CONCEPTION DES LOGICIELS

TOME 2: LES DÉMARCHES DE CONCEPTION

Auteur: Bernard GIACOMONI

Version	Date	Objet
1.0	18/11/2021	Version initiale

Table des matières

I.INTRODUCTION:

I.1.OBJECTIF VISÉ:

L'objectif de l'ouvrage (dont le document présent constitue le deuxième tome) n'est en aucun cas d'initier le lecteur à une méthode de conception ou à un formalisme particulier, mais plutôt de le familiariser à la problématique générale et aux enjeux de l'activité de conception des logiciels.

Dans ce cadre, les différents aspects de cette activité et les concepts et techniques qui lui sont rattachés sont présentés et expliqués en s'efforçant de faire ressortir les similitudes et différences entre les différentes méthodes et outils afin d'en tirer des enseignements généraux.

I.2.LES BESOINS DU DÉVELOPPEUR DE LOGICIELS:

I.2.1.CANALISER LA CRÉATIVITÉ SANS LA BRIDER:

La conception des logiciels, comme toutes les activités de conception, s'accorde assez mal avec des contraintes méthodologiques trop fortes: les méthodes doivent apporter un soutien à la créativité sans l'étouffer sous des formalismes trop lourds à gérer. L'important n'est pas d'appliquer les règles à la lettre, mais de comprendre pourquoi ces règles ont été instaurées, ce que leur respect peut apporter ou garantir et les danger qui peuvent surgir lorsque l'on s'en écarte trop.

I.2.2.S'ADAPTER A DES CADRES MÉTHODOLOGIQUES DIFFÉRENTS:

Un développeur de logiciel peut être amené à exercer dans différents contextes et dans différents cadres méthodologiques. Il doit donc pouvoir s'adapter à chacun de ces environnement avec un minimum d'efforts et dans un temps limité. De ce fait, la compréhension des différentes démarches (la MÉTHODOLOGIE de l'activité de conception des logiciels) lui est donc plus précieuse que la maîtrise de telle ou telle MÉTHODE ou formalisme particulier.

I.3.CONCEPTION ET MÉTHODES AGILES:

Certaines méthodes de développement actuelles ("méthodes agiles") contestent l'utilité d'une phase de conception effectuée EN PRÉLIMINAIRE aux activités de développement de chaque lot de réalisation, pour s'en remettre à une adaptation permanente de l'architecture au fur et à mesure de l'évolution des projets. Cette pratique n'aboutit pas la la disparition de l'activité de conception, mais plutôt à sa fragmentation en plusieurs itérations réparties tout au long du projet.

Dans ces contextes de travail, les compétences en matière de conception et d'architecture globale des logiciels sont d'autant plus précieuses qu'elles permettent d'éviter que ces différentes étapes d'adaptations n'aboutissent à des architectures incohérentes et faiblement performantes.

I.4. REMARQUES SUR LE CONTENU ET LE PLAN DE L'OUVRAGE:

- Ce document n'est pas un ouvrage d'initiation. Il s'adresse à des lecteurs ayant une expérience du développement des logiciels (au minimum, une pratique de la programmation) ainsi qu'une connaissance au moins basique de la structure et du fonctionnement des Unités de Traitement Informatiques. Une expérience de la conduite de projets informatiques est également bienvenue. Cependant, le chapitre II effectue de nombreux rappels qui peuvent aider à une "mise (ou remise) à niveau" dans ces domaines;
- Si des formalismes existants sont parfois utilisés dans cet ouvrage pour expliquer ou illustrer les différents concepts, ces formalismes peuvent être empruntés à des méthodes différentes ou même à aucune méthode existante: le but est plus de faire comprendre au lecteur les tenants et les aboutissants des différentes activités de conception que de l'inciter à utiliser un formalisme ou une méthode particuliers.

Pour diminuer sa complexité, le présent ouvrage a été divisé en trois tomes dont voici les contenus respectifs :

PREMIER TOME :

- Le chapitre I, qui est commun aux deux premiers tomes, s'attache à exposer les objectifs et le plan de l'ouvrage ;
- Le chapitre II est consacré à des rappels ou des "mises à niveau" concernant les notions et techniques de base de la spécification des besoins et de la conception des logiciels ;
- Le chapitre III présente les principaux outils et techniques spécifiques de l'activité de conception ;
- La suite est composée d'annexes permettant aux lecteurs d'acquérir ou de revoir des compétences indispensables à la compréhension des trois premiers chapitres. Le lecteur pourra s'y référer en cas de besoin :
  - Annexe I: Structure et fonctionnement d'un ordinateur ;
  - Annexe II : Traitements logiciels multitâches ;
  - Annexe III : Communication réseau entre processus distants ;
  - Annexe IV : Rappels sur la programmation objets ;

- Annexe V : Diagrammes de flux de données en analyse organique.

DEUXIÈME TOME :

Ce document est consacré à la présentation des démarches de conception en fonction des différents points de vue adoptés par le concepteur. Ces démarches sont appliquées à des exemples concrets. Sont ainsi abordées :

- Les démarches de conception au niveau du système informatique ;
- Les démarches de conception depuis les points de vue comportementaux et logiques ;
- La conception détaillée.
- Comme dans le cas du tome 1, la suite est composée d'annexes permettant aux lecteurs d'acquérir ou de revoir des compétences indispensables à la compréhension des premiers chapitres. Le lecteur pourra s'y référer en cas de besoin :
  - ANNEXE A - CONSOMMATION DE LA PUISSANCE CPU PAR UNE TÂCHE ;
  - ANNEXE B - CONSOMMATION DE LA MÉMOIRE VIVE PAR UNE TÂCHE ;
  - ANNEXE C - FIABILITÉ DES COMPOSANTS ÉLECTRONIQUES DIGITAUX ;
  - ANNEXE D - PROGRAMMATION EN ENVIRONNEMENT MULTITÂCHES ;
  - ANNEXE E - EXÉCUTABLES, PROCESSUS ET THREADS.

TROISIÈME TOME :

Ce dernier document illustre la démarche exposée par le tome II en l'appliquant à un cas concret.

II.AVANT PROPOS:

Comme les lecteurs du tome I de cet ouvrage ont pu le constater, pour que la démarche de création d'une application informatique aboutisse à la meilleure solution possible, il est indispensable d'examiner le problème suivant différents POINTS DE VUE.

Chaque point de vue permet d'étudier la réalisation suivant un aspect particulier. Le résultat de ces études est l'élaboration de MODÈLES concernant tout ou partie de cette application.

La phase de réalisation (codage, tests unitaires, intégration) consistera pour les développeurs à traduire en composants logiciels les entités identifiées et décrites par ces modèles et à s'assurer que le comportement de ces composants est bien conforme aux modèles établis.

Dans cet ouvrage, nous diviserons la phase de conception dite "préliminaire" ou "globale" (par opposition avec la conception dite "détaillée") en 5 étapes :

1. Une ÉTAPE PRÉLIMINAIRE de VALIDATION et COMPLÉTION de la S.T.B : cette étape a pour objectif de s'assurer que la S.T.B décrit tous les éléments qualitatifs (description des traitements et des interfaces) et quantitatifs (niveaux de performances et contraintes attendus) nécessaires à l'élaboration de l'application ;
2. L'ÉTAPE d'analyse COMPORTEMENTALE, qui a pour but d'élaborer le MODÈLE COMPORTEMENTAL de l'application, représentatif des règles et mécanismes qui permettront à l'application de respecter les contraintes d'ordre dynamique édictées par la S.T.B: concurrence des activités, temps de réponse, etc. Ce point de vue correspond sensiblement à la VUE DES PROCESSUS du langage de conception U.M.L ;
3. L'ÉTAPE de conception "SYSTÈME", dont le but est de proposer une solution pour l'architecture matérielle et les systèmes d'exploitation du SYSTÈME INFORMATIQUE d'accueil de l'application. Cette solution peut éventuellement se résumer à l'évolution du système existant ;
4. L'ÉTAPE de conception LOGIQUE, qui a pour but d'élaborer une architecture "statique" du logiciel compatible avec les contraintes fonctionnelles, opérationnelles et techniques affectant l'application. Cet axe d'étude correspond sensiblement à la VUE LOGIQUE du langage de conception U.M.L ;
5. L'ÉTAPE de SYNTHÈSE, dont l'objectif est de prendre en compte les résultats des trois étapes précédentes pour élaborer un "modèle d'exécution" de l'application :
    - Composants logiciels DYNAMIQUES (exécutables, processus et threads) ;
    - Composants STATIQUES (composants architecturaux actifs et passifs : classes, modules, etc.) ;

- Communication entre ces composants (échange de messages et de signaux) .

REMARQUE : il est évident que les options de conception prises lors de ces étapes peuvent présenter entre elles des incompatibilités. La démarche de conception générale peut donc présenter des ITÉRATIONS.

III.ÉTAPE 1 : VALIDATION ET COMPLÉTION DE LA S.T.B :

Dans un premier temps, il va falloir s'assurer que la spécification des besoins (ou le recueil des cas d'utilisation ou celui des scenarii clients, suivant la méthode) est suffisamment claire, précise et exhaustive pour permettre de définir le produit à développer. Ceci revient à répondre aux questions suivantes:

III.1.CONCERNANT LES UTILISATEURS DU SYSTÈME:

- Tous les types d'utilisateurs du systèmes sont-il identifiés?
- Les privilèges associés à chaque type d'utilisateurs (c'est à dire, les fonctions auxquelles ceux-ci ont accès) sont-ils bien définis ?
- Est-ce que les conditions d'exercice de ces utilisateur (localisation géographique, ambiance d'utilisation, etc.) sont suffisamment précisées (un utilisateur peut travailler en ambiance bruyante, à l'air libre, dans l'obscurité, depuis son domicile, etc. Ces conditions de travail influent sur la conception de son terminal de travail);
- Le nombre maximal de sessions d'utilisation simultanées et les délais de réponses admissibles sont-ils bien définis ?

III.2.CONCERNANT LES INTERFACES EXTERNES:

Il importe de s'assurer que toutes les interfaces externes sont bien définies (Description des IHM, format et volume des données échangées, protocoles d'échanges, contraintes, etc.).

III.3.CONCERNANT LES TRAITEMENTS INDUITS PAR LES EXIGENCES FONCTIONNELLES:

Il importe de s'assurer que l'expression de ces exigences est suffisamment claire et précise pour qu'on puisse préciser pour chacune d'entre elles :

- Les traitements et mécanismes mis en jeu ;
- Les données consommées (nature, format de représentation, précision des données, volume):
- Les données et services produits (nature, format de représentation, précision des données, etc.)
- Les contraintes associée (délais maximums de fourniture, déterminisme des dates de fourniture, etc.) .

La spécification des besoins doit être compréhensible par le client, car elle constitue une des clauses du contrat de fourniture qui va le lier au développeur. De ce fait, sa rédaction ne doit pas s'appuyer sur des formulations trop techniques. En conséquence, si, pour tel ou tel point, la formulation de la S.T.B semble trop imprécise au concepteur, celui-ci devra approfondir l'analyse en suivant une démarche plus rigoureuse.

Le langage U.M.L, en particulier, propose différents outils qui permettent de bien caractériser les mécanismes liés aux cas d'exécution. Les plus pertinents à ce niveau nous semblent être ;

- Les diagrammes d'état-transitions, qui permettent d'analyser le comportement d'une entité logicielle en l'assimilant à un automate d'états finis ;
- Les diagrammes d'activités, qui permettent d'analyser les ACTIVITÉS COMPLEXES impliquées par les cas d'utilisation ainsi que les interactions entre activités ;
- Les diagrammes de séquence, qui permettent de bien caractériser le déroulement des interactions entre objets (convient en particulier à la description détaillée de protocoles d'interaction).

III.4.CONCERNANT LES EXIGENCES EN MATIÈRE DE FIABILITÉ:

III.4.1.EXPRESSION DU TAUX DE DISPONIBILITÉ :

Il est souvent très difficile d'obtenir du maître d'ouvrage une évaluation VÉRIFIABLE du niveau de fiabilité qu'il attend. Le moyen le plus facile semble être de lui faire estimer le "taux de disponibilité" sur une échelle de 1 à 7 en utilisant le tableau suivant:

Classe	Indisponibilité sur un mois	Indisponibilité sur 1 an	Pourcentage de disponibilité
1	72 h	36,5 jours	90
2	7,2 h	3,6 jours	99
3	43,2 mn	8h 45 mn	99,9
4	4,32 mn	51,6 mn	99,99
5	25,9 s	5,2 mn	99,999
6	2,6 s	32 s	99,9999
7	0,3 s	3,2 s	99,99999

En effet, la proposition "accepter 43 mn d'indisponibilité du produit par mois" est beaucoup plus concrète pour un non spécialiste qu'une proposition concernant le M.T.B.F ou le taux de panne attendu. Rappelons d'autre part que le pourcentage de disponibilité est relié au M.T.T.F (temps moyen entre la mise en fonction et la prochaine pannes) et au M.T.T.R (temps moyen de remise en service) par la formule :

$$\text{Pourcentage de disponibilité} = \frac{M.T.T.F}{M.T.T.F + M.T.T.R} = \frac{M.T.T.F}{M.T.B.F}$$

Les taux de disponibilité exigés pour les différentes fonctions supportées par une application donnée ne sont pas forcément identiques. Ainsi, dans une application

bancaire, une défaillance de la fonction "transfert de compte à compte" est beaucoup plus problématique que celle de la fonction "visualisation d'un compte client". Il est donc important que la spécification du besoin comprenne une évaluation du taux de disponibilité attendu pour chacune des fonctions offertes par l'application.

III.4.2.IMPORTANCE DE LA DURÉE DE REMISE EN ÉTAT :

La formule ci-dessus montre qu'un même pourcentage de disponibilité peut être obtenu pour des couples (M.T.T.F, M.T.T.R) différents. Or, le temps moyen de réparation (M.T.T.R) peut être en lui-même une donnée déterminante. Par exemple, une panne sur un système frigorifique n'entraînera aucune conséquence si elle est réparée en moins de deux heures alors que les conséquences seront importantes si la réparation dure plusieurs jours.

La durée de remise en état dépend de divers facteur tels que :

- La présence d'un système assurant la redondance (à chaud ou à froid) ;
- La présence d'outils logiciels de surveillance et de visualisation en temps réel du fonctionnement ;
- l'architecture matérielle du système (facilité de l'accès aux composants, possibilité de changer un composant "à chaud");
- L'organisation du Service Maintenance (gestion du matériel de rechange, optimisation des procédures d'intervention, présence de personnels qualifiés sur les lieux) ;
- etc.

Les trois premiers points sont du domaine de la conception système.

III.4.3.EXPRESSION DU BESOIN DE FIABILITÉ :

Compte-tenu de ce qui précède, si le taux de disponibilité demeure un bon moyen de quantifier le besoin de fiabilité, la définition d'une durée de remise en état acceptable est également très importante pour affiner ce besoin.

III.4.4.REMARQUE:

L'exécution de la plupart des fonctions d'une application complexe fait appel à des ressources logicielles ou matérielles utilisées par d'autres fonctions.

Exemple : *les fonctions "transfert de compte à compte" et "visualisation d'un compte client" font toutes deux appel au système de gestion des bases de données du système informatique: une défaillance de cette entité se répercute donc sur ces deux fonctions.*

De ce fait, du point de vue du concepteur, la question de la fiabilisation d'une fonction particulière dépend de la solution de répartition qu'il aura choisie. Il conviendra donc de vérifier que les valeurs de ces deux paramètres sont définies (explicitement ou implicitement) dans la spécification du besoin.

III.5.CONCERNANT LES EXIGENCES EN MATIÈRE DE SÉCURISATION DES ACCÈS:

La sécurisation des accès est une problématique qui ne peut se traiter qu'au niveau du système informatique qui accueille l'application. En effet:

- D'une part, elle dépend pour la plus grande partie des dispositifs mis en place pour assurer la sécurité au niveau des réseaux et des machines. Ces dispositifs sont du ressort de l'administration du S.I.
- D'autre part, le fonctionnement d'une application donnée peut se trouver compromis par l'exploitation d'une faille logicielle située dans une autre application fonctionnant sur le même S.I.

Il importe donc que l'installation d'une nouvelle application ne dégrade pas le niveau de sécurité global du système par l'introduction de failles logicielles. La plupart des failles liées aux logiciels résultent d'imperfections dans la programmation (protection insuffisante contre les injections de code ou les dépassements de capacité de piles, non traitement de certaines combinaisons de paramètres dans les I.H.M, etc.).

Au niveau de la conception, on s'attachera surtout à vérifier :

- Que les accès aux fonctions "sensibles" sont réservés aux utilisateurs identifiés et que le procédé d'identification à employer est bien défini (ID. et mot de passe, lecture de badge, biométrie, etc.) ;
- Que les transmissions de données à caractère confidentiel sont bien protégées (par cryptage, tunnellisation, VPN ou tout autre moyen;
- Que dans l'ensemble des combinaisons de commandes I.H.M définis par les documents de spécification de besoin il n'en existe aucune dont le résultat soit imprévisible.

III.6.RÔLE DU CONCEPTEUR:

D'une manière générale, Il appartient au concepteur de s'assurer que les documents de spécification des besoins sont suffisamment explicites pour permettre d'engager la phase de développement.

En particulier, il doit s'assurer que les éléments énoncées dans les paragraphes ci-dessus figurent bien dans les documents de spécification des besoins qui ont été mis à sa disposition, car ceux-ci sont nécessaires pour élaborer une solution de conception. En cas d'absence ou d'insuffisance, le concepteur doit réclamer des compléments de spécifications auprès de la maîtrise d'ouvrage. Si celle-ci se révèle incapable de fournir ces compléments, le concepteur essaiera de les obtenir par toute autre voie possible, puis de les faire intégrer dans les documents de spécification APPROUVÉS PAR LE CLIENT.

IV.ÉTAPE II- ÉTUDE COMPORTEMENTALE :

IV.1.PRÉSENTATION :

Nous avons vu dans le tome I de cet ouvrage que le modèle comportemental d'une application peut être élaboré à partir des exigences et contraintes exprimées par sa S.T.B.

Il suffit pour cela d'analyser chacune des exigences et contraintes que cette dernière exprime avec les outils d'analyse comportementale qui sont à notre disposition :

- Diagrammes d'états-transition;
- Diagrammes activités-flux d'informations (Diagrammes SA-RT, diagrammes d'activités UML) ;
- Diagrammes de collaboration ou de séquences ;
- Etc.

Cette démarche correspond sensiblement à la VUE DES PROCESSUS d'U.M.L. Elle peut se décomposer en 3 étapes successives :

1. Étudier les ACTIVITÉS de l'application :
   - Identifier et caractériser les **activités logicielles** que l'application doit supporter ;
   - Identifier et caractériser les **flux d'informations** que ces activités échangent entre elles et avec la périphérie ;
   - Identifier et caractériser les **ressources partagées** entre ces activités ;
2. Repérer les **situations de concurrence** existant entre ces activités pour l'accès aux ressources partagées ;
3. Évaluer les **contraintes temporelles** auxquelles l'application doit satisfaire :
   - **Délais de réponse** à des sollicitations externes ;
   - **Déterminisme des dates** de réponse ;
   - **Durées moyennes ou maximales** des traitements .

Les paragraphes suivants détaillent ces étapes.

IV.2.IDENTIFICATION ET CARACTÉRISATION DES ACTIVITÉS :

IV.2.1.INTRODUCTION :

pour chacune des activités, il s'agit de déterminer:

- Les traitements qu'elle supporte (actions, sous-activités, nœuds d'activités);
- Les conditions de déclenchement de son exécution (ou l'événement déclencheur);
- Les données consommées;
- Les données, commandes et services fournis et les événements déclenchés;
- Les durées propres d'exécution (moyenne, maximale) ;

sans oublier d'associer à l'activité un identifiant unique utilisable dans l'ensemble de l'étude.

RAPPEL:
Dans le cadre de cet ouvrage, nous appellerons Durée Propre d'Exécution (DPE) d'une tâche la durée qui s'écoule entre le début et la fin d'exécution de cette tâche à condition que celle-ci soit EXÉCUTÉE EN PRIORITÉ PRÉEMPTIVE MAXIMALE par les systèmes d'exploitation. Cette valeur est estimée pour un CPU donné (par exemple, pour un CPU Intel I5), mais peut être recalculée pour un matériel différent. Pour plus de détails, voir l'annexe II du présent document.

IV.2.2.MÉTHODE ET OUTILS :

Divers outils d'analyse peuvent être utilisés. Parmi ceux-ci, nous pouvons citer :

- Les DIAGRAMMES DE FLUX de la méthode SA-RT (éventuellement accompagnés d'une description des activités par PSEUDO-CODE) ;
- Les DIAGRAMMES D'ACTIVITÉS d'UML.

(La description de ces outils est abordée dans l'annexe V du tome I du présent ouvrage)

Dans les deux cas, le résultat obtenu est sensiblement le même : Les deux outils permettent de faire ressortir les interactions entre activités ainsi que les ressources communes. En revanche,

- L'avantage de la démarche SA-RT, qui est de type "top-down", est qu'elle permet d'obtenir une vue d'ensemble du comportement à un certain niveau de détail. En revanche, l'aspect algorithmique des activité n'est pas abordé, d'où la nécessité de lui adjoindre une analyse de celles-ci par pseudo-code ;
- L'avantage de la démarche UML est qu'elle permet de bien décrire l'aspect algorithmique de chacune des activités. En revanche, il est plus compliqué d'obtenir une vue d'ensemble du comportement.

Dans la suite de ce sous-chapitre, un exemple d'application de ces deux outils à un même problème (distributeur de billets) est présenté :

IV.2.2.1. EXEMPLE D'UTILISATION DES DIAGRAMMES DE FLUX SA-RT :

DISTRIBUTEUR DE BILLETS :

Diagramme d'environnement :

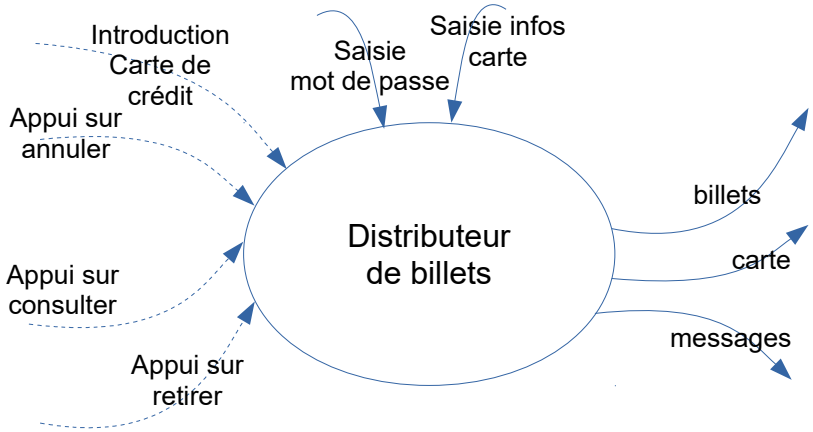
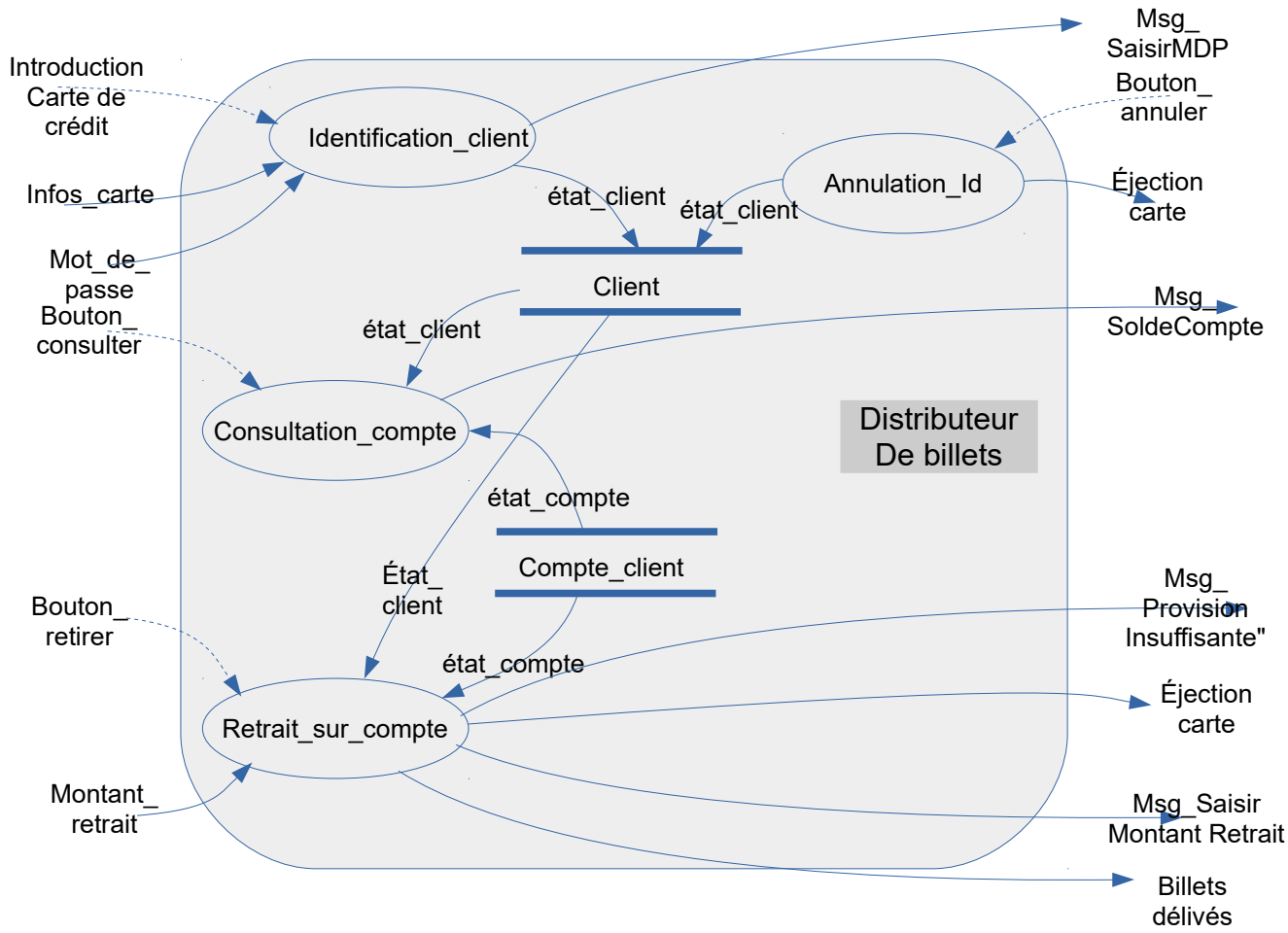


Diagramme de flux niveau 1 :



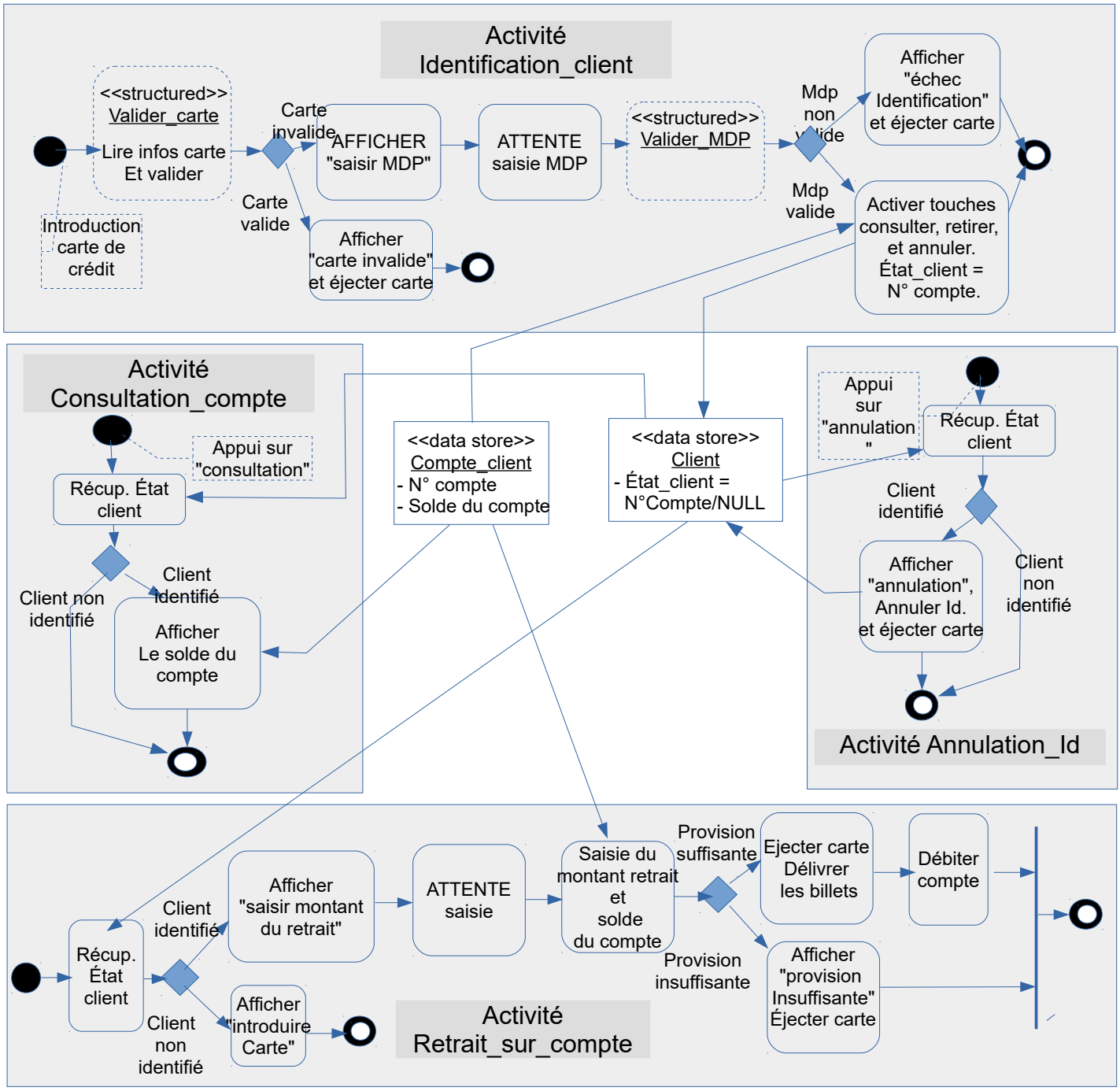
En complément du diagramme SA-RT, le tableau ci-dessous caractérise les activités identifiées :

Activité	Événements déclencheurs	Description sommaire des traitements supportés
Identification_ client	Introduction de la carte de crédit du client dans le lecteur ;	- Vérification de la compatibilité de la carte de crédit avec le distributeur (code banque et IBAN) ; - Affichage "saisir mot de passe" ; - Attente saisie ; - Saisie et vérification du mot de passe ; - SI (carte non traitable ou mot de passe erroné) ALORS Signaler "échec de l'identification", Éjecter la carte ; - SINON Activer les touches "consultation", "retrait" et "annuler" - FINSI
Annulation_Id	Appui sur la touche "annuler"	- Si (client identifié) ALORS Afficher "transaction annulée" ; Annuler l'identification ; Éjecter la carte ; - FINSI
Consultation_ compte	Appui sur la touche "consultation"	- SI (client identifié) ALORS Afficher le solde de son compte ; - FINSI
Retrait_sur_ compte	Appui sur la touche "retrait"	- SI (client identifié) ALORS Afficher "saisir le montant du retrait" ; Attente saisie ; Saisir la somme à retirer; - SI (provision suffisante) ALORS Éjecter la carte ; Délivrer les billets ; Débiter le compte ; - SINON Afficher "provision insuffisante ; Éjecter la carte. - FINSI - SINON, Afficher "introduire une carte de crédit dans le lecteur" ; - FINSI - Annuler l'identification ;

IV.2.2.2.EXEMPLE D'UTILISATION DES DIAGRAMMES D'ACTIVITÉS UML :

Par rapport aux diagrammes SA-RT, les diagrammes d'activités d'UML permettent de représenter, outre les interactions entre activités et les ressources partagées, la logique algorithmique de ces activités :

DISTRIBUTEUR DE BILLETS :



IV.2.3. SUITE DE L'ÉTUDE DES ACTIVITÉS :

L'identification et la caractérisation des activités, effectuée en s'appuyant sur les outils décrits ci-dessous doit être enrichie, pour chaque activité, par les rubriques suivantes :

- **Données consommées :** identification des données utilisées par l'activité au cours de son exécution (dans l'exemple ci-dessus, l'activité Identification_client consomme les données Infos_carte et Mot_de_passe) ;
- **Données ou services produits :** identification des données et services produits par l'activité au cours de son exécution (dans l'exemple ci-dessus, l'activité Identification_client produit la donnée État_client et émet de message Msg_SaisirMDP) ;
- **Capacités mémoires exigées** : évaluation sommaire des capacités de mémoire vive exigées pour l'exécution de l'activité (voir annexe B du présent volume) ;
- **Durées propres d'exécution :** durées propres maximales, minimales et moyennes d'exécution de l'activité dans une seule machine et en priorité maximale, par un type de CPU que l'on précisera (voir annexe A du présent volume).

IV.3.IDENTIFICATION ET CARACTÉRISATION DES RESSOURCES PARTAGÉES :

IV.3.1.NOTION DE RESSOURCE PARTAGÉE:

Chacune des activités induites par une application donnée peut être amenée à utiliser, au cours de son exécution, différentes RESSOURCES de nature matérielle ou logicielle.

Ces ressources peuvent être :

- Des équipements matériels : imprimantes, scanners, unités de stockage de données, etc ;
- Des entités logicielles : fichiers, bases de données, zones mémoires partageables, ports de communication, utilitaires logiciels (fonctions), etc.

Certaines de ces ressources sont susceptibles d'être utilisées par plusieurs activités. Ces ressources sont alors qualifiées de PARTAGÉES.

Par exemple, dans l'exemple du distributeur de billets, nous pouvons constater que la ressource Client est partagée entre les activités :

- Identification_client (en écriture) ;
- Annulation_id (en écriture) ,
- Consultation_compte (en lecture),
- Retrait_sur_compte (en lecture).

IV.3.2.PROBLÉMATIQUE DE LA CONCURRENCE SUR LES RESSOURCES :

IV.3.2.1.CONFLITS D'ACCÈS AUX RESSOURCES :

Dans un environnement d'exécution multitâches pouvant s'exécuter en simultanéité réelle (systèmes multiprocesseurs ou multicœurs) ou apparente (temps partagé), le déroulement d'une application informatique est en partie déterminé par l'état de DISPONIBILITÉ DES RESSOURCES invoquées par les différentes tâches.

En effet :

1. Certaines ressources ne peuvent être utilisées que par une seule tâche à la fois : c'est le cas de la plupart des ressources matérielles (imprimantes, disque dur), mais aussi des fichiers et des zones de mémoire partagées, des ports de communication ou des fonctions logicielles "non réentrantes". Dans ce cas, lorsque la ressource est en cours d'utilisation, les autres tâches désirant y accéder sont mises en attente
2. Certaines ressources peuvent supporter l'accès de plusieurs tâches en même temps, mais en nombre limité. C'est le cas, par exemple, d'un "pool d'imprimantes"

qui peut être utilisé simultanément par autant de tâche qu'il regroupe d'imprimantes.

Dans le cas ou une ressource R est utilisée simultanément par le nombre d'utilisateurs maximal qu'elle puisse accepter (une pour le cas n°1, N dans le cas n°2), toute nouvelle tâche T essayant d'accéder à cette ressource R risque de provoquer un conflit d'accès pouvant engendrer un dysfonctionnement fatal.

IV.3.2.2.**TRAITEMENT DES CONFLITS :**

Le seul moyen d'éviter ce problème consiste à interdire l'accès à la nouvelle tâche ou à placer cette tâche en attente de la libération de R par une des N tâches en cours d'utilisation. Ceci peut perturber de manière importante les temps d'exécution et les dates de fin des traitements.

L'étude doit donc s'efforcer d'identifier les ressources partagées par les différentes activités logicielles, et de repérer les situations où la concurrence des accès de ces activités à ces ressources peut s'avérer conflictuelle.

Ces problèmes de concurrence pourront alors être traités au niveau de la programmation de deux façons différentes :

- Soit par l'utilisation d'outils systèmes qui permettent de gérer les conflits d'accès à l'aide de mécanismes d'exclusions mutuelles ou de files d'attente (sémaphores, mutex, etc.) ;
- Soit par la synchronisation de ces accès par programme, ce qui permet de minimiser les conflits (programmation synchrone).

Pour chaque ressource, il conviendra donc de déterminer:

- Son identifiant ;
- Son type (zone de mémoire partagée, fichier disque, sheared memory, pilote de périphérique, etc);
- Sa fonction (ex : stockage de l'état courant, accès au pool d'imprimantes du siège, etc.) ;
- Les activités utilisatrices et le type d'utilisation (lecture, écriture ou les deux).

IV.3.2.3.**REMARQUE :**

dans les diagrammes SA-RT/DARTS, les zones de mémoire partagées sont identifiées comme des "puits de données persistantes". Dans les diagrammes d'activités U.M.L, elles correspondent aux "objets" prototypés <<data store>>.

IV.4.IDENTIFICATION ET CARACTÉRISATION DES FLUX D'INFORMATIONS :

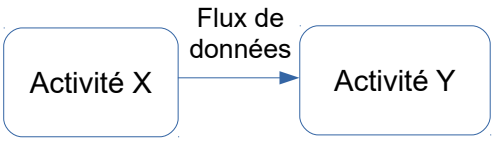
IV.4.1.DIFFÉRENTS TYPES DE FLUX D'INFORMATIONS:

Les différentes activités logicielles induites par une application échangent des informations entre elles et avec la périphérie du système (entrées-sorties) sous la forme de FLUX DE DONNÉES ou de FLUX DE COMMANDES.

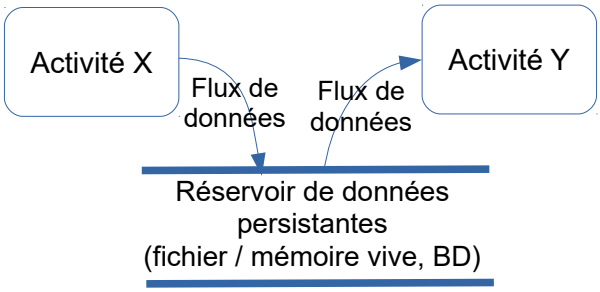
Ces informations (données, commandes, messages) peuvent être échangées :

1. Soit directement entre l'entité productrice et l'entité consommatrice (ce qui implique que les exécution de ces deux activités soient SYNCHRONISÉES au moment de l'échange) ;
2. Soit indirectement, par l'intermédiaire d'un réservoir (ou TAMPON) de données persistantes (fichier situé sur une mémoire de masse ou zone de mémoire vive permettant d'accueillir des données persistantes et partageables). Les exécutions des deux activités peuvent alors être ASYNCHRONES.

Les schémas ci-dessous illustrent les deux modes de transmission de données entre activités.



Échange Synchrone
Entre A et B



Échange Asynchrone
Entre A et B

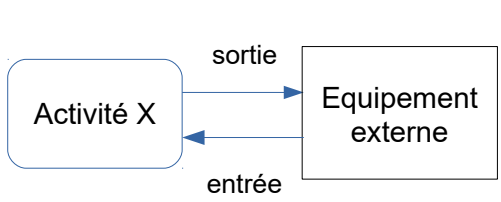
REMARQUE 1: *Lorsque le tampon est constitué par une zone de mémoire vive, nous pouvons distinguer deux variantes d'échanges asynchrones :*

- *La **BUFFERISATION** : Dans ce cas, la mémoire tampon est gérée comme une file d'attente FIFO :*
  - *Un émetteurs place le message qu'il veut transmettre en queue de file,*
  - *Un récepteur prélève le message situé en tête de file et ce message est supprimé de la file.*
- *La **BOITE AUX LETTRES** :*
  - *La zone tampon est divisée en plusieurs emplacements bien déterminés ;*

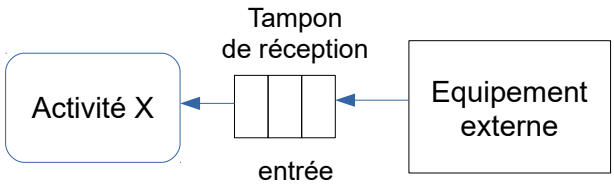
- *Un émetteur place le message qu'il veut transmettre dans un des emplacement du tampon (ce faisant, il écrase l'ancien contenu de cet emplacement) ;*
- *Un récepteur lit un des emplacements réservés de la file sans détruire son contenu ;*

REMARQUE 2 : a ce niveau de l'étude, la répartition des activités X et Y au sein du système informatique n'a pas encore été définie. Si ces activités sont finalement supportées par des machines différentes, ces flux de données circuleront alors sur des liaisons de données dédiées ou des réseaux ou par l'intermédiaire de fichiers situés sur des serveurs de fichiers ou dans des SGBD.

ÉCHANGES AVEC LA PÉRIPHÉRIE*:* Une activité peut également échanger des données avec un équipement extérieur au système. Ce type de flux est dit "d'entrée-sortie". Dans ce cas comme dans le cas précédent, les échanges peuvent être directs, ce qui implique la synchronisation du récepteur avec l'émetteur, ou indirects, par l'intermédiaire d'une mémoire tampon ("bufferisation").



Échanges Synchrone
Entre A et un équipement externe



Réception Asynchrone
Avec bufferisation

L'étude consistera à :

- Identifier les différents flux de données ;
- Préciser les types et quantités des données échangées, les contraintes temporelles pesant sur ces échanges d'informations et en déduire les débits nécessaires pour les réaliser.

Pour chaque flux de données circulant entre deux activités, entre une activité et une ressource ou entre la périphérie et une activité, il conviendra de préciser :

- Son identifiant
- Sa source (activité, ressource, équipement externe au logiciel) ;
- Sa destination (activité, ressource, équipement externe au logiciel) ;
- Son contenu (description et typage des blocs de données échangés) et son volume ;
- Sa récurrence ;

- Le type d'échange : synchrone, bufferisé, asynchrone (voir les annexes du tome 1) et s'il existe un mécanisme d'acquittement ;

REMARQUE :

La durée du transfert d'un message entre activités émettrices et destinataires dépend fortement de l'architecture physique du système d'informations. En effet :

- D'une part, un flux d'information qui relie deux machines distante n'a pas la même durée et n'est pas soumis aux mêmes contraintes que s'il se produit à l'intérieur d'une seule machine.
- D'autre part, dans le cas d'un échange entre deux machines, cette durée est fortement impactée par le type de médiat employé.

Or, au niveau d'étude ou nous nous trouvons, l'architecture physique du système d'accueil n'a pas encore été définis. Il est donc impossible d'évaluer, à ce niveau, les durées de transfert de données.

IV.5.PRISE EN COMPTE DES CONTRAINTES TEMPORELLES:

IV.5.1.INTRODUCTION :

Dans le cadre de la conception comportementale, la solution adoptée doit en premier lieu satisfaire aux contraintes temporelles qui pèsent sur l'application. Ces contraintes peuvent être de trois types :

- Le **respect d'une délais de réaction :** par exemple : "la commande d'arrêt d'exécution doit parvenir à l'équipement surveillé dans un délais inférieur à 50 ms après la réception du signal d'anomalie";
- Le **déterminisme des dates de réponse :** par exemple: "la consigne de correction de la trajectoire doit parvenir au système de guidage dans le créneau temporel allant de 20 à 25 ms après chaque seconde ronde" ;
- Le **respect d'une durées de traitement** (durée moyenne dans un espace de temps donné ou durée maximale acceptable).

Les contraintes pesant sur la réaction de l'applicatif à une sollicitation concernent en général la date ou le délais de délivrance de données ou de commandes à un équipement externe ou la date ou le délais d'affichage de la représentation de ces données sur un système de visualisation.

Ce type de contrainte est appelé "contraintes temps réel" car il caractérise la capacité de cet applicatif à répondre aux sollicitations de son environnement d'une manière satisfaisante par rapport aux contraintes temporelles exprimées par la spécification du besoin.

Le paragraphe suivant rappelle les différents niveaux de contraintes "temps réel" couramment admis.

IV.5.2.LES DIFFÉRENTS NIVEAUX DE CONTRAINTES TEMPORELLES :

Les contraintes de nature temporelle qui pèsent sur la prise en compte des interactions par l'application concernent essentiellement:

- **Les délais de réaction:** par exemple, la durée de la boucle de réaction : [acquisition d'une commande externe → traitement → restitution de la réponse à l'émetteur de cette commande], que l'on désigne souvent par "temps de réponse" ;
- **Le déterminisme des dates de fourniture des données** (exemple : une donnée doit être fournie à un système utilisateur entre 20 et 25 ms après le passage de l'heure universelle à 12h00m00s00.

Nous pouvons classer les contraintes temporelles concernant les délais et le déterminisme des dates en cinq niveaux de sévérité (voir le tableau ci-après) :

Niveau de contrainte	Temps de réponse (ms)	Déterminisme des dates (ms)	Types d'activités (exemples)
TEMPS RÉEL DUR	T < 1	ΔT < 1	- Conduite de processus critiques : applications de guidage embarquées, centrales nucléaires, etc. - Ce niveau de temps réel exige l'utilisation d'O.S spécialisés : Lynx OS, RT Linux, etc.
TEMPS RÉEL INTERMÉDIAIRE	1 <= T < 10	1 < ΔT< 10	Conduite de processus peu critiques pouvant être traitée par un OS à noyau Linux classique ou WIN 32/WIN 64
TEMPS RÉEL SOUPLE	10 <= T < 100	1 < ΔT< 100	
TEMPS RÉEL INTERACTIF	100 ≤ T < 1000 (temps de réaction "humain")	100 <= ΔT < 1000	Applications interactives (sites web, serveurs FTP ou mails, etc.
HORS TEMPS RÉEL	T ≥ 1000	T ≥ 1000	Applications de bureautique, de traitement photos, etc.

RAPPELS:

- *En informatique, un système ou une application sont qualifiés de TEMPS RÉEL lorsqu'ils sont capable de contrôler et de commander un processus physique avec des temps de réponse compatibles avec les exigences de celui-ci;*
- *Un système temps réel n'est pas forcément un système réagissant rapidement : sa principale qualité est plutôt d'avoir des temps de réaction aux événements PRÉVISIBLES (on emploie parfois le terme DÉTERMINISTE).*

IV.6.EXEMPLE D'ÉTUDE DES ACTIVITÉS :

IV.6.1.INTRODUCTION :

L'exemple qui suit a pour objet de matérialiser par un exemple concret la démarche d'étude des activités et de suggérer des outils et un formalisme de représentation.

IV.6.2.PRÉSENTATION DU CAS:

Supposons que le document de spécification des besoins d'un site web "vitrine" exprime les exigences suivantes:

EXIGENCE "Consulter_Catalogue"		
ÉNONCÉ	Un visiteur du site web doit pouvoir visualiser les articles contenus dans le catalogue.	
Critères	Domaine	Caractérisation
Critère n°1	Support de visualisation et de saisie des commandes visiteurs.	Navigateur web connecté au réseau internet
Critère n°2	Données à visualiser	- Nom commercial ; - Type du produit ; - Marque ; - Prix (TTC et HT) ; - Diaporama photos de l'article (au moins 3 photos couleurs * 48000 px).
Critère n°2	Commandes de visualisation accessibles au visiteur.	Filtrage des articles à visualiser par marques, types ou prix croissants.
Critère n°3	Utilisateurs concernés	Tous visiteurs du site.
Critère n°4	Délais de réponse entre commande utilisateur et affichage.	< 2 s (pour une taille du catalogue ≤ 1800 articles et pour moins de 50 connexions en cours).
Critère n°5	Taille maximale du catalogue	1800 articles.
Critère n°6	Disponibilité	- Taux de disponibilité : classe 3 (99,9%) ; - Durée de remise en état (M.T.T.R): < 50 mn

EXIGENCE "MAJ_Catalogue"		
ÉNONCÉ	L'administrateur du site web doit pouvoir mettre à jour la liste des articles contenus dans le catalogue et les données attachées à chacun de ces articles.	
Critères	Domaine	Caractérisation
Critère n°1	Actions permises sur les articles du catalogue	Création, modification, suppression. (Données concernées: - Nom commercial ; - Type du produit ; - Marque ; - Prix (TTC et HT) ; - Diaporama photos de l'article (au moins 3 photos couleurs * 48000 px).
Critère n°2	Menus de mise à jour	- Menu de choix de l'article à mettre à jour ou de création d'un nouvel article; - Menu de mise à jour ou suppression de l'article choisi.
Critère n°3	Utilisateurs concernés	Administrateurs du site, opérant à partir d'un navigateur web (client léger) ayant accès à internet.
Critère n°4	Délais de réponse maximal	< 2 secondes (pour une taille de catalogue ≤ 1800 articles) ;
Critère n°5	Disponibilité	- Taux de disponibilité : classe 3 (99,9%) ; - Durée de remise en état (M.T.T.R): < 50 mn

L'objectif est d'identifier et de caractériser :

- Les différentes activités induites par ces deux exigences ;
- Les flux de données et de commandes échangés entre elles et avec la périphérie ;
- Les ressources partagées par ces activités et les problèmes posés par leurs accès concurrents.

IV.6.3.ÉTUDE DES ACTIVITÉS:

IV.6.3.1.INTRODUCTION :

Pour mener à bien cette étude, qui exige une vision globale de l'application, il est commode de s'appuyer sur une représentation graphique globale. Nous pouvons utiliser pour cela les Diagrammes d'activité U.M.L, les diagrammes de flux de données SA/RT ou tout autre outil équivalent.

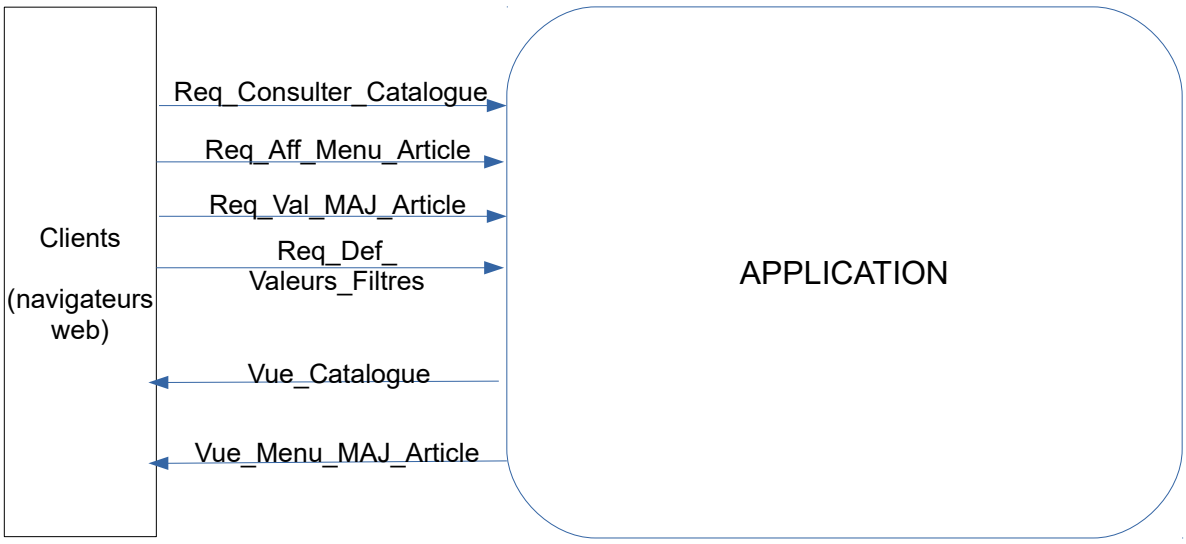
Dans le cas présent, nous avons choisi d'utiliser les diagrammes SA/RT, car son formalisme est moins complexe que celui des diagrammes d'activité. Un exemple d'étude d'activités par les diagrammes d'activité d'UML est donné dans le tome III.

NOTA :

- *voir l'annexe V du tome 1 pour la symbolique des diagrammes SA/RT ;*
- *Dans le schéma, MAJ est l'abréviation de "mise à jour";*
- *Les activités sont figurées sous la forme de rectangles à coins arrondis.*

IV.6.3.2.DIAGRAMME SA-RT D'ENVIRONNEMENT :

De l'expression des besoins, nous pouvons déduire le diagramme d'environnement suivant :



COMMENTAIRES:

A ce niveau de détail, nous pouvons identifier et caractériser les flux que l'application échange avec sa périphérie :

Id. FLUX	SENS (E/S)	User	CONTENU	Arguments
Req_Consulter_Catalogue	E	Tous	Demande affichage catalogue	aucun
Req_Aff_Menu_Article	E	Tous	Demande d'affichage du menu de création ou modification d'un article (sélectionné sur la page catalogue)	Id. article sélectionné
Req_Val_Menu_Article	E	Admin	Demande de validation de la création/modification/suppression d'un article	- Id. article ; - Id ; action (Création, MAJ, Suppression) ; - Valeurs des champs.
Req_Def_Valeurs_Filtres	E	Tous	Demande validation de la modification d'un filtre	- Type_Filtre - Valeur_Filtre
Vue_Catalogue	S	Tous	Affichage des articles du catalogue sélectionnés par les filtres et des menus de définition des filtres.	
Vue_Menu_Gestion_Article	S	Admin	Affichage du menu de mise à jour ou suppression de l'article sélectionné ou de création d'un nouvel article.	

IV.6.3.3.**DIAGRAMME DE FLUX (NIVEAU I) :**

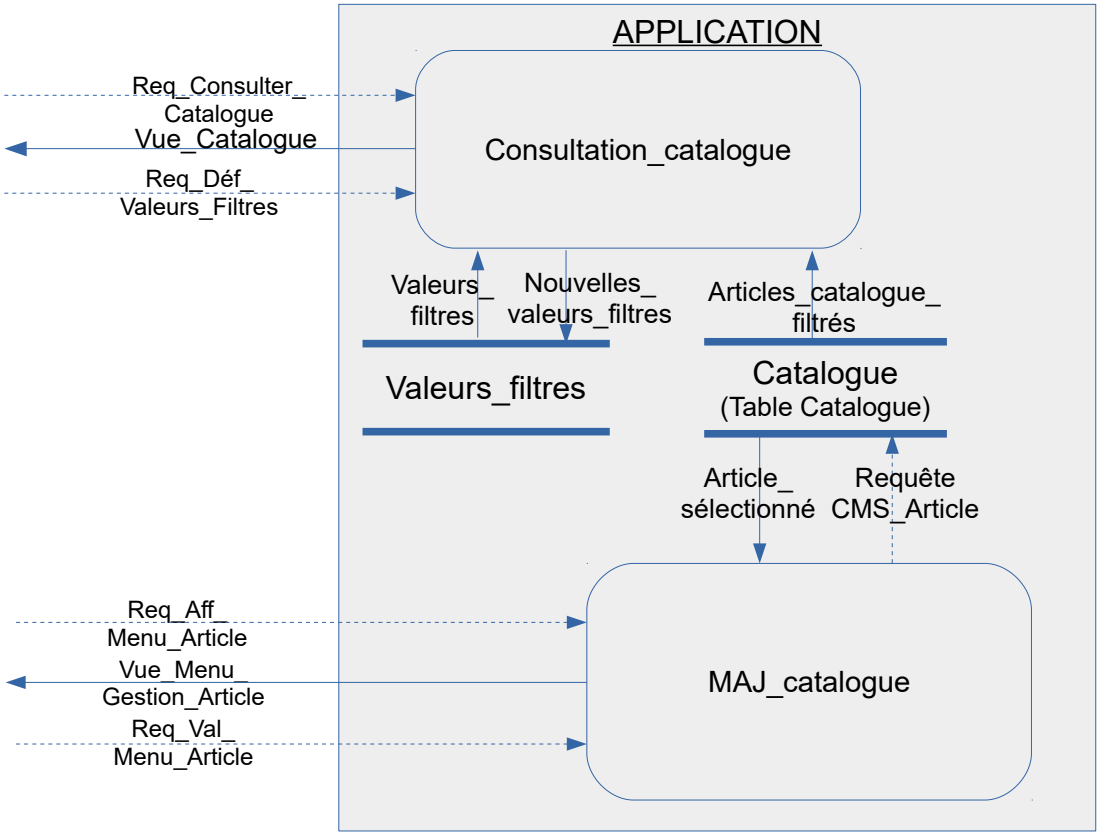
Deux exigences sont exprimées :

1. Un visiteur du site web doit pouvoir visualiser les articles contenus dans le catalogue,
2. L'administrateur du site web doit pouvoir mettre à jour la liste des articles contenus dans le catalogue et les données attachées à chacun de ces articles,

Nous pouvons en déduire deux activités globales:

- Consultation_catalogue
- MAJ_catalogue.

Le diagramme de flux SA-RT ci-dessous tente de modéliser l'activité globale de l'application :



COMMENTAIRES :

Cette modélisation permet de faire apparaître deux ressources communes aux deux activités :

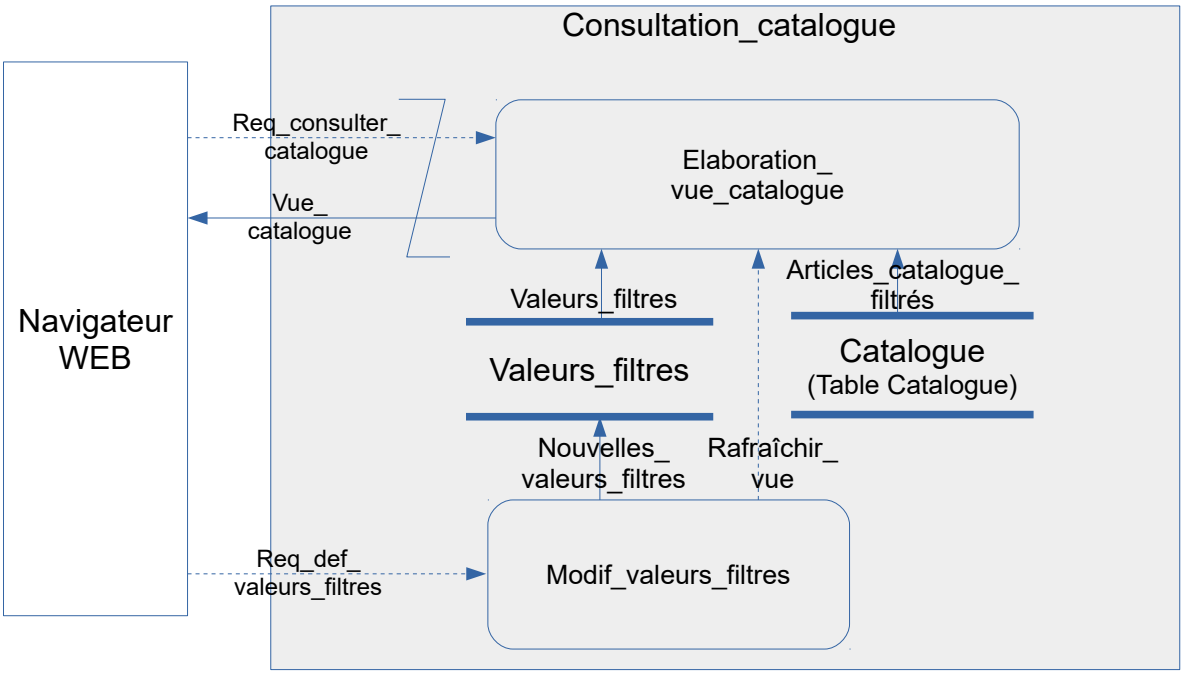
- La ressource Valeurs_filtres, qui stocke les valeurs courantes des filtres de sélection des articles pour l'affichage du catalogue (sélection par type, marque et prix). Il s'agit de données persistantes ;
- La ressource Catalogue, qui est constituée d'une table de la base de donnée (la table catalogue des articles).

