;
   }
      public void setMotDEpass(String motDEpass){
        this.motDEpass = motDEpass;
   }
   public String getProfile(){
        return profile;
   }
      public void setProfile(String profile){
        this.profile=profile;
   }
```

Les objets métier :
Composant « Utilitaire » : utilisable par tout formulaire accédant à la base de données, il sert à la fermeture des différents objets communiquant avec la base de données (Connection, Statement, ResultSet)

```
package com.Utilitaires;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public final class Utilitaire {
    private Utilitaire() {
  }
  /* Fermeture silencieuse du resultset */
  public static void fermetureSilencieuse( ResultSet resultSet ) {
    if ( resultSet != null ) {
      try {
        resultSet.close();
      } catch ( SQLException e ) {
        System.out.println( "échec de la fermeture du ResultSet : " + e.getMessage() );
    }} }
  /* Fermeture silencieuse du statement */
  public static void fermetureSilencieuse( Statement statement ) {
    if ( statement != null ) {
      try {
        statement.close();
      } catch ( SQLException e ) {
        System.out.println( "échec de la fermeture du Statement : " + e.getMessage() );
      }} }
    /* Fermeture silencieuse de la connexion */
  public static void fermetureSilencieuse( Connection connexion ) {
    if ( connexion != null ) {
      try {
        connexion.close();
      } catch ( SQLException e ) {
        System.out.println( "échec de la fermeture de la connexion : " + e.getMessage() );
 }}}
      /* Fermetures silencieuses du statement et de la connexion */
  public static void fermeturesSilencieuses( Statement statement, Connection connexion ) {
    fermetureSilencieuse( statement );
    fermetureSilencieuse( connexion );
  }
  /* Fermetures silencieuses du resultset, du statement et de la connexion */
  public static void fermeturesSilencieuses( ResultSet resultSet, Statement statement, Connection
connexion ) {
    fermetureSilencieuse( resultSet );
    fermetureSilencieuse( statement );
    fermetureSilencieuse( connexion );
}   }
```

ConnexionForm.java

```
package com.Forms;

import javax.servlet.http.HttpServletRequest;

public class ConnexionForm {
        private static final String CHAMP_PROFILE     = "profile";



   public String chercherProfile( HttpServletRequest request )  {
        String profile = getValeurChamp( request, CHAMP_PROFILE );


        return profile;
   }
      private static String getValeurChamp( HttpServletRequest request, String
nomChamp ) {
      String valeur = request.getParameter( nomChamp );
      if ( valeur == null || valeur.trim().length() == 0 ) {
         return null;
      } else {
         return valeur;
      }
   }


}
```

CreationUtilForm.java

```
package com.Forms;
import static com.Utilitaires.Utilitaire;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import com.mysql.jdbc.Connection;
public final class CreationUtilForm {
   private static final String CHAMP_NOM       = "nom";
   private static final String CHAMP_PRENOM       = "prenom";
   private static final String CHAMP_USER      = "user";
   private static final String CHAMP_PASS      = "motDEpasse";
   private static final String CHAMP_CONF      = "confirmation";
   private static final String CHAMP_PROFILE      = "profile";
          String   url = "jdbc:mysql://localhost:3306/Etablissement";
          String  utilisateur = "mysql";
          String  passwd = "mysql";
          Connection connexion = null;
   PreparedStatement preparedStatement = null;
   private List<String> messages = new ArrayList<String>();
   int statut = 0;
   public int inscrireUtilisateur( HttpServletRequest request )  {
      String nom = getValeurChamp( request, CHAMP_NOM );
      String prenom = getValeurChamp( request, CHAMP_PRENOM );
      String user = getValeurChamp( request, CHAMP_USER );
      String motDEpass = getValeurChamp( request, CHAMP_PASS );
      String confirmation = getValeurChamp( request, CHAMP_CONF );
      String profile = getValeurChamp( request, CHAMP_PROFILE );
Try{
Class.forName( "com.mysql.jdbc.Driver" );
            } catch ( ClassNotFoundException e ) {
                  e.getMessage();
                                    }
        try{
       connexion = (Connection) DriverManager.getConnection( url, utilisateur, passwd );
       preparedStatement = (PreparedStatement) connexion.prepareStatement ("INSERT
INTO Utilisateur(nom,prenom,mot_de_passe,user,profile) VALUES (?,?,?,?,?)");
       preparedStatement.setString( 1, nom );
       preparedStatement.setString( 2, prenom );
       preparedStatement.setString( 3, motDEpass );
       preparedStatement.setString( 4, user  );
       preparedStatement.setString( 5, profile );
        statut = preparedStatement.executeUpdate();
```

