جامعة أبو بكر بلقايد – تلمسان

**Université Abou Bakr Belkaïd de Tlemcen**

**Faculté de Technologie**

**Département de Génie électrique et Electronique**

**Laboratoire de Recherche de Génie Biomédical**

## MEMOIRE DE PROJET DE FIN D'ETUDES

**pour obtenir le Diplôme de**

## MASTER en GENIE BIOMEDICAL

***Spécialité :*** **Signaux et Images en Médecine**

**présenté par : LAHFA Chakib Mohammed et BELHADJ Ilyas**

# ETUDE ET ELABORATIO D'UNE PLATE FORME DE TELE OPHTALMOLOGIE

**Soutenu le 23 juin 2013 devant le Jury**

| | | | | |
|---|---|---|---|---|
| M. | HAMZA CHERIF Lotfi | *MAA* | Université de Tlemcen | Président |
| M. | BENABDELLAH Mohammed | *Prof* | Université de Tlemcen | Encadreur |
| M. | LOUDJEDI Salim | *Prof* | Université de Tlemcen | Examinateur |
| M. | BEHADADA Omar | *MAA* | Université de Tlemcen | Examinateur |

**Année universitaire 2012-2013**

# Chapitre I : Introduction à la télémédecine

# I- Introduction

## 1. L'œil

### 1.1. définition

L'œil est l'un des cinq organes de sens du corps humain, c'est l'organe de la vue. La vision est donc la perception de l'organe de la vue qui en est l'œil. Ce dernier est l'organe récepteur de la lumière. Sa fonction est de transformer l'information lumineuse en influx nerveux transmis au cerveau.

Pour « voir » un objet, il faut que de la lumière issue de cet objet pénètre dans l'œil, que celui-ci la transforme en influx nerveux transmis au cerveau et que celui-ci interprète à son tour les informations reçues.

### 1.2. Anatomie

L'anatomie de l'œil se divise en deux : celle du globe oculaire et celui de ses annexes (les muscles extra-oculaires, les nerfs, la paupière, le système lacrymal et l'orbite).

#### 1.2.1. L'anatomie du globe oculaire

- Il est grossièrement sphérique. Dimensions et poids :

*Diamètre sagittal ou antéro-postérieur 25 mm (emmétrope) ; Plus court chez les hypermétropes, Plus long chez les myopes

* Diamètre transversal 23,5 mm

* Diamètre vertical 23 mm

*Poids 7 grammes.

* Volume 6,5 cm.

- Le globe oculaire peut être décomposé en quatre parties principales:

* la couche protectrice : cornée et sclère

* la couche vasculaire (aussi nommée uvée) : iris, corps ciliaire et choroïde

- la couche visuelle : rétine et nerf optique

*le contenu de la cavité interne : humeur aqueuse, cristallin et corps vitré.

A l'avant de l'œil on délimite 2 zones principales :

*la chambre antérieure : qui se situe entre la cornée et l'iris et qui est remplie par l'humeur aqueuse.

* la chambre postérieure : entre l'iris et le cristallin. **[1](figure1)**

### 1.3. Les maladies des yeux

De nombreuses maladies peuvent affecter vos yeux : cataracte, dégénérescence maculaire liée à l'âge, glaucome, rétinopathie diabétique... Découvrez les informations essentielles sur les plus fréquentes de ces maladies.

#### 1.3.1. La cataracte

Le terme de « cataracte » est connu du Grand Public car cette affection est fréquente chez les seniors.

En effet, la cataracte touche plus particulièrement les plus de 65 ans. La proportion de personnes concernées augmente avec le vieillissement : 60% des personnes de plus de 85 ans sont atteintes de cataracte**. (figure2)**

La cataracte liée à l'âge, faute de soin, est la première cause de cécité dans le monde : 48% selon l'OMS, l'Organisation Mondiale de la Santé.

L'opération de la cataracte est l'acte chirurgical le plus pratiqué en France. Grâce aux progrès des techniques chirurgicales, elle représente aujourd'hui un diagnostic moins anxiogène qu'auparavant tant les résultats de sa prise en charge sont satisfaisants.

La cataracte peut plus rarement concerner une population plus jeune pour des causes diverses : suite d'un traumatisme oculaire, origine congénitale (avant la naissance), suite médicamenteuse, etc.

Il existe une cinquantaine de formes de cataractes.

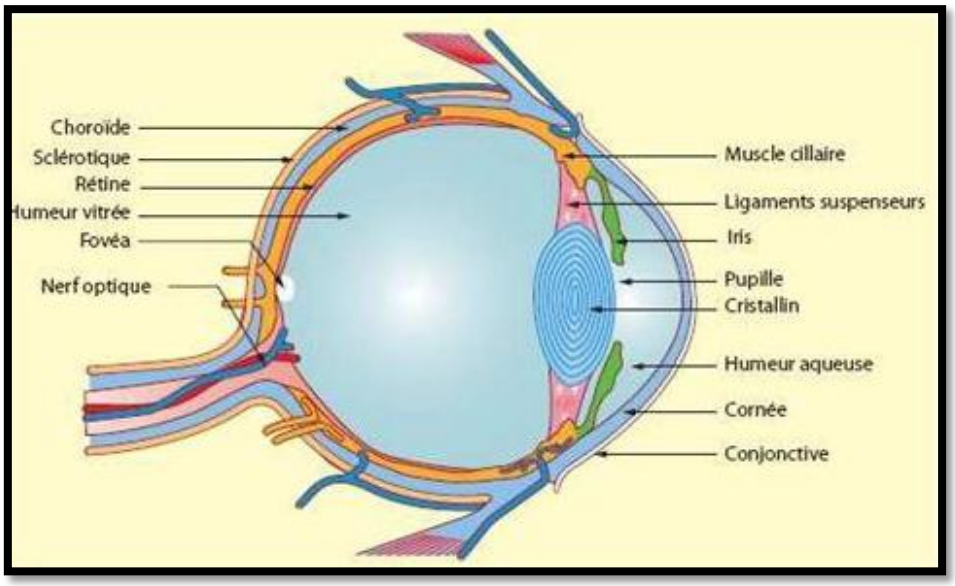



**Figure 1** : schéma de l'œil

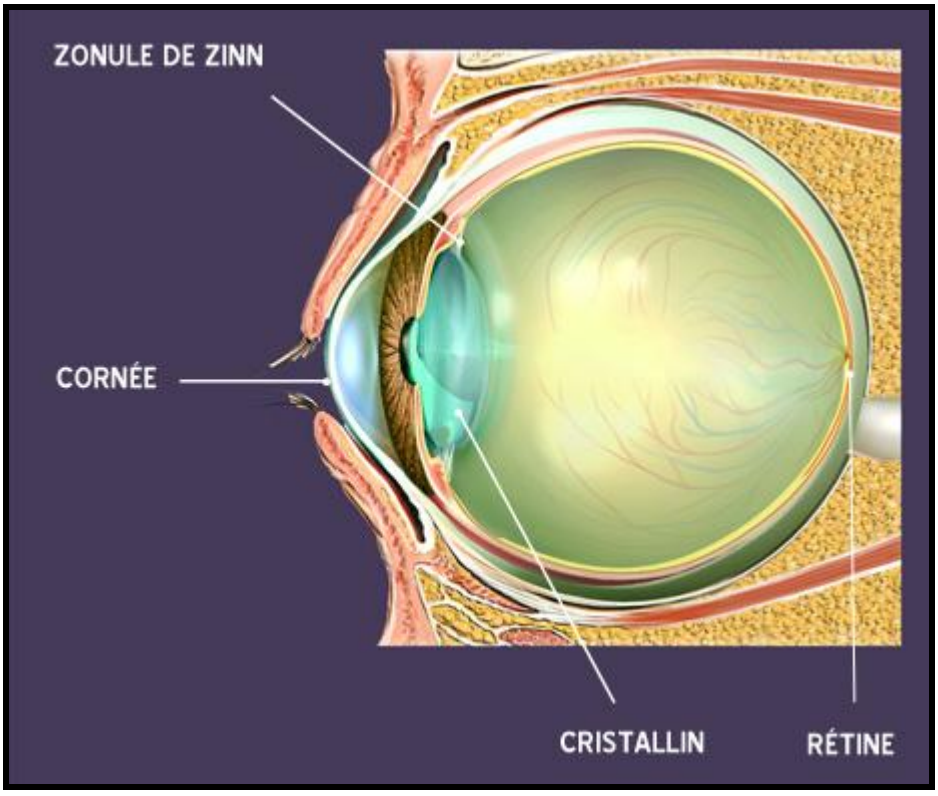



**Figure 2** : schéma cataracte

**Définition de la cataracte**

La cataracte est une perte totale ou partielle de la transparence du cristallin. Il devient plus ou moins opaque, la cataracte s'accompagne alors d'une diminution de l'acuité visuelle et/ou de la qualité visuelle : votre vue est moins nette, moins confortable.

Imaginez un verre de lunettes transparent, qui avec le temps deviendrait opaque, rendant ainsi votre vision trouble. Le phénomène de cataracte peut être comparé à cette illustration, le cristallin jouant la fonction du verre de lunettes.

La cataracte progresse en général lentement dans le temps.

Après un suivi médical régulier, l'ophtalmologiste décidera du moment opportun pour opérer, en fonction de l'évolution de l'opacification mais aussi de votre gène visuelle.

Le traitement sera chirurgical et consistera à substituer le cristallin opaque par un implant artificiel parfaitement transparent.

Quel organe de l'œil est concerné ?

Le cristallin, qui est une lentille « bi convexe » et normalement transparente. (Voir schéma ci-dessous)

Le rôle du cristallin est très important dans le mécanisme de notre système visuel. Ses fonctions sont multiples :

• il assure la mise au point pour voir net à toutes les distances, ce mécanisme appelé "accommodation" est réalisé grâce à une modification de la courbure du cristallin par tension ou relâchement de la zonule (notez que l'apparition de la presbytie limite la tonicité de ce processus d'accommodation)

• il représente un tiers de la puissance totale de l'œil, la cornée fournissant les deux tiers, et permet de focaliser les rayons lumineux sur la rétine

• il absorbe une partie des UV et protège ainsi la rétine, tissu qui tapisse le fond de l'œil

Le cristallin est constitué d'une capsule appelée cristalloïde ou capsule cristallinienne**. (figure3).**

### 1.3.2. Qu'est-ce qu'un glaucome ?

## Le glaucome

Le glaucome est une maladie oculaire qui altère le nerf optique, le plus souvent à cause d'une hypertension oculaire, il en résulte une pénalisation du champ visuel de la personne.

La pression intraoculaire est régie par l'humeur aqueuse, liquide qui remplit l'œil entre la cornée et le cristallin. L'humeur aqueuse est normalement évacuée au travers d'un filtre appelé trabéculum.

Lorsque le trabéculum qui fait donc office de filtre se bouche (1 du schéma), la bonne circulation de

l'humeur aqueuse est perturbée, cela provoque une augmentation de la pression intraoculaire **[2]** qui va elle-même altérer le nerf optique. La fonction du nerf optique est essentielle car il véhicule les informations visuelles vers le cerveau, son altération provoque donc des pertes de vision dans le champ visuel**. (figure4)**

Le glaucome est une maladie insidieuse car en dehors du **glaucome aigu** - le moins fréquent - aucune douleur n'est ressentie. Lorsque les symptômes visuels sont perçus, la maladie est déjà bien avancée, c'est pourquoi il est indispensable de la dépister lors d'examens réguliers chez un ophtalmologiste. De plus, c'est la périphérie du champ visuel qui est concernée ce qui rend les premiers symptômes visuels difficiles à réaliser par la personne concernée.

L'ophtalmologiste dépiste le glaucome en mesurant d'une part la pression intraoculaire, c'est-à-dire la pression à l'intérieur de votre œil qui est indépendante de la pression artérielle, d'autre part en réalisant un examen appelé « fond d'œil » qui permet de vérifier l'état de la tête du nerf optique située au fond de l'œil, aussi appelée la **papille optique**.

