Remerciements

En présentant ce travail, nous exprimons notre profonde reconnaissance à ALLAH le tout Puissant, notre Créateur.

Je tiens à exprimer ma grande gratitude à *Mr*. Badr BENAMMAR enseignant à UABT, pour avoir accepté de m'encadrer tout au long de ce travail, pour ses conseils et suggestions et pour toute l'aide morale qu'il n'a cessé de me prodiguer.

Je tiens à remercier *Mr* LAHSAINI pour l'honneur de présider mon jury de soutenance.

Je tiens également à remercier *Mm* LABRAOUI et *Mm DIDI* pour avoir bien voulu accepter d'examiner et de juger mon modeste travail.

Que tout enseignant qui m'a fait bénéficier de son savoir pendant ces cinq années passée a UABT, trouve ici l'expression de ma profonde gratitude.

À toute personne ayant contribué, de près ou de loin, à ce travail, je vous dis merci.

Dédicaces

Je dédie ce modeste travail aux personnes qui me sont les plus chers au monde, que j'aime et que j'adore, à mes chers parents.

À mes frères et sœurs. À mes meilleurs amis :

Zineb, Warda, Salah, Anes, Zaki ,ainsi que toute la promo RSD,

Et les personnes qui m'ont aidé et encouragé. Je vous dis grand merci.

Enfin, à toutes les personnes que j'estime et je respect.

Nadjib

Table de matières

Introduction général 1

Chapitre I : Systèmes distribués

I. Introduction 3

II. Définition d'un système distribué 3

III. Caractéristiques d'un système distribué 4

III.1. Transparence 4

III.2. Passage à l'échelle 5

III.3. Disponibilité 6

III.4. Autonomie 8

IV. Type de communication 9

IV.1. Langages Synchrones 9

IV.2. Langages asynchrones 10

V. Mode de communication 11

V.1. Communication bas niveau (socket) 11

V.2. Communication haut niveau (middleware) 12

V.2.1. Propriété de communication d'un middleware 13

V.2.2. Architecture des middlewares 14

VI. Quelque exemple des systèmes distribués 16

VI.1. Systèmes P2P 16

VI.2. Grilles informatiques ou grid 18

VI.3. Cloud 19

VII. Conclusion 21

Chapitre II : Architectures N-tiers

I. Introduction 22

II. Trois niveaux d'abstraction d'une application 22

II.1. Couche de présentation 22

II.2. Services métier 22
II.3. Persistance 22
III. L'architecture un tiers 23
I.1. Présentation 23
III.4. Les solutions sur site central (mainframe) 24
III.5. Les applications un tiers déployées 25
III.6. Avantages et limites 26
III.6.1. Avantage 26
III.6.2. Limite 26
IV. L'architecture 2-tiers 27
IV.1. Modèle client/serveur de données 27
IV.1.1. Concept de client/serveur 27
IV.1.2. Client/serveur de données 30
IV.2. Limites du client-serveur deux tiers : client lourd 34
V. L'architecture trois tiers 35
V.1. Objectifs 35
V.2. Premières tentatives 35
V.3. Répartition des traitements 36
V.4. Client léger 37
V.4.1. Présentation 37
V.4.2. Ergonomie 38
V.5. Exemples de clients légers 40
V.6. Service applicatif 41
V.6.1. Présentation 41
V.6.2. Gestion des transactions 41
V.7. Limitations 42
VI. Les architectures n-tiers 42

VI.1. Présentation 42
VI.2. Que de niveaux 43
VII. Rôle de l'approche objet 43
VII.2.1. Approche objet 43
VII.2.2. Objets métier 45
VIII.Conclusion 46

Chapitre III : Présentation de l'application réalisée

I. Introduction 47
II. Définition de l'e-learning 47
III. Les plateformes sélectionnées 48
III.1. Claroline 1.8.6 48
III.2. Ganesha 3.2.2 50
III.3. Moodle 1.8.2 52
III.4. Sakai 2.4.0 55
IV. Grilles d'évaluation des points-clés 57
IV.1. Les 10 points clés 57
IV. Présentation de de notre plateforme 60
V. Conclusion 66
Conclusion général 67
Références bibliographiques 68

Liste de figures

Figure I.1 : Architecture d'un Middlewares 15

Figure II.1 : Les trois niveaux d'une application informatique 23

Figure II.2 : Architecture d'une application sur un site central 24

Figure II.3 : Architecture d'une application un tiers déployées 26

Figure II.4 : Architecture Client/serveur 27

Figure II.5 : Le schéma du Gartner Group 28

Figure II.6 : Client/serveur de données 30

Figure II.7 : Dialogue entre clients et serveurs 31

Figure II.8 : Schéma de fonctionnement du Client/serveur..... 32

Figure II.9 : Les niveaux de l'architecture 3-tiers 36

Figure II.10 : Qulque évolution de l'informatique 42

Figure III.1 : Formulaire d'identification 60

Figure III.2 : Formulaire d'inscription 61

Figure III.3 : Remplir le formulaire d'inscription 61

Figure III.4 : Cours1 master 2 62

Figure III.5 : Cours4 master 1 62

Figure III.6 : QCM Master1 63

Figure III.7 : Réponse de l'étudiant 63

Figure III.8: Corriger du QCM..... 64

Figure III.9 : Profil de l'étudiant 65

Figure III.10 : Note obtenue aux QCM 65

Liste des abréviations

AADL	Architecture Analysis and Design Language
ADL	Architecture Design Language
API	Application Programming Interface
ASP	Active Server Pages
CCM	Corba Component Model
CNES	Centre National d'Etudes Spatiales
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DRE	Distributed Real Time Embedded
FAP	Format And Protocol
FIFO	First In First Out
GALS	Globally Asynchronous Locally Synchronous
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
IDL	Interface Description Language
IHM	Interaction Homme-Machine
IPC	Inter-Process Communication
ISO	Institut Supérieur d'Optique
JDBC	Java DataBase Connectivity
JAR	Java ARchive
KPN	Kahn Process Network
LTTA	Loosely Time-Triggered Architecture
MIR	Mid InfraRed
NC	Network Computer
NFS	Network File System
NIR	Near InfraRed
ODBC	Open Database Connectivity
PDA	*personal digital assistant*
PPS	Pulse Per Second
PTIDES	Programmong Temporal Integrated Distributed Embedded Systems
QoS	Quality of Service

SDF	Synchronous Data Flow
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SDF	Synchronous Data Flow
SGBD	système de gestion de base de données
UML	Unified Modeling Language
WAN	Wide Area Network

Introduction général

Aujourd'hui, la matière grise représente la principale richesse d'un pays. C'est aussi l'atout compétitif majeur d'une entreprise, ainsi que pour des parents où la formation de leurs enfants constitue aujourd'hui la meilleure dot : « You earn what you learn ». La formation devient donc un enjeu essentiel.

Or, chaque jour, les technologies progressent, les métiers évoluent, l'organisation change, les méthodes de management se transforment. Les besoins augmentent tant pour la formation initiale que continue. Mais les budgets disponibles et surtout le temps qu'il est possible de dégager ne sont pas extensibles à l'infini. C'est la raison pour laquelle les outils construits sur internet, qui offrent d'immenses atouts (outre l'économie considérable en temps et en déplacement) émergents à très grande vitesse dans les pays qui ont pris de l'avance (notamment en Amérique du Nord).

Avec l'avènement des NTIC, nous devons, dès à présent, « penser apprentissage rapide et efficace », avec un minimum de problèmes d'organisation, de logistique et surtout de perte de temps. L'e-learning est la solution. C'est le nom donné actuellement à une phase importante de l'introduction des NTIC dans la formation. Il s'agit d'une évolution rapide des technologies pour l'apprentissage, rendue possible par le développement planétaire de l'Internet.

L'objectif de notre projet est d'implémenter un système distribué en utilisant une architecture N-tiers. Nous avons fait le choix de réaliser une plateforme e-learning vu tous ce qui été cité précédemment comme motivation, donc notre système distribué représente la plateforme e-learning réaliser dans le cadre de ce PFE.

Pour ce faire, nous avons structuré notre document en trois parties :

- La première partie, présente une synthèse sur les systèmes distribués, les différents modes de communication ainsi quelque exemple de systèmes distribués.
- La seconde partie, s'intérésse aux différentes architectures n-tiers en présentant un exemple pour chaque architecture.

- La troisième partie présente notre contribution dans le cadre de ce PFE, à savoir les choix du système distribué implémenté qui est la plateforme e-learning. Nous avons fait une comparaison entre les quatre plateformes les plus connues, cette comparaison a été la brique de base pour l'implémentation de notre plateforme.

Chapitre I :

Systèmes distribués

I. Introduction :

La gestion des données dans un environnement à grande échelle est indispensable pour prendre en compte les besoins des nouvelles applications telles que les plateformes e-learning. Si la gestion des données dans les systèmes distribués a été largement étudiée dans les dernières années, des solutions efficaces et à bas coût tardent à voir le jour à cause de la complexité des problèmes introduits par la largeur de l'échelle et le caractère hétérogène et dynamique de l'environnement.

Plus particulièrement, l'étude des plateformes e-learning a permis de comprendre leurs exigences à satisfaire. Ces exigences qui peuvent se résumer en un grand débit transactionnel, une haute disponibilité et une latence faible nécessitent de nouvelles approches de conception et de gestion des systèmes distribués. Ceci se décline en deux problèmes qui sont, i) une bonne conception et implémentation des architectures réparties qui hébergent les services. et ii) des mécanismes efficaces de gestion des données utilisées par ces applications.

Dans ce chapitre, nous décrivons l'architecture et le fonctionnement des systèmes distribués, en présentant les caractéristiques, les avantages et le mode de communication.

II. Définition d'un système distribué :

Un système distribué (ou réparti) est composé d'éléments reliés par un système de communication. Les éléments possèdent des fonctions de traitement (processeurs), de stockage (mémoire) et de relation avec le monde extérieur (capteurs, actionneurs, communication). Le système de communication met en œuvre une méthode de communication (par exemple, par envoi de messages). Les différents éléments du système ne fonctionnent pas indépendamment, mais collaborent à une ou plusieurs tâches communes. En conséquence, une partie au moins de l'état global du système est distribuée entre plusieurs éléments.

Dans un système réparti sans aucune contrainte, les communications et les traitements sont asynchrones. Il n'y a pas de borne supérieure de temps de transition

d'un message et pas de bornes sur le rapport relatif des vitesses d'exécution sur deux sites. Si ces temps sont précisés, alors le système est dit réparti synchrone.

Dans un système réparti primitif, des messages indépendants unidirectionnels sont envoyés sur un réseau, fiable ou non fiable. Suivant les cas, on pourra considérer les communications comme FIFO (First In First Out).

L'exécution d'un processus est une suite d'événements (local, émission, réception) appelée histoire (ou trace) du processus. Cette suite est ordonnée par une horloge locale au processus. Un système réparti ne dispose généralement pas d'horloge globale. Cependant, il est utile de pouvoir définir un ordre entre évènements appartenant à plusieurs processus. Pour cela, il est possible par exemple de définir une relation globale de précédence, l'une des plus répandues étant la relation de causalité définie par Lamport [11]. Il existe divers mécanismes de codage de ces relations d'ordre (horloges logiques de Lamport, horloges vectorielles et matricielles...) [1].

III. Caractéristiques d'un système distribué :

Un système réparti doit assurer plusieurs propriétés pour être considéré comme performant.

Nous ne citerons dans cette section que celles que nous trouvons les plus connexes à notre contexte d'études : la transparence, le passage à l'échelle, la disponibilité et l'autonomie.

III.1. Transparence :

La transparence permet de cacher aux utilisateurs les détails techniques et organisationnels d'un système distribué ou complexe. L'objectif est de pouvoir faire bénéficier aux applications une multitude de services sans avoir besoin de connaître exactement la localisation ou les détails techniques des ressources qui les fournissent. Ceci rend plus simple, le développement des applications, mais aussi leur maintenance évolutive ou corrective. Selon la norme (ISO, 1995) la transparence a plusieurs niveaux:

- **Accès** : cacher l'organisation logique des ressources et les moyens d'accès à une ressource.

- **Localisation** : l'emplacement d'une ressource du système n'a pas à être connu.
- **Migration** : une ressource peut changer d'emplacement sans que cela ne soit aperçu.
- **Relocalisation** : cacher le fait qu'une ressource peut changer d'emplacement au moment où elle est utilisée.
- **Réplication** : les ressources sont dupliquées, mais les utilisateurs n'ont aucune connaissance de cela.
- **Panne** : si un nœud est en panne, l'utilisateur ne doit pas s'en rendre compte et encore moins de sa reprise après panne.
- **Concurrence** : rendre invisible le fait qu'une ressource peut être partagée ou sollicitée simultanément par plusieurs utilisateurs.

Le parcours de cette liste montre qu'il n'est pas évident d'assurer une transparence totale. En effet, masquer complètement les pannes des ressources est quasi impossible aussi bien d'un point de vue théorique que pratique. Ceci est d'autant plus vrai qu'il n'est pas trivial de dissocier une machine lente ou surchargée de celle qui est en panne ou dans un sous-réseau inaccessible [2].

III.2. Passage à l'échelle

Le concept de passage à l'échelle désigne la capacité d'un système à continuer à délivrer avec un temps de réponse constant un service même si le nombre de clients ou de données augmente de manière importante. Le passage à l'échelle peut être mesuré avec au moins trois dimensions :

- Le nombre d'utilisateurs et/ou de processus (passage à l'échelle en taille).
- La distance maximale physique qui sépare les nœuds ou ressources du système (passage à l'échelle géographique).
- Le nombre de domaines administratifs (passage à l'échelle administrative). Le dernier type de passage à l'échelle est d'une importance capitale dans les grilles informatiques, car il influe sur le degré d'hétérogénéité et donc sur la complexité du système.

Pour assurer le passage à l'échelle, une solution coûteuse consiste à ajouter de nouveaux serveurs puissants (plusieurs dizaines de CPU) pour garder le même niveau de performance en présence de fortes charges. Ce type de passage à l'échelle est plus connu sous le nom de Scale Up. Le problème avec cette solution est que si le système est sous-chargé les ressources consomment de l'énergie inutilement et ne servent à rien. D'autres solutions moins chères consistent à utiliser plusieurs machines moins puissantes (d'un à deux CPU par machine) pour faire face aux pics des charges, on parle alors de Scale Out. Trois techniques peuvent être utilisées pour favoriser le passage à l'échelle à faible coût. La première technique consiste à ne pas attendre la fin d'une tâche pour commencer une autre si elles sont indépendantes. Cela permet de cacher la latence du réseau ou la lenteur d'une ressource. Ce modèle de fonctionnement est appelé modèle asynchrone. La deuxième technique consiste à partitionner les données et les stocker sur plusieurs serveurs. Ceci permet de distribuer la charge applicative et de réduire le temps de traitement global des tâches. Il est aussi possible d'effectuer certains traitements au niveau des clients (Java Applets). La troisième technique est l'utilisation de la réplication et/ou des mécanismes de cache. L'accès à des informations stockées dans un cache réduit les accès distants et donne de bonnes performances aux applications de type web. La réplication, quant à elle, permet une distribution des traitements sur plusieurs sites permettant ainsi une amélioration du débit du traitement.

Cependant, l'utilisation de ces techniques présente quelques inconvénients. En effet, le partitionnement et la distribution d'un traitement nécessitent des mécanismes de contrôle plus complexes pour intégrer les résultats et assurer leur cohérence. En plus, garder plusieurs copies d'une même donnée (caches ou répliques) peut entraîner des incohérences à chaque fois que l'on met une copie à jour. Pour éviter ce problème, il faut penser à faire des synchronisations, ce qui est souvent contradictoire avec la première technique de passage à l'échelle à savoir faire de l'asynchronisme.

En conclusion, les besoins de passage à l'échelle et de cohérence sont en opposition, d'où la nécessité de trouver des compromis en fonction des besoins des applications cible [2].

III.3. Disponibilité :

Un système est dit disponible s'il est en mesure de délivrer correctement le ou les services de manière conforme à sa spécification. Pour rendre un système disponible,

il faut donc le rendre capable de faire face à tout obstacle qui peut compromettre son bon fonctionnement. En effet, l'indisponibilité d'un système peut être causée par plusieurs sources parmi lesquelles nous pouvons citer :

- Les pannes qui sont des conditions ou évènements accidentels empêchant le système, ou un de ses composants, de fonctionner de manière conforme à sa spécification.
- Les surcharges qui sont des sollicitations excessives d'une ressource du système entraînant sa congestion et la dégradation des performances du système.

Les attaques de sécurité qui sont des tentatives délibérées pour perturber le fonctionnement du système, engendrant des pertes de données et de cohérences ou l'arrêt du système. Pour faire face aux pannes, deux solutions sont généralement utilisées. La première consiste à détecter la panne et à la résoudre, et ce dans un délai très court. La détection des pannes nécessite des mécanismes de surveillance qui s'appuient en général sur des timeouts ou des envois de messages périodiques entre ressources surveillées et ressources surveillantes. Cette détection, outre la surcharge qu'elle induit sur le système, ne donne pas toujours des résultats fiables. En effet, avec l'utilisation des timeouts, à chaque fois qu'un timeout est expiré, la ressource sollicitée, ne pouvant être contactée ou n'arrivant pas à envoyer une réponse, est en général suspectée d'être en panne. Par conséquent, une simple variation de la latence du réseau ou de la surcharge d'une ressource peut entraîner l'envoi et/ou la réception d'une réponse en dehors du timeout et donc conduire à des fausses suspicions. Par ailleurs, une fois la panne détectée, il faut des mécanismes de résolution efficace pour arriver à la cacher aux clients. Cette tâche est loin d'être triviale, car il existe plusieurs types de pannes et chacune d'entre elles requiert un traitement spécifique.