```
} catch ( SQLException e ) {

                e.getMessage() ;
            }finally {
           fermeturesSilencieuses( preparedStatement, connexion );
              }


 return statut;


   }
private static String getValeurChamp( HttpServletRequest request, String
nomChamp ) {
      String valeur = request.getParameter( nomChamp );
      if ( valeur == null || valeur.trim().length() == 0 ) {
         return null;
      } else {
         return valeur;
      }
   }
}
```

EtabForm.java

```
package com.Forms;
import static com.Utilitaires.Utilitaire;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.servlet.http.HttpServletRequest;
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.PreparedStatement;

public class EtabForms {
        private static final String CHAMP_CODE        = "codeEtab";
          private static final String CHAMP_NOM       = "nomEtab";
          private static final String CHAMP_ADRESSE   = "adresseEtab";

          String url = "jdbc:mysql://localhost:3306/Etablissement";
          String utilisateur = "mysql";
          String motDePasse = "mysql";
          Connection connexion = null;
          PreparedStatement preparedStatement = null;
          int resultat = 0;

       public int ajouterEtab (HttpServletRequest request){
                String code = getValeurChamp( request, CHAMP_CODE );
    String nom = getValeurChamp( request, CHAMP_NOM );
    String adresse = getValeurChamp( request, CHAMP_ADRESSE );

                 /* Chargement du driver JDBC pour MySQL */
          try {
           Class.forName( "com.mysql.jdbc.Driver" );
          } catch ( ClassNotFoundException e ) {
                  e.getMessage();
                              }
                   try {
  connexion = (Connection) DriverManager.getConnection( url, utilisateur, motDePasse );
preparedStatement = (PreparedStatement) connexion.prepareStatement ("INSERT INTO
etablissement(codeEtab,nomEtab,adresseEtab) VALUES (?,?,?)");
              preparedStatement.setString( 1, code );
              preparedStatement.setString( 2, nom );
              preparedStatement.setString( 3, adresse );
              resultat = preparedStatement.executeUpdate();
          } catch ( SQLException e ) {
              e.getMessage() ;
          }finally {
        fermeturesSilencieuses( preparedStatement, connexion );
          }
                return resultat;
  }
       private static String getValeurChamp( HttpServletRequest request, String nomChamp ) {
    String valeur = request.getParameter( nomChamp );
    if ( valeur == null || valeur.trim().length() == 0 ) {
      return null;
    } else {
      return valeur;
    }
       }}
```

BilanForm.java

```
package com.Forms;
import static com.Utilitaires.Utilitaire;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.PreparedStatement;

public class BilanForm  {
            private List<String> messages = new ArrayList<String>();
              String url = "jdbc:mysql://localhost:3306/Etablissement";
            String utilisateur = "mysql";
            String motDePasse = "mysql";
            Connection connexion = null;
            PreparedStatement preparedStatement = null;
            ResultSet resultat = null;
          public List<String> afficherBilan ( HttpServletRequest request ){
                    try {
                      Class.forName( "com.mysql.jdbc.Driver" );
                      } catch ( ClassNotFoundException e ) {
                              e.getMessage();
                                                  }
                      try {
 connexion = (Connection) DriverManager.getConnection( url, utilisateur, motDePasse );
                          messages.add( "Connexion réussie !" );
                          preparedStatement = (PreparedStatement) connexion.prepareStatement
("SELECT codeEtab,nomEtab,adresseEtab FROM etablissement;");  messages.add( "Objet
requête créé !" );
 resultat = preparedStatement.executeQuery(); messages.add( "Requête : SELECT
codeEtab,nomEtab,adresseEtab FROM Utilisateur;" );
 while ( resultat.next() ) {
 String codeEtablissement = resultat.getString( "codeEtab" );
 String nomEtablissement = resultat.getString( "nomEtab" );
 String adresseEtablissement = resultat.getString( "adresseEtab" );
 messages.add( "Données retournées par la requête : codeEtablissement = " + codeEtablissement +
", nomEtablissement = " + nomEtablissement +", adresseEtablissement = " +
adresseEtablissement);
                          }
        } catch ( SQLException e ) {
            messages.add( "Erreur lors de la connexion : <br/>"
                                      + e.getMessage() );
            }finally {
                    fermeturesSilencieuses( resultat,preparedStatement, connexion );
                      }