Il arrive qu'un glaucome soit diagnostiqué alors que la pression de l'œil est normale : c'est « le glaucome à pression normale ». Si la tête du nerf optique est plus excavée que « la norme », il y a suspicion de glaucome. Bien entendu, il peut arriver que la papille présente une excavation constitutionnelle, innée, sans signification pathologique**. (Figure5, Figure6)**

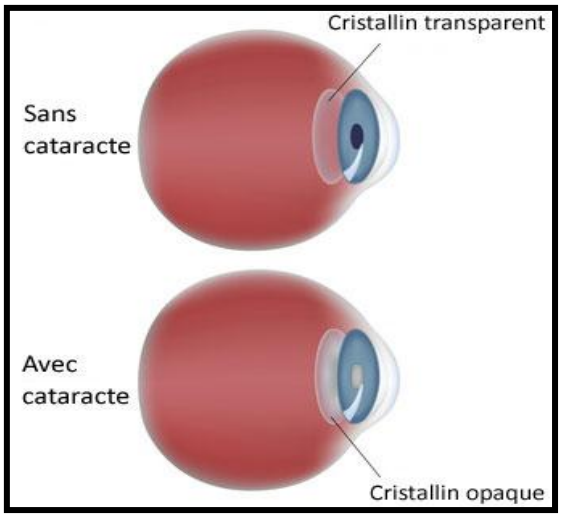



**Figure 3** : Comparaison cataracte

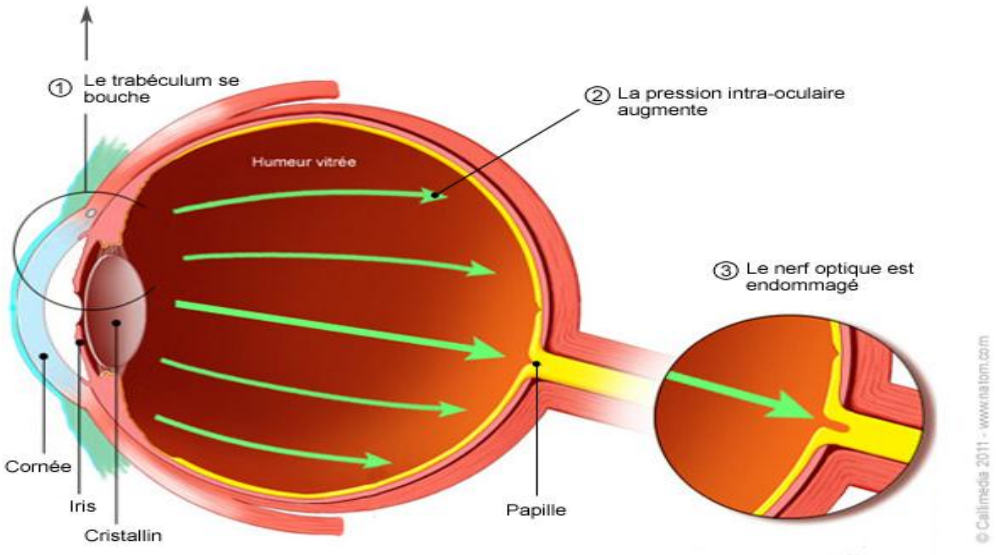



**Figure 4** : illustration glaucome


**Figure 5** : Champ de vision avec un glaucome

**Figure 6** : Champ de vision sans un glaucome

### 1.3.3. DMLA : menace sur la vision

La DMLA, Dégénérescence Maculaire Liée à l'Âge, est une maladie qui condamne les récepteurs visuels appelés « cônes » qui permettent la précision de vision, la vision des couleurs... C'est une « mort » prématurée de ces photorécepteurs de la rétine qui tapisse le fond de l'œil.

Cette altération du tissu de la rétine provoque à plus ou moins long terme un retentissement sur l'activité cellulaire de la zone maculaire. La macula est le centre de la rétine, elle permet de bien voir le centre du champ visuel : fixer du regard, observer... Sa richesse de cellules visuelles et leurs fonctions s'en trouvent dégradées.

L'impact sur la qualité de vision est certain, de niveau plus ou moins important suivant le degré de développement de l'affection, et plus ou moins rapidement en fonction de la forme de la maladie.

Le centre du champ de vision est menacé, ce qui est très pénalisant dans la vie quotidienne puisque le champ de vision central permet l'observation : lire, regarder son interlocuteur, se déplacer... La périphérie du champ de vision n'est généralement pas endommagée.

Il existe deux formes de DMLA : la DMLA "sèche" et la DMLA "humide". Lors de l'observation de l'état de la macula, la forme sèche se caractérise par l'apparition de petits amas blanchâtres appelés "drusen" alors que la forme humide fera apparaître des zones d'hémorragies. (Figure8)

**La DMLA "sèche"**

C'est la forme de DMLA la plus répandue.

Elle se caractérise par la diminution progressive du nombre de cellules visuelles, un appauvrissement par mort prématurée du nombre de récepteurs, les « cônes », permettant la précision de vision. Son évolution est lente, c'est-à-dire sur plusieurs années.

L'ophtalmologiste constatera, par l'observation du fond de l'œil, des amas blanchâtres au centre de la rétine, ce sont les drusen, appelés aussi druses.

Une prise en charge médicale précoce permet de freiner l'évolution de la maladie et ainsi de préserver le champ de vision jusqu'alors épargné.

**La DMLA "exsudative" dite "humide"**

C'est la forme la plus grave qui nécessite une prise en charge médicale urgente. La macula est envahie de petits vaisseaux sanguins anormaux qui se développent de façon anarchique.

Ces néo-vaisseaux sont issus de la choroïde, l'une des couches de la rétine qui permet d'alimenter l'iris et les photorécepteurs de la rétine. La DMLA humide se caractérise par une évolution brutale, souvent en quelques jours seulement. Elle est précédée de métamorphopsies, c'est-à-dire de déformations de la vision, en l'occurrence ces déformations se traduisent par une perception visuelle

ondulée des lignes droites (voir l'illustration dans les symptômes de la DMLA). (2)

Conjonctivites et autres infections :

Les infections de l'œil sont courantes. Si certaines sont bénignes comme les conjonctivites, d'autres peuvent cependant entraîner des séquelles. Mais les ophtalmologistes disposent aujourd'hui de nombreux médicaments pour venir à bout de ces maladies.

**Qu'est-ce que la rétinopathie diabétique ?**

**La rétinopathie diabétique**

C'est une maladie qui atteint la rétine des sujets diabétiques. La rétine est un tissu nerveux qui tapisse le fond de l'œil, elle reçoit la lumière puis la transmet au cerveau par l'intermédiaire du nerf optique, c'est ce mécanisme qui déclenche la vision. Elle est « nourrie » par le système sanguin. Artères et veines sont donc réparties sur la rétine.

Le diabète va abîmer les veines et créer des anomalies au niveau de la rétine qui provoqueront une baisse de la vue :

- des hémorragies : écoulement de sang hors du vaisseau sanguin
- des anévrismes : dilatation d'un vaisseau sanguin
- des ramifications de néovaisseaux : création de vaisseaux anormaux fragiles qui provoquent des hémorragies
- d'autres anomalies au niveau de la rétine

La rétinopathie diabétique peut être à l'origine d'autres complications comme le décollement de la rétine ou les hémorragies intra-vitréennes (le vitré est le liquide situé entre le cristallin et la rétine).

La rétinopathie diabétique sera d'autant plus avancée que le diabète sera ancien. **(Figure9).**

-Des pigments dans la rétine

-Depuis l'identification du premier gène impliqué dans la rétinite pigmentaire en 1984, plus de cent gènes ont été identifié. C'est dire si la recherche progresse… Cependant, aucun traitement n'a encore été trouvé pour freiner cette dégénérescence des cellules de la rétine responsable de cécité.

-Kératites, ulcères, abcès de cornée... Ne les laissez pas s'aggraver

La cornée, cette fine membrane transparente qui recouvre la pupille et l'iris, est fragile. Ses blessures, qu'elles soient superficielles (kératites) ou plus profondes (ulcères), font courir un risque infectieux (abcès cornéens) et/ou de mauvaise cicatrisation, engageant le pronostic visuel.

Zona ophtalmique

Affection assez rare mais potentiellement grave, le zona ophtalmique nécessite d'être pris en charge rapidement. Quels en sont les symptômes ? Comment le traiter ? Explications des Drs Timsit et Arnavielle, tous deux médecins ophtalmologues à Paris. **[3]**

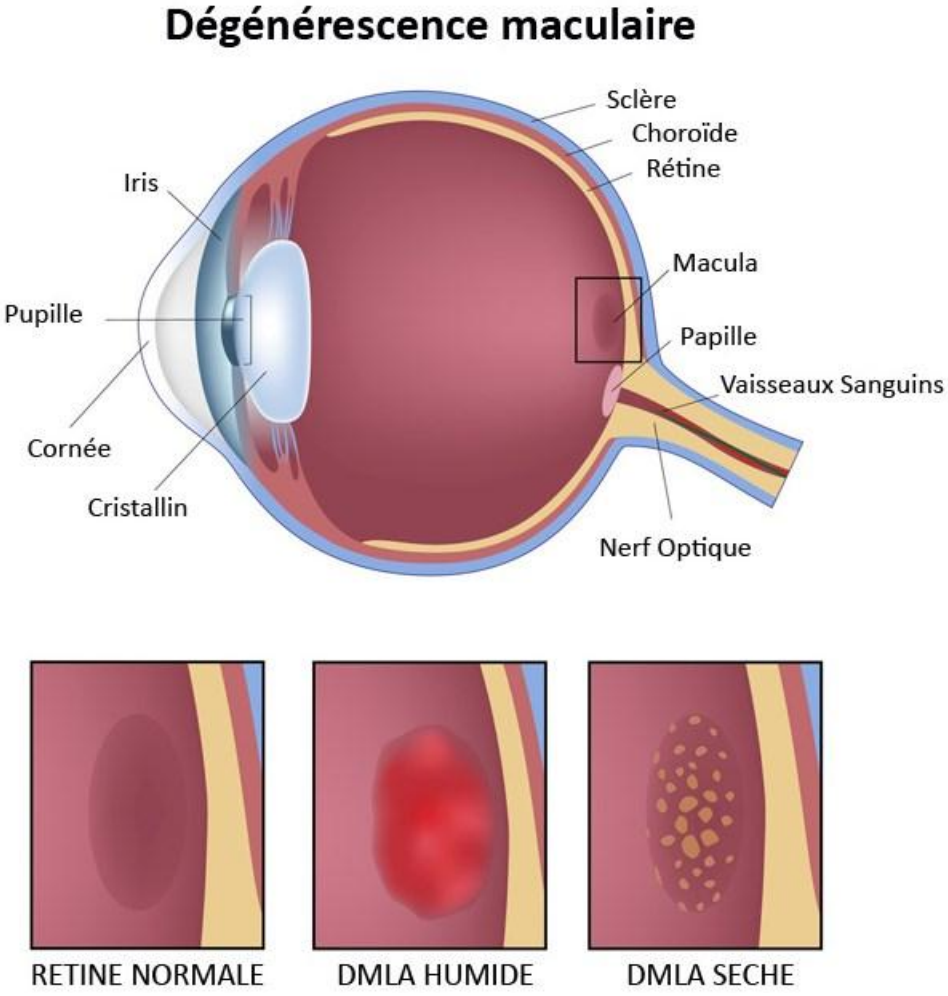



**Figure 8** : schéma DLMA

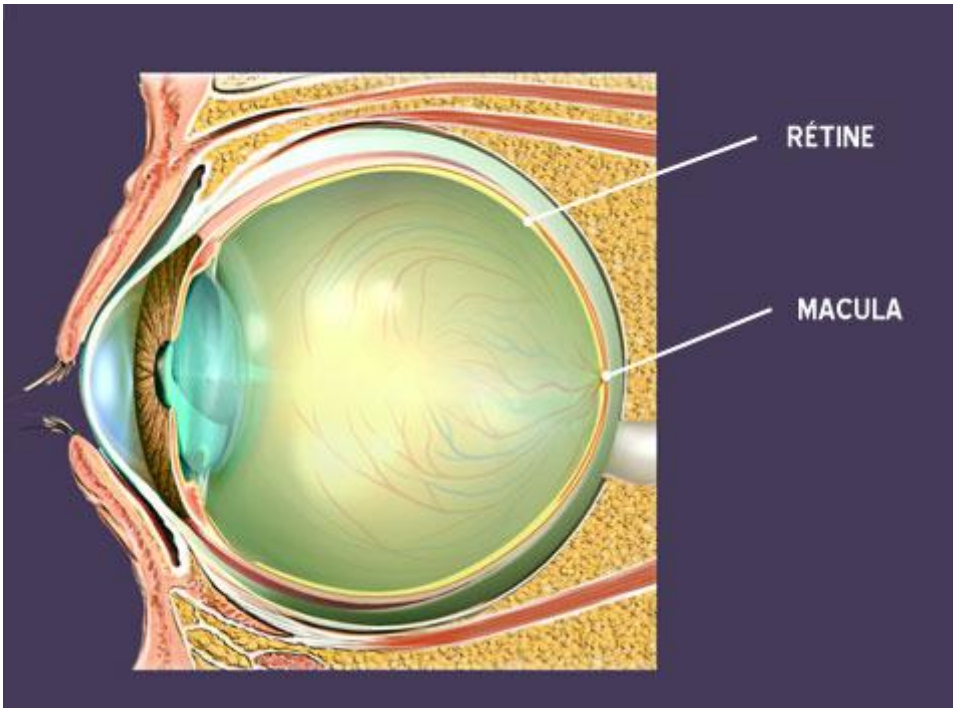



**Figure 9**: schéma rétinopathie

# II. Télémédecine

### 1. Définition de la télémédecine

La télémédecine est une forme de pratique médicale à distance mettant en rapport, entre eux ou avec un patient, un ou plusieurs professionnels de santé, parmi lesquels figure nécessairement un professionnel médical et, le cas échéant, d'autres professionnels apportant leurs soins au patient.

### 2. Objectifs de la télémédecine

- Améliorer la couverture médicale des usagers du système de santé et le suivi des malades souffrant de maladies chroniques ou de handicaps ;
- Assurer une continuité des soins à domicile ;
- Prévenir les complications et limiter ainsi les hospitalisations.