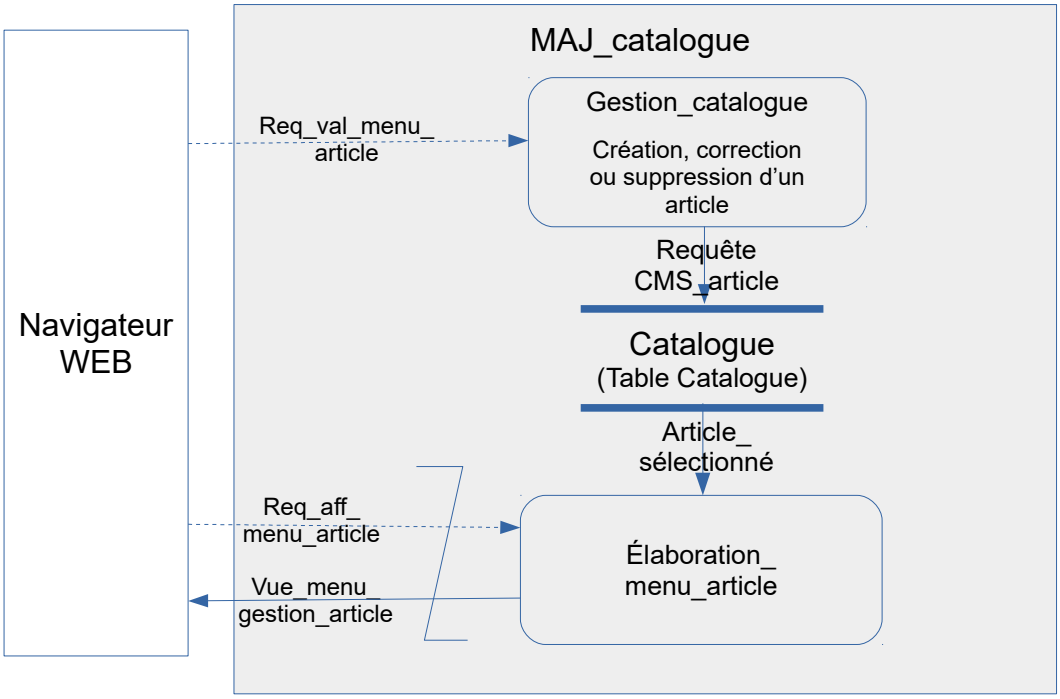
Elle permet également d'identifier les flux de données internes suivants :

Id. Flux	contenu
Valeurs_filtres	Valeurs courantes des filtres d'affichage du catalogue (marques, types ou prix croissants)
Nouvelles_valeurs_filtres	Nouvelles valeurs des filtres d'affichage
Articles_catalogue_filtrés	Articles sélectionnées par les filtres d'affichage
Article_sélectionné	Id. article sélectionnée pour être créé, mis à jour ou supprimé (si Id. = 0, l'article doit être créé
Requête_CMS_Article	Requête de création, mise à jour ou suppression d'un article

*IV.6.3.4.**DIAGRAMME DE FLUX (NIVEAU II) :***

Détaillons maintenant les deux activités identifiées plus haut. Nous obtenons les deux diagrammes de détail suivants :





COMMENTAIRES :

- Ces deux diagrammes de détail mettent en évidence 4 SOUS- ACTIVITÉS :

ACTIVITÉ	ÉVÉNEMENT DÉCLENCHEUR	TRAITEMENTS SUPPORTÉS	INDUITE PAR EXIGENCES
Modif_valeurs_filtres	Réception de la requête HTTP Req_def_valeurs_filtres	En fonction des valeurs de filtrage reçues dans Req_Def_Valeurs_Filtres, modifier les valeurs de filtrage courantes dans la ressource Valeurs_filtres	Consulter_ Catalogue
Élaboration_vue_ catalogue	Réception de la requête HTTP Req_def_valeurs_filtres	Construire la page web de présentation du catalogue en tenant compte des valeurs courantes des filtres.	Consulter_ Catalogue
Élaboration_menu_ article	Réception de la requête HTTP Req_aff_menu_article	Construire la page web "menu de gestion d'un article" en renseignant les champs en fonction de l'Id. Article figurant dans la requête Req_aff_menu_article	MAJ_Catalogue
Gestion_catalogue	Réception de la requête HTTP Req_val_menu_article	Créer, mettre à jour ou supprimer un article en fonction de l'Id. article, du type d'action et des valeurs reçues dans la requête req_val_menu_article	MAJ_Catalogue

- Ils mettent également en évidence les puits de données "Valeurs_filtres" (qui stocke les paramètres de filtrage des articles à afficher: marques, types, prix croissants, prix décroissants) et "Catalogue", qui stocke le catalogue des articles ;
- Ils permettent également d'identifier les différents flux de données et de contrôle échangés entre les différents traitements et entre les traitements et la périphérie, et d'en déterminer les volumes, les types des données transportées et les contraintes temporelles à respecter.

IV.6.3.5.ÉTUDE DE CHAQUE ACTIVITÉ :

Il s'agit alors d'étudier chaque activité en nous appuyant sur la spécification du besoin. Nous avons choisi de le faire sous la forme de tableaux.

ACTIVITÉ Élaborer_vue_catalogue		
Rubriques	Identification	Caractérisation
Description sommaire.	Élabore la page web présentant les articles du catalogue sélectionnés par les filtres.	- Initialiser la vue Catalogue - Lire les articles sélectionnés dans la table des articles. - POUR (chaque article du catalogue) FAIRE SI (accepté par filtres courants) ALORS Introduire l'article dans la vue Catalogue ; - FINFAIRE - Émettre la nouvelle vue vers le client web
Événement initiateur	Réception requête HTTP Req_consulter_catalogue.	- Événement aléatoire ; - Probabilité de survenance : dépend de la fréquentation du site ;
Données consommées	- Articles sélectionnés pour l'affichage : - Valeurs courantes des filtres .	- Articles extraits de la table catalogue de la base de données et sélectionnés par les valeurs de filtres courantes. - Valeurs courantes des filtres affichage contenues dans la ressource Valeurs_filtres
Données ou services produits	Code HTML de la page à visualiser.	Volume ≤ 50 Ko
Capacités mémoires exigées	Taille du code HTML en mémoire	Fonction du nombre d'articles maxi. Visualisés (autour de 50 kbytes).
Durées propres d'exécution.	- Durée maximale : correspond à la visualisation de 1800 articles. - Durée moyenne : correspond à la visualisation d'une vingtaine d'articles ;	CPU: I5, langage : PHP - Durée maxi: 40 ms; - Durée moyenne: 6 ms. ***REMARQUE :*** - Durée requêtes SELECT pour 1800 articles ~ 39 ms ; - Durée pour 20 articles ~ 5 ms ; - Plus 1 ms pour la gestion de l'algorithme.

ACTIVITÉ Élaboration_menu_article"		
Rubriques	Identification	Caractérisation
Description sommaire.	Élabore la vue contenant un menu permettant à l'administrateur de : - Créer un article ; - Définir les valeurs des différents champs de l'article sélectionné ; - Supprimer cet article ;	- <Créer le formulaire de saisie des valeurs d'un article>; - Récupérer le paramètre Id_article de la requête Req_aff_menu_article ; - SI (Id. article > 0) ALORS - Initialiser le mode à "mise à jour" ; - Lire l'article à modifier ; - Initialiser les champs à partir des valeurs de champs courantes de l'article ; - SINON SI (Id. article = 0) ALORS - Initialiser le mode à "création" ; - SINON - Initialiser le mode à "suppression" - FINSI - Émettre la page web vers le client.
Structure du menu	Formulaire permettant de créer, mettre à jour ou supprimer un article.	**Champs "Mode de fonctionnement" :** Création , mise à jour ou suppression. **Champs de saisie des nouvelles valeurs de paramètres (initialisés aux valeurs courantes)**: - Nom commercial ; - Type de produit ; - Marque ; - Prix ; - Photos de l'article (3 photos couleurs * 48000 px). **Bouton Validation.**
Événement initiateur	Réception requête HTTP Req_aff_menu_article	Événement aléatoire (activité de mise à jour du catalogue par les administrateurs).
Données consommées	- Id. de l'article sélectionné; - Valeurs courantes des champs de l'article sélectionné ; - Photos de l'article ;	Pour une création : Id. article = 0.
Données ou services produits	Code HTML de la vue contenant le menu de gestion de l'	50 ko.
Capacités mémoires exigées	Taille code HTML en mémoire	50 ko.
Durées propres d'exécution.	- Durée maximale : 0,8 ms (lecture d'un article en BDD).	CPU: I5, langage : PHP Durée maxi: 0,8 ms .

ACTIVITÉ Gestion_catalogue		
Rubriques	Identification	Caractérisation
Description sommaire.	Mettre a jour la table des articles en base de données suite à la réception de la validation du menu de gestion d'un article.	Supprime ou met à jour un article existant ou en crée un nouveau.
Événement initiateur	Réception de la requête HTTP Req_val_menu_article	Événement aléatoire (activité de mise à jour du catalogue par les administrateurs).
Données consommées	- Id. article concerné ; - Type d'opération à réaliser (créer, supprimer, modifier l'article) ; - Valeurs de champs de l'article ;	
Données ou services produits	Code HTML de la vue.	50 ko.
Capacités mémoires exigées	Taille code HTML en mémoire	50 ko.
Durées propres d'exécution.	- Durée maximale : 1 ms (création, modification ou suppression d'un article en BDD).	CPU: I5, langage : PHP Durée maxi: 0,8 ms . (durée requête SELECT pour 1 article ~ 0,7 ms).

ACTIVITÉ **Modif_Valeurs_Filtres**		
Rubriques	Identification	Caractérisation
Description sommaire.	Mettre a jour les valeurs courantes des filtres à partir des valeurs reçues avec la requête Req_Def_Valeurs_ Filtres : Type_Filtre et Valeur_Filtre.	SUIVANT LE CAS (TypeFiltre) FAIRE Type 1 : Valeur courante = ValeurFiltre_1 ; Type 2 : Valeur courante = ValeurFiltre_2; etc ; FINCAS
Événement initiateur	Réception Requête HTTP Req_def_valeurs_filtre	Événement aléatoire (activité de mise à jour du catalogue par les administrateurs).
Données consommées	- TypeFiltre - Nouvelle valeur du filtre	Arguments de la requête Req_def_valeurs_filtre
Données ou services produits	Mise à jour de la ressource Valeurs_filtres	
Capacités mémoires exigées	Négligeable	
Durées propres d'exécution.	- Durée maximale : 200 ns.	CPU: I5, langage : PHP

REMARQUE:

A partir de ces quatre tableaux, il est facile de calculer les durées maximales et moyennes propres de la "boucle" réception requêtes → émission page web dans le cas de la consultation du catalogue:

- Durée propre maximale : ~ 30 ms ;
- Durée propre moyenne : ~ 5 ms.

Avec un seul serveur en ligne, une moyenne de 50 utilisateurs connectés simultanément (spécifié par critère N°6 de l'exigence) un traitement CPU en temps partagé, la durée moyenne de traitement d'une requête utilisateur sera, au niveau du serveur, de l'ordre de 300 ms. Pour trouver le temps moyen de réaction à une requête (délais entre envoi de la requête et affichage sur l'IHM), il faudrait rajouter les durées des échanges réseaux entre clients et serveur qui dépendent de la charge du réseau. Cependant, la durée de réaction maximale spécifiée par le critère n°6 (2 s) devrait être facilement respectée.

Si ce n'était pas le cas, il faudrait envisager soit d'augmenter la puissance de la machine serveur, soit de déployer un cluster de plusieurs serveurs.

IV.6.4.ÉTUDE DES FLUX DE DONNÉES INTERNES ET EXTERNES :

L'étude du diagramme général va également nous servir à identifier et dimensionner les flux de données circulant entre activités ou entre activités et équipements externes. Cette étude nous sera utile lors de l'étude de l'impact de la répartition des activités sur les différentes machines du système informatique d'accueil.

Le tableau suivant donne un exemple d'étude des flux :

Id.	Émetteur/source	Récepteur/ réceptacle	Contenu et volume	Fréquence
Requêtes_ HTTP	Client web	Activité Traiter_Requête_H TTP	Requêtes HTTP get et post (longueur maxi : 2048 octets	Aléatoire : pour un client donné, 0,5 hz maxi.
Consulter_ Catalogue	Traiter_Requête_ HTTP	Élaborer_vue_ catalogue	Signal	Idem.
Afficher_ Menu_Article	Traiter_Requête_ HTTP	Elaborer_vue_ gestion_article	Signal + Id. article	Idem.
Valider_ Menu_Article	Traiter_Requête_ HTTP	Gestion_article	Signal + Id article + contenu	Idem.
Vue_ Catalogue	Activité Élaborer_vue_catalo gue	Client Web	Réponses à requêtes HTML (longueur max : 50 ko)	Idem.
Def_Valeurs_ filtres	Activité Traiter_Requête_ HTTP	Activité Modifer_Valeurs_ Filtres	- TypeFiltre (Id. du filtre à modifier) ; - ValeurFiltre (Nouvelle valeur du filtre).	Idem.
Valeurs_ filtres	Ressource Valeurs Filtres	Activité Elaborer_Vue_ Catalogue	Valeurs articles. Volume max : 200 ko	Idem.
Articles_ sélectionnés	Ressource Base de Données (table Catalogue)	Ressource Valeurs_Filtres	Valeurs articles. Volume max : 200 ko	Idem.
Article_à_MAJ	Ressource Table Catalogue	Elaborer_Vue_ Gestion_Article	Valeurs d'un article Volume : 100 o	Idem.

REMARQUE : comme il a été dit précédemment, il n'est pas forcément nécessaire d'étudier chaque flux avec le degrés de détail atteint dans ces tableaux : un concepteur expérimenté pourra se limiter aux flux "dimensionnants".

IV.6.5.ÉTUDE DES RESSOURCES PARTAGÉES:

L'étude des ressources partagées par les activités permet d'identifier les conflits pouvant survenir lors des accès concurrents des activités à ces ressources. Les résultats de l'étude pourront influer sur la conception des couches métier et données de l'application. Par exemple, ces résultats pourront déterminer :

- L'adoption d'une architecture d'intégration des bases de données qui permet l'accès simultané de plusieurs activités aux mêmes données (comme certains RAID, par exemple) ;
- Le choix d'un pool d'imprimantes partageables plutôt qu'une imprimante unique ;
- La nécessité de concevoir certaines fonctions logicielles de façon à ce qu'elle permettent la réentrance;
- Etc.

Dans l'exemple étudié, le diagramme de flux ne fait apparaître que deux ressources partagées. Le tableau suivant donne une solution pour la présentation des travaux :

Ressource partagée	Activités en concurrence sur la ressource	Sens d'accès
Valeurs_filtres	Traiter_Requête_HTTP	Écriture
	Élaborer_Vue_Catalogue	Lecture
Base de Données	Élaborer_Vue_Catalogue	Lecture
	Gestion_articles	Écriture

A priori, les activités Traiter_Requête_HTTP et Élaborer_Vue_Catalogue sont synchrones l'une de l'autre. Leur accès concurrent sur la ressource Valeurs_filtres ne devrait donc pas poser de problème.

En revanche, les activités Élaborer_Vue_Catalogue et Gestion_articles sont totalement asynchrones l'une de l'autre. Leurs accès concurrents à la base de donnée devront donc être gérés par un mécanisme d'exclusion mutuelle.

V.ÉTAPE III: CONCEPTION SYSTÈME:

V.1.OBJECTIFS VISES :

Cette étude a pour but de DIMENSIONNER le système d'accueil de l'application (infrastructure matérielle et systèmes d'exploitation) pour qu'il puisse supporter les contraintes imposées à l'application future. Les résultats seront utilisées pour élaborer et proposer une solution architecturale pour le futur système d'information support de l'application (nouveau système ou redimensionnement d'un système existant pour lui permettre d'intégrer cette nouvelle application).

REMARQUE : Il ne faut pas oublier qu'à ce niveau du projet, il ne s'agit pas d'élaborer **LA** solution définitive, mais plutôt de concevoir une "maquette de solution" qui permettra :

- Soit de prouver la faisabilité de ce projet dans le cadre des exigences et contraintes imposées
- Soit de justifier une modification de ces exigences et contraintes, soit encore de justifier la majoration du prix de la prestation ou l'extension des délais de la réalisation.

V.2.ÉTUDE DE LA COUCHE PRÉSENTATION:

V.2.1.PRÉSENTATION ET ANALYSE DES DIFFÉRENTES SOLUTIONS:

V.2.1.1.REMARQUES PRÉLIMINAIRES :

La couche PRÉSENTATION a pour rôle principal de gérer les interfaces homme-machine. Ceci implique qu'elle supporte les traitement d'élaboration et d'affichage des différentes vues nécessaires à l'exploitation de l'application ainsi que l'acquisition des commandes des utilisateurs.

Une caractéristique des logiciels d'I.H.M est qu'ils se comportent vis à vis de l'application qu'ils contrôlent comme des CLIENTS d'une architecture CLIENT-SERVEUR. En effet, ils transmettent à l'application des requêtes (de commande, de changement d'affichage, d'événements, etc.) et reçoivent en retour les données nécessaires à l'actualisation de l'affichage correspondant aux changements d'états provoqués par ces requêtes.

De ce qui précède, on pourrait déduire dans un premier temps que la place "naturelle" des traitements de la couche PRÉSENTATION doit être dans le TIER CLIENT. Cependant, l'évolution des architectures des applications pendant les vingt-cinq dernières années nuance cette conclusion. Actuellement, trois types de CLIENTS peuvent être distingués:

- Les CLIENTS LÉGERS, qui n'hébergent que la sous-couche affichage (la sous-couche élaboration des vues étant alors hébergée sur le serveur d'applications);

- Les CLIENTS RICHES qui hébergent la sous-couche affichage et une partie des traitements d'élaboration des vues (le reste de l'élaboration étant hébergée sur le serveur d'applications);
- Les CLIENTS LOURDS, qui hébergent toute la couche présentation (et parfois une bonne partie des couches métier et données);

Les paragraphes suivants étudient ces trois types de répartition.

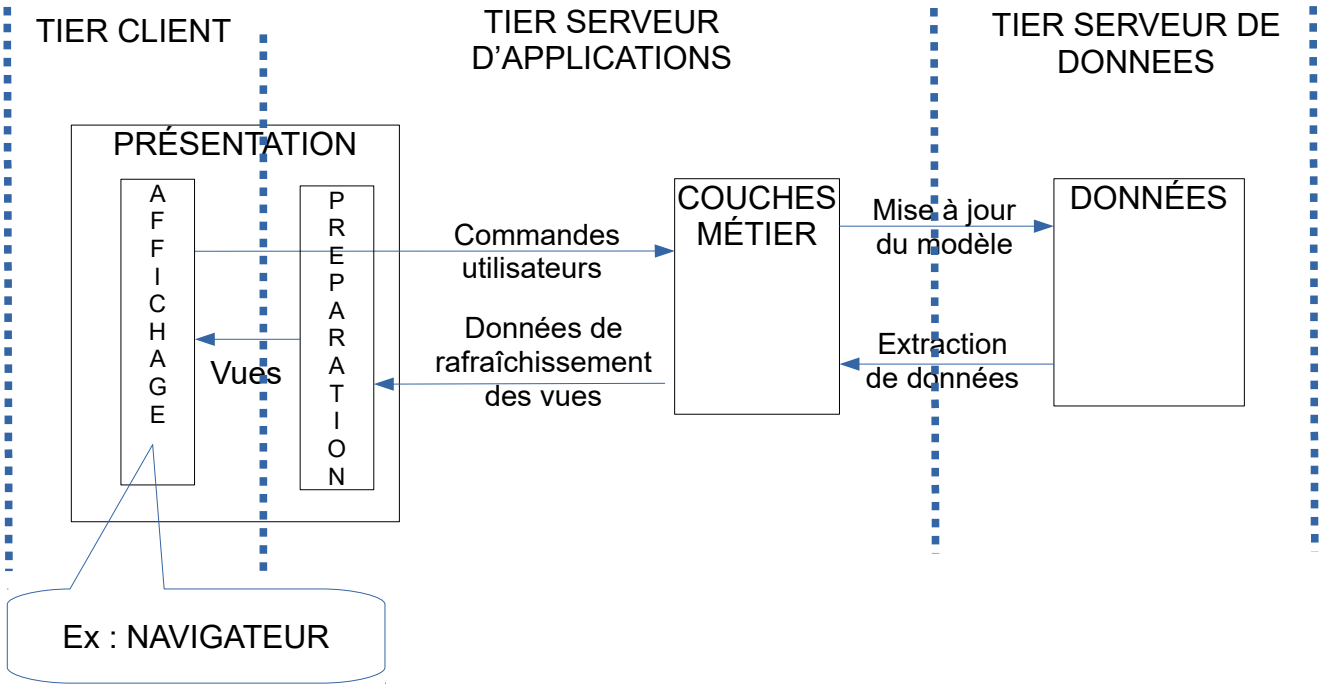
V.2.1.2.ARCHITECTURE AVEC CLIENT LÉGER:

V.2.1.2.1.PRINCIPE:

Un logiciel CLIENT LÉGER est un logiciel I.H.M graphique qui n'assure que l'affichage des vues sur l'écran local ainsi que la prise en compte des interactions avec les utilisateurs (clavier et souris) et leur émission vers un serveur capable de les traiter. Les vues elles-mêmes doivent être élaborées au niveau du serveur d'application. Les exemples de clients légers les plus répandus sont les NAVIGATEURS WEB, lorsqu'on n'utilise ni javascript/ajax, ni les applets java.

Par extension, on appelle également CLIENT LÉGER une machine hébergeant un client léger.

Le schéma ci-dessous représente la répartition correspondant à l'utilisation d'un client léger:



REMARQUE : dans ce schéma, la couche logicielle PRÉSENTATION a été scindée en deux sous-couches :

- La sous-couche PRÉPARATION dont le rôle est de préparer les différentes VUES à afficher sur l'écran de l'I.H.M et de les agencer conformément au LAYOUT choisi pour structurer l'image complète ;
- La sous-couche AFFICHAGE dont le rôle est d'afficher l'image complète sur l'écran de l'I.H.M.

*V.2.1.2.2.**AVANTAGES:***

- Le logiciel client léger consommant peu de ressources, il peut fonctionner sur des postes de travail clients peu puissants (P.C anciens de récupération, par exemple) ou sur des postes dédiés prioritairement à d'autres tâches;
- La possibilité d'utiliser un navigateur (logiciel gratuit et déjà disponible dans les configurations de base des postes de travail) sur les postes clients facilite grandement le déploiement de l'application et diminue les coûts d'acquisition et de maintenance (sous réserve de maintenir à jour les versions);

*V.2.1.2.3.**INCONVÉNIENTS:***

- Lorsqu'on utilise un client léger, le flux de données circulant dans le sens SERVEUR → CLIENT est en général très important. En effet, le client léger n'élaborant pas lui-même les vues qu'il affiche, il faut lui fournir toutes les informations décrivant ces vues, c'est à dire les informations de contenu, mais aussi celles concernant la présentation graphique (positionnements, couleurs, styles, taille et polices de caractères, etc.). Par exemple, un navigateur reçoit les descriptions en HTML des différentes vues, ce qui peut représenter des centaines de milliers d'octets. Le réseau peut donc subir des congestions importantes lors des échanges SERVEUR → IHM (avec le facteur aggravant qu'il s'agit de communications en mode connecté avec répétition des envois en cas d'erreur de transmission). Ces épisodes de congestion peuvent perturber d'autres échanges de l'application, notamment avec des serveurs d'acquisition, ce qui entraîne souvent la nécessité de séparer les réseaux d'acquisition de données et les réseaux de commande;
- Les délais d'affichage des données et de transmission des commandes opérateurs peuvent être assez élevés et surtout peu déterministes du fait du protocole de communication réseau connecté entre le client et le serveur: ceci peut poser problème si des contraintes de type "temps réel" portent sur cet affichage;
- Les clients graphiques légers offrent des possibilités d'affichage relativement faibles par rapport à des IHM élaborés à partir d'ateliers graphiques (liste de widgets plutôt réduite, obligation de rafraîchir l'écran d'un seul bloc limitant la vitesse de rafraîchissement, etc.).

V.2.1.3.ARCHITECTURE AVEC CLIENT LOURD:

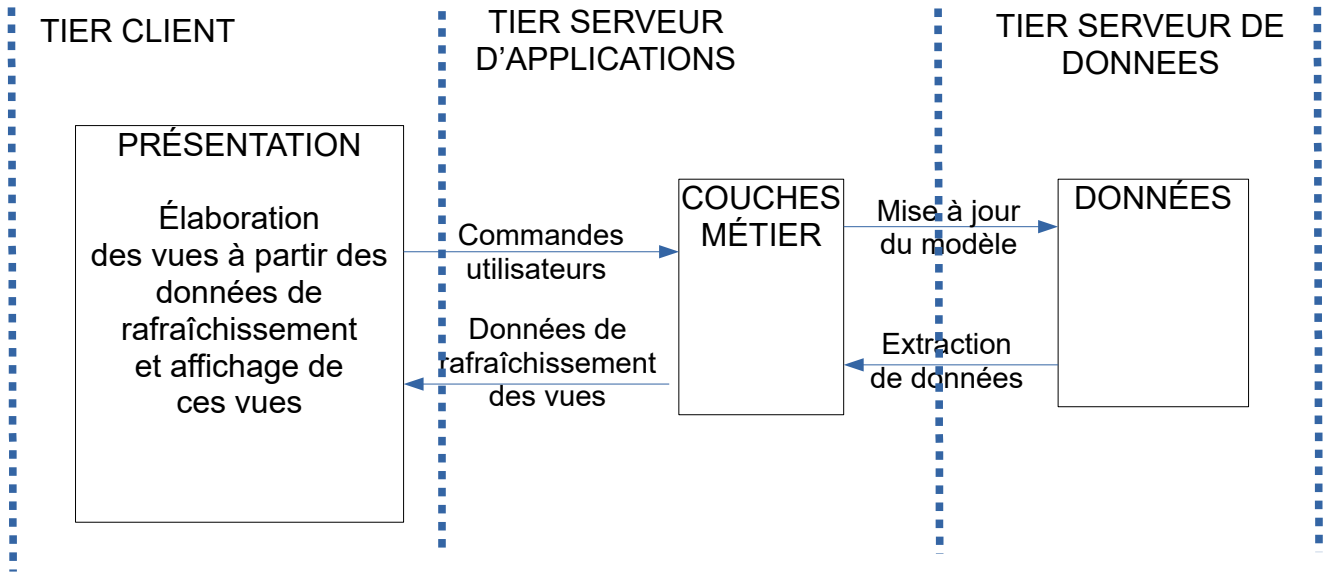
V.2.1.3.1.DÉFINITION:

Un CLIENT LOURD peut être:

- Soit un logiciel autonome installé entièrement sur un seul poste de travail ("stand alone") et possédant une interface graphique évoluée et complexe, ce qui correspond à une architecture physique 1-TIER (par exemple, un traitement de texte installé en local);
- Soit une interface graphique évoluée et complexe supportant toute la couche de présentation (élaboration des vues à partir des données fournies par un serveur, affichage de ces vues, transmission des commandes des utilisateurs).

Par extension, on appelle également CLIENT LOURD une machine hébergeant un logiciel client lourd.

Le schéma ci-après représente la répartition correspondant à l'utilisation d'un client lourd:



V.2.1.3.2.AVANTAGE:

- Réduction du flux d'informations entre le serveur et les clients:

  ***EXEMPLE**: supposons que l'IHM affiche un cadran à déviation d'aiguille représentant la valeur d'une vitesse. Un client lourd n'aura besoin que de la valeur de la vitesse pour rafraîchir la vue: il se chargera lui-même de transformer la donnée vitesse en un angle, puis de créer la vue du cadran avec la position de l'aiguille adéquate, puis d'afficher cette vue. Le flux "Données de rafraîchissement des vues" circulant entre le TIER serveur et le TIER CLIENT sera donc beaucoup*

moins important que dans le cas d'un client léger où ce flux correspondra au code HTML de la page.

De ce fait, l'impact de la communication CLIENT-SERVEUR sur le réseau sera réduit au minimum;

- Les délais d'affichage des données peuvent être bien maîtrisés, surtout si les données de rafraîchissement sont transmises en mode non connecté (UDP): cette solution convient donc aux applications à contraintes "temps réel";
- L'interface graphique peut être élaboré à partir d'ateliers graphiques (DELPHY, BORLAND C++, MOTIF, etc.) possédant des bibliothèques de widgets beaucoup plus riches que celle proposée par le langage HTML;

*V.2.1.3.3.**INCONVÉNIENTS:***

- Le logiciel client est beaucoup plus conséquent et compliqué à développer. Ceci entraîne un surcoût par rapport à la solution "client léger";
- L'I.H.M consomme beaucoup de ressources de la machine CLIENT qui, de ce fait, doit être relativement puissante (donc plus chère);
- Contrairement à la solution "clients légers", les logiciels clients doivent être déployés sur les postes utilisateurs, ce qui entraîne également des délais et des surcoûts;
- La maintenance et l'évolution des IHM est également plus compliquée. Elle implique l'achat de licences onéreuses pour l'utilisation des ateliers graphiques.

V.2.1.4.ARCHITECTURE AVEC CLIENT RICHE:

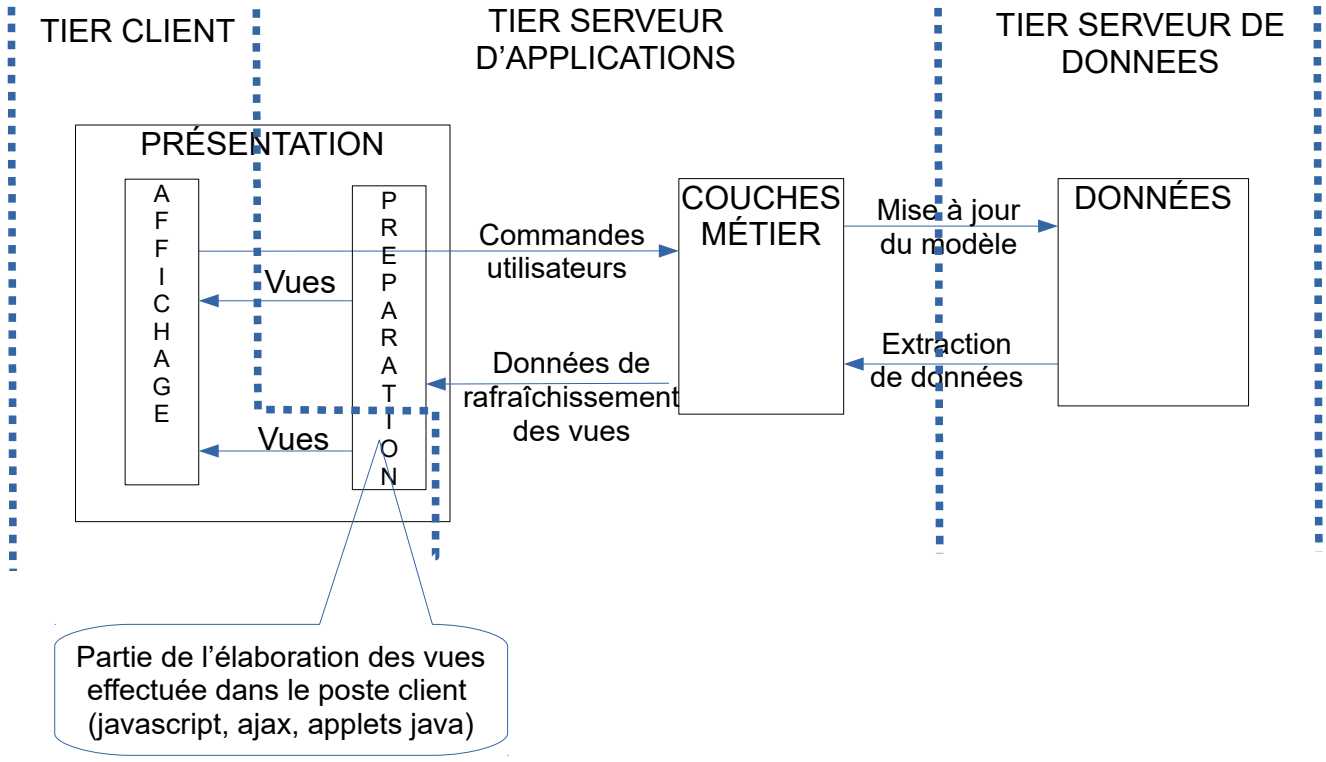
V.2.1.4.1.DÉFINITION:

Nous pouvons définir un CLIENT RICHE comme un CLIENT LÉGER capable d'effectuer une partie de l'activité d'élaboration des vues en local (côté client).

EXEMPLES:

- La plupart des navigateurs web sont capable d'exécuter le langage JAVASCRIPT et son extension AJAX. Les traitements javascript sont exécutés dans la machine CLIENT. L'extension AJAX permet de récupérer des données du serveur et de les afficher sans renouveler totalement la page affichée. Ceci permet non seulement d'élaborer des vues dans la machine cliente, mais aussi d'y effectuer une partie du travail du serveur d'application et d'augmenter la réactivité de l'I.H.M;
- Les APPLETS JAVA permettent d'élaborer dans un client léger des représentations graphiques qui n'existent pas dans la bibliothèque graphique de ce client.

Le schéma ci-dessous représente la répartition correspondant à l'utilisation d'un client riche:



V.2.1.4.2.AVANTAGES:

- Les CLIENTS RICHES permettent de diminuer la charge de travail des serveurs d'applications tout en conservant certains des avantages des clients légers:

- Logiciels clients standards et gratuits;
- Aucun déploiement de logiciels clients.

- Ils permettent d'améliorer les délais d'affichage de l'IHM:
- Ils permettent également de réduire le trafic des données entre serveur et client et les risques de congestion qui en découlent.

*V.2.1.4.3.**INCONVÉNIENTS:***

- Les clients riches peuvent consommer beaucoup plus de ressources de la machine qu'un client léger. En conséquence, la machine cliente est beaucoup plus sollicitée;
- L'emploi de langages "clients" supplémentaires tels que JAVASCRIPT et JAVA (applets) a pour conséquence de rendre la couche PRÉSENTATION plus difficile à élaborer et à maintenir.

V.2.2.CHOIX D'UNE SOLUTION:

En fonction des analyses exposées dans les trois paragraphes précédents, il est possible de déterminer le type de TIER CLIENT qui convient le mieux pour une application donnée. A titre d'exemple, nous donnons ci-après quelques éléments de choix:

V.2.2.1.INFLUENCE DU NOMBRE DE POSTES UTILISATEURS:

V.2.2.1.1.APPLICATIONS DESTINÉES À ÊTRE UTILISÉE SIMULTANÉMENT PAR DES UTILISATEURS EN NOMBRE INDÉTERMINÉ ET PAS FORCÉMENT IDENTIFIÉS

EXEMPLES : les sites web, les serveurs FTP, etc.

Dans ce cas, choisir la solution d'un client lourd revient à imposer à ces utilisateurs de charger ce client sur leur propre poste, sachant que cette option est pénalisante pour la "fréquentation" de l'application, qu'elle consomme plus de ressources CPU et mémoire des postes utilisateurs et qu'elle est plus coûteuse en frais de développement et de maintenance.

La solution CLIENT LÉGER (le navigateur de l'utilisateur) voire CLIENT RICHE (ce navigateur utilisant Javascript, AJAX, les applets JAVA, etc.) doit donc être envisagée en priorité dans ce cas.

V.2.2.1.2.APPLICATIONS DESTINÉE À ÊTRE UTILISÉE PAR DES UTILISATEURS BIEN IDENTIFIABLES

EXEMPLES: les personnels d'une entreprise ou d'un service.

Les performances des navigateurs en matière de temps de réponse, d'objets graphiques disponibles, etc. peuvent être insuffisantes lorsqu'il s'agit d'implanter un I.H.M complexe ou soumis à des contraintes de réactivité (application "temps réel" stricte, par exemple): ces caractéristiques imposent alors une solution "client lourd" qui limite le volume et la durée des échanges entre TIER CLIENT et TIER SERVEUR.

La solution CLIENT LOURD présente également beaucoup d'avantages en termes de confidentialité: l'installation du logiciel client peut être contrôlée et limitée aux postes qui en ont besoin (dans une entreprise) ou qui ont souscrit un abonnement (accès grand public).

REMARQUE: Dans ce cas où l'application est mono-utilisateur, l'hébergement de la couche PRÉSENTATION sur le TIER SERVEUR D'APPLICATION est souvent la solution la plus favorable en termes de coût de développement et de maintenance, de fiabilité et de confidentialité.

*V.2.2.1.3.**CHOIX ENTRE CLIENT LOURD ET CLIENT RICHE:***

Nous avons vu qu'un CLIENT RICHE est un CLIENT LÉGER sur lequel ont été déportés certains traitements d'élaboration des VUES (par exemple, sous forme d'APPLETS JAVA ou de modules JAVASCRIPT ou AJAX s'exécutant das l'espace du navigateur.

Cette pratique permet:

- D'alléger la charge du SERVEUR en la reportant sur le CLIENT (calculs mathématiques, élaboration de graphisme, etc.).;
- De présenter sur le CLIENT des graphismes qui ne sont pas réalisables par sa bibliothèque graphique de base (notamment grâce aux APPLETS JAVA);
- D'augmenter la réactivité de l'I.H.M en s'affranchissant de la durée de ré-affichage de la page entière (modules AJAX).

L'inconvénient est que cette pratique augmente la charge des postes CLIENTS: si ces machines ne sont pas assez puissantes, l'affichage de l'I.H.M risque d'en pâtir.

V.3.ÉTUDE DES COUCHES MÉTIER:

V.3.1.INTRODUCTION:

Dans les architectures logicielles les plus répandues, la couche MÉTIER (subdivisée en couches CONTRÔLE, SERVICE et DOMAINE dans le modèle à 5 couches) est entièrement supportée par le même TIER (en général appelé TIER SERVEUR D'APPLICATIONS). En effet, du fait des dépendances étroites qui existent entre les traitements spécifiques de ces couches (dépendances logiques et procédurales), leur répartition sur plusieurs TIER entraîne forcément une augmentation de la complexité et une diminution des performances.

Ceci ne veut pas dire que les couches MÉTIER sont forcément localisées sur une seule machine: diverses raisons peuvent conduire les développeurs à répartir l'exécution des couches MÉTIER sur plusieurs machines. Nous pouvons citer en particulier:

- L'augmentation de la PUISSANCE D'EXÉCUTION globale par l'augmentation du nombre de machines de traitement et la répartition de la charge sur ces machines;
- L'augmentation de la SÉCURITÉ DE FONCTIONNEMENT par la création de redondances matérielles.

V.3.2.LES DIFFÉRENTS TYPES D'ARCHITECTURE:

V.3.2.1.LES ARCHITECTURES COMPACTES MONO CPU:

Principe:

Une seule machine puissante dotée d'un seul CPU assure l'exécution de l'ensemble des couches métier.

Avantages:

- L'application n'est pas répartie. Elle est donc plus simple à concevoir;
- La mémoire vive est commune et accessible par toutes les entités logicielles de l'application;
- Le fonctionnement des couches métier ne nécessite pas d'échanges de messages externes.

Inconvénients:

- Au cas où la puissance de traitement devient insuffisante, il est difficile de l'augmenter sans changer de machine;
- D'autre part, une configuration mono machine ne permet aucune redondance matérielle: la fiabilité est donc faible.

V.3.2.2.LES ARCHITECTURES COMPACTES MULTI CPU:

Principe:

L'exécution de l'ensemble des couches métier est effectuée sur une seule machine multiprocesseurs ou un système construit à partir d'un bus d'intégration (V.M.E, PCI express, etc.).

Avantages:

- Comme dans le cas précédent, l'application n'est pas répartie physiquement. Elle est donc plus simple à concevoir*;*
- Une partie de la mémoire vive peut être commune, donc accessible par toutes les entités logicielles de l'application, ce qui permet d'éviter l'échange de messages externes entre les entités des couches métier;
- La parallélisation des traitements des couches métier et la répartition de leur exécution sur les différents CPU est assez facile à réaliser et très efficace;
- La possibilité d'ajouter des CPU permet de faire évoluer la puissance de traitement assez facilement et de manière importante;

Inconvénients:

- Ce type de configuration physique est en général très onéreux;
- Les solutions basées sur une machine multiprocesseurs ne permettent aucune redondance en cas de panne matérielle de cette machine;
- Celles qui sont basées sur un bus d'intégration permettent en général de pallier le dysfonctionnement d'un CPU, mais certaines pannes peuvent tout de même bloquer l'ensemble du bus.

V.3.2.3.LES ARCHITECTURE COMPACTES MONO OU MULTI CPU AVEC REDONDANCE MATÉRIELLE:

Principe:

L'infrastructure matérielle est caractérisée par un groupe d'au moins deux machines mono ou multiprocesseurs entièrement redondantes les unes des autres. A un instant donné, l'une d'entre elle (la machine "maîtresse") supporte l'ensemble des fonctions de l'application alors que la ou les autres machines (machines "adjointes") se tiennent prêtes à remplacer la machine maîtresse en cas de défaillance de celle-ci. Le basculement peut être effectué à froid où à chaud.

Avantage:

Ce type de configuration résout le problème de fiabilité lié à la configuration mono-machine (surtout si le basculement maîtresse → adjointe est effectué à chaud);

Inconvénient:

- Le fait d'avoir à se doter de machines adjointes supportant la même périphérie et les mêmes logiciels que la machine maîtresse constitue évidemment une source de surcoût non négligeable;
- En fonctionnement nominal, la puissance des machines "adjointes" n'est pratiquement pas utilisée.

Une solution pour diminuer ce surcoût est d'utiliser pour constituer la machine adjointe le "lot de rechange" de la machine maîtresse, mais dans ce cas, la fiabilité après un premier basculement est très dégradée.

V.3.2.4.LES ARCHITECTURES RÉPARTIES:

Principe:

L'exécution des traitements concernés par les couches métiers est répartie sur plusieurs machines. Cette répartition peut être statique (l'exécution d'un traitement donné est toujours confié à la même machine) ou dynamique (la répartition est effectuée en fonction de l'état de ces machines et de leur charge instantanée, de façon à optimiser leur utilisation). Elle est effectuée par un élément logiciel appelé "répartiteur de charge (load balancing)".

Avantages:

La répartition de l'exécution des couche métier sur plusieurs machines offre plusieurs pistes d'optimisation des traitements:

- Elle offre la possibilité d'exploiter les possibilités de parallélisation des traitements, ce qui intrinsèquement permet d'améliorer leur durée apparente;

- Si la répartition de la charge **est dynamique et tient compte de l'état des différentes machines de la grappe**, elle induit naturellement une augmentation de la fiabilité, car un nœud défaillant sera automatiquement écarté de la répartition, permettant ainsi son remplacement ou sa remise en condition opérationnelle sans perturbation notable de l'exécution en cours.
- Une structure répartie est beaucoup plus évolutive qu'une structure mono machine car l'ajout d'un nœud dans la grappe ne demande qu'un minimum d'étude d'impact;
- Enfin, l'acquisition d'un cluster de deux ou trois machines de moyenne puissance est souvent mois onéreuse que l'acquisition d'une seule machine puissante.

Inconvénients:

- Une application répartie est souvent beaucoup plus complexe à réaliser qu'une application compacte. L'accroissement de la complexité est dû à la nécessaire synchronisation des taches réparties et à l'absence de mémoire partageable qui entraîne un accroissement des échanges de données entre les nœuds;
- La répartition exige l'ajout de traitements supplémentaires pour gérer la parallélisation. Il faut veiller à ce que cet accroissement n'annule pas les gains acquis par la parallélisation.

V.3.3.**TECHNIQUES DE RÉPARTITION DE LA CHARGE:**

V.3.3.1.**REMARQUES PRÉLIMINAIRES:**

- Pour que deux activités logicielles puissent être exécutées simultanément par un même CPU (simultanéité apparente) ou par deux CPU différents (simultanéité réelle), la condition nécessaire est qu"il n'existe aucune relation de précédence entre elles. Autrement dit, il faut que l'exécution d'une de ces activités ne soit pas conditionnée par la fin d'exécution de l'autre;
- L'augmentation du nombre de CPU permet d'accroître la capacité d'exécution disponible pour l'application à condition que les traitements à exécuter comportent des activités PARALLÉLISABLES et que la capacité d'exécution CPU consommée par la répartition et la parallélisation de ces activités soit inférieure à l'accroissement obtenu par cet accroîssement.

V.3.3.2.**PRINCIPE:**

Lorsque des activités affectées au TIER SERVEUR D'APPLICATION peuvent s'exécuter indépendamment les unes des autres, il est possible de paralléliser leur exécution:

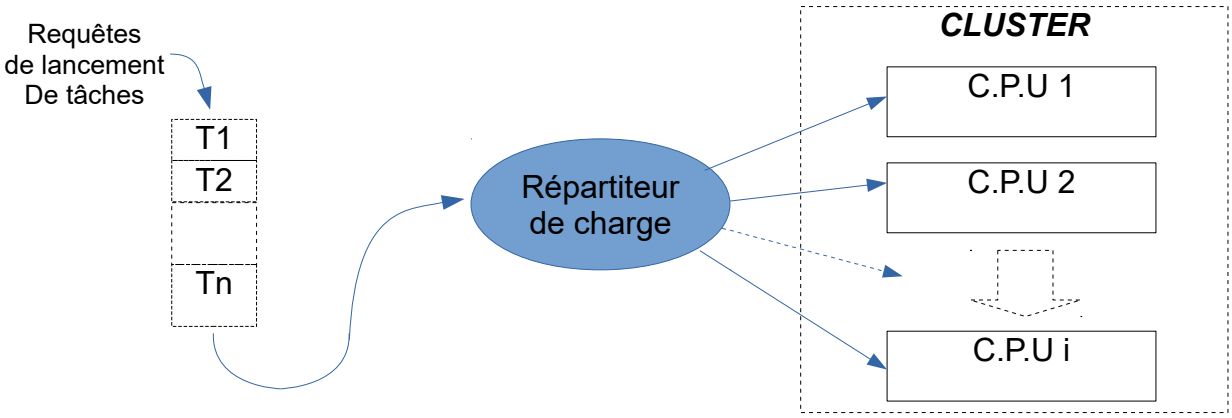
- Soit en les répartissant sur plus de machines (GRAPPES ou CLUSTERS de machines);

- Soit en augmentant le nombre de CPU disponibles dans la ou les machines (machines MULTIPROCESSEURS).

V.3.3.3.LES TECHNIQUES DE PARALLÉLISATION:

V.3.3.3.1.PRINCIPE GÉNÉRAL:

Le schéma ci-dessous modélise le principe général de la parallélisation des tâches:



COMMENTAIRES:

- Le schéma ci-dessus représente le traitement de requêtes adressées à un système constitué par une GRAPPE de CPUs. Chacune des requêtes empilées dans la file d'attente sollicite l'exécution d'une TÂCHE participant à la réalisation d'une des ACTIVITÉS de l'application. A ce niveau, une tâche peut être définie comme un "fil d'exécution" (un thread);
- La GRAPPE peut représenter soit des ORDINATEURS indépendants munis chacun de leur CPU, soit les CPU d'un SYSTÈME MULTIPROCESSEUR;
- L'entité RÉPARTITEUR DE CHARGE (LOAD BALANCING en anglais) est un LOGICIEL chargé d'attribuer l'exécution de chacune des tâches demandées aux CPUs de la grappe, suivant une logique sensée optimiser le fonctionnement du système. Ce logiciel peut être installé sur une machine dédiée ou réparti sur les machines formant la grappe.

V.3.3.3.2.PRINCIPAUX ALGORITHMES DE RÉPARTITION DE LA CHARGE:

Le tableau suivant explicite les principaux types d'algorithmes de répartition de la charge, les autres types n'étant que des combinaisons ou des variantes:

TYPE DE RÉPARTITION	DESCRIPTION	COMMENTAIRES
Répartition statique asymétrique	Chacune des tâches est attribuée à un processeur particulier de la grappe en fonction de sa nature et des logiciels qu'il supporte.	**Exemple 1**: attribuer toutes les tâches comportant des impressions de documents au processeur de la machine qui supporte ce périphérique (imprimante point à point); **Exemple 2:** attribuer les tâches comportant des calculs de graphisme 3D à un processus de la grappe à capacité superscalaire. **Remarque:** dans le cas d'une machine multiprocesseurs, un de ceux-ci est très souvent réservé à l'exécution de système d'exploitation.
Répartition statique circulaire (round robbin)	Les tâches sont attribuées successivement aux processeurs de la grappe dans un ordre immuable: processeur 1, processeur 2, …. , processeur n, rebouclage au processus 1, etc.,	Les différents nœuds de la grappe doivent supporter les mêmes logiciels. Éventuellement, l'état des nœuds de la grappe peut être testé avant de leur attribuer des tâches. Les nœuds hors service peuvent alors être exclus du "round robbin" jusqu'à leur remplacement ou leur remise en état de marche. Lorsque l'état des nœuds est ainsi testé, ce type d'algorithme participe à la sécurisation du fonctionnement.
Répartition "dynamique" en fonction des charges courantes des processeurs	Chaque tâche est attribuée au nœud disponible "le moins chargé" de la grappe.	Ce fonctionnement tend à égaliser les charges entre les CPUs. Il implique un dialogue assez fourni entre le "load balancer" et les processeurs. Comme précédemment, les différents nœuds de la grappe doivent supporter les mêmes logiciels. Lorsque l'état des nœuds est ainsi testé, ce type d'algorithme participe à la sécurisation du fonctionnement.

COMMENTAIRES:

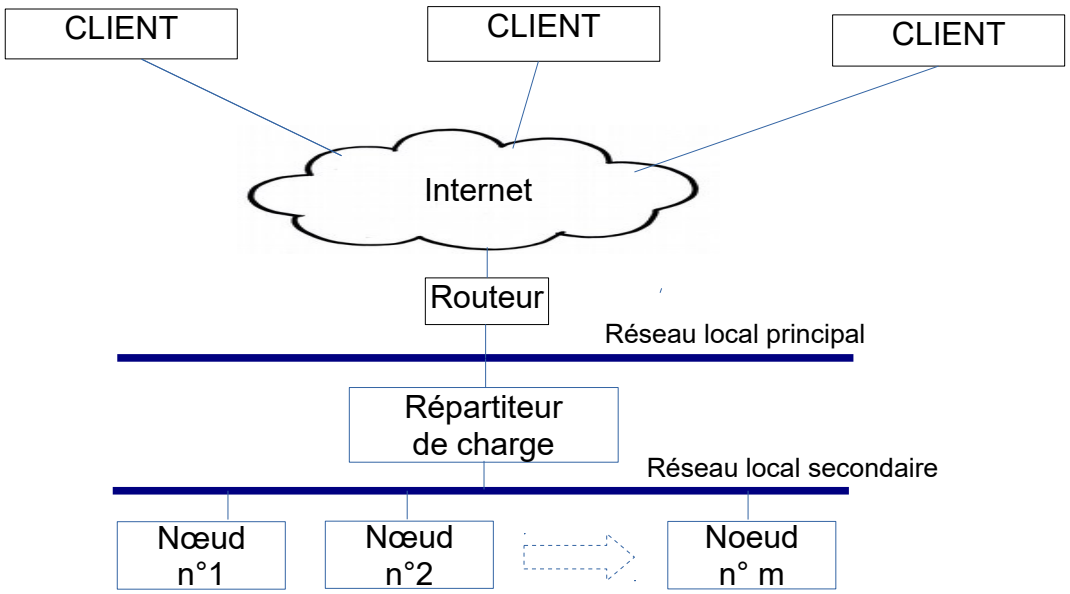
- Dans le cas de la répartition statique asymétrique, cette répartition est faite à priori. De ce fait, l'algorithme du répartiteur de charge est très simple. Le principal inconvénient est que l'indisponibilité d'un des éléments de la grappe entraîne l'indisponibilité de tout le système. D'autre part, la parallélisation des traitements est loin d'être optimale;
- Dans le cas de la répartition statique circulaire, la parallélisation n'est optimale que si toutes les tâches ont des temps d'exécution assez semblables. Ceci impliquerait de morceler, au niveau de la programmation, les traitements en tâches parallélisables de durée d'exécution très semblables. Ceci est très difficile à réaliser pour tous les traitements, car certains d'entre eux peuvent contenir des itérations variables en fonction du contexte ou des temps d'attente (d'une ressource, d'un signal, etc.). Ce type de répartition présente plusieurs variantes, et en particulier le WORK-STEALING (vol de travail) qui autorise un processeur peu chargé à s'approprier des tâches attribuées à des processeurs en surcharge;
- Le cas de la répartition "dynamique" en fonction des charges courantes des processus est en théorie le plus à même de répartir la charge également. Cependant, ce type de répartition ne peut être adopté intégralement que si tous les CPU de la grappe sont capables d'effectuer toutes les taches à répartir. Il faut, en particulier, que les entrées-sorties concernées par ces tâches se fassent uniquement par un réseau auquel toutes les machines du cluster sont connectées;
- Plus un algorithme de répartition tient compte des états courants des CPU, plus le trafic entre le répartiteur de tâches et la grappe qu'il contrôle est important. De même, plus l'algorithme de répartition est sophistiqué, plus il consomme de la puissance CPU. Le danger est que le système finisse par annihiler les gains dus à la parallélisation par une surcharge due à la gestion de la répartition.

De ce qui précède, nous pouvons facilement déduire que les solutions de répartition sont le plus souvent des HYBRIDES, combinant au moins deux des trois types présentés plus haut. Par exemple, les tâches peuvent être réparties suivant la charge courante des CPU, sauf celles qui, du fait des types de traitements qu'elle exécutent (entrées-sorties, capacités superscalaires, etc.), ne peuvent être exécutées que sur certains de ces CPUs.

V.3.3.3.3.SUPPORT MATÉRIEL DU RÉPARTITEUR DE CHARGE:

Architecture générale:

Le RÉPARTITEUR DE CHARGE (ou LOAD BALANCER) est un LOGICIEL. Celui-ci peut être installé sur une machine spécifique, comme le suggère le schéma ci-dessous:



Fonctionnement:

- ***Réception des requêtes:*** Le répartiteur reçoit les requêtes émises par les clients de l'application. Il se charge de router celles-ci vers le nœud de la grappe qu'il à choisi pour l'exécuter (en fonction de l'algorithme de répartition choisi);
- ***Réponses aux requêtes:*** il existe deux solutions:
  - ***Network Address Translation:*** Les nœuds de la grappe retournent leurs réponses au répartiteur de charge qui se charge de les faire suivre aux différents clients. Cette technique a l'avantage de la simplicité mais elle oblige le répartiteur à assumer toute la charge du trafic réseau;
  - ***Direct Routing:*** Dans ce cas, le répartiteur relaie les requêtes entrantes vers les nœuds en remplaçant dans ces requêtes sa propre adresse par l'adresse du client. Le nœud répond alors directement au client qui l'a sollicité. Cette solution allège considérablement le trafic réseau du répartiteur de charge.
- ***Échanges entre répartiteur et nœuds***: indépendamment du trafic concernant le traitement des requêtes, le répartiteur entretient avec chacun des nœuds des échanges de messages dont les finalités peuvent être les suivantes:
  - Détection des nœuds défaillant afin de les exclure de la répartition;

- Message d'un nœud vers le répartiteur pour lui signaler la fin de traitement d'une requête;
- Échanges entre le répartiteur et un nœud pour signaler la charge instantanée de ce nœud;
- Etc.

- ***Reprises en cas de pannes:*** Ce traitement est supporté par le protocole Virtual Router Redundancy Protocol (VRRP).

REMARQUES:

- Il est possible d'installer le logiciel répartiteur sur une des machines de la grappe de serveurs, cette machine hébergeant en même temps le logiciel d'application. Cette solution économique possède toutefois le désavantage de mélanger les flux de données entre clients et serveurs et les messages de service entre répartiteur et nœuds;
- Dans tous les cas, le répartiteur de charge est installé en FRONTAL (on dit aussi en PROXY): pour atteindre les nœuds de la grappe, les clients n'ont besoin de connaître que l'adresse IP du répartiteur: leurs requêtes lui sont toutes adressées. Une telle architecture est parfois appelée VIRTUALISATION DES SERVEURS. En effet, le répartiteur de charge permet au client de faire abstraction de la grappe de serveurs pour ne "voir" qu'un seul serveur "virtuel".

V.3.3.4.APPLICATION DES TECHNIQUES DE PARALLÉLISATION:

V.3.3.4.1.INTRODUCTION:

Les techniques de répartition de charges que nous avons exposées précédemment correspondent à ce qui est dit sur les sites des fournisseurs de "load balancers". Il est assez facile de se rendre compte que ces techniques ciblent en premier lieu le domaine des applications interactives. En effet:

- Le problème traité concerne la répartition de traitements initiés par des REQUÊTES adressés par des CLIENTS à des grappes de SERVEURS;
- Chaque requête émise par un client déclenche une activité logicielles dont l'exécution est indépendante (du point de vue de la précédence) des activités relatives aux autres requêtes en cours: de ce fait, leurs exécutions respectives peuvent être confiées à des machines différentes sans nécessiter d'interaction entre ces machines;

Ces particularités sont caractéristiques des applications à structure CLIENT-SERVEUR comme les sites web, les services FTP, la messagerie électronique ou les logiciels de bureautique accessibles sur le cloud.

En revanche, elles s'adaptent assez mal à des traitements qui ne peuvent pas s'assimiler à un fonctionnement client-serveur. C'est le cas notamment des applications de conduite de processus.