             return messages;
                      }
}
```

1.b.Les différentes pages JSP (la vue):

connexion.jsp

```
<%@ page pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8" />
      <title>Inscription</title>
      <link type="text/css" rel="stylesheet" href="inc/style.css" />
   </head>
   <body>
      <form method="post" action="coonnexion">
         <fieldset>
            <legend>Connexion</legend>

            <p>Vous pouvez  vous connecter via ce formulaire.</p>


            <label for="user">User<span class="requis">*</span></label>
            <input type="text" id="user" name="user" value=" " size="20" maxlength="20" />

            <br />

            <label for="motdepasse">Mot de passe <span class="requis">*</span></label>
            <input type="password" id="motDEpasse" name="motDEpasse" value="" size="20"
maxlength="20" />

            <br />

             <label for="profile">Profile Utilisateur <span class="requis">*</span></label>
            <input type="text" id="profile" name="profile" value="" size="20" maxlength="20" />

            <br />


            <input type="submit" value="connection" class="sansLabel" />
            <br />


         </fieldset>
      </form>
            <c:forEach items="${ messages }" var="message" varStatus="boucle">
         <p>${ boucle.count }. ${ message }</p>
      </c:forEach>
     </body>
```

Inscription.jsp

```
<%@ page pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8" />
      <title>Inscription</title>
      <link type="text/css" rel="stylesheet" href="inc/style.css" />
   </head>
   <body>
      <form method="post" action="creationUtil">
         <fieldset>
            <legend>Inscription</legend>

            <p>Vous pouvez  inscrire un utilisateur via ce formulaire.</p>


           <label for="nom">Nom Utilisateur</label>
            <input type="text" id="nom" name="nom" value=" " size="20" maxlength="20" />
             <br />

           <label for="prenom">Prenom Utilisateur</label>
            <input type="text" id="prenom" name="prenom" value=" " size="20" maxlength="20" />
            <br />

            <label for="user">User<span class="requis">*</span></label>
            <input type="text" id="user" name="user" value=" " size="20" maxlength="20" />
             <br />

            <label for="motdepasse">Mot de passe <span class="requis">*</span></label>
            <input type="password" id="motDEpasse" name="motDEpasse" value="" size="20"
maxlength="20" />
            <br />
            <label for="confirmation">Confirmation du mot de passe <span
class="requis">*</span></label>
            <input type="password" id="confirmation" name="confirmation" value="" size="20"
maxlength="20" />
           <br />
          <label for="profile">Profile Utilisateur <span class="requis">*</span></label>
            <input type="text" id="profile" name="profile" value="" size="20" maxlength="20" />
           <br />
                        <input type="submit" value="creation" class="sansLabel" />
            <br />
          </fieldset>
      </form>
   </body>
</html>
```

Etablissement.jsp

```
<%@ page pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8" />
      <title>Inscription</title>
      <link type="text/css" rel="stylesheet" href="inc/style.css" />
   </head>
   <body>
      <form method="post" action="etablissement">
         <fieldset>
            <legend>etablissement</legend>

            <p>Vous pouvez ajouter un nouvel etablissement via ce formulaire.</p>


            <label for="code"> Code Etablissement<span class="requis">*</span></label>
            <input type="text" id="codeEtab" name="codeEtab" value=" " size="20" maxlength="20"
/>

            <br />


           <label for="nom">Nom Etablissement</label>
            <input type="text" id="nomEtab" name="nomEtab" value=" " size="20" maxlength="20"
/>

            <br />


           <label for="adresse">Adresse Etablissement</label>
            <input type="text" id="adresseEtab" name="adresseEtab" value=" " size="20"
maxlength="20" />