### 3. Objet de la télémédecine

- Etablir un diagnostic,
- Assurer, pour un patient à risque, un suivi à visée préventive ou un suivi post-thérapeutique,
- Requérir un avis spécialisé,
- Préparer une décision thérapeutique,
- Prescrire des produits,
- Prescrire ou réaliser des prestations ou des actes,
- Ou effectuer une surveillance de l'état des patients.

### 4. Les types d'intervention à distance par les professionnels médicaux

#### 4.1.La téléconsultation

Consultation à distance par un professionnel médical. Un professionnel de santé ou un psychologue peut être présent auprès du patient et, le cas échéant, assister le professionnel médical au cours de la téléconsultation.

#### 4.2.La télé expertise

Solliciter à distance l'avis d'un ou de plusieurs professionnels médicaux en raison de leurs formations ou de leurs compétences particulières, sur la base des informations médicales liées à la prise en charge d'un patient.

#### 4.3.La télésurveillance médicale

Interpréter à distance les données nécessaires au suivi médical d'un patient et, le cas échéant,

prendre des décisions relatives à sa prise en charge.

**4.4.La téléassistance médicale**

Assister à distance un autre professionnel de santé au cours de la réalisation d'un acte.

**4.5.La réponse médicale**

Apportée dans le cadre la régulation médicale des permanences des soins et d'aide médicale urgente.

**5**. **Modalités de mise en œuvre de la télémédecine**

- Recueillir le consentement libre et éclairer du patient ;
- Avoir la possibilité d'échanger des informations relatives au patient dûment informé, sauf opposition de sa part ;
- Garantir obligatoirement l'authentification des professionnels de santé intervenant, l'identification du patient, l'accès des professionnels aux données médicales nécessaires à la réalisation de l'acte, et, le cas échéant, la formation ou la préparation du patient à l'utilisation du dispositif de télémédecine ;
- Inscrire dans le dossier du patient et dans la fiche d'observation :
- le compte rendu de la réalisation de l'acte ;
- les actes et les prescriptions médicamenteuses effectués dans le cadre de l'acte de télémédecine ;
- l'identité des professionnels de santé participant à l'acte ;

  La date et l'heure de l'acte ;
- et, le cas échéant, les incidents techniques survenus au cours de l'acte. **[4]**

Etude et élaboration d'une plate-forme télé-ophtalmologique

# Chapitre II : Capture de l'image

# I. Explications sur les différents ports

## 1. Port parallèle

Lorsqu'IBM a lancé son Personale Computer, il lui fallait un connecteur pour se brancher sur une imprimante. A l'époque, le leader des imprimantes était Centronics et IBM voulait que ses PC fonctionnent avec ces imprimantes. Mais, au lieu de "simplement" utiliser la même interface (36 connecteurs), IBM a conçu son propre port à 25 connecteurs (DB-25). Ainsi, on a ces câbles "imprimante", "hybrides" entre un Centronics du côté imprimante et un DB-25 du côté PC.

Dans une communication parallèle, les données sont transférées de l'émetteur au receveur par groupe de 8 bits, à travers 8 lignes (ou circuits). La transmission va ainsi très vite (beaucoup plus vite qu'en série) mais les prises sont plus encombrantes et, du fait que le protocole de communication parallèle a été étendu (voir EPP et ECP ci-dessous), les périphériques peuvent contenir plus d'électronique (i.e. être plus chers) pour implémenter ce protocole.

### 1.1. Principe de communication :

Le standard IEEE 1284 (1994) définit 5 modes d'opération pour un port parallèle :

**1.**Le mode compatible (Compatibility Mode, SPP)

**2.**Le mode Nibble (Nibble Mode)

**3.**Le mode Byte (Byte Mode)

**4.**Le mode Port Parallèle Amélioré (EPP)

**5.**Le mode Port aux Capacités Etendues (ECP)

#### 1.1.1. Port Parallèle Standard (SPP)

Le protocole standard (SPP, Standard Parallèle Port) a été défini comme tel par après, lors de la définition de l'EPP (voir ci-dessous).

##### 1.1.1.1. Mode Compatible

En SPP, lorsque le PC envoie des données par le port parallèle (mode compatible : du PC vers le périphérique), il envoie 8 bits de données à la fois (8 bits transmis en parallèle).

La vitesse élevée est également obtenue par le fait que les données circulent en parallèle dans le bus du PC et qu'elles ne doivent donc pas être réarrangées pour être envoyée en série.

Comme le port parallèle a été conçu pour communiquer avec une imprimante, voyons ce qui se passe quand on veut imprimer.

L'ordinateur sait que l'imprimante est allumée parce qu'il reçoit une tension constante de 5 volts sur le connecteur 13 (online). Quand le PC veut commencer une impression, il initialise l'imprimante en diminuant la tension du connecteur 16 (Initialize). Le PC demande à l'imprimante de charger du papier en envoyant une tension (5 volts) sur le connecteur 14 (auto-feed).

Pour envoyer des données, le PC écrit ces données via les connecteurs 2 à 9 (un "1" met la tension à

5 volts ; un "0" met la tension à ... 0 volts !). Le PC vérifie si l'imprimante est occupée (connecteur 11). Si l'imprimante est prête, l'ordinateur diminue alors la tension à 0.5 volts sur le connecteur 1 (strobe, sinon la tension dans ce connecteur est maintenue entre 2.8 et 5 volts). Cela donne le signal à l'imprimante pour collecter les données sur les connecteurs 2 à 9. Le PC remonte tout de suite (après environ 5 microsecondes) la tension sur le connecteur 1 à 5 volts.

Le PC sait que les données ont été reçues car l'imprimante diminue la tension sous les 0.5 volts dans le connecteur 10 (acknowledge, normalement maintenu à 5 volts, comme le connecteur 1) lorsque les données ont été effectivement reçues.

Plusieurs problèmes peuvent se produire, certains d'entre eux sont indiqués au niveau des connecteurs. Les connecteurs restant servent pour la terre (voir connecteurs et assignation ci-dessous).

#### 1.1.1.2. Mode Unidirectionnel

Au début, les données ne pouvaient aller que dans un seul sens : du PC vers le périphérique. On dit que le protocole est unidirectionnel. Mais, très vite, le protocole a ajouté la possibilité d'envoyer les données dans les deux sens (PC <-> périphérique) : il est devenu bidirectionnel. Afin d'envoyer des données du périphérique vers le PC, il faut changer de mode (pour rappel, du PC vers le périphérique, c'est le mode compatible).

### 1.1.1.3. Modes Bidirectionnels : Nibble et Byte

Au début du bidirectionnel, les données allaient dans les 2 sens mais sur les mêmes connecteurs (2 à 9). La communication ne pouvait se faire que dans un seul sens à la fois (half-duplex). Pour envoyer des données vers le PC en half-duplex, le protocole change pour le mode Nibble, où il envoie un nibble (4 bits) en sens inverse.

Par la suite, on a utilisé les connecteurs 18 à 25 comme second canal de communication. Ainsi, on pouvait envoyer des données dans un sens sur les connecteurs 2 à 9 et, en même temps, dans l'autre sens, sur les connecteurs 18 à 25 (communication full-duplex). Pour envoyer des données vers le PC en full-duplex, le protocole change pour le mode Byte, où il envoie un byte (8 bits) en sens inverse.

### 1.1.1.4. Vitesse

En SPP, le port peut envoyer de 50 à 100 kilo bytes de données par seconde.

## 1.1.2. Port Parallèle Amélioré (EPP)

Devant les limitations du SPP, Intel, Xircom et Zenith ont développé l'EPP (Enhanced Parallel Port, 1991). La principale amélioration a été le débit : on est maintenant à 500 kbytes à 2 Mb par seconde. La cible est principalement les périphériques autres que les imprimantes, comme les

périphériques de stockage qui nécessitent des taux de transfert les plus haut possible.

EPP augmente sa vitesse en laissant au matériel le soin de vérifier si l'imprimante est occupée et de générer le strobe adéquat ou le handshaking (accordage des paramètres entre les 2 machines). Ainsi, en sortant ces étapes du protocole, une seule instruction a besoin d'être envoyée.

Le protocole EPP est compatible avec le protocole standard (backwards compatible) : on pourra utiliser un périphérique ne comprenant que le protocole standard en lui donnant une partie des instructions du protocole EPP.

### 1.1.3. Port aux Capacités Etendues (ECP)

Pas encore contents du EPP, Microsoft et Hewlett Packard ont créé l'ECP (Extended Capabilities Port, 1992) pour donner encore plus de vitesse et de fonctionnalités aux imprimantes, cette fois. ECP utilise deux modes de communication : un mode bi-directionnel rapide et un mode bi-directionnel encore plus rapide (en compressant les données). Le protocole ECP est compatible avec le protocole standard (backwards compatible).

La vitesse est augmentée de plusieurs manières :

- en prenant avantage des canaux DMA (accès direct à la mémoire) et des tamponsFIFO.
- en compressant les données à transmettre avec la méthode RLE (Run Length Encoding)

La compression RLE est un système de compression simple des données, au niveau des bytes. Elle va compresser de manière efficace les longues séquences du même byte en utilisant un code à 2 bytes : le byte répété et un byte pour le nombre de répétition. Ces 2 bytes sont transmis via le port parallèle. Le nombre de répétition est de maximum 128 bytes (pour que ce nombre puisse tenir dans le byte transmis). Cela signifie que la compression maximum est de 64:1. Cette méthode est bonne pour les images qui contiennent souvent de longues séries des mêmes bytes mais cette méthode est assez mauvaise pour un texte.

De plus, un seul port ECP peut accepter plusieur périphériques (maximum 128). Pour accomplir cette tâche, le port utilise son propre schéma d'adressage, il envoie la commande d'adresse de canal (channel address command) sur les connecteurs de données. En faisant cela, le port dit à tous les périphériques sauf celui à qui est destiné les données d'ignorer les données qui vont suivre. Si aucune commande d'adresse de canal n'est envoyée avant les données, celles-ci sont destinées au périphérique à l'adresse 0, par défaut.

### 1.1.4. Et comment on choisit son mode ?

Les modes SPP, EPP et ECP ont été officialisé dans une norme IEEE (1284, en 1994). Ainsi, pour fonctionner avec une norme, autant l'ordinateur que le périphérique doivent supporter les spécifications de cette norme. Ça paraît évident mais, avant, tous les systèmes d'exploitation ne le permettaient pas. Maintenant, tous les systèmes détectent quel mode doit être utilisé. Ainsi, si on veut spécifiquement choisir un mode, on peut le faire aussi via le BIOS.**[6]**

### 1.2 Matériel :

Derrière les ordinateurs, le port parallèle contient 25 connecteurs.

Ce qui suit est l'assignation SPP :

| *db25* | | | *centronic 36 pin* | |
|---|---|---|---|---|
| **connecteur** | **nom du signal** | | **connecteur** | **nom du signal** |
| 1 | strobe | - | 1 | strobe |
| 2 | data 0 | - | 2 | data 0 |
| 3 | data 1 | - | 3 | data 1 |
| 4 | data 2 | - | 4 | data 2 |
| 5 | data 3 | - | 5 | data 3 |
| 6 | data 4 | - | 6 | data 4 |
| 7 | data 5 | - | 7 | data 5 |
| 8 | data 6 | - | 8 | data 6 |
| 9 | data 7 | - | 9 | data 7 |
| 10 | acknowledge | - | 10 | acknowledge |
| 11 | busy | - | 11 | busy |
| 12 | paper end | - | 12 | paper end |
| 13 | select | - | 13 | select |
| 14 | auto feed | - | 14 | auto feed |
| 15 | error | - | 15 | nc |
| 16 | init | - | 16 | nc |
| 17 | select in | - | 17 | nc |
| 18 | ground | - | 18 | nc |
| 19 | ground | - | 19 | ground |
| 20 | ground | - | 20 | ground |
| 21 | ground | - | 21 | ground |
| 22 | ground | - | 22 | ground |
| 23 | ground | - | 23 | ground |
| 24 | ground | - | 24 | ground |
| 25 | ground | - | 25 | ground |
| 26 | ground | | 26 | ground |
| 27 | ground | | 27 | ground |
| 28 | ground | | 28 | ground |
| 29 | ground | | 29 | ground |
| 30 | ground | | 30 | ground |
| 31 | init | | 31 | init |
| 32 | error | | 32 | error |

| | | | | |
|---|---|---|---|---|
| 33 | ground | | 33 | ground |
| 34 | nc | | 34 | nc |
| 35 | nc | | 35 | nc |
| 36 | select in | | 36 | select in |

Outre les connecteurs déjà cités (voir port parallèle standard ci-dessus), voici la signification des quelques connecteurs qui restent :

Connecteur 11 (busy) : si l'imprimante est occupée, elle va mettre 5 volts sur ce connecteurs. Quand elle sera prête, elle va diminuer la tension à 0.5 volts connecteur 12 (out of paper) : s'il n'y a plus de papier dans l'imprimante, elle enverra 5 volts sur ce connecteur (aussi longtemps qu'il n'y a plus de papier) connecteur 15 (problems) : tout problème à l'imprimante diminue la tension à 0.5 volts, pour indiquer à l'ordinateur que quelque chose ne va pas connecteur 17 (offline) : tant que le PC enverra 5 volts sur ce connecteur, l'imprimante sera éteinte (à distance, donc)