La deuxième solution consiste à masquer les pannes en introduisant de la réplication. Ainsi, quand une ressource est en panne, le traitement qu'elle effectuait est déplacé sur une autre ressource disponible. La réplication peut être aussi utilisée pour faire face à la seconde cause d'indisponibilité d'un système (surcharge du système). Pour réduire la surcharge d'une ressource, les tâches sont traitées parallèlement sur plusieurs répliques ou sur les différentes répliques disponibles à tour de rôle (tourniquet). Une autre technique qui permet de réduire la surcharge d'une ressource consiste à distribuer les services (ou les données) sur plusieurs sites et donc de les

solliciter de manière parallèle. Le partitionnement des services ou des données permet d'isoler les pannes et donc de les contrôler plus simplement [2].

Enfin, la gestion de la dernière source d'indisponibilité nécessite des politiques de sécurité sur l'accès et l'utilisation des ressources. Ces politiques ont pour objet la mise en œuvre de mécanismes permettant de garantir les deux propriétés suivantes : la confidentialité et l'intégrité des ressources ou informations sensibles. La confidentialité permet de protéger les accès en lecture aux informations, alors que l'intégrité permet de protéger les accès en écriture. S'il existe une seule politique de sécurité pour l'ensemble des ressources, la sécurité est dite centralisée, dans le cas contraire, elle est dite distribuée et permet à chaque partie du système d'avoir sa propre politique.

Il est à noter qu'une politique de sécurité distribuée permet plus d'hétérogénéité et s'adapte à des systèmes comme les grilles informatiques, mais nécessite des mécanismes plus complexes.

Nous venons de voir que quelle que soit la manière dont la gestion de la disponibilité est assurée, des modules supplémentaires sont requis (gestion de réplication, détection et résolution des pannes, politique de sécurité, etc.). Ceci entraîne d'une part, une surcharge du système (nombre de messages très important, collection de processus en arrière-plan) et d'autre part, une complexité du système. Une solution idéale serait de minimiser la complexité, mais aussi la surcharge introduite pour assurer la disponibilité. Ainsi, les modules supplémentaires intégrés doivent être très légers (requièrent peu de ressources pour leur fonctionnement) et les techniques de détection/résolution des pannes ne doivent pas être réalisées sur la base d'une communication qui génère plusieurs messages (ex : broadcast).

III.4. Autonomie

Un système ou un composant est dit autonome si son fonctionnement ou son intégration dans un système existant ne nécessite aucune modification des composants du système hôte. L'autonomie des composants d'un système favorise l'adaptabilité, l'extensibilité et la réutilisation des ressources de ce système. Par exemple, une ressource autonome peut être remplacée avec une autre ressource plus riche en termes de fonctionnalités, ce qui étend les services du système. Le changement du pilote d'accès à une base de données ODBC par un pilote JDBC illustre bien cette notion d'autonomie et son impact dans le fonctionnement d'un système, puisqu'aucune modification ne sera effectuée au niveau du SGBD.

L'une des motivations de maintenir l'autonomie est que les applications existantes sont difficiles à remplacer, car d'une part, elles sont le fruit d'une expertise développée pendant plusieurs années et d'autre part, leur code n'est pas souvent accessible, i.e. les SGBD tels que Oracle, SQL Server, Sybase, etc. Pourtant, il est indispensable de s'appuyer sur les applications existantes, car les clients ont leurs préférences et ne sont pas nécessairement prêts à confier leur traitement à une nouvelle application ou de stocker leurs données sur un SGBD nouveau qui ne présente pas les mêmes garanties qu'un système connu, fiable et maintenu. Une solution pour garder l'autonomie d'une application ou d'un SGBD est d'intégrer toute nouvelle fonctionnalité supplémentaire et spécifique à une application sous forme d'intergiciel. Cependant, l'implémentation de l'intergiciel introduit de nouvelles surcharges en ajoutant une couche de plus à l'accès aux données. À cela, s'ajoute que l'intergiciel devient indispensable au fonctionnement du système et peut devenir très rapidement source de congestion s'il est partagé par plusieurs applications. Pour minimiser ces impacts négatifs, l'intergiciel doit être conçu de telles sortes qu'il soit toujours disponible et non surchargé. La complexité de son implémentation doit être contrôlée afin de minimiser le coût de sa traversée. Pour ce faire, il faut éviter de concevoir un intergiciel très générique à destination de plusieurs applications au profit d'un intergiciel ad-hoc [2].

IV. Type de communication :

IV.1. Langages Synchrones :

Dans les langages synchrones, le temps est modélisé comme une suite d'instants logiques. L'hypothèse synchrone consiste à considérer que les actions du système temps réel sont produites dans le même instant que les événements qui les ont déclenchées. Cette hypothèse permet de définir un temps logique où il est fait abstraction des temps d'exécution, ce qui rend la spécification du logiciel indépendante de l'architecture matérielle. Ainsi, dans les langages synchrones, on ne s'intéresse qu'à l'ordre des événements d'entrée et de sortie. La sémantique de ces langages est basée sur des modèles mathématiques rigoureux. La compilation des programmes produit un automate sur lequel on peut effectuer des vérifications, par exemple de vivacité

(l'automate ne comporte pas d'état dans lequel il ne pourrait sortir), d'atteignabilité (un état peut être ou ne pas être atteint à partir d'un autre état), etc., reflétant différentes propriétés du système (il n'existe pas d'interblocage, un événement peut ou ne peut se produire, etc.). L'automate ainsi vérifié peut être traduit automatiquement en code exécutable séquentiel (C, assembleur, Fortran . . .).

Le modèle synchrone, surprenant au premier abord, est une transposition à l'informatique du modèle utilisé par l'électronicien lors de la conception des circuits numériques : l'établissement des potentiels électriques dans les circuits est considéré comme non observable à l'échelle de l'horloge du système, le basculement de toutes les portes logiques est considéré comme instantané aux instants définis par l'horloge.

Ce modèle conduit principalement à deux notions :

- La *simultanéité* exprimant le fait que deux événements sont présents en même temps.
- L'*atomicité* des réactions exprimant le fait que les actions sont produites dans le même instant que l'apparition des événements qui les ont déclenchées.

Pour que l'implantation d'une spécification à partir d'un langage synchrone soit valide, l'hypothèse synchrone doit être vérifiée par cette implantation. Pour que les hypothèses de simultanéité et d'atomicité soient réalistes, il est théoriquement nécessaire que la machine qui calcule les réactions soit infiniment rapide. En pratique, on peut simplement considérer un intervalle de temps maximum pendant lequel l'état du processus n'a pas le temps d'évoluer. Cet intervalle définit en quelque sorte une échelle de temps logique représentant les instants d'évolution du processus, c'est-à-dire les événements d'entrée du système temps réel. Il suffit alors que les actions du système temps réel soient produites en un temps inférieur à cet intervalle pour que, si on se place sur l'échelle de temps définissant les instants d'évolution du processus, les réactions puissent être considérées comme atomiques et donc que les actions sont simultanées avec les événements d'entrée [3].

IV.2. Langages asynchrones :

Dans les langages asynchrones le temps est considéré comme continu, il n'existe donc pas de notion de simultanéité. Il n'existe pas non plus de notions d'atomicité. On s'intéresse ici à la durée des calculs, il est donc possible qu'un événement déclenche une réaction alors que la réaction déclenchée par un événement survenu plutôt n'est pas encore achevée. Ceci conduit à un chevauchement temporel des deux réactions. Ce problème d'exécution simultanée est résolu à l'implantation par des techniques de *préemption* des calculs. La préemption peut poser des problèmes d'indéterminisme (l'occurrence d'un événement peut conduire à plusieurs exécutions différentes du même programme) et il peut être difficile de borner les temps d'exécution et donc impossible de garantir le respect des contraintes temps réel [3].

V. Mode de communication :

V.1. Communication bas niveau (socket) :

Une socket est un point de connexion (ou point d'extrémité) servant d'élément de référence dans les échanges entre processus locaux ou distants. Son but étant de faire communiquer des processus via la pile TCP/IP, la plus grande difficulté est la préparation et la création de celles-ci. Une fois cette phase achevée, le programmeur se retrouve en possession d'un descripteur (à la manière des fonctions comme ***open()*** pour les fichiers classiques et ***xxxget()*** pour les IPC). Ce descripteur est ensuite utilisé, comme descripteur de fichier, par les autres appels systèmes tels que ***read()*** et ***write()*** mis en œuvre dans les différentes phases de la communication.

- Le fait qu'une socket possède un descripteur au même titre qu'un fichier fait qu'on pourra rediriger les entrées/sorties standards sur une socket.
- Tout nouveau processus créé par un ***fork()*** hérite des descripteurs, et donc des sockets du processus père .

Un des avantages apporté par les sockets est que celles-ci peuvent être utilisées avec plusieurs protocoles de communication. Afin que les processus puissent communiquer entre eux, il faut qu'ils utilisent les mêmes conventions d'adressage. On

définit ainsi des domaines de communications qui doivent être spécifiés lors de la création de la socket [4].

Le type d'une socket définit un ensemble de propriétés des communications dans lesquelles elle est impliquée. Cette typologie indique donc la nature de la communication supportée par la socket.

Le type d'une socket est choisi lors de sa création avec la primitive ***socket()***.

Les principales propriétés recherchées sont :

- Fiabilité de la transmission.
- Préservation de l'ordre de transmission des données.
- Non-duplication des données émises.
- Communication en mode connecté (l'émission ne commence que quand la connexion est établie.
- Le chemin reste inchangé durant toute la durée de l'émission.
- Envoi de messages urgents.
- Préservation des limites des messages.

V.2. Communication haut niveau (middleware) :

Le mot middleware (intergiciel ou logiciel médiateur en français) désigne un ensemble de logiciels ou de technologies informatiques qui servent d'intermédiaire entre les applications. Il peut être défini aussi comme un outil de communication entre des clients et des serveurs. Ainsi, il fournit un moyen aux clients de trouver leurs serveurs et aux serveurs de trouver leurs clients et en général de trouver n'importe quel objet atteignable. L'intérêt que présente un middleware est multiple et peut être résumé en quatre points:

Cacher la répartition, c'est-à-dire le fait qu'une application est constituée de parties interconnectées s'exécutant à des emplacements géographiquement répartis.

- Cacher l'hétérogénéité des composants matériels, des systèmes d'exploitation et des protocoles de communication utilisés par les différentes parties d'une application.

- Fournir des interfaces uniformes, normalisées, et de haut niveau aux équipes de développement et d'intégration, pour faciliter la construction, la réutilisation, le portage et l'interopérabilité des applications.
- Fournir un ensemble de services communs réalisant des fonctions d'intérêt général, pour éviter la duplication des efforts et faciliter la coopération entre applications.

Un middleware peut être générique ou spécifique à un type d'applications. Il faut remarquer que les gains introduits par un middleware ne vont pas sans inconvénients. En effet, un inconvénient potentiel est la perte de performances liée à la traversée de couches supplémentaires de logiciel.

L'utilisation de techniques intergiciel implique par ailleurs de prévoir la formation des équipes de développement.

V.2.1. Propriété de communication d'un middleware :

L'infrastructure de communication sur laquelle se repose un middleware peut être caractérisée par plusieurs propriétés qui permettent une première classification.

Topologie statique ou dynamique. Avec un système de communication statique, les entités communicantes sont logées dans des endroits fixes et la configuration du système ne change pas. Si toutefois, la configuration doit changer, elle est programmée en avance, peu fréquente et bien intégrée dans le fonctionnement du middleware. Par contre, un système de communication dynamique donne la possibilité aux entités communicantes de changer de localisation, de se connecter et/ou se déconnecter pendant le fonctionnement de l'application (téléphones mobiles, PDA, P2P, ...).

Comportement prévisible ou imprévisible. Dans certains systèmes de communication, des bornes peuvent être établies dans l'optique de maintenir les facteurs de performances des applications (par exemple la latence). Hélas, dans la plupart des cas pratiques, ces bornes ne sont pas connues, car les facteurs de performances dépendent de la charge des composants du système, mais aussi du débit du réseau de communication. Le système de communication est dit synchrone si le temps de transmission d'un message est borné. Si par contre, cette borne ne peut être établie, le système est dit asynchrone.

Il est possible de faire une combinaison de ces différentes propriétés pour obtenir :

- Topologie statique, comportement prévisible.
- Topologie dynamique, comportement prévisible.
- Topologie statique, comportement imprévisible.
- Topologie dynamique, comportement imprévisible.

La dernière combinaison inclut les applications déployées sur les systèmes mobiles ou P2P avec lesquelles un noeud peut à tout moment joindre ou quitter le système. Cependant, le caractère imprévisible de cette classe de système impose une surcharge supplémentaire au middleware afin qu'il puisse maintenir à un niveau acceptable les performances du système.

V.2.2. Architecture des middlewares :

Les middlewares peuvent être aussi catégorisés en s'appuyant sur leur architecture. Il existe deux types d'architectures : l'une centralisée et l'autre distribuée.

Avec l'architecture centralisée, l'ensemble des modules du middleware sont stockés sur un seul site (machine). Cette approche est beaucoup plus facile à mettre en œuvre, mais elle facilite aussi la maintenance et l'exploitation cohérente du système. Cependant, si le nombre de composants pris en compte devient important, cela peut induire à une source de contention et donc réduire les performances du système.

L'approche distribuée répartit les tâches du middleware sur plusieurs sites, ce qui donne la possibilité d'accéder au middleware de manière parallèle. Ce type d'architecture est beaucoup plus tolérant aux pics de charges grâce à la répartition des demandes sur les différents composants répartis du middleware. Cependant, la maintenance et la gestion cohérente des entités du systèmes deviennent beaucoup plus fastidieuse. En fait, les composants du middleware sont obligés de travailler de manière collaborative pour éviter des incohérences.

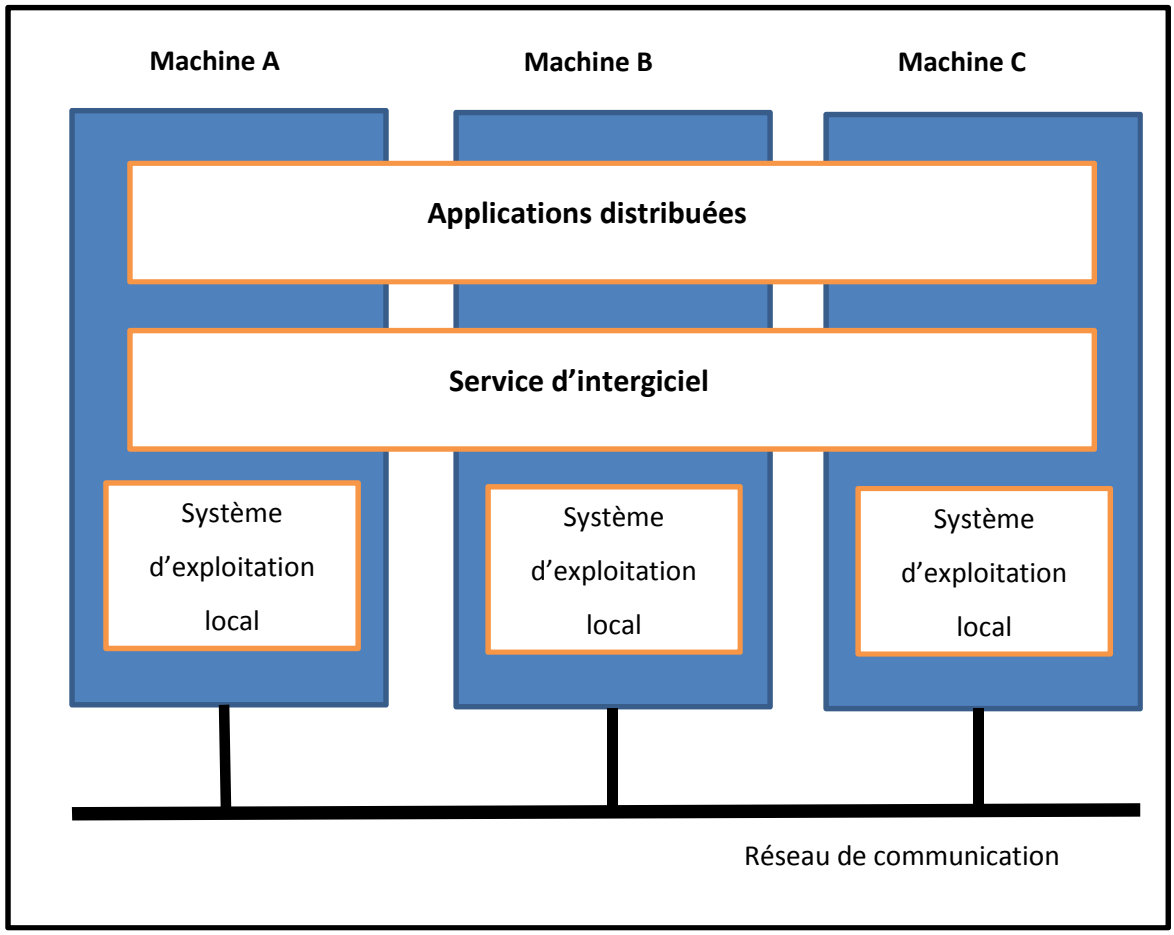

Figure I.1 : Architecture d'un Middlewares

Ce qui nécessite l'envoi de messages ou le partage de structures de données et par conséquent, des surcharges supplémentaires. Une autre approche de la distribution d'un middleware est d'avoir plusieurs instances du middleware qui s'exécute sur plusieurs sites. Cette technique, outre la tolérance aux pics de charges qu'elle offre, assure aussi une tolérance aux pannes. En fait, la duplication du middleware sur plusieurs sites permet de masquer la panne de l'une des instances du middleware ou du site qui l'héberge [2].

VI. Quelque exemple des systèmes distribués :

VI.1. Systèmes P2P :

Comme nous l'avons mentionné ci-avant, le terme P2P fait référence à une classe de systèmes distribués qui utilisent des ressources distribuées pour réaliser une tâche particulière de manière décentralisée. Les ressources sont composées d'entités de calcul (ordinateur ou PDA), de stockage de données, d'un réseau de communication, etc. La tâche à exécuter peut être du calcul distribué, du partage de données (ou de contenu), de la communication et collaboration, d'une plateforme de services, etc. La décentralisation, quant à elle, peut s'appliquer soit aux algorithmes, soit aux données, soit aux métadonnées, soit à plusieurs d'entre eux.