            <br />

             <input type="submit" value="Inscription" class="sansLabel" />
            <br />

         </fieldset>
      </form>
   </body>
</html>
```

Bilan.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Bilan</title>
        <link type="text/css" rel="stylesheet" href="<c:url value="/inc/form.css"/>" />
    </head>
    <body>
        <h1>Afficher le bilan</h1>

        <c:forEach items="${ messages }" var="message" varStatus="boucle">
            <p>${ boucle.count }. ${ message }</p>
        </c:forEach>
    </body>
</html>
```

1.c. les servlets (contrôle) : permettent d'afficher les pages jsp après avoir instancié les différents objets métiers.

Connexion.java

```
package com.Servlets;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import com.Beans.Utilisateur;
import com.Forms.ConnexionForm;
public class Connexion extends HttpServlet{
        public static final String VUE            = "/WEB-INF/connexion.jsp";
        public static final String VUEINSC        = "/creationUtil";
        public static final String VUEBILAN       = "/bilan";
        public static final String VUESAISIE      ="/etablissement";
        public static final String ATT_SESSION_USER = "sessionUtilisateur";

public void doGet( HttpServletRequest request, HttpServletResponse response ) throws ServletException,
IOException {
      /* Affichage de la page d'inscription */
      this.getServletContext().getRequestDispatcher( VUE ).forward( request, response );
   }
 public void doPost( HttpServletRequest request, HttpServletResponse response ) throws ServletException,
IOException {
      /* Préparation de l'objet formulaire */
         ConnexionForm1 form = new CoonnexionForm1();
         String profile = null;
         HttpSession session = request.getSession();
         session.setAttribute(ATT_SESSION_USER,profile);
         profile = form.chercherProfile(request);
    switch (profile){
     case "admin":
response.sendRedirect( request.getContextPath() + VUEINSC );
break;
case "saisie":
response.sendRedirect( request.getContextPath() + VUESAISIE    );
break;
case "consultation":
response.sendRedirect( request.getContextPath() + VUEBILAN  );
break;
default:
      this.getServletContext().getRequestDispatcher( VUE ).forward( request, response );
   }}}
```

```
package com.Servlets;
import java.io.IOException;
import java.sql.SQLException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.Forms.CreationUtilForm;


public class CreationUtil extends HttpServlet{
   public static final String ATT_FORM       = "form";
   public static final String VUE            = "/WEB-INF/inscriptionUtili.jsp";

   public void doGet( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {
       /* Affichage de la page d'inscription */
       this.getServletContext().getRequestDispatcher( VUE ).forward( request, response );
   }
   public void doPost( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {
       /* Préparation de l'objet formulaire */
  CreationUtilForm form = new CreationUtilForm();
 int statue = form.inscrireUtilisateur(request);
       /* Stockage du formulaire et du bean dans l'objet request */
       his.getServletContext().getRequestDispatcher( VUE ).forward( request, response );
   }
}
```

Etablissement.java

```
package com.Servlets;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.Forms.EtabForms;

public class Etablissement extends HttpServlet{

        public static final String VUE          = "/WEB-INF/etablissement.jsp";

        public void doGet( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {
            /* Affichage de la page etablissement */
            this.getServletContext().getRequestDispatcher( VUE ).forward( request, response );
          }

        public void doPost( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {
                EtabForms form = new EtabForms();
                int res = form.ajouterEtab(request);
this.getServletContext().getRequestDispatcher( VUE ).forward( request, response );
} }
```

Bilan.java

```
package com.Servlets;
import java.io.IOException;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.Forms.BilanForm;


public class Bilan extends HttpServlet {

        public static final String ATT_MESSAGES = "messages";
   public static final String VUE          = "/WEB-INF/bilan.jsp";

   public void doGet( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {
      /* Initialisation de l'objet Java et récupération des messages */
      BilanForm test = new BilanForm();
      List<String> messages = test.afficherBilan( request );
      /* Enregistrement de la liste des messages dans l'objet requête */
      request.setAttribute( ATT_MESSAGES, messages );
      /* Transmission vers la page en charge de l'affichage des résultats */
      this.getServletContext().getRequestDispatcher( VUE ).forward( request, response );
   }

}
```