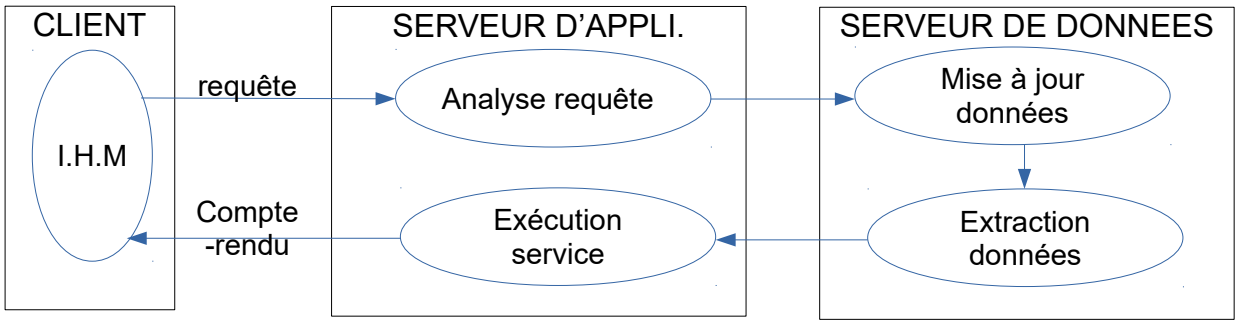
Nous allons dans un premier temps étudier le cas des applications interactives, puis nous aborderons les techniques qu'il convient d'appliquer pour les autres types d'applications.

V.3.3.4.2.APPLICATION AUX TRAITEMENTS À CARACTÈRE INTERACTIF:

CARACTÉRISATION DES TRAITEMENTS A CARACTÈRE INTERACTIF:

Ce sont des traitements initiés par une commande déclenchée par un OPÉRATEUR HUMAIN et aboutissant à l'exécution d'un service avec retour d'un compte-rendu à CE MÊME OPÉRATEUR. Ce fonctionnement est typiques des applications CLIENTS-SERVEUR telles que les sites web, les serveurs ftp ou les serveurs de mails.

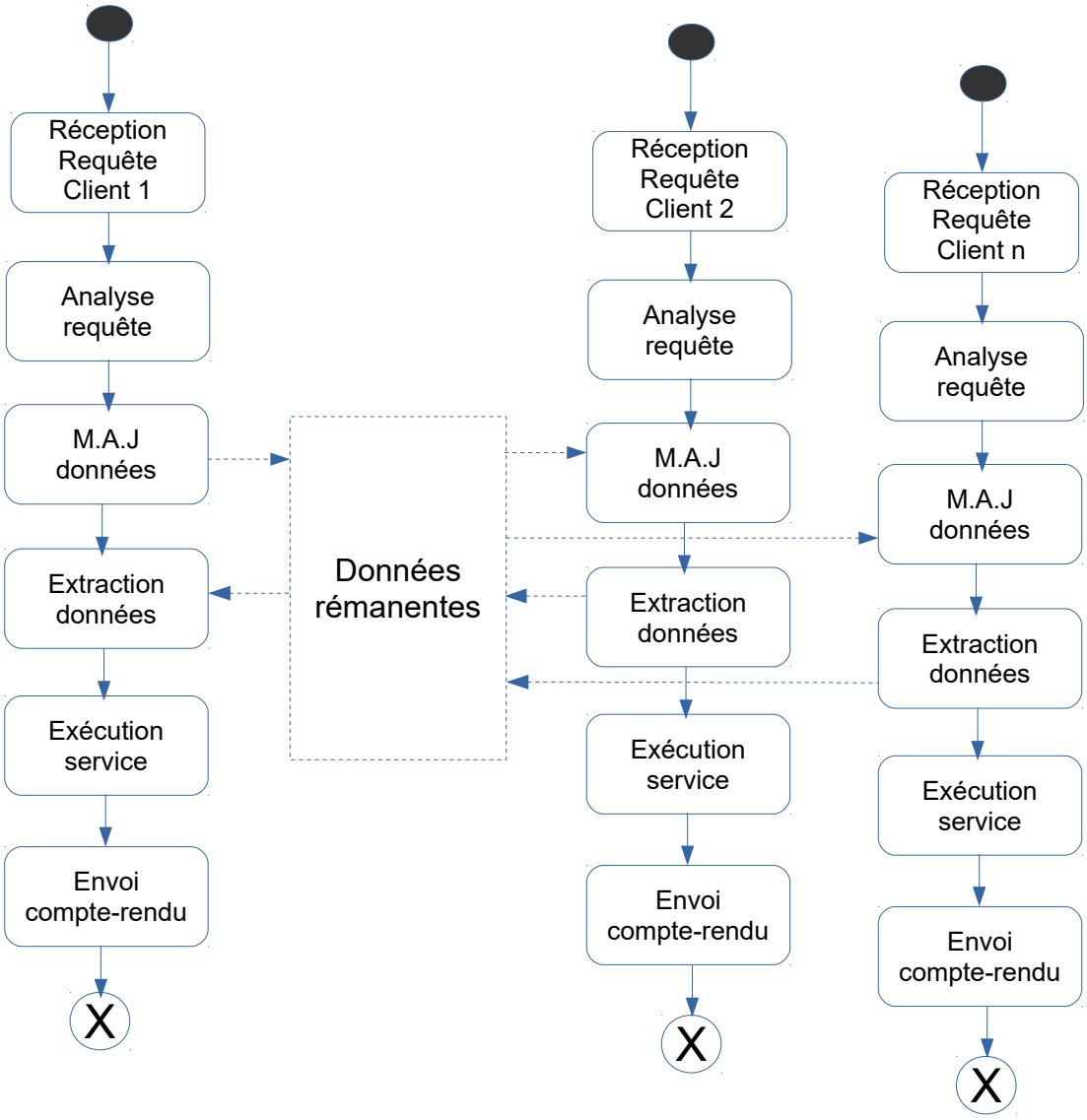
Le schéma ci-après décrit le fonctionnement général de ce type d'application:



COMMENTAIRES:

- Chaque requête issue de l'IHM utilisateur déclenche une activité. Ces activités sont indépendantes l'une de l'autre du point de vue procédural: elles sont donc parallélisables (parallélisme réel ou apparent);
- Les temps de réponse sont peu contraints et peuvent varier largement (le temps de réaction d'un opérateur humain est de l'ordre de la seconde). Cependant un délais de réponse trop long est souvent rédhibitoire (une réponse supérieure à 3 seconde à l'appel d'une page web entraîne très souvent l'abandon de la session par le visiteur);
- Les différentes activités initiées par les utilisateurs sont en concurrence pour l'accès aux données persistantes (serveur de données). Cependant, si cette concurrence peut allonger les temps de réponse, elle n'a pas d'autre influence sur le rendu du service.

Le diagramme d'activité général peut être représenté comme suit:



SOLUTIONS DE PARALLÉLISATION:

Nous voyons que les activités du diagramme précédent peuvent être exécutées:

- Soit au niveau d'un seul CPU, en parallélisme apparent (exécution en temps partagé);
- Soit en parallélisme réel, réparties entre divers CPU d'un système multiprocesseur;
- Soit en parallélisme réel, réparties entre plusieurs machines physiques.

L'algorithme du répartiteur de charges peut se contenter d'une répartition statique circulaire (round robbin), mais une répartition "dynamique" en fonction des charges courantes des nœuds de la grappe peut optimiser nettement les temps de réponse

moyens dans le cas où l'exécution de certaines requêtes d'utilisateurs peu durer beaucoup plus que la durée moyenne des requêtes.

EXEMPLE DE PARALLÉLISATION D'UNE APPLICATION INTERACTIVE:

Supposons qu'un serveur web soit supporté par une seule machine. Cette machine traite nécessairement la totalité des requêtes HTTP en provenance des différents visiteurs du site. Chacune de ces requêtes déclenche dans le serveur une activité dont le but est de créer une page web et de l'émettre vers le poste du visiteur. Ces différentes activités sont indépendantes les une des autres du point de vue procédural.

Supposons maintenant que la fréquentation du site augmente à un point tel que la surcharge du CPU de la machine entraîne un ralentissement trop important du traitement des requêtes. Pour résoudre ce problème, trois solutions peuvent être envisagées:

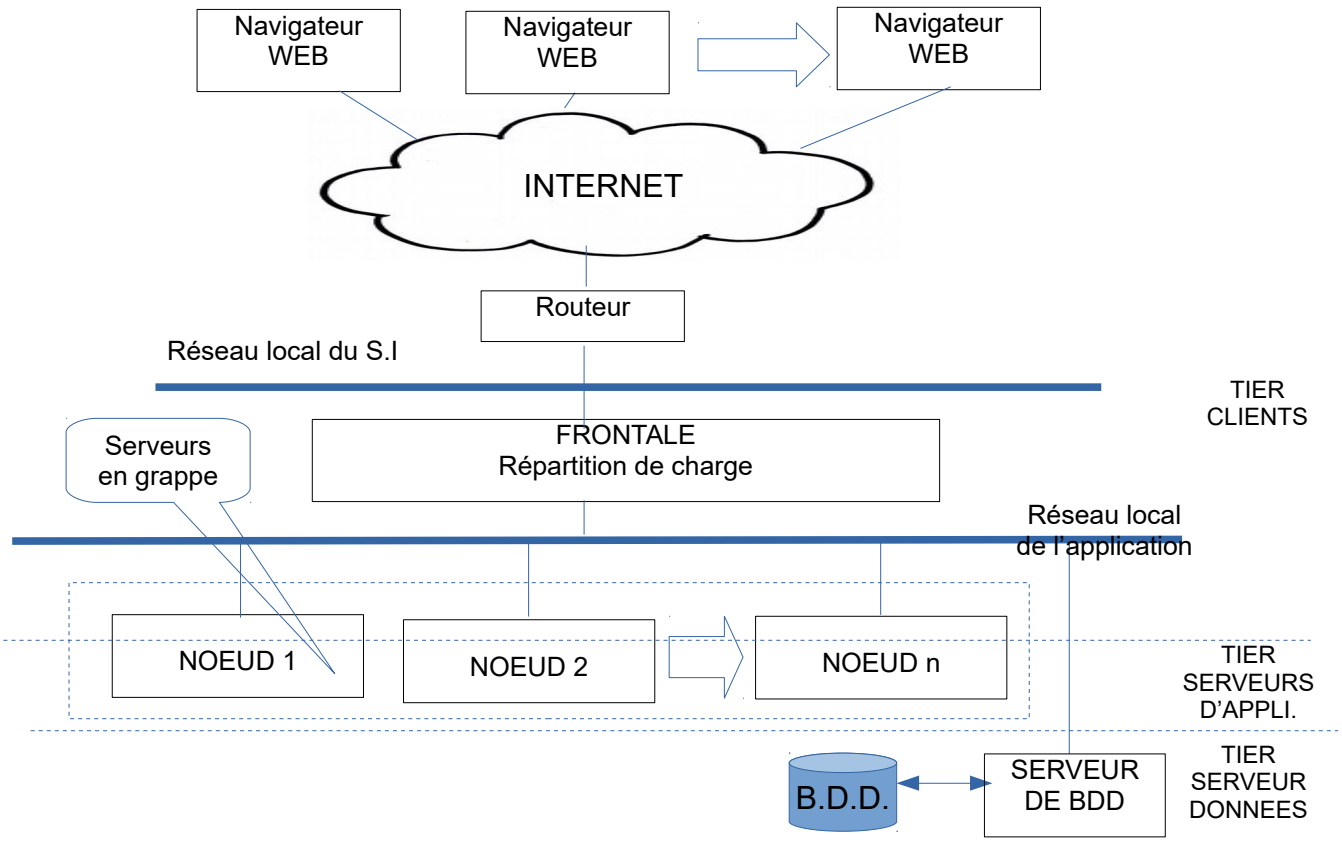
1. Le remplacement de la machine serveur par une machine dotée d'une plus grande puissance CPU (nombre de cœurs plus élevé, cœurs plus puissants, plus de mémoire vive, etc.);
2. Le remplacement de la machine serveur par une machine équipée de plus de processeurs (multiprocesseurs);
3. La répartition des activités sur plusieurs serveurs (une GRAPPE de serveurs, encore appelée CLUSTER en anglais).

ANALYSE DES SOLUTIONS:

- La première solution est évidemment la plus simple à réaliser, mais pour qu'elle constitue une solution à long terme, il faut que la nouvelle machine soit très nettement plus puissante que l'ancienne, donc beaucoup plus chère. D'autre part, l'ancienne machine n'aura plus d'utilité.
- La deuxième solution (ajout de processeurs) présente l'avantage d'être plus évolutive que la première, à condition que la machine existante puisse accueillir au moins un processeur supplémentaire. Cependant, à puissance égale, ce type de machine coûte beaucoup plus cher qu'une machine monoprocesseur.
- La troisième solution (constitution d'une grappe de serveurs) a pour avantage de coûter relativement peu cher (il est possible de constituer la grappe avec l'ancien serveur auquel on ajoutera des machines de puissance équivalente, donc peu chères). Elle est donc également très évolutive.

C'est pour ces raisons que la constitution d'une grappe de serveurs est la solution le plus souvent adoptée pour les applications de type interactif (on parle quelquefois à son propos de VIRTUALISATION des serveurs)

L'architecture physique correspondant à cette troisième solution pourrait ressembler au schéma suivant:



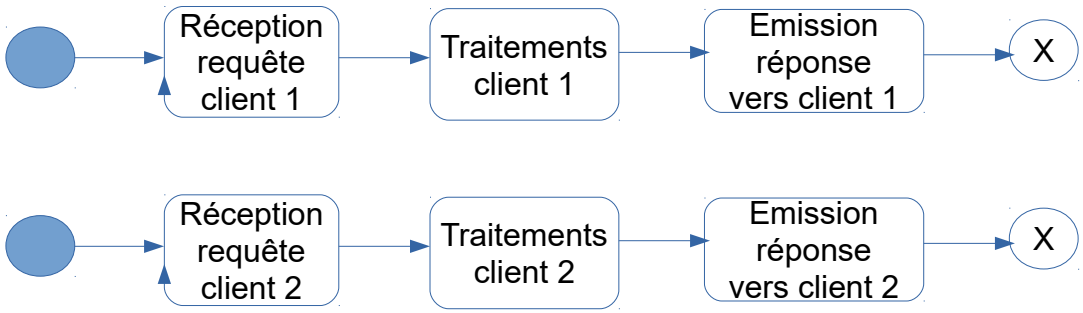
COMMENTAIRES:

- La machine "FRONTALE" héberge le répartiteur de charge. C'est la seule qui est reliée directement au réseau local du Système Informatique d'accueil. De ce fait, le trafic interne de l'application ne perturbe pas le trafic du S.I. (et inversement);
- Les serveurs de la grappe doivent héberger le même logiciel applicatif. Il est souhaitable que ces serveurs soient assez équivalents du point de vue de la capacité mémoire et de la puissance CPU;
- Les serveurs en grappe communiquent avec le frontal et avec le serveur de données par un réseau propre à l'application. En fonction du trafic interne prévisible, ce réseau peut être choisi plus ou moins performant;
- Une telle architecture peut facilement être adaptée à une augmentation du nombre des visiteurs: il suffit d'ajouter des serveurs dans la grappe.

V.3.3.4.3.AUTRES TYPES DE TRAITEMENTS:

INTRODUCTION:

Nous avons vu au paragraphe précédent que les traitements du type CLIENT-SERVEURS peuvent être décrits comme un ensemble d'activités se déroulant parallèlement l'une de l'autre sans relation de précédence ou de synchronisation entre elles. Ces activités sont initiées par un CLIENT et leurs résultats sont destinés à ce CLIENT uniquement:



→ Etc.

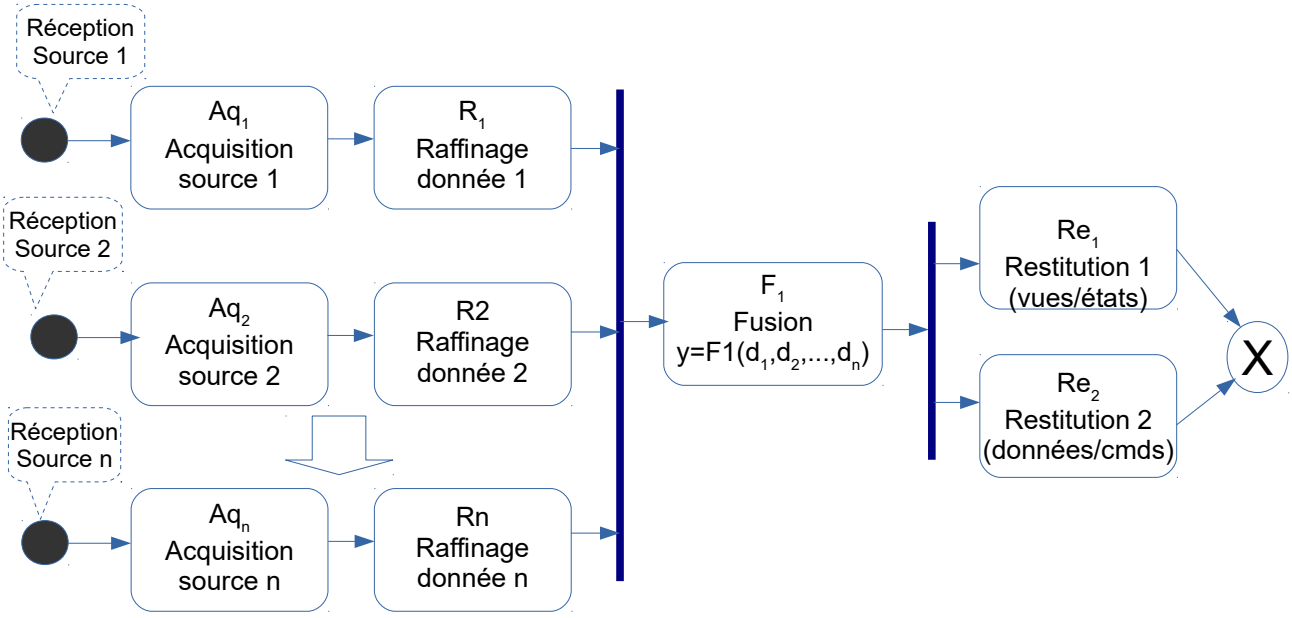
Cependant, dans beaucoup d'autres applications, le diagramme d'activités ne peut pas se résumer à ces chaînes de traitement parallèles.

Nous pouvons citer en particulier les applications de conduite de processus industriels dont la finalité principale n'est pas de fournir des services à des postes clients indépendants les uns des autres, mais bien de CONTRÔLER EN TEMPS RÉEL le déroulement d'un processus et, en fonction des valeurs des paramètres de fonctionnement observés, émettre vers ce processus les COMMANDES qui s'imposent pour corriger les dérives ou modifier les consignes de fonctionnement.

L'exemple qui suit est caractéristique de ces types de traitement:

TRAITEMENT DU TYPE "FUSION DE DONNÉES":

Ce type de traitement est caractérisé par le fait que les informations acquises par un certain nombre de sources sont combinées entre elles pour fournir une (ou des) données synthétiques qui vont être restituées à l'environnement sous forme d'états ou de vues destinées à des utilisateurs humains ou sous forme de commandes destinées à des équipements. Le diagramme d'activité suivant est caractéristique d'un traitement de fusion de données:



COMMENTAIRES:

- Le diagramme d'activité représente une application exploitant n sources de données et exécutant une activité de fusion de ces données;
- Les différentes sources de données peuvent être des capteurs physiques ou des commandes ou paramètres fournis par des IHM;
- Les événements de réception des données afférentes aux différentes sources peuvent survenir de façon totalement asynchrone;
- Les activités de raffinage de données consistent à appliquer à ces données divers algorithmes permettant de changer de systèmes de coordonnées, de référentiel ou d'unités, de filtrer les valeurs aberrantes, de lisser les évolutions, etc;
- Les activité de restitution peuvent consister à afficher des vues ou des états sur un IHM ou bien à fournir à des équipements physiques extérieurs des données ou des commandes influant sur leur comportement;
- Les différentes branches d'activités correspondant au traitement des données acquises (Aq_1,Aq_2,...,Aq_n) ainsi que les différentes activités de restitution (Re_1, Re_2,...,Re_n) peuvent être parallélisées;
- Le début des activités de fusion est conditionné à la fin d'exécution de l'ensemble des activités d'acquisition et de raffinage qui les concernent et leur fin conditionne le démarrage des activités de restitution qui sont concernées par leurs résultats;

CONSÉQUENCES SUR LA GESTION DE LA RÉPARTITION:

Les algorithmes de répartition des activités et de parallélisation de leur exécution sont donc plus complexes pour ce type d'application que dans le cas des applications interactives car ils doivent tenir compte des dépendances procédurales existant entre ces activités. En effet, pour que l'exécution d'une activité T puisse être distribuée à un nœud donné, il ne suffit pas que ce nœud soit en état de la traiter: il faut en plus que toutes les activités qui ont un lien de précédence avec T soient terminées. Par exemple, dans le diagramme précédent:

- Quel que soit i, l'activité Aq_i a un lien de précédence avec l'activité Ri. De ce fait, la condition pour que l'activité Ri soit lancée est que Aq_i soit terminée;
- Quel que soit i, les activités Ri ont toutes un lien de précédence avec F_1. De ce fait, la condition pour que l'activité Fi soit lancée est que toutes les activités R_i soient terminées.

SOLUTION POUR L'ALGORITHME DE RÉPARTITION:

La solution algorithmique proposée ci-après consiste à exploiter un tableau dont voici un exemple (adapté à l'application de fusion de données dont le diagramme d'activité est donné plus haut):

TACHE	Identificateur	PRÉCÉDENCES	ÉTAT (Terminé/Non Terminé)
Aq_1	1	Init	NT
Aq_i	I	Init	NT
Aq_n	n	Init	NT
R_1	n+1	Aq_1	NT
R_j	n+j	Aq_i	NT
R_n	n+n	Aq_n	NT
F_1	2*n+1	R_1, ….., R_i, ….., R_n	NT
Re_1	2*n+2	F_1	NT
Re_2	2*n+3	F_1	NT

L'algorithme exploitant ce tableau est détaillé ci-dessous (les structures QUAND < événement> … FAIRE … FINFAIRE modélisent des gestionnaires d'événement):

```
QUAND Le signal général de début d'activité est reçu FAIRE
        Marquer toutes les tâches à l'état "non terminées" dans le tableau;
        TANT QUE ( Il existe des taches non terminées dans le tableau ) FAIRE
                I = 1;
                TANT QUE ( I<2*n+4) FAIRE
                        SI ( toutes les précédentes de la tache I sont terminées ) ALORS
                                ROUTINE Distribuer l'exécution de la tâche I sur un nœud;
                        FINSI
                        I = I+1;
                FIN FAIRE
        FINFAIRE
FINFAIRE

QUAND La tache J est signalée terminée par un nœud FAIRE
        Marquer la tâche J à l'état "terminée" dans le tableau;
FINFAIRE
```

REMARQUES:

- Le tableau et l'algorithme présentés ci-dessous peuvent être adaptés à la plupart des types de traitements complexes.
- Le tableau et l'algorithme présentés ci-dessous conviennent aussi bien pour une répartition sur plusieurs machines que pour la répartition sur plusieurs CPU d'une même machine (mono machine multiprocesseurs).
- La routine "Distribuer l'exécution de la tâche I sur un nœud" correspond au travail d'un répartition de la charge "classique". L'algorithme choisi peut être le "round robbin" statique, le "round robin" avec prise en compte du niveau de charge des nœuds ou tout autre algorithme de répartition.

*V.3.3.4.4.**ACCÈS AUX DONNÉES COMMUNES PERSISTANTES:***

Les différentes tâches qui s'exécutent dans le cadre d'une application ont besoin de partager un certain nombre de données persistantes.

Lorsque ces tâches s'exécutent dans une même machine, il existe trois moyens de partager ces données:

1. Soit les conserver dans une base de données;
2. Soit les conserver dans des fichiers "à plat" situés dans des unités de stockage (attachées localement ou par réseau);
3. Soit recourir à un système de partage de la mémoire vive (sheared memory). Les mémoires partagées constituent en fait des "fichiers en mémoire".

Dans les deux premiers cas, les temps d'accès aux données sont de l'ordre de quelques ms. Dans le troisième cas, ils sont de l'ordre d'un accès en mémoire vive. La solution n°3 est surtout utilisée pour les applications à contraintes "temps réel" (ces durées ne prennent pas en compte les temps d'attente dus à une éventuelle indisponibilité de la ressource).

Lorsque ces tâches sont réparties sur des machines différentes, seules les deux premières solutions sont disponibles. Les fichiers ou bases de données à partager doivent être mutualisés (grâce aux différentes techniques abordées dans l'étude des couches "données" ou aux mécanismes de partage de fichiers inclus dans les systèmes windows ou linux.

REMARQUES:

- Les temps d'accès peuvent se retrouver majorés dans de grandes proportions par ces divers mécanisme et surtout devenir extrêmement aléatoires;
- Les données persistantes des sites web sont la plupart du temps hébergées dans des bases de données.

V.4.ÉTUDE DE LA COUCHE "DONNÉES":

V.4.1.INTRODUCTION:

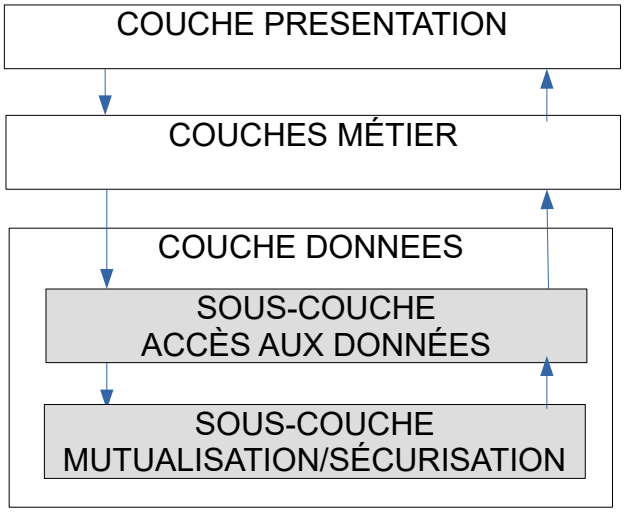
La couche DONNÉES encapsule les modules logiciels qui permettent d'accéder aux données persistantes des applications en lecture et en écriture. Ces données sont stockées dans des mémoires de masse sous la forme de FICHIERS À PLAT ou de BASES DE DONNÉES (qui sont des ensembles de fichiers organisés entre eux).

Les machines hébergeant les logiciels de la couche DONNÉES peuvent donc être appelées, selon le cas: SERVEURS DE FICHIERS ou SERVEURS DE BASES DE DONNÉES, mais rien n'empêche qu'elles jouent les deux rôles à la fois.

REMARQUE: *Les mémoires de masse peuvent être connectées directement aux machines hébergeant la couche données (par des liaisons privées point à point). Elles peuvent également être accessibles par réseau à ces machines.*

V.4.2.LES DEUX SOUS-COUCHES DE LA COUCHE "DONNÉES":

Afin de mieux appréhender les solutions de répartition de la couche données, nous pouvons la diviser en deux sous-couches: une sous-couche "ACCÈS AUX DONNÉES" et une sous-couche "MUTUALISATION/SÉCURISATION". Le schéma suivant situe ces deux sous-couches dans l'architecture générale:



V.4.2.1.SOUS-COUCHE "ACCÈS AUX DONNÉES":

Nous appellerons "ACCÈS AUX DONNÉES" la sous-couche supérieure de la couche DONNÉES car c'est elle qui met à disposition de l'application les modules logiciels qui permettent de lire ou d'écrire les données attachées aux entités logicielles traitées par les couches MÉTIER. Ainsi, ces modules permettront de sélectionner des lignes dans un fichier texte ou des utilisateurs d'un certain type dans une table d'une base de données.

V.4.2.2.SOUS-COUCHE "MUTUALISATION ET SÉCURISATION":

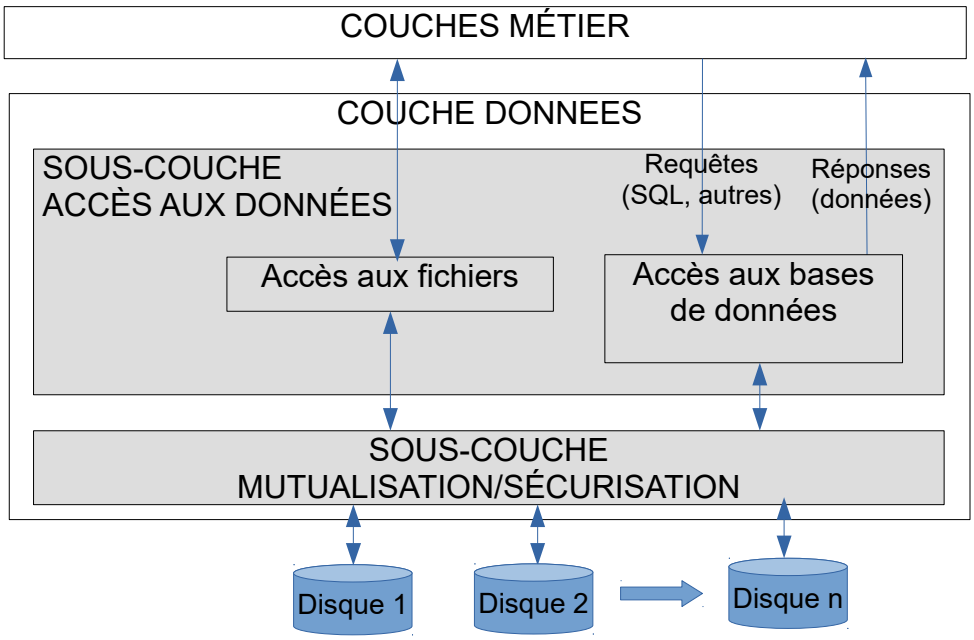
Nous appellerons "MUTUALISATION ET SÉCURISATION" la sous-couche inférieure de la couche DONNÉES car c'est cette sous-couche qui permet de présenter à la sous-couche supérieures une interfaces réalisant une abstraction de la structure physique du système de stockage et de la répartition des blocs de données sur les supports physiques.

En effet, si le système de stockage se compose souvent de disques durs utilisés indépendamment les uns des autres, il peut aussi, dans certains cas "critiques", être constitués de plusieurs supports de stockage "mutualisés" et "sécurisés" dans lesquels les blocs de données sont répartis sur plusieurs volumes de façon à créer des redondances (technologies NAS, SAN, RAID, clusters de bases de données partagées, etc.).

Grâce à cette sous-couche, la sous-couche ACCÈS AUX DONNÉES "voit" donc le système de stockage comme un seul espace physique et fait abstraction de ces problèmes de répartition et de redondance des blocs de données:

V.4.2.3.RÉPARTITION DE CES DEUX SOUS-COUCHES:

La sous-couche ACCÈS AUX DONNÉES héberge les modules logiciels qui permettent aux couches MÉTIER d'accéder aux données persistantes dont elles ont besoin. Ces données sont deentités "logiques". Ces modules utilisent pour cela les logiciels de la couche MUTUALISATION ET SÉCURISATION qui, eux, permettent d'établir un lien entre ces entités logiques et les espaces physiques de stockage:



La sous-couche ACCÈS AUX DONNÉES est donc hébergée par les machines qui constituent le TIER SERVEUR DE DONNÉES.

En revanche, la sous-couche "MUTUALISATION ET SÉCURISATION" peut soit être hébergée dans le TIER SERVEUR DE DONNÉES (c'est le cas, par exemple, d'un RAID Logiciel), soit déportée sur un composant jouant le rôle de serveur de fichiers (par exemple, un NAS) ou de serveur de bases de données (par exemple, un SQL CLUSTER).

Le choix de cette répartition dépend de certains critères que nous détaillons ci-après

CRITÈRE	ÉNONCÉ
TEMPS D'ACCÈS	Les temps d'accès aux données doivent être compatibles avec les contraintes imposées par les exigences des utilisateurs.
ÉVOLUTIVITÉ DE LA CAPACITÉ DE STOCKAGE	Les capacités du système de stockage doivent, bien sûr, être suffisantes pour permettre le bon fonctionnement l'application. Il est également très important que cette capacité puisse évoluer facilement pour s'adapter aux besoins nouveaux.
SÉCURITÉ DE FONCTIONNEMENT	La résistance aux pannes du système de stockage doit satisfaire les exigences imposées par les utilisateur et le niveau de criticité de l'application. Cette résistance aux pannes dépend: • De la fiabilité des supports de stockage; • Des redondances qui peuvent être établies; • De la rapidité de la remise en état de ces supports en cas de panne.
SÉCURITÉ D'ACCÈS AUX DONNÉES	Le degrés de sécurisation de l'accès aux données doit être compatible avec les exigences imposées par les utilisateur et le niveau de criticité de l'application.
COÛT DE RÉALISATION	Le coût de réalisation doit être compatible avec les ressources financières et humaines affectées à la conception et à la réalisation de l'application, mais aussi à son exploitation et à sa maintenance

La solution sera forcément un compromis entre ces différentes exigences.

V.4.3.SOLUTIONS D'ATTACHEMENT DES SUPPORTS DE STOCKAGE:

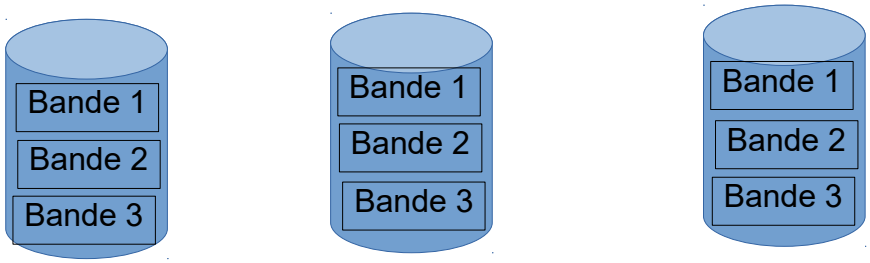
V.4.3.1.RAPPEL-LA TECHNOLOGIE RAID:

Cette technologie permet de constituer une unité de stockage à partir de plusieurs disques durs. Suivant la solution de répartition des informations sur les supports, l'unité ainsi créée (appelée grappe ou cluster) peut avoir pour avantage:

- Soit une plus grande tolérance aux pannes, grâce à la redondance des informations sur les différents supports;
- Soit une plus grande vitesse d'écriture ou de lecture grâce à la possibilité de lire les informations attachées à un bloc de données en parallèle sur plusieurs disques simultanément;
- Soit les deux en même temps.

Il existe plusieurs "niveaux" de RAID. Chacun de ces niveaux correspond à un modèle de répartition des données sur les différents supports de stockage pris en compte et correspond à des exigences différentes en ce qui concerne la vitesse d'accès aux données ou la sécurité de celles-ci.

EXEMPLE: l'exemple qui suit représente l'organisation physiques des données dans un RAID de niveau 1 (appelé MIRRORING):



Nous pouvons constater que chaque bloc de données est fragmenté en plusieurs BANDES (fragments). Chacune de ces bandes est stockée sur tous les disques du RAID: dans le schéma, la donnée A est constituée des Bandes 1, 2 et 3. Nous pouvons constater dans cet exemple:

- Que le RAID renforce considérablement la sécurité de fonctionnement (le système présenté ci-dessus peut résister au dysfonctionnement de 2 supports sur 3);
- Que le RAID peut diminuer considérablement les temps d'accès.. En effet, est possible de lire ou écrire en parallèle les bandes appartenant à un même bloc de données.

V.4.3.2.LES TROIS SOLUTIONS D'ATTACHEMENT:

Nous pouvons distinguer principalement trois types de solutions pour assurer la connexion physique d'un support de stockage à une machine:

1. L'attachement direct, qui consiste à relier directement chacune des mémoires de masse aux machines de traitement par un B.U.S spécifique (SATA, PATA, USB, etc) accédant directement à la carte mère de la machine par l'intermédiaire de son bus d'entrée-sorties;
2. L'attachement par réseau (Network Attached Storage ou N.A.S), qui consiste à utiliser un serveur de stockage en réseau ou NAS (Network Attached Server);
3. L'attachement par un réseau de stockage (Storage Area Network ou S.A.N). Un SAN est un réseau à haut débit (fibre channel, iSCSI, etc.) qui est utilisé uniquement pour relier les unités de stockage aux unités de traitement (serveurs de données).

Quel que soit le type d'attachement, la technologie RAID (Redundant Array of Independent Disks) est très souvent utilisée pour le stockage des blocs de données.

V.4.3.3.L'ATTACHEMENT DIRECT (DIRECT ATTACHED STORAGE OU D.A.S):

V.4.3.3.1.PRÉSENTATION:

Celui-ci consiste à relier directement chacune des mémoires de masse à la machine par un B.U.S spécifique (SATA, PATA, USB, etc) accédant directement à la carte mère de la machine par l'intermédiaire de son bus d'entrée-sorties.

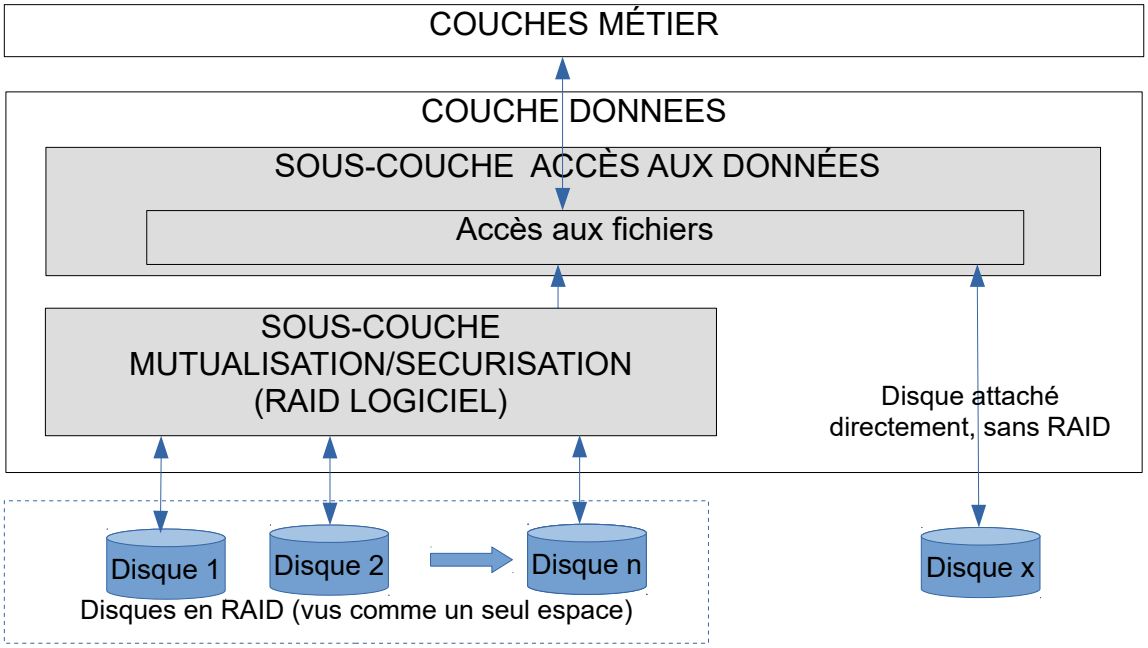
A lui seul, ce type d'attachement ne supporte aucun dispositif permettant la mutualisation des espaces de stockage ou la sécurisation des données en cas de panne d'une des mémoires de masse. Il faut donc que l'application elle-même les prenne en compte.

Il existe cependant des solutions logicielles "clef en main" pour mutualiser ces espaces disques et sécuriser les données en cas de crash d'un des supports de stockage: ce sont les RAID "Logiciels":

DÉFINITION:

Un RAID LOGICIEL est une fonction logicielle du système d'exploitation qui permet de construire une "grappe virtuelle" de supports de stockage reliés en attachement direct à l'unité de traitement et de gérer le stockage des fichiers dans la grappe suivant la technologie RAID. Ces disposition réalisent de fait une mutualisation des espaces de stockage et sécurisent les données grâce à l'exploitation des redondances. (les POOLS DE DISQUES, option supportée par les dernières versions de Windows constituent de fait des RAID).

Lorsqu'un RAID logiciel est utilisé pour gérer un "pool" de supports de stockage en attachement direct, ces supports sont vus par les utilisateurs comme un seul espace.



*V.4.3.3.2.**ANALYSE DE LA SOLUTION:***

- ***TEMPS D'ACCÈS:*** D'un manière générale, l'attachement direct est le mode de connexion le plus rapide. En effet, les transferts de données n'utilisent aucun équipement intermédiaire entre le support de stockage et l'unité de traitement. L'emploi du RAID (surtout le RAID 1), réduit encore la durée des échanges en permettant la parallélisation des accès aux blocs de données;
- ***ÉVOLUTIVITÉ DE LA CAPACITÉ DE STOCKAGE:*** L'ajout d'unités de stockage est relativement difficile car il nécessite des interventions dans les systèmes d'exploitation des systèmes de traitement;
- ***SÉCURITÉ DE FONCTIONNEMENT:*** Elle est faible si l'on n'utilise qu'un seul support de données ou si l'on n'utilise ni la technologie RAID ni une technologie équivalente. Elle est excellente à partir de deux supports mutualisés gérés par RAID;
- ***SÉCURITÉ D'ACCÈS AUX DONNÉES:*** la sécurité d'accès aux données dépend entièrement des mesures de sécurité déployées sur la machine support des unités de stockage. En effet, les médias d'attachement sont difficilement accessibles depuis l'extérieur;
- ***COÛT DE RÉALISATION:*** il se réduit au coût des unités de stockage.

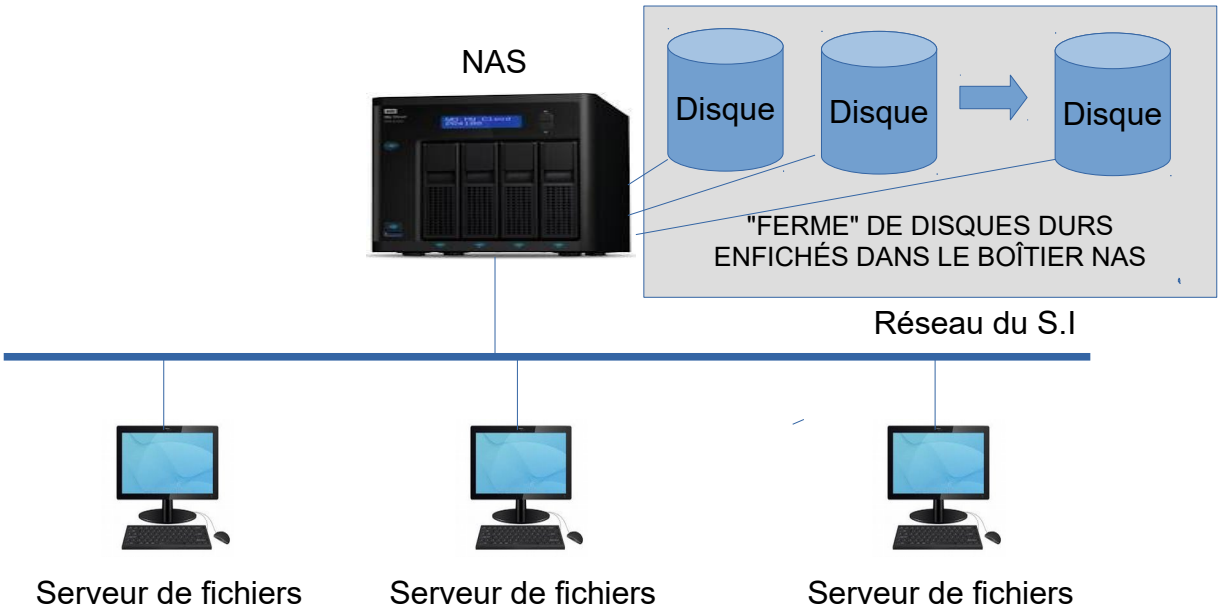
V.4.3.4.L'ATTACHEMENT PAR RÉSEAU (NETWORK ATTACHED STORAGE OU N.A.S):

V.4.3.4.1.PRÉSENTATION:

Cette solution consiste à utiliser un serveur de stockage en réseau ou NAS (Network Attached Server) Celui-ci est un équipement autonome relié au Système Informatique par l'intermédiaire du réseau de celui-ci et supportant un certain nombre d'unités de stockage (Grappes ou Clusters de disques durs) ainsi que d'un système d'exploitation supportant des fonctions logicielles permettant la mutualisation des espaces de stockage et la sécurisation en cas de panne (mutualisation et redondance des informations), mais aussi des fonctions "matérielles" telles que la possibilité de changer "à chaud" un disque en panne).

En général, un NAS peut être paramétré depuis un navigateur installé depuis n'importe quel poste de travail relié au réseau.

Du point de vue des clients qui l'utilisent, un NAS est vu comme un seul espace disque.



NOTA: Un serveur NAS peut intégrer la technologie de sécurisation RAID autorisant la défaillance d'un disque sans perte de données.

V.4.3.4.2.ANALYSE DE LA SOLUTION:

- ***TEMPS D'ACCÈS:*** Par rapport à un attachement direct, le temps d'accès est majoré par la vitesse de transmission sur le réseau du S.I, mais aussi par les éventuelles congestions de ce réseau, qui est partagé par toutes les machines de ce réseau. Il est, bien sûr, possible de créer un réseau indépendant pour accueillir le NAS, mais le coût s'en trouve augmenté.

- ***ÉVOLUTIVITÉ DE LA CAPACITÉ DE STOCKAGE:*** L'ajout d'unités de stockage est très facile tant qu'il existe des emplacements libres sur la baie NAS;
- ***SÉCURITÉ DE FONCTIONNEMENT:*** Elle est excellente: le fonctionnement des baies NAS est très sécurisé (Par exemple, leur alimentation peut être doublée). Certaines baies NAS permettent le remplacement à chaud d'une unité de stockage et la reconstitution automatique de son contenu;
- ***SÉCURITÉ D'ACCÈS AUX DONNÉES:*** Les NAS ont leur propre système de contrôle des accès. De ce fait, la sécurité des données n'est pas compromise par une compromission des machines clientes;
- ***COÛT DE RÉALISATION:*** Les NAS sont des composants peu coûteux (certains NAS peuvent être considérés comme des équipements "grand public").

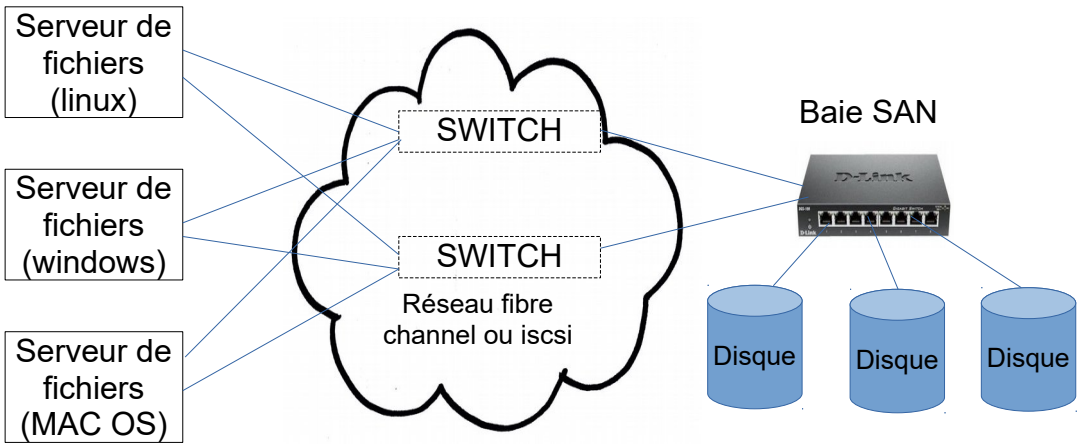
V.4.3.5.LES RÉSEAU DE STOCKAGE (STORAGE AREA NETWORK OU S.A.N):

V.4.3.5.1.PRÉSENTATION:

Un SAN est un réseau spécialisé permettant aux serveurs de fichiers de communiquer avec une baie SAN accueillant les unités de stockage et supportant la mutualisation de ces ressources.

Contrairement à un NAS, qui se connecte sur le réseau généraliste du système informatique, un SAN utilise un réseau spécifique pour communiquer avec les serveurs (en général, c'est un réseau très haut débit comme Fibre Channel ou iSCSI). Chaque serveur voit l'ensemble des espaces de stockage comme un espace unique.

Un réseau SAN utilise plusieurs équipements d'interconnexion. Les principaux sont les BAIES SAN qui supportent les disques mutualisés et les commutateurs réseau (SWITCHES SAN) qui permettent de constituer le maillage du réseau haut débit connectant les baies SAN aux serveurs de fichier par plusieurs chemins redondants (ce maillage est appelé "FABRIC"):



NOTA: il est également possible de sécuriser les données de la grappe de disques durs par la technologie RAID.

V.4.3.5.2.ANALYSE DE LA SOLUTION:

- ***TEMPS D'ACCÈS:*** Du fait de l'utilisation d'une solution à très haut débit pour le réseau de stockage, le temps d'accès par un SAN est en général plus faible que celui d'un NAS
- ***ÉVOLUTIVITÉ DE LA CAPACITÉ DE STOCKAGE:*** L'ajout d'unités de stockage est très facile: il suffit d'ajouter des unités sur les baies SAN existantes, ou sinon, d'ajouter des baies;

- ***SÉCURITÉ DE FONCTIONNEMENT:*** Elle est excellente: le fonctionnement des baies et commutateurs SAN est, comme les baies NAS, très sécurisé;
- ***SÉCURITÉ D'ACCÈS AUX DONNÉES:*** Les baies SAN ont leur propre système de contrôle des accès. De ce fait, la sécurité des données n'est pas compromise par une compromission des machines clientes;
- ***COÛT DE RÉALISATION:*** Les composants SAN sont très coûteux. De ce fait, les solutions SAN sont surtout utilisés pour de gros systèmes supportant des fonctionnalités critiques (big data).

V.4.3.6. CHOIX D'UNE SOLUTION D'ATTACHEMENT:

Le tableau ci-après peut être utilisé pour choisir une solution en fonction des critères définis au paragraphe précédent :

SOLUTION	TEMPS D'ACCÈS	ÉVOLUTIVITÉ CAPACITÉ DE STOCKAGE	FIABILITÉ	SÉCURITÉ ACCÈS AUX DONNÉES	COÛT DE RÉALISATION
Attachement direct (sans RAID logiciel)	Dépend du temps d'accès de chaque unité	Dépend de la capacité de chaque unité. Pas de mutualisation.	Faible (une panne d'une seule unité fait tomber le système).	Faible (repose sur la sécurité de la machine support).	Solution peu coûteuse (se réduit au coût de chaque unité de stockage)
Attachement direct (avec RAID logiel)	Le RAID permet de répartir les données de chaque bloc sur plusieurs supports, ce qui permet la parallélisation des accès.	- Permet de mutualiser les espaces de stockage. - Permet d'augmenter la capacité de stockage en ajoutant des supports.	Améliorée par rapport au précédent si le nbre de supports est suffisant et si le niveau de RAID est suffisant (au moins RAID 1).	Faible (repose sur la sécurité de la machine support et de l'application).	Identique à la solution précédente.
NAS	Temps d'accès forcément dégradé par rapport à un attachement direct, puisque les transferts se font à travers le réseau du S.I (qui est partagé avec les autres trafics du S.I) et la baie NAS.	Dépend des capacités d'accueil de la baie NAS. La capacité d'accueil est évolutive (L'ajout d'unité de stockage dans la baie est facile et rapide).	- Dépend du niveau de RAID supporté par la baie. A partir du niveau I, la sécurité est excellente. - Certaines baies NAS permettent le remplacement à chaud d'une unité de stockage défaillante.	Forte: les baies NAS supportent un système de contrôle des accès.	Moyen (un boîtier NAS n'est pas un matériel coûteux).
SAN	Le temps d'accès à travers un SAN est optimisé par rapport au NAS car le SAN utilise pour cele un réseau rapide (fiber channel, iscsi, etc.) distinct des autres réseaux du S.I.	Dépend des capacités d'accueil des baies SAN. La capacité d'accueil est très évolutive.	Dépend du niveau de RAID supporté par la baie. A partir du niveau I, la sécurité est excellente.	Forte: les baies SAN supportent un système de contrôle des accès.	Élevé (les baies et matériels de commutation SAN sont des matériels professionnels qui ont un coût très élevé). La solution SAN convient surtout à des systèmes très étendus et soumis à des contraintes d'exploitation fortes.

V.4.4.MODÈLES DE RÉPARTITION DES BASES DE DONNÉES:

V.4.4.1.RAPPELS : ARCHITECTURE DES SERVEURS DE BASES DE DONNÉES:

Du point de vue physique, une base de données n'est autre qu'un ensemble de fichiers. Ce qui caractérise une base de données est l'ORGANISATION LOGIQUE de ces fichiers, c'est à dire les RELATIONS qui existent entre ces différents fichiers. De ce fait, tout ce qui a été dit précédemment sur les différentes modalités de stockage des données s'applique aux bases de données (par exemple, les fichiers constituant une BDD peuvent être stockée sur un NAS suivant les techniques du RAID).

L'accès aux données se fait obligatoirement par l'intermédiaire d'un logiciel particulier appelé Système de Gestion de Bases de Données (SGBD), capable d'interpréter les relations existant entre les différents fichiers de la BDD. Les logiciel qui utilisent une base de données doivent donc utiliser un SGBD compatible avec le type de cette base de données (MySQL, Firebird, ORACLE, etc.).

Un S.G.B.D est donc une application SERVEUR DE BASES DE DONNÉES hébergée sur une machine connectée à un réseau (en général, TCP/IP). Une application CLIENTE communique avec un S.G.B.D grâce à des requêtes particulières utilisant ce réseau. L'accès à un S.G.B.D nécessite la saisie par l'utilisateur d'un couple "identificateur-mot de passe".

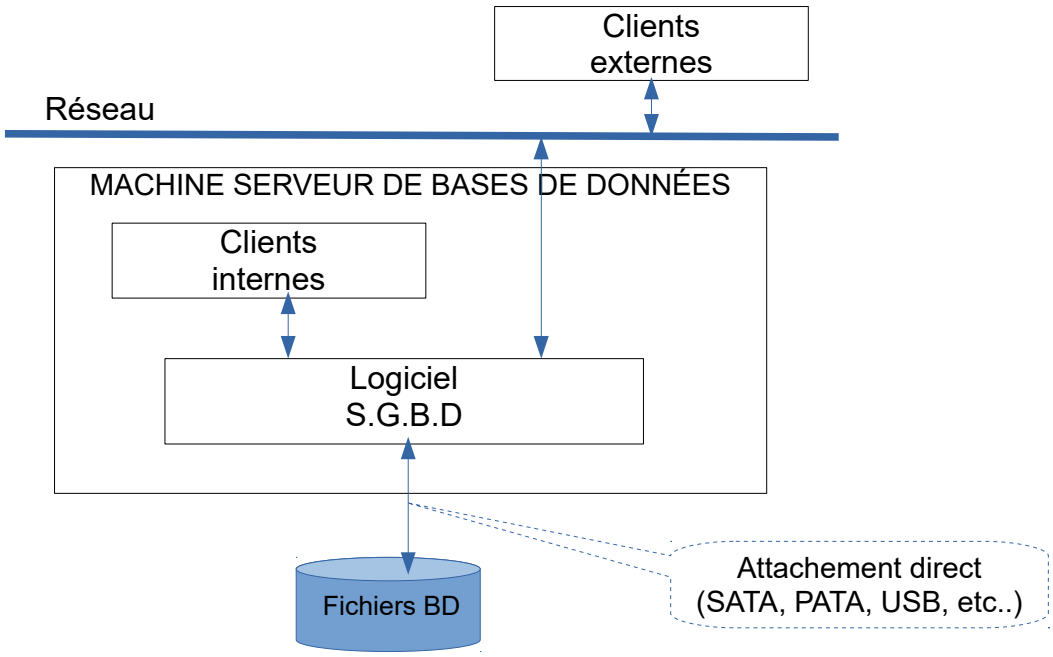
Il résulte de ce qui précède que le logiciel S.G.B.D et les applications utilisatrices (clientes) peuvent être localisés sur la même machine ou sur des machines différentes connectées en réseau.

De même, les unités de stockage supportant les fichiers constituant les bases de données peuvent être attachées directement aux machines supportant le S.G.B.D ou connectées en réseau à ces machines par l'intermédiaire d'équipements spécifiques (NAS, SAN).

Les principales solutions de répartition sont exposées ci-après:

V.4.4.2.**ARCHITECTURE COMPACTE:**

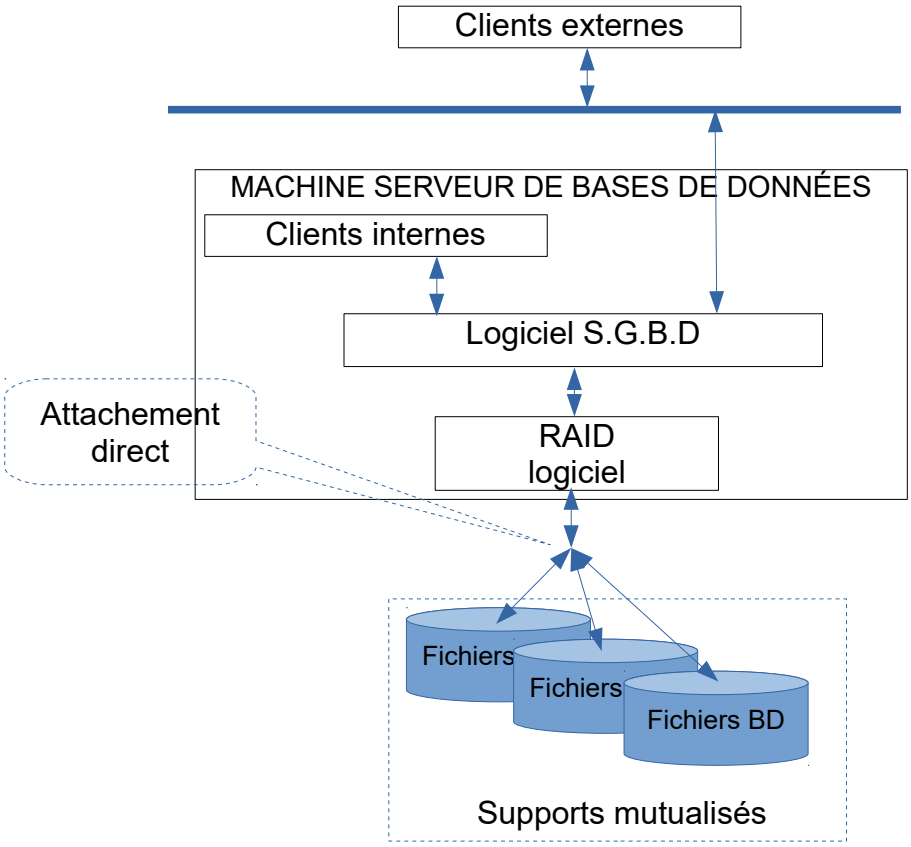
Dans ce type d'architecture, le ou les supports de stockage sont attachés directement à la machine hébergeant le logiciel S.G.B.D (serveur de bases de données). Le schéma ci-dessous représente l'architecture générale de cette solution:



COMMENTAIRES:

- Cette architecture a l'avantage de minimiser les délais de transmission entre le serveur et les supports de stockage (L'attachement direct par une liaison point-à-point spécifique est généralement beaucoup plus performant en termes de vitesse et de volume de transmission qu'un attachement par réseau (surtout si ce réseau est partagé avec d'autres activités). Il est également plus sécurisé contre les intrusions;
- Cette architecture est également peu coûteuse: un simple ordinateur de bureau muni d'un CPU de puissance moyenne, d'un disque suffisamment performant en termes de capacité et de temps d'accès et d'un logiciel SGBD libre peut constituer un serveur de bases de données acceptable pour beaucoup d'applications;
- Cependant, la sécurité de fonctionnement est minimale: aucune redondance matérielle ou logicielle des données n'existe. De ce fait, une panne disque fait "tomber" tout le système.