L'une des particularités des systèmes P2P est que tous les nœuds (pairs) sont en général symétriques, c'est à dire qu'ils jouent à la fois le rôle de client et de serveur. En particulier, les systèmes de partage de fichiers permettent de rendre les objets d'autant plus disponibles qu'ils sont populaires, en les répliquant sur un grand nombre de nœuds. Cela permet alors de diminuer la charge (en nombre de requêtes) imposée aux noeuds partageant les fichiers populaires, ce qui facilite l'augmentation du nombre de clients et donc le passage à l'échelle en taille des données.

Les pairs sont organisés autour d'une architecture qui peut être centralisée ou non. Dans l'architecture centralisée, un nœud joue le rôle de coordinateur central (serveur, ou index central) et gère soit les partages, soit la recherche, soit l'insertion d'informations et de nouveaux nœuds. Cependant l'échange d'informations entre nœuds se passe directement d'un nœud à l'autre. D'aucuns considèrent que de telles architectures ne sont pas pair-à-pair, car un serveur central intervient dans le processus. Par contre d'autres arguent que ce sont bien des systèmes pair-à-pair, car les fichiers transférés ne passent pas par le serveur central. Néanmoins, c'est une solution fragile puisque le serveur central est indispensable au réseau. Ainsi, s'il est en panne ou non accessible, tout le réseau s'effondre. En plus, le système n'est pas transparent puisque les nœuds ont besoin de savoir à tout moment l'emplacement de l'index et sont sensibles à tout changement de celui-ci. Un exemple de solution P2P centralisée est Napster (Nap), qui utilise un serveur pour stocker un index ou pour initialiser le réseau [2].

Pour faire face à ces insuffisances, une architecture distribuée s'impose, puisqu'un noeud pourrait solliciter plusieurs serveurs en même temps. Le système est ainsi plus robuste, mais la recherche d'informations est plus difficile. Elle peut s'effectuer dans des systèmes décentralisés non-structurés, comme Gnutella (Gnu). Dans ce système, la recherche se fait par propagation de la requête aux voisins jusqu'à trouver les résultats, ce qui nécessite dès lors un nombre de messages élevé, proportionnel au nombre d'utilisateurs du réseau (et exponentiel suivant la profondeur de recherche). Dans les systèmes décentralisés structurés, une organisation de connexion et de répartition des objets partagés est maintenue entre les nœuds. Souvent cette organisation est basée sur les tables de hachage distribuées, permettant de réaliser des recherches en un nombre de messages croissant de façon logarithmique avec le nombre d'utilisateurs du réseau, comme CAN, Chord, Kademlia, Pastry, etc.

Une autre solution possible est l'utilisation de « super-pairs », nœuds du réseau choisis en fonction de leur puissance de calcul et de leur bande passante, réalisant des fonctions utiles au système comme l'indexation des informations et le rôle d'intermédiaire dans les requêtes. Cette solution que l'on retrouve dans Kazaa (KaZ), rend le système un peu moins tolérant aux pannes que les systèmes complètement décentralisés et englobe un peu l'idée de client-serveur entre pairs et super-pairs.

Comme tous les nœuds sont au même niveau avec les architectures complètement distribuées, celle-ci offrent plus de transparence dans l'organisation et la localisation des ressources que les architectures centralisées ou semi-centralisées (super-pairs).

Les systèmes P2P soulèvent plusieurs problèmes bien qu'ils facilitent le passage à l'échelle et la disponibilité des ressources avec un faible coût. Il faut noter en premier lieu que les nœuds du système sont totalement autonomes et donc qu'ils peuvent choisir de partager ou non leur CPU et leur capacité de stockage. Cette autonomie leur confère aussi le choix de rejoindre ou quitter le système à tout moment. Cela a pour effet de compromettre la capacité de calcul totale et réelle du système, mais aussi la disponibilité des ressources (informations, capacité de stockage, etc.). En second lieu, le système de communication utilisé est de faible bande passante (en général Internet), ce qui peut créer une surcharge du système et une latence plus élevée que dans les clusters. Enfin, l'absence d'infrastructures de

contrôle sur les systèmes P2P rend ces derniers moins pratiques pour prendre en compte certains types d'applications qui exigent une grande qualité ou des services transactionnels. Néanmoins, les systèmes P2P sont devenus incontournables aujourd'hui dans le domaine du partage de fichiers, de la recherche d'informations et de la collaboration [2].

VI.2. Grilles informatiques ou grid :

Le terme anglais grid désigne un système distribué d'électricité. Initialement, le concept de grille partait du principe d'un tel système : les ressources d'un ordinateur (processeur, mémoire, espace disque) étaient mises à la disposition d'un utilisateur aussi facilement que l'on branche un appareil électrique à une prise électrique. Une grille informatique est une infrastructure virtuelle constituée d'un ensemble de ressources informatiques potentiellement partagées, distribuées, hétérogènes, délocalisées et autonomes. Une grille est en effet une infrastructure, c'est-à-dire des équipements techniques d'ordre matériel et logiciel. Cette infrastructure est qualifiée de virtuelle, car les relations entre les entités qui la composent n'existent pas sur le plan matériel, mais d'un point de vue logique. D'un point de vue architectural, la grille peut être définie comme un système distribué constitué de l'agrégation de ressources réparties sur plusieurs sites et mises à disposition par plusieurs organisations différentes. Un site est un lieu géographique regroupant plusieurs ressources informatiques administrées de manière autonome et uniforme. Il peut être composé d'un supercalculateur ou d'une grappe de machines (cluster).

Contrairement aux systèmes P2P, une grille garantit des qualités de service non triviales, c'est-à-dire qu'elle répond adéquatement à des exigences (accessibilité, disponibilité, fiabilité, ...) compte tenu de la puissance de calcul ou de stockage qu'elle peut fournir.

Il existe plusieurs projets de grilles qui on été mis en place aussi bien à des échelles nationales qu'internationales : la grille expérimentale française Grid'5000, la grille de calcul scientifique nord-américain TeraGrid, la grille chinoise CNGrid, la grille Asie-Pacifique ApGrid , etc.

Les grilles informatiques sont caractérisées par une forte hétérogénéité et une grande dynamicité.

En effet, les machines d'une grappe sont reliés par un réseau gigabits alors que les sites sont liés par un réseau WAN, dont la latence peut aller jusqu'à 100 millisecondes. De là, nous pouvons noter une différence importante de la latence entre deux machines d'un même site et celle entre deux machines de deux sites. En outre, chaque site est administré de manière autonome et par conséquent les politiques de sécurité varient d'un site à l'autre. Un autre exemple d'hétérogénéité relève de la composition interne d'un site. Chaque site est libre de choisir le type de processeur de ses machines (Intel, AMD, IBM, ...), la capacité de stockage et le réseau d'interconnexion entre machines (Gigabit Ethernet, Infiniband, ...). Enfin l'infrastructure d'une grille est composée d'un nombre important de sites et de machines qui sont susceptibles de tomber en panne à tout moment.

À cela s'ajoute le fait que de nouveaux sites peuvent être ajoutés ou retirés de la grille sans trop impacter le fonctionnement de la grille. De par leur hétérogénéité, leur gestion décentralisée et leur taille, les grilles sont des infrastructures très complexes à mettre en oeuvre. La transparence n'est assurée qu'à moitié parce qu' il faut avoir une information précise des ressources dans un site pour pouvoir distribuer les tâches convenablement. Cependant à l'intérieur d'un site, la machine qui effectue concrètement la tâche n'est pas connue en général [2].

VI.3. Cloud :

Le cloud est un concept plus récent dont une définition unanime tarde à voir le jour. Cette divergence découle des principes considérés par les chercheurs pour définir le cloud. En effet , certains auteurs mettent l'accent sur le passage à l'échelle et la mutualisation de l'usage des ressources alors que d'autres privilégient le concept de virtualisation ou le business model (collaboration et pay-as-you-go). Cependant, il est unanimement reconnu que le cloud permet l'utilisation de la mémoire et des capacités de calcul des ordinateurs et des serveurs répartis dans le monde entier et liés par un réseau, tel Internet.

Les ressources sont en général logées dans des data centres qui sont géographiquement distribués dans l'optique de garantir le passage à l'échelle et la disponibilité. Avec le cloud, les utilisateurs ne sont plus propriétaires de leurs serveurs informatiques, mais peuvent ainsi accéder de manière évolutive à de nombreux services en ligne sans avoir à gérer l'infrastructure sous-jacente, souvent complexe. C'est pourquoi, on peut considérer le cloud comme une extension des

ASP. Les applications et les données ne se trouvent plus sur l'ordinateur local, mais dans un nuage (cloud) composé d'un certain nombre de serveurs distants interconnectés au moyen d'une excellente bande passante indispensable à la fluidité du système. L'accès au service se fait par une application standard facilement disponible, la plupart du temps un navigateur Web. Les service offerts par le cloud sont nombreux parmi lesquels nous avons [2]:

- Infrastructure as a service (IaaS) : c'est un service qui donne à l'utilisateur un ensemble de ressources de traitement et de stockage dont il a besoin à un instant précis pour effectuer ses tâches.
- Platform as a Service (PaaS) : ce service, en dehors de fournir une infrastructure virtuelle, assure aussi la plateforme logicielle pour que les applications de l'utilisateur puissent tourner.
- Software as a Service (SaaS) : c'est une alternative de toute application locale. Un exemple de ce cas est l'utilisation en ligne de la suite bureautique de Microsoft Office.

Pour assurer ces services qui varient d'un utilisateur à un autre, l'architecture des clouds est composée à son niveau le plus basique d'une couche logique composée d'un ensemble de machines virtuelles et d'une couche physique composée de data centres. Ainsi, plusieurs machines virtuelles peuvent être démarrées dynamiquement sur une seule machine physique pour satisfaire les besoins de plusieurs services. Les data centres qui regroupent les machines physiques sont en général répartis sur des sites géographiquement distants afin d'assurer une haute disponibilité. En plus cette répartition permet aussi d'assurer un niveau de performances élevé en branchant un utilisateur sur le site le plus proche de son emplacement afin de réduire les délais de communication.

La mutualisation du matériel permet d'optimiser les coûts par rapport aux systèmes conventionnels et de développer des applications partagées sans avoir besoin de posséder ses propres machines dédiées au calcul. Comme pour la virtualisation, l'informatique dans le nuage est plus économique grâce à son évolutivité. En effet, le coût est fonction de la durée de l'utilisation du service rendu et ne nécessite aucun investissement préalable (homme ou machine). Notons également que l'élasticité du nuage permet de fournir des services évolutifs et donc de supporter les montées de charges. Par exemple, Salesforce.com, pionnier dans le

domaine de l'informatique dans le nuage gère les données de 54 000 entreprises, et leurs 1,5 millions d'employés, avec seulement 1000 serveurs (mars 2009). De plus, les services sont extrêmement fiables, car basés sur des infrastructures performantes possédant des politiques efficaces de tolérance aux pannes (notamment des répliques). Grâce à ses avantages, on assiste aujourd'hui à une multiplication rapide d'entreprises qui proposent des solutions cloud parmi lesquelles figurent : Amazon, IBM, Microsoft, Google, etc.

Cependant, le problème fondamental reste d'une part la sécurisation de l'accès à l'application entre le client et le serveur distant. On peut aussi ajouter le problème de sécurité générale du réseau de l'entreprise : sans le cloud computing, une entreprise peut mettre une partie de son réseau en local et sans aucune connexion (directe ou indirecte) à internet, pour des raisons de haute confidentialité par exemple ; dans le cas du cloud computing, elle devra connecter ces postes à internet (directement ou pas) et ainsi les exposer à un risque d'attaque ou a des violations de confidentialité [2].

VII. Conclusion :

Dans ce chapitre nous avons fait une synthése sur les systèmes distribués, nous avons présenté en détail :

- Les caractéristiques d'un système distribué.
- Les types de communications (synchrones et asynchrones).
- Les Modes de communications (Socket, middleware).

Dans le chapitre suivant on s'intéressera aux différentes architectures N-tiers avec une présentation de quelques exemples de chaque architecture.

Chapitre II :

Architecture N-tiers

I. Introduction :

L'objectif de ce chapitre et de présenter brièvement les modèles de type un tiers, deux tiers et trois tiers, et une synthèse approfondie sur les architectures logicielles n-tiers. On abordera les technologies utilisées et les moyens à mettre en œuvre. Nous donnons des explications des termes clients légers, clients lourds et une étude sur les avantages et les inconvénients des différentes architectures.

II. Trois niveaux d'abstraction d'une application :

En règle générale, une application informatique peut être découpée en trois niveaux d'abstraction distincts :

II.1. Couche de présentation :

Encore appelée IHM, permet l'interaction de l'application avec l'utilisateur. Cette couche gère les saisies au clavier, à la souris et la présentation des informations à l'écran. Dans la mesure du possible, elle doit être conviviale et ergonomique [6].

II.2. Services métier :

Décrivant les travaux à réaliser par l'application. Ils peuvent être découpés en deux familles :

- **Locaux :** regroupant les contrôles effectués au niveau du dialogue avec l'IHM, visant essentiellement le contrôle et l'aide à la saisie.
- **Globaux :** constituant l'application elle-même. Cette couche, appelée Business Logic ou couche métier, contient les règles internes qui régissent une entreprise donnée [6].

II.3. Persistance :

Ou plus exactement l'accès aux données, regroupant l'ensemble des mécanismes permettant la gestion des informations stockées par l'application.

Ces trois niveaux peuvent être imbriqués ou répartis de différentes manières entre plusieurs machines physiques.

Le noyau de l'application est composé de la logique de l'affichage et la logique des traitements. Le découpage et la répartition de ce noyau permettent de distinguer les architectures applicatives suivantes [6] :

- L'architecture 1-tiers.
- L'architecture 2-tiers.
- L'architecture 3-tiers.
- Les architectures n-tiers.

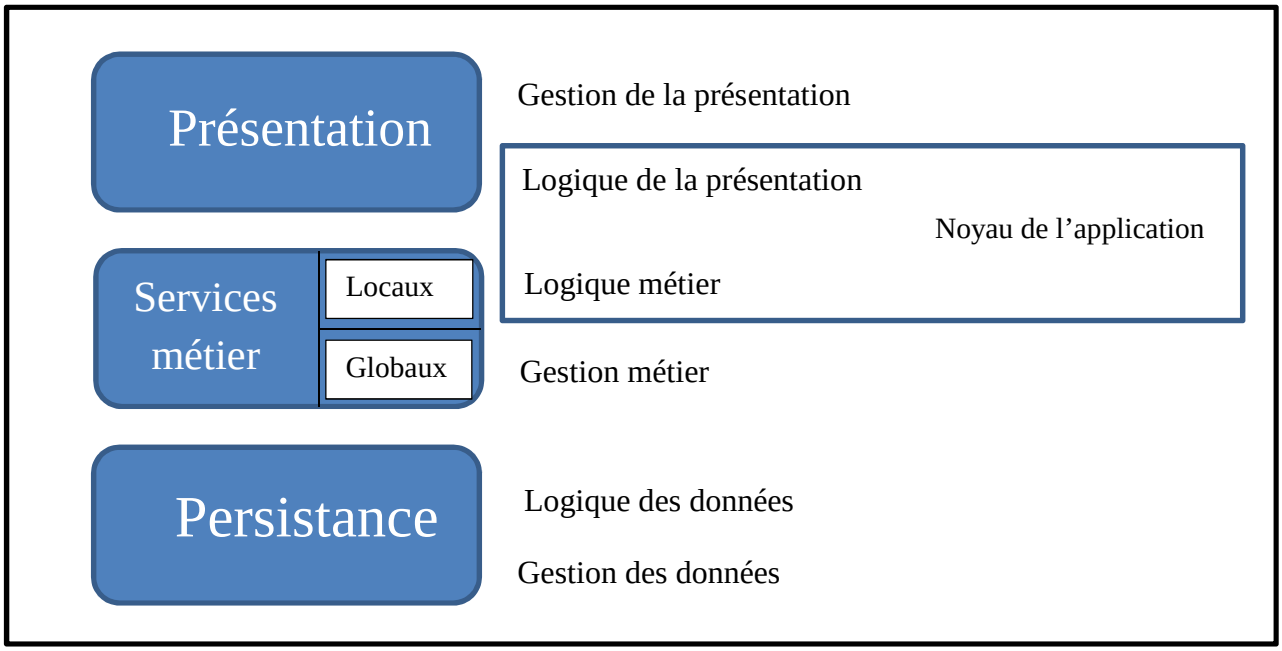

Figure II.1 : Les trois niveaux d'une application informatique.

III. L'architecture un tiers :

I.1. Présentation :

Dans une application un tiers, les trois couches applicatives sont intimement liées et s'exécutent sur le même ordinateur. On ne parle pas ici d'architecture client-serveur, mais d'informatique centralisée.

Dans un contexte multi-utilisateurs, on peut rencontrer deux types d'architecture mettant en œuvre des applications un tiers :

- Des applications sur site central.
- Des applications réparties sur des machines indépendantes communiquant par partage de fichiers.

III.4. Solutions sur site central (mainframe) :

Historiquement, les applications sur site central furent les premières à proposer un accès multiutilisateurs.

Dans ce contexte, les utilisateurs se connectent aux applications exécutées par le serveur central (le *mainframe*) à l'aide de terminaux passifs se comportant en esclaves. C'est le serveur central qui prend en charge l'intégralité des traitements, y compris l'affichage qui est simplement déporté sur des terminaux passifs.