1.d. Le fichier web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>ProjetEtablissement</display-name>

  <servlet>
    <servlet-name>Connexion</servlet-name>
    <servlet-class>com.Servlets.Connexion</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Connexion</servlet-name>
    <url-pattern>/connexion</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>Etablissement</servlet-name>
    <servlet-class>com.Servlets.Etablissement</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Etablissement</servlet-name>
    <url-pattern>/etablissement</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>CreationUtil</servlet-name>
    <servlet-class>com.Servlets.CreationUtil</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>CreationUtil</servlet-name>
    <url-pattern>/creationUtil</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>Bilan</servlet-name>
    <servlet-class>com.Servlets.Bilan</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Bilan</servlet-name>
    <url-pattern>/bilan</url-pattern>
  </servlet-mapping>

  <filter>
        <filter-name>restreidreApplication</filter-name>
        <filter-class>com.Filters.restreidreApplication</filter-class>
 </filter>
  <filter-mapping>
        <filter-name>restreidreApplication</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>

</web-app>
```

1.e. le code du filtre :

```
package com.Filters;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;


public class restreidreApplication implements Filter {
        public static final String VUE             = "/connexion";
        public static final String ATT_SESSION_USER = "sessionUtilisateur";

        public void destroy() {

        }

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws
IOException, ServletException {
    HttpServletRequest request = (HttpServletRequest) req;
      HttpServletResponse response = (HttpServletResponse) res;

      String chemin = request.getRequestURI().substring( request.getContextPath().length() );
      if ( chemin.startsWith( "/inc" ) ) {
         chain.doFilter( request, response );
         return;
      }