REMARQUE: la fiabilité peut être améliorée en remplaçant le disque unique par une grappe de plusieurs disques dont la somme des capacités est supérieure ou égale à celle du disque unique et en mutualisant cette grappe par un RAID:



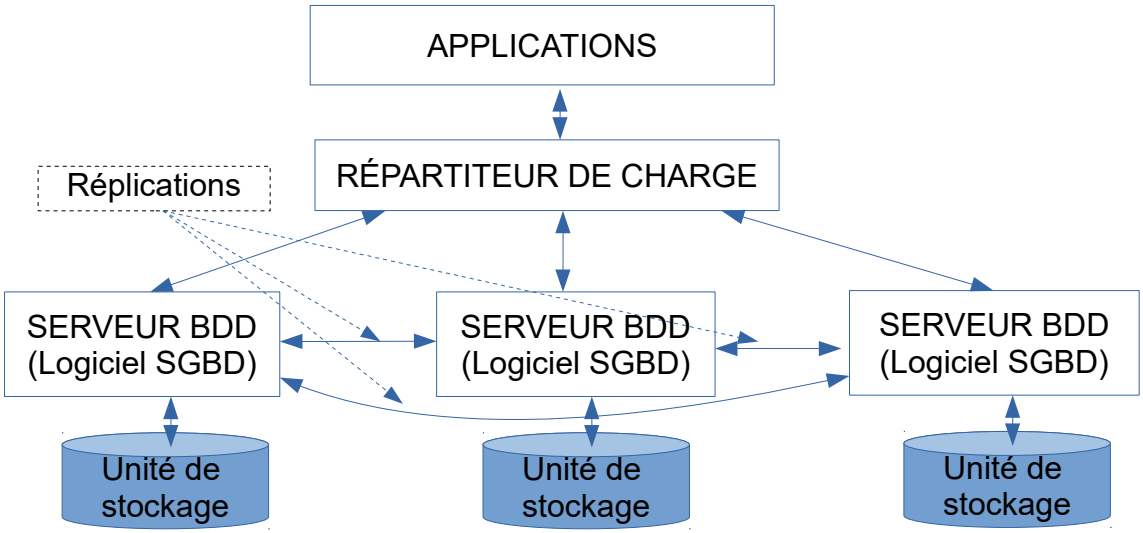
APPLICATIONS:

Ces configurations conviennent donc pour les activités logicielles contraintes en temps de réponse mais au fonctionnement peu ou moyennement critique et exploitant des bases de données de taille réduite (jusqu'à 3 ou 400 Mo).

V.4.4.3.ARCHITECTURE EN GRAPPE:

CARACTÉRISTIQUES PRINCIPALES:

Ce type d'architecture utilise une GRAPPE de serveurs de bases de données. Les nœuds de cette grappe (au moins 2) peuvent être attachés directement à une même machine ou connectés à un réseau commun. Le schéma suivant modélise l'architecture LOGIQUE de cette solution:



COMMENTAIRES:

- L'ensemble des espaces de stockage disponibles sur les nœuds est vu par les utilisateurs comme un espace unique mutualisé.
- L'architecture en grappe vise en premier à garantir la REDONDANCE DES DONNÉES, de façon à pouvoir pallier la défaillance d'une partie des nœud. Ceci implique:
  - Que chaque nœud possède la totalité des données de chacune des base de données hébergées par la grappe;
  - Que toute opération d'écriture concernant l'un des nœuds soit répliquée systématiquement pour les autres nœuds. Cette réplication peut être immédiate (réplication synchrone) ou différée (réplication asynchrone).
- L'architecture en grappe inclue des mécanismes visant à RÉPARTIR LES REQUÊTES des utilisateurs sur les différents nœuds de façon à optimiser leur utilisation (système de répartition de la charge ou "load balancing");

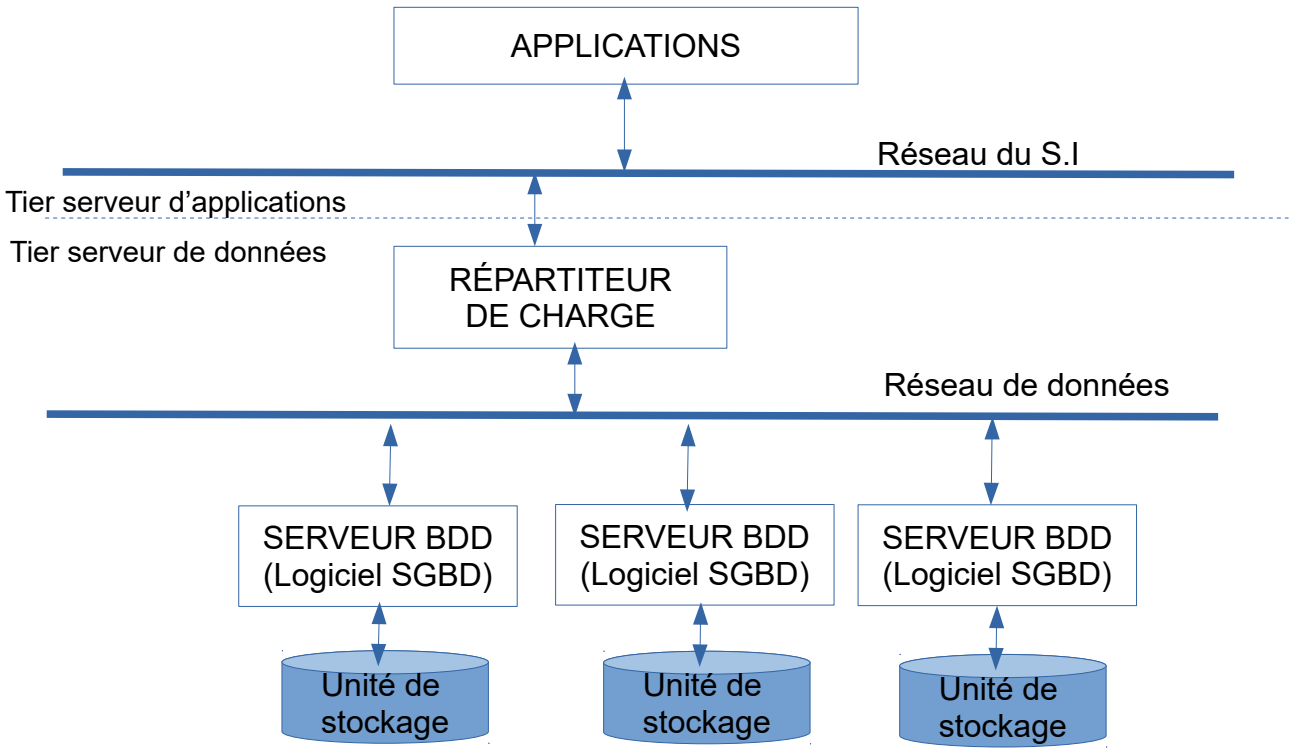
- La répartition des tables de chacune des bases de données sur les différents nœuds de la grappe peut être conçue de façon à permettre de PARALLÉLISER L'EXÉCUTION de certaines requêtes sur les bases de données. Ce but est atteint par le partitionnement des fichiers contenant les tables des bases de données. Ce partitionnement peut être réalisé de deux manières:
  - Partitionnement "horizontal": l'ensemble des LIGNES composant chaque tables est divisé en plusieurs partitions enregistrées sur des nœuds différents;
  - Partitionnement "vertical": l'ensemble des COLONNES composant chaque tables est divisé en plusieurs partitions enregistrées sur des nœuds différents.

EXEMPLE: MySQL CLUSTER:

Depuis sa version 4.1, le logiciel libre MySQL dispose du moteur de stockage NDB qui lui permet de gérer une grappe de serveurs. A partir de deux nœuds, il est possible de créer un CLUSTER. Le logiciel répartiteur de charge peut être installé sur une machine dédiée ou sur l'un des nœuds du cluster.

ARCHITECTURE PHYSIQUE:

Le schéma ci-dessous représente l'architecture physique de la solution:



REMARQUES:

- Éventuellement, la machine "répartiteur de charge" peut supporter un S.G.B.D et un support de stockage lui permettant de se constituer en nœud du cluster. Cette possibilité permet de constituer une grappe avec seulement deux machines;
- Afin de garantir la rapidité et le déterminisme temporel des échanges entre les nœuds de la grappe, les clients et le répartiteur de charge, il est recommandé dédier le réseau de données uniquement aux échanges entre les différents éléments de la grappe.

APPLICATIONS:

L'avantage de ce type de solution est l'augmentation de la sécurité de fonctionnement: le système continue à être opérationnel même si un seul nœud du cluster reste en état de marche. D'autre part, la remise en fonction d'un nœud après réparation est très facile: il suffit de le reconnecter au réseau, normalement, le contenu du supports de stockage sera réinstallé automatiquement.

La solution reste peu onéreuse car les machines supportant les nœuds n'ont pas besoin d'être très performantes.

Cette solution convient donc pour les activités logicielles peu contraintes en temps de réponse (tout dépend des performances et de l'encombrement du réseau de données), mais au fonctionnement assez critique.

V.4.4.4.ARCHITECTURES RÉPARTIES:

CARACTÉRISTIQUES PRINCIPALES:

Le concept de base de données RÉPARTIE (ou DISTRIBUÉE) est beaucoup plus complexe que celui de base de données en grappe: il s'agit de mutualiser des espaces de stockage gérés par des serveurs pouvant être distants et hétérogènes du point de vue de leurs technologies. Nous pouvons distinguer en premier lieu:

- Les BASES DE DONNÉES HOMOGÈNES: dans ce cas, tous les sites stockent les bases de données de manière identique en ce qui concerne le S.G.B.D employé et les structures de données utilisées. Ces caractéristiques rendent relativement simple la gestion de l'ensemble;
- Les BASES DE DONNÉES HÉTÉROGÈNES, qui mutualisent différents sites pouvant utiliser des systèmes d'exploitation, des S.G.B.D et des schémas de données différents;

L'architecture générale correspond plus au concept de GRILLE (GRID en anglais) qu'à celui de grappe (surtout en ce qui concerne les bases de données hétérogènes).

L'ensemble des espaces de stockage disponibles sur les nœuds est vu par les utilisateurs comme un espace unique mutualisé.

Comme dans l'architecture en grappes, les données constitutives des différentes tables sont partitionnées et distribuées sur les différents supports de stockage mutualisés de façon à créer des redondances. Ce partitionnement peut être réalisé de deux manières:

- Partitionnement "horizontal": l'ensemble des LIGNES composant chaque tables est divisé en plusieurs partitions enregistrées sur des nœuds différents;
- Partitionnement "vertical": l'ensemble des COLONNES composant chaque tables est divisé en plusieurs partitions enregistrées sur des nœuds différents.

APPLICATIONS: Les bases de données réparties sont des solutions complexes du point de vue de leur conception et de leur réalisation. De ce fait ce type de solution doit être justifié par des nécessités impérieuses telles que la gestion de très gros volumes de données, pouvant évoluer considérablement dans le temps (big data) ou la nécessité de fusionner de systèmes informatiques hétérogènes ou distants.

V.5.SÉCURISATION DU FONCTIONNEMENT:

V.5.1.PRINCIPE:

Pour éviter qu'une panne matérielle d'une des machines de la plate-forme d'accueil d'une application interrompe ou dégrade le fonctionnement de celle-ci, le moyen le plus évident consiste à disposer d'une autre machine munie des mêmes logiciels que la machine défaillante et prête à reprendre l'exécution de ceux-ci (cette autre machine est une REDONDANCE de la première). Ce basculement d'une machine à l'autre est appelé en anglais FAIL OVER. Le mode de fonctionnement avant basculement est dit NOMINAL. Après basculement, il est dit DÉGRADÉ car la sécurité de fonctionnement a diminué.

Dans le cas ou une redondance est prévue, la machine qui est choisie au lancement de l'application pour exécuter les traitements et fournir les résultats est nommée MAÎTRE. La deuxième machine, nommée ESCLAVE ou ADJOINTE peut:

- Soit se contenter de recopier périodiquement l'état des données persistantes de la machine MAÎTRE pour être capable de reprendre les traitements au point où ils en étaient au moment de la panne;
- Soit exécuter les mêmes traitements que la machine maître, à partir des mêmes données, sans fournir ses résultats au reste de l'application. Cette solution permet de comparer en permanence les résultats des deux machines redondantes et, de ce fait, de détecter des dysfonctionnements beaucoup plus élaborés que la simple absence de fourniture de résultats ou de réponse à un message. En particulier, il permet de détecter les dysfonctionnements de la machine ADJOINTE.

V.5.2.REDONDANCE A FROID, REDONDANCE A CHAUD ET TEMPS DE LATENCE:

Un système de redondance est dit "A FROID" quand la reprise des traitements supportés par la machine MAÎTRE par la machine ADJOINTE est déclenchée MANUELLEMENT (par action sur une I.H.M, par exemple);

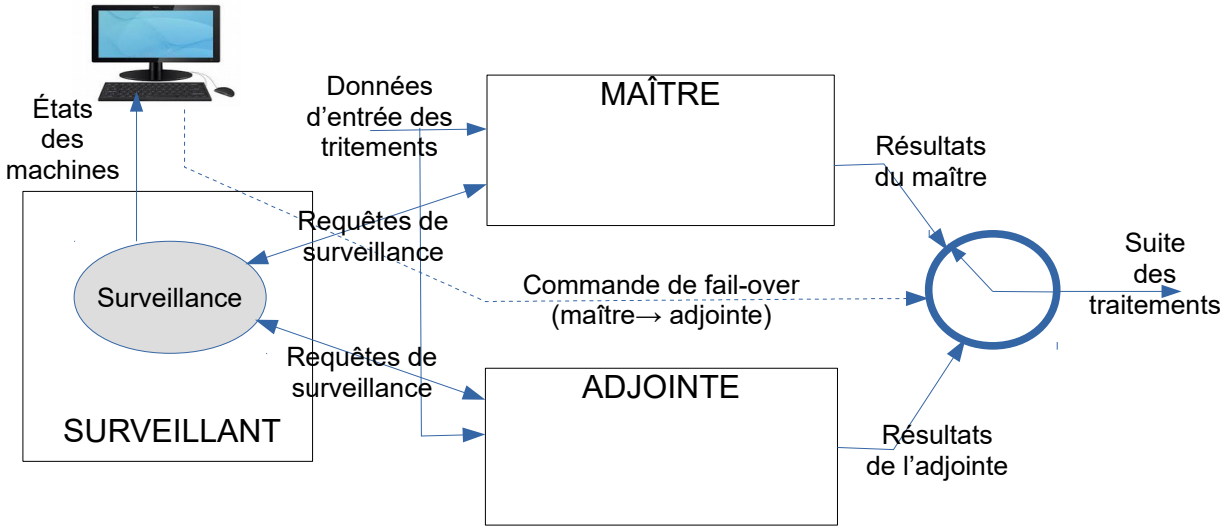
Un système de redondance est dite "A CHAUD" quand cette reprise est déclenchée AUTOMATIQUEMENT par un système automatique de surveillance du fonctionnement.

La durée entre la détection de la panne et la reprise du fonctionnement de l'application après basculement est appelée TEMPS DE LATENCE. Celui-ci peut être de quelques dizaines de millisecondes en cas de redondance à chaud et de plusieurs secondes en cas de redondance à froid.

V.5.3.MÉCANISME DE LA REDONDANCE:

REMARQUE: les schémas ci-après sont des schémas de principe. Ils ne préjugent pas de l'architecture matérielle du système.

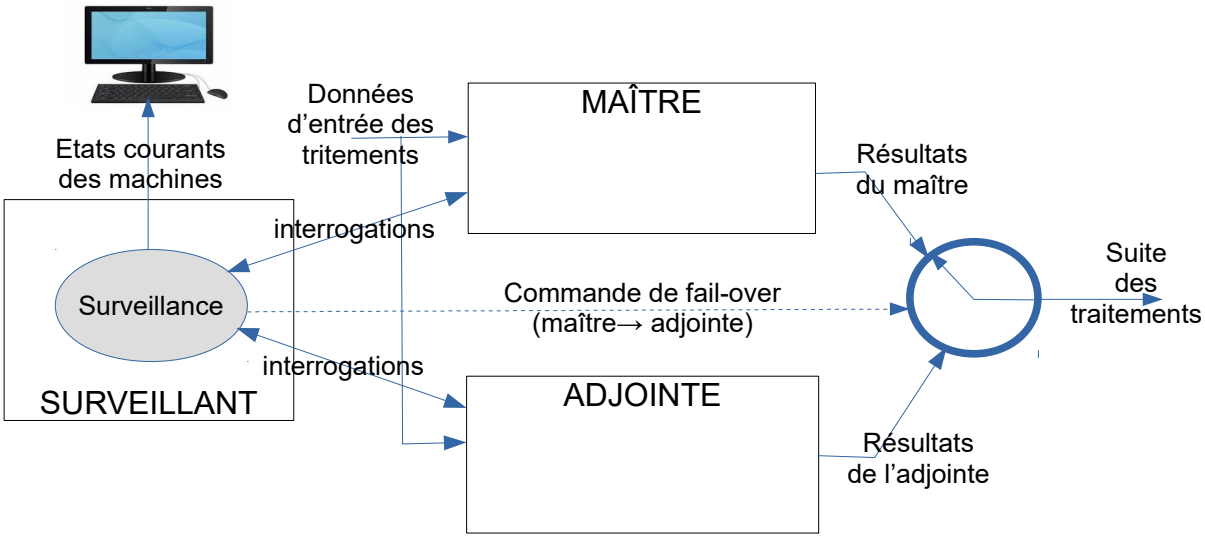
V.5.3.1.REDONDANCE A FROID:



COMMENTAIRES:

- Le SURVEILLANT teste en permanence les deux machines redondantes: il leur envoie des requêtes auxquelles ces machines doivent répondre en un temps inférieur à une certaine limite (time-out). Si la réponse est erronée ou si aucune réponse n'est reçue avant le time-out, la machine surveillée est déclarée "hors service";
- L'état courant des machines est affiché sur une I.H.M. Le passage d'une machine à l'état "hors service" peut être également signalé à l'exploitant par un signal sonore;
- Si le MAÎTRE passe à l'état hors service, c'est l'exploitant qui déclenche le "fail over" à partir de l'I.H.M. Il peut ensuite déclencher une action de maintenance ou de remplacement pour cette machine;
- Si l'ADJOINTE passe à l'état hors service, l'exploitant peut déclencher une action de maintenance ou de remplacement pour cette machine.

V.5.3.2.REDONDANCE A CHAUD:



COMMENTAIRES:

- Le SURVEILLANT teste en permanence les deux machines redondantes: comme précédemment, il leur envoie des requêtes auxquelles ces machines doivent répondre en un temps inférieur à une certaine limite (time-out). Si la réponse est erronée ou si aucune réponse n'est reçue avant le time-out, la machine surveillée est déclarée "hors service" et le basculement maître → adjointe est déclenché AUTOMATIQUEMENT;
- L'état courant des machines est affiché sur un I.H.M. Le passage d'une machine à l'état "hors service" peut être également signalé à l'exploitant par un signal sonore.

V.5.3.3.REDONDANCE A CHAUD AVEC COMPARAISON DES RÉSULTATS:

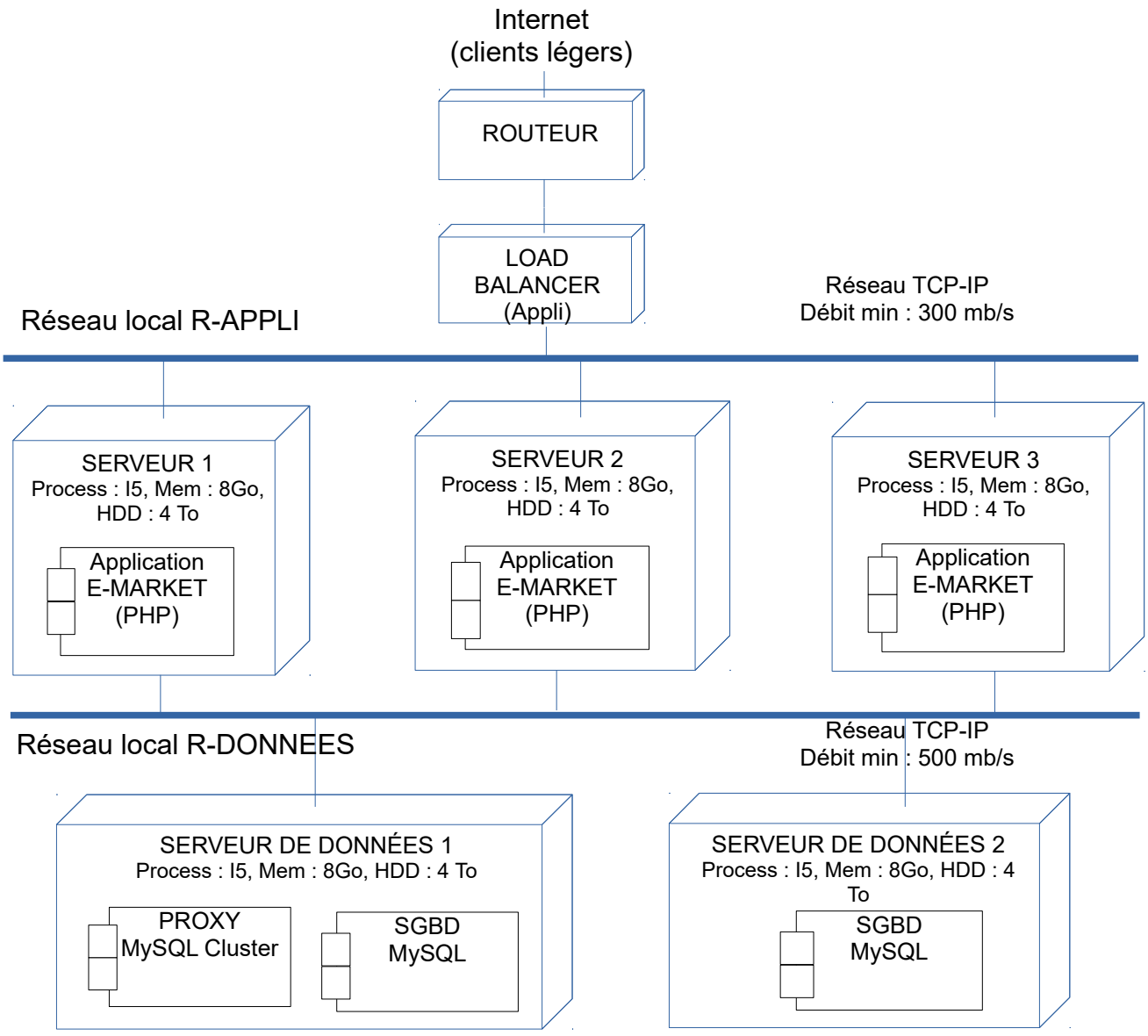
Si la machine adjointe exécute effectivement les mêmes traitements que la machine maître, les résultats de ces traitements peuvent être renvoyés au SURVEILLANT afin d'être comparés entre eux. Ceci permet d'affiner la détection des dysfonctionnements des machines. Dans les cas où la fiabilité du système est primordiale, on peut envisager d'utiliser deux machines adjointes, ce qui permet de disposer de trois jeux de résultats et de déclarer hors service la machine dont les résultats divergent par rapport aux deux autres (règle de la majorité).

V.6.PRÉSENTATION DE LA SOLUTION D'ARCHITECTURE SYSTÈME :

Les enseignements obtenus lors des étapes précédentes et les choix architecturaux effectués permettent d'élaborer une solution architecturale au niveau du système informatique.

Cette solution peut être représentée de diverses façons. L'option que nous avons choisi ici est d'utiliser la symbolique des diagrammes de déploiement U.M.L, accompagnée de commentaires littéraux explicitant et justifiant les choix effectués.

EXEMPLE : le diagramme de déploiement suivant peut correspondre à la conception d'un site web marchand (Application E-MARKET) :



Ce diagramme pourra être accompagné de commentaires littéraux explicitant et justifiant la solution choisie. Ce qui suit est un exemple de rédaction :

ARCHITECTURE DU TIERS CLIENT :

SOLUTION PROPOSÉE : Solution de type "client léger" :

- les traitements d'affichage des vues seront assurés par le navigateur des visiteur.
- Les traitements de préparation des vue seront supportés par les mêmes machines que les couches "métier".

JUSTIFICATION :

- La destination de l'application (Site WEB marchand ouvert à tout public) impose de choisir une solution de répartition de type "client léger" ;
- La charge représentée par ces traitements de préparation des vue et les contraintes concernant les temps de réponse ne justifient par de les distribuer sur des machines dédiées à ce seul usage.

ARCHITECTURE DU TIER SERVEUR DE DONNÉES :

SOLUTION PROPOSÉE :

Système de gestion MySQL Cluster, composée de deux machines (dont une supportant le répartiteur de charge), raccordées à un réseau local dédié (R-DONNÉES).

JUSTIFICATION :

- L'utilisation d'un site web marchand par un visiteur sans privilège particulier induit beaucoup plus de lectures en bases de données que d'écriture. En revanche, les traitements d'administration du site ou de gestion du contenu peuvent induire de nombreuses écritures en base de données, mais il s'agit de traitements déclenchés peu souvent et par un petit nombre d'utilisateurs ;
- De ce fait, compte tenu de la cible en termes de nombre de visiteurs, une solution à base de MySQL Cluster semble bien adaptée : elle a l'avantage d'augmenter la fiabilité du système de stockage tout en améliorant les temps d'accès aux données en lecture et en facilitant l'évolution des capacités de stockage.

ARCHITECTURE DU TIER SERVEUR D'APPLICATIONS :

SOLUTION PROPOSÉE :

Distribution de l'application sur un cluster de trois serveurs d'application redondants connectés à un réseau local TCP-IP de débit minimal > 300 mb (R-APPLI). La répartition des charges sera assurée par un "load balancer" utilisant un algorithme de distribution de charge "dynamique". Ce logiciel sera supporté par une machine dédiée connectée au routeur internet et au réseau R-APPLI.

L’application E-MARKET, comprenant les couches "préparation de l’affichage", "métier" et "données", sera hébergée par les machines du tier "serveur d’application".

JUSTIFICATION :
L’application étant du type "client-serveurs", cette architecture permet d’optimiser l’utilisation de la puissance de calcul du cluster de serveurs tout en garantissant une excellente fiabilité de fonctionnement, du fait de la redondance des deux serveurs d’application.

VI.ÉTAPE IV - LA CONCEPTION, DU POINT DE VUE LOGIQUE:

VI.1.INTRODUCTION:

Nous avons vu précédemment que le point de vue LOGIQUE concerne l'agencement des différents composants de l'architecture (identification et caractérisation des composants principaux et des relations existantes entre eux).

VI.2.PASSAGE DU FONCTIONNEL A L'ORGANIQUE:

VI.2.1.PRINCIPE :

Nous avons vu lors de l'étude de la conception au niveau du système informatique que l'analyse d'une spécification des besoins bien élaborée et suffisamment précise permet à un développeur qualifié d'identifier les ACTIVITÉS nécessaires à la satisfaction des exigences et contraintes exprimées par la S.T.B de l'application ainsi que les traitements correspondant à ces activités.

D'une manière analogue, le passage de la spécification du besoin à la conception logique du logiciel amène le concepteur à identifier des entités logicielle dites STATIQUES, comme des structures de données, des fonctions ou des classes, et des relations "statiques" entre ces entités.

REMARQUE:

Ces entités et relations peuvent être qualifiées de STATIQUES dans la mesure où elles existent même en dehors des sessions d'exécution de l'application à laquelle ils appartiennent, contrairement à des entités telles que les processus, les threads ainsi que les flux de données ou de contrôle qui n'existent que pendant l'exécution de l'application (ces entités sont parfois dites DYNAMIQUES).

De ce fait, on s'intéressera ici uniquement aux DONNÉES PERSISTANTES, celles qui SURVIVENT aux traitements qui les utilisent jusqu'à ce qu'un de ces traitements les supprime (fichiers, bases de données, zones de mémoire partagées entre processus, etc.). Les données transitoires qui sont créées et utilisées uniquement durant l'exécution d'un traitement donné ne concernent pas la conception LOGIQUE.

VI.2.2.DÉMARCHES PRINCIPALES:

Le passage du point de vue FONCTIONNEL d'un logiciel (expression des exigences et contraintes) au point de vue LOGIQUE est l'un des points les plus délicat et controversés du génie logiciel. La plupart des méthodes de développent s'appuient sur l'un ou l'autre des paradigmes suivants :

- Soit déduire la solution architecturale de la hiérarchie des traitements établie à partir de la spécification des besoins: c'est la DÉMARCHE PROCÉDURALE;
- Soit la déduire de la structure des données: c'est la DÉMARCHE GUIDÉE PAR LES DONNÉES, dont la CONCEPTION PAR OBJETS est l'illustration principale.

Dans la suite de ce chapitre, nous allons étudier en détail ces démarches.

VI.3.LA DÉMARCHE PROCÉDURALE:

VI.3.1.GÉNÉRALITÉS:

La démarche de conception procédurale consiste à représenter le logiciel sous la forme d'un ensemble d'ENTITÉS encapsulant les TRAITEMENTS qui permettent de satisfaire les EXIGENCES issues de la spécification des besoins. C'est donc une démarche GUIDÉE PAR LES TRAITEMENT.

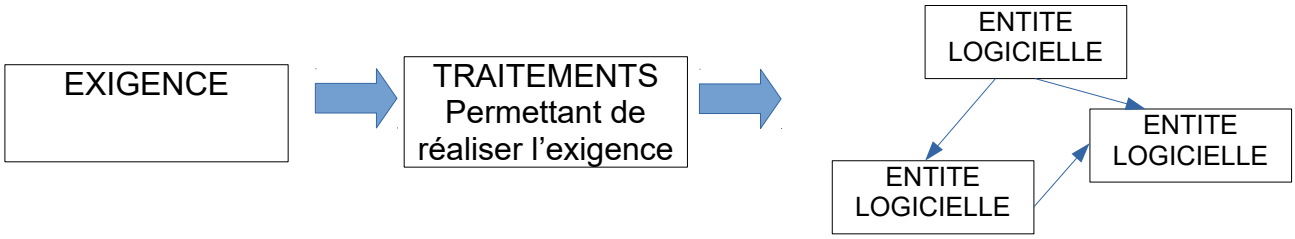


Fig. IV.2.2.1-A : démarche procédurale

REMARQUE: Dans la très grande majorité des cas, ces entités sont appelées FONCTIONS (bien que cette appellation présente un risque de confusion avec l'analyse fonctionnelle). En fait, cette appellation se justifie par la parenté logique de ces entités avec les PROCÉDURES ou FONCTIONS des langages de programmation (c, c++, java, etc..): on peut leur associer une INTERFACE décrivant la manière de les utiliser et un CORPS qui implémente l'algorithme de traitement qu'elles encapsulent.

VI.3.2.LE PASSAGE DU FONCTIONNEL A L'ORGANIQUE:

La démarche procédurale cherche à déduire de chacune des EXIGENCES FONCTIONNELLES définies par la S.T.B les TRAITEMENTS qui permettront de satisfaire ces exigences.

EXEMPLE: si une exigence concernant un site web marchand peut s'exprimer ainsi:

"Le visiteur doit pouvoir créer un compte client".

Il est évidemment trivial d'en déduire que le logiciel qui supportera cette exigence devra comprendre les traitements qui permettent d'effectuer cette opération. Nous déduirons donc de cette exigence une ou des FONCTIONS (au sens procédural) encapsulant les traitements nécessaires à la création d'un compte client.

VI.3.3.LA TECHNIQUE DU RAFFINAGE:

Dans le domaine de la conception des logiciel, RAFFINER un élément de conception consiste à détailler sa structure interne en plusieurs sous-éléments reliés entre eux par des relations de dépendance ou de collaboration. Cette démarche peut être ITÉRATIVE,

chacun des sous-composants pouvant lui-même être raffiné. A chaque itération correspond donc à une GRANULARITÉ de plus en plus fine.

VI.3.4.EXEMPLE DE RAFFINAGE:

VI.3.4.1.DESCRIPTION DU CAS:

Prenons l'exemple d'un site web marchand. La S.T.B de cette application énonce les exigences suivantes:

- Un visiteur du site doit pouvoir consulter le CATALOGUE du site;
- Un VISITEUR du site doit pouvoir ouvrir un compte CLIENT;
- Un CLIENT doit pouvoir acheter un PRODUIT figurant au CATALOGUE du site;

REMARQUE: Afin de ne pas trop compliquer cet exemple, nous considérerons que le client ne peut acheter qu'un type de produit à la fois. D'autre part, il a été fait abstraction des mécanismes de facturation et de paiement.

VI.3.4.2.PASSAGE DE L'EXIGENCE A LA FONCTION:

Intéressons nous à l'exigence n°2: "Les visiteurs doivent pouvoir créer un compte client". En conception procédurale, cette exigence conduit logiquement à imaginer une fonction qui réalise cette exigence. Nous l'appellerons "CreerCompteClient".

VI.3.4.2.1.PREMIER NIVEAU DE RAFFINAGE:

Le raffinage consiste à décomposer cette fonction en éléments plus simples. Pour cela, il faut avoir une idée assez précise de l'algorithme qu'elle doit encapsuler. Nous pouvons analyser le traitement en utilisant un PSEUDO-CODE:

```
fonction CreerCompteClient ()
DÉBUT
    {Afficher le menu de saisie des paramètres d'identification d'un visiteur};
    TANT QUE {des paramètres d'identification acceptables n'ont pas été saisis} FAIRE
        { Tenter d'acquérir des paramètres d'identification }
        SI {ces paramètres sont acceptables} ALORS
            {Créer un nouveau client dans la liste des clients};
            {Signaler la création du compte};
        SINON
            {Ré-afficher le menu de saisie des paramètres d'identification de client};
            {Signaler l'échec de la création du compte};
        FINSI
    FINFAIRE
FIN
```

REMARQUES:

- Ce pseudo-code ne constitue qu'une solution parmi d'autres;
- A ce niveau, aucun des traitements partiels figurant dans le pseudo-code ne doit être détaillé. Chacun d'eux est représenté par sa spécification "fonctionnelle" (que l'on représente, en général, entourée d'accolades). On fait donc abstraction des détails d'implémentation de ces traitements. En revanche, la logique d'enchaînement de ces traitements est bien définie;
- Les spécifications figurant dans les accolades doivent concorder avec les exigences exprimées dans la S.T.B, et en particulier avec les CRITÈRES attachés à ces exigences. En effet, une spécification telle que la condition "SI {ces paramètres sont acceptables} ALORS" suppose que la S.T.B ait bien défini ces critères d'acceptation (longueur minimale des mots de passe, composition, unicité des couples Id, mot de passe, etc.). Dans le cas contraire, il faudra peut-être compléter la S.T.B. Ceci ne pose en général aucun problème dans un cycle "agile", mais peut être plus pénalisant dans un cycle "classique";
- ***NOTA:*** *le pseudo-code ci-dessus permet d'identifier une donnée persistante importante de l'application: c'est la "liste des clients". Cette donnée sera probablement implémentée par une TABLE de la base de données de l'application.*

Chacun des traitements identifiés dans le pseudo-code est ensuite RAFFINÉ. Cette technique consiste:

- Soit à remplacer dans ce premier algorithme les traitements par la description algorithmique fine de ceux-ci, quand cela est possible sans trop complexifier l'algorithme;
- Soit, dans le cas contraire, à encapsuler ces traitements dans une fonction dont on va se contenter de définir les spécifications et l'interface.

EXEMPLE:

Dans l'algorithme précédent, les traitements qui feront l'objet d'une encapsulation seront:

TRAITEMENT (SPÉCIFICATIONS)	FONCTIONS D'ENCAPSULATION
Afficher le menu de saisie des paramètres d'identification d'un visiteur	AfficherMenuSaisieIdentificateurs ():
des paramètres d'identification acceptables n'ont pas été saisis	Boolean VerifierIdentificateurs (Id, MDP);
Créer un nouveau client dans la liste des clients	Boolean CreerNouveauClient (Id, MDP);

l'algorithme précédent devient alors

```
fonction CreerCompteClient ()
DÉBUT
    boolean:   ParametresCorrects = faux;
    boolean:   ClientCree = faux;
    chaîne:    Id = "";
    chaîne:    MDP = "";

    AfficherMenuSaisieIdentificateurs ();
    TANT QUE ParametresAcceptables  = faux FAIRE
        LIRE  id, MDP;
        ParametresAcceptables = VerifierIdentificateurs ( Id, MDP );
        SI ParametresAcceptables = vrai ALORS
            ClientCree = CreerNouveauClient ( Id, MDP );
            SI (ClientCree = vrai ) ALORS AFFICHER "Le compte a été créé";
            SINON AFFICHER "Erreur de création du compte";
            FINSI
        SINON
            AFFICHER "Paramètres d'identification incorrects";
            AfficherMenuSaisieIdentificateurs ();
        FINSI
    FINFAIRE
FIN
```

La démarche de raffinage conduit à une structuration du logiciel que l'on peut représenter par le GRAPHE STRUCTUREL suivant:

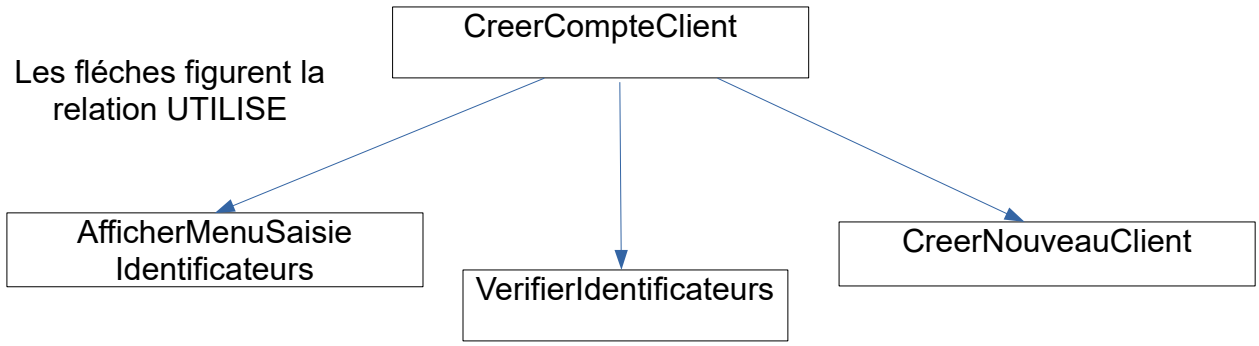


Fig. IV.3.2.3-A : Raffinage d'une fonction CreerCompteClient (niveau 1)

REMARQUES:

- Ce schéma montre que la démarche de raffinage conduit à définir des arborescences de traitements indépendantes les une des autres à partir de chaque

exigence fonctionnelle: une autre arborescence pourrait être développée à partir de l'exigence "Un client peut acheter un produit";

- Il ne s'agit pas ici d'une décomposition fonctionnelle: les 3 entités de niveau inférieur ne sont pas équivalentes à l'entité de niveau supérieur. On a simplement "vidé" l'entité supérieure des détails d'implémentation trop complexes pour être étudiés à ce niveau pour ne garder que la logique des traitements;.

*VI.3.4.2.2.**DEUXIÈME NIVEAU DE RAFFINAGE:***

Itérer le raffinage revient à appliquer aux fonctions de niveau inférieur la même démarche:

- Exprimer le contenu algorithmique sous la forme de pseudo-code;
- Identifier les blocs de traitement mis en évidence à des fonctions de niveau inférieur (définir les interfaces de ces fonctions et les traitements encapsulés.

RAFFINAGE DE LA FONCTION AfficherMenuSaisieIdentificateurs:

AfficherMenuSaisieIdentificateurs ()

DEBUT

{Afficher l'en-tête du menu: "Saisie des identificateurs"};
{Afficher le label: "Pseudo:"}
{Afficher le champ de saisie du pseudo du client}
{Afficher le label: "Mot de passe:"}
{Afficher le champ de saisie du mot de passe du client}
{Aller à la ligne}
{Afficher le bouton de saisie "valider}

FIN

Pour faire abstraction des détails d'implémentation, nous allons renvoyer certains traitements dans 4 sous-fonctions:

- AfficherChaine (chaîne C);
- AfficherChampSaisie (chaîne Type, chaîne Nom);
- AfficherBouton (chaîne Label, chaîne Nom);
- ALaLigne ();

DEBUT

AfficherChaine ("Saisie des identificateurs"};
ALaLigne ();
AfficherChaine ("Pseudo: "};
AfficherChampSaisie ("chaine", "Pseudo");
AfficherChaine ("Mot de passe: "};
AfficherChampSaisie ("chaine", "MDP");
ALaLigne ();

```
    AfficherBouton ( "Valider", "Validation" );
FIN
```

RAFFINAGE DE LA FONCTION VerifierIdentificateurs:

```
boolean VerifierIdentificateurs (Id, MDP)
DEBUT
    boolean: resultat = vrai;
    SI {le mot de passe a moins de 8 caractère} ALORS resultat = faux;
    SINON SI {le mot de passe contient moins de deux chiffres} ALORS resultat = faux;
    SINON SI {le mot de passe comprend moins d'une majuscule} ALORS resultat = faux;
    FINSI
    RETOURNER résultat;
FIN
```

Pour faire abstraction des détails d'implémentation, nous allons renvoyer certains traitements dans trois sous-fonctions:

- entier LgChaine (Chaine C); // calcul longueur d'une chaîne de caractères
- entier NbChiffres (Chaine C); // nombre de chiffres dans une chaîne
- entier NbMajuscules (Chaine C); // nombre de majuscules dans une chaîne

Le pseudo-code devient:

```
boolean VerifierIdentificateurs (Id, MDP)
DEBUT
    boolean: resultat = vrai;
    SI          LgChaine ( MDP ) { 9      ALORS      resultat = faux
    SINON SI    NbChiffres (MDP) { 2      ALORS      resultat = faux;
    SINON Si    NbMajuscules { 1          ALORS      resultat = faux;
    FINSI
    RETOURNER résultat;
FIN
```

RAFFINAGE DE LA FONCTION CreerNouveauClient

booleen CreerNouveauClient (chaine Id, chaine MDP)
DEBUT
booleen R;

R = AjouterClient (Clients, NouveauClient, Id, MDP);
RETOURNER R;
FIN

GRAPHE DE STRUCTURE CreerCompteClient

A ce niveau, le raffinage pourrait être représenté par le graphe suivant:

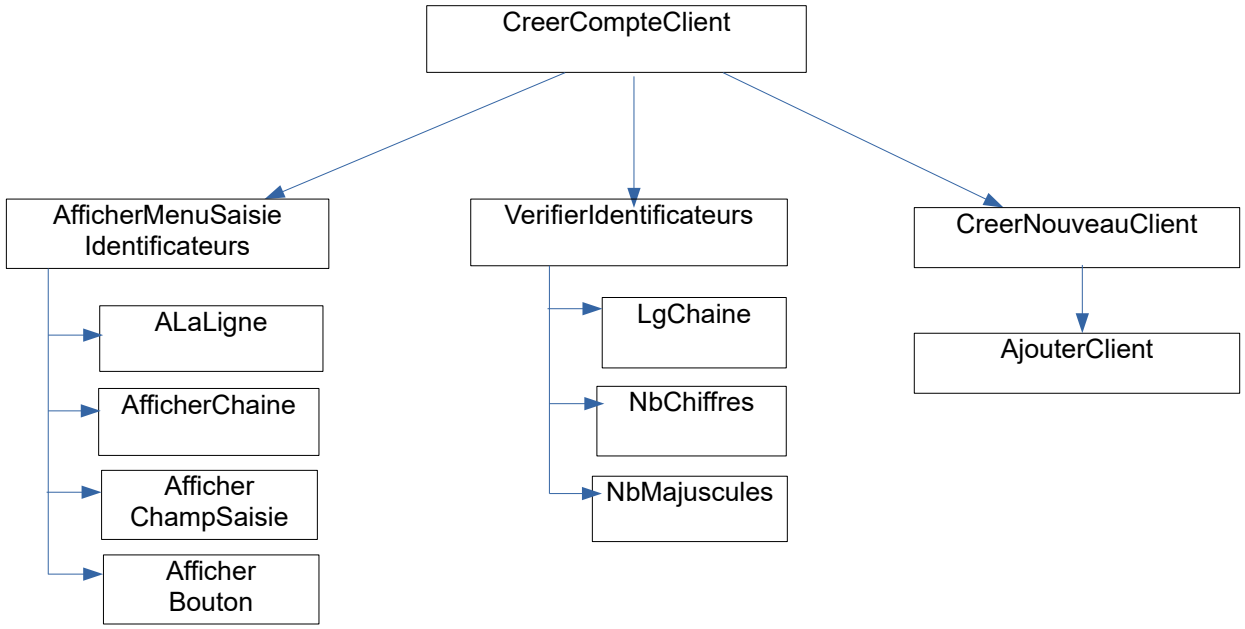


Fig. IV.3.2.3-A : Raffinage d'une fonction CreerCompteClient (Niveau 2)

*VI.3.4.2.3.**ARRÊT DU RAFFINAGE:***

L'exemple ci-dessus montre un raffinage itéré jusqu'au moment où le niveau de détail obtenu est équivalent à celui d'un programme informatique écrit dans le langage que l'on a choisi pour le codage. En effet, les fonctions identifiées au deuxième niveau peuvent être codées immédiatement.

Dans cette hypothèse, les phases de conception préliminaires et détaillées seraient confondues et l'on passerait directement en phase de codage. C'est ce qui se passe pour les méthodes agiles. Cependant, cette solution ne peut convenir que pour des applications de petit ou moyen volume, menées par des équipes de taille réduite.

Dans des projets de plus gros volume ou plus complexes, le raffinage des fonctions exige des compétences de plus en plus "pointues" au fur et à mesure des itérations. En effet, plus on avance dans le raffinage, plus les traitements encapsulés par les fonctions touchent des compétences particulières que les développeurs chargés de la conception générale ne maîtrisent pas forcément et n'ont en général pas le temps d'acquérir dans le cadres de la phase de conception générale.

Ce sera alors le moment pour eux de "passer la main" à des équipes de conception détaillée qui pourront approfondir les problématiques pendant qu'eux-mêmes continueront à élaborer l'architecture et le comportement général de l'application.

EXEMPLE: Dans l'exemple proposé, on peut trouver inutile, dans le cadres de la conception générale, de raffiner les fonctions AfficherMenuSaisieIdentificateurs, VerifierIdentificateurs et CreerNouveauClient: l'analyse de ces fonctions pourra être terminée dans le cadre de la CONCEPTION DÉTAILLÉE.

VI.3.1.RÉSULTAT DE LA DÉMARCHE PROCÉDURALE:

Dans les paragraphes précédents, nous nous sommes contentés d'analyser une seule exigence: "Le visiteur doit pouvoir ouvrir un compte client".

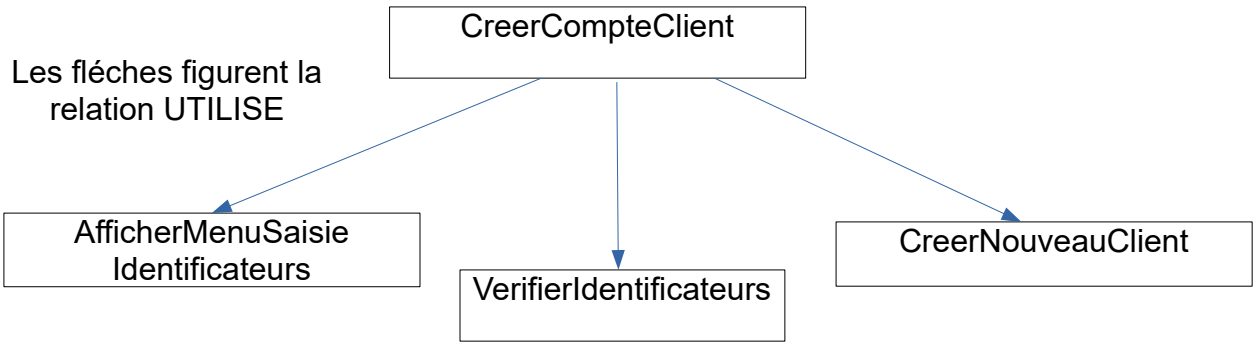


Fig. IV.3.2.3-A : Raffinage d'une fonction CreerCompteClient (niveau 1)

Il va maintenant falloir analyser les deux autres exigences:

- "Un Client doit pouvoir acheter un produit du catalogue"
- "Un visiteur doit pouvoir consulter le catalogue des produits proposés à la vente";

L'analyse procédurale de chacune de ces exigence va aboutir à un graphe structurel. Dans le cas de la première exigence, celle-ci se présentera comme suit:

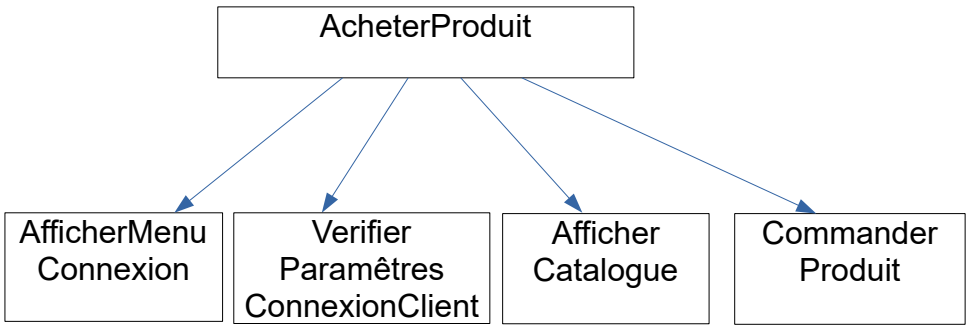


Fig. IV.3.2.6-A : Raffinage d'une fonction AcheterProduit (premier niveau)

Dans le cas de la deuxième exigence, elle se résumera à:

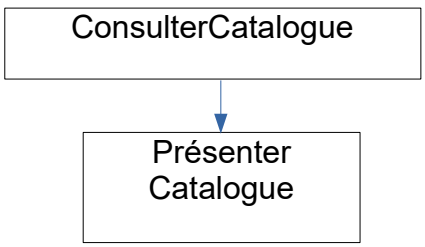


Fig. IV.3.2.6-B : Raffinage d'une fonction ConsulterCatalogue (premier niveau)

Le résultat des travaux de conception préliminaire devront donc faire apparaître au minimum le modèle architectural de l'application, par exemple sous la forme du diagramme suivant:

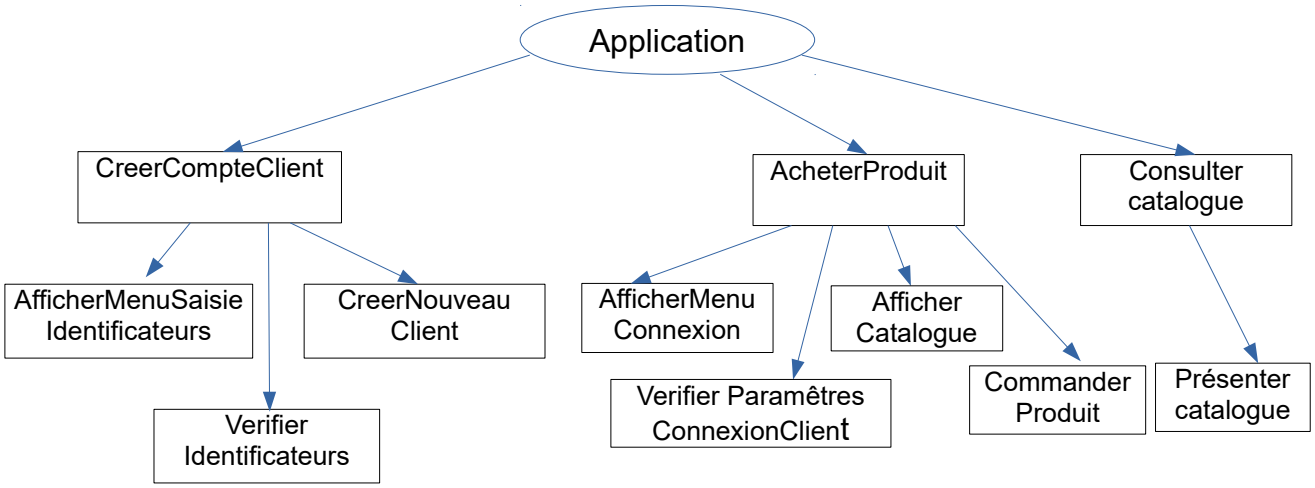


Fig. IV.3.3.1-A-D : Diagramme de structure "fusionnés"

REMARQUE: Pour constituer un DOSSIER DE CONCEPTION GÉNÉRALE, ce diagramme sera accompagné au minimum de la description des interfaces de chacune des fonctions figurant dans ce diagramme ainsi que des pseudo-codes des fonctions de premier niveau et des spécifications des fonctions de niveau inférieur.

VI.3.2.**CRITIQUE D'UNE DÉMARCHE PUREMENT PROCÉDURALE:**

NOTA: Nous dirons qu'une démarche est "purement" procédurale si cette démarche est guidée par les traitements et menée uniquement par raffinements successifs: il s'agit donc d'une démarche entièrement descendante (Top-down).

VI.3.2.1.**LES AVANTAGES:**

- La démarche procédurale étant très intuitive, elle demande relativement peu d'efforts de formation;
- Elle peut être appliquée à tous les niveaux de la conception (conception globale et conception détaillée).
- Elle conduit à développer des "lots fonctionnels". Ceci veut dire que l'analyse de chaque exigence aboutira à un logiciel qui pourra supporter les fonctionnalités correspondant à cette exigence sans l'aide du reste de l'application et pourra donc être livrée sans attendre au client. Cette particularité constitue un atout dans le cadre des méthodes agiles (principe de livraison "au plus tôt").

VI.3.2.2.**LES INCONVÉNIENTS:**

- La technique du raffinage construit l'architecture de l'application en s'inspirant uniquement de la hiérarchie des traitements. Les composants logiciels sont conçus sans tenir compte des données qu'ils manipulent. De ce fait, les données persistantes constituent des entités logicielles indépendantes des traitements qui les exploitent. Or, dans une application, les données persistantes (fichiers, bases de données) sont plus stables dans le temps que les algorithmes. De ce fait, un logiciel conçu à partir des traitements qu'il supporte a tendance à être plus difficile à maintenir qu'un logiciel conçu à partir des données qu'il manipule;
- La conception sous forme d'arborescence de fonctions favorise la duplication des traitements. En effet, comme nous l'avons vu plus haut, chaque exigence fonctionnelle va engendrer une arborescence de traitements indépendante des autres arborescences. De ce fait, les OPPORTUNITÉS DE RÉUTILISATION d'une fonction à un autre endroit de l'application ou dans une autre application sont difficiles à repérer, et ne le sont que très tard dans le processus de conception.

EXEMPLES: Dans les exemple présenté ci-dessus,

- Les développeurs chargés de l'arborescence AcheterProduit vont avoir besoin d'accéder à la liste des clients. Il en sera de même pour ceux qui sont chargés de l'arborescence CréerCompteClient. Cette donnée n'étant pas encore définie, cette situation est potentiellement une source d'erreur;

- Les fonctions AfficherCatalogue et PrésenterCatalogue sont probablement identiques du point de vue fonctionnel. Cependant, comme leur développement sera peut-être confié à des équipes différentes, cette opportunité de réutilisation ne sera pas détectée ou détectée trop tard pour être exploitée.

VI.3.2.3.**CONCLUSION:**

Une démarche procédurale menée uniquement par raffinages successifs ne convient que pour la conception d'applications de petit volume, dans des délais très contraints et n'exigeant pas des normes de qualité élevées (surtout en ce qui concerne la lisibilité, la maintenabilité et la ré-utilisabilité des composants). C'est le cas par exemple pour les applications de maquettage, conçues uniquement pour être utilisées dans le cadre d'une étude, d'une thèse, etc.

VI.3.3.AMÉLIORATIONS DE LA DÉMARCHE PROCÉDURALE "PURE":

VI.3.3.1.IDENTIFIER LES DONNÉES ET LES OPPORTUNITÉS DE RÉUTILISATION:

VI.3.3.1.1.IDENTIFICATION DES DONNÉES:

Nous avons vu précédemment qu'un des défauts de la démarche par raffinages successifs est de ne pas tenir compte des données dans le processus d'analyse, même si la représentation des traitements algorithmiques sous forme de pseudo-code permet de les mettre en évidence.

Or, l'identification et la description des données persistantes partagées dès la phase de conception générale est indispensable pour permettre la conception détaillée de chaque "lot".

Une solution consiste à faire apparaître ces données et leurs relations avec les fonctions dans les diagrammes de structure:

EXEMPLE:

Le raffinage des différentes exigences effectué dans les paragraphes précédents nous a permis d'identifier un certain nombre de structures de données persistantes de l'application.

Ainsi, le raffinage de la fonction CreerCompteClient a fait apparaître une sous-fonction CreerNouveauClient dont le but est d'introduire une nouvelle occurrence dans une donnée de type liste rassemblant les données de chacun des clients déclarés, que nous appellerons Clients. Il est possible de faire figurer cette structure de données dans le diagramme de structure, par exemple avec la notation suivante:

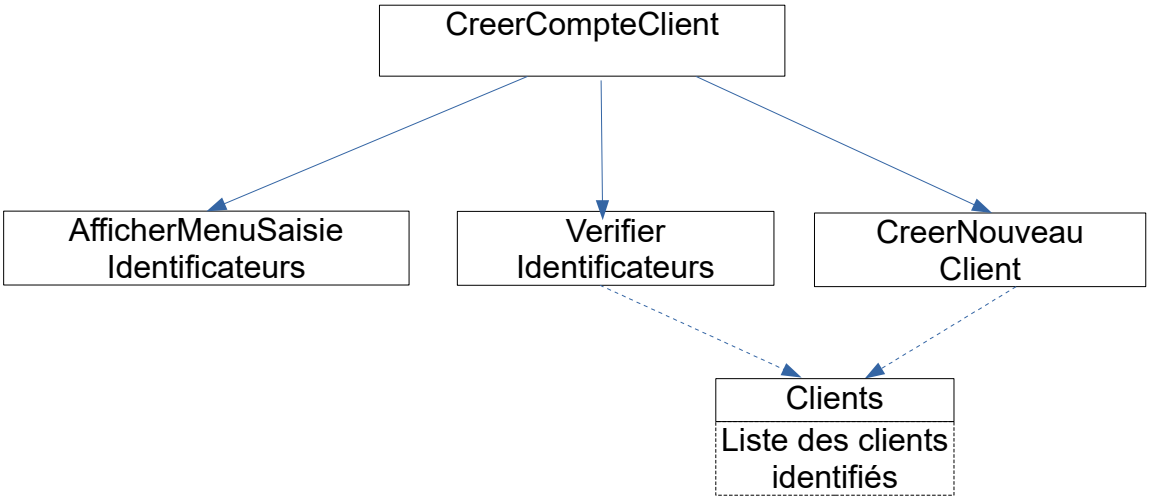


Fig IV.3.3.1-A-A : CreerCompteClient : Diagramme "enrichi" des données.