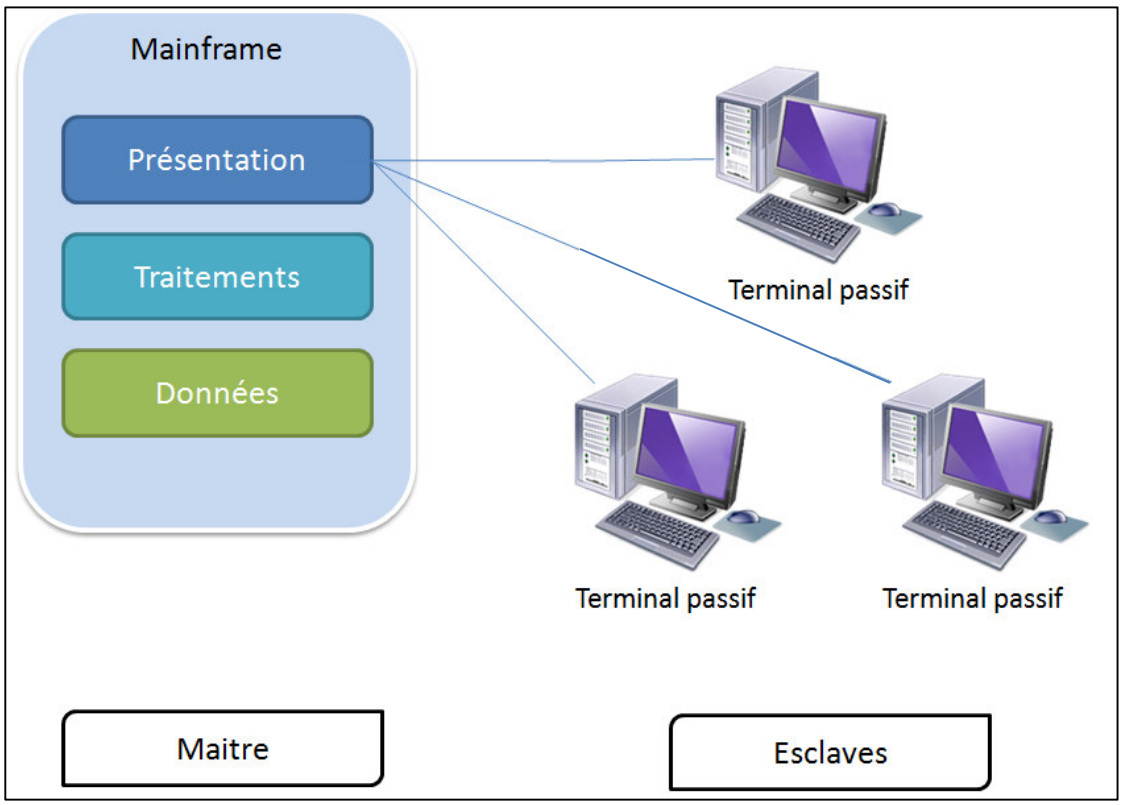

Figure II.2: Architecture d'une application sur site central.

Ce type d'organisation brille par sa grande facilité d'administration et sa haute disponibilité. Il bénéficie aujourd'hui d'une large palette d'outils de conception, de programmation et d'administration ayant atteint un niveau de maturité et de fiabilité leur permettant encore de soutenir la comparaison avec des solutions beaucoup plus modernes. De plus, la centralisation de la puissance sur une seule et même machine permet une utilisation optimale des ressources et se prête très bien à des traitements par lots (en batch) [7].

L'émergence des interfaces utilisateur de type Windows - Macintosh, démocratisées par les outils bureautiques sur micro-ordinateur a fortement démodé les applications mainframe. Ces dernières exploitaient exclusivement une interface utilisateur en mode caractère jugée obsolète par les utilisateurs, qui réclamaient alors massivement une convergence entre la bureautique et le système d'information de l'entreprise.

Le ré-habillage d'application centralisée, ou revamping, permet de rajeunir ces dernières en leur faisant profiter d'une interface graphique moderne (GUI), mais au prix d'une telle lourdeur3.4 qu'il doit être considéré comme une simple étape vers une architecture plus moderne [7].

III.5. Les applications un tiers déployées :

Avec l'arrivée dans l'entreprise des premiers PC en réseau, il est devenu possible de déployer une application un tiers sur plusieurs ordinateurs indépendants.

Ce type de programme est simple à concevoir et à mettre en œuvre. Il existe pour cela denombreux environnements de développement (dBase, Ms Access, Lotus Approach, Paradox...) qui sont souvent intégrés aux principales suites bureautiques. L'ergonomie des applications mises en œuvre, basée sur celle des outils bureautiques, est très riche.

Ce type d'application peut être très satisfaisant pour répondre aux besoins d'un utilisateur isolé et sa mise en œuvre dans un environnement multi-utilisateur est envisageable.

Dans ce contexte, plusieurs utilisateurs se partagent des fichiers de données stockés sur un serveur commun. Le moteur de base de données est exécuté indépendamment sur chaque poste client. La gestion des conflits d'accès aux données doit être prise en charge par chaque programme de façon indépendante, ce qui n'est pas toujours évident. Lors de l'exécution d'une requête, l'intégralité des données nécessaires doit transiter sur le réseau et on arrive vite à saturer ce dernier. De plus, la cohabitation de plusieurs moteurs de base de données indépendants manipulant les mêmes données peut devenir assez instable. Il n'est pas rare de rencontrer des conflits lors de la consultation ou de la modification simultanée d'un même enregistrement par plusieurs utilisateurs. Ces conflits peuvent altérer l'intégrité des données. Enfin, il est difficile d'assurer la confidentialité des données.

Ce type de solution est donc à réserver à des applications non critiques exploitées par de petits groupes de travail (une dizaine de personnes au maximum) [7].

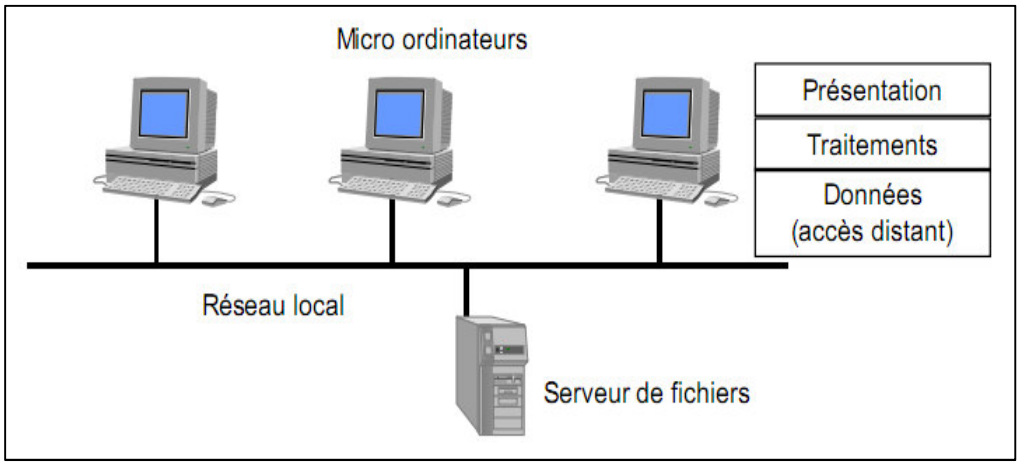

Figure II.3: Architecture d'une application un tiers déployées

III.6. Avantages et limites :

III.6.1. Avantage :

- **Mainframe** : la fiabilité des solutions sur site central qui gèrent les données de façon centralisée
- **Un tiers déployé :** l'interface utilisateur moderne des applications.

III.6.2. Limite :

- **Mainframe :** interface utilisateur en mode caractères
- **Un tiers déployé :** cohabitation d'applications exploitant des données communes peu fiable au-delà d'un certain nombre d'utilisateurs [7].

Il a donc fallu trouver une solution conciliant les avantages de cette architecture. Pour se faire, il a fallu scinder les applications en plusieurs parties distinctes et coopérantes :

- Gestion centralisée des données.
- Gestion locale de l'interface utilisateur.

IV. L'architecture 2-tiers :

IV.1. Modèle client/serveur de données :

IV.1.1. Concept de client/serveur :

a. **Définition :**

La définition d'une architecture client/serveur est la suivante : ensemble de machines pouvant communiquer entre elles, et s'échangeant des services : traitement, données ou ressources.

Concrètement, il s'agit de plusieurs machines reliées par un réseau local ou étendu. Certaines de ces machines sont capables d'offrir des services, tels que l'exécution d'une procédure, la recherche de données, le partage d'une ressource : ce sont des **serveurs.**

D'autres vont utiliser ces services : ce sont les **clients.**

Si l'on s'en tient à la définition toute machine peut être client ou serveur ou les deux à la fois ou alternativement l'un puis l'autre [8]**.**

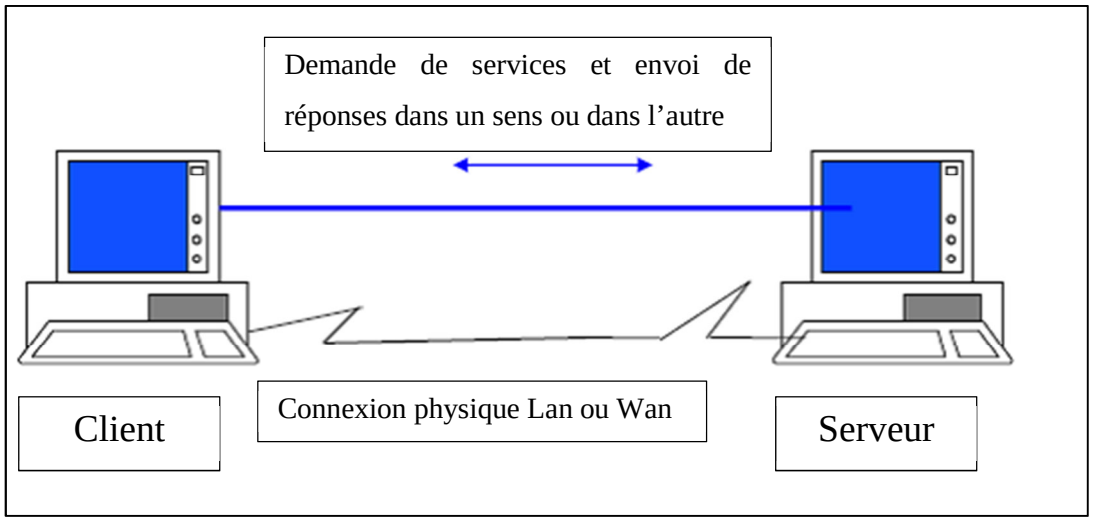

Figure II.4 : Architecture Client/serveur.

b. Le schéma du Gartner Group :

Un groupe d'étude composé d'acteurs du marché a défini 6 types d'architectures client/serveur, en considérant qu'une application de gestion quelle qu'elle soit se décompose en trois parties : présentation, traitements et données.

- **Présentation (ou interface) :**

C'est à dire à la fois la définition du scénario de saisie et l'algorithme permettant d'interpréter les interactions de l'utilisateur, tel que l'API Windows.

- **Traitements :**

C'est à dire les algorithmes propres à l'application (calcul du montant de la facture, enchaînement des traitements, etc...) et des algorithmes réutilisables par plusieurs applications (tris, édition, calculs statistiques, etc...).

- **Données :**

Il faut distinguer la logique des données, c'est à dire la vue logique des données utilisée par l'application et la gestion des données, c'est à dire tout le travail que fait le S.G.B.D. (recherche des données, stockage, contrôle des accès, etc...) [8].

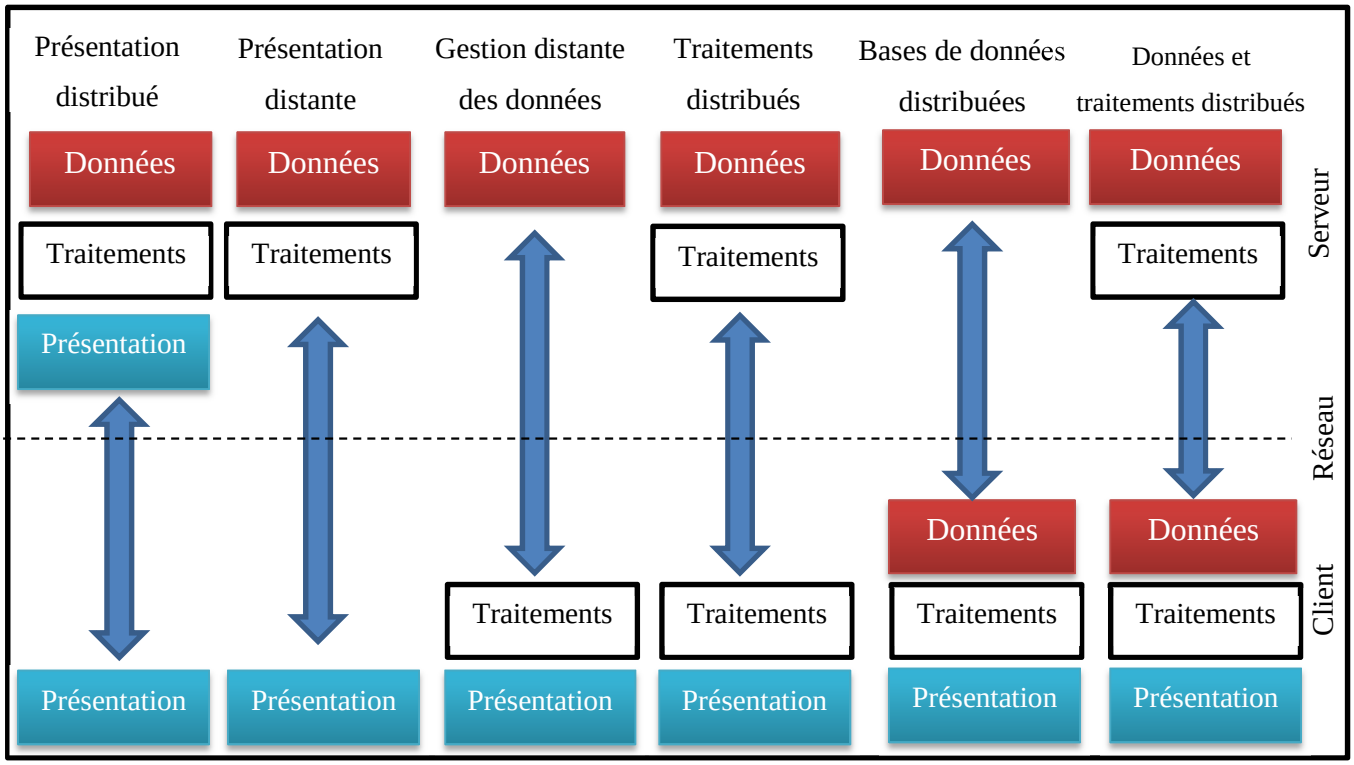

Figure II.5 : Le schéma du Gartner Group.

Dans la Figure II.5, le trait horizontal représente le réseau et les flèches entre client et serveur, le trafic réseau généré par la conversation entre client et serveur.

Le **Gartner Group** distingue les types de client-serveur suivants, en fonction du type de service déporté du cœur de l'application :

- **Présentation distribuée** : Correspond à l'habillage « graphique» de l'affichage en mode caractères d'applications fonctionnant sur site central. Cette solution est aussi appelée *revamping*. La classification «client-serveur» du revamping est souvent jugée abusive, du fait que l'intégralité des traitements originaux est conservée et que le poste client conserve une position d'esclave par rapport au serveur.
- **Présentation distante** : Encore appelée client-serveur de présentation. L'ensemble des traitements est exécuté par le serveur, le client ne prend en charge que l'affichage. Ce type d'application présentait jusqu'à présent l'inconvénient de générer un fort trafic réseau et de ne permettre aucune répartition de la charge entre client et serveur. S'il n'était que rarement retenu dans sa forme primitive, il connaît aujourd'hui un très fort regain d'intérêt avec l'exploitation des standards Internet.
- **Gestion distante des données** : Correspond au client-serveur de données, sans doute le type de client-serveur le plus répandu. L'application fonctionne dans sa totalité sur le client, la gestion des données et le contrôle de leur intégrité sont assurés par un SGBD centralisé. Cette architecture, de part sa souplesse, s'adapte très bien aux applications de type *infocentre*, interrogeant la base de façon ponctuelle. Il génère toutefois un trafic réseau assez important et ne soulage pas énormément le poste client, qui réalise encore la grande majorité des traitements.
- **Traitement distribué** : Correspond au client-serveur de traitements. Le découpage de l'application se fait ici au plus près de son noyau et les traitements sont distribués entre le client et le(s) serveur(s). Le client-serveur de traitements s'appuie, soit un mécanisme d'appel de procédure distante, soit sur la notion de procédure stockée proposée par les principaux SGBD du marché. Cette architecture permet d'optimiser la répartition de la charge de traitement entre machines et limite le trafic réseau. Par contre il n'offre pas la même souplesse que le client-serveur

de données puisque les traitements doivent être connus du serveur à l'avance.

- **Bases de données distribuées** : Il s'agit d'une variante du client-serveur de données dans laquelle une partie de données est prise en charge par le client. Ce modèle est intéressant si l'application doit gérer de gros volumes de données, si l'on souhaite disposer de temps d'accès très rapides sur certaines données ou pour répondre à de fortes contraintes de confidentialité. Ce modèle est aussi puissant que complexe à mettre en œuvre.
- **Données et traitements distribués** : Ce modèle est très puissant et tire partie de la notion de composants réutilisables et distribuables pour répartir au mieux la charge entre client et serveur.

  C'est, bien entendu, l'architecture la plus complexe à mettre en œuvre [8].

IV.1.2. Client/serveur de données :

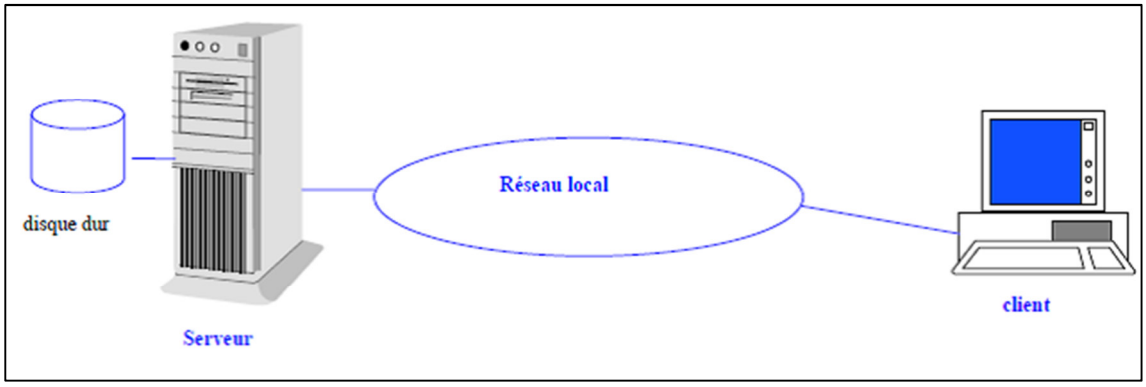

Figure II.6 : client/serveur de données.