Connecteurs 18 à 25 sont reliés à la terre (cas communication unidirectionnelle) et servent de références pour la basse tension (0.5 volts) ; dans le cas des communications bidirectionnelles, ces connecteurs servent à envoyer des données

En EPP, les connecteurs diffèrent peu :

| connecteur | nom du signal | connecteur | nom du signal | connecteur | nom du signal |
|---|---|---|---|---|---|
| 1 | write | 10 | interrupt | 19 | Ground |
| 2 | data 1 | 11 | wait | 20 | Ground |
| 3 | data 2 | 12 | spare | 21 | Ground |
| 4 | data 3 | 13 | spare | 22 | Ground |
| 5 | data 4 | 14 | data strobe | 23 | Ground |
| 6 | data 5 | 15 | spare | 24 | Ground |
| 7 | data 6 | 16 | reset | 25 | Ground |
| 8 | data 7 | 17 | address strobe | | |
| 9 | data 8 | 18 | ground | | |

En ECP, par contre, les connecteurs changent de nom (et donc de fonction) :

| connecteur | nom du signal | connecteur | nom du signal | connecteur | nom du signal |
|---|---|---|---|---|---|
| 1 | HostCLK | 10 | PeriphCLK | 19 | ground |
| 2 | data 1 | 11 | PeriphACK | 20 | ground |
| 3 | data 2 | 12 | nAckReverse | 21 | ground |
| 4 | data 3 | 13 | X-Flag | 22 | ground |
| 5 | data 4 | 14 | Host Ack | 23 | ground |
| 6 | data 5 | 15 | PeriphRequest | 24 | ground |

| | | | | | |
|---|---|---|---|---|---|
| 7 | data 6 | 16 | nReverseRequest | 25 | ground |
| 8 | data 7 | 17 | 1284 Active | | |
| 9 | data 8 | 18 | ground | | |

### 1.3. Jouons avec notre PC

#### 1.3.1. Avec MS-Windows

Sous MS-Windows, les ports parallèles sont nommés LPT1, LPT2, LPT3, etc.

Pour découvrir les ports parallèle qu'on a sous MS-Windows, il faut aller dans le panneau de contrôle, cliquer sur l'icône "Système" et y trouver le "Gestionnaire de périphériques". Dedans, il faut trouver l'entrée "Ports (COM & LPT) et l'étendre. Cela devrait donner à peu près ceci ...

Pour connaître (et éventuellement modifier) les paramètres d'un port parallèle, il suffit de cliquer sur son item.

#### 1.3.2. Avec GNU/Linux

Sous GNU/Linux, les ports parallèles font partie du répertoire /dev et sont nommés

lp0, lp1, lp3, etc. **[7]**

## 2. Port série

### 2.1 Introduction

Le port série supporte le standard RS-232, créé (1960) avec un seul objectif : permettre la communication entre un terminal (Data Terminal Equipment, DTE ; en clair : un ordinateur) et un appareil communiquant (Data Communications Equipment, DCE ; en clair : un modem, par exemple).

Dans une communication série, les données sont transférées de l'émetteur au receveur un bit à la fois, à travers une seule ligne (ou circuit). Les avantages est le faible coût et le faible encombrement. Le désavantage est que la transmission va moins vite que si on transmettait un certain nombre de bits en même temps (voir port parallèle).

### 2.2. Principe de communication

La communication se fait bit à bit : chaque bit d'information est transféré de manière séquentielle (sérielle) d'un endroit à un autre. Le port série prend 8, 16 ou 32 bits "parallèles" du bus du PC et les convertit en un flux séquentiel de 8, 16 ou 32 bits.

En théorie, on n'aurait alors besoin que de 2 fils : une ligne pour le signal et une pour la terre. Mais, en pratique, il y a des perturbations extérieures et des erreurs qui rendent ce besoin simple impossible. En effet, si 1 bit est envoyé mais manque du côté récepteur, tous les bits suivants sont

décalés, donnant des données incorrectes lorsque le flux est transformé du côté récepteur. Il faut donc des processus permettant de contrôler si une telle erreur est survenue.

### 2.2.1. Méthodes de transmission

Deux méthodes différentes sont utilisées :

5. la communication synchrone ;
6. la communication asynchrone.

#### 2.2.1.1. Communication synchrone

Dans la communication synchrone, l'émetteur et le récepteur sont synchronisés par une horloge qui compte précisément la période séparant chaque bit. En contrôlant l'horloge, le récepteur peut déterminer si un bit a été perdu ou s'il y en a un en trop (habituellement, par induction électrique). Si jamais une des deux parties perd le signal d'horloge, la communication est terminée car rendue impossible.

Cette méthode est peu utilisée en informatique généraliste.

#### 2.2.1.2. Communication asynchrone

Dans la communication asynchrone, un "marqueur" est ajouté au-début du flux de bits pour aider à positionner les bits dans le flux. Lorsque le récepteur reçoit le bit de début (start bit, toujours égale à 0), pour peu que les deux ports soient à la même vitesse, le récepteur déclenche un timer et récolte les bits de données dans un intervalle de temps donné. En gardant les flux non contrôlés (bits de données après le start bit) court, on restreint la possibilité d'erreur.

Cette méthode est la plus souvent employée en informatique généraliste (c'est celle des modems, par exemple).

### 2.2.2. Les bits, leur parité et leur vitesse

Chaque flux de bits est composé de 5 à 8 bits, appelés mots (words). Habituellement, dans un environnement PC, on trouvera des mots de 7 ou 8 bits (7 bits pour les 127 premiers codes ASCII, 8 bits pour correspondre exactement à un byte).

Lors d'une communication, l'émetteur encode chaque mot en ajoutant un bit de départ au début et 1 ou 2 bits de fin (stop bits) à la fin du flux. Parfois, afin de vérifier l'intégrité des données, l'émetteur ajoutera un bit de parité entre le dernier bit d'un mot et le bit de départ du mot suivant. On parlera ainsi de fenêtre de données (data frame).

| Bit | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Contenu | départ | bit 1, mot 1 | bit 2, mot 1 | bit 3, mot 1 | bit 4, mot 1 | bit 5, mot 1 | bit 6, mot 1 | bit 7, mot 1 | bit 8, mot 1 | stop | parité | départ | bit 1, mot 2 | etc. |

**Tableau1**. Exemple de fenêtre de donnée sur 8 bits, avec bits de départ, d'arrêt et de parité (8 bits data frame)

5 bits de parité différents peuvent être utilisés :

--le bit de parité de marquage (mark parity bit) : toujours égal à 1 (de type logique)

--le bit de parité d'espace (space parity bit) : toujours égale à 0 (de type logique)

--le bit de parité paire (even parity bit) : égal à 1 lorque le nombre de bits dans le mot est pair

--le bit de parité impaire (odd parity bit) : égal à 1 lorsque le nombre de bits dans le mot est impair

--pas de bit de parité (non parity bit frame) : cela permet de gagner de la place (1 bit par fenêtre) et est rendu possible car le bit de parité n'est pas obligatoire

La vitesse de transmission est un point très important dans la communication asynchrone. Cette vitesse influence directement les moments auxquels le récepteur s'attend à trouver un bit de donnée (voir la figure ci-dessus : ligne d'horloge). De nos jours, la plus petite vitesse couramment utilisée est le 300 bps (bit par seconde). On trouve encore des vitesses de 600, 1200, 2400, 4800 bps mais, le plus souvent, on utilise 9600, 19200 ou 38400 bps. Seul, le processeur ne peut aller que jusque 19200 bps (19.2 kbps). Mais, avec les bus PCI et, surtout le DMA (Direct Memory Access, accès direct à la mémoire), on peut atteindre les 38400 bps (38.4 kbps).

Notez qu'on voit parfois la notion de baud en communication série. La fréquence de baud est une mesure du nombre de fois par seconde qu'un signal varie (dans un canal de communication), ou effectue une transition entre états (càd. fréquences ou niveaux de voltage, ...). Un baud est un de ces changements. Donc, le signal d'un modem à 300 bauds change 300 fois d'étant par seconde ; le signal d'un modem à 600 bauds change d'état 600 fois par seconde. Mais cela ne veut pas dire qu'un modem à 300 ou 600 bauds transmet respectivement 300 ou 600 bits par seconde (300 bauds/s != 300 bps).

En effet, un modem peut transmettre 1 bit avec chaque baud (ou changement d'état) ou +/- 1 bit par baud. Cela dépend de la technique de modulation utilisée. Ainsi, pour un modem qui transmet 1 bit par baud, on aura effectivement 300 bauds/s = 300 bps. Mais, pour un modem qui transmet 1,5 bit par baud, on aura 300 bauds/s = 450 bps !

### 2.3. Matériel

La plupart des équipements utilisent un port série à 25 connecteurs (DB-25). Derrière les ordinateurs, le port série contient généralement seulement 9 connecteurs (les seuls nécessaires en mode asynchrone, voir ci-dessus).

#### 2.3.1. Connecteur et assignation

Le rôle de chaque connecteur était, au début, spécifié pour l'utilisation de modems. C'est pourquoi

la nomenclature suit de près cet usage.

Voici l'assignation des 25 connecteurs d'une prise DB25 :

1.Pas utilisé (terre)

2.Transmission de données (Transmit Data) : l'ordinateur envoie des données au périphérique

3.Réception de données (Receive Data) : l'ordinateur reçoit des données du périphérique

4.Demande d'envoi (Request To Send, RTS) : l'ordinateur demande au périphérique s'il peut envoyer de l'information

5.Prêt à envoyer (Clear To Send, CTS) : le périphérique dit à l'ordinateur qu'il est prêt à envoyer des données

6.Prêt pour données (Data Set Ready, DSR) : le périphérique dit à l'ordinateur qu'il est prêt à parler

7.Terre (Signal Ground) : terre

8.Détection de signal sur la ligne (Received Line Signal Detector) : détermine si le modem est connecté à une ligne téléphonique qui fonctionne

9.Pas utilisé : transmet le retour de courant de boucle (+)

10.Pas utilisé

11.Pas utilisé : transmet le courant de boucle des données (-)

12.Pas utilisé

13.Pas utilisé

14.Pas utilisé

15.Pas utilisé (parfois timing du signal transmis pour le DCE)

16.Pas utilisé

17.Pas utilisé

18.Pas utilisé : reçoit le courant de boucle des données (+)

19.Pas utilisé

20.Terminal prêt (Data Terminal Ready, DTR) : l'ordinateur dit au périphérique qu'il est prêt pour parler

21.Pas utilisé (parfois détection de la qualité du signal)

22.Indicateur de sonnerie (Ring Indicator) : une fois que l'appel a été effectué, l'ordinateur reconnaît le signal (envoyé par le modem) qu'une sonnerie est détectée

23.Pas utilisé (parfois détection de la vitesse du signal)

24.Pas utilisé (parfois timing du signal transmis pour le DTE)

25.Pas utilisé : reçoit le retour de courant de boucle (-)

Comme on n'utilise pas les connecteurs 1, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 23, 24 et 25 (16 connecteurs inutilisés), il y en a 9 seulement de réellement utiles (connecteurs 2, 3, 4, 5, 6, 7, 8,

20, 22). On a alors fabriqué des ports qui n'ont que ces 9 connecteurs (sur tous les PC récents).

Voici l'assignation des 9 connecteurs d'une prise DB9 :

1.Détection de signal sur la ligne (Received Line Signal Detector) : détermine si le modem est connecté à une ligne téléphonique qui fonctionne

2.Réception de données (Receive Data) : l'ordinateur reçoit des données du périphérique

3.Transmission de données (Transmit Data) : l'ordinateur envoie des données au périphérique

4.Terminal prêt (Data Terminal Ready, DTR) : l'ordinateur dit au périphérique qu'il est prêt pour parler

5.Terre (Signal Ground) : terre

6.Prêt pour données (Data Set Ready, DSR) : le périphérique dit à l'ordinateur qu'il est prêt à parler

7.Demande d'envoi (Request To Send, RTS) : l'ordinateur demande au périphérique s'il peut envoyer de l'information

8.Prêt à envoyer (Clear To Send, CTS) : le périphérique dit à l'ordinateur qu'il est prêt à envoyer des données

9.Indicateur de sonnerie (Ring Indicator) : une fois que l'appel a été effectué, l'ordinateur reconnaît le signal (envoyé par le modem) qu'une sonnerie est détectée

Le voltage envoyé peut être de deux types : on ou off. Un signal On (valeur binaire 1) équivaut à un voltage entre -3 et -25 volts. Un signal Off (valeur binaire 0) équivaut à un voltage entre +3 et +25 volts.

**2.3.2. Contrôleur matériel et contrôle de flux**

On a vu (ci-dessus) qu'avec un processeur seul, la vitesse est limitée à 19.2 kbps. Depuis quelques temps, la gestion du port série se fait en dehors du processeur, via un receveur/transmetteur asynchrone universel (Universal Asynchronous Receiver/Transmitter,UART).