REMARQUE: pour que le résultat de la conception générale soit utilisable directement par les équipes de conception détaillée, cette représentation du diagramme de structure complété par l'identification des données persistantes et de leur relation avec les fonction doit être accompagnée des informations structurelles suivantes:

- Description de l'interface et de l'algorithme interne de la fonction racine (CreerCompteClient);
- Description de l'interface et des spécifications internes de chaque sous-fonction;
- Description de la structure et du typage des données.

Intéressons nous également au raffinage de la fonction AcheterProduit. Deux données persistantes peuvent être identifiées à cette occasion: la liste des clients, la liste des produits en vente sur le site et la liste des commandes:

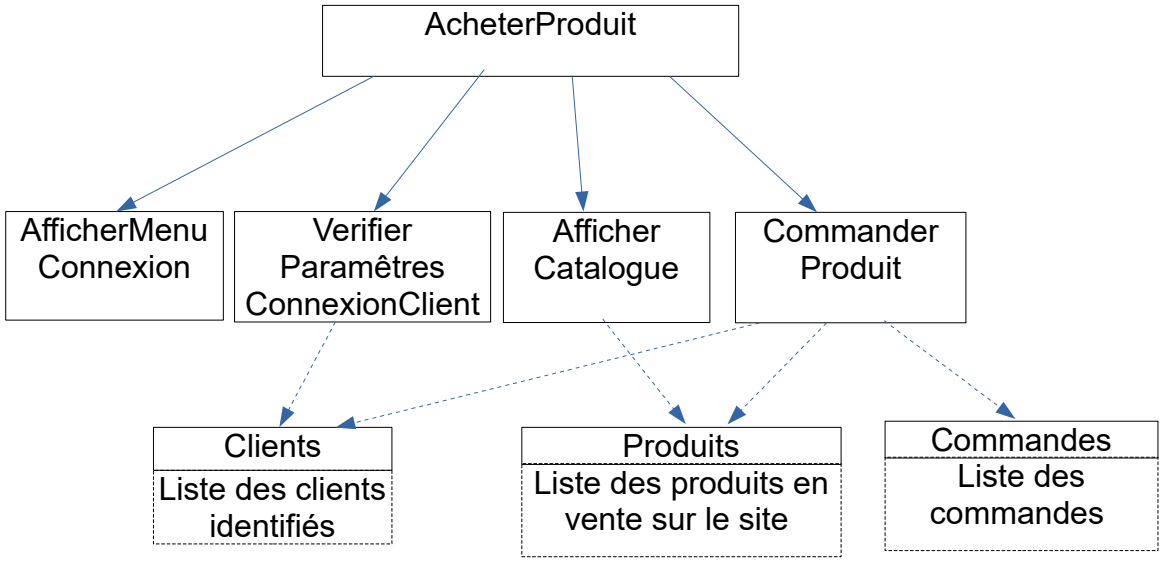


Fig. IV.3.3.1-A-B : AcheterProduit : Diagramme enrichi des données.

De même, le raffinage de la fonction "ConsulterCatalogue" conduira au diagramme enrichi suivant:

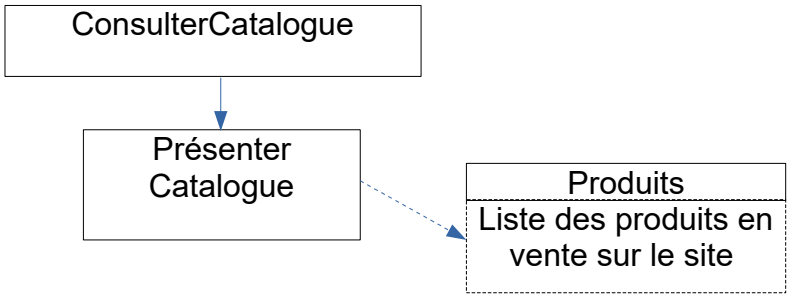


Fig. IV.3.3.1-A-C : ConsulterCatalogue : diagramme enrichi des données

Le diagramme de structure de l'application "enrichi" de l'identification des donnés se présentera alors comme suit:

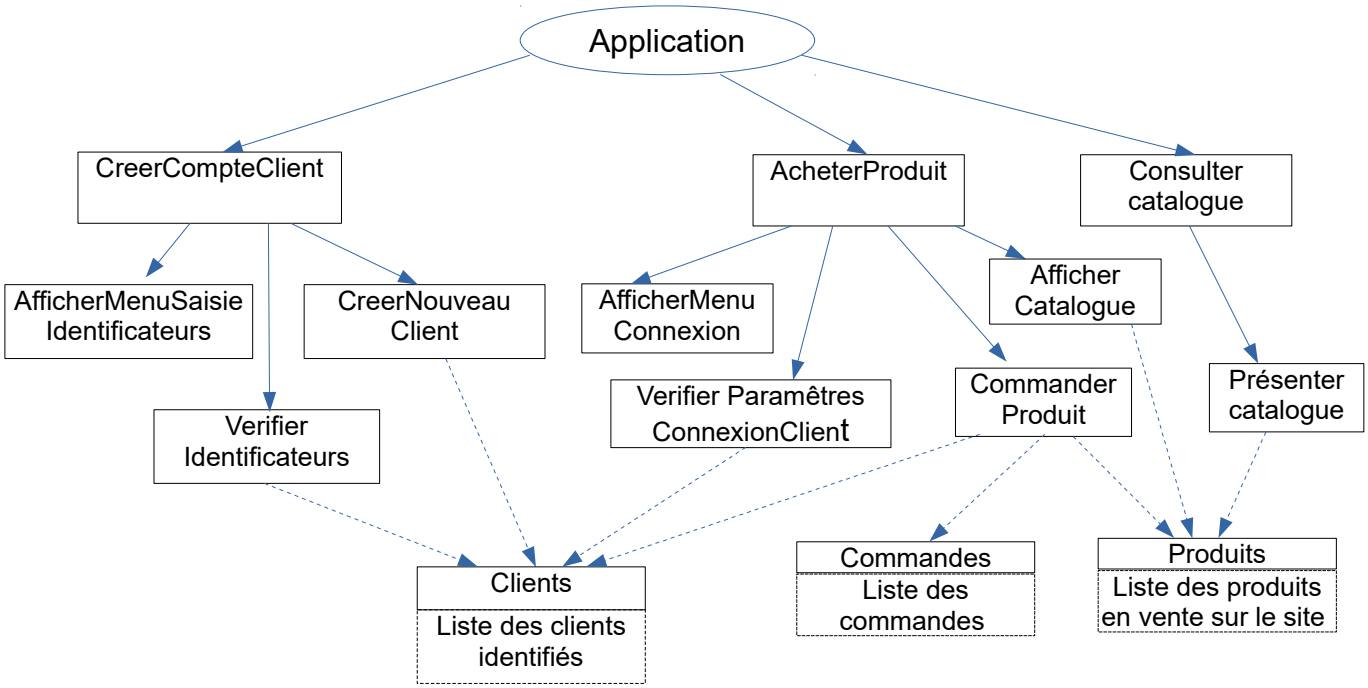


Fig. IV.3.3.1-A-D : Diagramme de structure "fusionnés"

VI.3.3.1.2.IDENTIFICATION DES OPPORTUNITÉS DE RÉUTILISATION:

Une autre critique adressée à la technique de raffinage est de ne pas mettre en évidence les fonctions qui peuvent être réutilisées à plusieurs endroits d'un logiciel (ou, par extension, dans de logiciels différents). Pour pouvoir être exploités, ces cas où la réutilisation est possible doivent de préférence être détectés lors de la CONCEPTION GLOBALE (surtout dans un "cycle en V": dans une conception itérative, ils pourront être pris en compte par le "rétro-ingeneering" effectué à chaque cycle de développement).

Cependant, le diagramme de structure fusionné permet de mettre en évidence ces possibilités de réutilisation. Ainsi, dans le schéma IV.3.3.1-A-D:

- Les fonctions AfficherMenuSaisieIdentificateurs et AfficherMenuConnexion sont probablement identiques: il peut être envisagé de les fusionner en une fonction AfficherMenuIdentification;
- Les fonctions AfficherCatalogue et PrésenterCatalogue sont probablement identiques. On peut également les remplacer par une fonction réutilisable AfficherCatalogueProduits.

Le diagramme de structure suivant fait apparaître les fonctions réutilisées (ces fonctions sont identifiées par un fond vert):

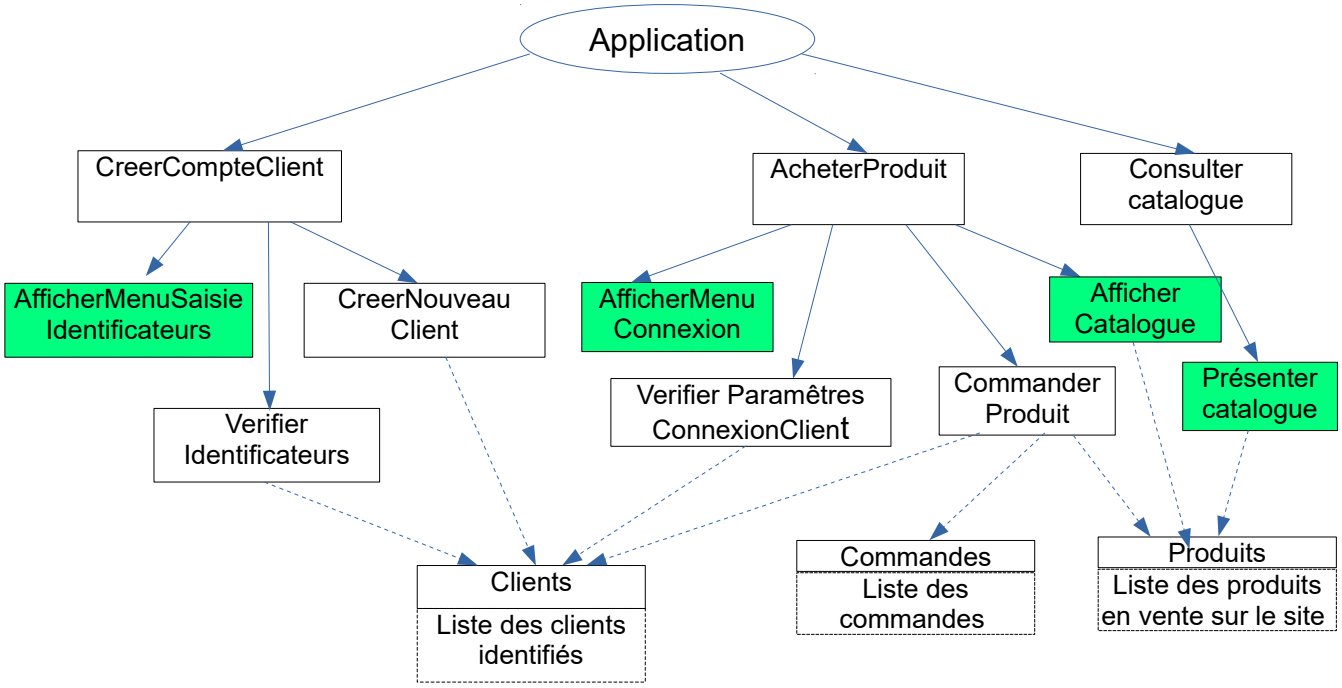


Fig. IV.3.3.1-A-D : Visualisation des opportunités de réutilisation.

En remplaçant des fonctions par des fonctions réutilisables, on obtient:

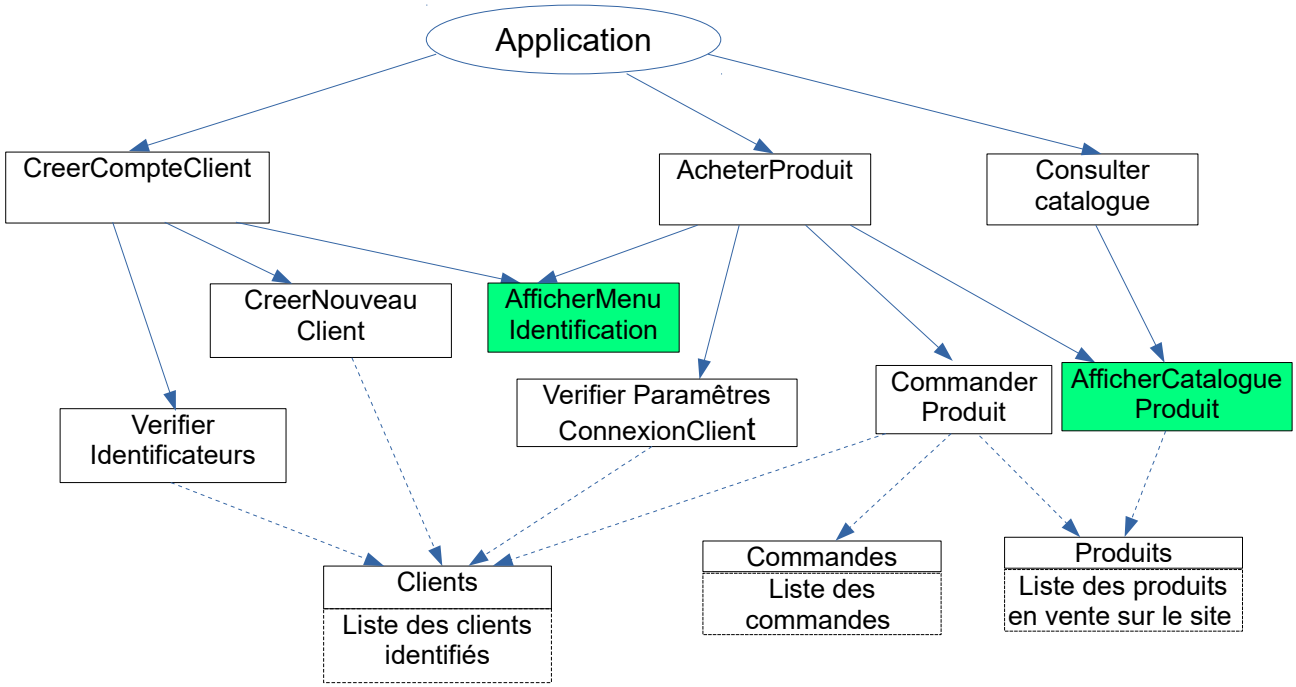


Fig. IV.3.3.1-A-D : Remplacer par des fonctions réutilisables

VI.3.3.1.3.ORGANISER LES FONCTIONS EN COUCHES LOGICIELLES:

RAPPEL: INTÉRÊT DE LA STRUCTURATION EN COUCHES:

Nous avons vu au chapitre III la technique de la structuration des logiciels en couches et les avantages qu'elle apporte du point de vue de la qualité des logiciels:

- **Maîtrise de la complexité des algorithmes**: l'imposition de règles strictes dans les interactions entre composants permet de limiter la complexité du "graphe des appels" en imposant une structure hiérarchique et en évitant les rebouclages de dépendances;
- **Amélioration de la communication à l'intérieur d'une application**, en structurant les échanges entre les différentes couches.
- **Optimisation de la durée du développement** en faisant ressortir des opportunités de réutilisation;
- **Rationalisation et unification de l'architecture:** chaque couche produit des services pour la couche supérieure, éventuellement en utilisant pour cela la couche inférieure (communication de type Client-Serveur).

OBTENTION D'UNE STRUCTURATION EN COUCHES:

A priori, le nombre optimal de couches dépend du volume et de la complexité de l'application. Le tableau ci-dessous résume les principes directeurs des structurations en 3 et 5 couches, qui sont les modèles les plus fréquents:

DESCRIPTION (modèle à 5 couches)	MODÈLE 5 COUCHES	MODÈLE 3 COUCHE
Interface affichage utilisateur, Interfaces homme-machine (affichage et acquisition des interactions utilisateur).	PRÉSENTATION	PRÉSENTATION
Gestion des sessions, habilitations, droits d'accès, gestion des erreurs.	CONTRÔLE	MÉTIER
Implémente la logique "métier" de l'application (Services "métier" enchaînements des règles métier). Ex: virement de compte à compte, souscription d'un prêt en ligne, etc.	SERVICE	
Fournit les services de base du domaine nécessaires aux règles métier. Ex: Gestion d'un compte bancaire, calcul d'un prêt, etc.	DOMAINE	
Gère les données persistantes de l'application (En particulier, la Base de Données).	PERSISTANCE	PERSISTANCE

Pour obtenir par raffinements successifs une structure qui corresponde à un de ces modèles, un moyen simple consiste à procéder comme suit:

*VI.3.3.1.4.**PREMIÈRE PHASE DE RAFFINAGE:***

La première phase de raffinage ne doit laisser subsister dans les fonctions du premier niveau que la logique relative à la réalisation des exigences externes (logique d'utilisation). Les détails des traitements mis en œuvre doivent être repoussés dans la couche de deuxième niveau (logique de métier). Cette première couche constituera la couche PRÉSENTATION.

*VI.3.3.1.5.**DEUXIÈME PHASE DE RAFFINAGE:***

SI, lors de la deuxième phase de raffinage, on s'aperçoit que l'abstraction des traitements de la persistance conduit à des algorithmes de taille et de complexité acceptable, c'est qu'un modèle à trois couches suffit. On peut donc passer à la dernière phase de raffinage.

SINON, si on s'aperçoit que la seule abstraction des traitements de la persistance conduit à obtenir des algorithmes trop complexes, on ne conservera dans la couche de niveau 2 que les traitements relatifs au contrôle (Gestion des sessions, habilitations, droits d'accès, gestion des erreurs). La couche 2 sera donc une couche "CONTRÔLE" Les autres traitements seront renvoyés à la couche inférieure (niveau 3);

*VI.3.3.1.6.**TROISIÈME PHASE DE RAFFINAGE:***

SI les algorithmes de cette couche 3 sont acceptables si l'on fait abstraction des traitements liés à la persistance, cette couche 3 deviendra une couche METIER;

SINON SI ces algorithmes sont trop volumineux ou trop complexes, il faudra envisages de la scinder en deux couches: SERVICES et DOMAINE.

*VI.3.3.1.7.**DERNIÈRE PHASE DE RAFFINAGE:***

Dans les deux cas, les traitements relatif aux données persistantes seront repoussés dans une dernière couche PERSISTANCE.

VI.3.3.2.**STRUCTURATION EN MODULES:**

Celle-ci va consister à regrouper des sous-fonctions en MODULES. Ces modules se présenteront dans ce cas comme des bibliothèques de fonctions dédiées à la gestion d'une donnée.

EXEMPLE: Le diagramme ci-dessus permet de constater que 4 des sous-fonctions permettent de manipuler la donnée Clients tandis que les deux autres manipulent la donnée Produits. Nous pouvons donc regrouper ces fonctions en deux modules: le module CLIENTS et le module PRODUITS qui encapsulent les fonctions correspondantes. D'autre part, les trois fonctions de niveau inférieur peuvent être regroupées dans un module que nous appellerons CONTRÔLE:

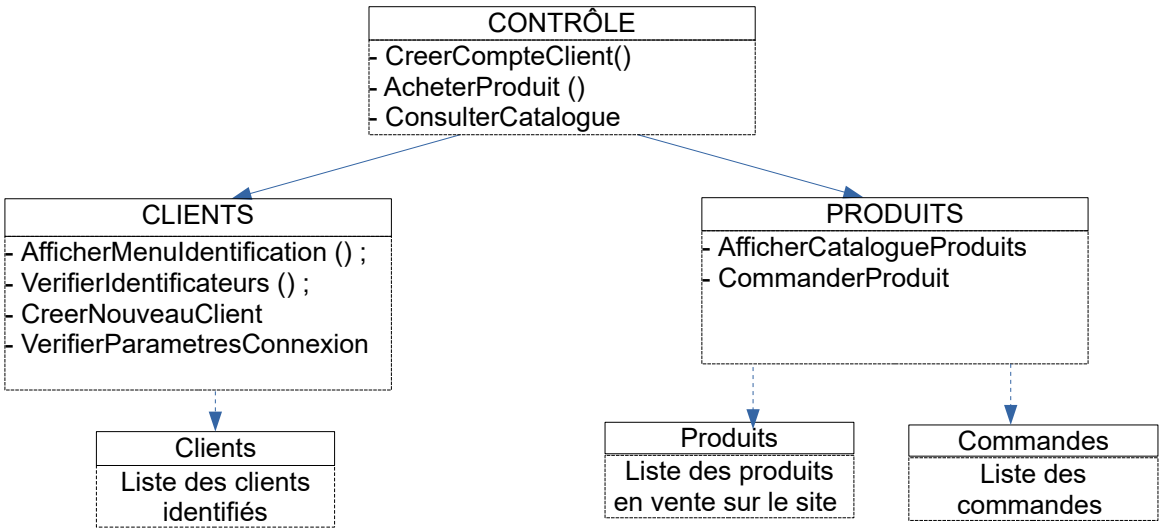


Fig. IV.3.3.2-A : Modularisation de l'architecture.

L'architecture obtenue ainsi est à la fois plus simple à appréhender, plus robuste et plus évolutive que les architectures précédentes. Elle se rapproche d'une conception par objets, avec cependant une différence fondamentale: traitements et données sont considérés séparément.

VI.4.LA DÉMARCHE DE CONCEPTION PAR OBJETS:

VI.4.1.INTRODUCTION:

La conception procédurale a l'avantage d'être très intuitive. Elle a pour inconvénient de ne pas suffisamment prendre en compte les DONNÉES (du moins au début de la conception). Or, les données traitées par une application sont souvent plus STABLES que les traitements: par exemple, dans un traitement de texte, les FONCTIONS offertes à l'utilisateur peuvent évoluer rapidement dans le temps (Ajouts, perfectionnements, etc.) alors que les FORMATS des fichiers restent beaucoup plus stable.

La CONCEPTION PAR OBJETS, au contraire, a pour caractéristique de tenir compte des DONNÉES en même temps que des TRAITEMENTS: en effet, un OBJET encapsule à la fois des TRAITEMENTS et les DONNÉES concernées par ces traitements.

De ce fait, un logiciel conçu suivant la démarche procédurale est en général plus difficile à maintenir que le même logiciel conçu suivant une démarche guidée par les données.

VI.4.2.CONCEPTION OBJETS ET PROGRAMMATION ORIENTÉE OBJETS:

La démarche de Conception par Objets (C.O) est très liée au style de programmation appelé Programmation Orientée Objet (P.O.O). En effet:

- Dans le cas de la C.O, un objet est une ENTITÉ LOGIQUE associant des informations et des traitements agissant sur ces informations;
- Dans le cas de la P.O.O, un objet est une STRUCTURE (au sens des langages informatiques) encapsulant des DONNÉES et des MÉTHODES qui sont en fait des fonctions ou des procédures agissant sur ces informations;

Les deux concepts sont évidemment très voisins, ce qui facilite grandement le passage d'une C.O à sa programmation en un langage "objets" (C++, JAVA, etc.). Cependant, la P.O.O peut être également utilisée dans le cas d'une conception procédurale, car un objet peut être utilisé "à minima" comme une bibliothèque de fonction.

La compréhension de la démarche de conception par objets est difficile pour quelqu'un qui n'a pas un minimum de connaissance des principes généraux de la P.O.O. En cas de difficultés, veuillez vous reporter aux annexes de ce document pour une présentation rapide de cette technique.

VI.4.3.LES ÉTAPES DE LA CONCEPTION PAR OBJETS:

VI.4.3.1.PRINCIPE DU PASSAGE DU FONCTIONNEL A L'OBJET:

La démarche procédurale déduit des exigences (exprimées pas la définition du besoin) les TRAITEMENTS nécessaires à la satisfaction de ces besoins, puis RAFFINE ces traitement jusqu'à obtenir des "briques" assimilables à des FONCTIONS LOGICIELLES directement codables:

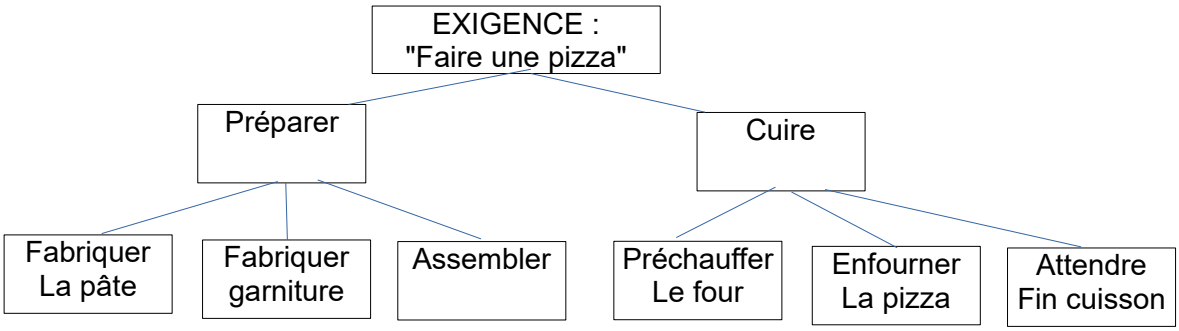


Fig. IV.3.4.1-a : Démarche procédurale.

En revanche, la démarche objets cherche à associer à ces exigences des "OBJETS" susceptibles de concourir à leur réalisation. Ces entités peuvent être des ACTEURS du processus (Utilisateurs humains ou automatiques du logiciel) ou des ACTANTS (entités utilisées au cours du processus).

Ces entités, qui sont souvent appelées à ce niveau des CATÉGORIES, on pour objectif de représenter les objets du monde réel qui interagissent dans la problématique à résoudre. Elles encapsulent des ATTRIBUTS qui sont les données caractéristiques des objets réels qu'elles modélisent:

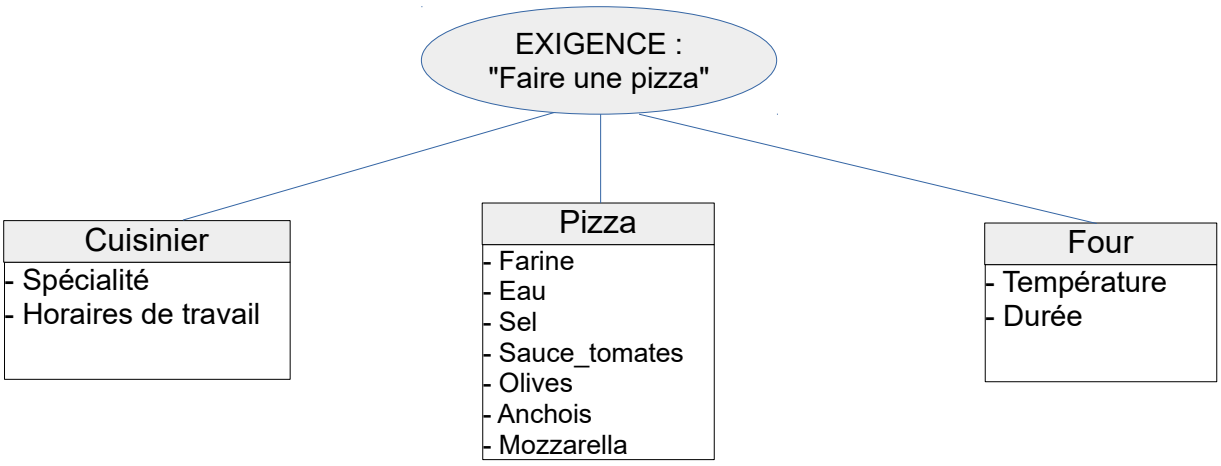


Fig. IV.3.4.1-b: Démarche "objets" (initiale)

La démarche se poursuit alors par l'étude des RELATIONS existant entre les catégories identifiées:

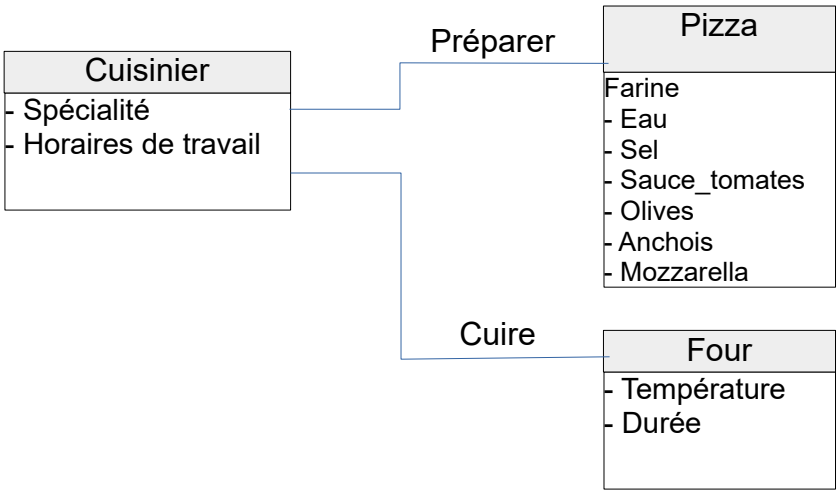
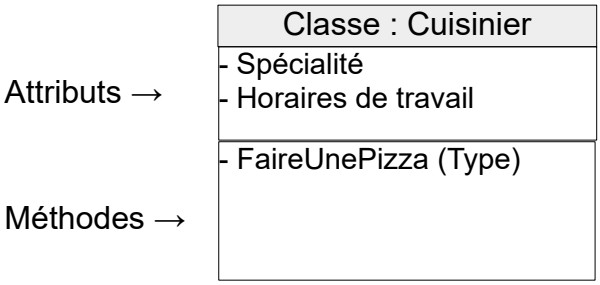


Fig. IV.3.4.1-c: Relations entre catégories

L'étude des relations existant entre ces catégories va permettre d'identifier les interactions nécessaires pour assurer le fonctionnement du logiciel. Ces interactions permettront d'identifier les futures MÉTHODES des classes qui seront bâties à partir des catégories).

Par exemple, a la catégorie CUISINIER pourra être associé le traitement FaireUnePizza avec un paramètre d'appel qui pourra être Type (le type de pizza à faire). Cette catégorie munie de ce traitement devient une CLASSE:



REMARQUE: cette classe Cuisinier pourra s'enrichir de nouvelles méthodes comme: **FaireBoeufBourguignon** (Quantité) ou **FaireGrillade** (Quantité, TypeViande) en fonction des nécessités de l'application en cours de développement ou au cours d'un développement ultérieur.

VI.4.4.**EXEMPLE:**

VI.4.4.1.**DESCRIPTION DU CAS:**

Reprenons l'exemple du site marchand utilisé dans le cadre de la conception procédurale. La S.T.B de cette application énonce les exigences suivantes:

- Un visiteur du site doit pouvoir consulter le CATALOGUE du site;
- Un VISITEUR du site doit pouvoir ouvrir un compte CLIENT;
- Un CLIENT doit pouvoir acheter un PRODUIT figurant au CATALOGUE du site;

Comme précédemment, afin de ne pas trop compliquer cet exemple, nous considérerons que le client ne peut acheter qu'un type de produit à la fois. D'autre part, il a été fait abstraction des mécanismes de facturation et de paiement.

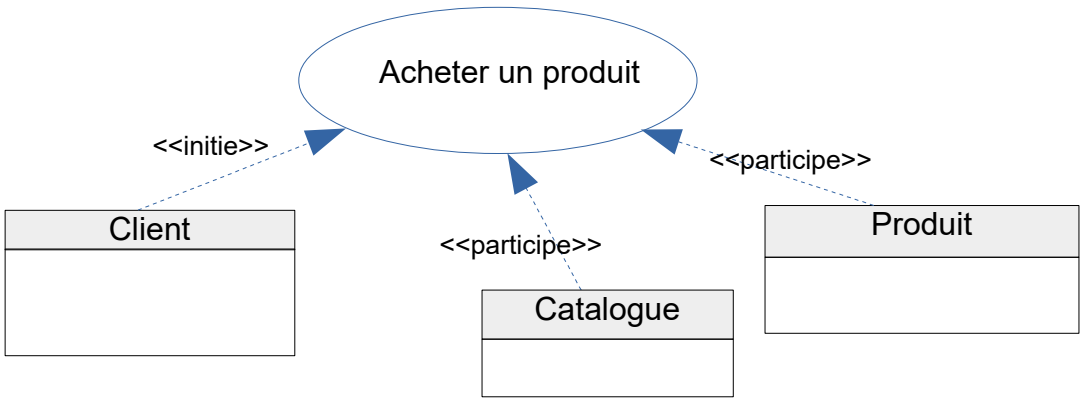
VI.4.4.2.**ÉTUDE DU CAS D'UTILISATION "Acheter un produit":**

Dans un premier temps, nous allons étudier le cas d'utilisation:

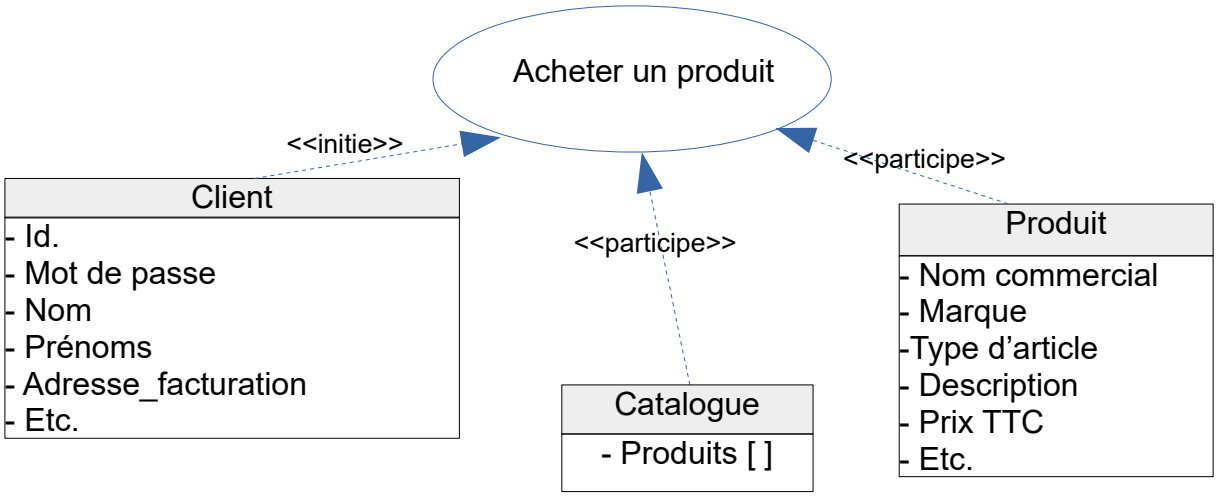
Un CLIENT doit pouvoir acheter un PRODUIT figurant au CATALOGUE du site;

Première étape: identification des catégories principales et de leurs attributs

En première analyse, les catégories qui participent à l'achat d'un produit sur un site web sont le CLIENT (qui initie la démarche d'achat), le CATALOGUE(qui permet au client de trouver le produit qu'il désire acheter) et le PRODUIT choisi qui va être commandé par le client:

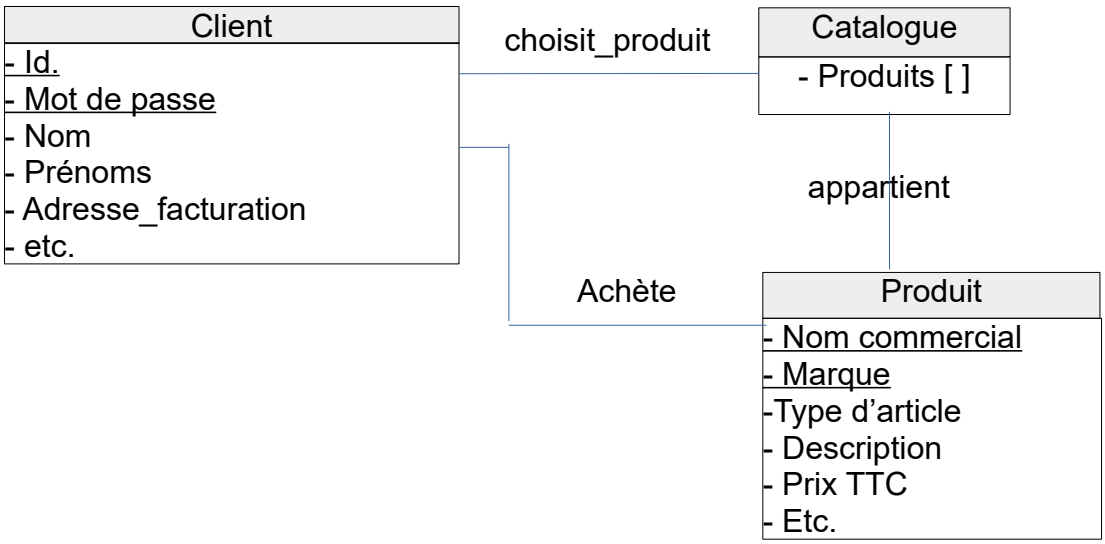


A chacune de ces catégories, nous pouvons associer des ATTRIBUTS. Par exemple, dans une action d'achat, un client est caractérisé par ses identificateurs de connexion (Identificateur client et mot de passe), mais a aussi besoin des paramètres personnels qui lui permettront d'effectuer la commande. A ce niveau, nous identifierons les paramètres suivants:

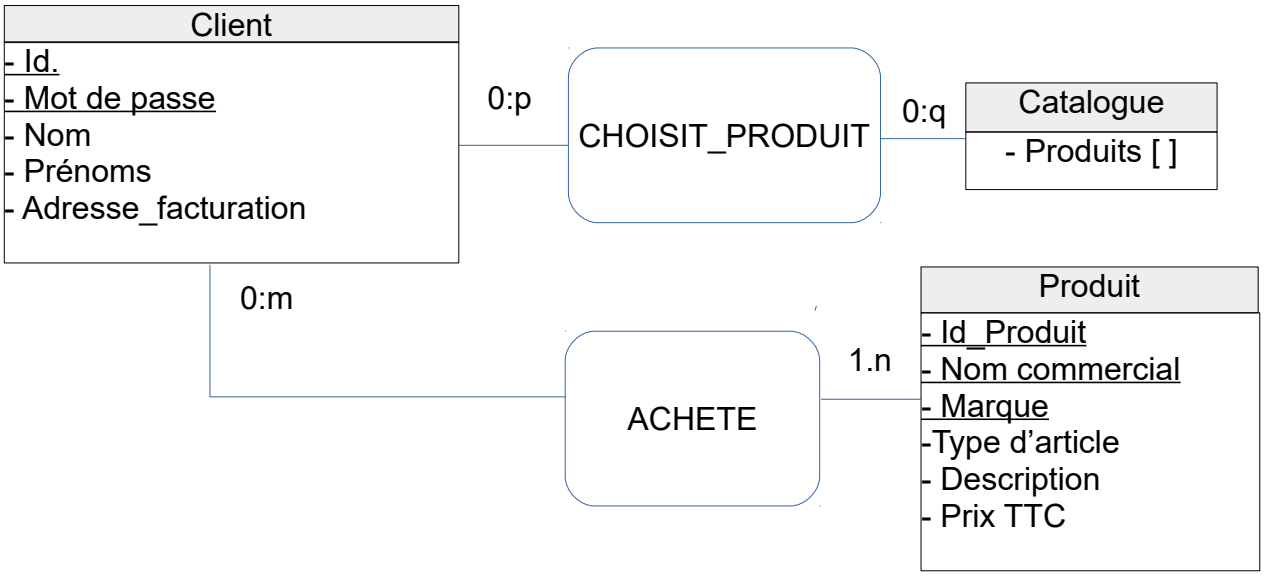


Deuxième étape: étude des relations entre catégories

Nous allons maintenant étudier les relations existant entre ces catégories. Pour cela, nous pouvons utiliser un diagramme de classes U.M.L (simplifié):



REMARQUE: Ce diagramme constitue un MODÈLE LOGIQUE de l'application à réaliser. Nous pouvons remarquer que, si l'on fait abstraction des différences de présentation, il est probablement très proche du modèle CONCEPTUEL de la base de données de l'application:



Ceci prouve que la conception par objets est une démarche guidée par les données.

Troisième étape: identification des méthodes

A partir du diagramme ci-dessus, nous pouvons déduire les MÉTHODES qui seront nécessaires à l'acte d'achat. La démarche est la suivante:

Chacune de ces catégories modélise un objet intervenant dans l'acte d'achat. Pour chacune d'entre elles, il convient de se poser la question: "comment peut-on l'utiliser dans le cadre d'un achat?".

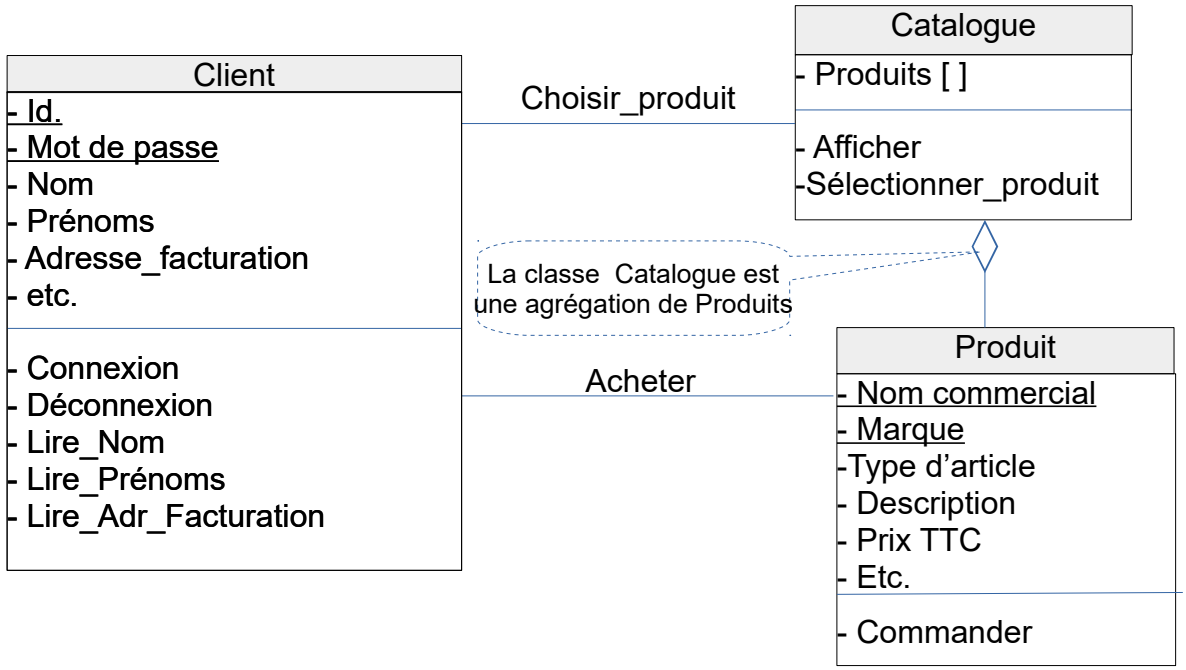
EXEMPLE: Si nous appliquons cette démarche à la catégorie Catalogue, nous trouverons les deux utilisations suivantes: afficher le catalogue et sélectionner un produit dans le catalogue. Nous aurons donc besoin de munir cette catégorie de deux méthodes de comportement: **Afficher** et **Sélectionner_produit**.

Le tableau suivant présente pour chaque catégorie une liste des méthodes nécessaires pour résoudre le cas d'utilisation "acheter un produit":

NOM	FONCTION	PARAMÈTRES	VALEUR RETOURNÉE
Classe CLIENT			
Connexion	Effectue la connexion d'un client	Id: identifiant client Mdp: mot de passe	Objet Client_Connecté ou false si echec
Déconnexion	Déconnecte un client		Aucune
Lire_Nom	Retourne le nom d'un client	Aucun	Chaîne Nom du client
Lire_Prénoms	Retourne les prénoms d'un client	Aucun	Chaîne Prénoms du client

NOM	FONCTION	PARAMÈTRES	VALEUR RETOURNÉE
Lire_Adr_Facturation	Retourne l'adresse de facturation du client	Aucun	Chaîne Adresse de facturation
Classe Catalogue			
Afficher	Afficher les produits du catalogue	Type_aff: type d'affichage (par catégories de produits ou par marques),	Aucune
Sélectionner_produit	Sélectionner un produit du catalogue	Aucun	Objet Produit_sélectionné
Classe Produit			
Commander	Commander le produit	Client: objet Client	Entier: nombre d'exemplaires commandés ou false (-1) si échec de la commande.

Munies des méthodes ci-dessus, les catégories Client, Catalogue et Produit_choisi peuvent être assimilées à des CLASSES, au sens de la programmation orientée objets:



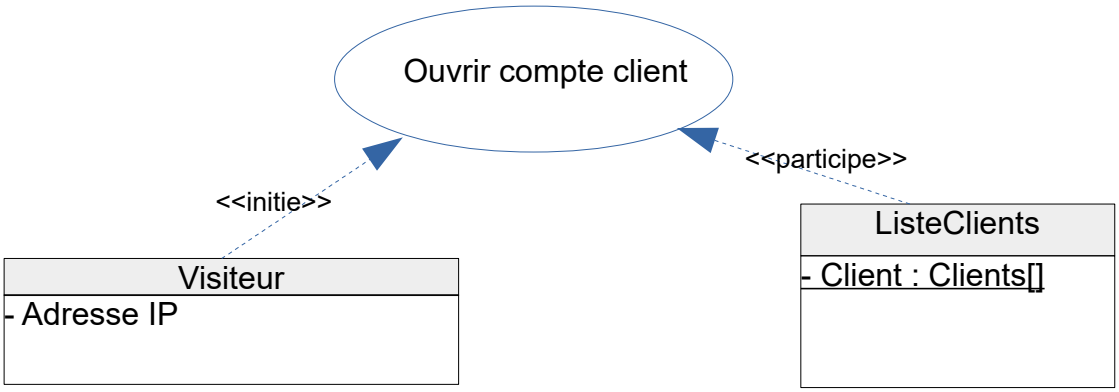
REMARQUES:

- La classe Catalogue est composée d'une liste d'objets de la classe Produit: ce sont les différents articles du catalogue:
- A ce niveau, nous n'avons pas encore défini de constructeur spécifique pour ces différentes classes.

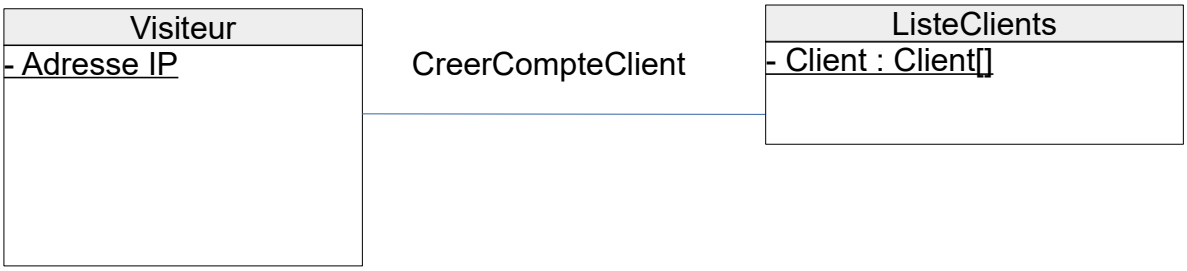
VI.4.4.3.ÉTUDE DU CAS D'UTILISATION "Ouvrir un compte client":

Nous allons appliquer le même traitement au cas "Ouvrir un compte client":

Identification des catégories et des attributs principaux:



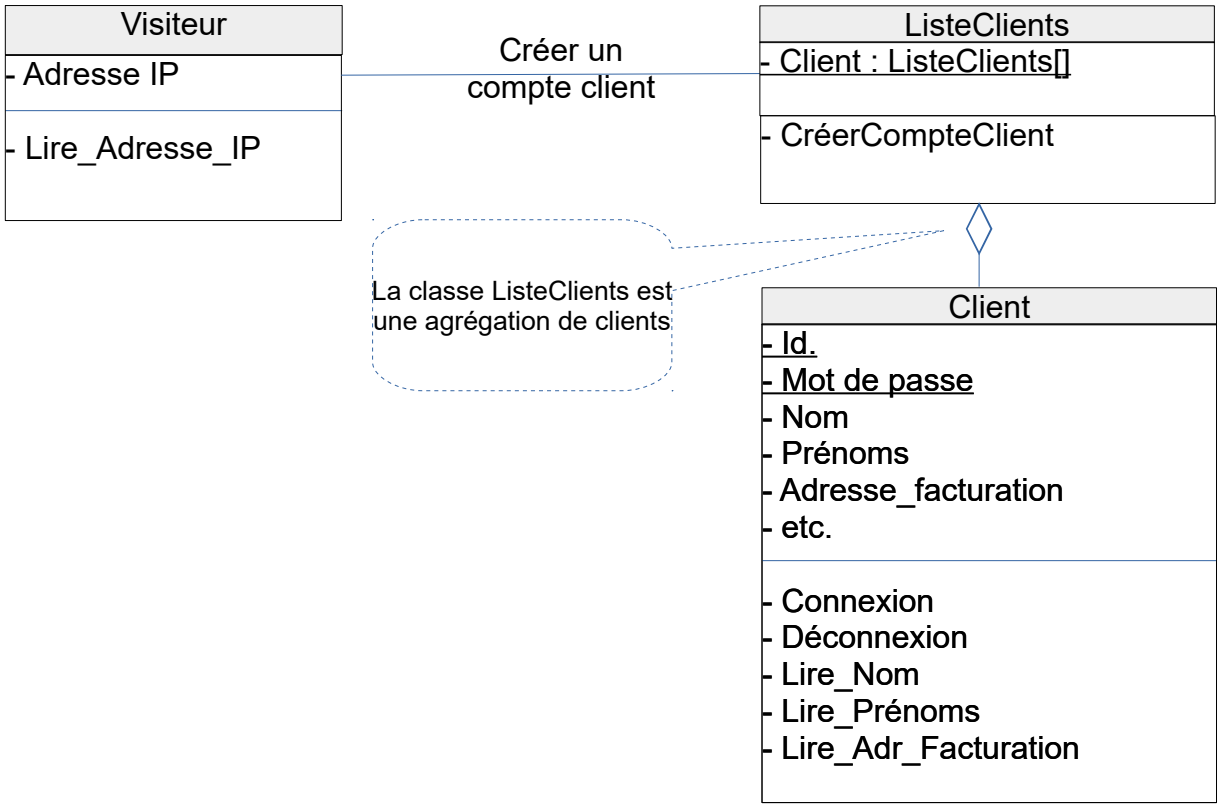
Relations entre catégories:



Identification des méthodes:

NOM	FONCTION	PARAMÈTRES	VALEUR RETOURNÉE
Classe ListeClients			
CreerCompteClient	Créer un compte client	Id: identifiant client Mdp: mot de passe	Aucune
Classe Visiteur			
Lire_Adresse_IP	Retourne l'adresse IP du visiteur	Aucun	Chaîne adresse ip du visiteur.

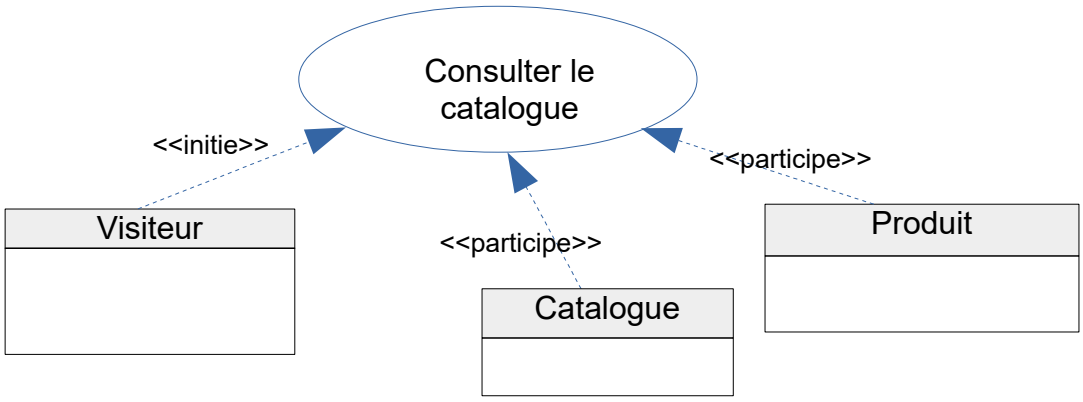
Diagramme de classes:



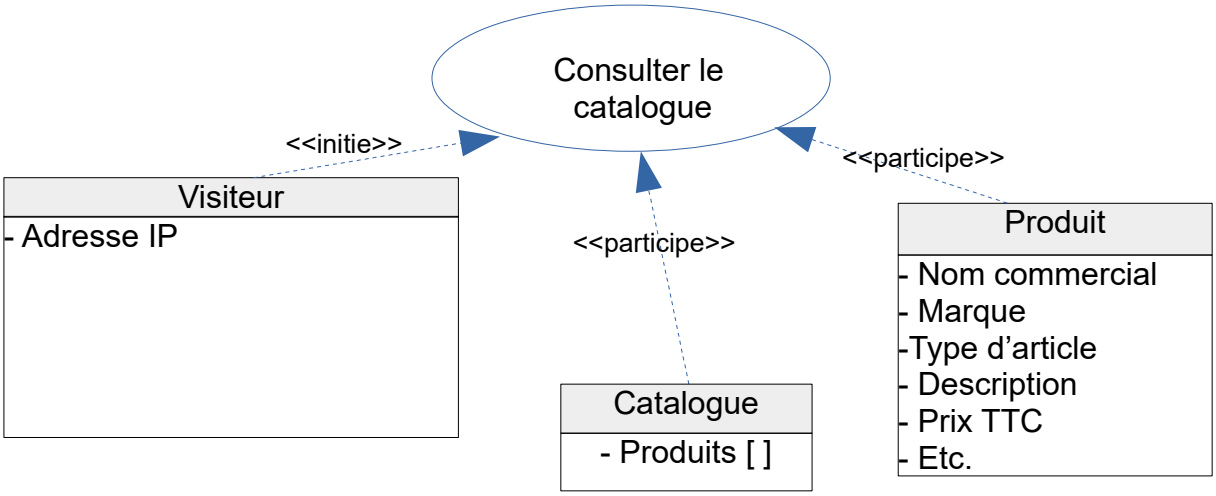
VI.4.4.4.ÉTUDE DU CAS D'UTILISATION "ConsulterCatalogue":

Même traitement pours le cas "Consulter le catalogue":

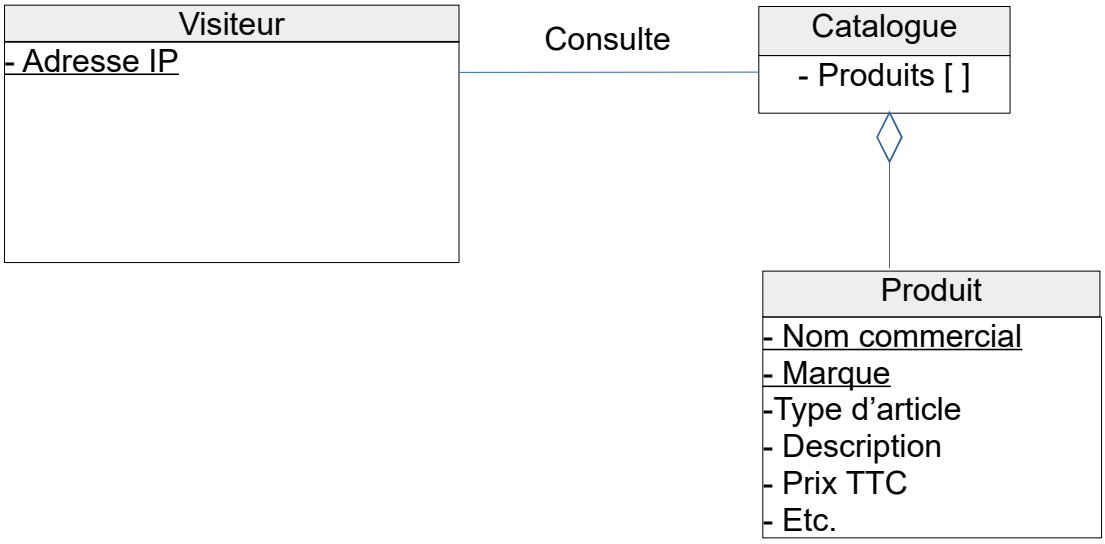
Diagramme de cas d'utilisation:



Identification des attributs des catégories:



Étude des relations entre catégories:

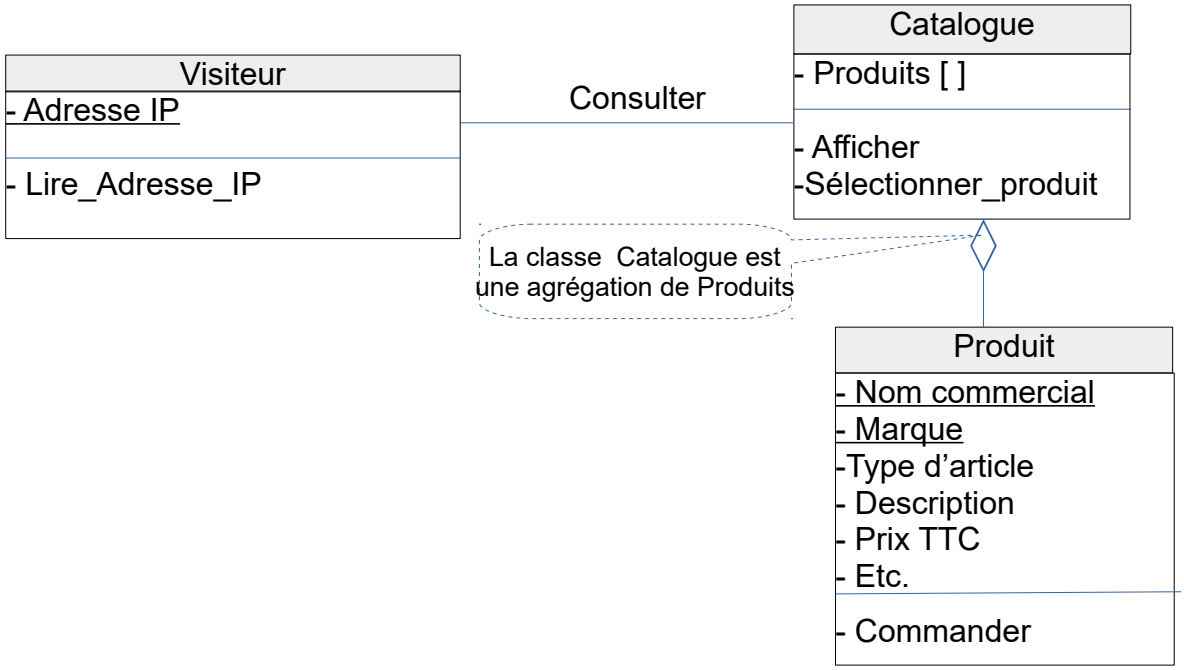


Identification des méthodes

NOM	FONCTION	PARAMÈTRES	VALEUR RETOURNÉE
Classe Visiteur			
Lire_Adresse_IP	Retourne l'adresse IP du visiteur	Aucun	Adresse IP du visiteur
Classe Catalogue			

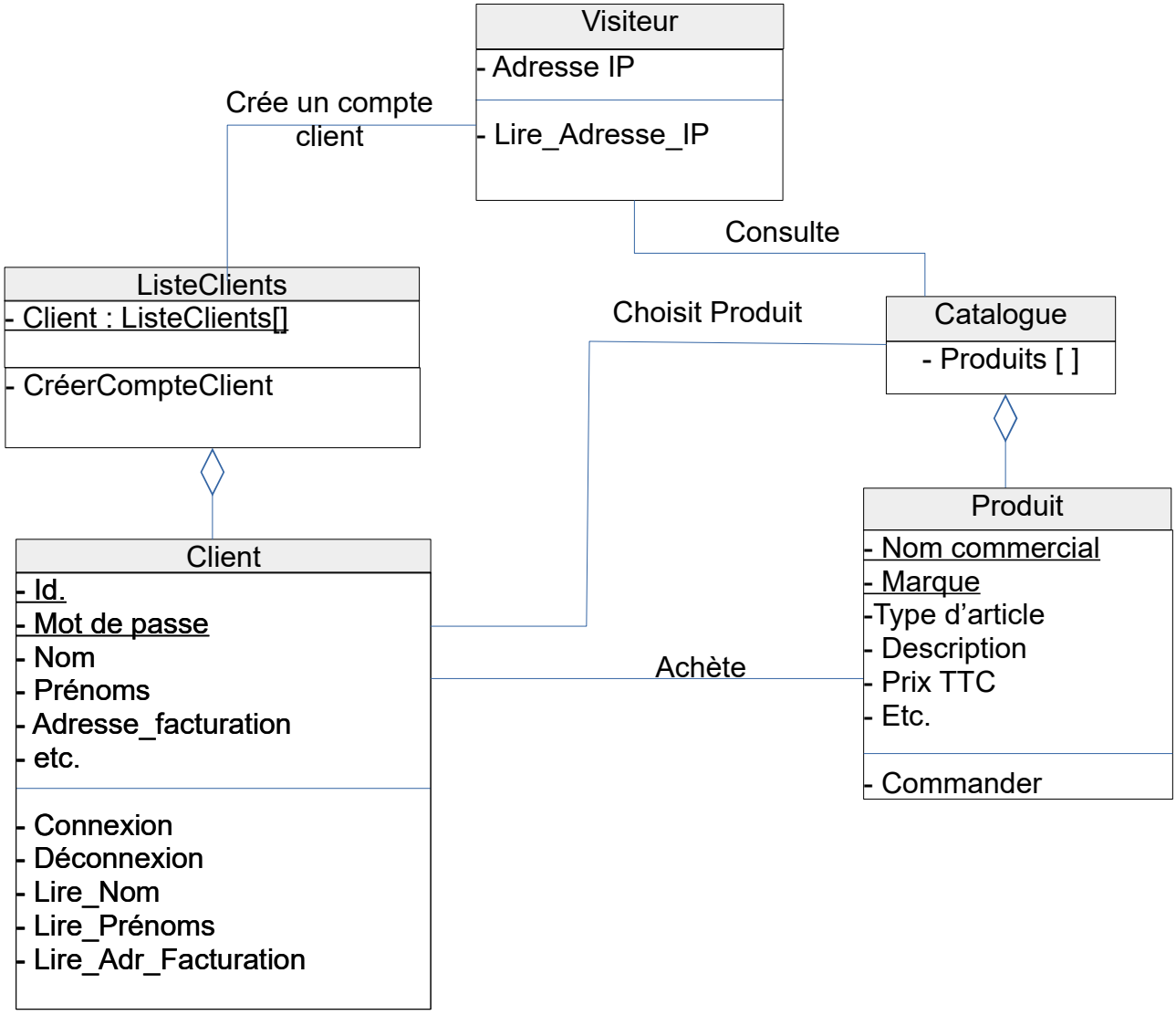
NOM	FONCTION	PARAMÈTRES	VALEUR RETOURNÉE
Afficher	Afficher les produits du catalogue	Type_aff: type d'affichage (par catégories de produits ou par marques),	Aucune
Sélectionner_produit	Sélectionner un produit du catalogue	Aucun	Objet Produit_sélectionné
Classe Produit			
Commander	Commander le produit	Client: objet Client	Entier: nombre d'exemplaires commandés ou false (-1) si échec de la commande.