Dans une architecture deux tiers, encore appelée client-serveur de première génération ou client-serveur de données, le poste client se contente de déléguer la gestion des données à un service spécialisé. Le cas typique de cette architecture est l'application de gestion fonctionnant sous Ms-Windows et exploitant un SGBD centralisé.

Ce type d'application permet de tirer parti de la puissance des ordinateurs déployés en réseau pour fournir à l'utilisateur une interface riche, tout en garantissant la cohérence des données, qui restent gérées de façon centralisée.

La gestion des données est prise en charge par un SGBD centralisé, s'exécutant le plus souvent sur un serveur dédié. Ce dernier est interrogé en utilisant un langage de requête qui, le plus souvent, est SQL…

Le dialogue entre client et serveur se résume donc à l'envoi de requêtes et au retour des données correspondant aux requêtes. Ce dialogue nécessite l'instauration d'une communication entre client et serveur [8].

a. Le dialogue avec les clients :

Le principe du dialogue entre clients et serveurs peut être schématisé ainsi (voir **Figure II.7**) :

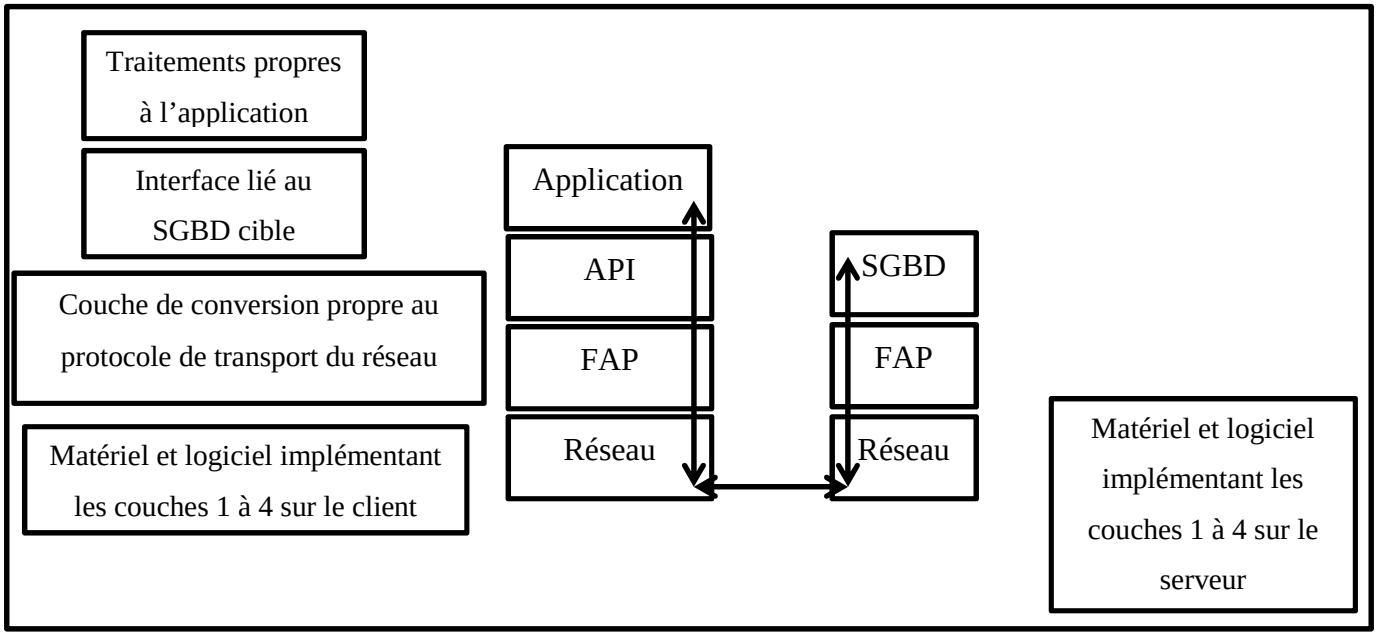

Figure II.7 : Dialogue entre clients et serveurs.

Le client envoie une requête au serveur de données. Cette requête est :

- Traduite dans un format compréhensible par le SGBD (API).
- Formatée en une suite de trames compatibles avec le protocole réseau (FAP).
- Véhiculée par le réseau jusqu'au serveur concerné.

Sur le serveur l'opération inverse est effectuée :

- Recomposition de la requête à partir des trames reçues (FAP).
- Le SGBD interprète la requête et l'exécute.

Le résultat de la requête prend ensuite le chemin inverse [8].

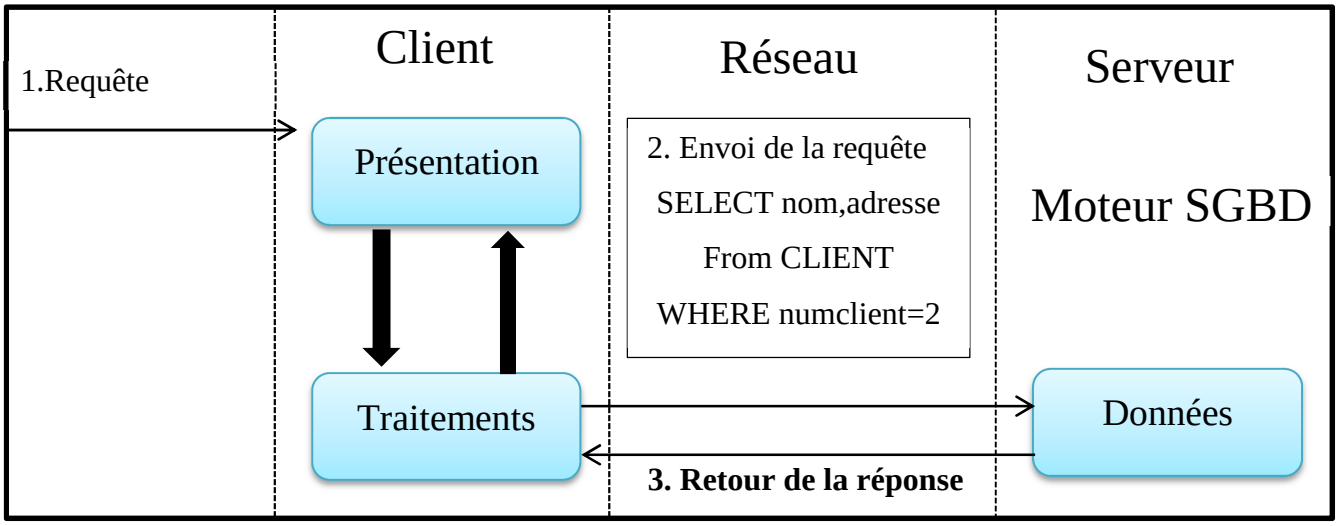

Figure II.8 : Schéma de fonctionnement du Client/serveur.

Cet échange de messages transite à travers le réseau reliant les deux machines. Il met en œuvre des mécanismes relativement complexes qui sont, en général, pris en charge par un middleware.

b. Le rôle du serveur de données :

Il est à l'écoute des requêtes provenant des clients. Il y répond en faisant appel aux fonctions d'un SGBD classique.

- **Langage de Description de Données (L.D.D.) :**

Ce langage permet de décrire les données (Type, Longueur, Nature ...), les relations entre les données, les règles de gestion, les domaines de valeur, etc... En langage SQL c'est par exemple l'instruction CREATE ….

- **Langage de Manipulation de Données (L.M.D.) :**

Ce langage sert à exécuter les opérations d'ajout, suppression, modification des données. Il permet également l'interrogation des données. C'est en général le langage utilisé dans les requêtes. En langage SQL c'est par exemple l'instruction SELECT ….

- G**estion du stockage :**

Le SGBD organise le stockage des données sur disque. Cette organisation interne est transparente pour l'utilisateur et le programmeur. Il offre une vue logique des données.

- **Partage des données :**

Dans ce contexte multi-utilisateurs, c'est le SGBD qui gère le partage des données entre les différents utilisateurs (ou clients) C'est lui qui en s'appuyant en général, sur le système d'exploitation évite les situations de conflit et les situations d'interblocage ou étreinte fatale.

- C**onfidentialité :**

Le SGBD assure la confidentialité des données : il contrôle les droits d'accès des utilisateurs. C'est à dire dans ce contexte des applications clientes lorsqu'elles demandent la connexion.

- S**écurité, fiabilité, cohérence :**

Le SGBD assure la sécurité des données. C'est à dire qu'il doit veiller à ce que les données restent cohérentes. Par exemple il doit faire en sorte qu'une commande ne puisse pas être créée tant que toutes les lignes de commande ne sont pas créés, ou qu'un client ne soit pas supprimé sans avoir au préalable supprimé toutes les commandes attachées à ce client.

Dans les deux cas ci-dessus (création d'une commande ou suppression d'un client) il y a plusieurs actions consécutives à exécuter dans la base de données. Pour que la base de données reste cohérente, il faut que ces actions soient toutes exécutées ou alors aucunes.

L'ensemble de ces actions est appelé **transaction.**

Le SGBD doit rendre les transactions interruptibles.

Si une interruption intervient au cours d'une transaction, le système remet les données dans l'état où elles étaient avant le début de la transaction.

- S**auvegarde et restauration :**

La sauvegarde et la restauration sont deux outils du SGBD permettant de mémoriser puis de retrouver une base de données dans un état cohérent.

- C**ontrôle d'intégrité :**

La plupart des SGBD offrent la possibilité de décrire les règles de gestion du système d'informations de l'entreprise. Par exemple, si un client n'existe pas, on ne peut pas lui associer de facture.

Une fois toutes les règles décrites, le SGBD vérifie que les mises à jour effectuées sur les données respectent ces règles. Ainsi, il refusera la création de la facture tant que le client ne sera pas créé [8].

IV.2. Limites du client-serveur deux tiers : client lourd

L'expérience a démontré qu'il était coûteux 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, ou *fat client*.

L'architecture client-serveur de première génération s'est heurtée à ce constat à l'heure des premiers bilans :

- On ne peut pas soulager la charge du poste client, qui supporte la grande majorité des traitements applicatifs.
- Le poste client est fortement sollicité, il devient de plus en plus complexe et doit être mis à jour régulièrement pour répondre aux besoins des utilisateurs.
- La conversation entre client et serveur est assez bruyante et s'adapte mal à des bandes passantes étroites. De ce fait, ce type d'application est souvent cantonné au réseau local de l'entreprise.
- Les applications se prêtent assez mal aux fortes montées en charge car il est difficile de modifier l'architecture initiale.
- La relation étroite qui existe entre le programme client et l'organisation de la partie serveur complique les évolutions de cette dernière.
- Ce type d'architecture est grandement rigidifié par les coûts et la complexité de sa maintenance.

Malgré tout, l'architecture deux tiers présente de nombreux avantages qui lui permettent de présenter un bilan globalement positif :

- Elle permet l'utilisation d'une interface utilisateur riche.

- Elle a permis l'appropriation des applications par l'utilisateur.
- Elle a introduit la notion d'interopérabilité [8].

Pour résoudre les limitations du client-serveur deux 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.

V. L'architecture trois tiers :

V.1. Objectifs :

Les limites de l'architecture deux tiers proviennent en grande partie de la nature du client utilisé :

- Le frontal est complexe et non standard (même s'il s'agit presque toujours d'un PC sous Windows).
- Le middleware entre client et serveur n'est pas standard.

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 [8].

V.2. Premières tentatives :

Les premières tentatives de mise en œuvre d'architecture trois tiers proposaient l'introduction d'un serveur d'application centralisé exploité par les postes clients à l'aide dialogue RPC propriétaire.

Les mécanismes nécessaires à la mise en place de telles solutions existent depuis longtemps :

- UNIX propose un mécanisme de RPC intégré au système NFS.
- NetDDE, puis DCOM de Microsoft permettent l'instauration d'un dialogue RPC entre client et serveur.
- Certains environnements de développement facilitent la répartition des traitements entre le poste client et le serveur.

- Des solutions propriétaires comme ForT ou Implicite permettent l'exploitation d'un serveur d'application par des clients simplement équipés d'un environnement d'exécution (runtime).

Cependant, l'application de ces technologies dans une architecture trois tiers est complexe et demande des compétences très pointues, du fait du manque de standards. Malgré ses avantages techniques évidents, ce type d'application fut souvent jugé trop coûteux à mettre en place.

Ce premier essai s'est soldé par un échec dû en grande partie à l'absence de standard et à la complexité des mécanismes mis en œuvre [9].

V.3. Répartition des traitements :

L'architecture trois tiers, encore appelée client-serveur de deuxième génération ou client-serveur distribué, sépare l'application en trois niveaux de service distincts :

- **Premier niveau :** l'affichage et les traitements locaux (contrôles de saisie, mise en forme de données...) sont pris en charge par le poste client.
- D**euxième niveau :** les traitements applicatifs globaux sont pris en charge par le service applicatif.
- T**roisième niveau :** les services de base de données sont pris en charge par un SGBD.

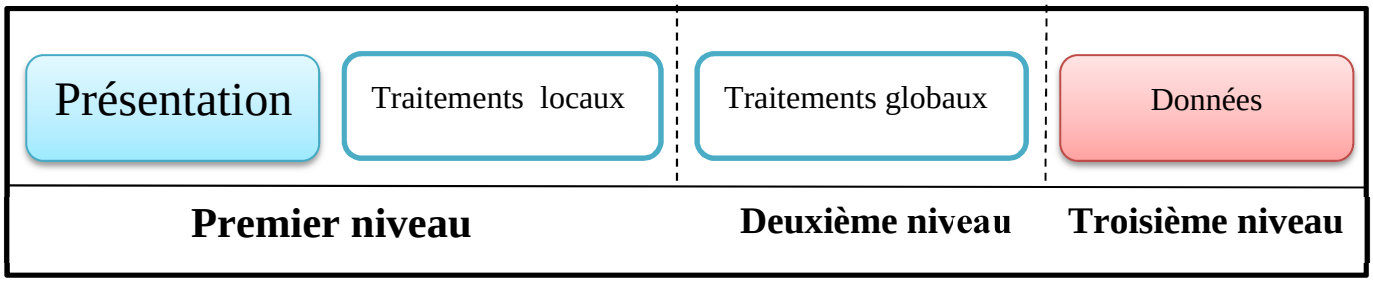

Figure II.9 : Les niveaux de l'architecture 3-tiers.

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

- Le poste client ne supporte plus l'ensemble des traitements, il est moins sollicité et peut être moins évolué, donc moins coûteux.

- Les ressources présentes sur le réseau sont mieux exploitées, puisque les traitements applicatifs peuvent être partagés ou regroupés (le serveur d'application peut s'exécuter sur la même machine que le SGBD).
- La fiabilité et les performances de certains traitements se trouvent améliorées par leur centralisation.
- Il est relativement simple de faire face à une forte montée en charge, en renforçant le service applicatif.

Dans le cadre d'un Intranet, le poste client prend la forme d'un simple navigateur Web, le service applicatif est assuré par un serveur HTTP et la communication avec le SGBD met en œuvre les mécanismes bien connus des applications client-serveur de la première génération.

Ce type d'architecture fait une distinction nette entre deux tronçons de communication indépendants et délimités par le serveur HTTP :

- Le premier tronçon relie le poste client au serveur Web pour permettre l'interaction avec l'utilisateur et la visualisation des résultats. On l'appelle **circuit froid** et n'est composé que de standards (principalement HTML et HTTP). Le serveur Web tient le rôle de « façade HTTP ».
- Le deuxième tronçon permet la collecte des données, il est aussi appelé **circuit chaud**. Les mécanismes utilisés sont comparables à ceux mis en œuvre pour une application deux tiers. Ils ne franchissent jamais la façade HTTP et, de ce fait, peuvent évoluer sans impacter la configuration des postes clients [9].

V.4. Client léger :

V.4.1. Présentation

Dans l'architecture trois tiers, le poste client est communément appelé client léger ou *Thin Client*, par opposition au client lourd des architectures 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 données. Il est souvent constitué d'un simple navigateur Internet.

Le poste client ne communique qu'avec la façade HTTP de l'application et ne dispose d'aucune connaissance des traitements applicatifs ou de la structure des données

exploitées. Les évolutions de l'application sont donc possibles sans nécessiter de modification de la partie cliente.

Par exemple, un internaute se connectant à *www.yahoo.fr* pour effectuer une recherche provoque, sans même le savoir, l'exécution de traitements sur le serveur. Si ces traitements évoluent, ce qui doit arriver relativement souvent, le client continuera à utiliser le service sans se rendre compte des changements (sauf s'ils lui apportent de nouveaux services).

De plus, ce même internaute peut se connecter au serveur en utilisant tout type de poste client disposant d'un navigateur compatible HTML (PC sous Windows, Macintosh, Station Unix, WebPhone...).

On voit donc ici la force des architectures trois tiers par rapport au client-serveur de première génération. Le déploiement est immédiat, les évolutions peuvent être transparentes pour l'utilisateur et les caractéristiques du poste client sont libres.

V.4.2. Ergonomie :

a. Utilisation d'HTML :

Les pages HTML, même avec l'aide de langages de script, sont loin d'atteindre les possibilités offertes par l'environnement Windows :

- Le multifenêtrage n'est pas facile à mettre en œuvre.
- Le déroulement de l'application doit se faire séquentiellement.
- Les pages affichées sont relativement statiques.
- Le développement multi-plateforme peut être contraignant.
- L'ergonomie de l'application est limitée aux possibilités du navigateur.

Pour ces raisons, certaines applications ne sont pas réalisables dans une architecture de type Intranet (infocentres, applications bureautiques...).