Ce contrôleur matériel prend les bits "parallèles" sur le bus du PC et les ré-arrange en séquence pour les transmettre sur le port série. Pour fonctionner plus rapidement, la plupart des UART ont un tampon intégré (16 à 64 kb). Ainsi, les données entrantes (provenant du bus) peuvent être mises en cache en attendant d'être ré-arrangé. On arrive ainsi (déchargement du processeur et cache) à des vitesses de transfert de 115 kbps.

Outre les bits de parité (voir ci-dessus), il existe aussi un contrôle du flux de données. Par ce contrôle, un appareil est capable de dire à l'autre d'arrêter d'envoyer des données pour un moment, par l'utilisation des commandes Request to send (RTS), Clear to send(CTS), Data Terminal Ready (DTR) et Data Set Ready (DSR).

C'est particulièrement utile avec les modems ... Généralement, les modems communiquent entre eux à 56 kbps. Or, la communication entre le PC et le modem peut se faire à 115 kbps, ce qui est

plus que le double ! Comme le modem reçoit plus de données qu'il ne peut en transmettre, même avec une mémoire tampon, il ne pourra plus rapidement traiter toutes les données qui lui sont fournies.

Avec le contrôle de flux, le modem peut arrêter le flux provenant de l'ordinateur avant que les données ne saturent la mémoire tampon. L'ordinateur envoie constamment un signal RTS(demande d'envoi) et vérifie constamment s'il n'y a pas de signal CTS (prêt à envoyer). S'il n'y a pas de réponse CTS, l'ordinateur arrête d'envoyer des données au modem et attend jusqu'au prochain CTS. Ainsi, cela permet aux données de passer sans problème du PC au modem.

**2.3.3. A sens unique ou non**

Finalement, le câblage et la manière dont les ports sont arrangés sont important. Certains ports utilisent le même connecteur (pin) pour recevoir et émettre des données. Le récepteur doit donc attendre d'avoir tout reçu avant de pouvoir émettre à son tour. On parlera de port demi-duplex (half-duplex). Par contre, d'autres ports utilisent des connecteurs différents pour recevoir et émettre des données. Chaque appareil peut donc se comporter, en même temps, comme émetteur et comme receveur. On parlera de port full-duplex ou bidirectionnel. Un port bidirectionnel permettra de réduire le temps mis pour une tâche à s'accomplir. **[8]**

**2.4. Jouons avec notre PC**

**2.4.1. Avec MS-Windows**

Sous MS-Windows, les ports série sont nommés COM1, COM2, COM3, etc.

Pour découvrir les ports série qu'on a sous MS-Windows, il faut aller dans le panneau de contrôle, cliquer sur l'icône "Système" et y trouver le "Gestionnaire de périphériques". Dedans, il faut trouver l'entrée "Ports (COM & LPT) et l'étendre. Cela devrait donner à peu près ceci ...

Pour connaître (et éventuellement modifier) les paramètres d'un port série, il suffit de cliquer sur son item et de choisir l'onglet ... "Paramètres du port".

**2.4.2. Avec GNU/Linux**

Sous GNU/Linux, les ports série font partie du répertoire /dev et sont nommés ttys0,ttys1, ttys3, etc.

Il y a plusieurs manières de découvrir les ports série qu'on a sous GNU/Linux. En ligne de commande (ou dans une console), vous pouvez tapez la commande set serial -g /dev/ttys0 (par exemple). Cela devrait donner ce résultat ...

La commande set serial permet également de voir les paramètres d'un port série et de les modifier. Consultez la page ad hoc du manuel (man set serial) pour plus de renseignements.

En 1995, un groupe de constructeurs de PC ont élaboré la norme USB (Universel Serial Bus) 1.0 pour remplacer l'interface série qui montrait ses limites.

En effet, avec un port série, le principal désavantage est devenu sa lenteur (et le peu de bande

passante associé). Mais les périphériques étaient souvent associés à un port et en changer nous obligeait à tout reconfigurer. Et on ne pouvait pas non plus y attacher plus d'un appareil à la fois (sauf astuces). Etc.

Donc, l'USB a été créé pour résoudre ces problèmes et bien plus ...

- vitesse beaucoup plus élevée que les ports série ou parallèle (12 Mbits par seconde pour la norme 1.1 ; 480 Mbits par seconde pour la norme 2.0 "Hi-Speed") ;
- un périphérique USB peut se connecter sur n'importe quel port USB ;
- les périphériques peuvent prendre leur alimentation électrique dans le port USB même (maximum 0.5 ampère de puissance pour 5 volts) ;
- on peut connecter jusqu'à 127 périphériques sur un PC, soit directement, soir via un hub ;
- un hub joue le rôle de multiplexeur (connection de plusieurs périphériques à un même câble) mais aussi de répétiteur, d'amplificateur, de contrôleur du signal et de fournisseur de courant ;
- les câbles peuvent faire 5 mètres de long (30 mètres s'il y a un hub) ;
- support du hot-swap : on peut brancher et débrancher les périphériques quand on veut, l'ordinateur charge et décharge automatiquement les pilotes ;
- support des fonctions de mise en veille par l'ordinateur ;

**2.4.3. Principe de communication**

L'ordinateur joue le rôle de hôte (host).

**2.4.4. Enumération**

Lors de son démarrage, l'ordinateur interroge tous les périphériques USB connectés à son bus et leur assigne à chacun une adresse. Lorsqu'on connecte un périphérique USB à chaud (càd. alors que la machine est déjà allumée), le même processus se produit. Le PC détecte également à ce moment quel type de transfert de données est adéquat.

Si le périphérique n'a jamais été rencontré auparavant, le système d'exploitation le détecte et demande pour le pilote (ou le charge s'il a déjà le pilote dans ses fichiers). Si le pilote est déjà installé, l'ordinateur l'active et commence à dialoguer avec le périphérique.

**2.4.5. Types de transfert de données**

Les périphériques n'ont pas tous les mêmes demandes en termes de transfert de données. Comme n'importe quel périphérique doit pouvoir se connecter sur un port USB, on a défini trois types de transferts de données.

- Par interruption (interrupt), utilisé par les périphériques qui envoient très peu de données, seulement à des moments précis (engendrant des interruptions) : clavier, souris, ...

- En gros (bulk), utilisé par les périphériques qui reçoivent (ou envoient) les données par gros paquets (exemple : imprimante). Dans ce mode, un bloc de données (64 bytes) est envoyé et vérifié (pour être sûr qu'il a été envoyé correctement).
- Mode isochrone (isochronous), utilisé par les périphériques qui doivent échanger les données en temps réel (exemple : hauts-parleurs). Les données sont envoyées sous forme de flux, en temps réel, entre le PC et le périphérique. Il n'y a pas de correction d'erreur.

De plus, le PC peut envoyer et demander des paquets de contôles (qui seraient un 4ème type de transfert).

Notez que tous les transferts se font en série.

**2.4.6. Bande passante et frames**

Après l'énumération, le PC tient compte de toute la bande passante que les périphériques communiquant par interruption et de manière isochrone demandent. Ces périphériques peuvent consommer jusqu'à 90 % de la bande passante totale. Si les 90 % sont atteints, le PC interdit tout accès à de nouveaux périphériques par interruption ou isochrones. Les paquets de contrôle et les transferts en gros utilisent la bande passante restante (soit un minimum de 10 %).

Le contrôleur USB divise la bande passante en fenêtres (frames) et contrôle ces frames. Un frame contient 1500 bytes. Un nouveau frame début chaque seconde. Durant un frame, les périphériques par interruption et isochrones en ont une partie afin de leur garantir la bande passante dont ils ont besoin. Les transferts en gros et les paquets de contrôle utilisent l'espace restant.

USB 1.0 (1995) était limité à 12 Mbits par secondes. Avec le standard USB 2.0 (aussi nommé "Hi-Speed", 2002), la limite est de 480 Mbits par secondes (40 fois plus rapide). C'est la principale différence entre les deux.

De nouveaux, le standard 2.0 est "backward compatible", càd qu'on pourra brancher un appareil "limité" à l'USB 1 sur un port ou hub USB 2. Le périphérique communiquera avec le port à 12 Mbit/s. Idem pour l'inverse : connecter un périphérique USB 2 sur un port USB 1 marchera sans problème, sauf que la vitesse de communication sera limitée à 12 Mbits/s.

**2.4.7. Matériel**

Sur les ordinateurs, les ports série sont généralement de type A (voir les autres types ci-dessous).

**2.4.8. Connecteurs**

Il y a 4 fils dans un câble USB :

- 2 fils pour le courant (+5 volts et la terre) et
- 2 fils pour les données.

Le port USB de l'ordinateur est toujours de la même forme. De même, toute extrémité de câble USB

qui se connecte à l'ordinateur est toujours de type "A".

Afin de s'adapter à toutes les dimensions d'appareils, il y a deux types de connecteurs du côté périphérique : le B (notamment pour les imprimantes) et le mini (notamment pour les appareils photos).

**2.4.9. A propos des hubs**

Généralement, un PC possède de 2 à 4 ports USB. Mais, avec une webcam USB, une souris USB, une imprimante USB, un scanner USB, ... on arrive vite manquer de ports. La solution (pas très chère) est l'ajout d'un hub USB, sorte de réplicateur de ports. Connecté à un port USB du PC, le hub va permettre de connecter au-moins 4 périphériques. Et on peut connecter un hub à un autre hub, etc.

Un hub va jouer le rôle de multiplexeur (connection de plusieurs périphériques à un même câble) mais aussi de répétiteur, d'amplificateur, de contrôleur du signal et de fournisseur de courant. Normalement, faisant partie du standard, le hub doit être totalement transparent pour l'utilisateur.

Comme les périphériques peuvent prendre l'alimentation dans le port USB, le hub se doit de pouvoir jouer le même rôle. Bien sûr, les périphériques gros consommateurs d'énergie ont leur propre bloc fournisseur d'électricité (généralement, un transformateur). Mais les claviers, souris, webcam, etc. prennent généralement directement leur électricité dans le port. Si on a beaucoup de tel périphérique, cela peut devenir un problème (pas assez de puissance fournie par le PC) et le hub est la solution.

**2.5 Jouons avec notre PC**

Avec MS-Windows :

Pour découvrir les ports USB qu'on a sous MS-Windows, il faut aller dans le panneau de contrôle, cliquer sur l'icône "Système" et y trouver le "Gestionnaire de périphériques". Dedans, il faut trouver l'entrée "Contrôleurs USB" et l'étendre. Cela devrait donner à peu près ceci ...

Pour connaître (et éventuellement modifier) les paramètres d'un port USB, il suffit de cliquer sur son item. **[8]**

**3-FireWire :**

**3.1. Définition**

FireWire est le nom commercial donné par Apple1 à une interface série multiplexée, aussi connue sous la norme IEEE 1394 et également connue sous le nom d'interface i.LINK1, nom commercial utilisé par Sony. Il s'agit d'un bus informatique véhiculant à la fois des données et des signaux de commandes des différents appareils qu'il relie.

Plug and Play, on peut l'utiliser pour brancher toutes sortes de périphériques friands en bande

passante et qui nécessitent que le débit de données soit stable, notamment dans le cadre desdisques durs et des caméscopes numériques. Elle permet d'alimenter un périphérique, ainsi que de raccorder 63 périphériques par bus, garantissant leur branchement/débranchement pendant que le système est en marche (dit "à chaud" familièrement). On peut raccorder jusqu'à 1 024 bus par l'intermédiaire de passerelles.

FireWire a été inventé par Apple dans les années 1995 et peut atteindre des débits de plusieurs dizaines de Mo/s.

**3.2 Technologie**

FireWire utilise un multiplexage temporel : le temps est découpé en tranches de 125 microsecondes (8 000 cycles par seconde), les données étant découpées en paquets. Dans chaque tranche sont tout d'abord transmis les paquets isochrones (son, vidéo…) puis les paquets asynchrones (données informatiques). Ce système garantit la bande passante pour les flux vidéo évitant ainsi des effets de saccades et autres pertes de qualité. Les flux isochrones sont identifiés par un canal (maximum : 63), et doivent tous avoir un paquet par tranche ; une fois les paquets isochrones émis le reste du cycle est utilisé pour les paquets asynchrones identifiés non pas par un canal mais par l'identifiant du périphérique émetteur et l'identifiant du périphérique destinataire.

**3.3 Branchement**

FireWire est dit Hot Plug (branchement à chaud) ; la connexion ou la déconnexion d'un périphérique déclenche un événement bus reset chez tous les autres périphériques : ainsi tout le monde sait à tout moment qui est présent sur le bus. À chaque bus reset les périphériques reçoivent un numéro d'identification de 0 à 62 ; celui qui a le plus grand numéro est élu chef du bus ou root, et c'est lui notamment qui est chargé de marquer le début des cycles de 125 microsecondes. Tout périphérique peut ainsi être root contrairement à l'USB où ce rôle est assuré par l'hôte auquel les périphériques sont reliés.

Bien qu'il serve le plus souvent à connecter des disques durs ou des caméscopes pour réaliser des montages vidéo, ou pour réaliser des captures audio via des cartes son externes, le port FireWire peut aussi, pour des besoins ponctuels, servir à relier deux machines en réseau ; il apparaît donc comme faisant partie des périphériques de « Connexions réseau » sous Windows XP et comme interface réseau sous les systèmes utilisant un noyau Linux ou UNIX.