      HttpSession session = request.getSession();
      if ( session.getAttribute( ATT_SESSION_USER ) == null ) {
      request.getRequestDispatcher( VUE ).forward( request, response );
      } else {
                chain.doFilter(request, response);
      }}
public void init(FilterConfig fConfig) throws ServletException {
        }}
```

2. Serveur de base de données :

2. a. Présentation du SGBD MySQL :

Le serveur de base de données utilisé est le SGBD MySQL.

MySQL est un Système de Gestion de Base de Données Relationnel. Aussi est un véritable serveur de base de données SQL multi−utilisateur et multi−thread.

MySQL fonctionne sous:

· Linux
· Windows 95/98/NT/2000/XP/2003 Server
· Solaris
· Mac Os X Serveur
· Les pilotes ODBC, OleDB, JDBC...
· Des API pour les langages C et C++.
· Une interface d'administration graphique: MySQL Control Center

2.b. Connexion de l'application à l' SGBD MySQL

La connexion de l'application au serveur de base de données est effectuée à travers le middleware d'accès aux données JDBC.

Comme il existe plusieurs SGBD différents et bien qu'ils se basent tous sur le langage SQL, chacun a sa manière de gérer les données.

L'avantage de JDBC, c'est qu'il est nativement prévu pour pouvoir s'adapter à n'importe quel SGBD. Ainsi pour faire en sorte que notre application puisse dialoguer avec MySQL, nous aurons simplement besoin d'ajouter à notre projet un driver (com.mysql.jdbc.driver) qui est spécifique à MySQL.

Il suffit ensuite de copier ce driver dans le répertoire /lib présent dans le dossier d'installation du serveur Tomcat, et tous les projets déployés sur ce serveur pourront alors communiquer avec une base de données MySQL.

Conclusion

Dans le cadre de ce travail, on a réalisé deux types d'applications tout en suivant deux approches de conception des systèmes d'information et deux environnements de développement.

La première application a pour objectif d'automatiser les procédures de travail, dans le but d'exploiter les informations saisies au lieu d'effectuer un travail répétitif et onéreux. Elle a été réalisée en suivant l'approche classique.

La deuxième application a un objectif pédagogique consiste à réaliser des fonctionnalités tout en utilisant l'approche objet dans la conception et la programmation et en respectant les règles du patron de conception MVC, ainsi que l'utilisation de l'environnement Eclipse sans Framework. Ce travail m'a permet de pratiquer l'approche orienté objet et développer des classes réutilisables, d'explorer les différentes technologies de l'environnement JEE, à savoir les servlets, les JSP, la JSTL.

Un résumé d'un cours javaEE est réalisé, je souhaite qu'il soit utile pour les étudiants débutants en javaEE avec Eclipse.

Perspectives :

L'application javaEE est réalisée dans le cadre du patron de conception MVC, cependant il n'est pas le seul patron de conception qui pourrait être utilisé dans l'environnement javaEE, on peut utiliser d'autres pattern comme le modèle DAO (Data Access Object) qui permet de mieux organiser le code présent dans la partie Modèle du MVC.
Le pattern DAO permet la souplesse du changement du système de stockage de données (le passage de MySQL vers Oracle par exemple).

BIBLIOGRAPHIE

[1]: **Site web:** www.developpez.com
[2]: **Site web:** www.wikepedia.org
[3] : **Livre :** UML 2 analyse et conception
Josef Gabay&David Gabay
Editions *DUNOD*
[4] : **Plate forme européenne du e-learning :**www.openclassrooms.com
[5] : **Livre :** introduction aux systèmes d'information
M.c BELAID & D.BOYAKOUB
Editions pages bleu internationales
[6] : **Site web :** www.memoireonline.com › Informatique et Télécommunications.
[7]: **Site Web: www**.entreprisetic.com/Menu-Gauche/.../**Architecture-Client-Serveur**
[8]: Mme Dalila BOUYACOUB Ep.TAOURI, « Répartitions, organisation et distribution des données dans un environnement mobile », **Thèse de magister en informatique**, UMMTO 2007

Glossaire des mots techniques

Partie II :

Chapitre I : Architecture client/serveur

Architecture distribuée : L'architecture d'un environnement informatique ou d'un réseau est dite **distribuée** quand toutes les ressources ne se trouvent pas au même endroit ou sur la même machine. On parle également d'informatique distribuée.

Les architectures distribuées reposent sur la possibilité d'utiliser des objets qui s'exécutent sur des machines réparties sur le réseau et communiquent par messages au travers du réseau

Interopérabilité : L'interopérabilité ou interfonctionnement en informatique est la capacité que possède un système informatique à fonctionner avec d'autres produits ou systèmes informatiques, existants ou futurs, sans restriction d'accès ou de mise en œuvre.

Serveur d'objets : Architecture client-serveur d'un SGBDO où l'unité de transfert entre le serveur et une station de travail cliente est l'objet, ou le groupe d'objets.

Courtier d'objet : courtier de requêtes d'objets (*Object Request Broker (ORB)) :* Un ORB est un ensemble de fonctions (classes Java, bibliothèques C++, …) qui implémentent un « bus logiciel » par lequel des objets envoient et reçoivent des requêtes et des réponses, de manière transparente et portable ; il s'agit de l'activation ou de l'invocation à distance par un objet, d'une méthode d'un autre objet distribué. En pratique les objets invoqués sont souvent des services.

Les ORB appartiennent à la famille des middleware.

Deux ORB peuvent communiquer entre eux au travers du protocole IIOP (*Internet Inter-ORB Protocol*, voir également *General Inter-ORB Protocol).*

Groupeware : Un groupware ou logiciel de groupe, est un type de logiciel qui permet à un groupe de personnes de partager des documents à distance pour favoriser le travail collaboratif.

Chapitre II : Environnement javaEE

Objet métier : L'objet métier désigne un concept ayant une réalité pour le métier de l'entreprise.

multi−utilisateur : SGBD multi-utilisateur ; Une même donnée peut être manipulée par plusieurs utilisateurs à la fois.

multi−threaded : Un processeur est dit multithread s'il est capable d'exécuter efficacement plusieurs threads simultanément. Contrairement aux systèmes multiprocesseurs, les threads d'un même doivent partager les ressources d'un unique processeur : les unités de traitement, le cache processeur ; certaines parties sont néanmoins dupliquées : chaque *thread* dispose de ses propres registres et de son propre pointeur d'instruction.

Là où les systèmes multiprocesseurs incluent plusieurs unités de traitement complètes, le *multithreading* a pour but d' augmenter l'utilisation d'un seul processeur en tirant profit des propriétés des *threads* et du parallélisme au niveau des instructions.