Dans les autres cas, l'appauvrissement de l'interface utilisateur est souvent le gage d'une plus grande facilité de prise en main de l'application.

Il est rare en effet de devoir suivre une formation pour apprendre à se servir d'un site Web particulier. La plupart du temps, une formation générale à l'ergonomie des sites Web suffit.

Cette perte de richesse peut donc se transformer en avantage, à condition de respecter une charte graphique et ergonomique cohérente pour toutes les applications Intranet d'une entreprise.

Il est possible d'aller au-delà des possibilités offertes par le langage HTML en y introduisant des applets Java ou des contrôles ActiveX. Nous ne parlerons pas ici des modules d'extension du navigateur, encore appelés *plug-in*, qui étaient en vogue avant l'arrivée des solutions Java et ActiveX, car cette solution est trop contraignante à déployer.

b. Utilisation de Java :

Java est un langage de développement orienté objet et multi-plateforme introduit par Sun en 1995. Il s'agit de l'adaptation à Internet du langage OAK, initialement étudié pour les environnements de petite taille. Il permet, entre autre, d'écrire de petites applications, appelées *applet*, pouvant être intégrées à des pages HTML pour en enrichir le contenu.

Le caractère multi-plateforme de Java se prête bien à une utilisation sur Internet, où les caractéristiques des postes clients ne sont pas maîtrisées. Il repose sur l'utilisation d'un interpréteur de pseudo-code Java, pompeusement appelé « machine virtuelle ».

Les programmes Java ne sont pas compilés en code machine, mais en pseudo-code Java uniquement compréhensible par la machine virtuelle. Cette dernière interprète le code et se charge de lier les modules au moment de l'exécution, en les téléchargeant si nécessaire. La liaison dynamique des modules au moment de l'exécution permet d'optimiser le trafic réseau, puisqu'on ne charge que le strict nécessaire.

Pour ces raisons, un programme Java s'exécute plus lentement que son équivalent compilé. La compilation à la volée des programmes Java et plus encore, la technologie d'optimisation dynamique de code *HotSpot* de Sun, permettent de réduire l'écart de performance avec des programmes compilés.

Java propose aujourd'hui une large palette de composants graphiques et multimédia qui permettent d'atteindre la richesse fonctionnelle des applications Windows.

Une applet Java est aussi capable d'exploiter directement un serveur de données en utilisant JDBC ou de faire appel à des procédures distantes en utilisant RMI ou CORBA.

c. Utilisation d'ActiveX :

Quand Microsoft entend parler de « multi-plateforme », il comprend « fonctionnement en dehors de Windows » et, par extension directe, « danger ». De ce fait, il ne pouvait rester sans réponse devant le phénomène Java.

Après avoir essayé, sans succès, de confiner Java dans le rôle d'un simple langage de programmation dépendant de Windows, Microsoft appuie sa politique Intranet sur sa technologie ActiveX.

ActiveX est une adaptation Internet des composants OCX constituant la clef de voûte de l'architecture DNA. Les composants ActiveX exploitent les possibilités de Windows et sont donc capables d'offrir une grande richesse fonctionnelle aux applications qui les utilisent.

Ils peuvent être intégrés à une page HTML mais, à la différence des applets Java, ils persistent sur le poste client après utilisation. Ils ne sont alors remplacés que si une nouvelle version du composant est utilisée. Ce mécanisme permet d'optimiser les transferts de composants sur le réseau mais rappelle grandement l'installation locale d'application, avec ses effets négatifs sur l'alourdissement du poste client.

Les composants ActiveX peuvent communiquer entre eux en utilisant la technologie DCOM ou avec des bases de données, en utilisant ODBC.

En fait, l'utilisation des composants ActiveX rend les applications dépendantes de la plateforme Windows. On se prive alors des possibilités offertes par d'autres systèmes d'exploitation ou de postes clients plus simples et donc, de la possibilité de faire baisser le coût de possession des postes clients.

En prenant du recul, on s'aperçoit que l'utilisation d'objets ActiveX dans une application Intranet fait perdre une grande partie de l'intérêt de cette architecture et rappelle fortement le client-serveur deux tiers.

V.5. Exemples de clients légers

Le client léger peut prendre plusieurs formes :

- Un poste Windows équipé d'un navigateur HTML.
- Une station réseau du type NC. Contrairement à un simple terminal X, ce type de station prend en charge des traitements locaux (affichage, contrôle de saisie, mise en forme de donnée...). Ce type de poste client se démarque par un coût de possession très largement inférieur à celui d'un PC sous Windows.

- Un terminal Windows (NetPC) correspondant à un PC minimal.

Le client léger à souvent été présenté comme le successeur du PC sous Windows. En fait, le client léger ne prétend pas remplacer tous les PC dans l'entreprise, il se présente plutôt comme une alternative à ce dernier pour certains besoins particuliers et cohabite très bien avec l'existant.

V.6. Service applicatif :

V.6.1. Présentation :

Dans une architecture trois tiers, la logique applicative est prise en charge par le serveur HTTP. Ce dernier se retrouve dans la position du poste client d'une application deux tiers et les échanges avec le serveur de données mettent en œuvre les mécanismes déjà vus dans ce type d'application.

Les développements mis en œuvre sur le serveur HTTP doivent être conçus spécifiquement. Ils peuvent mettre en œuvre :

- CGI : le mécanisme standard et reconnu de tous, mais grand consommateur de ressources.
- NSAPI ou ISAPI : les API de Netscape et Microsoft permettant l'écriture d'applications multithread intégrées au serveur http.
- Les scripts serveur comme ASP (Active Server Page pour IIS) ou PHP (l'équivalent pour Apache) sont interprétés par le serveur pour générer des pages dynamiquement.
- Les servlets Java : qui appliquent le mécanisme des applets aux traitements réalisés sur le serveur.

V.6.2. Gestion des transactions :

Comme le protocole HTTP n'assure pas de gestion d'états, les applications transactionnelles doivent gérer elles-mêmes le contexte utilisateur afin de contrôler :

- Le cheminement de l'utilisateur dans l'application.
- Les actions et saisies de l'utilisateur.
- L'identité de l'utilisateur.
- La gestion des transactions.

Il existe différentes méthodes de gestion de contexte :

- Attribution d'un identifiant à chaque étape de la transaction.

- Stockage de l'ensemble du contexte sur le poste client.

Ces méthodes nécessitent le stockage d'informations sur le poste client :

- Dans un cookie déposé sur le poste client.
- Dans une URL longue.
- Dans des variables cachées au sein de la page HTML.

V.7. Limitations :

L'architecture trois tiers a corrigé les excès du client lourd en centralisant une grande partie de la logique applicative sur un serveur HTTP. Le poste client, qui ne prend à sa charge que la présentation et les contrôles de saisie, s'est trouvé ainsi soulagé et plus simple à gérer.

Par contre, le serveur HTTP constitue la pierre angulaire de l'architecture et se trouve souvent fortement sollicité et 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 oeuvre sont relativement complexes à maintenir et la gestion des sessions est compliquée.

Les contraintes semblent inversées par rapport à celles rencontrées avec les architectures deux tiers : le client est soulagé, mais le serveur est fortement sollicité. Le phénoméne fait penser à un retour de balancier.

Le juste équilibrage de la charge entre client et serveur semble atteint avec la génération suivante : les architectures n-tiers.

VI. Les architectures n-tiers :

VI.1. Présentation :

L'architecture n-tiers a été pensée pour pallier aux limitations des architectures 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 trois tiers met en oeuvre une approche objet pour offrir une plus grande souplesse d'implémentation et faciliter la réutilisation des développements.

Théoriquement, ce type d'architecture supprime tous les inconvénients des architectures précédentes:

- Elle permet l'utilisation d'interfaces utilisateurs riches.
- Elle sépare nettement tous les niveaux de l'application.
- Elle offre de grandes capacités d'extension.
- Elle facilite la gestion des sessions.

VI.2. Que de niveaux :

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 (les trois niveaux d'une application informatique). En fait, l'architecture n-tiers qualifie la distribution d'application entre de multiples services et non la multiplication des niveaux de service.

Cette distribution est facilitée par l'utilisation de composants « métier », spécialisés et indépendants, introduits par les concepts orientés objets (langages de programmation et middleware). Elle permet de tirer pleinement partie de la notion de composants métiers réutilisables.

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

La distribution des services applicatifs facilite aussi l'intégration de traitements existants dans les nouvelles applications. On peut ainsi envisager de connecter un programme de prise de commande existant sur le site central de l'entreprise à une application distribuée en utilisant un middleware adapté [8].

VII. Rôle de l'approche objet :

VII.2.1. Approche objet :

Les évolutions successives de l'informatique permettent de masquer la complexité des mécanismes mis en œuvre derrière une approche de plus en plus conceptuelle.

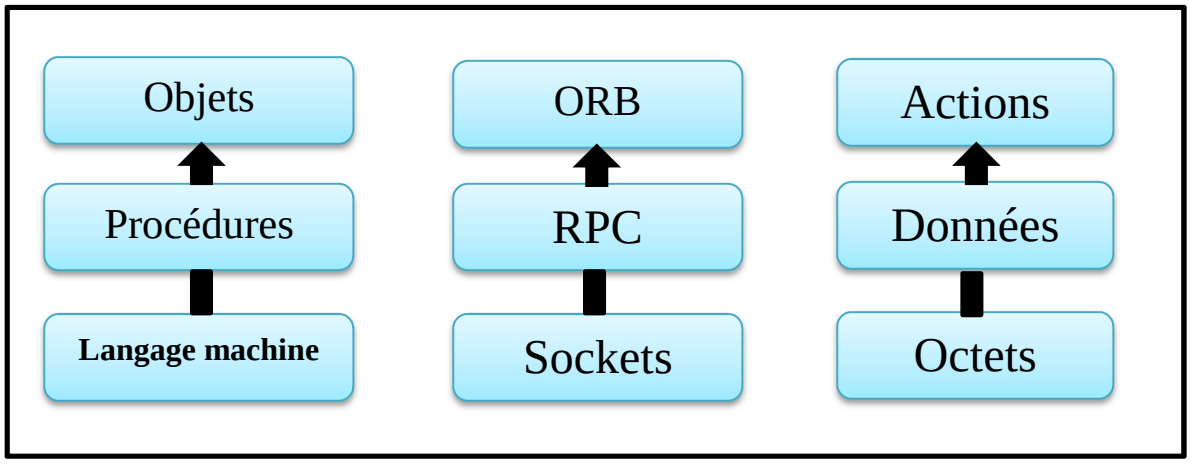

Figure II.10 : Quelque évolution de l'informatique.

Ainsi, les langages de programmation ont tout d'abord été très proches de la machine (langage machine de bas niveau), puis procéduraux et, enfin, orientés objets. Le succès du langage Java a véritablement popularisé ce mode de programmation.

Les protocoles réseau ont suivi le même type d'évolution. Ils furent d'abord très proches de la couche physique, avec les mécanismes de sockets orientés octets. Ensuite, la notion de RPC a permis de faire abstraction des protocoles de communication et, ainsi, a facilité la mise en place d'application client-serveur. Aujourd'hui, l'utilisation d'ORB permet une totale transparence des appels distants et permet de manipuler un objet distant comme s'il était local. Le flux d'informations fut donc initialement constitué d'octets, puis de données et, enfin, de messages.

Les méthodes de conception orientées objet telles qu'UML ou OMT permettent une modélisation plus concrète des besoins et facilitent le passage de la conception à la réalisation.

Aucune de ces évolutions ne constitue en soit une révolution, mais elles rendent économiquement réalisables des architectures qui n'étaient jusqu'à présent que techniquement envisageables [8].

VII.2.2. Objets métier :

a. Java Beans :

Selon les spécifications de Sun, un Java Beans est un composant logiciel réutilisable qui peut être manipulé par un outil d'assemblage. Cette notion assez large englobe aussi bien un simple bouton qu'une application complète. Concrètement, les Java Beans sont des classes Java utilisant des interfaces particulières.

Un Java Beans peut être utilisé indépendamment, sous la forme d'une simple applet, ou intégré à un développement Java. Il peut dévoiler son comportement à ses futurs utilisateurs à l'aide :

- Des propriétés qu'il expose et rend accessible à l'aide d'accesseurs.
- Des méthodes qu'il permet d'invoquer, comme tout objet Java.
- Des événements qu'il peut générer pour avertir d'autres composants.

Le mécanisme d'introspection permet une analyse automatique du composant qui, ainsi, s'auto-décrit.

Les Java Beans proposent deux modes de fonctionnement :

- Le mode de configuration permettant de paramètrer l'intégration du composant à l'aide d'un outil d'assemblage. Dans ce mode, le Java Beans propose un interface graphique de configuration.
- Le mode d'exécution utilisé par l'application intégrant le Java Beans.

Les Java Beans sont gérés comme tout objet Java, notamment ce qui concerne son cycle de vie. Ils sont livrés sous forme de fichiers JAR.

b. EJB (Entreprise Java Beans) :

Les Enterprise Java Beans sont des Java Beans destinés à s'exécuter côté serveur :

- Ils ne comportent pas forcément de partie visible.
- Ils prennent en charge des fonctions de sécurité, de gestion des transactions et d'état.
- Ils peuvent communiquer avec des Java Beans côté client de façon indépendante du protocole (IIOP, RMI, DCOM).
- Ils s'exécutent dans un environnement « Beanstalk » s'appuyant sur l'API Java Server et offrant des fonctionnalités de gestion de la charge d'exécution, de la sécurité d'accès, de communication, d'accès aux services transactionnels.

c. Microsoft OLE-COM-ActiveX :

Le modèle de communication COM permet de mettre en place une communication orientée objet entre des applications s'exécutant sur une même machine.

DCOM est une extension de ce modèle pour les architectures distribuées. Il repose sur le modèle DCE défini par l'OSF et met en oeuvre un serveur d'objets situé sur chaquemachine.

Les contrôles ActiveX, anciennement dénommés OCX, sont des composants logiciels basés sur le modèles COM. Ils peuvent être intégrés à des applications où à des documents sous Windows [8].

Avec les applications n-tiers, on dispose enfin d'une vision cohérente du système d'information. Tous les services sont représentés sous la forme d'objets interchangeables qu'il est possible d'implanter librement, en fonction des besoins.

Le potentiel de ce type d'application est très important et permettrait enfin l'utilisation d'objets métiers réutilisables.

Serions-nous en présence de l'architecture parfaite, nous permettant de mettre en œuvre les applications les plus ambitieuses au moindre coût et de les faire évoluer librement en fonction des besoins ?

Si, sur le papier, on pourrait répondre par l'affirmative, l'expérience des précédentes « révolutions » de l'informatique nous pousse à plus de modération. L'architecture n-tiers propose effectivement un potentiel énorme, reste à voir comment il sera utilisé, à quel coût et, surtout, comment sera gérée la transition depuis les environnements actuels.

Dans le prochain chapitre on va voir une présentation du e-learning et une comparaison de quatre plateformes d’e-learning et à la fin on verra une présentation de plateforme élaborer pendant ceux mémoire.

VIII. Conclusion :

Dans le chapitre suivant, on présentera notre contribution dans le cadre de ce PFE à savoir le choix du système distribué implémenté qui est la plateforme e-learning.

Chapitre III :

Présentation de l'application réalisée

I. Introduction :

Après une période d'enthousiasme où l'e-learning a été perçu comme un moyen de résoudre les problèmes de formation (logistiques et budgétaires) en remplaçant les formations traditionnelles en présentiel, nous pouvons dire qu'aujourd'hui « l'e-learning atteint l'âge de raison ». Dans ce chapitre on présentera notre contribution dans le cadre de ce PFE. Pour ce fait, nous avons réalisé une synthèse sur quelque plateforme e-learning, nous allons voir les avantages et inconvénients de chaque implémentation afin de mettre en œuvre notre plateforme e-learning.

II. Définition de l'e-learning :

L'application des technologies de l'information et de la communication au domaine de la formation a conduit à la création de l'e-learning. Sommairement décrit comme le mariage du multimédia (le son, l'image, le texte) et de l'internet (la diffusion on line, l'interactivité) l'e-learning apparaît pour beaucoup comme le second souffle du marché de ces technologies.

L'e-learning a, suivant les différents acteurs, plusieurs définitions. La plus courante est la suivante : « amener la formation à des apprenants avec l'aide des nouvelles technologies, c'est-à-dire à chaque fois que l'on utilise les médias interactifs pour la formation (intranet, internet, CD-ROM)».

Par ailleurs, l'e-learning peut être considéré comme le moyen donné pour des apprenants géographiquement dispersés d'accéder à des matériels pédagogiques, des tuteurs, ainsi qu'à leur parcours de formation à l'endroit souhaité et à leur convenance.

Pour la formation traditionnelle, e-learning signifie formation à distance : celle-ci est organisée par les acteurs. Elle est personnalisée et flexible, permettant aux acteurs un apprentissage complémentaire et individualisé se libérant des contraintes de lieu et de ressources humaines.

En réalité, l'e-learning désigne tout dispositif de formation utilisant internet comme canal de diffusion. C'est l'acte pédagogique qui se vit pour tout ou parti en ligne.

De l'autoformation - mode d'apprentissage individuel qui permet à l'apprenant de se former à son rythme en utilisant des ressources créées à cet effet, à la classe

virtuelle synchrone - dispositif de formation à distance durant laquelle l'apprenant est en contact simultané avec son formateur et/ou les membres de sa classe virtuelle et peut échanger avec eux au moyen de chat, de tableau blanc partagé, etc. , ses formes sont très variées.

L'e-formation, de manière plus large, désigne tout système de formation reposant globalement sur l'usage des technologies issues de l'internet. C'est le processus de formation dans son ensemble (et pas simplement l'action de former) qui est repensé par l'usage des technologies internet.

Grâce aux multiples outils disponibles du monde internet, une grande variété de méthodes pédagogiques et de modalités de travail sont possibles.

L'e-learning intègre les qualités de :

- La formation à distance de masse utilisant principalement l'écrit et parfois l'audiovisuel.
- La formation individualisée dans des centres d'autoformation ou sur le poste de travail.
- Les classes virtuelles reliées grâce à la vidéotransmission ou à la visioconférence [10].