**3.4 Câble**

Deux brochages distincts existent en s400 et s800 : le format à 6 broches permettant l'alimentation des périphériques et le format à 4 broches sans alimentation. Le format à quatre broches est celui des PC portables et des caméscopes à bandes mini DV.

En s800 les connecteurs ont 9 broches. s400 et s800 sont compatibles : on peut connecter un

périphérique s800 avec un s400 en utilisant un câble 9 broches vers 6 broches.

Le câble le plus répandu est constitué de fils de cuivre torsadés. Sa longueur maximale pour tous les protocoles FireWire est de 4,5 m. Il existe également une transmission par fibre optique, très coûteuse mais permettant d'atteindre 100 m.

Brochage

7. 1 → VCC : 30 V
8. 2 → Masse
9. 3 → TPB- : (Twisted-pair B) signaux différentiels
10. 4 → TPB+ : (Twisted-pair B) signaux différentiels
11. 5 → TPA- : (Twisted-pair A) signaux différentiels
12. 6 → TPA+ : (Twisted-pair A) signaux différentiels

Débits :

Le FireWire permet de disposer de débits théoriques atteignant :

- 100 Mb/s en version 1 (IEEE 1394a-s100)
- 200 Mb/s en version 1 (IEEE 1394a-s200)
- 400 Mb/s en version 1 (IEEE 1394a-s400)
- 800 Mb/s en version 2 (IEEE 1394b-s800)
- 1 200 Mb/s en version 2 (IEEE 1394b-s1200)
- 1 600 Mb/s en version 2 (IEEE 1394b-s1600)
- 3 200 Mb/s en version 2 (**IEEE** 1394b-s3200)

La norme IEEE 1394b peut également être appelée FireWire Gigabit, FireWire2 ou FireWire 800.

Le s1600 et s3200 ont été adoptés par l'IEEE en août 2008. **[9]**

### 4. Thunderbolt

#### 4.1. C'est quoi ?

Thunderbolt (anciennement nommé Light Peak) est une nouvelle technologie pour connecter des périphériques, développée par Intel en collaboration avec Apple. Il est présent dans toute la nouvelle gamme des MacBook Pro.

Basée sur les architectures du PCI Express et du DisplayPort, Thunderbolt permet des connexions à haut-débit pour les disques durs, les boîtiers RAID, les interfaces d'acquisition vidéo, et les interfaces réseau. Il permet également de véhiculer de la vidéo haute-définition via le protocole DisplayPort. Chaque portThunderbolt fourni en outre jusqu'à 10 Watts pour alimenter les périphériques.

**4.2. Thunderbolt est-il différent de Light Peak ?**

Light Peak était le nom de code de Thunderbolt donné par Intel pendant le développement, il s'agit de deux noms pour la même technologie. Cependant, même si Thunderbolt est conçu pour fonctionner par câble électrique ou par fibre optique, Apple n'utilise actuellement que le câble, permettant d'alimenter en courant électrique. Intel s'attend à ce que la plupart des fabricants utilisent également les connexions électriques, tant pour l'alimentation en courant que pour réduire les coûts. La fibre optique ne sera probablement utilisée que pour les longueurs de câble supérieures à 3 mètres.

**4.3. Quel rapport avec le PCI Express ?**

Le PCI Express est l'architecture à grande vitesse qui est utilisée pour connecter la plupart des composants internes d'un Mac, comme le processeur, la carte graphique, et le disque dur. On peut considérer le PCI Express comme une autoroute pour transporter les informations de manière rapide et efficace entre ces différents points. Thunderbolt est directement connecté au bus PCI Express, ce qui explique ses performances.

**4.4 Quelle vitesse réelle ?**

En théorie, ça trace ! Un canal Thunderbolt dispose théoriquement de 10 Giga-bits par seconde (Gbps), et chaque port Thunderbolt contient 2 canaux. Thunderbolt est également bi-directionnel, il peut envoyer et recevoir des données simultanément. Même en tablant sur des performances réelles de 8 Gbps, Thunderbolt reste nettement plus rapide que le Firewire 800 et l'USB 3. Il est également plus rapide que l'eSATA.

Bien entendu, comme avec toute interface de connexion rapide, les performances des périphériques seront souvent bien en deçà, à cause des limitations intrinsèques du périphérique. Par exemple, la plupart des disques SATA sont limités à 3 Gbps, même le SATA 3.0 plafonne à 6 Gbps théoriques. De même, un périphérique plus lent au millieu de la chaîne Thunderbolt peut ralentir le débit des périphériques rapides chaînés après lui (on y revient dans un instant).

**4.5. Quels sont les avantages de Thunderbolt sur les connexions existantes ?**

Les performances sus-mentionnées sont évidemment le principal avantage par rapport au Firewire, USB, eSATA, ... Un autre point important réside dans la capacité de Thunderbolt à supporter les données, la vidéo, l'audio, et l'alimentation dans un seul port - et un seul câble- pour connecter les périphériques. Enfin, quand les périphériques et adaptateurs seront disponibles.

**4.6. Thunderbolt, ou la revanche de l'Apple Display Connector ?**

Pas vraiment, même si l'idée est similaire. Thunderbolt transporte la vidéo, l'audio, les données et l'alimentation, réduisant par là le nombre de câbles, il ne transporte pas assez de courant pour alimenter un grand écran (l'Apple Display Connector pouvait délivrer jusqu'à 100 Watts). Mais

l'ADC nécessitait une carte graphique dédiée (et coûteuse), tandis que Thunderbolt utilise un port mini-DisplayPort standard.

**4.7. Quel type de prise utilise Thunderbolt ?**

Thunderbolt utilise un connecteur compatible avec le port mini-DisplayPort, intégré à tout Mac récent.

Sur les nouveaux MacBook Pro, le port Thunderbolt est commun avec la sortie vidéo mini-DisplayPort.

**4.8. Comment relier mon écran, sans sortie mini-DisplayPort séparée ?**

Thunderbolt transmet ses données PCI Express et la vidéo DisplayPort simultanément, donc tout écran mini-DisplayPort (ou autre, avec le bon adaptateur mini-DisplayPort) peut être branché directement au port Thunderbolt, ou chaîné à la suite de périphériques Thunderbolt

**4.9. Quid des capacités audio et vidéo de Thunderbolt, versus DisplayPort ?**

Chaque port Thunderbolt inclut simultanément les connexions DisplayPort et PCI Express, donc un port Thunderbolt dispose des mêmes capacités audio et vidéo qu'un DisplayPort. En matière de vidéo, la limitation principale viendra de la carte graphique. Les nouveaux MacBook Pro supportent un écran externe jusqu'à 2560*1600 pixels, en millions de couleurs. Sur un Mac de bureau, le port Thunderbolt supportera deux écrans haute résolution. On peut y relier un écran DisplayPort, mais aussi VGA, DVI ou HDMI avec le bon adaptateur.

**4.10. Thunderbolt est-il compatible USB et Firewire ?**

Au printemps, les fabricants devraient vendre des adaptateurs permettant de relier au port Thunderbolt des périphériques USB, Firewire 400 et 800. Thunderbolt ne les rendra pas plus rapide (ils resteront limités par leurs propres performances). Par exemple, un périphérique Firewire 800 sur un bus Thundebolt sera toujours limité à 800 Mbps.

**4.11. Et les autres connecteurs ?**

Thunderbolt transportant des données, vidéos, audio, réseau, et alimentation, on peut s'attendre à voir des adaptateurs audio et Ethernet. Peut-être verra-t-on des câbles utilisant le courant du port Thunderbolt, pour alimenter un périphérique Firewire ou USB par exemple.

**4.12. Peut-on chaîner plusieurs périphériques Thunderbolt à un même port?**

Chaque port Thunderbolt supporte 6 périphériques, à chaîner en reliant les périphériques entre eux. Bien entendu, cela implique que chaque périphérique de la chaine dispose de deux ports Thunderbolt (ou d'autres ports, avec adaptateurs Thunderbolt), l'un pour se relier au périphérique précédent et l'autre pour connecter le périphérique suivant.

**4.13. La connexion de plusieurs périphériques affecte-elle les performances ?**

Contrairement à l'USB 2, où la connexion d'un périphérique USB 1 pouvait ralentir tout le bus,

Thunderbolt est conçu pour gérer des périphériques ayant des performances variables, sans affecter le bus lui-même. La bande passante globale reste partagée entre les différents périphériques sur le bus Thunderbolt, ce qui peut limiter ponctuellement les performances de certains périphériques si les autres transmettent beaucoup de données simultanément, mais les performances du bus Thunderbolt lui-même ne devraient pas baisser.

**4.14. Et pour les périphériques non-Thunderbolt ?**

Ça dépend. Si les périphériques non-Thunderbolt (plus lents) sont connectés en fin de chaîne, ils ne devraient pas affecter les performances des périphériques en amont. L'ordre de branchement sur la chaîne est donc important dans le cas de périphériques non-Thunderbolt.
Par exemple, si on utilise deux adaptateurs Thunderbolt-Firewire pour placer un disque dur Firewire au sein d'une chaîne Thunderbolt, les performances de la fin de chaîne seront réduites par la vitesse du bus Firewire, qui formera un goulet d'étranglement au milieu de la chaîne Thunderbolt.

Cependant, on devrait voir arriver des adaptateurs spécialisés et de hubs qui préservent les performances du bus Thunderbolt tout en permettant d'y connecter des périphériques Firewire, USB, Ethernet, vidéo ou audio. Des simples câbles en T comprenant deux ports Thunderbolt (en plus de l'ancien connecteur), aux hubs multi-ports permettant de relier plusieurs anciens périphériques. En utilisant ce genre d'adaptateurs spécialisés, les performances du bus Thunderbolt devraient être préservées.

En attendant ce genre de hubs, le plus gros problème reste de brancher l'écran à la fin de la chaîne, les MacBook Pro n'ayant qu'un seul port Thunderbolt et les écrans actuels n'ont pas de "sortie" Thunderbolt pour y chaîner un autre périphérique derrière. Cela rendra problématique la déconnexion d'un périphérique de milieu de chaîne (comme un disque dur Thunderbolt), puisqu'il faudra alors débrancher l'écran.

En attendant, les anciens ports (USB, Firewire, Ethernet, audio) sont toujours présents sur les Mac pour y relier les périphériques existants.

**4.15. Thunderbolt va-t-il remplacer le Firewire et l'USB sur les Mac ?**

Peut-être, mais pas avant longtemps. Thunderbolt vient d'arriver, et il va falloir attendre avant de devenir aussi populaire que l'USB ou le Firewire. Une adoption massive est attendue dans les prochaines années par les fabricants et vendeurs, mais tant que les périphériques les plus populaires n'auront pas de port Thunderbolt, il est improbable que les ports actuels disparaissent des Mac.
Cela dit, le premier iMac a adopté l'USB en 1998 -bien avant que les périphériques soient

démocratisés- et on imagine bien que l'idée d'un connecteur unique plait à Apple et à Steve Jobs. Il n'y a qu'à voir le connecteur dock des iPhone, iPad et iPod. D'ailleurs...

### 4.16. Quid des périphériques Thunderbolt disponibles actuellement ?

Thunderbolt vient tout juste de sortir, et même si plusieurs vendeurs ont annoncé des périphériques Thunderbolt, aucun n'est encore disponible.

Par exemple, Promise a annoncé le Pegasus Thunderbolt Technology DAS, un boîtier RAID 4 ou 6 baies, et LaCie a annoncé une déclinaison Thunderbolt du Little Big Disk portable contenant deux disques durs ou deux SSD.**[10]**

Etude et élaboration d'une plate-forme télé-ophtalmologique

# Chapitre III : Traitement d'images

**1-Définition**

Le traitement d'images est une discipline de l'informatique et des mathématiques appliquées qui étudie les images numériques et leurs transformations, dans le but d'améliorer leur qualité ou d'en extraire de l'information.

Il s'agit d'un sous-ensemble du traitement du signal dédié aux images et aux données dérivées comme la vidéo (par opposition aux parties du traitement du signal consacrées à d'autres types de données :son et autres signaux monodimensionnels notamment), tout en opérant dans le domaine numérique (par opposition aux techniques analogiques de traitement du signal, comme la photographie ou la télévision traditionnelles).

Dans le contexte de la vision artificielle, le traitement d'images se place après les étapes d'acquisition et de numérisation, assurant les transformations d'images et la partie de calcul permettant d'aller vers une interprétation des images traitées. Cette phase d'interprétation est d'ailleurs de plus en plus intégrée dans le traitement d'images, en faisant appel notamment à l'intelligence artificielle pour manipuler des connaissances, principalement sur les informations dont on dispose à propos de ce que représentent les images traitées (connaissance du « domaine »).

La compréhension du traitement d'images commence par la compréhension de ce qu'est une image. Le mode et les conditions d'acquisition et de numérisation des images traitées conditionnent largement les opérations qu'il faudra réaliser pour extraire de l'information. En effet, de nombreux paramètres entrent en compte, les principaux étant :

- la résolution d'acquisition et le mode de codage utilisé lors de la numérisation, qui déterminent le degré de précision des éventuelles mesures de dimensions,
- les réglages optiques utilisés, (dont la mise au point) qui déterminent par exemple la netteté de l'image,
- les conditions d'éclairage, qui déterminent une partie de la variabilité des images traitées,
- le bruit de la chaîne de transmission d'image.