Diagramme de classes:



VI.4.4.5.ÉLABORATION DU DIAGRAMME DE CLASSE GÉNÉRAL:

Compte-tenu de ce qui précède, le diagramme de classe de l'application peut se présenter comme suit:



VII.ÉTAPE V - SYNTHÈSE:

VII.1.BILAN DES DÉMARCHES DE CONCEPTION :

A l'issue des démarches de conception préliminaire précédentes, nous disposons des résultats suivants:

1. Une ANALYSE COMPORTEMENTALE du logiciel comprenant :
2. Un MODÈLE D'ACTIVITÉ de l'application comprenant L'identification et la caractérisation :
   - Des ACTIVITÉS LOGICIELLES à mettre en œuvre pour satisfaire à la S.T.B de l'application ;
   - Des FLUX D'INFORMATIONS échangés par ces activités;
   - Des RESSOURCES communes à ces activités ;
   - Des SITUATIONS DE CONCURRENCE pour les accès de ces activités à ces ressources ;
   - Des TEMPS DE RÉPONSE à satisfaire.
3. Une SOLUTION ARCHITECTURALE pour la plate-forme d'accueil de l'application ainsi que pour la répartition de l'application sur cette plate-forme ;
4. Un MODÈLE LOGIQUE de l'application comprenant les définition externe d'une collection de modules (classes ou bibliothèques logicielles) destinés à encapsuler la logique algorithmique des traitements de l'application (interfaces de programmation et définition fonctionnelle des traitements affectés).

La fourniture N°3 permet de commencer immédiatement les travaux de réalisation ou de mise à niveau de la plate-forme d'accueil :

- Approvisionnement des matériels et logiciels ;
- Intégration ;
- Vérification du fonctionnement et des performances.

La synthèse des fournitures N°1, 2, et 4 va permettre de planifier et de lancer les travaux de conception détaillée, de codage et d'intégration.

VII.2.OBJECTIFS DE LA SYNTHÈSE:

L'objectif de cette synthèse consiste, en s'appuyant sur les résultats des études système, logiques et comportementales, en l'élaboration d'une solution pour regrouper les activités dans des processus de priorités différenciées, en fonction :

- Des **contraintes temporelles** à respecter ;

- Du soucis d'**optimisation de l'utilisation des ressources** (CPU, mémoire vive, etc);
- Des impératifs de **sûreté de fonctionnement** (minimisation des situations de conflits, limitation des "effets de de bords" en cas de dysfonctionnement).

VII.3.DÉMARCHE DE REGROUPEMENT :

VII.3.1.VUE GÉNÉRALE:

Il s'agit de définir et caractériser les différents PROCESSUS qui seront activés lors de l'exécution de l'application. Pour chacun de ces processus, il faudra préciser :

- Les ACTIVITÉS qu'il encapsulera et les THREADS qui contrôleront leur exécution ;
- Les caractéristiques de SCHEDULING qu'il faudra employer (temps partagé ou priorités préemptives, niveau de priorité logicielle);
- Les MESSAGES et SIGNAUX échangés avec les autres processus.

Ces choix doivent évidemment être justifiés par un certain nombre de critères d'ordre technique ou méthodologique. Nous pouvons citer en particulier :

- La ***priorité à accorder à chaque activité*** dans l'accès aux ressources CPU ;
- Les ***contraintes temporelles attachées à chaque activit***é (niveau de contrainte "temps réel") ;
- Les ***interactions et couplages*** entre activités ;
- La limitation des ***effets de bord*** entre activités ***;***
- Dans un système à plusieurs processeurs, la ***possibilité de répartir l'exécution*** des activités sur des processeurs différents (exécution en parallélisme réel) ;

Les paragraphes suivants explicitement ces différents principes .

VII.3.2.INFLUENCE DE LA PRIORITÉ À ACCORDER A CHAQUE ACTIVITÉ :

REMARQUE : *nous parlons ici de priorités PRÉEMPTIVES, dans le cadre d'une exécution en mode prioritaire.*

Toute activité identifiée comme prioritaire par rapport aux autres activités de l'application gagne à être encapsulées seule dans un processus de priorité plus élevée que les autres processus de l'application. Cette disposition, qui assure à cette activité prioritaire un accès suffisant à la ressource CPU même en cas de "congestion" des autres activités, permet de garantir sa durée d'exécution et le déterminisme de cette durée : elle est donc indispensable si cette activité est soumise à des contraintes temps réel élevées.

EXEMPLE : *il est recommandé de séparer les activités INTERACTIVES (interfaces humain-machine) dont les contraintes "temps réel" sont faibles, des activités de traitement de processus physiques dont les contraintes "temps réel" sont, en général, plus élevées. Ces deux types d'activités seront donc confiées à des processus différents.*

VII.3.3.INFLUENCE DES CONTRAINTES TEMPORELLES :

- Lorsque des activités concourent à la délivrance de données, de commandes ou de signalisations dans des délais contraints, le regroupements de ces activités dans un même processus (et même dans le même thread) permet de minorer la durée d'exécution et d'augmenter le déterminisme des dates. De ce fait, les activités à contraintes temporelles peu sévères (comme le traitement des IHM) doivent être encapsulées dans des processus de priorité basse et peuvent être exécutées en temps partagé.
- La commutation de contexte entre threads de processus différents étant beaucoup plus longue que la commutation entre threads d'un même processus, le regroupement d'activités dans un même processus permet de gagner en vitesse d'exécution. Ce fait peut intervenir en faveur du regroupement d'activité commutant fréquemment de l'une à l'autre dans un contexte temps réel "DUR".

VII.3.4.INFLUENCE DES INTERACTIONS ET COUPLAGES AVEC LES AUTRES ACTIVITÉS :

Le regroupement d'activités à l'intérieur d'un même processus permet à celles-ci de partager des espaces de données communs, sans recourir à des zones de mémoire partageables inter-processus (sheared memories). Ces données peuvent faire partie des DONNÉES STATIQUES du processus (créées au lancement des exécutables) ou correspondre à des DONNÉES DYNAMIQUES réservées en cours d'exécution par des primitives d'allocation temporaire de mémoire dans le TAS (heap) du processus (par exemple, par la primitive malloc en langage c).

En conséquence, le fait que des activités échangent beaucoup d'informations ou partagent beaucoup de ressources est un argument pour les regrouper dans un même processus.

Remarquons que les accès de threads d'un même processus à des ressources communes (en particulier, a des espaces de données partagées) implique que ces accès soient soit contrôlés par des mécanismes d'exclusion mutuelle (programmation concurrente), soit synchronisés (programmation synchrone).

VII.3.5.LIMITATION DES RISQUES D'EFFETS DE BORD :

Le regroupement d'activités dans un même processus peut augmenter les risques de dysfonctionnement. En effet, certaines anomalies de fonctionnement se produisant dans un thread d'un processus (division par zéro, double ouverture d'un port d'entrée réseau, etc.) peuvent provoquer l'arrêt de ce processus, avec tous les traitements qu'il supporte.
Il est donc recommandé d'encapsuler les activités dont le fonctionnement est "critique" pour l'application dans des processus séparés.

REMARQUE: deux activités s'exécutant dans des processus différents peuvent malgré tout être affectées par des interblocages. Il suffit pour cela qu'elles se partagent une ressource commune. Par exemple :

- Il suffit que l'une d'entre elles "tombe" (par exemple, à cause d'une alarme système non gérée) alors qu'elle occupe cette ressource. La ressource ne sera jamais libérée, bloquant ainsi le fonctionnement de l'autre processus ;
- Elles peuvent être affectées par des phénomènes d'interblocages comme les "dead locks" ou de "cas de famine" (voir en annexe).

VII.3.6.INFLUENCE DU NOMBRE DE PROCESSEURS DISPONIBLES:

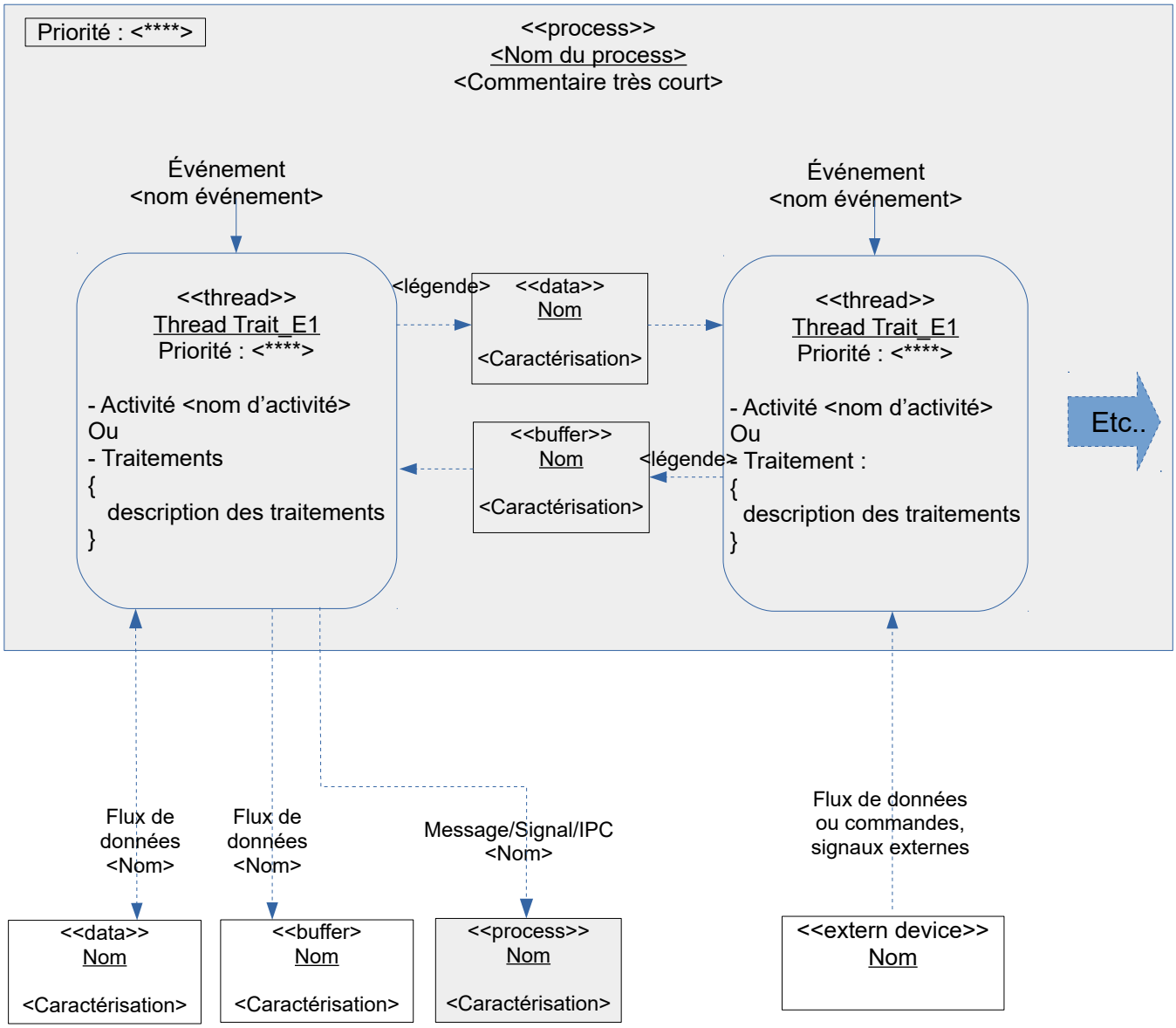
Lorsque la plate-forme d'exécution dispose de plusieurs processeurs, l'exécution de chacune des activité très prioritaires peut être distribuée à des processus différents. Cette distribution peut être décidée à priori ou automatiquement par un "système de répartition de charge" (load balancing).

Dans un système informatique, les activités propres au système d'exploitation coexistent avec les activités liées aux applications. Ces activités (gestion des entrées-sorties, surveillance, scheduling, etc.) ont pour particularité de survenir d'une manière totalement asynchrone des activités applicatives. Certaines de ces activités sont traitées d'une manière plus prioritaire que celles des applications. D'autres peuvent être très voraces en capacité mémoire ou en puissance CPU. De ce fait, une bonne pratique consiste à consacrer un processeur particulier à l'exécution du système d'exploitation, les autres processeurs traitant les tâches applicatives.

VII.4.MODÉLISATION GRAPHIQUE DE LA DÉMARCHE DE REGROUPEMENT :

Les diagrammes décrits ci-après ont pour but de modéliser la démarche de regroupement des activités en processus.

REMARQUE : *ces diagrammes, que nous appellerons DIAGRAMMES DE PROCESSUS, sont proposés ici dans un but pédagogique, car ils constituent à notre sens des outils commodes pour donner une représentation graphique de la démarche de transition entre la notion d'activité et les notions de processus et de threads.*



COMMENTAIRES SUR LA NOTATION :

Dans le diagramme ci-dessus :

- Les rectangle à fond grisé représentent des processus logiciels. Le processus est identifié par le stéréotype <<process>> suivi du nom du processus et d'un commentaire très court. Le cartouche **Priorité : <PP>** situé en haute et à gauche permet de spécifier la priorité PP du processus ;
- A l'intérieur de ces rectangles grisés :
  - Les "surfaces arrondis" représentent les threads qui s'exécutent dans ces espace. Elles sont surmontées de la représentation de l'événement déclencheur du thread. Si le thread est le thread initial du process, on indiquera "thread initial" à la place de l'indication d'événement ;
  - Les rectangles stéréotypés <<buffer>> représentent des tampons réservés à l'intérieur du process et gérés comme des files fifo ;
  - Les rectangles stéréotypés <<data>> représentent des données partagées par plusieurs threads internes ;
  - Les interactions entre ces objets et les threads sont identifiées par des flèches légendées ou non.
- Les symboles graphiques threads sont identifiés par le stéréotype <<thread>> suivi du nom du thread.
- A l'intérieur du symbole thread peuvent être indiqués :
  - Les noms des activités identifiées dont l'exécution est contrôlée par ce thread, sous la forme : - <nom activité> ;
  - Des "descriptions de traitements" dont l'exécution est contrôlée par ce thread, sous la forme : - Traitement : {description du traitement (texte libre ou pseudo-code)} ;
  - La priorité relative du thread par rapport à celle du processus, sous la forme. **Priorité:<PR>** .
- A l'extérieur du rectangle grisé représentant le processus peuvent être représentés :
  - Les rectangles stéréotypés <<buffer>>, représentant ici des tampons réservés à l'extérieur des processus et gérés comme des files fifo ;
  - Les rectangles stéréotypés <<data>> représentent des données réservés à l'extérieur des processus (shared memories);
  - Les rectangles grisés stéréotypés <<process>> représentent des processus de l'environnement interagissant avec le processus décrit ;
  - Des sources ou des puits de données, de commandes ou de signaux situées dans la périphérie de l'application (stéréotypées <<extern device>>).
- Les interactions du processus avec son environnement sont représentées par des flèches en pointillé légendées. Elles représentent des flux de données ou de

commandes ou encore des IPC échangés avec les autres processus de l'application ou la périphérie.

REMARQUE : les valeurs des priorités absolues des processus et relatives des threads dépendent du système d'exploitation choisi pour la plate-forme.

VII.5.**PRISE EN COMPTE DES RÉSULTATS DE LA CONCEPTION LOGIQUE :**

VII.5.1.**RAPPELS:**

Rappelons que dans un environnement d'exécution MULTITÂCHES, l'exécution des différentes activités logicielles est contrôlée par des THREADS (ou fils d'exécution en français). Ces différents threads sont regroupés dans différents PROCESSUS.

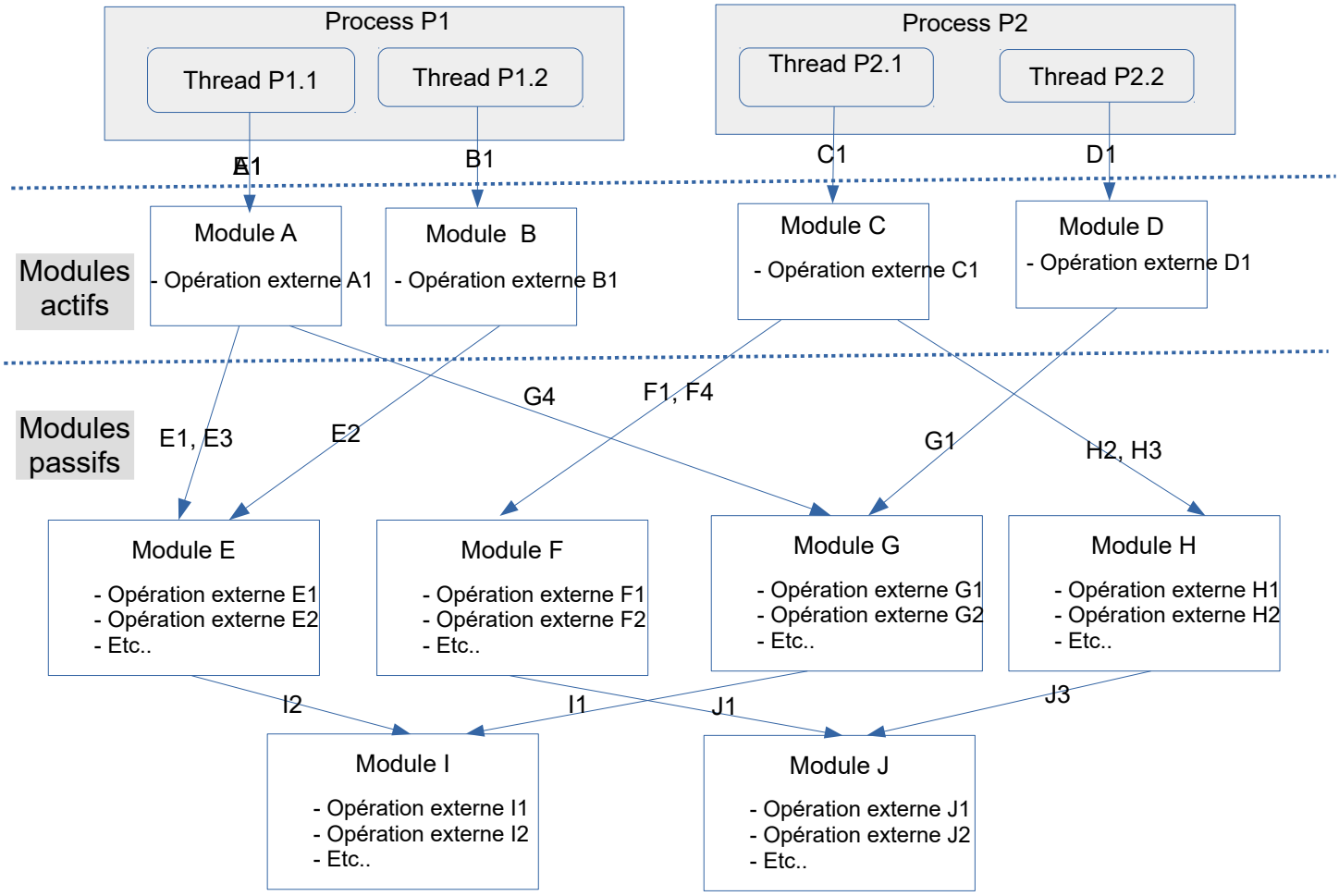
A chaque thread est associé un POINT D'ENTRÉE vers un PROGRAMME (exécutable ou interprétable) dont il va contrôler l'exécution ou l'interprétation. Un point d'entrée est soit une MÉTHODE, soit une OPÉRATION EXTERNE (c'est à dire "publique") appartenant à un composant du programme en question. Ce type de composant directement contrôlé par un thread est appelé COMPOSANT ACTIF (en conception objet, on parle d'objet actif, en conception modulaire, de module actif). Les composants qui ne sont pas DIRECTEMENT activés par un thread sont dits PASSIFS.

VII.5.2.**DÉMARCHE D'INTÉGRATION DE LA CONCEPTION LOGIQUE :**

La conception logique a permis de construire un MODÈLE LOGIQUE de l'application se présentant sous la forme d'une hiérarchie de composants architecturaux (classes ou modules) encapsulant la logique fonctionnelle de l'application. Remarquons qu'à ce niveau de l'étude, ces composants sont des "boîtes noires" dont seules les interfaces externes ont été décrites.

Le développement interne de ces composants fera l'objet de la phase de "conception détaillée-codage-tests unitaires".

Le diagramme ci-dessous résume cette démarche :



VIII.EXEMPLE SIMPLE DE CONCEPTION GLOBALE :

VIII.1.EXPOSÉ DU CAS :

VIII.1.1.GÉNÉRALITÉS :

L'application à étudier participe au système de pilotage automatiquement d'un petit aéronef piloté à partir :

- Des données de position et de vitesse fournies par un GPS embarqué à la fréquence de 10 Hz ;
- Des consignes fournies par le pilote au moyen d'une IHM graphique (paramétrage de l'itinéraire de vol).

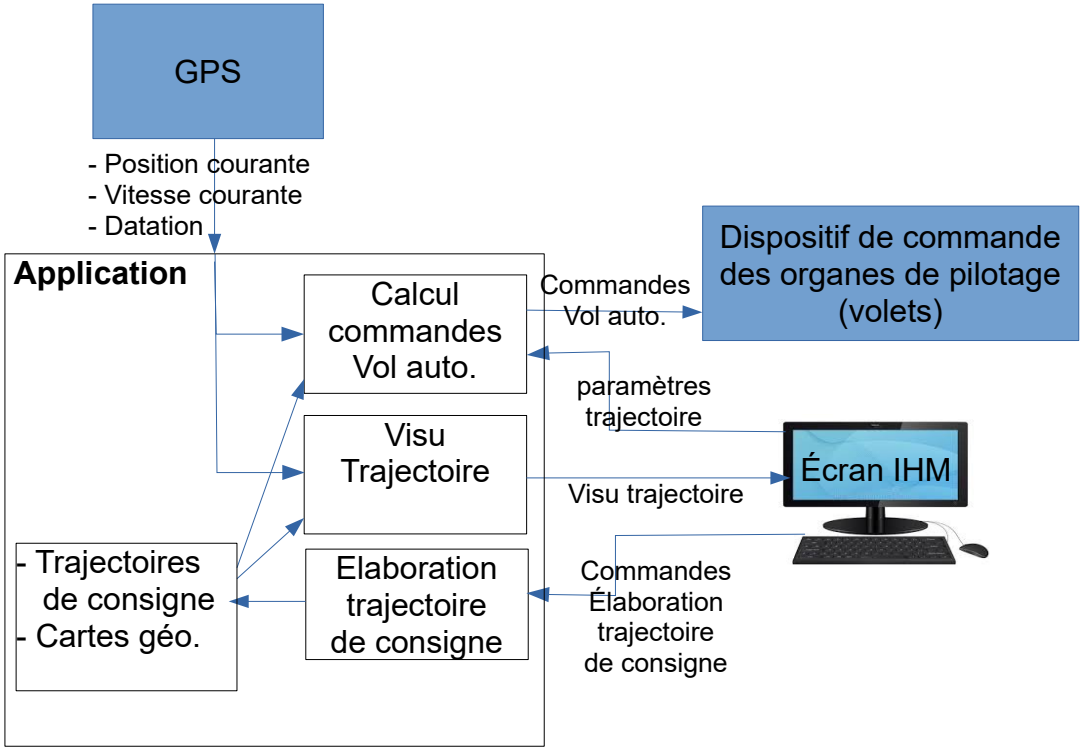
VIII.1.2.RECUEIL DES EXIGENCES ET CONTRAINTES :

EXIGENCE N°1 (pilotage automatique)		
ÉNONCÉ:	L'application doit permettre de GUIDER l'aéronef sur un ITINÉRAIRE DE CONSIGNE, soit manuellement, soit automatiquement.	
Critère	Domaine	Caractérisation
1.1	Précision du guidage	± 50 m par rapport à l'itinéraire prédéterminé
1.2	Itinéraire de référence	Tableau x, y, z, t en coordonnées cartésiennes locales (coordonnées Lambert de la zone)
1.3	Source de positionnement de l'aéronef	Coordonnées géographiques + datation données par le GPS embarqué (précision linéaire = 10 m)
1.4	Pilotage de l'aéronef	Grâce à un boîtier d'interfaçage avec les dispositifs de commande des gouvernes (volets gauche et droit, de direction et de profondeur).
1.5	Choix entre pilotage manuel et automatique	grâce une commande manuelle qui accouple ou désaccouple les commandes automatiques au boîtier d'interfaçage.
1.6	Réaction à un écart entre la trajectoire réelle lissée et l'itinéraire préétabli.	La trajectoire réelle doit être lissée sur un panneau constitué des 10 dernières mesures GPS acquises par la méthode des moindres carrés. Les commandes de direction élaborées par l'activité CTRL et délivrées aux équipements de direction doivent : - Tenir compte de la dernière position délivrée par le GPS ; - Être émise moins de 30 ms après l'acquisition de cette dernière position.

EXIGENCE N°2 (trajectographie)		
ÉNONCÉ :	L'application doit permettre au PILOTE de VISUALISER la trajectoire de l'aéronef ainsi que l'ITINÉRAIRE DE CONSIGNE sur un FOND DE CARTE géographique.	
N° de Critère	Domaine	Caractérisation
1.1	Précision	± 50 m
1.2	Référentiel d'affichage	Projection horizontale Lambert centrée sur la position courante de l'aéronef.
1.3	Limites de la zone affichée.	Zone de forme carrée, de 80 km de côté, centrée sur la position courante de l'aéronef.
1.4	Source de positionnement de l'aéronef	Coordonnées géographiques + datation données par le GPS embarqué (précision linéaire = 10 m)
1.5	Écart entre acquisition d'une position GPS et affichage de cette position	< 200 ms.

EXIGENCE N°3 (gestion itinéraire de consigne))		
ÉNONCÉ :	L'application doit permettre au PILOTE d'ÉLABORER des ITINÉRAIRES DE CONSIGNE et de les STOCKER sur un support de masse.	
N° de Critère	Domaine	Caractérisation
1.1	Précision	± 1 m
1.2	Référentiel d'affichage	Projection horizontale Lambert centrée sur la position courante de l'aéronef.

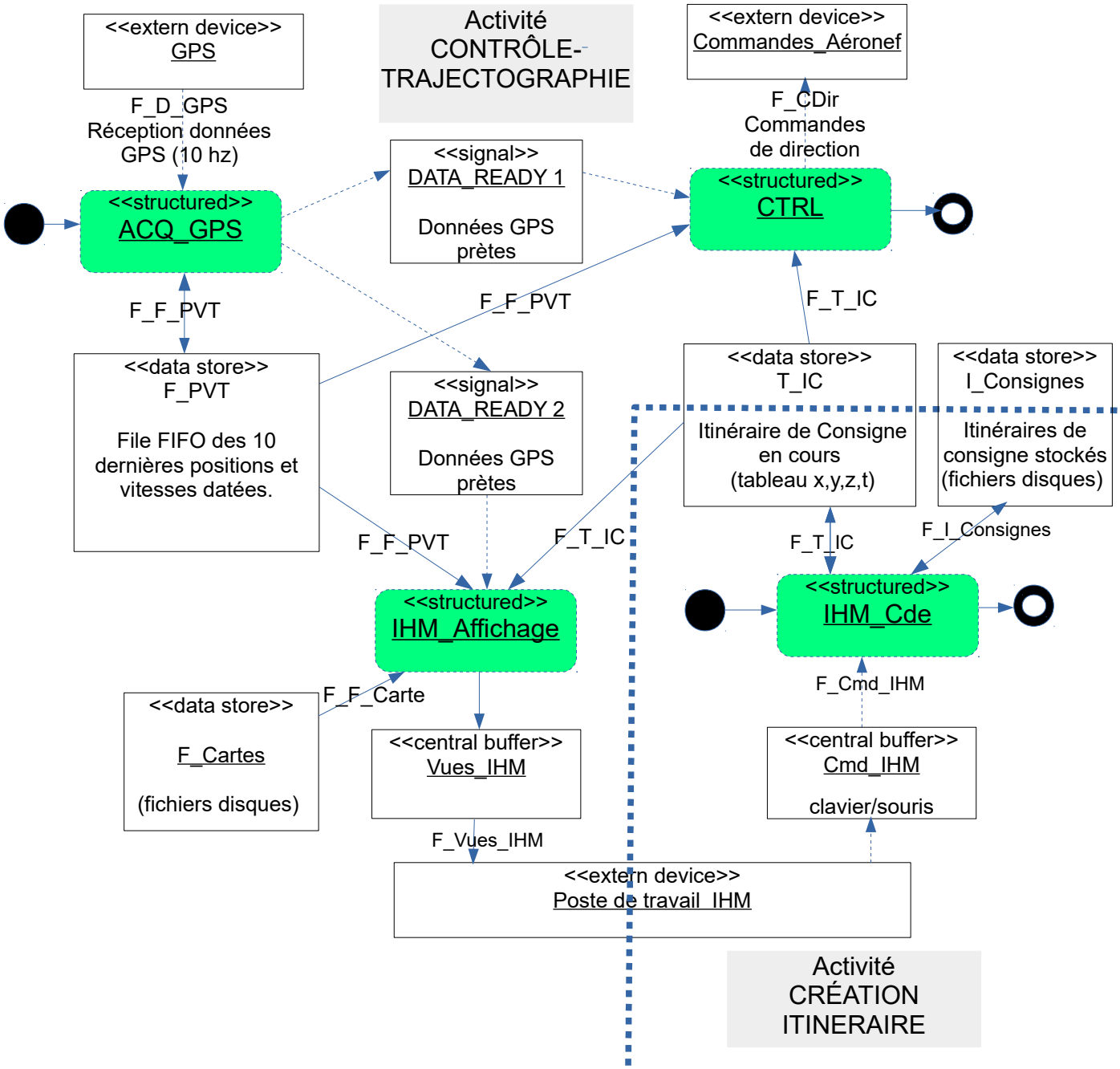
VIII.1.3.**SCHÉMA DE PRINCIPE :**



VIII.2.ANALYSE COMPORTEMENTALE

VIII.2.1.ANALYSE DE L'ACTIVITÉ GLOBALE DE L'APPLICATION :

Le diagramme d'activités suivant modélise le comportement de l'application en régime d'exploitation :



ÉTUDE DES NŒUDS D'ACTIVITÉS STRUCTURÉS					
nœud d'activité	Description	Événement initiateur	Infos consommées	Infos/services produits	Durée propre
ACQ_GPS	Acquisition des données GPS et gestion mise en file d'attente des 10 dernières données	Réception GPS (10 Hz)	D_GPS Données fournies par le GPS	F_PVT (File FIFO des 10 dernières données acquises du GPS)	> 1 ms
CTRL	Contrôle du guidage de l'aéronef sur la trajectoire de Référence.	Signal DATA_READY_1	F_PVT T_IC	Commandes de direction de l'aéronef	> 1 ms
IHM _Affichage	Affichage de l'IHM sur l'écran IHM	Signal DATA_READY_2	T_PVT T_IC F_Carte	Affichage IHM sur l'écran IHM	> 30 ms
IHM_Cde	Acquisition des commandes d'édition des trajectoires de référence.	Interactions exploitants	Messages clavier/souris	Trajectoires de référence	Activité interactive

ÉTUDE DES RESSOURCES PARTAGÉES			
Nom	Description	Format	Activités Utilisatrices

F_PVT	File FIFO des 10 dernières positions géographiques et vitesses datées.	F PVT[300]
		{
		double x, y, z ;
		double vx, vy, vz;
		long int Date ;
		}
		(repère Lambert local)
			Acq_GPS (rw)
			CTRL (r)
			IHM_Affichage (r)

T_IC	Tableau Itinéraire de Consigne	Int I_Max ; // nombre de points ;
		T_IC[I_Max]
		{
		double x, y, z, Date ;
		}
			IHM_Cde (w)
			CTRL(r)
			IHM_Affichage (r)

F_Cartes	Fichiers de cartes géographiques	Format IGN	IHM_Affichage (r)
I_Consignes	Fichiers d'itinéraires de consigne stockés	Format T_IC	IHM_Cde

ÉTUDE DES FLUX D'INFORMATIONS				
Nom	Description	Format	Activités Utilisatrices	Fréquence

F_D_GPS	Données fournies par le GPS à chaque cycle de 100 ms	F_D_GPS
		{
		double Lat, lon, Att_Mer ;
		double V_Est, V_Nord, VH ;
		long int Date ;
		}
			ACQ_GPS (r)	10 Hz

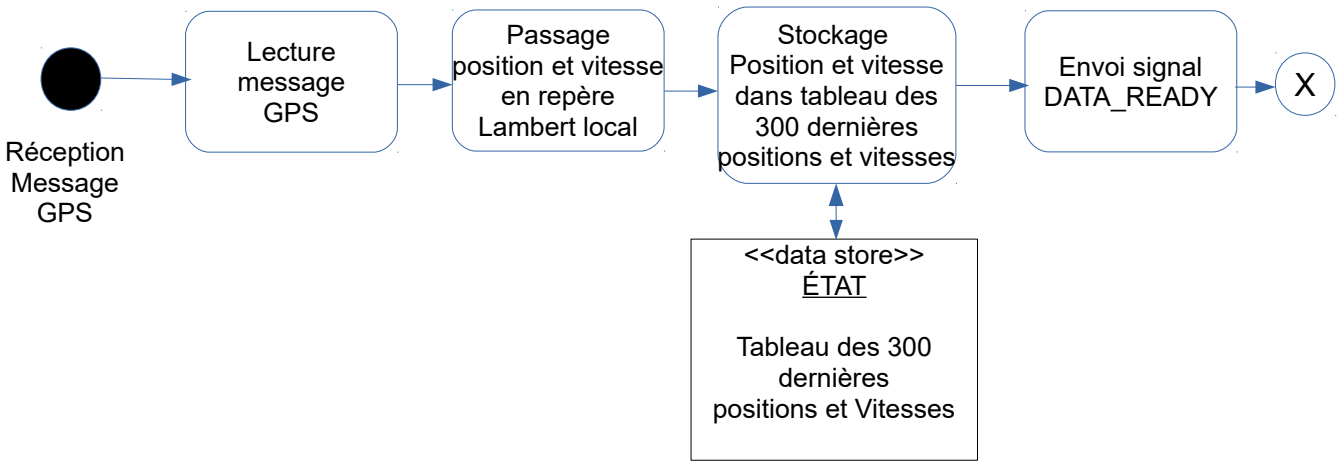
F_F_PVT	Tableau des 10 dernières positions géographiques et vitesses datées fournies par le GPS		Acq_GPS (rw) CTRL (r) IHM_Affichage (r)	10 Hz

F_T_IC	Tableau Itinéraire de Consigne	Int I_Max ; // nombre de points ;
		T_IC[I_Max]
		{
		double X, Y, Z, Date ;
		}
			IHM_Cde (w)
			CTRL(r)
			IHM_Affichage (r)
				10 Hz

F_I_ Consigne	Tableau Itinéraire de consigne à stocker ou choisi	Format T_IC	IHM_Cde (rw)	5 hz
F_F_Carte	Fichier de carte géographique choisi comme fond de carte	Format IGN	IHM_Affichage (r)	Aléatoire
F_Cmd_ IHM	Commandes issues de l'IHM	Requêtes IHM	IHM_Cde	Aléatoire
F_Vues _IHM	Vues à afficher sur l'IHM	Réponses à requêtes IHM	Client IHM	5 hz

VIII.2.2.ÉTUDE DÉTAILLÉE DES ACTIVITÉS :

VIII.2.2.1.ACTIVITÉ ACQ_GPS :

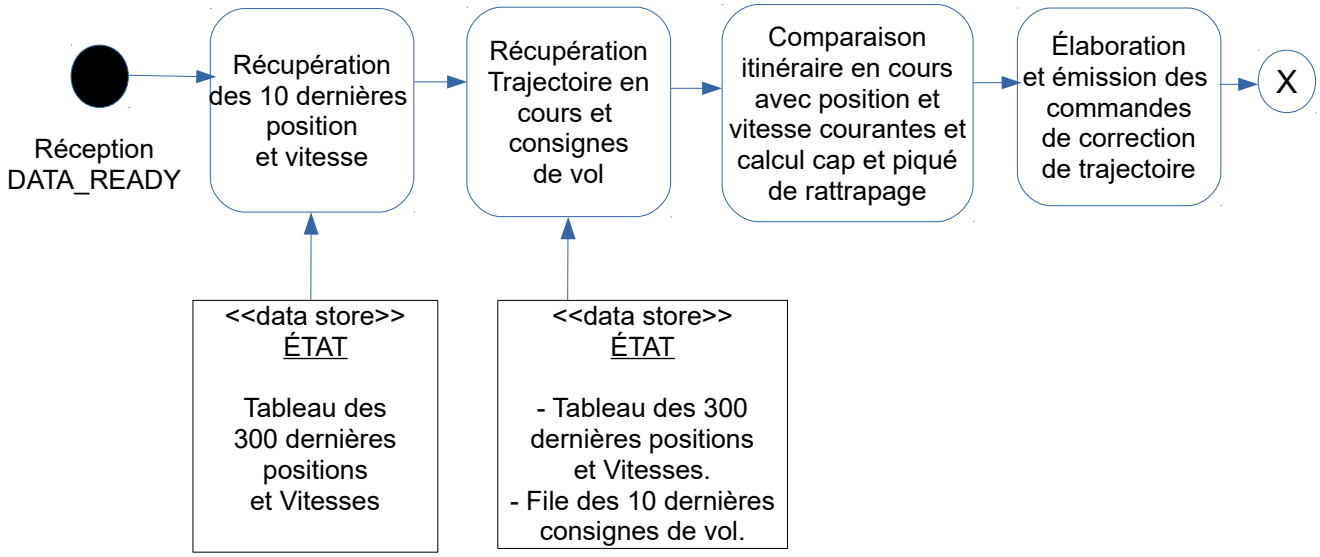


COMMENTAIRES :

- Estimation durée de la lecture du message GPS plus passage en coordonnées locales < 1 ms ;
- Estimation envoi du signal DATA_READY < 1 ms ;
- Estimation stockage des position et vitesse reçue dans la file des 300 dernières positions < 1 ms ;

De ce fait, la durée propre de l'activité ACQ_GPS est sûrement < 3 ms.

VIII.2.2.2.ACTIVITÉ CTRL

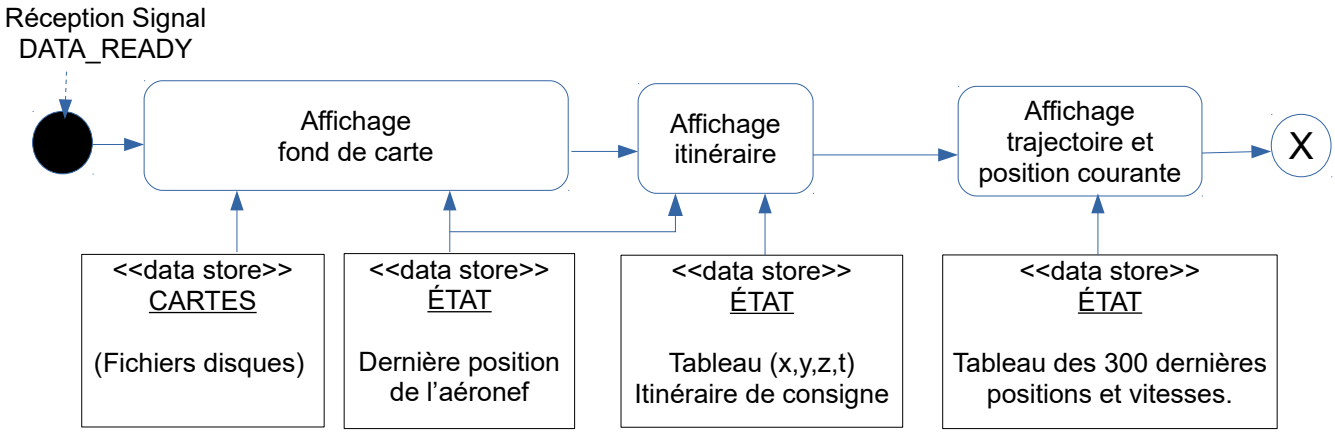


COMMENTAIRES:

- Nous pouvons voir que l'activité CTRL se décompose en 4 actions dont les deux premières ont des durées très courtes (cumul certainement inférieur à la ms).
- Les deux dernières mettent en jeu des algorithmiques beaucoup plus importantes (lissage de la trajectoire réelle et extraction des vitesses, calcul de la trajectoires de raccordement avec l'itinéraire, élaboration des commandes de correction correspondantes, émission vers les organes de commande des volets). Cependant, nous pouvons estimer ces calculs à moins de 10000 instructions de code machine, sans période d'attente.

De ce fait nous pouvons estimer la durée propre d'exécution de CTRL (sur un processeur I5) inférieure à 5 ms.

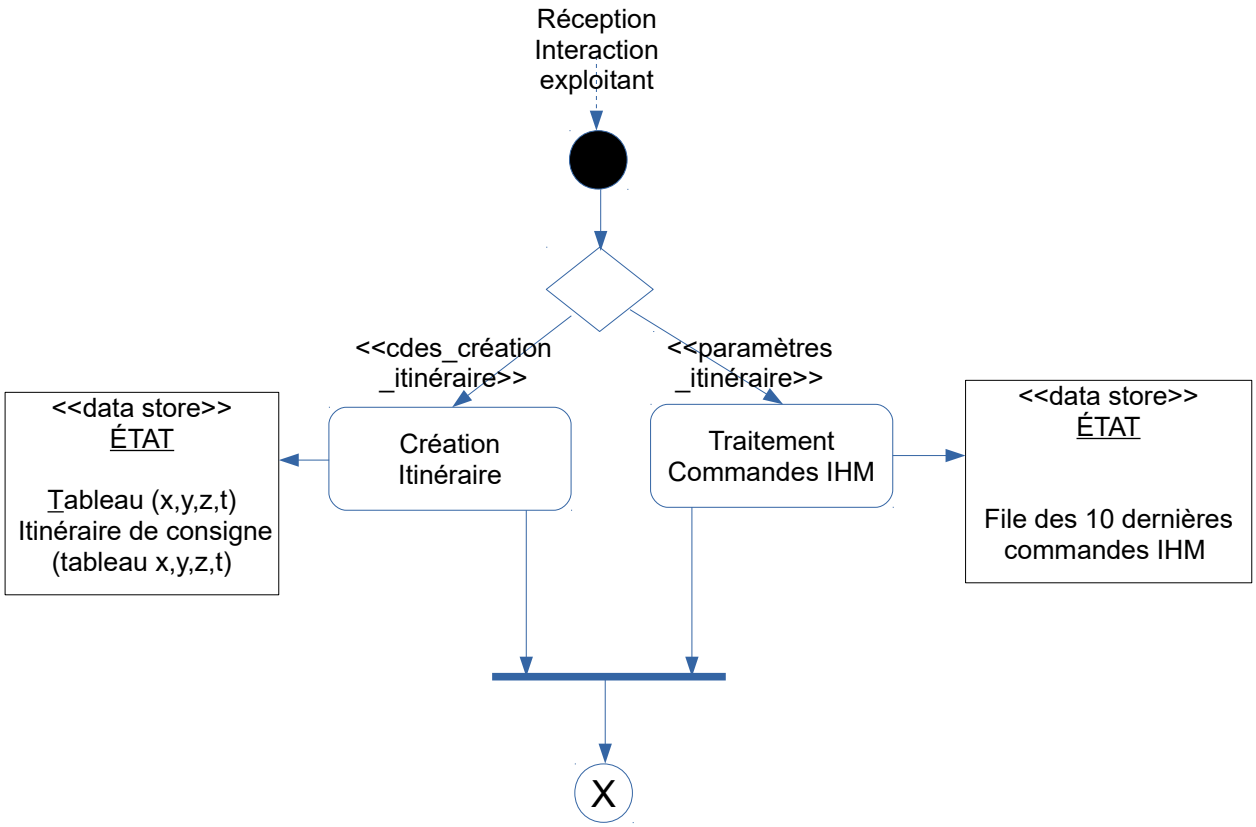
VIII.2.2.3.**ACTIVITÉ IHM _Affichage :**



COMMENTAIRES:

- L'activité IHM_Affichage fait partie de la chaîne de traitement [Acquisition GPS → Affichage de la position géographique courante] qui est soumise à une contrainte temps réel souple (affichage des positions acquises dans un délais inférieur à 200 ms) ;
- L'affichage du fond de carte doit être centré sur la position courante de l'aéronef, ce qui nécessite la récupération de cette donnée dans l'objet ÉTAT ;
- La lecture d'un fichier carte numérisé n'est nécessaire qu'au lancement de l'application et lorsque la zone d'affichage (80*80 km) déborde de la zone couverte par la carte en cours d'exploitation.

VIII.2.3.**ACTIVITÉ IHM _CDE :**



COMMENTAIRE : cette activité n'est soumise qu'à des contraintes temporelles de type "interactif".

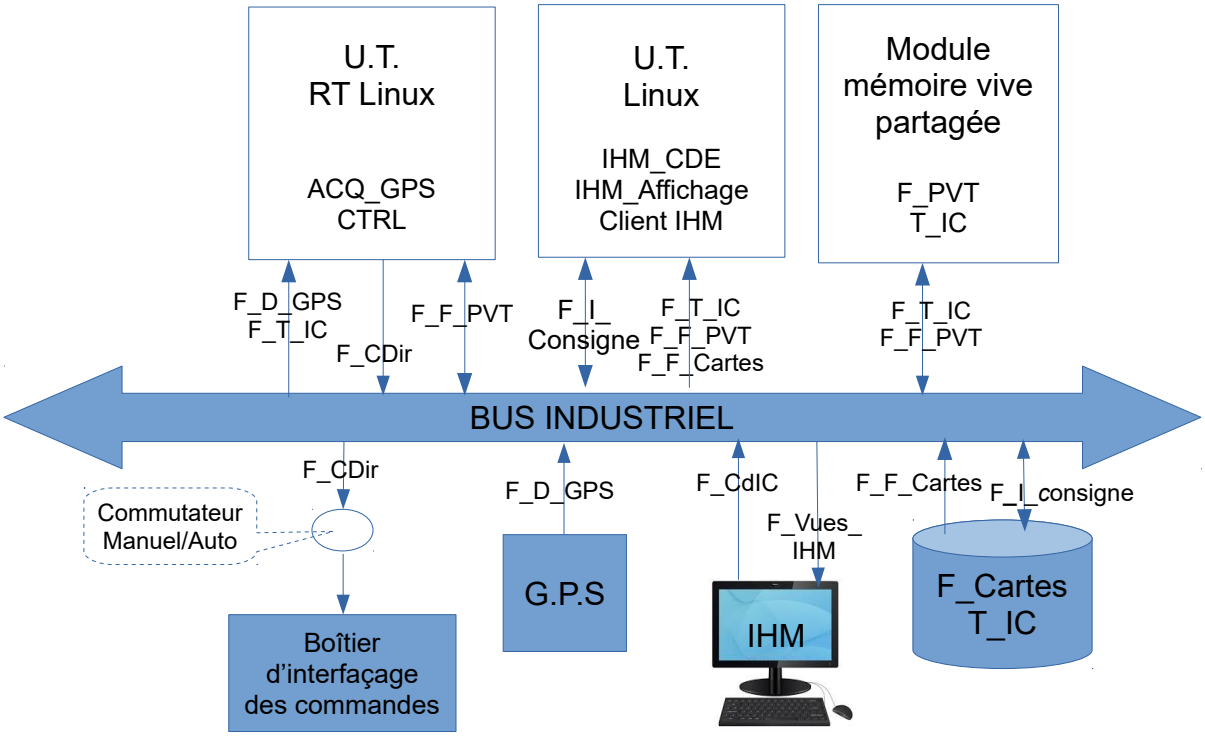
VIII.2.4.**SOLUTION ARCHITECTURALE AU NIVEAU "SYSTÈME":**

Compte-tenu :

- Des contraintes liées à l'emploi (le système doit être transportable, donc compact, poids inférieur à 5 kg) ;
- Des contraintes "temps réel" d'affichage (affichage de la trajectoire dans les 200 ms après acquisition de la dernière position GPS acquise) ;
- Des contraintes temps réel de la boucle Acquisition → émission commandes de pilotage ;

La solution architecturale choisie pour la plate-forme d'accueil est un bus industriel accueillant deux cartes processeurs et une carte mémoire vive partagée:

- Un module unité de traitement sous OS RT Linux accueillant l'activité CONTRÔLE-TRAJECTOGRAPHIE moins l'affichage IHM ;
- Un module unité de traitement sous Linux UBUNTU accueillant l'activité CRÉATION ITINÉRAIRE, l'affichage IHM et le client IHM ;
- Un module mémoire vive avec accès partagé aux deux modules de traitement.

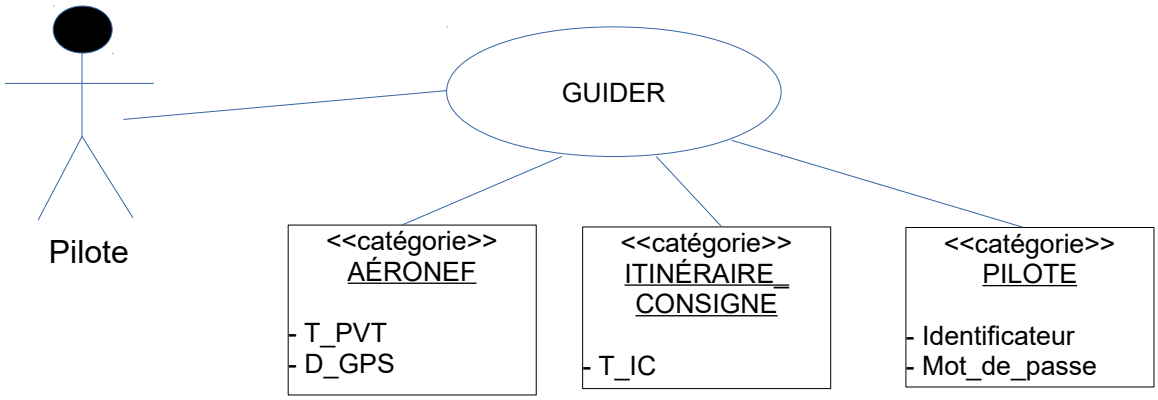


REMARQUE : cette architecture a pour avantage de minimiser la durée des échanges entre activités, car ceux-ci sont soit des échanges de données locaux entre tâches, soit des échanges réseaux dans la boucle locale.

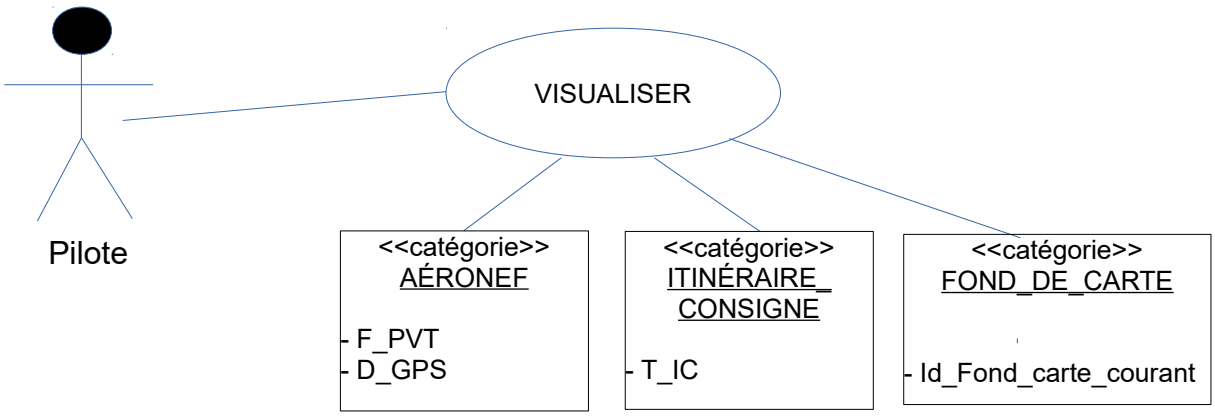
VIII.3.CONCEPTION LOGIQUE :

VIII.3.1.IDENTIFICATION DES CATÉGORIES :

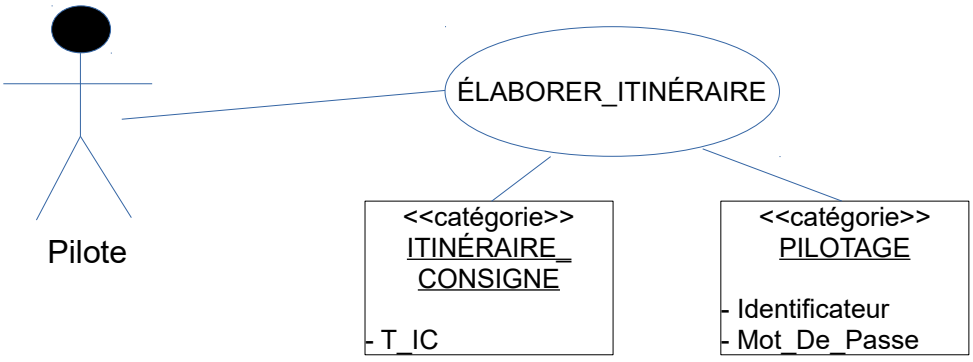
L'application doit permettre au PILOTE de GUIDER l'AÉRONEF sur un ITINÉRAIRE DE CONSIGNE, soit manuellement, soit automatiquement.



L'application doit permettre au PILOTE de VISUALISER la trajectoire de l'AÉRONEF ainsi que de l'ITINÉRAIRE DE CONSIGNE sur un FOND DE CARTE géographique.



L'application doit permettre au PILOTE d'ÉLABORER des ITINÉRAIRES DE CONSIGNE et de les STOCKER sur un support de masse.