III.Les plateformes sélectionnées :

III.1. Claroline 1.8.6 :

Claroline est une plateforme de formation à distance et de travail collaboratif développée en 2002 par l'université de Louvain en Belgique. Elle permet aux formateurs de créer des espaces de cours en ligne et de gérer des activités de formation sur Internet. Traduite en 35 langues, Claroline bénéficie de l'appui d'une communauté mondiale d'utilisateurs et de développeurs.

Utilisée par des centaines d'institutions issues de 84 pays, elle permet de créer sans coût de licence des espaces de travail et des cours en ligne. Pour chaque cours, le formateur dispose d'une série d'outils lui permettant de :

- Rédiger une description du cours.
- Publier des documents dans tous les formats nécessaires (texte, PDF, HTML, vidéo...).

- Administrer des forums de discussion publics ou privés.
- Élaborer des parcours pédagogiques au standard SCORM 1.2 ou composés de documents.
- Créer des groupes de participants ayant des documents en commun et des forums privés.
- Composer des exercices (QCM).
- Structurer un agenda avec des tâches et des échéances.
- Publier des annonces (envoyées aussi par messagerie électronique).
- Proposer des travaux à rendre en ligne.
- Consulter les statistiques de fréquentation et de réussite aux exercices.
- Utiliser le Wiki pour rédiger des documents collaboratifs.

Adaptable à différents contextes de formation, Claroline est utilisée non seulement dans les écoles et les universités, mais également dans les centres de formation, les associations et les entreprises. Elle est personnalisable et offre un environnement de travail flexible et sur mesure.

Extrêmement facile d'utilisation tant du côté étudiant que du côté enseignant, elle se caractérise par une prise en main rapide et très intuitive.

Claroline a été développée sur base de l'expertise pédagogique des professeurs et en fonction de leurs besoins. Elle offre une gestion sobre et intuitive des outils et des espaces d'administration.

La gestion quotidienne de la plateforme ne requiert aucune compétence technique particulière. La plateforme s'installe aisément et l'usage d'un navigateur Internet permet de gérer les différents espaces ainsi que les utilisateurs enregistrés [9].

III.1.1. Points faibles de la plateforme :

- Groupes : le dossier documents et liens du groupe a peu de capacité de stockage (7 Mo), un étudiant peut s'inscrire dans un groupe, mais pas se désinscrire.
- « Chat » ou messagerie instantanée : l'outil n'est pas très ergonomique, mais on peut maintenant grâce aux modules changer l'outil et mettre celui de son choix.
- Les espaces de travail restent cloisonnés, un gestionnaire d'espace ne peut pas copier les informations qu'il a créées dans un autre espace (planning, annonces, pages HTML).

- Peu d'outils synchrones, seulement l'outil « Discussion », une fonctionnalité comme un tableau blanc ou un outil de visioconférence serait complémentaire.
- Interface ergonomique des parcours pédagogiques à améliorer... trop d'écrans pour atteindre le contenu. Si la ressource a beaucoup d'items, il n'y a pas de menu dynamique, cela fait beaucoup de « scrolls » et de clics pour suivre le parcours.
- Le SCORM n'est pas complètement interprété par la plateforme, certains packages produits par des chaînes éditoriales comme Scenari2 par exemple ne fonctionnent pas sous Claroline [9].

III.2. Ganesha 3.2.2 :

Ganesha est une plateforme de téléformation ou LMS (Learning Management System) créée et éditée par la société de formation spécialisée en e-learning : ANEMA.

Cette plateforme permet à un formateur ou un service de formation, dans le cadre d'une formation à distance ou pour enrichir le présentiel, de mettre à la disposition d'un ou plusieurs groupes de stagiaires, un ou plusieurs modules de formation avec supports de cours, compléments, quiz et tests d'évaluation, ainsi que des outils collaboratifs.

Ses autres fonctionnalités sont :

- Une messagerie interne à la plateforme donne la possibilité d'envoyer des pièces jointes, cela permet aux apprenants de ne pas avoir de messagerie personnelle.
- Un forum permet aux stagiaires et aux tuteurs de poster des messages qui seront accessibles à l'ensemble des membres du groupe de formation, de répondre aux messages déjà postés et ainsi engager une discussion sur un sujet donné.
- Un chat (ou messagerie instantanée) permet à l'ensemble des membres du groupe de discuter en temps réel.
- Une zone de dépôt de documents pédagogiques permettant de proposer des documents sous format numérique à l'ensemble du groupe, et de laisser des commentaires sur les documents postés.
- Un quizeur Flash afin de réaliser des questionnaires à choix multiples ou simples QCM/QCU à partir de la plateforme.

En utilisant les spécifications techniques de l'e-learning comme le SCORM 1.2 ou 2004, pour produire les modules, le suivi des utilisateurs (tracking) se fera automatiquement. Les modules « sur étagère » venant d'éditeurs, doivent respecter ces standards e-learning pour s'insérer facilement dans Ganesha.

- Le tableau de bord permet à un stagiaire d'accéder à ses modules de formation, d'avoir une synthèse de nombreuses informations :
  - **Individuelles:** son état d'avancement dans les modules, ses statistiques de connexion, les nouveaux messages sur sa messagerie et l'accès à sa fiche d'inscription.
  - **Collectives:** derniers messages postés sur le forum, état d'avancement de l'ensemble du groupe, CV des tuteurs et le planning des présentiels (regroupement physique des stagiaires et des tuteurs).

Plusieurs profils spécifiques sont possibles sur la plateforme :

- Stagiaire ou apprenant qui suivent la formation en consultant les modules de formation et en participant aux activités pédagogiques en ligne.
- Demandeur : peu réaliser un parcours en analyse des besoins et y affecter des stagiaires.
- Tuteur en entreprise : peut être rattaché à plusieurs apprenants pour suivre leur besoin.
- Conseiller : peuvent affecter un parcours en analyse des besoins à une session (groupe de stagiaires).
- Formateur : responsable de formation, intégrateur ou conseiller anime les sessions en ligne via les outils collaboratifs.
- Intégrateur : peu intégrer des ressources, des parcours ou des modules sur la plateforme.
- Administrateur qui gère la plateforme. Plusieurs administrateurs peuvent assurer la gestion.

Cette gestion des profils permet de créer :

- Des comptes pour les stagiaires, les tuteurs et les auteurs des modules.
- Les sessions et les groupes de stagiaires.
- Les forums, d'identifier les modules de formation, les espaces de dépôt de documents.

L'administrateur peut individualiser le parcours de formation en attribuant spécifiquement un ou plusieurs modules à un stagiaire et en attribuant un ou plusieurs tuteurs à un groupe de stagiaires.

L'administration de la plateforme s'effectue via une interface Web très simple d'utilisation.

Il est possible de s'inscrire en ligne sur Ganesha. Chaque stagiaire peut choisir son interface graphique et sa langue de consultation.

III.2.1. Points faibles de la plateforme :

- Les formateurs-tuteurs ne peuvent pas paramétrer leur cours de manière indépendante, seul l'administrateur de la plateforme peut organiser les formations en ligne contrairement à beaucoup d'autres plateformes. C'est un choix organisationnel à prendre en compte.
- Messagerie interne limitée (envoi d'un seul fichier à la fois, mais possibilité d'envoyer un fichier compressé de plusieurs).
- Un forum par groupe permettant de poster des messages accessibles à l'ensemble des membres du groupe de formation et d'y répondre, mais aux fonctionnalités réduites.
- Le Chat est peu ergonomique et n'a pas de traces.
- Wiki, blog, vidéoconférence non implémentée.
- Planning, agenda, non implémenté (accès à un planning créé à l'aide d'outils externes).
- Authentification : mots de passe des utilisateurs non cryptés.
- Connexion LDAP non fonctionnelle.

III.3. Moodle 1.8.2 :

Moodle est une plateforme d'apprentissage en ligne servant à créer des communautés d'apprenants autour de contenus et d'activités pédagogiques. Le terme Moodle est l'acronyme de Modular Object-Oriented Dynamic Learning Environment, mais il veut aussi dire « flâner » en anglais.

Dotée d'un système de gestion de contenu - (SGC) performante, Moodle a aussi des fonctions pédagogiques ou communicatives lui permettant de créer un environnement d'apprentissage en ligne. Ces fonctionnalités permettent de créer des interactions entre pédagogues, apprenants et ressources pédagogiques formant ainsi un réseau et parfois même une véritable communauté autour d'un thème choisi par les membres de la plateforme (apprentissage d'un logiciel comme utilisation de la plateforme). C'est une particularité que l'on retrouve peu dans les autres plateformes étudiées.

Moodle fait partie des systèmes d'E. formation qui sont appelés dispositifs de «Formation ouverte et à Distance » (FOAD), pour favoriser un cadre de formation socioconstructiviste. Ce courant de pensée affirme que les gens construisent activement leurs nouvelles connaissances en interagissant avec leur voisinage. Tout ce que vous lisez, voyez, entendez, ressentez et touchez est comparé à vos connaissances antérieures et si cela est viable dans votre monde mental, cela pourra former une nouvelle connaissance qui vous appartiendra. La connaissance est renforcée si vous pouvez l'utiliser avec succès dans un environnement plus large. Cela met l'accent sur le fait qu'il n'y a pas seulement un transfert d'information d'un cerveau à un autre, mais que tout est conditionné par l'interprétation et l'interaction avec un groupe social.

L'interface de Moodle se présente sous forme de différents blocs : le bloc central présente les documents et les activités du cours. Ils peuvent être classés selon plusieurs formats :

- **Thématique** : en fonction de thèmes ou de sujets du cours.
- **Hebdomadaire** : en fonction d'un agenda ou du calendrier.
- **Informel** : en fonction de sujets de discussion et de forums.

Pour présenter les ressources l'enseignant peut composer une page de texte : texte court sans mise en forme, créer une page web : page au format HTML qui peut être éditée à l'aide de l'éditeur intégré à la plateforme, mettre un lien vers un site Internet

ou un fichier déposé sur le serveur, afficher le contenu d'un dossier : lien vers une liste de fichiers déposés sur le serveur, ajouter un fichier IMS Content Package : lien vers un fichier de format SCORM ou AICC, et insérer une étiquette permettant de commenter la ressource.

Les blocs latéraux affichés sur les pages web donnent accès aux différents outils et liens du cours, par exemple :

- **Personnes** : liste des inscrits au cours + la liste des différents sous-groupes + accès à son profil.
- **Cours** : la liste des cours auxquels est inscrit l'usager.
- **Recherche** : outil de recherche dans les forums du cours.
- **Administration** : relevé des notes de l'usager…
- **Dernières nouvelles** : les dernières brèves publiées sur le forum.
- **Prochains événements** : les activités inscrites au calendrier de son cours.
- **Calendrier** : les activités classées en fonction du calendrier.
- **Utilisateurs en ligne** : la liste des personnes, enseignants et usagers, connectés au cours.
- Fils RSS …

Les membres d'un cours ont accès aux activités suivantes si l'enseignant les a sélectionnées :

- **Messagerie électronique** : "chat" ou salon de discussion (possibilité de l'ouvrir certain jour, à heure précise, de manière hebdomadaire, etc.).
- **Forum** : différents types de forums (sujets imposés par l'enseignant, sujets proposés par les étudiants, évaluation ou commentaire possibles, etc.).
- **Devoir** *:* remise de travaux avec évaluation de l'enseignant.
- **Test** *:* suite de QCM (Questionnaire à Choix multiples), de questions vraies/fausses, appariement, etc.
- **Leçon** *:* document comprenant des questions et plusieurs parcours possibles en fonction des réponses (évaluation possible).
- **Atelier** *:* remise de travaux avec évaluation par les étudiants.
- **Glossaire** *:* production collective d'un document organisé alphabétiquement (commentaire et évaluation possible).
- **Wiki** *:* production collective d'un document hypertexte (commentaires possibles de l'enseignant).

- **Journal** *:* rédaction d'un journal personnel (commentaires possibles).
- **Dialogues** *:* messagerie interne entre membres du cours.

Toutes les activités sont paramétrables par l'enseignant.

L'enseignant peut diviser son groupe classe en plusieurs sous-groupes de manière à faciliter la communication entre les personnes. Les groupes ont des outils dédiés : forum, sondage, chat.

Moodle a été créée de manière modulaire : elle permet de répondre aux besoins d'un formateur isolé comme d'une institution académique. Aujourd'hui, le développement de Moodle est fortement influencé par les demandes de la communauté d'administrateurs et d'utilisateurs (enseignants, pédagogues). Ce projet bénéficie d'un développement très actif à l'échelle mondiale.

III.3.1. Points faibles de la plateforme :

- Possibilité de test de positionnement, mais gestion manuelle, pas d'affectation automatique de parcours.
- Agenda du cours (visualisation des nouveautés, prévision d'événements pour tout type d'utilisateur, les travaux à rendre apparaissent automatiquement dans l'agenda), mais pas d'agenda personnel.
- Communication en mode synchrone : clavardage, mais pas de mode vidéoconférence sauf si l'on rajoute un module.
- Moodle est une plateforme très riche en fonctionnalités, sa prise en main par les apprenants peut nécessiter un temps d'adaptation, car les pages peuvent être très chargées en informations.
- Pour les enseignants, la diversité et la spécificité de tous les paramétrages des outils peuvent paraître trop complexes aux yeux d'utilisateurs peu familiers en FOAD. Cependant, de nombreux tutoriels en ligne existent, il est aussi important que l'administrateur de la plateforme ou le coordinateur du projet puisse être disponible pour aider les enseignants à appréhender la totalité des fonctionnalités de Moodle. Afin d'éviter les échecs, l'un des premiers cours à installer sur la plateforme est celui sur l'utilisation de Moodle.

III.4. Sakai 2.4.0 :

Le projet Sakai a été lancé en 2004 par quatre universités américaines avec pour objectif de consolider leurs développements en matière de plateforme d'apprentissage. Chacune de ces universités, soit Indiana University, Massachusetts Institute of Technology (MIT), Stanford University et University of Michigan utilisait des systèmes de gestion de cours différents, souvent développés en interne. Au groupe se sont joints des membres de uPortal2 ainsi que du projet OKI (Open Knowledge Initiative).

Sakai est un projet, auquel sont rattachés une fondation, un comité de direction et des partenaires institutionnels et commerciaux ; une communauté qui regroupe plusieurs institutions qui coopèrent et maintiennent Sakai. Elle regroupe actuellement plus de 80 universités dans plusieurs pays dans le monde.

Sakai est une plateforme utilisable dans un contexte d'éducation et de formation basée sur une structure ouverte et extensible, incluant une suite d'outils pour le support de l'apprentissage, de la collaboration et de la recherche.

L'objectif de Sakai est de produire une plateforme complète de gestion de cours libre de qualité équivalente ou supérieure aux autres produits du marché. Le produit s'adresse, dans un premier temps, aux institutions universitaires. Ce sont ces dernières qui définissent les orientations, les spécifications et les priorités quant au développement du projet. La structure ainsi que les outils développés sont génériques et peuvent être réutilisés comme base pour la construction d'outils de collaboration ou d'apprentissage dans n'importe quel domaine.

Sakai contient les outils suivants :

- Annonces.
- Casier de documents.
- Messagerie avec archivage.
- Ressources.
- Salle de clavardage (chat).
- Forums.
- Fil de discussion.
- Centre de message et message du jour.
- Nouveautés et RSS.

- Préférences.
- Présentation avec Syllabus.
- Gestion des profils.
- Recherche dans un entrepôt de documents.
- Emploi du temps.
- Site web gestion de fichiers avec WebDAV.
- Wiki.
- Paramétrage du site.
- Parcours pédagogiques avec Melete.

III.4.1. Points faibles de la plateforme :

- Sakai est une plateforme en pleine mutation, mais qui évolue rapidement... Elle manque encore d'interopérabilité :
  - Les importations de modules IMS 1.1.2 ne sont pas toujours opérationnelles, cela dépend des logiciels auteur.
  - Le SCORM n'est pas encore implémenté.
- Sa documentation : Sakaïpedia est aujourd'hui, exclusivement en anglais, ce qui peut gêner les utilisateurs qui ne parlent pas cette langue. Cependant beaucoup d'institutions au Québec s'intéressent au projet Sakai, ils ont fondé une association à but non lucratif : Sakai Québec et sont en train de développer une traduction en français et en arabe pour les pays maghrébins du projet POLLES.
- La plateforme est développée en Java, ce qui demande un administrateur compétent dans ce langage, l'installation est complexe, l'outil administration n'est pas encore traduit.
- Beaucoup de projets sont en cours de développement pour la plateforme (vidéoconférence, traduction en français, portfolio, intégration du SCORM, de CanCore1, de NORMETIC2...) [9].

IV.Grilles d'évaluation des points-clés :

IV.1. Les 10 points clés :

1. Importante communauté d'utilisateurs et de développeurs, dynamique et d'envergure internationale.

2. Documentation en ligne de l'installation de la plateforme à l'utilisation par les enseignants et les apprenants.

3. Plateforme pouvant gérer un grand nombre d'utilisateurs.

4. Outils collaboratifs dédiés aux échanges autour d'apprentissages communs.

5. Adaptabilité et modularité de la plateforme.

6. Intégration de spécifications techniques et de standards comme l'AICC/SCORM

(Aviation Industry CBT Committee/Sharable Content Object Reference Model), le LOM (Learning Object Metadata) et éventuellement IMS-LD (Instructionnal Management Systems – Learning Design).

7. Installation et gestion de la plateforme simple basée sur des technologies Web courantes.

8. Adaptation possible de la charte graphique.

9. Multisystèmes d'exploitation : côté serveur (quels systèmes sont supportés) et côté client (quels navigateurs ? besoin de plug-ins particuliers ?).

10. Ergonomie, utilisabilité des plateformes pour les enseignants comme pour les apprenants.

Dans le tableau suivant, des « smileys » illustrent les comparaisons :

 : Évaluation positive, la plateforme répond au critère.

 : Point de vigilance, il faut mettre en place une solution pour pallier ce défaut.