Quelques exemples types d'informations qu'il est possible d'obtenir d'une image numérique :

- La luminance moyenne
- Le contraste moyen
- La couleur prédominante
- Le taux d'acuité moyen (précis ou flou)

- Le taux d'uniformité des couleurs
- La présence ou l'absence de certains objets [11]

## *2*-Définition du Filtrage

Le principe du filtrage est de modifier la valeur des pixels d'une image, généralement dans le but d'améliorer son aspect. En pratique, il s'agit de créer une nouvelle image en se servant des valeurs des pixels de l'image d'origine.

N'entrent pas dans la catégorie du filtrage toutes les transformations de l'image d'origine : zoom, découpage, projections, …

### 2-A. Filtrage Global

Dans le filtrage global, chaque pixel de la nouvelle image est calculé en prenant en compte la totalité des pixels de l'image de départ. Dans cette catégorie on trouve, par exemple, les opérations sur les histogrammes ou les opérations qui nécessitent de passer dans l'espace de Fourier.

### 2-B. Filtrage Local

Dans le filtrage local, chaque pixel de la nouvelle image est calculé en prenant en compte seulement un voisinage du pixel correspondant dans l'image d'origine. Il est d'usage de choisir un voisinage carré et symétrique autour du pixel considéré. Ces voisinages sont donc assimilables à des tableaux à deux dimensions (matrices) de taille impaire.

Voisinage 3x3 du pixel x,y

Voisinage 5x5 du pixel x,y

Gestion des bords

Lorsque le pixel considéré est proche du bord de l'image, certains points du voisinage sont en dehors de l'image d'origine. Il convient alors de choisir une stratégie pour gérer ces pixels extérieurs. Les stratégies couramment employées :

- Mise à zéro : Si un pixel du voisinage est en dehors de l'image d'origine, sa valeur est considérée comme nulle. C'est à dire : Image [-1][y]=0
- Continuité : Si un pixel du voisinage est en dehors de l'image d'origine, sa valeur est celle du pixel le plus proche qui est dans l'image d'origine. C'est à dire : Image [-1][y]= Image[0][y]
- Miroir : Si un pixel du voisinage est en dehors de l'image d'origine, sa valeur est celle du pixel symétrique par rapport au bord de l'image. C'est à dire : Image[-1][y]= Image[1][y]

• Sphérique : Si un pixel du voisinage est en dehors de l'image d'origine, sa valeur est celle du pixel correspondant si l'image était projetée sur une sphère. C'est à dire : Image [-1][y]= Image[Largeur-1][y]

**2-B-1. Filtrage Local Linéaire**

Le filtre local est dit linéaire si la valeur du nouveau pixel est une combinaison linéaire des valeurs des pixels du voisinage.
combinaison linéaire des pixels du voisinage avec i,j variant entre -h et +h, la demi taille du voisinage (pour 3x3 h=1, pour 5x5 h=2, …)
et Ai,j = valeur, entière ou réelle, spécifique au filtre linéaire.

Normalisation

Si la valeur obtenue n'est dans les limites imposées par le format d'image (entier(s) entre 0…255), alors la valeur doit être normalisée. Le facteur de normalisation peut être facilement calculé en cherchant la valeur maximale (positive) et minimale (négative) que peut atteindre la combinaison linéaire.

La valeur maximale est atteinte lorsque tous les pixels associés aux coefficients positifs ont la valeur maximale (255) et ceux associés aux coefficients négatifs ont la valeur minimale (0). Il suffit donc de faire la somme des coefficients positifs pour connaître le facteur de normalisation positif.

Idem pour le facteur de normalisation négatif, qui est la somme des coefficients négatifs.

Le facteur de normalisation est le plus grand (en valeur absolu) de ces deux facteurs.

**Facteur de normalisation du filtre linéaire**

Une fois la combinaison linéaire calculée, il suffit de diviser le résultat par le facteur de normalisation (et de convertir en entier, si besoin). La valeur obtenue peut être négative.

Une fois la normalisation effectuée, les valeurs obtenues sont dans la plage [-255,+255]. La conversion finale à effectuer (pour être dans la plage [0,255]) dépend du type de filtre et du résultat visuel souhaité : seuillage à zéro, mise à l'échelle, valeur absolue, …

Noyau

Il est d'usage de présenter les coefficients sous forme d'une matrice (appelée noyau de convolution) facilitant ainsi la mise en correspondance avec les valeurs du voisinage.

**2-B-2. Filtrage Local Non-Linéaire**

Si le filtre ne peut pas être exprimé par une combinaison linéaire, il est appelé " non-linéaire ". Les filtres non-linéaires sont plus complexes à mettre en œuvre que les filtres linéaires. Cependant les résultats obtenus avec les filtres non-linéaires sont très souvent de meilleure

qualité que ceux obtenus par les filtres linéaires.[12]

**3. Flou uniforme (uniform blur)**

- Description : Atténuation des changements brusques d'intensité
- Noyau : noyau=3x3 normalisations=9
- Calcul du noyau : tous les coefficients sont à 1.
- Principe : calculer la moyenne arithmétique des valeurs du voisinage

**4. Flou Gaussien (gaussian blur)**

- Description : Atténuation des changements brusques d'intensité
- Noyau : noyau=5x5 normalisations=273
- Calcul du noyau : coefficients du noyau gaussien
- Principe : calculer la moyenne pondérée des valeurs du voisinage. Les pixels du voisinage qui sont proches du pixel central ont un poids plus fort (= plus d'influence) que ceux qui sont plus éloignés.

**5. Détection de contours**

Le but de la détection de contours est de repérer les points d'une image numérique qui correspondent à un changement brutal de l'intensité lumineuse. Ces changements de propriétés de l'image traduisent en général des événements importants ou des changements dans les propriétés du monde. Ils incluent des discontinuités dans la profondeur, dans l'orientation d'une surface, dans les propriétés d'un matériau et dans l'éclairage d'une scène. La détection de contour est un champ de la recherche qui appartient au traitement d'image et à la vision par ordinateur, particulièrement dans le domaine de l'extraction de caractéristiques.

La détection des contours d'une image réduit de manière significative la quantité de données et élimine les informations qu'on peut juger moins pertinentes, tout en préservant les propriétés structurelles importantes de l'image. Il existe un grand nombre de méthodes de détection de l'image mais la plupart d'entre elles peuvent être regroupées en deux catégories. La première recherche les extremums de la dérivée première, en général les maximums locaux de l'intensité du gradient. La seconde recherche les annulations de la dérivée seconde, en général les annulations du laplacien ou d'une expression différentielle non-linéaire.

Dans une image en niveaux de gris, un contour est caractérisé par un changement brutal de la valeur. Le but de l'opération est de transformer cette image en une autre dans laquelle les contours apparaissent par convention en blanc sur fond noir.

Dans une section horizontale (ou verticale) de l'image rectangulaire, les variations de la valeur sont décrites par une courbe. Sur celle-ci un point d'un contour est associé à un maximum de la pente, c'est-à-dire à un extremum (maximum ou minimum) de la dérivée première. Cet extremum peut aussi s'interpréter comme un zéro de la dérivée seconde.

Dans une image numérisée, à chaque pixel est associée une valeur qui est en général différente de la valeur des pixels voisins. La notion de dérivée correspondant à une variation infiniment petite doit donc être remplacée par l'approximation différence finie utilisée en calcul numérique. Le problème est simplifié car on ne s'intéresse ici qu'aux comparaisons entre dérivées indépendamment de leurs valeurs. Ainsi la dérivée première au niveau d'un pixel se réduit à la différence entre les valeurs des deux pixels voisins, la dérivée seconde étant la différence entre les dérivées moyennées aux frontières des pixels.

Si on considère maintenant l'image rectangulaire, les dérivées premières dans les deux directions perpendiculaires s'écrivent pour le pixel centre en fonction des valeurs des pixels aux quatre points cardinaux :

$Dx = E - W \quad Dy = N - S$

Les dérivées secondes sont données par

$D2x = E - 2C + W \quad D2y = N - 2C + S$

Enfin, il faut remarquer qu'en considérant classiquement les fonctions comme des sommes de sinusoïdes, la dérivation multiplie leur amplitude par leur pulsation : c'est donc un filtre passe-haut qui introduit un bruit haute fréquence à l'origine de faux contours. Les dérivées ne sont donc en principe pas utilisées telles quelles mais associées à des filtres passe-bas . L'utilisateur peut effectuer lui-même un prétraitement avec un filtre gaussien par exemple. Des méthodes plus élaborées incluent ce prétraitement dans le filtre.[13]

### 6.Filtrage optimal

Les opérations décrites précédemment se traduisent par des filtres à appliquer à l'image. Trois critères ont été définis afin d'obtenir un filtre optimal pour la détection de contour1 :

1. **bonne détection** : détecter un maximum de contours ;
2. **bonne localisation** : les points détectés doivent être les plus proches possibles du vrai contour ;

3. **réponse unique** : minimiser le nombre de contours détectés plusieurs fois.

Ces critères se traduisent par des conditions sur la réponse impulsionnelle du filtre et débouchent sur des détecteurs de contours très performants.

**Méthode de base**

L'opérateur de Roberts décrit dans une image rectangulaire, ayant un contour rampe, les dérivées premières elles-mêmes sont d'un intérêt limité car la pente maximale a peu de chances de se trouver sur l'une des deux directions considérées. Ce qui importe c'est la longueur du vecteur gradient dont elles sont les composantes. Cette longueur se calcule en principe par le théorème de Pythagore au prix d'un calcul sur les réels et on l'accélère considérablement en utilisant une approximation entière :

Indépendamment de la précision du calcul, il faut transformer ce résultat en un filtre numérique. La notion physique de filtre correspond à la notion mathématique de convolution. Lorsqu'il s'agit de données numérisées comme dans le cas du traitement d'images, la relation entre les valeurs des pixels de sortie et celle des pixels d'entrée est décrite par un tableau de nombres appelé matrice de convolution.

En terme simple, le filtre calcule le gradient d'intensité lumineuse de l'image à chaque point, donnant la direction et le taux de la plus grande décroissance. Le résultat nous indique les changements abrupts de luminosité de l'image et donc exhibe les contours probables de celle-ci. En pratique cette technique est plus fiable et facile à mettre en œuvre qu'un algorithme plus directe.

Techniquement, il s'agit d'un opérateur différentiel discret calculant une approximation du Gradient d'intensité lumineuse d'une image.

**Formulation**

Mathématiquement, le filtre est composé de deux matrices 3x3 que l'on va convoluer avec l'image originale pour calculer une approximation de sa dérivée en tout point. La première matrice donne la dérivée horizontale et la seconde donne la dérivée verticale. Si nous définissons comme étant l'image source, et et les deux images dérivées horizontales et verticales de l'intensité lumineuse de l'image, nous pouvons les calculer ainsi avec l'opération de convolution 2d suivantes:

Cet opérateur ainsi définit pouvant se décomposer en un produit d'une moyenne et d'une différentielle, il calcule le gradient lissé.

Enfin, nous pouvons calculer l'amplitude du gradient via la norme du vecteur ainsi définit en tout point:

De même, nous pouvons calculer sa direction par exemple, elle correspond à un front vertical qui est plus sombre du côté droit.

Pour faire simple, l'opérateur calcule le gradient de l'intensité de chaque pixel. Ceci indique la direction de la plus forte variation du clair au sombre, ainsi que le taux de changement dans cette direction. On connaît alors les points de changement soudain de luminosité, correspondant probablement à des bords, ainsi que l'orientation de ces bords.

En termes mathématiques, le gradient d'une fonction de deux variables (ici l'intensité en fonction des coordonnées de l'image) est un vecteur de dimension 2 dont les coordonnées sont les dérivées selon les directions horizontale et verticale. En chaque point, le gradient pointe dans la direction du plus fort changement d'intensité, et sa longueur représente le taux de variation dans cette direction. Le gradient dans une zone d'intensité constante est donc nul. Au niveau d'un contour, le gradient traverse le contour, des intensités les plus sombres aux intensités les plus claires.

**Formulation**

L'opérateur utilise des matrices de convolution. La matrice (généralement de taille 3×3) subit une convolution avec l'image pour calculer des approximations des dérivées horizontale et verticale. Soit l'image source, et deux images qui en chaque point contiennent des approximations respectivement de la dérivée horizontale et verticale de chaque point.

En chaque point, les approximations des gradients horizontaux et verticaux peuvent être combinées comme suit pour obtenir une approximation de la norme du gradient:

où, par exemple,  vaut 0 pour un contour vertical plus foncé à gauche.

**Précisions**

Puisque l'intensité d'une image numérique est discrète, les dérivées de cette fonction ne peuvent pas être définies si ce n'est sous une hypothèse de continuité de la fonction intensité continue qui a été échantillonnée. En pratique on peut calculer des approximations plus ou moins fidèles du gradient en chaque point. [14]

**7.Filtres utilisés**

**7.1.Filtre de Sobel**

Le filtre de Sobel calcule une approximation assez inexacte du gradient d'intensité, mais cela suffit en pratique dans beaucoup de cas. En effet, il n'utilise qu'un voisinage (généralement de taille 3×3) autour de chaque point pour calculer le gradient, et les poids utilisés pour le calcul du gradient sont entiers.