VIII.3.2.**IDENTIFICATION DES ATTRIBUTS :**

NOM	FONCTION	TYPE
D_GPS	Données fournies par le GPS à chaque cycle de fonctionnement (100 ms).	- Position et vitesse courantes de l'aéronef (t, x, y, z, vx, vy, vz); - Datation de ces mesures ; - x, y, z : flottants doubles (position cartésienne dans repère lambert local); - vx, vy, vz : flottants doubles (vecteur vitesse en m/s dans le même repère ; - T : flottant double (date posix+dixièmes de secondes) ;
F_PVT	File FIFO rassemblant les 300 dernières positions et vitesses datées en provenance du GPS de l'aéronef ;	Tableau (t, x, y, z, vx, vy, vz) ;
T_IC	Tableau représentant un itinéraire	Tableau (t, x, y, z) ; - x, y, z : flottants doubles (position cartésienne dans repère lambert local); - T : flottant double (date posix+dixièmes de secondes) ;
Identificateur	Identificateur d'accès au logiciel	Chaîne de 8 à 20 caractères quelconques
Mot_de_passe	Mot de passe correspondant à l'identificateur	Chaîne de 8 à 20 caractères dont au moins un chiffre et un caractère spécial.
Id_fond_carte_courant	Identificateur du fond de carte en cours d'utilisation	Nom du fichier correspondant sans extension.

*VIII.3.2.1.1.**IDENTIFICATION DES CLASSES :***

A.RAPPEL :

Pour passer des CATÉGORIES de l'application aux CLASSES, il suffit d'identifier et caractériser les méthodes applicables à chaque catégorie.

B.CLASSE AÉRONEF :

MÉTHODE	FONCTION	PROTOTYPE
Acquérir_données_GPS	Acquérir les données fournies par le GPS	D_GPS : Acquérir_données_GPS ()
Ecrire_PVT	Introduire les dernières données GPS acquises dans la file F_PVT des 10 dernières données acquises	Ecrire_PVT (F_PVT)

Lire_PVT	Extraire la donnée F_PVT	Lire_PVT (F_PVT)

C.CLASSE ITINÉRAIRE_CONSIGNE :

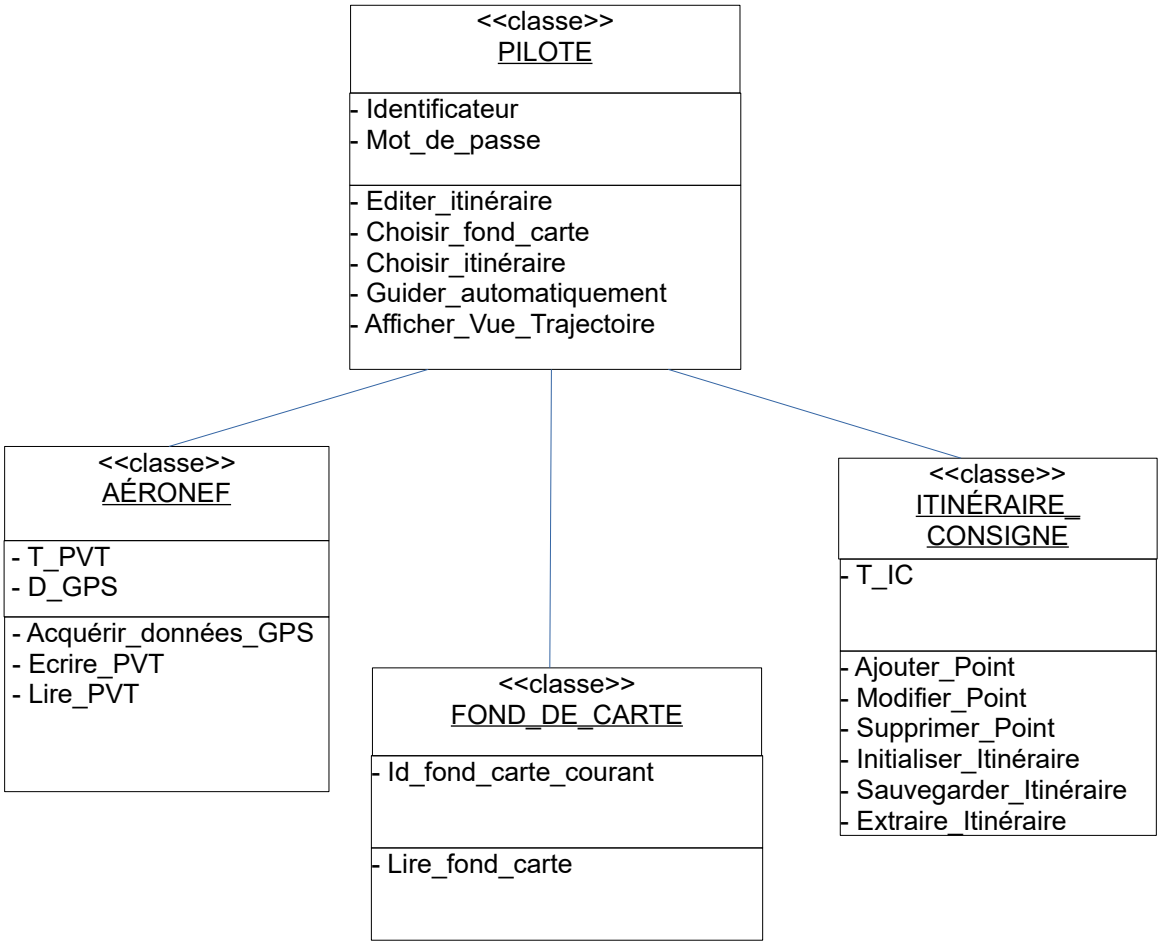
MÉTHODE	FONCTION	PROTOTYPE
Ajouter_point	Ajouter un point à l'itinéraire en construction	Ajouter_point (x, y, z, t)
Modifier_point	Modifier un point de l'itinéraire en construction	Modifier_point (x, y, z, t)
Supprimer_point	supprimer un point de l'itinéraire en construction	Supprimer_point (t)
Initialiser_itinéraire	Effacer l'itinéraire en cours	Initialiser_itinéraire ()
Sauvegarder_itinéraire	Sauvegarder l'itinéraire en construction	Sauvegarder_itinéraire ("nom")
Extraire_itinéraire	Extraire l'itinéraire choisi de l'archive disque	Extraire_itinéraire ("nom")

D.CLASSE FOND_DE_Carte

MÉTHODE	FONCTION	PROTOTYPE
Lire_fond_carte	Extraire un fond de carte de l'archive disque.	F_Carte : Lire_fond_carte

E.CLASSE PILOTE

MÉTHODE	FONCTION	PROTOTYPE
Editer_itinéraire	Lancer l'éditeur d'itinéraire	Editer_itinéraire ()
Choisir_fond_carte	Choisir le fond de carte, la position du centre et la largeur et hauteur (en m) de la représentation	Choisir_fond_carte ("nom", origine, largeur, hauteur)
Choisir_itinéraire	Lire l'itinéraire choisi sur le disque	Choisir_itinéraire ("nom")
Guider_automatiquement	Elaborer les commandes de guidage à partir : - de l'itinéraire choisi ; - De la position courante de l'aéronef.	Cdir : Guider_automatiquement ()


<<classe>>
PILOTE
- Identificateur
- Mot_de_passe
- Editer_itinéraire
- Choisir_fond_carte
- Choisir_itinéraire
- Guider_automatiquement
- Afficher_Vue_Trajectoire
<<classe>>
AÉRONEF
- T_PVT
- D_GPS
- Acquérir_données_GPS
- Ecrire_PVT
- Lire_PVT
<<classe>>
FOND_DE_CARTE
- Id_fond_carte_courant
- Lire_fond_carte
<<classe>>
ITINÉRAIRE_
CONSIGNE
- T_IC
- Ajouter_Point
- Modifier_Point
- Supprimer_Point
- Initialiser_Itinéraire
- Sauvegarder_Itinéraire
- Extraire_Itinéraire

VIII.3.3.ÉLABORATION DE LA SOLUTION :

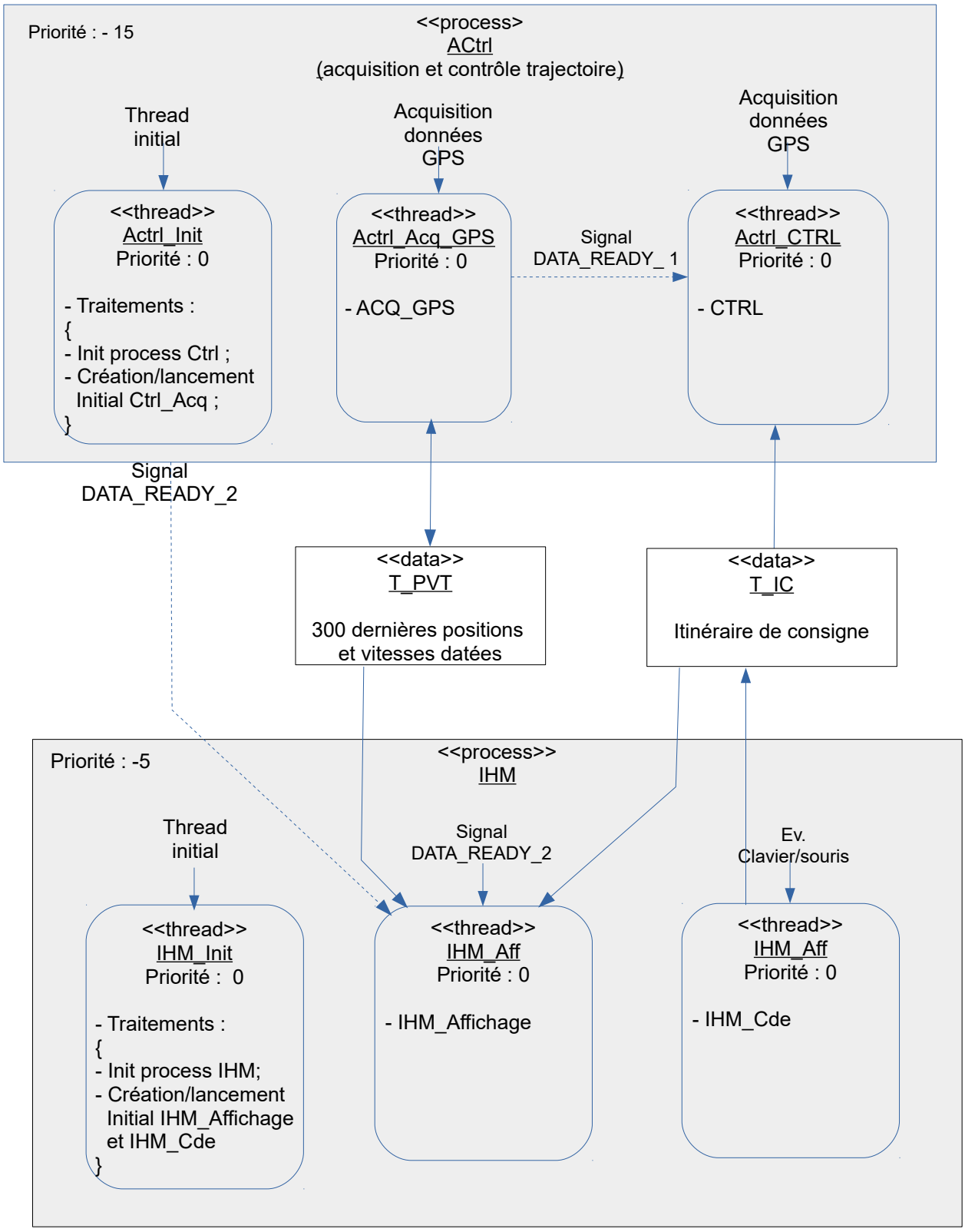
Les contraintes temporelles pesant sur les temps de réponse de l'application affectent deux chaînes de traitement :

Contrainte	Chaîne de traitement concernée	Contrainte temporelle
1	Réception données GPS → ACQ_GPS → CTRL-→ Émission commandes de pilotage	Durée < 30 ms
2	Réception données GPS → ACQ_GPS → IHM → Affichage trajectoire	Durée < 200 ms

Compte tenu des caractéristiques de ces activités, il est décidé :

- D'encapsuler les activités ACQ_GPS et CTRL dans un processus A, géré en mode prioritaire avec une priorité logicielle élevée ;
- D'encapsuler les activité liées à l'IHM dans un processus B géré en temps partagé.

Le résultat de cette démarche d'analyse et de synthèse peut être représenté graphiquement par le diagramme ci-dessous (non normalisé) :


Priorité : - 15
<<process>>
ACtrl
(acquisition et contrôle trajectoire)
Thread initial
Acquisition données GPS
Acquisition données GPS
<<thread>>
Actrl_Init
Priorité : 0
- Traitements :
{
- Init process Ctrl ;
- Création/lancement Initial Ctrl_Acq ;
}
<<thread>>
Actrl_Acq_GPS
Priorité : 0
- ACQ_GPS
Signal DATA_READY_ 1
<<thread>>
Actrl_CTRL
Priorité : 0
- CTRL
Signal DATA_READY_2
<<data>>
T_PVT
300 dernières positions et vitesses datées
<<data>>
T_IC
Itinéraire de consigne
Priorité : -5
<<process>>
IHM
Thread initial
Signal DATA_READY_2
Ev. Clavier/souris
<<thread>>
IHM_Init
Priorité : 0
- Traitements :
{
- Init process IHM;
- Création/lancement Initial IHM_Affichage et IHM_Cde
}
<<thread>>
IHM_Aff
Priorité : 0
- IHM_Affichage
<<thread>>
IHM_Aff
Priorité : 0
- IHM_Cde

COMMENTAIRES :

- L'application a été répartie sur deux processus :
  - Le Processus ACtrl, qui supporte les activités d'acquisition et de contrôle de la trajectoire en temps réel ;
  - Le processus IHM qui supporte les activités de visualisation de la trajectoire et du fond de carte ainsi que la gestion des commandes des utilisateurs.
- La priorité du processus ACtrl a été choisie élevée (la valeur -15 correspond à une priorité Linux élevée) de façon a favoriser les traitements de la boucle de contrôle de la trajectoire (acquisition position courante→ élaboration et émission des commandes de pilotage) ;
- La priorité du processus IHM (-5) a été choisie supérieure à celle des tâches de fond du système, car il existe une contrainte temporelle sur l'affichage (retard < 200 ms) qui ne pourrait être respectée si le processus s'exécutait en priorité "normale" ;
- Le thread ACtrl_Acq_GPS du processus ACtrl contrôle l'exécution de l'exécutable représentant l'activité ACQ_GPS;
- Le thread Actrl_CTRL du processus ACtrl contrôle l'exécution de l'exécutable représentant l'activité CTRL;
- Le thread IHM_Aff du processus IHM contrôle l'exécution de l'exécutable représentant l'activité IHM_Affichage ;
- Le thread IHM_Cde du processus IHM contrôle l'exécution de l'exécutable représentant l'activité IHM_CDE;
- Les threads Actrl_Acq_GPS, Actrl_CTRL et IHM_Aff ont tous 3 besoin d'accéder à la ressource T_PVT. Ils se trouvent donc en concurrence pour l'accès à cette ressource. De ce fait, ces accès doivent être contrôlés par un mécanisme d'exclusion mutuelle (mutex, sémaphore, etc.) ;
- Les threads IHM_Cde et Actrl_CTRL ont besoin d'accéder à la ressource T_IC, mais à priori, ils n'accèdent jamais en même temps à la même carte (on ne modifie pas une carte pendant qu'elle est utilisée). Donc , inutile de protéger l'accès ;
- La communication inter processus est ici assurée par des signaux appartenant aux mécanismes I.P.C (Inter Process Communication).

IX.LA CONCEPTION DÉTAILLÉE:

IX.1.DÉFINITION:

Quelles que soient les démarches suivies, la phase de conception dite PRÉLIMINAIRE ou GLOBALE d'un logiciel aboutit à la définition d'un ensemble de MODULES LOGICIELS qui peuvent être des bibliothèques de FONCTIONS (démarche procédurale) ou des CLASSES (démarche par objets). Ces MODULES sont définis uniquement par:

- La ou les fonctions qu'ils sont sensé offrir au reste de l'application;
- La description de leurs interfaces externes (avec la périphérie physique) et internes (prototypes des fonction ou méthodes publiques), protocoles d'échanges, formats des données échangées;
- La description des contraintes auxquelles ils doivent satisfaire (temps de réponse, durée d'exécution, etc.).

Ils apparaissent donc à ce niveau comme des BOÎTES NOIRES dont on ne connaît rien des détails de leurs mécanismes internes.

Dans son sens le plus strict, la conception détaillée consiste à élaborer l'architecture INTERNE de ces modules d'une manière assez précise pour en permettre le CODAGE INFORMATIQUE dans la phase suivante.

IX.2.DÉMARCHES ET TECHNIQUES EMPLOYÉES:

Lorsque des projets logiciels concernent des réalisations très importantes, tant par le nombre et la diversité des utilisateurs potentiels que par la diversité des problématiques abordées, les travaux de conception globale peuvent aboutir à définir des modules de réalisation extrêmement complexes. La réalisation de ces module peut alors correspondre à un véritable sous-projet incluant des travaux conception logique et comportementale complète. Ces travaux peuvent être menés suivant les mêmes démarches et en utilisant les mêmes outils que ceux que nous avons vus jusqu'à présent.

En revanche, dans le cas de projets de faible ou moyenne complexité, les mécanismes encapsulés par les modules de réalisation issus de la conception générale ne nécessitent que peu d'études architecturales avant de procéder à leur codage: il peut s'agir par exemple:

- De la définition de sous-fonctions permettant de structurer et de rationaliser le codage des fonctions ou méthodes externes;
- De la conception de mécanismes chargés de régler le problèmes de concurrence d'accès aux ressources internes du module;
- etc.

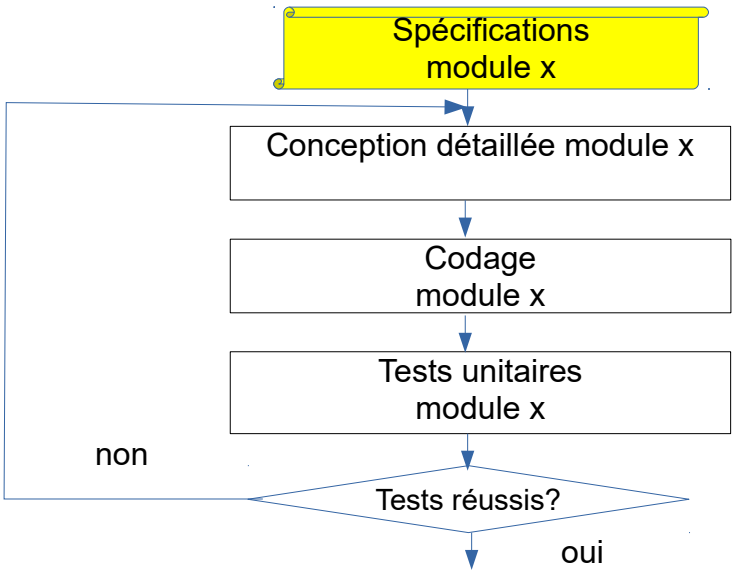
IX.3.CAS DES MÉTHODES AGILES:

En général, les méthodes agiles tendent à supprimer, ou tout au moins à réduire au minimum la phase de CONCEPTION PRÉLIMINAIRE au profit d'une démarche par retouches et adaptation successives de l'architecture globale à l'occasion des différentes itérations. Cette pratique est appelée REFACTORING (réusinage, remaniement, refactorisation).

Les modules de réalisation sont directement issus des travaux initiaux de définition des besoins (collecte des "users stories" ou des "scénariis clients"). Les travaux aboutissent à la définition de MODULES FONCTIONNELS dont la réalisation fait l'objet d'itérations successives du processus de développement. Au fur et à mesure de ces réalisations, l'architecture globale est sensé émerger par des retouches successives destinées à résoudre les problèmes de compatibilité entre les différents modules.

De ce fait, dans une démarche agile, la conception détaillée de chaque module de réalisation suit immédiatement les travaux de spécification de ce module. L'ensemble des trois phases conception détaillée, codage et tests unitaire constitue une boucle d'itération qui permet de converger vers une solution de conception du module. Par analogie avec la démarche TEST-DRIVEN DEVELOPMENT (développement guidé par les tests), qui concerne uniquement l'itération des phases de codage et de tests unitaire, la démarche agile pourrait être appelée CONCEPTION GUIDÉE PAR LES TESTS.

Le schéma suivant donne une représentation visuelle de cette démarche itérative:



X.ANNEXE A - CONSOMMATION DE LA PUISSANCE CPU PAR UNE TÂCHE:

X.1.RAPPELS:

X.1.1.NOTION DE TÂCHE:

Une TÂCHE est souvent définie comme un PROCESSUS LÉGER, c'est à dire un processus s'exécutant sur un seul THREAD. Il s'agit donc d'un traitement logiciel s'exécutant sur un seul flot d'exécution, sans parallélisation.

X.1.2.NOTION DE CPU:

L'acronyme CPU (Central Processing Unit en anglais) désigne l'unité Centrale de Traitement d'une machine de traitement informatique. Celle-ci peut comprendre un ou plusieurs PROCESSEURS (machines multi-processeurs), tandis qu'un processeur peut comprendre plusieurs CŒURS.

Dans la littérature spécialisée française, l'acronyme CPU est souvent traité comme un mot masculin, bien qu'il désigne en fait une "unité". Nous nous conformerons à cet usage.

X.2.INTRODUCTION:

L'activité de conception des applications logicielles requiers de celui qui l'exerce la capacité d'évaluer les besoins des logiciels composant ces applications en termes de consommation des ressources (matérielles et systèmes) de la plate-forme d'accueil.

Pendant son exécution, un logiciel utilise une partie de la puissance du CPU. Cette PUISSANCE CPU est représentative de la QUANTITÉ DE TRAITEMENTS que ce CPU peut exécuter en une unité de temps donnée.

Cette quantité de traitements ne peut être assimilée au nombre d'opérations de code d'ordre que le CPU peut exécuter par unité de temps: en effet, les CPU actuels sont munis de mécanismes permettant de paralléliser l'exécution des séquences de code d'ordre et d'anticiper les accès mémoires (multiples cœurs, pipelines, mémoires caches, etc.) qui permettent de gagner du temps par rapport à la simple exécution en séquence. De ce fait, les tests de capacité sont basés sur des traitements complexes permettant de tenir compte de ces mécanismes (par exemple, la "quantité de traitements par unité de temps" va être représentée par le nombre de "sommets" d'une image 3D que le CPU peut calculer en une unité de temps).

X.3.DÉFINITION DE LA PUISSANCE CPU:

La PUISSANCE d'un CPU peut donc être définie comme la QUANTITÉ MAXIMALE de TRAITEMENTS qu'un CPU est capable d'assurer pendant l'unité de temps choisie. Dans la pratique, cette grandeur dépend:

- Du nombre maximum d'instructions exécutables par les CŒURS du processeurs physiques par unité de temps. Cette donnée dépend en grande partie de la fréquence de base du cœur considéré (un cœur de fréquence 3,5 ghz est normalement plus "rapide" qu'un cœur de fréquence 2,5 ghz), mais d'autres facteurs interviennent, comme la rapidité d'accès à la mémoire vive, la taille et le nombre des caches, etc ;
- Du nombre de cœurs disponibles dans le processeur et des traitements de parallélisation intégrés à ce processeur;
- De l'efficacité du système de "pipeline" installé sur chaque cœur ;
- De la longueur des "mots" traités par le code d'ordre (à égalité de fréquence d'horloge, un processeur à mots de 64 bits est capable d'assurer plus de traitements qu'un processeur 32 bits);
- D'autres facteurs peuvent également intervenir comme le style de programmation (qui peut favoriser la parallélisation au niveau des cœurs) ou la présences d'instructions spécialisées dans le code d'ordre (qui peuvent accélérer certains traitements).

Certains outils de mesure existants (comme le logiciel gratuit CPU-Z, par exemple) essaient de quantifier la puissance d'un CPU en l'assimilant au nombre de traitements d'un certain type que le CPU peut exécuter en un certain temps. Les valeurs ainsi trouvées dépendent donc du choix de ces traitements et n'ont pas de signification en valeur absolue. En revanche, ces outils permettent de se faire une idée du rapport existant entre les puissances de deux CPU de marques ou de types différents.

X.4.QUANTITÉ DE TRAITEMENTS EXÉCUTABLE PAR UN CPU:

Si la PUISSANCE d'un CPU a été évaluée à P (Par les procédés décrits précédemment), nous pouvons définir la QUANTITÉ DE TRAITEMENTS que ce CPU peut fournir durant un intervalle de temps T par la formule:

$$Q = P * T$$

X.5.CONSOMMATION DE TRAITEMENTS CPU PAR UNE TÂCHE:

Dans une machine donnée, une partie de la puissance CPU totale est consommée par le système d'exploitation. Le reste (que nous appellerons la puissance disponible) est consommé par les différentes applications lancées par les utilisateurs.

Si une tâche s'exécutant en priorité maximale sur un CPU dont la puissance disponible est P a une durée d'exécution de T (hors périodes d'attente) et si le CPU disponible est entièrement alloué à cette tâche, sa consommation totale de traitements CPU est:

$$CT = P * T$$

Il faut cependant bien comprendre que la puissance CPU consommé par une tâche à un instant donné dépend de la PRIORITÉ que le moniteur de tâches lui accorde:

- Si le système d'exploitation est capable de gérer des priorités préemptives entre tâches, la tâche la plus prioritaire se verra allouer LA TOTALITÉ de la puissance CPU disponible pour les applications pendant son exécution. La durée d'exécution de cette tâche sera donc optimale, mais les autres taches seront mises en attente durant son exécution;
- Si, au contraire, le système d'exploitation ne traite les tâches qu'en temps partagé (TIME SHEARING), la puissance CPU sera partagée également entre les différentes tâches actives, par tranches d'une durée donnée (en général, 100 ms). Par rapport à une exécution en mode "prioritaire", la durée d'exécution d'une tâche peut donc être énormément rallongée, surtout si elle est en concurrence avec beaucoup d'autres tâches.

De ce fait, une tâche qui a besoin pour s'exécuter sur un CPU de puissance P de consommer une quantité de traitements CPU égale à C durera:

$$T1 = \frac{C}{P}$$

si elle est exécutée en priorité maximale. En revanche, si le système d'exploitation ne lui alloue qu'un pourcentage p de la puissance disponible, sa durée d'exécution sera au minimum de:

$$T2 = \frac{100 * C}{p * P}$$

REMARQUE: les systèmes d'exploitation les plus utilisés (Linux, Windows, Mac OS) sont capables de gérer des priorités préemptives. Ils traitent en priorité les tâches possédant ce type de priorité tout en continuant de gérer les autres tâches en temps partagé quand aucune tâche prioritaire ne s'exécute (exécution "en tâche de fond").

X.6.OUTILS DE MESURE DIRECTE DE LA CONSOMMATION D'UNE TÂCHE:

La commande > top (sous linux) ou le gestionnaire de tâches sous windows nous renseignent sur le POURCENTAGE de la puissance CPU totale consommée par chacune des tâches en cours d'exécution. Il s'agit en général de la moyenne des pourcentages, calculée sur une période donnée.

X.7.DURÉE PROPRE D'EXÉCUTION D'UNE TÂCHE:

X.7.1.DÉFINITION:

Dans le cadre de cet ouvrage, nous appellerons Durée Propre d'Exécution (DPE) d'une tâche la durée qui s'écoule entre le début et la fin d'exécution de cette tâche à condition que cette tâche soit EXÉCUTÉE EN PRIORITÉ PRÉEMPTIVE MAXIMALE par le système d'exploitation.

Ceci revient à dire que pendant son exécution, cette tâche requiers la totalité de la puissance CPU disponible pour les applications. En effet, quel que soit le mode d'exécution des tâches applicatives, une partie de la puissance CPU est consommée par le système d'exploitation (pour l'ordonnancement des tâches, la gestion des événements externes, etc.).

En fait, même dans les conditions citées ci-dessus, la durée propre d'exécution d'une tâche n'est pas constante car elle dépend de plusieurs facteurs:

- La disponibilité des ressources qu'elle utilise;
- Les valeurs des "paramètres d'exécution" utilisés par cette tâche.

X.7.2.INFLUENCE DE LA DISPONIBILITÉ DES RESSOURCES:

L'exécution d'une tâche peut être suspendue pour une durée variable par l'attente d'un événement (Ex: événement de synchronisation) ou de la disponibilité d'une ressource (Ex: attente de la libération d'un périphérique ou de la réception d'une donnée). La durée de ces attentes est par nature peu prévisible.

REMARQUE: pendant ces périodes d'attente, la tâche ne consomme pas de puissance CPU, puisqu'elle ne s'exécute pas. En revanche, ces périodes augmentent la durée d'exécution effective.

X.7.3.INFLUENCES DES PARAMÈTRES D'EXÉCUTION:

Une tâche utilise des "valeurs de paramètres d'entrée" qu'elle trouve soit dans son contexte d'exécution, soit en lisant des données en provenant de la périphérie (données d'entrée-sortie). Ces valeurs agissent sur les structures alternatives et répétitives du code en modifiant les parcours d'exécution ou le nombre d'itérations, entraînant des variations de la durée d'exécution qui peuvent être très importantes (mais qui sont dans ce cas, prévisibles et reproductibles).

X.7.4.**CONCLUSION:**

Même si les conditions spécifiées dans sa définition sont remplies, la durée propre d'exécution d'une tâche n'est pas constante puisqu'elle dépend du contexte d'exécution. Les seules durées qu'il est possible d'évaluer sont:

- La Durée Propre d'Exécution Maximale DPE_Max, qui représente la durée d'exécution quand les temps d'attente sont tous maximums et quand, simultanément, les "paramètres d'entrée" de la tâche entraînent le chemin d'exécution le plus long possible;
- La Durée Propre d'Exécution Minimale DPE_Min, qui représente la durée d'exécution quand les temps d'attente sont tous minimums et quand, simultanément, les "paramètres d'entrée" de la tâche entraînent le chemin d'exécution le plus court possible.

La durée d'exécution d'une tâche est d'autre part fortement impactée par le type de "scheduling" pratiqué par le système d'exploitation:

- Si cette tâche est exécutée en mode prioritaire et avec priorité maximale, sa durée d'exécution sera très proche de la durée nominale;
- Si cette tâche est exécutée en temps partagé, la durée d'exécution peut être beaucoup plus longue que la durée propre.

Le concepteur devra donc tenir compte dans sa réflexion du type de scheduling à choisir et de la répartition des activités sur les ordinateurs en fonction de leur caractère prioritaire.

X.8.ESTIMATION DE LA DURÉE PROPRE D'EXÉCUTION D'UN LOGICIEL :

X.8.1.INTRODUCTION:

La mesure de la durée d'exécution d'un traitement informatique existant est un problème relativement simple : Il est possible de mesurer directement cette durée, en utilisant, par exemple, un "lanceur" qui itérera un certain nombre d'exécutions en mode prioritaire et calculera la durée moyenne d'une itération pour chaque "cas d'exécution" possible. Il est également possible d'utiliser les outils supportés par les systèmes d'exploitation.

En revanche, en phase de conception, nous sommes souvent amenés à estimer la durée de logiciels non encore réalisés afin de dimensionner les plates-formes qui l'accueilleront. Ce logiciel n'existant pas encore sous une forme exécutable, il est impossible de procéder comme indiqué précédemment. Il nous faut donc trouver d'autres pistes d'estimation, sachant qu'au niveau de la conception, nous n'avons besoin que d'un ordre de grandeur et non d'une estimation précise.

X.8.2.MÉTHODE GÉNÉRALE:

X.8.2.1.PREMIÈRE ÉTAPE : ANALYSE ALGORITHMIQUE :

Dans un premier temps, on effectue une analyse algorithmique rapide des traitements à réaliser, en recourant si le besoin s'en fait sentir à des raffinages successifs. La démarche s'arrête lorsque les blocs de traitement identifiés peuvent être assimilés à des traitements ou à des segments de codes objets dont la durée propre d'exécution peut être approximativement estimée par les concepteurs.

REMARQUES :

- Pour effectuer cette analyse, il est possible d'utiliser différents outils : une description algorithmique par pseudo-code, un diagramme d'activités U.M.L, etc ;
- Quel que soit l'outil choisi, il est important de faire apparaître les périodes d'attente (temporisation, attente d'événements, etc.) car elle permettront d'estimer la variabilité de la durée d'exécution ;
- Il est également important de faire ressortir les branches de traitement parallélisables. De ce point de vue, les diagrammes d'activités U.ML sont plus indiquées que les pseudo-codes.

X.8.2.2.DEUXIÈME ÉTAPE :ESTIMATION DE LA DURÉE PROPRE DES BLOCS DE TRAITEMENT IDENTIFIES:

Si l'analyse algorithmique a été menée jusqu'au bout, il est alors possible d'estimer approximativement la durée propre d'exécution de chacun blocs de traitement composant le pseudo-code, soit en les assimilant à des blocs de traitement dont la durée est connue,

soit en calculant approximation le nombre d'instructions de base que leur eécution met en jeu (en tenant compte des alternatives et des itérations).
Le tableau suivant donne la durée propre moyenne (très approximative) de quelques traitements ou segments de code pour un CPU de moyenne gamme actuelle (4 cœurs, fréquence de 3 ghz) :

FONCTION OU SEGMENT DE CODE	DURÉE PROPRE (± 20%) ms
Une instruction élémentaire (addition, multiplication, etc.) exécutée entièrement sans rupture du pipeline.	0,3 ns
Une ligne de pseudo-code n'impliquant aucun appel à un sous-algorithme (correspond en moyenne à 10 instructions de base).	Langage compilé : 3 ns Langage interprété: suivant langage : - PHP, javascript: 30 ns - JAVA : 6 ns etc.
Fonction trigonométrique (sin, cos, tan, asin, acos, atant etc.	70 ns
Fonction pow()	150 ns
Polynôme de degrès 5 (avec fonction pow)	280 ns
Fonction log()	130 ns
Fonction time()	12 ns
Requête de type SELECT dans une table d'une base de données locale	Suivant la taille T de la table et le rang R des occurrences dans cette table. Formule très approximative : max durée (ms) = 0,5 + T/1500
Etc...	

X.8.2.3.*ESTIMATION DE LA DURÉE D'EXÉCUTION TOTALE:*

En ce qui concerne la durée d'exécution totale, il faut bien sûr évaluer cette durée pour tous les chemins d'exécution possibles en fonction des réponses des structures alternatives et itératives aux données du contexte.

Comme chaque chemin d'exécution aura une durée propre, Il faudra s'efforcer de déduire de ces données une durée maximale et une durée moyenne approximative du logiciel analysé.

X.8.3.**EXEMPLE :**

Supposons que nous ayons à développer une routine permettant d'identifier un visiteur dans une liste de visiteurs enregistrée dans la table suivante d'une base de donnée mysql :

Visiteur (id, mdp) (avec id = identificateur et mdp = mot de passe)

Nous pouvons décrire cette routine par le pseudo-code suivant :

```
ROUTINE booleen IdVisiteur ( chaine id, chaine mdp)
        booleen Identifie = false;
        QUAND ( réception de la validation du menu de connexion ) FAIRE
                Rechercher dans la BDD un utilisateur correspondant à l'Id. et au mdp saisis;
                SI ( Un utilisateur est trouvé ) ALORS     Identifie = true;
                SINON                                      Identifie = false;
                FINSI
        FINFAIRE
FIN
```

Nous supposerons d'autre part que la liste des utilisateur contient au plus 1500 occurrences.

Dans l'exemple choisi, un développeur expérimenté identifiera le sous-bloc "Rechercher dans la BDD un utilisateur correspondant à l'id. et au mdp saisis", dont la durée correspondra sensiblement à une requête de type SELECT dans la base de données.
La durée de cette recherche dépendra de divers facteurs:

- Longueur de la table des utilisateurs identifiés;
- Rang de l'enregistrement de l'utilisateur dans la table;
- Localisation du S.G.B.D (sera-t-il dans la même machine que le bloc étudié?);
- Langage de programmation utilisé;
- Type de SGBD utilisé;
- Puissance des CPU exécutant le bloc et le SGBD.

Il est facile de vérifier que pour une longueur de table de 1500 articles, avec un SGBD MySQL et une exécution sur un CPU i5, cette durée varie approximativement entre 0,8 et 1,1 ms suivant le rang de l'enregistrement recherché dans la table.

Supposons d'autre part que le langage de programmation choisi soit le langage C. Le pseudo-code comprend 6 lignes dont 5 n'impliquent pas d'appel a des sous-algorithmes. Nous pouvons estimer la durée d'exécution de ces 5 lignes à 15 ns.

La durée maximale du logiciel ne devrait donc pas dépasser 1,1 ms. Quant à la durée moyenne, elle devrait avoisiner 0,95 ms.

REMARQUE :

Il est également possible d'écrire un code source correspondant à l'algorithme obtenu en replaçant les blocs de traitement obtenus par raffinement par l'appel de fonctions simulant

la durée d'exécution de ces blocs, puis de mesurer la durée d'exécution de ce code source pour chacune des configurations du contexte correspondant aux divers chemins d'exécution possibles.

X.8.4.INFLUENCE DE LA PLATE-FORME D'EXÉCUTION:

Il ne faut pas oublier que les durées obtenues ne sont valables que pour le CPU qui a été choisi pour faire cette évaluation. Pour un autre CPU, le temps d'exécution sera modifié en fonction du rapport de puissance entre le CPU utilisé pour l'évaluation et le CPU destinataire.

REMARQUE : *certains outils logiciels libres permettent d'estimer le rapport de puissance entre deux CPU. C'est le cas, par exemple, de CPU-Z.*

X.8.5.INFLUENCE DU LANGAGE DE PROGRAMMATION:

D'autre part, la durée d'exécution varie considérablement suivant que les traitements sont codés en langage interprétable (PHP, Javascript, etc) ou en langage compilable (C, C++, etc.). Le rapport de rapidité est d'environ un à 10 en faveur de ces derniers. En ce qui concerne Java, la rapidité d'exécution peut être assimilée à celle d'un code compilé. De ce fait, lorsqu'on assimile un traitement A à un autre traitement B dont on connaît la durée, il faut tenir compte du langage de programmation qui sera utilisé.

XI. ANNEXE B -CONSOMMATION DE LA MÉMOIRE VIVE PAR UNE TÂCHE:

Pendant son exécution, un logiciel consomme principalement 2 sortes de ressources:

- Une partie du volume de mémoire vive mis à disposition des applications;
- La "puissance CPU" ou puissance de l'Unité Centrale de Traitement (Central Processing Unit en anglais). Cette puissance est représentative du "volume de traitements" que cette U.C.T peut exécuter en une unité de temps donnée.

Une tâche quelconque a besoin d'espaces dans la mémoire vive des unités centrales de traitements sur lesquelles elle se déroule pour deux raisons:

- Pour y charger son code exécutable durant son exécution;
- Pour y charger temporairement les données qu'elle manipule (et en particulier les contenus des fichiers qu'elle est appelée à utiliser).

Si, pour une raison ou pour une autre, un ordinateur ne peut lui allouer assez de mémoire vive, des mécanismes de partage de la mémoire vive restante entre les tâches en cours peuvent s'activer, mais ils ont pour effet d'AUGMENTER LA DURÉE D'EXÉCUTION. En effet, ce type de fonctionnement, appelé OVERLAY (recouvrement), occasionne des transferts de données entre la mémoire vive et une mémoire de masse qui consomment de la puissance CPU. Or, une tâche donnée est amenée à tout instant à partager la mémoire vive avec les autres tâches se déroulant au même instant dans cette machine.

De ce fait, il est important de s'assurer que le volume de mémoire vive des ordinateurs est suffisant pour "étaler" les pics d'occupation de la mémoire.

REMARQUE: La commande > top (sous linux) ou le gestionnaire de tâches sous windows permettent d'obtenir des renseignements sur la consommation de mémoire des différents processus logiciels.

XII.ANNEXE C -FIABILITÉ DES COMPOSANTS ÉLECTRONIQUES DIGITAUX:

XII.1.INTRODUCTION:

Ce sous-chapitre expose sommairement les principales notions liées à la fiabilité des composants électroniques digitaux. Ces composants ont pour caractéristique principale de ne pas présenter de signes d'usure précédant une panne (contrairement à des composants électroniques analogiques ou des composants mécaniques, par exemple).

En effet, un composant digital traite des signaux électriques qui ne prennent que deux valeurs (0/5 volts, -5/5 volts, etc.). Le composant continue de fonctionner parfaitement tant que ces valeurs ne s'écartent pas trop des valeurs nominales. Lorsque l'écart devient trop important, le composant (ou l'ensemble de composants auquel il appartient) cesse tout simplement de fonctionner. Le remplacement du composant défaillant rétablira pleinement le fonctionnement et la fiabilité: les composants électroniques sont donc ENTIÈREMENT RÉPARABLES. De ce fait, entre le début d'utilisation et la survenue d'une panne, le fonctionnement global ne se dégrade pratiquement pas.

XII.2.MESURE DE LA FIABILITÉ:

XII.2.1.NOTION DE M.T.B.F:

La fiabilité d'un composant électronique est souvent exprimée par son M.T.B.F (Mean Time Between Failures). Le M.T.B.F, exprimé en HEURES, est représentatif de la moyenne arithmétique des durées qui s'écoulent entre deux pannes de ce composant s'il fonctionne en continu et si le composant est entièrement remis en état après chaque panne. L'ordre de grandeur du M.T.B.F d'un composant complexe "grand public" (comme un ordinateur, par exemple) est de l'ordre de 25000 heures, soit environ 3 ans, mais certains composants de qualité professionnelle (routeurs, disques durs, unités centrales) sont donnés pour dépasser les 100 000 heures. Dans ce document, nous noterons le M.T.B.F par la lettre grecque μ.

XII.2.2.NOTIONS DE M.T.T.F, M.T.T.R ET TAUX DE DISPONIBILITÉ :

Le M.T.T.F (Mean Time To Failure) ou "temps moyen avant panne" est également utilisé. La différence avec le M.T.B.F. est qu'il est représentatif de la durée s'écoulant entre sa première mise en fonction (éventuellement après dépannage) et la première panne. Le M.T.T.F se note habituellement par la lettre grecque θ.

La différence entre M.T.B.F et M.T.T.F est le M.T.T.R (Mean Time To Repair), c'est à dire le délai moyen de remise en service après la survenue d'une panne :

$$M.T.B.F = M.T.T.F + M.T.T.R$$

En pratique, pour les composants électroniques, M.T.B.F et M.T.T.F ont des valeurs très voisines car la durée de dépannage est très courte (elle se réduit souvent au remplacement de l'élément défectueux).

Le TAUX DE DISPONIBILITÉ est, pour une durée D donnée, le pourcentage de cette durée pendant lequel le système est disponible. On peut donc écrire:

$$\text{Taux de disponibilité} = \frac{M.T.B.F}{M.T.B.F+M.T.T.R} = \frac{M.T.B.F}{M.T.T.F}$$

REMARQUE: Le taux de disponibilité est souvent exprimé sur une échelle de 7 classes:

Classe	Indisponibilité sur un mois	Indisponibilité sur 1 an	Pourcentage de disponibilité
1	72 h	36,5 jours	90
2	7,2 h	3,6 jours	99
3	43,2 mn	8h 45 mn	99,9
4	4,32 mn	51,6 mn	99,99
5	25,9 s	5,2 mn	99,999
6	2,6 s	32 s	99,9999
7	0,3 s	3,2 s	99,99999

Mnémonique : *la classe n correspond à un pourcentage de disponibilité exprimé avec n chiffres 9 (classe 3 = 99,9%).*

Nous pouvons voir que la disponibilité d'un composant dépend à la fois de sa fiabilité "structurelle" et de la rapidité de sa procédure de dépannage, qui dépend en partie de la conception d'ensemble (accessibilité du composant, possibilité de le remplacer "à chaud", etc.).

XII.2.3.**NOTION DE TAUX DE PANNES:**

Le TAUX DE PANNES d'un composant est le nombre de pannes prévisibles par unité de temps (on l'appelle également DENSITÉ DE PANNES). Il se note par la lettre grecque λ (lambda).. Pour un composant électronique, λ est considéré comme constant.

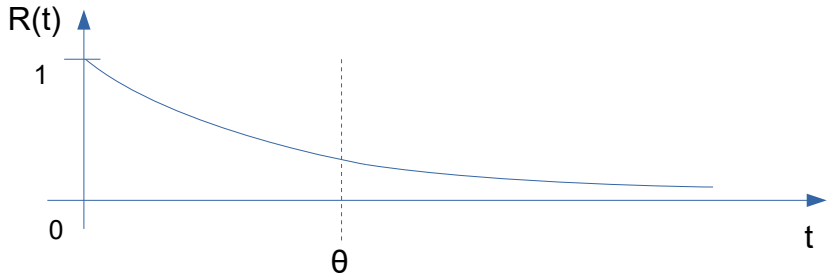
XII.2.4.LOI DE FIABILITÉ:

On note R(t) la probabilité pour un composant d'être encore en service à l'instant t après sa mise en fonction en continu. C'est donc la probabilité d'avoir une durée de vie supérieure à t. Pour un composant électronique, on considère que la loi de fiabilité est une fonction exponentielle:

$$R(t) = e^{-\lambda t}$$

(λ étant supposé constant)

Comme nous pouvons le voir, la fiabilité est égale à 1 (100%) au début du fonctionnement (t=0) et décroît en fonction du temps jusqu'à tendre vers 0 quand t tend vers l'infini:



Statistiquement, si λ est constant, le nombre de pannes survenant pendant l'intervalle θ (M.T.T.F) doit être égal à 1, puisque μ représente "le temps moyen avant panne". De ce fait, on peut écrire: $1 = \lambda.\theta$ et: $\theta = 1/\lambda$

La probabilité pour qu'un composant ne connaisse aucune défaillance avant le M.T.B.F est donc :

$$R(t) = e^{-\lambda t} = e^{-\theta/\theta} = 1/e \simeq 0{,}37\ (37\ \%)$$

avec $e \simeq 2{,}718$

REMARQUE: Les valeurs de M.T.B.F et M.T.T.F sont valables pour une exploitation en continu dans des conditions normales d'exploitation (température, humidité, conformité et régularité de l'alimentation électrique, perturbation électromagnétique et électrostatique acceptables, etc.). Si ces conditions ne sont pas respectées, le M.T.B.T et le M.T.T.F peuvent beaucoup diminuer.

XII.2.5.FONCTION PROBABILITÉ DE PANNE:

Nous avons vu plus haut que la fonction R(t) exprimait la probabilité pour le composant concerné d'avoir une durée de vie supérieure à t. La fonction F(t), qui exprime la probabilité pour le composant concerné d'être en panne à l'instant t peut alors être définie comme suit:

$$F(t) = 1-R(t)$$

EXEMPLE: nous avons vu que pour t = M.T.T.F, R(t) = 0,37. De ce fait,

$$F(\theta) = 1\text{-}0{,}37 = 0{,}63.$$

La probabilité pour qu'une panne survienne avant le M.T.B.F est donc d'environ 63 %.

XII.3.FIABILITÉ D'UN ENSEMBLE DE COMPOSANTS:

XII.3.1.INTRODUCTION:

Pour s'exécuter, une application logicielle utilise un certain nombre de composants physiques de la plate-forme matérielle qui la supporte. Par exemple: des ordinateurs, des disques durs, des liaisons de données, etc.

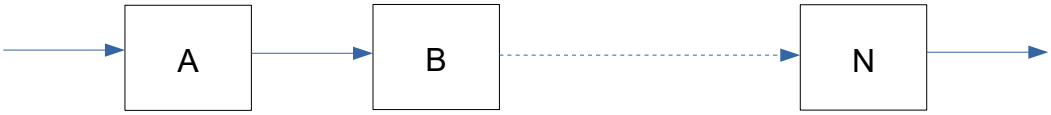
En général, ces composants sont pris "sur étagère" et leurs paramètres de fiabilité (M.T.B.F, taux de pannes, etc.) sont donnés par les constructeurs. Dans le cas contraire où ces composants sont des assemblages réalisés spécifiquement pour le projet (par exemple, un système assemblé autour d'un bus industriel), il faudra calculer leurs paramètres de fiabilité à partir de ceux de leurs composants de base.

Dans les deux cas, le problème à résoudre pour le concepteur revient à calculer la fiabilité d'un ensemble de composants dont on connaît les paramètres de fiabilité. En fonction de la manière dont chacun des composants affecte la fiabilité globale de l'ensemble, nous pouvons distinguer 3 cas:

- Le cas ou le fonctionnement de l'ensemble est conditionné par le fonctionnement de chacun des composants;
- Le cas ou le fonctionnement de l'ensemble est assuré si un au moins des composants fonctionne (redondance matérielle);
- Le cas hybrides ou le fonctionnement de certains des composants conditionne le fonctionnement global alors que d'autres composants sont redondants entre eux.

XII.3.2.CAS OU LE FONCTIONNEMENT DE L'ENSEMBLE EST CONDITIONNÉ PAR LE FONCTIONNEMENT DE CHACUN DES COMPOSANTS:

Nous pouvons modéliser ce cas sous la forme de composants agissant "en série" pour réaliser la fonction de l'ensemble:



C'est le cas, par exemple, du branchement "en série" d'ampoules à incandescence: la coupure du filament de n'importe quelle ampoule de la guirlande entraîne l'extinction de toutes les ampoules.

On dit dans ce cas que chacun des composants est CRITIQUE pour le fonctionnement de l'ensemble.

La fiabilité RT d'un ensemble de n composants a, b, c,…, n, tous critiques pour le fonctionnement de l'ensemble est égale au produit des fiabilités de chaque élément:

$$RS = R_a \text{ X } R_b \text{ X } R_c \text{ X } ..... \ R_n$$

$$\Rightarrow RS(t) = e^{-\lambda 1.t} \text{ X } e^{-\lambda 2.t} \text{ X } e^{-\lambda 3.t} \text{ X.....X } e^{-(\lambda 1+\lambda 2+...+\lambda n).t} = e^{-\lambda S.t}$$

λS étant la somme des taux de panne des différents éléments. Nous pouvons en déduire que le taux de pannes d'un ensemble de composants "en série" est la somme des taux de panne des différents composants, puisque les taux de pannes s'ajoutent. De ce fait, la "probabilité de survie" de l'ensemble est inférieure à celle de l'élément le moins fiable.

Le M.T.T.F de l'ensemble est l'inverse du taux de pannes de l'ensemble:

$$M.T.T.F = 1/ (\lambda 1+\lambda 2+...+\lambda n)$$

En particulier, si les n éléments ont le même taux de panne λ, $RS(t) = e^{-n\lambda t}$

CAS PARTICULIER DE DEUX COMPOSANTS:

Nous pouvons écrire;

$$RS(t) = e^{-\lambda 1.t} \cdot e^{-\lambda 2.t} = e^{-(\lambda 1 + \lambda 2).t}$$

De ce fait, $\lambda S = \lambda 1 + \lambda 2 = 1/\mu 1+1/\mu 2 = (\mu 1 + \mu 2)/(\mu 1.\mu 2)$ et le MTBF de l'ensemble vaut:

$$\boldsymbol{\mu S = (\mu 1.\mu 2) / (\mu 1 + \mu 2)}$$

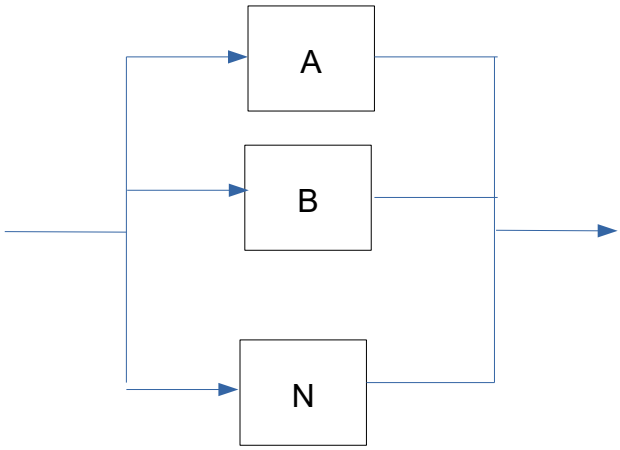
CAS DE N COMPOSANTS AYANT LE MÊME TAUX DE PANNES:

En reprenant la formule générale, on obtient:

- Pour deux composants, μS = μ/2;
- Pour trois composants, μS = μ/3;
- Pour n composants, μS = μ/n.

XII.3.3.CAS OU LE FONCTIONNEMENT DE L'ENSEMBLE EST ASSURÉ DÉS QU'UN AU MOINS DES COMPOSANTS FONCTIONNE:

Dans ce cas, le fonctionnement de l'ensemble peut être assuré tant qu'au moins un des composants fonctionne. On dit alors que ces composants sont REDONDANTS:



C'est le cas, par exemple, du branchement "en parallèle" d'ampoules à incandescence: la coupure du filament de n'importe quelle ampoule de la guirlande n'entraîne pas l'extinction de toutes les ampoules. Seule la défaillance de toutes les ampoules provoque l'absence d'éclairage.

La probabilité de panne d'un ensemble composé d'éléments "montés en parallèle" (c'est à dire dont chacun des éléments peut assurer le fonctionnement de l'ensemble) est égale au produit des probabilités de pannes de chacun des composants:

$$FT = F_1 X\ F_2\ X...X\ F_n = (\ 1\text{-}R_1\ )\ x\ (\ 1\text{-}R_2\ )\ x\ \ldots\ x\ (\ 1\text{-}R_n\ )\quad (\ car\ F_i = 1 - R_i\ )$$

Les valeurs des facteurs $1\text{-}R_i$ appartenant à l'intervalle [0, 1], cette formule montre que la probabilité de panne diminue rapidement avec le nombre de composants concernés. En particulier, si tous les composants ont la même probabilité, de panne, on obtient:

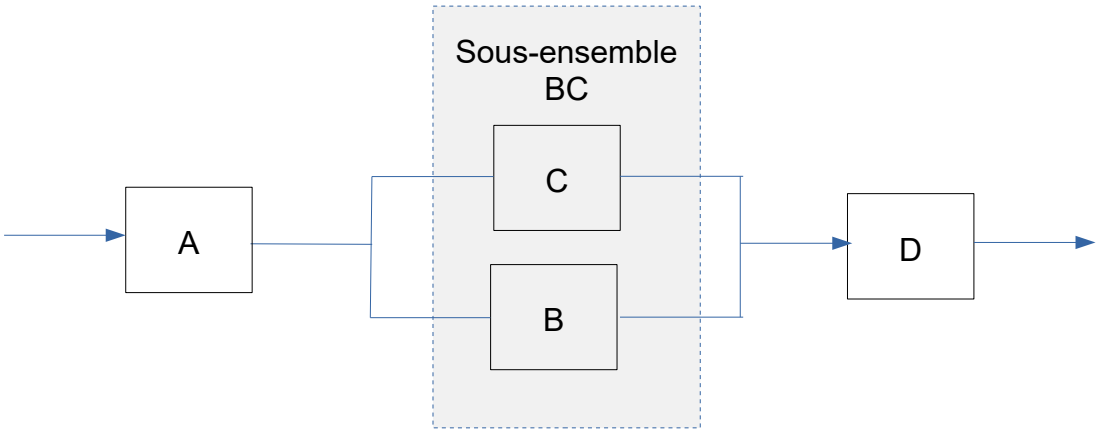
$$FT = (\ 1 - R\ )^n \quad et\ RT = 1 - (\ 1 - R\ )^n$$

REMARQUE: en cas de panne de composants, le fonctionnement global est assuré par les composants encore en état, ce qui augmente l'espérance de vie de l'ensemble, mais la fiabilité de cet ensemble diminue.

EXEMPLE: Supposons qu'un ensemble soit composé de deux machines redondantes identiques A et B. Pour une valeur de t égale au M.T.T.F de chacune des machines la fiabilité de celles-ci est R = 0,63 (63%). La fiabilité du groupe est alors $RT = 1-(1-0,63)^2 = 0,86$ (soit 86%).

XII.3.4.CAS HYBRIDE: CERTAINS DES COMPOSANTS SONT CRITIQUES, D'AUTRES SONT REDONDANTS:

Le schéma ci-dessus donne une représentation graphique "symbolique" de ce cas:



La fiabilité de l'ensemble est le produit des fiabilités des composants non redondants et des fiabilité des sous-ensembles redondants:

EXEMPLE: Dans l'exemple ci-dessus:

On calcule la probabilité de panne de l'ensemble redondant BC:

$$F_{bc} = F_b \text{ X } F_c = (1 - R_b) \text{ X } (1 - R_c)$$

On en déduit la fiabilité de BC:

$$R_{bc} = 1 - F_{bc} = 1 - (1 - R_b) \text{ X } (1 - R_c) = R_c + R_b - R_b X R_c$$

Puis on calcule la fiabilité de l'ensemble composé des éléments A, BC et D disposés "en série":

$$R_S = R_A \text{ X } R_{BC} \text{ X } R_D$$

XIII.ANNEXE D – PROGRAMMATION EN ENVIRONNEMENT MULTITÂCHES:

XIII.1.INTRODUCTION :

La plupart des applications informatiques sont amenées à échanger des messages avec les différentes entités qui composent leur environnement d'exécution (plate-forme matérielle et logicielle ou équipements physiques connectés à ces plate-formes). Ces messages peuvent transporter des informations de plusieurs types (données, commandes, signaux, exceptions, etc.).

Les contraintes de nature temporelle qui pèsent sur la prise en compte de ces messages par l'application ou sur leur délivrance aux entités extérieures destinataires concernent essentiellement:

- **Les délais de réaction:** par exemple, la durée de la boucle de réaction acquisition d'une commande externe → traitement → restitution de la réponse à l'émetteur de cette commande, que l'on désigne souvent par "temps de réponse" ;
- **Le déterminisme des dates** d'acquisition ou de restitution des réponses .

Les contraintes temporelles concernant les délais et le déterminisme des dates sont habituellement classées en quatre niveaux de sévérité:

Niveau de contrainte	Temps de réponse (ms)	Déterminisme des dates (ms)	Types d'activités (exemples)
TEMPS RÉEL DUR	< 1	ΔT < 1	Conduite de processus critiques, applications de guidage embarquées, etc., nécessitant un système d'exploitation adapté (Lynx OS, RT Linux, etc.)
TEMPS RÉEL INTERMÉDIAIRE	1 <= T < 10	1 <= ΔT< 10	Temps réel peu contraint, pouvant s'exécuter sous win32
TEMPS RÉEL SOUPLE	10 <= T < 100	10 <= ΔT < 100	Conduite de processus peu contraints, jeux vidéo, etc.
TEMPS RÉEL INTERACTIF	100 <= T < 1000 (temps de réaction "humain")	100 <= ΔT < 1000	Applications interactives
HORS TEMPS RÉEL	Aucune exigence	Aucune exigence	Applications de bureautique, de traitement photos, etc.