 : Évaluation négative, problème majeur, la plateforme ne répond pas au critère.

	Claroline	Genesha	Moodle	Sakai
1. Communauté, dynamisme, international				
2. Documentation				
3. Gestion du nombre d'utilisateurs				
4. Outils collaboratifs				
5. Adaptabilité et modularité				
6. Intégration de spécifications et standards				
7. Installation et gestion				
8. Adaptation de la charte graphique				
9. Systèmes d'exploitation; navigateurs, plug-ins clients				
10. Ergonomie et utilisabilité				

Résultats			
Plateformes			
Claroline	8	0	2
Genesha	3	0	7
Moodle	9	1	0
Sakai	7	0	3

Table III.1 : Tableau comparatif des quatre plateformes choisies.

IV. Présentation de de notre plateforme :

En fonctions de la synthèse précédente, nous allons essayer de mettre en œuvre notre propre solution. Cette dernière consiste à l'implémentation d'une plateforme e-learning destinée aux personnes qui veullent apprendre un langage de programmation. Nous avons choisi le langage JAVA comme exemple de formation et nous avons opté pour une architecture 2-tiers pour la mise en œuvre de la solution proposée.

Pour cette solution nous avons utilisé un ensemble d'outils à savoir :JSP, HTML, CSS, MySQL et l'IDE Netbeans.

Dans ce qui suit, nous allons présenter un scénario d'utilisations de notre plateforme illustré par des prises d'écran.

La Figure III.1 montre le formulaire d'identification utilisé par un étudiant déjà inscrit, les informations relatives à cet étudiant sont stocké dans une Base De Donnée MySQL.

Figure III.1 : Formulaire d'identification.

La Figure III.2 montre le formulaire d'inscription.

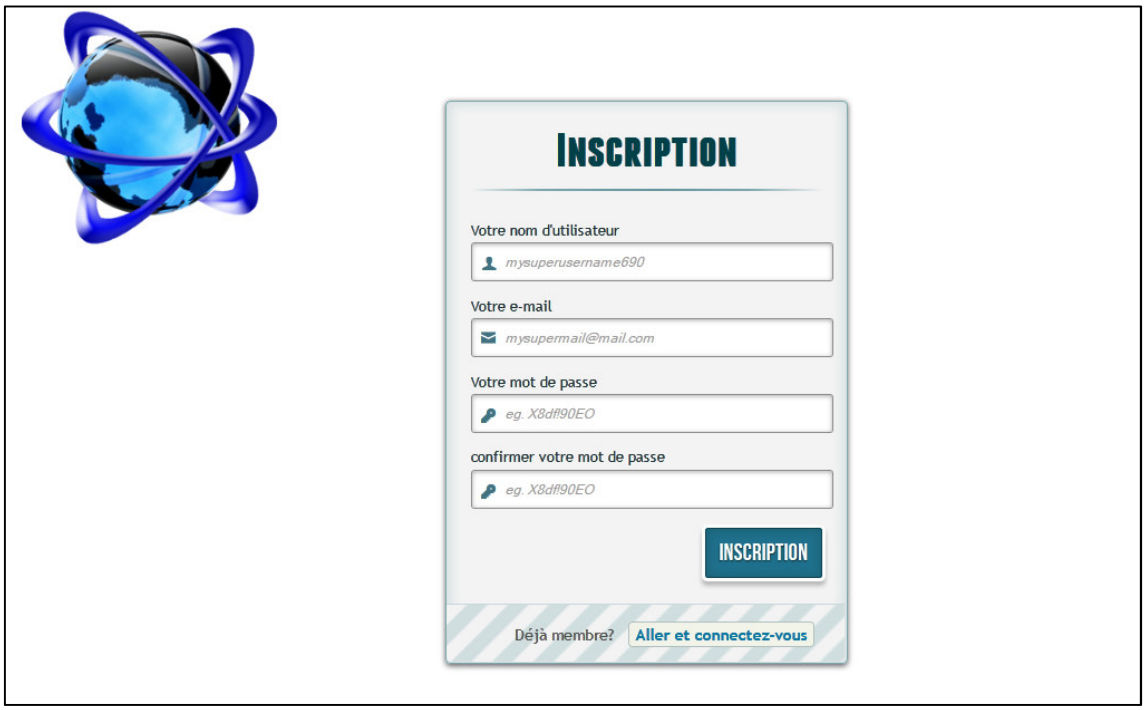

Figure III.2 : Formulaire d'inscription.

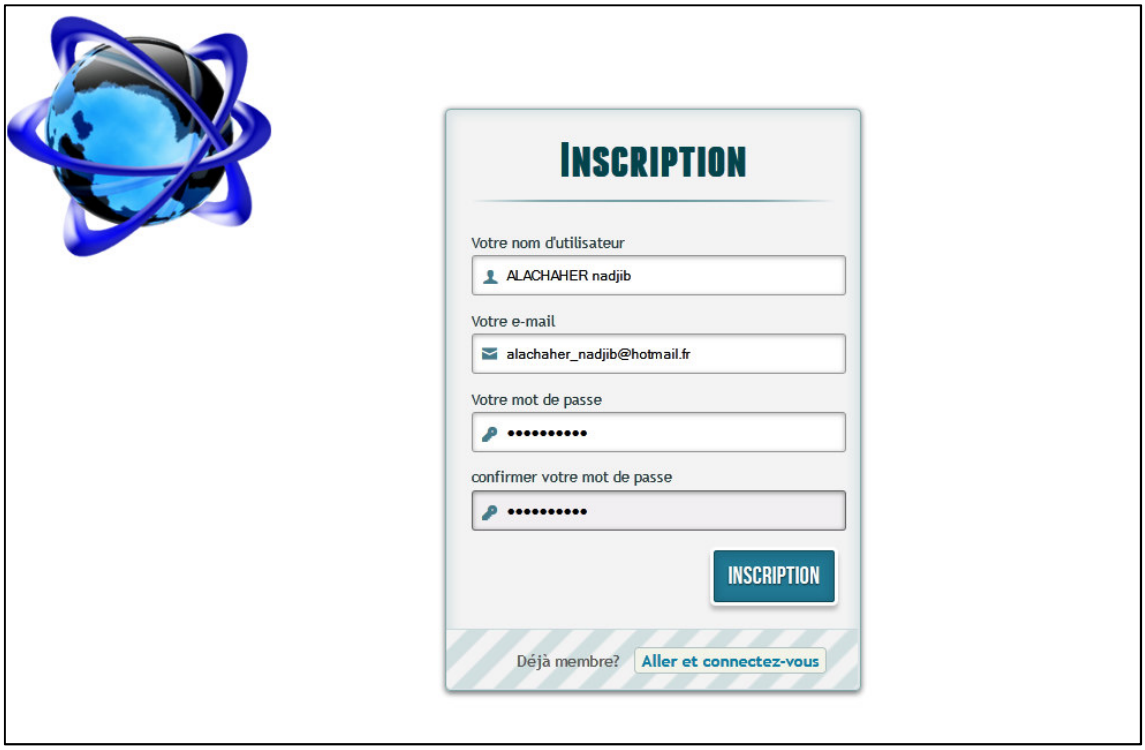

Figure III.3 : Remplir le formulaire d'inscription.

Après authentification, un étudiant peut accéder aux différent cours et QCM proposés par la plateforme, les cours sont stockés sous format PDF

Figure III.4 : Cours1 master 2.

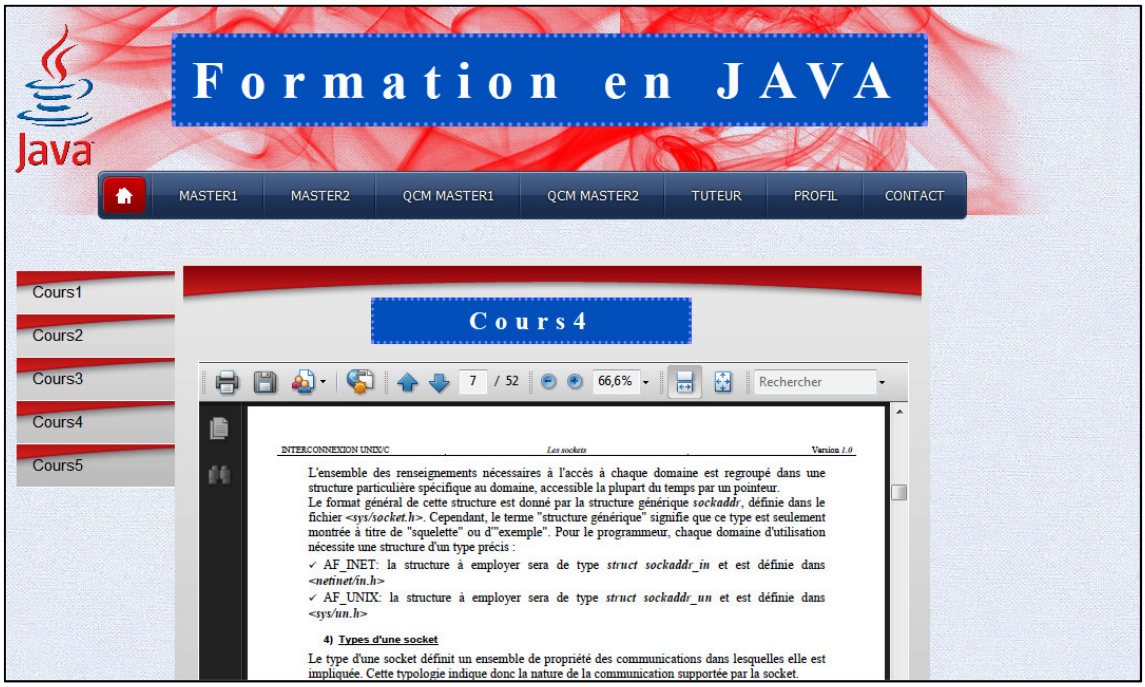

Figure III.5 : Cours4 master 2.

La Figure III.6 montre un exemple de QCM proposé pour les étudiants en Master 1

Figure III.6 : QCM Master1.

Figure III.7 : Réponse de l'étudiant.

La Figure III.8 est après la validation du QCM montre la note obtenue par l'étudiant mais également le corrigé du QCM, une réponse écrite en vert est juste et donne 2 points à l'étudiant, une réponse écrite en rouge est la bonne réponse à la question mais elle est différente de ce que l'étudiant à répondue.

Figure III.8 : Corriger du QCM.

Figure III.9 : Profil de l'étudiant.

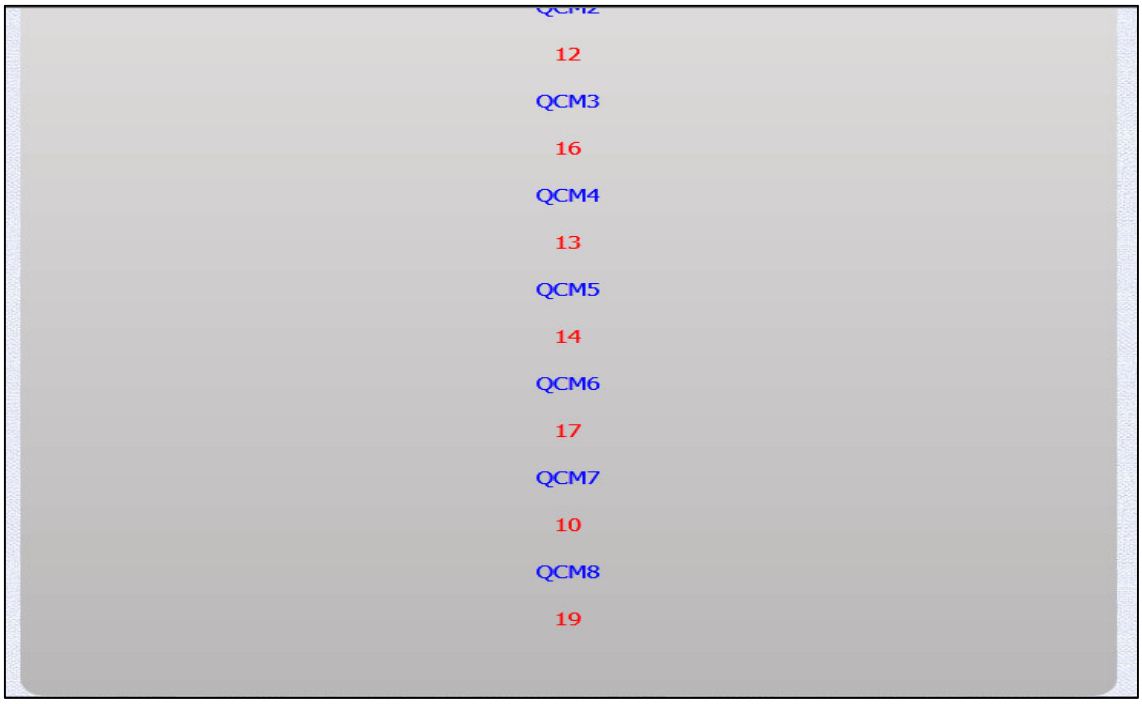

Figure III.10 : Note obtenue aux QCM.

L'étudiant peut consulter à tout moment son profil pour voir les notes obtenu dans les différent QCM et avoir une idée sur son état d'avancement par rapport à l'apprentissage du JAVA, toute ses informations sont stockées coté MySQL.

V. Conclusion :

La troisième partie présente notre contribution dans le cadre de ce PFE, à savoir le choix du système distribué implémenté qui est la plateforme e-learning. Nous avons fait une comparaison en présentant les quatre plateformes les plus connues. Cette comparaison a été la brique de base pour l’implémentation de notre plateforme.

Conclusion général

Le marché du e-learning est en pleine expansion. Aux Etats-Unis, le e-learning a envahi le marché de l'éducation. En Algérie, le e-learning est encore timide, mais il est promis à une forte croissance

Le e-learning prend son essor et bouscule les habitudes pédagogiques des enseignants qui deviennent des "intégrateurs de savoirs" mis à la disposition de tous via les réseaux et doivent gérer de nouvelles communautés virtuelles de connaissance.

Le e-learning ne trouve pas encore sa place dans le milieu de l'éducation, face aux formations plus traditionnelles. Cette peur du changement est due à la méconnaissance du e-learning ainsi que par un manque d'outils informatiques.

Le potentiel pédagogique et économique du e-learning n'est pas remis en cause. Néanmoins, le rythme de développement du marché sera beaucoup plus lent que prévu. Par ailleurs, le e-learning devrait davantage être un complément des dispositifs de formation traditionnels qu'un substitut. Les conditions de développement sont les suivantes :

- Education réaliste du marché, vendre le e-learning crée des attentes non satisfaites et risque de compromettre son avenir.
- Simplification de l'offre, pour en améliorer la visibilité.
- Standardisation, pour diminuer les prix et faciliter les arbitrages budgétaires des entreprises.
- Adaptation des dispositifs d'évaluation pour mesurer l'efficacité des dispositifs.
- Enrichissement des supports, ce qui dépend de la généralisation de l'accès à Internet à haut débit.

Le e-learning doit donc parvenir à convaincre de son rapport efficacité/prix et conquérir une véritable légitimité, ce qui soulève la question de la reconnaissance professionnelle des diplômes en ligne.

Dans ce PFE, nous avons implémenté notre propre plateforme e-learning, toute fois beaucoup de travail reste à faire notamment l'ajout de fonctionnalité supplémentaire, l'ajout de formation supplémentaire ainsi que la gestion de pannes coté serveur.

Références Bibliographiques

[1] Yamine AIT AMEUR, Françoise SIMONOT-LION, Association cohérente de données dans les systèmes temps réel à base de composants - Application aux logiciels spatiaux, thèse de doctorat, université de Toulouse, 2009.

[2] Rachid GUERRAOUI, Esther PACITTI, Routage des Transactions dans les Bases de Données à Large Echelle, thèse de doctorat, l'Université Pierre et Marie Curie (Paris VI), 2010.

[3] Jean-Pierre ELLOY, Bernard ESPIAU, Sur l'optimisation des systèmes ditribués réel embarqués, l'université de ROUEN, 2000.

[4] Frédéric Lang, INTERCONNEXION UNIX/C, socket.

[5] Frédéric Desprez, Bruno Raffin, Analyse, conception et réalisation d'un environnement pour le pilotage et la visualisation en ligne de simulations numériques parallèles, thèse de doctorat, L'UNIVERSITÉ DE BORDEAUX I, 2005.

[6] Rémi LEBLOND, Vers une architecture n-tiers, Oral probatoire, http://remi.leblond.free.fr/probatoire/probatoire.html , 2003.

[8] Badr FERRASSI, LES SERVEURS DE DONNEES L'ARCHITECTURE DEUX TIERS, cours, http://www.ista.ma, 2012.

[9] Dossou Anani Koffi DOGBE-SEMANOU, Anne Durand, Marie LEPROUST, Hélène VANDERSTICHEL, Etude comparative de plates-formes de formation à distance, cours, 2008.

[10] GOTTRAND Séverine, QUEANT Vanessa, Le e-learning comme innovation en Ressources Humaines, Article, UNIVERSITE DE LILLE 1, 2012.

[11] Leslie Lamport. Time, clocks and the ordering of events in a distributed system. Communications of the ACM, 21(7) :558–565, jul 1978.

Résumé

L’objectif de notre PFE est d’implémenter un système distribué en utilisant une architecture N-tiers. Vu l’intérêt grandissant du e-learning, nous avons opté pour la réalisation d’une plateforme e-learning afin d’aider les étudiants qui souhaitent à apprendre à distance des nouveaux langages de programmation.

Our objective is to implement a PFE distributed system using N-tier architecture. Given the growing interest in e-learning, we opted for the realization of e-learning platform to help students who wish to learn at a distance of new programming languages.

هدفنا هو تطبيق نظام توزيع باستخدام نبنية الطبقة. نظرا للاهتمام المتزايد في مجال التعليم الإلكتروني، اخترنا لتحقيق التعليم الإلكتروني منصة لمساعدة الطلاب الذين يرغبون في تعلم على مسافة من لغات البرمجة الجديدة.