RAPPELS:

- *En informatique, un système ou une application sont qualifiés de TEMPS RÉEL lorsqu'ils sont capable de contrôler et de commander un processus physique avec des temps de réponse compatibles avec les exigences de celui-ci en la matière ;*

- *Un système temps réel n'est pas forcément un système réagissant rapidement : sa principale qualité est plutôt d'avoir des temps de réaction aux événements PRÉVISIBLES (on emploie parfois le terme DÉTERMINISTE).*

XIII.2.PARTAGE DES RESSOURCES EN MODE MULTITÂCHES :

XIII.2.1.NOTION DE RESSOURCE :

En informatique, le terme RESSOURCE s'applique a toute entité matérielle ou logicielle à laquelle une tâche a besoin d'accéder pour s'exécuter. Il peut s'agir d'un fichiers, d'une connexion réseau, d'une zone de mémoire vive, d'un périphérique externe, d'un sous-programme d'une bibliothèque logicielle, ou même d'entités plus abstraites comme une partie de la puissance d'un processeur ou de la bande passante d'un réseau.

XIII.2.2.NOTION DE RESSOURCE CRITIQUE :

Certaines ressources ne peuvent être utilisées SIMULTANÉMENT que par un seul utilisateur (un seul thread) ou par un nombre d'utilisateurs limité. Par exemple :

- Une imprimante ne peut être utilisée que par un seul thread à la fois ;
- Un fichier disque ne peut être utilisé **en écriture** que par un seul thread à la fois (en revanche, il peut être utilisé "simultanément" par plusieurs threads **en lecture**) ;
- Un **pool** de 5 imprimantes peut être utilisé par jusqu'à cinq threads à la fois;
- Une ZONE DE MÉMOIRE partageable stockant des informations récurrentes d'une application ne peut être utilisée simultanément par plusieurs threads sans risque pour la cohérence des données (par exemple, une zone structurée de données ne peut être lue par un thread alors qu'elle est en cours de modification par un autre thread sans risquer de rendre le contenu de cette structure incohérent) ;
- Un PORT RÉSEAU ne peut être ouvert en réception que par un seul processus à la fois ;
- etc.

Ce type de ressource est appelé RESSOURCE CRITIQUE.

REMARQUE : en programmation multitâches, un exécuteur fixe (ou un cœur de processus) peut être considéré comme une ressource partageable. Ainsi, un processeur à 4 cœurs peut être considéré comme une ressource admettant 4 utilisateurs simultanés.

XIII.3.COMMUNICATION ET SYNCHRONISATION INTER-PROCESSUS :

XIII.3.1.INTRODUCTION :

La principale difficulté de la programmation dans un contexte multitâches où les différents processus peuvent s'exécuter simultanément (en simultanéité apparente ou réelle) est qu'elle introduit un certain indéterminisme dans l'ordre d'exécution des traitements. De ce fait, il est possible que différents threads, pouvant appartenir à des processus différents, essaient d'accéder simultanément à une même RESSOURCE. Or, nous avons vu que l'accès SIMULTANÉ de plusieurs threads à une même ressource peut parfois provoquer des dysfonctionnements.

Une autre difficulté majeure réside dans le fait que chaque processus possède un contexte d'exécution qui lui est propre : lorsqu'un processus tente d'accéder directement à une adresse mémoire située en dehors de son espace d'exécution, il est sanctionnée par une alarme (violation mémoire).

De ce fait, les différents langages de programmation mettent à disposition des développeurs des bibliothèques d'outils (fonctions ou classes d'objets) qui leur permettent de surmonter ces difficultés. Ces bibliothèques sont désignées par le sigle I.P.C (Inter Process Communication – Communication inter processus).

Les outils I.P.C peuvent être classés en trois types :

- Les outils de synchronisation qui permettent de gérer les accès concurrents aux ressources partagées (verrous, sémaphores, mutex, etc.) ;
- Les outils de communication qui permettent aux processus d'échanger des messages (signaux, tubes, etc.) ;
- Les outils de partage de zones mémoires (sheared memories – zones de mémoires partageables).

REMARQUE : certains mécanismes existants qui ne sont pas classés parmi les I.P.C permettent également à deux processus de communiquer entre eux. Nous pouvons citer :

- Les communications réseau en mode TCP/IP (ou équivalents) ;
- Les appels de procédures à distance (CORBA, RPC, etc.).

XIII.3.2.PARTAGE DE DONNÉES ENTRE PROCESSUS:

XIII.3.2.1.LES FICHIERS:

Plusieurs processus peuvent avoir accès aux données contenues dans un même fichier situé sur un support de masse. Cette solution permet de partager des données entre processus ou de transmettre des données d'un processus à un autre de manière asynchrone (l'émetteur dépose son message dans le fichier, le destinataire le récupère

quand il en a besoin). Elle a l'avantage d'être disponible même si l'on ne dispose pas de bibliothèque I.P.C. L'inconvénient est la lenteur relative des accès (de l'ordre de quelques ms).

XIII.3.2.2.LES BASES DE DONNÉES:

Les bases de données sont également utilisées pour partager des données entre processus. Comme en ce qui concerne les fichiers "à plat", cette solution a l'avantage d'être disponible même si l'on ne dispose pas de bibliothèque I.P.C. Là aussi, l'inconvénient est la lenteur relative des accès (de l'ordre de quelques ms).

XIII.3.2.3.LES ZONES MÉMOIRES PARTAGÉES (SHEARED MEMORIES) :

Les threads d'un processus donné ne peuvent accéder à l'espace d'exécution d'un autre processus. De ce fait, les systèmes d'exploitation mettent à disposition des utilisateurs (développeurs) des mécanismes qui permettent à des threads appartenant à des processus non apparentés de partager des zones mémoires. Il s'agit des mécanismes de SHEARED MEMORY (mémoire partagée). Une "Sheared Memory" peut être assimilée à un FICHIER EN MÉMOIRE.

XIII.3.3.MÉCANISMES DE GESTION DES ACCÈS AUX RESSOURCES :

XIII.3.3.1.NOTION DE SECTION CRITIQUE :

Une SECTION CRITIQUE est une portion de code caractérisé par le fait qu'elle tente d'accéder à une ressource critique pouvant être partagée entre plusieurs threads de manière asynchrone. L'accès simultané de plusieurs threads à cette ressource peut, dans ce cas, entraîner des dysfonctionnements ou des incohérences.

EXEMPLE : Supposons qu'une application de gestion de stocks gère un catalogue d'articles enregistré dans un fichier disque sous format *.csv (fichier tableur). Dans cette table, chaque ligne (entrée) correspond à un article particulier et une des colonne est dédiée au stock disponible pour chaque article. Exemple :

NomArticle	Marque	NumNomenclature	Prix TH (euros)	StockDisponible
Boite de vitesses	Renault	8787777575789	1894	15
Embrayage	Citroën	6785645343134	789	32
Frein avant droit	Mercedes	1233423231444	342	43
Etc….				

Une routines (sous-programme) de l'application permet de tenir à jour les stocks. L'interface d'appel de cette routine correspond au schéma suivant :

entier MiseAJourStock (NumNomenclature, Quantité) ;

La routine renvoie l'état du stock après mise à jour ou -1 si le stock est insuffisant.

Supposons également que la routine soit susceptible d'être utilisée simultanément par plusieurs utilisateurs utilisant des postes clients déportés. Dans ce cas, il peut exister un conflit potentiel pour l'accès à la ressource Fichier Catalogue..

En effet, dans cette routine, nous allons probablement rencontrer le code suivante :

```
entier MiseAJourStock ( NumNomenclature, Quantité )
{
        chaîne      TABLE [1500][5];      // 1500 articles, 5 champs par article
        entier      Stock;
        ……
        Table = Lire FichierCatalogue ();
        Stock =Table [ NumNomenclature][ StockDisponible];
        SI ( Stock ≥ Quantité )
        {
                Stock = Stock - Quantité;
                Table [ NunNomenclature][StockDisponible] = Stock;
                Ecrire FichierCatalogue ( Table );
```

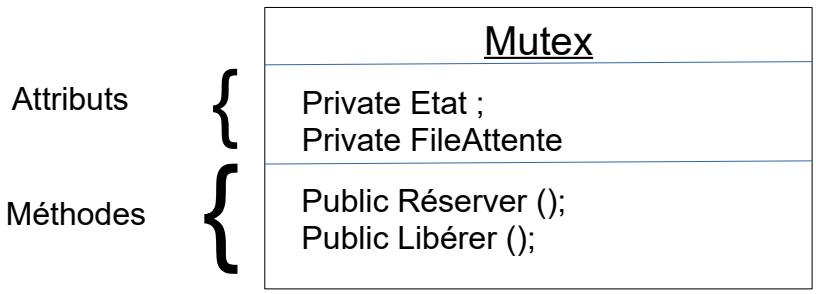
```
        }
        SINON
                Stock = -1;
        FINSI
        ……
        Retourner Stock;
}
```

Si deux threads accèdent en même temps à la section marquée en rouge, un des threads peut acquérir la valeur du stock de l'article dans le fichier alors que cette valeur est en cours de modification par l'autre thread, mais pas encore enregistrée dans ce fichier. La valeur du stock restant peut donc devenir aléatoire.
La section de code marquée en rouge est donc une section critique (car accédant à la ressource critique qu'est ici le fichier catalogue). Cette section doit donc être protégée des accès simultanés. Nous verrons plus tard les méthodes permettant cette protection.

XIII.3.3.1.**LES MUTEX:**

Les MUTEX (Mutual Exclusion) permettent à un thread donné de se réserver l'accès à une ressource partagée si celle-ci est inoccupée. La réservation d'une ressource bloque l'accès de cette ressource pour les autres threads (exclusion mutuelle): ceux-ci sont placés en attente de sa libération.

Un mutex peut être représenté par un objet M dont la classe peut être décrite comme suit :



- L'attribut privé État est un booléen représentant les deux états possibles du mutex ("réservé" ou "disponible) ;
- L'attribut privé FileAttente est une file qui permet de stocker les identificateurs des threads en attente de la libération du mutex ;
- La méthode Réserver interroge l'état du mutex :
  - Si cet état est "disponible", elle le fait passer à l'état "réservé" et laisse le thread appelant se dérouler (donc, utiliser la ressource) ;

    - Si cet état est "réservé", elle introduit l'identifiant du thread appelant dans la file d'attente et place ce thread à l'état "attente" ;
- La méthode Libérer replace le mutex à l'état "disponible" et fait repasser le premier thread de la fille d'attente à l'état "actif".

Supposons que le développeur décide d'affecter le mutex M à la protection d'une ressource R. Dans ce cas, si un thread veut utiliser cette ressource, il devra auparavant interroger le mutex M en utilisant la méthode M.Réserver ().

1. Si M est à l'état "disponible", la ressource est libre. Un thread appelant continuera de s'exécuter et État passera à l'état "réservé".
2. Si M est à l'état "réservé", le thread appelant sera placé à l'état "attente" et son Id. sera introduite dans la file d'attente ;
3. Lorsqu'un thread ayant réussi à réserver le mutex et à accéder à la ressource a terminé son utilisation de la ressource, il appelle la méthode M.Libérer(), ce qui a pour effet de faire passer l'état du mutex à "disponible". Le mutex fait alors passer le premier thread ÉLIGIBLE (voir ci-dessous) de la file d'attente à l'état "actif" (si la file d'attente n'est pas vide) et lui permet d'accéder à la ressource tout en repassant à l'état "réservé" ;
4. Et ainsi de suite.

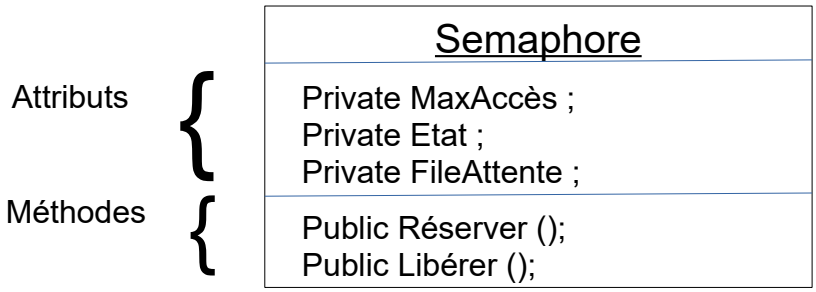
REMARQUES:

- Un mutex n'autorise donc qu'un seul thread à la fois à accéder à la ressource qu'il protège ;
- Nous avons vu plus haut (action n°3 de la liste ci-dessous) que lorsqu'un thread libère le mutex, celui-ci débloque le premier thread ÉLIGIBLE de la file d'attente à l'état actif et le fait accéder à la ressource. Suivant la police adoptée par le mutex, le premier thread éligible peut être :
    - Le thread le plus ancien dans la file d'attente (police FIFO) ;
    - Le thread le plus prioritaire de la file d'attente;
    - Une combinaison de ces deux cas : par exemple, certains algorithmes augmentent la priorité d'un thread en fonction de la durée de son attente dans la file : ceci permet de garantir qu'un thread ne restera pas indéfiniment en attente (cas de famine).

XIII.3.3.2.LES SÉMAPHORES:

Les sémaphores sont des mutex évolués qui permettent à plusieurs threads d'accéder simultanément à une ressource partagée, si celle-ci le permet.

Un sémaphore peut être représenté par un objet M dont la classe peut être décrite comme suit :



Dans ce cas, MaxAcces représente le nombre d'accès autorisés pour la ressource. Il est initialisé lors de l'instanciation de la classe. État contient le nombre d'accès réservés à un instant T (MaxAcces > État > 0).

Le mécanisme d'un sémaphore est identique à celui d'un mutex à par le fait que le sémaphore peut permettre à plusieurs threads d'accéder simultanément à la ressource.

Supposons que le développeur décide d'affecter le sémaphore M à la protection d'une ressource R admettant X accès. Dans ce cas, le sémaphore doit être créé avec MaxAccès = X.Si un thread veut utiliser cette ressource, il devra auparavant interroger le sémaphore S en utilisant la méthode M.Réserver ().

- Si S.Etat > 0, la ressource peut admettre un accès supplémentaire. le thread appelant continuera de s'exécuter et État sera décrémenté de 1.
- Si S.Etat = 0, le thread appelant sera placé à l'état "attente" et son Id. sera introduite dans la file d'attente ;
- Lorsqu'un thread ayant réussi à réserver le mutex et à accéder à la ressource a terminé son utilisation de la ressource, il appelle la méthode S.Libérer(), ce qui a pour effet d'incrémenter S.Etat de 1 (si S.Etat < MaxAccès) et de faire passer le premier thread de la file d'attente à l'état "actif" si la file d'attente n'est pas vide;
- Ce thread activé réinterrogera alors automatiquement le sémaphore par S.Réserver() ;
- Et ainsi de suite.

REMARQUE : un sémaphore dont MaxAccès est initialisé à 1 se comporte comme un mutex.

XIII.3.3.3.UTILISATION DES SÉMAPHORES ET DES MUTEX :

Un thread utilise les méthodes M.Réserver () et M.Libérer () respectivement à l'entrée et à la sortie d'une SECTION CRITIQUE.

SCHÉMA DE PROGRAMMATION:

```
DÉBUT THREAD
        S = Instanciation de SEMAPHORE ( NbAccès ) ;
        ------------------------------------------------------------
        ------------------------------------------------------------
        S.Réserver () ;
        < utiliser la ressource R > ;    // Section critique à protéger
        S.Libérer ;
        ……  Suite du thread …… ;
```

XIII.3.4.ÉCHANGE DE DONNÉES ENTRE PROCESSUS LOCAUX:

XIII.3.4.1.LES TUBES (PIPES):

Les TUBES sont des mécanismes de communication qui permettent à un processus (l'émetteur) de communiquer des données à un autre processus QUI LUI EST APPARENTÉ (le récepteur) et à un seul. Les tubes sont implémentés dans les systèmes d'exploitation Unix/Linux et Windows.

Un TUBE est matérialisé dans l'OS par un descripteur qui définit ses extrémités d'entrée et de sortie. La communication est unidirectionnelle. Le tube fonctionne comme une file d'attente FIFO. La taille de cette file est fixée lors de la création du tube.

Il existe deux sortes de tubes :

- Les tubes ANONYMES qui sont créés uniquement dans le cadre de la communication entre deux processus bien déterminés et ne "survivent" pas à ceux-ci ;
- Les tubes NOMMÉS , qui reçoivent un identificateur système et doivent être explicitement supprimés.

Lorsqu'un processus tente d'introduire des données dans un tube qui ne possède plus assez d'espace pour les contenir, l'OS passe ce processus en attente jusqu'à ce qu'assez d'espace se libère. De même, lorsqu'un processus tente de lire un tube vide, l'OS le passe en attente jusqu'à ce que le tube contienne de nouveau des données.

XIII.3.4.2.LES FILES DE MESSAGES (MESSAGES QUEUES) :

Les files d'attente de messages (Messages Queues) sont des listes chaînée de messages gérées par les systèmes d'exploitation. Elles sont identifiées dans le système d'exploitation par un identificateur particulier.

Lorsqu'un nouveau message est ajouté par un processus dans la file, il est placé à la fin de la file.

Chaque message est composé d'un champ d'identification, d'un champ "longueur" et d'une suite d'octets correspondant aux données à transmettre. Les processus récepteurs peuvent récupérer les messages qui les intéressent directement grâce à leurs identificateurs.

Tous les processus (même non apparentés) peuvent échanger des informations via une file d'attente : Le processus émetteur introduit un message dans la file. Tout autre processeur peut lire ce message à condition de partager une CLEF commune avec l'émetteur.

XIII.3.5.LES SIGNAUX ET ÉVÉNEMENTS INTER PROCESSUS :

XIII.3.5.1.MÉCANISMES DES SIGNAUX (O.S DE TYPE UNIX) :

PRINCIPE :

Le mécanisme des signaux est très proche dans son principe de celui des interruptions prioritaires. En effet, il consiste à activer dans le processus destinataire du signal un thread chargé de traiter ce signal. Ce thread exécute donc un "gestionnaire de signal" comme une interruption prioritaire lance l'exécution d'un "gestionnaire d'interruption". Le mécanisme des signaux revient donc à activer des "interruptions logicielles" dans le processus cible.

MÉCANISME :

Émission de signaux :

Le processus expéditeur du signal utilise pour déclenchée l'émission une primitive dont le prototype ressemble à :

```
<valeur de retour> envoyer_signal
(
        <Id. process. récepteur>, <Id. signal> [, <argument à transmettre>] ;
)
```

Exemple: *en langage c, cette primitive peut être : kill (id_recepteur, id_signal) ou bien sigqueue (id_recepteur, id_signal, argument).*

Réception de signaux :

Le processus récepteur utilise pour fixer le comportement attendu à la réception du signal une primitive dont le prototype le plus simple pourrait être :

```
<valeur de retour>  configurer_signal
(
        <Id. signal>, <fonction gestionnaire du signal> ;
)
```

Exemple: *en langage c, cette primitive peut être : signal (num_signal, nom_gestionnaire) ou bien sigaction (num_signal, descripteur, ancien_descripteur) ; dans laquelle les structures descripteurs et ancien_descripteur permettent de définir le gestionnaire de signal à utiliser (ou le gestionnaire par défaut pour les signaux standards), les valeurs des masques associés aux signaux et certaines autres caractéristiques du comportement.*

Comportement du récepteur :

Après avoir fixé les modalités de son comportement lors de la survenue du signal (gestionnaire de signal, masques, etc.), le processus récepteur peut adopter un des deux comportements suivant :

- Soit il continue de s'exécute sans se préoccuper de l'arrivée du signal : c'est le mode non bloquant (l'arrivée du signal va de toute façons lancer un thread qui va exécuter le gestionnaire de signal) ;
- Soit il met un thread en attente de l'arrivée du signal : c'est le mode bloquant.

Dans ce dernier cas, il utilise une procédure dont le prototype peut être décrit comme suit :

```
<valeur de retour>  attendre_signal
(
        <collection de signaux attendus>, <informations transmises par le signal> ;
)
```

Nous voyons qu'un thread peut être mis en attente de plusieurs signaux.

Exemple : *en langage c, on utilise par exemple la primitive sigwaitinfo :*

*sigwaitinfo (const sigset_t *set, siginfo_t *info);*

SIGNAUX STANDARDS :

A l'origine, le mécanisme des signaux intégré aux O.S. de type UNIX (UNIX/LINUX) était destiné à permettre au système d'exploitation de signaler aux processus des anomalies survenues au cours de leur exécution et éventuellement de les arrêter ("les tuer"). Par exemple, un signal pouvait être envoyé à un processus pour lui signaler qu'il tentait d'accéder à l'extérieur de son espace de travail ou qu'il effectuait une division par zéro.

Les signaux concernés par ce premier mécanisme sont souvent appelés "signaux standards" et associés à des identifiants littéraux tels que SIGKILL (arrêt du processus récepteur), SIGILL (instructions illégale) ou SIGFPE (erreur de virgule flottante). Ils ne sont pas configurables par les développeurs.

SIGNAUX DITS "TEMPS RÉEL":

Dans un environnement multitâche et un contexte de concurrence des processus, le besoin s'est rapidement fait sentir de disposer d'un mécanismes logiciel permettant à un processus de signaler sans délais et de manière asynchrone aux autres processus des événements en relation avec les traitements applicatifs en cours d'exécution. Les signaux dits "temps réel" ont des identificateurs différents des signaux standards. Leur utilisation est entièrement fixée par le développeur.

XIII.3.5.2.MÉCANISMES DES ÉVÉNEMENTS (NOYAU WIN32-64)

Le concept de signal n'existe pas dans les systèmes d'exploitation WINDOWS. En revanche, les noyaux win32 et win64 de Windows propose le mécanisme des ÉVÉNEMENTS qui permet de remplacer en partie les signaux.

Ce mécanisme est basé sur le concept d'OBJET ÉVÉNEMENT :

- Un ÉVÉNEMENT est un objet "système" qui peut prendre deux états : "signalé" et "non signalé".
- Les primitives CreateEvent, SetEvent, ResetEvent permettent à un thread de créer un événement, de le placer à l'état signalé ou de le repasser à l'état non signalé ;
- Les primitives OpenEvent et WaitForSingleObject permettent à un thread de créer un lien avec un objet événement, puis de se mettre en attente du passage de cet objet à l'état signalé. La primitive WaitForMultipleObjects permet à un thread de se mettre en attente de plusieurs objets.

On peut donc dire que le mécanisme des événements de win32/64 est très semblable au mécanisme des signaux unix/linux utilisé en mode bloquant.

XIII.3.6.ÉCHANGES ENTRE PROCESSUS DISTANTS :

XIII.3.6.1.LES ÉCHANGES DE DONNÉES PAR RÉSEAU :

Tout processus s'exécutant dans une machine donnée peut échanger des messages avec un autre processus s'exécutant dans la même machine ou dans une autre machine reliée à la première par un réseau, par l'intermédiaire du mécanisme des SOCKETS (TCP/IP).

RAPPEL : *Un SOCKET est identifié par un couple d'informations formé par une adresse IP et un numéro de port.*

NOTA : La communication de données par réseau entre deux processus permet non seulement d'échanger des informations, mais aussi de synchroniser le récepteur avec l'émetteur, si la lecture est faite en MODE BLOQUANT : dans ce mode, le thread du récepteur qui effectue la lecture est mis en attente de la réception d'un message de l'émetteur. Cette réception le fait repasser à l'état actif. La réception a donc le même effet qu'un signal inter processus.

XIII.3.6.2.APPELS DE PROCÉDURES A DISTANCE :

Le mécanisme des appels de procédures à distance consiste en l'envoi sur un réseau d'un message particulier destiné à déclencher dans la machine réceptrice l'exécution d'un traitement. Celui-ci peut se présenter comme une procédure (suite de commandes systèmes), une fonction exécutable ou une méthode. En retour, la machine réceptrice renvoie un compte-rendu d'exécution à la machine émettrice.

Le protocole réseau RPC (remote procedure call) permet d'effectuer des appels de procédures vers un ordinateur distant. Ce protocole est du type client-serveur : le processus client émet des appels à distance à destination d'un serveur d'application qui exécute la procédure demandée dans l'environnement de ce serveur et retourne le compte-rendu au client.

L'architecture CORBA (Common Object Request Broker Architecture) permet d'activer à distance des MÉTHODES. CORBA permet de construire des applications complètes écrites dans des langages de programmation distincts et exécutés dans des processus séparés, sur des machines distinctes.

XIII.4.LA PROGRAMMATION CONCURRENTE :

XIII.4.1.INTRODUCTION:

Les différents mécanismes logiciels que nous avons abordés dans les trois sous-chapitres précédents permettent à un programmeur de gérer, dans un environnement multitâches, les interactions entre processus et les accès CONCURRENTS de ces processus aux ressources qu'ils partagent.

En ce qui concerne ce problème, il est d'usage de distinguer deux PARADIGMES DE PROGRAMMATION, opposés dans leur principe :

- La PROGRAMMATION SYNCHRONE tend à éviter cette concurrence en maîtrisant (par la conception du logiciel) les dates d'accès des différentes tâches à ces ressources. Ce résultat peut être obtenu de diverses manières (emploi d'un "langage synchrone", ajout à l'application d'un module moniteur des tâches, etc.). Dans un tel contexte, les mécanismes de gestion des accès aux ressources étudiés plus haut perdent de leur utilité, puisque les conflits sont évités ou minimisés au niveau de l'application ;
- La PROGRAMMATION CONCURRENTE, au contraire, se contente d'ORGANISER l'accès des différentes tâches aux ressources au niveau de chaque accès,en utilisant à plein ces mécanismes ainsi que les possibilités offertes par les priorités logicielles préemptives. La synchronisation des accès aux ressources ne se fait donc plus "à priori", au niveau de la conception, mais en cours d'exécution, au moment des demandes d'accès.

Ces deux paradigmes de programmation seront approfondis au moment de l'étude de la conception comportementale. En ce qui concerne la programmation concurrente, souvent plus difficile à maîtriser que la programmation synchrone, la suite du chapitre expose ses avantages et inconvénients.

XIII.4.2.AVANTAGES DE LA PROGRAMMATION CONCURRENTE:

Parmi les avantages de la programmation concurrente par rapport à une programmation synchrone, nous pouvons citer :

- L'évolutivité et l'adaptabilité des applications : l'ajout, le retrait ou la modification d'un processus ne remet pas en cause le fonctionnement général;
- La minimisation des temps d'inoccupation des processeurs. En effet, les périodes d'attente des threads sont réutilisées automatiquement par d'autres threads;
- La rapidité de la prise en compte des événements d'origine externe ou interne.

XIII.4.3.INCONVÉNIENTS:

- Une application concurrente est en général plus difficile à concevoir qu'une application synchrone. Elle est également beaucoup plus difficile à mettre au point et à tester ;
- Les changements de contextes induits par la commutation d'un processus à l'autre consomment une part non négligeable de la puissance des C.P.U. Il faut donc éviter les commutations inutiles.
- D'autre part, la gestion concurrente des accès aux ressources, lorsqu'elle s'effectue rigoureusement suivant le principe des priorités préemptive (le thread le plus prioritaire préempte le processeur au détriment de tous les autres threads), se heurte à un certain nombre de difficultés techniques. Les principales sont les cas de famine, les inversions de priorités et les interblocages qui sont étudiées dans les paragraphes suivants :

XIII.4.3.1.CAS DE FAMINE :

Un cas de famine est une situation dans laquelle un thread donné ne peut jamais accéder à une ressource nécessaire à la poursuite de son exécution. Cette ressource peut être :

- Le processeur si un ou des threads plus prioritaires accaparent celui-ci au point de ne jamais le laisser libre;
- Une autre ressource si lorsque ce thread est en concurrence pour y accéder, un autre thread plus prioritaire est toujours présent dans la file d'attente du sémaphore ou du mutex.

REMARQUE : *Cette situation peut se produire dès l'instant où la sortie d'une file d'attente de tâches (que ce soit celle du scheduler ou celle d'un mutex ou sémaphore) ne se fait pas strictement suivant la police FIFO.*

Pour éviter les cas de famine un certain nombre de dispositions peuvent être prises, telles que :

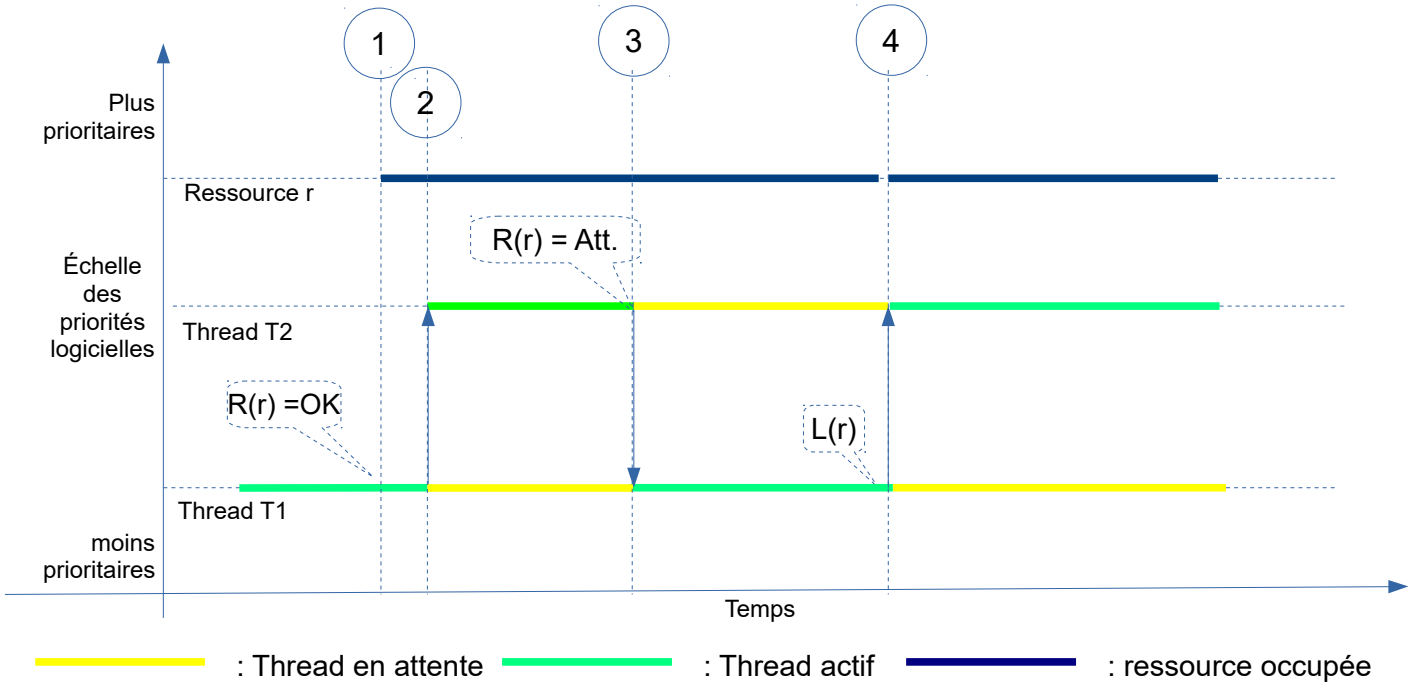
- La durée d'occupation d'une ressource par un thread doit être la plus courte possible ;
- Les threads de haute priorité doivent être réservés à la prise en compte des événements.
- Les gestionnaires d'événements doivent être le plus "maigres" possibles : Ils ne doivent contenir que les traitements strictement nécessaires à la prise en compte de ces événements (acquisition de données "au plus tôt", émission de données à des dates très précises, réveil de tâches, etc.). Le traitement des données elles-mêmes doit être autant que possible renvoyé à des thread de priorité moindre;

- Lorsque les files d'attente de tâches doivent être traitées par priorités préemptives, il importe de s'assurer que les polices de traitement des files d'attente prévoient des dispositifs assurant à chaque tâche un accès minimal aux ressources. Sinon, il faut intégrer de tels dispositifs dans l'application elle-même.

XIII.4.3.2.INVERSION DES PRIORITÉS :

Le phénomène d'inversion des priorités se produit lorsqu'un thread de faible priorité ayant réservé une ressource partagée R est interrompu par un thread plus prioritaire que lui alors qu'il n'a pas encore libéré cette ressource.

EXEMPLE 1: Soit T1 le premier thread. T1 réserve la ressource r, puis est interrompu par le thread T2 plus prioritaire. Si le thread T2 veut lui-même accéder à r, il sera contraint d'attendre que T1 ait libéré cette ressource. Il se produit donc momentanément une inversion de priorité entre T1 et T2.



COMMENTAIRES SUR LE SCHÉMA:

- 1 : Le thread T1 réserve la ressource r ;
- 2 : Le thread T2 interrompt le déroulement du thread T1 qui passe en attente ;
- 3 : Le thread T2 essaie de réserver la ressource r et échoue car elle n'a pas été libérée par T1. T2 passe en attente de la ressource, ce qui libère l'exécution de T1. Ce thread est alors traité en priorité aux dépend de T2, ce qui constitue une inversion des priorités ;
- 4 : l'inversion des priorités se poursuit jusqu'à ce que T1 ait libéré r ;

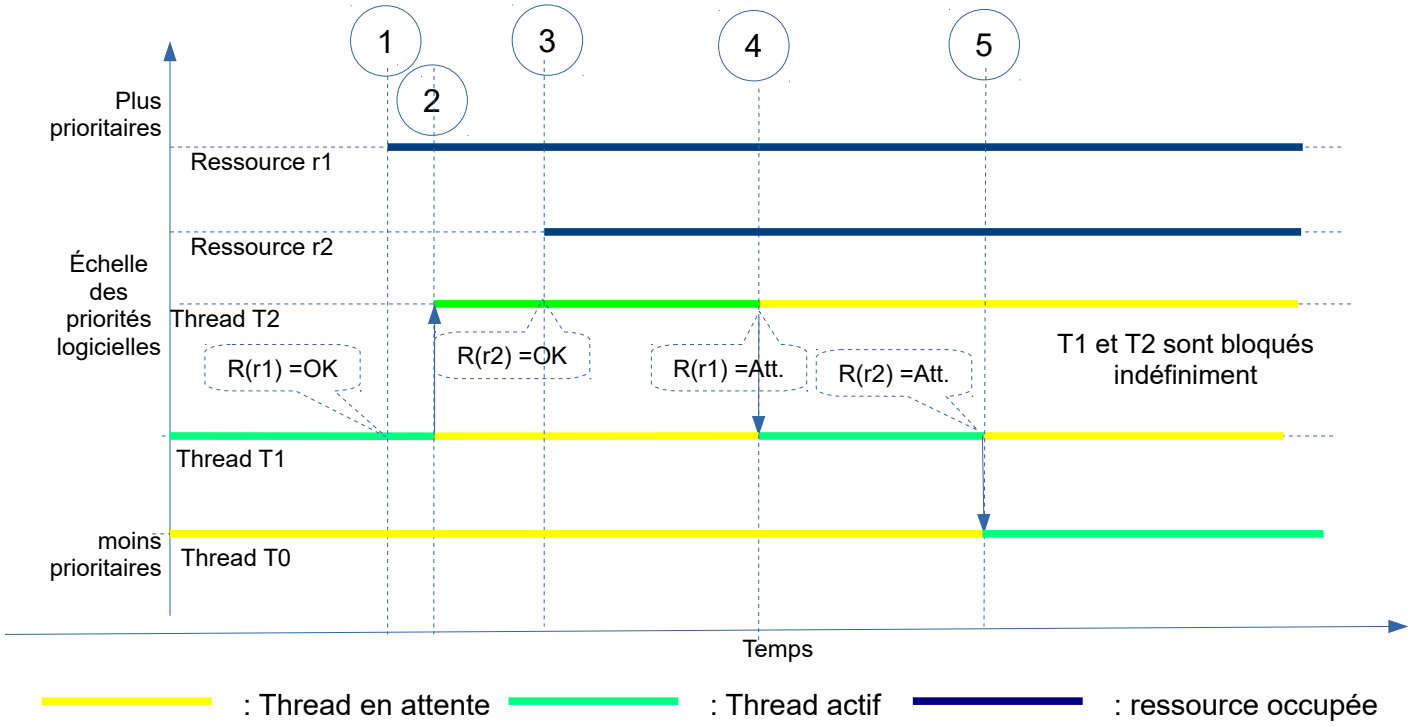
REMARQUE : l'inversion de priorité peut perturber considérablement l'exécution de l'application si le thread T1 est en concurrence avec de nombreux threads de priorité supérieure qui vont préempter le processeur et de ce fait, ralentir l'exécution De T1 Dans ce cas, le thread T2 (et également tous les threads qui requièrent la ressource r) se trouveront bloqués jusqu'à ce que T1 relâche cette ressource. Les solutions susceptible d'éviter ce type de perturbation sont :

- Éviter autant que possible qu'une ressource soit utilisée par trop de threads concurrents de priorités différentes ;
- Libérer les ressources le plus rapidement possible.

XIII.4.3.3.INTERBLOCAGES (DEADLOCKS) :

Les DEADLOCKS (étreintes mortelles) se produisent lorsque deux threads cherchent à utiliser tous deux les deux mêmes ressources, mais dans un ordre différent :

EXEMPLE :



COMMENTAIRES SUR LE SCHÉMA :

- Les threads T1 et T2 vont tous deux requérir les requêtes r1 et r2, mais dans un ordure différent et sans les relâcher entre les deux requêtes ;

- 1 : T1 est actif. Il réserve la requête r1 ;
- 2 : T2, plus prioritaire que T1, préempte de processeur aux dépens de T1 qui passe en attente ;
- 3 : T2 réserve la ressource r2 ;
- 4 : T2 essaie de réserver la ressource r1 qui est toujours réservée par T1. Il échoue donc et passe en attente. T1 redevient actif ;
- 5 : T1 essaie de réserver r1 qui est toujours réservé par T2. T1 passe en attente. T2 étant en attente de r1, un troisième thread T0 est lancé par le scheduler ;
- Conséquences : T1 et T2 se retrouvent indéfiniment bloqué et les ressources r1 et r2 sont inaccessibles.

REMARQUE : un deadlock bloque donc indéfiniment l'exécution des threads mais également celle des threads utilisant les deux ressources concernés. Il s'agit donc d'un dysfonctionnement majeur dont il est difficile de sortir par des dispositifs logiciels. Parmi les solutions susceptibles d'éviter ce type de dysfonctionnement, nous pouvons citer :

- Éviter autant que possible qu'un thread tente de réserves successivement deux ressources sans relâcher la première entre-temps ;
- S'il est impossible de faire autrement, contrôler l'accès aux deux ressources par un même sémaphore ;
- Sinon, s'assurer que les deux ressources seront toujours réservées dans le même ordre (par exemple en encapsulant la séquence de réservation dans un module logiciel qui devra impérativement être utilisé par tous les threads concernés).

XIV. ANNEXE E: EXÉCUTABLES, PROCESSUS ET THREADS :

XIV.1. NOTION D'EXÉCUTABLE :

Les outils de conception LOGIQUE modélisent souvent l'application sous la forme de GRAPHES dont les nœuds sont les COMPOSANTS LOGICIELS constituant l'APPLICATION (modules, classes, données partagées) et donc les arcs représentent les RELATIONS À CARACTÈRE STATIQUE existant entre ces composants (relation qui existent indépendamment de l'état d'exécution du logiciel : dépendances procédurales, héritages, agrégations, compositions).

Cette démarche s'intéresse donc à la STRUCTURE LOGIQUE de l'application en faisant abstraction de la manière dont les composants de cette structure vont réagir lors de l'exécution.

Lors de la PRODUCTION de l'application, ces composants logiciels sont combinés entre eux pour former des fichiers EXÉCUTABLES ou INTERPRÉTABLES. En effet, chacun de ces fichiers contient un code capable d'être soit CHARGÉ EN MÉMOIRE puis EXÉCUTÉ par les processeurs de la machine support (code source C, C++, etc.), soit INTERPRÉTÉ par un programme interpréteur (codes sources Java, Python, etc.).

Ces fichiers (qui correspondent à ce que l'on appelle usuellement des PROGRAMMES) offrent aux utilisateurs un point d'entrée unique (correspondant dans le code objet de l'application à une fonction ou une méthode notées généralement "main"), qui permet de lancer leur exécution.

XIV.2. PROCESSUS ET THREADS :

XIV.2.1. DÉFINITION D'UN PROCESSUS :

Nous pouvons définir un PROCESSUS comme étant un programme en cours d'exécution auquel sont associées des ressources systèmes et un espace d'exécution privatif en mémoire. Un processus renferme au moins un THREAD en état d'être exécuté.

XIV.2.2. DÉFINITION D'UN THREAD :

Un thread est un "fil d'exécution", c'est à dire une séquence de traitements dont l'exécution peut être confiée à un PROCESSEUR unique par le système d'exploitation.

Un thread s'exécute dans l'ESPACE D'EXÉCUTION d'un PROCESSUS et a accès à une partie des ressources de celui-ci. En particulier, il partage:

- Les données STATIQUES réservées et initialisées lors du lancement du processus ;
- Les données allouées dynamiquement dans le TAS (heap) du processus.

- Le code exécutable du processus.

En revanche, il possède sa propre PILE (stack) et son propre CONTEXTE D'EXÉCUTION (valeurs instantanées des différents REGISTRES de l'exécuteur fixe).

A un instant donné, un thread peut se trouver dans un des états suivants :

- ***Prêt (ou new):*** il est chargé en mémoire vive et prêt à s'exécuter ;
- ***Actif (ou runnable) :*** il est en cours d'exécution (le système d'exploitation lui a attribué un processeur) ;
- ***En attente (ou not runnable)*** : il est en attente de la disponibilité d'une ressource ou de la survenue d'un signal ;
- ***Mort (ou dead) :*** il a été arrêté par une commande ou une exception.

XIV.2.3.CRÉATION D'UN PROCESSUS :

XIV.2.3.1.PAR UNE COMMANDE DU SYSTÈME D'EXPLOITATION :

Le lancement d'un EXÉCUTABLE par le système d'exploitation (fichier .exe ou .bat sous windows, fichier exécutable sous linux) entraîne la création d'un PROCESSUS LOGICIEL. Dans l'espace d'exécution privatif de ce processus, un premier thread est créé, dont la "cible exécutable" est le point d'entrée du fichier exécutable.

Rappelons qu'un exécutable peut être lancé plusieurs fois, créant à chaque fois un processus différent, tous ces processus exécutant le même code, mais à l'intérieur d'espaces d'exécution différents. Dans un système multitâches, ces processus pourront s'exécuter simultanément :

- En simultanéité apparente sur une machine mono processeur mono cœur ;
- Possiblement en simultanéité réelle sur une machine multiprocesseurs ou multi cœurs.

XIV.2.3.2.PAR PROGRAMME :

Les langages de programmation multitâches possèdent tous une instruction qui permet de créer un nouveau processus à partir d'un processus en cours d'exécution. Cette instruction (appelée fork en langage C), crée un processus FILS du processus en cours. Ceci signifie que le nouveau processus va être doté d'un espace d'exécution hérité de celui de son père.

XIV.2.4.CRÉATION ET EXÉCUTION D'UN THREAD:

Nous avons vu qu'un processus possède toujours au moins un thread actif ou en état de l'être. Pour ajouter un thread à un processus, il faut utiliser l'instruction adéquate du langage utilisé. Cette instruction utilise au moins deux arguments d'appel :

- Un "handler" que l'instruction va renseigner et qui permettra de pointer sur l'objet thread créé pour le manipuler (le démarrer, l'arrêter, etc.) ;
- L'adresse de la fonction ou méthode qui sera lancée au démarrage du thread (sa "cible exécutable").

EXEMPLE : En langage c, cette instruction est pthread_create.

Produit
- Nom commercial - Marque -Type d'article - Description - Prix TTC - Etc. - Commander

XV.ANNEXE F: DÉMARCHE CONCURRENTE OU DÉMARCHE SYNCHRONE :

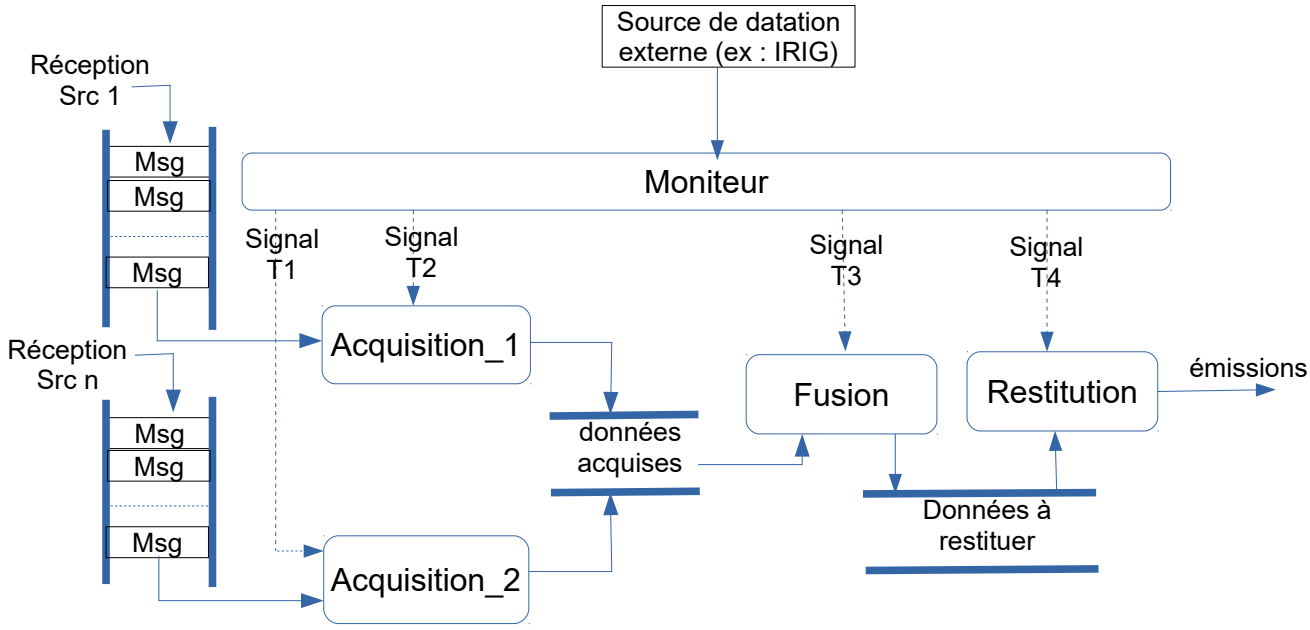
XV.1.INTRODUCTION :

Les solutions qui permettent de régler la problématique des conflits d'accès aux ressources peuvent être classées en deux catégories :

- Celle qui consiste à empêcher (ou rendre très improbables) ces conflits en répartissant dans le temps, au niveau de la conception, l'exécution des différentes tâches de façon à ce qu'elles n'accèdent jamais en même temps aux mêmes ressources. Cette démarche consiste donc à SYNCHRONISER l'exécution des différentes tâches utilisant une même ressource ;
- Celle qui consiste à laisser les différentes tâches s'exécuter indépendamment les unes des autres (d'une manière ASYNCHRONE) tout en gérant, au niveau de l'exécution, la CONCURRENCE de ces tâches pour l'accès aux ressources partagées de façon à assurer à chacune d'entre elles un accès correspondant à ses besoins.

XV.2.LA DÉMARCHE SYNCHRONE :

Dans le cadre de cette démarche, le développeur tente d'ordonner dans le temps l'accès des différentes tâches aux ressources partagées de façon à éviter tout conflit d'accès à ces ressources. Cette synchronisation est en général obtenue par l'inclusion dans l'application d'un module logiciel particulier chargé de répartir dans le temps l'exécution des autres tâches. Ce module, souvent appelé MONITEUR, utilise pour cela les signaux fournis par un générateur de signaux externes précis (autour de la milliseconde), comme par exemple, les boîtiers IRIG ou PTP). Le schéma ci-dessous modélise ce type de solution :



COMMENTAIRES :

- Dans ce type de solution, les messages reçus ne sont pas acquis immédiatement par l'application. Ils ne sont cependant pas perdus, car ils sont mis en attente par les pilotes d'équipements dans des files FIFO ("bufferisation") ;
- Le moniteur et les différents traitements qu'il active sont rassemblés dans un même processus doté d'une priorité élevée, de façon a éviter qu'il soit perturbé par l'exécution de tâches moins critiques de l'application (les I.H.M, par exemple) ;
- Le module moniteur se charge d'étaler le déclenchement des différentes tâches (ici, acquisition, fusion, restitution) de façon à éviter les conflits d'accès aux différentes ressources. Pour cela, à chacun de ses appels, il engendre des signaux de synchronisation (timers) destinés à déclencher l'exécution des tâches qu'il gère ;
- Les traitements gérés par le moniteur (ici, Acquisition_1, Acquisition$_2$, Fusion et Restitution), sont supportés par des threads. Chacun de ceux-ci est, au lancement du processus, placé en attente d'un signal particulier du moniteur (ici, T1, T2, T3 et T4). Lorsque ce signal survient, ces threads s'exécutent puis, soit retournent en attente de la prochaine survenue (traitements cycliques), soit se terminent ;
- Dans les applications cycliques (conduite de processus, simulations), l'exécution du moniteur lui-même est synchronisée avec la survenue d'un signal cyclique issu d'une source de datation externe précise et d'une résolution suffisante qui fournit le "rythme de base" de l'exécution de l'application ;
- Dans l'exemple, les tâches d'acquisition des messages viennent dépiler les messages présents dans les files d'attente à des dates bien précises pour lesquelles il est certain que les messages sont arrivés.

AVANTAGES: le principal avantage de cette solution est d'éviter les conflits d'accès aux ressources. Par exemple, dans ce schéma, si les dates de déclenchement T1, T2, T3, T4 sont bien choisies il ne peut y avoir de conflit d'accès aux ressources "données acquises" et "données à restituer". Le "timing" d'exécution est entièrement déterministe et aucune tâche ne perd du temps en attente.

INCONVÉNIENTS : Le bon fonctionnement de l'ensemble est très fortement dépendant de la conception et du paramétrage du module moniteur. Toute évolution du "timing" de réception ou d'émission des différents message peut remettre en cause le fonctionnement général et exiger une adaptation du moniteur.

CONCLUSION : La programmation synchrone convient surtout aux applications à l'environnement peu évolutif et dont le déroulement est rythmé par des événements survenant à des dates récurrentes et déterministes, comme les applications de conduite de processus de simulation ou de guidage d'engins.

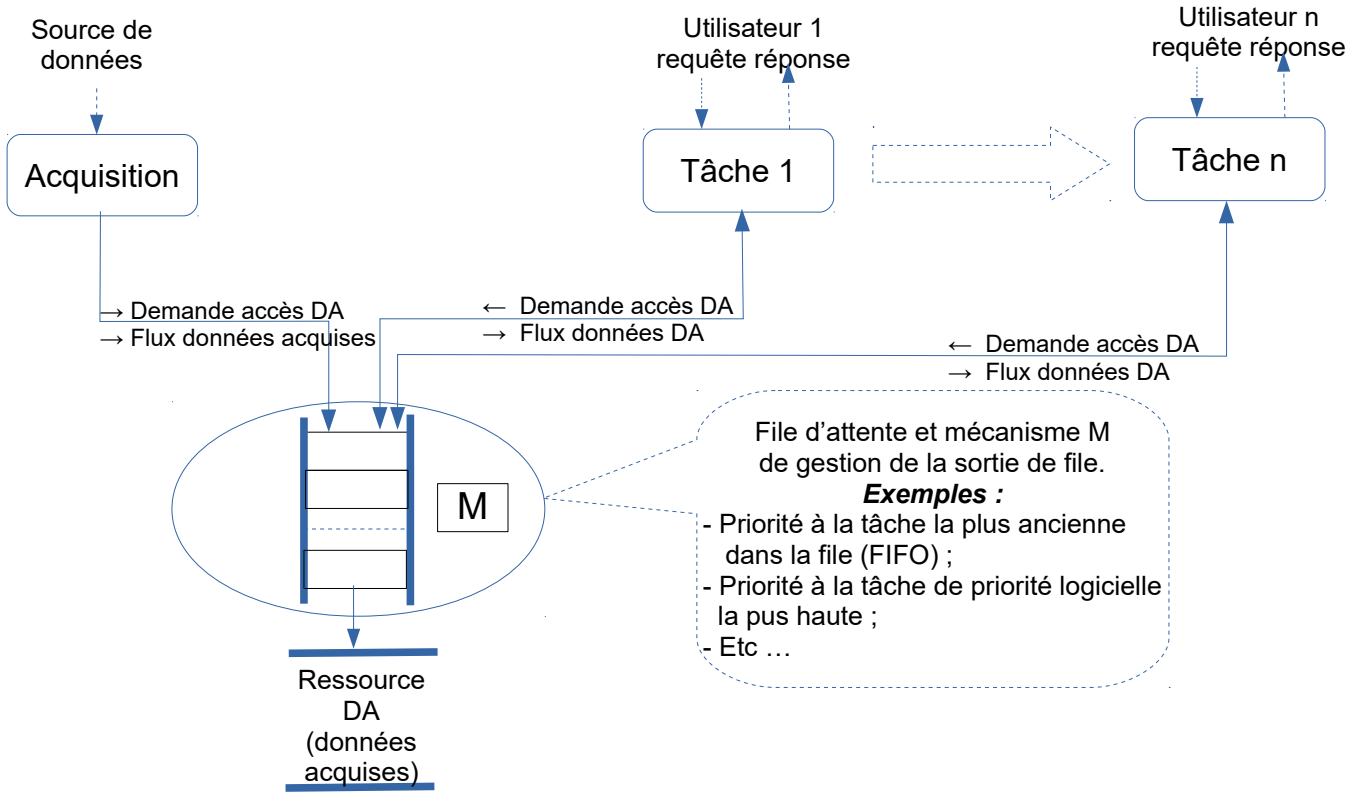
XV.3.LA DÉMARCHE CONCURRENTE :

Dans ce cas, au lieu de tenter d'éviter les conflits d'accès des différentes tâches aux ressources, le développeur se contente d'en ORGANISER l'accès en gérant des files d'attente de tâche dont la sortie peut s'effectuer selon différentes modalités:

- En FIFO (First In First out - premier entré, premier sorti) ;
- En tenant compte des priorités relatives des tâches ;
- En augmentant la priorité des tâches en fonction de la durée d'attente ;
- etc.

La synchronisation des accès aux ressources ne se fait donc plus "à priori", au niveau de la conception, mais à l'exécution, au moment des demandes d'accès. Les mécanismes permettant de gérer ces accès sont appelés "verrous inter-tâches", "mutex", "sémaphores" , etc.

Les différentes tâches sont le plus souvent déclenchées "au plus tôt" par l'arrivée des messages (réception d'une donnée, d'une commande ou d'un signal). Le schéma ci-dessous donne un exemple de ce type de solution :



COMMENTAIRES :

- L'application informatique correspondant au schéma ci-dessus pourrait être un serveur informatique qui permet à divers clients (utilisateurs 1, 2, … n) d'obtenir sur leur demande la situation météorologique fournie par une station (c'est la source de données) ;
- La source fournit des états de la situation météorologique sous la forme d'une structure de données complexes comprenant la température, la pression, la vitesse et la direction du vent pour diverses altitudes relevées par les capteurs des ballons sondes qu'elle lâche de temps en temps en fonction de l'évolution de la situation. Les dates de fourniture des états par la source est donc aléatoire. La réception d'un état déclenche une tâche d'acquisition qui sauvegarde cet état dans la zone mémoire partagée DA (données acquises).
- Chaque utilisateur du serveur interroge celui-ci en fonction de ses besoins (donc, d'une manière totalement asynchrone par rapport aux acquisition). Il reçoit en retour un état personnalisé. Chaque requête utilisateur déclenche une tâche (T1, T2, …, Tn). Ces tâches utilisent les données acquises sauvegardées dans la zone mémoire partagée DA ;
- Il existe donc une possibilité de conflit d'accès entre la tâche d'acquisition et les tâches utilisateurs concernant la données DA. En effet, les états étant des données non atomiques, la possibilité pour qu'une tâche utilisateur vienne lire un état pendant que la tâche d'acquisition est en train de le modifier (rendant ainsi la donnée incohérente) doit être considérée. Cependant, dans ce cas de figure, il est très difficile de résoudre ce problème par une solution "synchrone", en utilisant un moniteur applicatif (il faudrait envisager une boucle interrogeant périodiquement et suivant une période suffisamment rapide la survenue des divers événements, ce qui conduirait à consommer inutilement de la puissance CPU).

De ce fait, la solution présentée par le schéma ORGANISE LA CONCURRENCE des différentes tâches pour l'accès à la ressource DA. Elle utilise pour cela un mécanisme de synchronisation M basé sur une file d'attente :

- Chaque tâche T utilisant en écriture ou lecture la ressource DA doit d'abord demander à M de lui réserver l'accès à cette ressource. Si la ressource est occupée, la demande de T est placée en file d'attente. Sinon, T accède à DA, puis libère la ressource ;
- Lorsque la ressource se libère, la tâche correspondant à la première demande en attente éligible est relancée et peut accéder à DA ;
- Les conditions d'éligibilité des tâches en attente sont paramétrables : par exemple, dans l'exemple présenté, il est préférable de favoriser l'accès de la tâche d'acquisition par rapport aux autres tâches.

AVANTAGES DE LA DÉMARCHE CONCURRENTE :

L'organisation des tâches obtenue est très souple et évolutive : l'ajout d'une nouvelle tâche ne nécessité que très rarement la reprise de la conception et les risques d'effets de bord sont minimisés. La maintenance et l'évolution de l'application s'en trouvent donc facilitées.

INCONVÉNIENTS DE LA DÉMARCHE CONCURRENTE :

- Concevoir un comportement logiciel basé sur la concurrence des tâches est beaucoup plus difficile que concevoir une solution synchrone ;
- Une application basée sur la concurrence des tâches est également très difficile à tester du fait du nombre de cas d'exécution à prendre en compte.

XV.4.**CONCLUSION :**

La conception comportementale doit aboutir à une solution compatible avec les niveaux de contraintes temporelles pesant sur les différentes tâches de l'application. Notons que chacune de celles-ci peut être soumise à un niveau de contrainte qui lui est propre. Ainsi, dans la plupart des applications, nous pourrons trouver :

- Des tâches interactives sur lesquelles ne pèse aucune contrainte temps réel, comme, par exemple, les I.HM de paramétrage initiaux ;
- Des tâches interactives soumises à des contraintes temps réel "souples" (comme les I.H.M de contrôle-commande de processus) ;
- Des tâches soumises à des contraintes temps réel plus strictes, comme l'acquisition et l'exploitation de données récurrentes.

Beaucoup d'applications comprennent à la fois des tâches justifiant l'utilisation de la démarche synchrone synchrones et d'autres justifiant de la démarche concurrente .

Par exemple, une application de conduite de processus comprend à la fois une boucle de régulation dont les tâches s'exécutent de façon récurrente (qui relèvent plutôt d'une démarche synchrone) et un I.H.M de paramétrage qui, du fait du caractère aléatoire et asynchrone de ses activités, relève plutôt de la programmation concurrente.

Le choix entre traitement synchrone et traitement concurrent s'effectue donc plutôt au niveau de chacune des différentes tâches de l'application.