**Détails d'implémentation**

De par sa simplicité, le filtre de Sobel peut être aisément implémenté de manière logicielle ou même matérielle : seulement huit points autour du point considéré sont nécessaires pour calculer le gradient. Ce calcul utilise simplement des calculs sur les entiers. De plus, les filtres horizontal et vertical sont séparables et les deux dérivées peuvent être calculées comme suit :

La séparabilité peut être mise à profit dans certains types d'implémentation pour permettre moins d'opérations lors du calcul.

Plus fondamentalement, en divisant par quatre, ces formules montrent que la dérivation dans une direction est associée à un lissage triangulaire dans l'autre direction destiné à éliminer les « faux contours », la même technique étant utilisée dans le filtre de Prewitt avec un lissage rectangulaire qui introduit des changements de phase [15]

**7.2 Filtre de Canny :**

Le filtre de Canny (ou détecteur de Canny) (1986) est utilisé en traitement d'images pour la détection des contours. L'auteur l'a conçu pour être optimal suivant trois critères clairement explicités :

1. bonne détection : faible taux d'erreur dans la signalisation des contours,
2. bonne localisation : minimisation des distances entre les contours détectés et les contours réels,
3. clarté de la réponse : une seule réponse par contour et pas de faux positifs

**7.3 Réduction du bruit**

La première étape est de réduire le bruit de l'image originale avant d'en détecter les contours. Ceci permet d'éliminer les pixels isolés qui pourraient induire de fortes réponses lors du calcul du gradient, conduisant ainsi à de faux positifs.

Un filtrage gaussien 2D est utilisé (voir Lissage de l'image), dont voici l'opérateur de convolution :

et un exemple de masque 5×5 discret avec $\sigma=1,4$ :

Usuellement, un filtre est de taille plus réduite que l'image filtrée. Plus le masque est grand, moins le détecteur est sensible au bruit et plus l'erreur de localisation grandit.

**7.4 Gradient d'intensité**

Après le filtrage, l'étape suivante est d'appliquer un gradient qui retourne l'intensité des contours. L'opérateur utilisé permet de calculer le gradient suivant les directions X et Y, il est composé d'une paire de deux masques de convolution, un de dimension 3×1 et l'autre 1×3:

**7.5 Direction des contours**

Nous obtenons finalement une carte des gradients d'intensité en chaque point de l'image accompagnée des directions des contours.

**7.6 Suppression des non-maxima**

La carte des gradients obtenue précédemment fournit une intensité en chaque point de l'image. Une forte intensité indique une forte probabilité de présence d'un contour. Toutefois, cette intensité ne suffit pas à décider si un point correspond à un contour ou non. Seuls les points correspondant à des maxima locaux sont considérés comme correspondant à des contours, et sont conservés pour la prochaine étape de la détection.

Un maximum local est présent sur les extrema du gradient, c'est-à-dire là où sa dérivée s'annule.

**7.7 Seuillage des contours**

La différenciation des contours sur la carte générée se fait par seuillage à hysteresis.

Cela nécessite deux seuils, un haut et un bas; qui seront comparés à l'intensité du gradient de chaque point. Le critère de décision est le suivant. Pour chaque point, si l'intensité de son gradient est :

- Inférieur au seuil bas, le point est rejeté ;
- Supérieur au seuil haut, le point est accepté comme formant un contour ;
- Entre le seuil bas et le seuil haut, le point est accepté s'il est connecté à un point déjà accepté.

Une fois ceci réalisé, l'image obtenue est binaire avec d'un côté les pixels appartenant aux contours et les autres.

**7..8 Paramètres**

Les deux paramètres principaux déterminant le temps de calcul et l'acuité de l'algorithme sont la taille du filtre gaussien et les deux seuils.

- Taille du filtre: le filtre utilisé lors de la réduction du bruit a une influence directe sur le comportement de l'algorithme. Un filtre de petite taille produit un effet de flou moins prononcé, ce qui permet la détection de petites lignes bien marquées. Un filtre de taille plus grande produit un effet de flou plus important, ce qui permet de détecter des contours moins nets, par exemple celui d'un arc-en-ciel.

- Seuils: l'utilisation de deux seuils au lieu d'un améliore la flexibilité mais certains problèmes propres au seuillage demeurent. Ainsi, un seuil trop bas peut conduire à la

détection de faux positifs. Inversement, un seuil trop haut peut empêcher la détection de contours peu marqués mais représentant de l'information utile.

Il n'existe pas actuellement de méthode générique pour déterminer des seuils produisant des résultats satisfaisants sur tous les types d'images.[16]

# Chapitre IV : Transfert de données

## 1. Protocol TCP / IP :

### 1.1 Adressage IP v4

Une adresse IP est un entier écrit sur quatre octets, elle peut donc prendre des valeurs entre 0 et $2^{32}$ - 1. Pour plus de commodité, on note les adresses en donnant les valeurs de chaque octet séparés par des points ; par exemple, 11000000101010000000000100001101 s'écrit : 11000000 10101000 00000001 00001101. devient 192.168.1.13.

Une adresse IP est constituée de deux parties : l'adresse du réseau et l'adresse de la machine, elle permet donc de distinguer une machine sur un réseau. Deux machines se trouvant sur un même réseau possèdent la même adresse réseau mais pas la même adresse machine.

### 1.2 Masques réseau

Ce découpage en deux parties est effectué en attribuant certains bits d'une adresse à la partie réseau et le reste à la partie machine. Il est représenté en utilisant un « masque réseau » où sont placés à 1 les bits de la partie réseau et à 0 ceux de la partie machine.

Par exemple 207.142.131.245 est une adresse IP (celle de Wikilivres, en fait) et 255.255.255.0 un masque réseau indiquant que les trois premiers octet (les 24 premiers bits) sont utilisés pour adresser le réseau et le dernier octet (les 8 derniers bits) pour la machine. 207.142.131.245/255.255.255.0 désigne donc la machine d'adresse 245 sur le réseau d'adresse 207.142.131.0.

Lorsque les bits du masque réseau sont contigus, on utilise une notation plus courte : IP/nombre de bits à 1. 207.142.131.245/255.255.255.0 peut donc aussi se noter 207.142.131.245/24.

### 1.3 Classes d'adresses

Il existe différents découpages possible que l'on appelle « classes d'adresses ». À chacune de ces classes correspond un masque réseau différent :

| Classe | premiers bits | premier octet | masque |
|---|---|---|---|
| A | 0 | 0-127 | 255.0.0.0 |
| B | 10 | 128-191 | 255.255.0.0 |
| C | 110 | 192-223 | 255.255.255.0 |
| D | 1110 | 224-239 | |
| E | 1111 | 240-255 | |

Les adresses de classe A permettent donc de créer des réseaux avec plus de machines, par

contre, il y a beaucoup plus de réseaux de classe C possibles que de réseaux de classe A ou B.

La classe D est une classe utilisée pour le « multicast » (envoie à plusieurs destinataires) et la classe E est réservée.

Adresses réseaux et adresses de diffusion

Une adresse réseau est une adresse IP qui désigne un réseau et non pas une machine de ce réseau. Elle est obtenue en plaçant tous les bits de la partie machine à zéro.

Une adresse de diffusion (« broadcast » en anglais) est une adresse permettant de désigner toutes les machines d'un réseau, elle est obtenue en plaçant tous les bits de la partie machine à un.

Par exemple :

| lasse) | ue | sse réseau | sse de diffusion |
|---|---|---|---|
| ).10.10 (A) | ).0.0 | 0.0 | 5.255.255 |
| 68.150.35 (C) | 255.255.0 | 68.150.0 | 68.150.255 |

Adresses déconseillées et réseaux privés

Pour éviter les ambiguïtés avec les adresses de réseau et les adresses de diffusion, les adresses « tout à zéro » et « tout à un » sont déconseillées pour désigner des machines sur un réseau.

Dans chaque classe d'adresses, certaines adresses réseaux sont réservées aux réseaux privés.

| s e | u privé |
|---|---|
| | 0.0 |
| | ).0.0 |
| | '2.16.0.0 à 172.31.0.0 |
| | )2.168.0.0 à 192.168.255.0 |

Le cas du réseau 127.0.0.1 est particulier : il désigne la boucle locale.

**1.4 Sous-réseaux**

Il est possible de découper un réseau en sous-réseaux en utilisant un masque de sous-réseau. Un masque de sous-réseau permet d'attribuer des bits supplémentaires à la partie réseau d'une adresse IP.

Supposons que l'on dispose d'une adresse de classe C, elle permet normalement d'adresser 254 machines avec le masque 255.255.255.0. Il est possible de découper ce réseau en deux

sous réseaux de 126 machines avec le masque 255.255.255.128 ($128 = 10000000_2$).

**2. Adressage IP**

**2.1 Le protocole IP**

Le protocole IP (Internet Protocol) est un des protocoles majeurs de la pile TCP/IP. Il s'agit d'un protocole réseau (niveau 3 dans le modèle OSI). Il n'est pas orienté connexion, c'est a dire qu'il n'est pas fiable.C'est la couche transport qui peut le rendre fiable.

**2.2 Adresse IP**

Dans un réseau IP, chaque interface possède une adresse IP fixée par l'administrateur du réseau ou attribuée de façon dynamique via des protocoles comme DHCP. Par extension, pour une machine simple, un PC, avec une seule interface Ethernet, on dira que cette machine a une adresse IP. Il est déconseillé de donner la même adresse à 2 machines différentes sous peine de problèmes (collisions).

Une adresse IP (IPv4 pour être précis) est une suite de 32 bits notée en général a.b.c.d avec a, b, c, et d des entiers entre 0 et 255. Chaque valeur a, b, c ou d représente dans ce cas une suite de 8 bits.

**Exemple :** une machine a comme adresse IP 134.214.80.12. a vaut 134 soit (1000 0110) en binaire. b vaut 214 soit (1101 0110) en binaire. c vaut 80 soit (0101 0000) et d vaut 12 vaut (0000 1100). En binaire, l'adresse IP s'écrit donc 1000 0110 1101 0110 0101 0000 0000 1100.

**2.3 Taille des réseaux IP**

Un réseau IP peut avoir une taille très variable :

**8.**une entreprise moyenne aura un réseau comportant une centaine de machines.

**9.**un campus universitaire aura un réseau comportant de quelques milliers à quelques dizaines de milliers de machines.

**10.**un grand fournisseur d'accès peut raccorder des millions de postes.

**11.**tous ces différents réseaux peuvent être interconnectés.

**Les numéros de réseau (net-id) et de station (host-id)**

Au sein d'un même réseau IP, toutes les adresses IP commencent par la même suite de bits. L'adresse IP d'une machine va en conséquence être composée de 2 parties : le net-id (la partie fixe) et le host-id (la partie variable).

**2.4 Masque d'un réseau IP**

Le masque du réseau permet de connaître le nombre de bits du net-id. On appelle N ce

nombre. Il s'agit d'une suite de 32 bits composée en binaire de N bits à 1 suivis de 32-N bits à 0.

**•Exemple de masque Classe A**

Le réseau d'une multinationale comprend toutes les adresses IP commençant par 5 (ici 5 n'est évidemment donné qu'à valeur informative). Une adresse IP sera du type 5.*.*.*. Le net-id comporte 8 bits et le host-id comporte 24 bits. Le masque s'écrira donc en binaire 8 bits à 1 suivi de 24 bits à 0 soit 1111 1111 0000 0000 0000 0000 0000 0000. Le masque sera donc 255.0.0.0 Un tel réseau peut comporter $2^{24}$machines soit 16 millions environ.

- **Exemple de masque Classe B**

Le réseau d'un campus universitaire comprend toutes les adresses IP commençant par 134.214. Une adresse IP sera du type 134.214.*.*. Le net-id comporte 16 bits et le host-id comporte 16 bits. Le masque s'écrira donc en binaire 16 bits à 1 suivi de 16 bits à 0 soit 1111 1111 1111 1111 0000 0000 0000 0000. Le masque sera donc 255.255.0.0. Un tel réseau peut contenir au maximum $2^{16}$ machines soit 65536 machines.

- **Exemple de masque Classe C**

le réseau d'une PME comprend toutes les adresses IP commençant par 200.150.17. Une adresse IP sera du type 200.150.17.* Le net-id comporte 24 bits et le host-id comporte 8 bits. Le masque s'écrira donc en binaire 24 bits à 1 suivi de 8 bits à 0 soit 1111 1111 1111 1111 1111 1111 0000 0000. Le masque sera donc 255.255.255.0. Un tel réseau peut contenir au maximum $2^8$ machines soit 256 machines.**[17]**

## 2.5 Adresse réseau

Chaque réseau IP a une adresse qui est celle obtenue en mettant tous les bits de l'host-id à 0. Le réseau de l'exemple 3 a comme adresse réseau 200.150.17.0. Un réseau IP est complètement défini par son adresse de réseau et son masque de réseau.

## 2.6 Notation CIDR

La notation CIDR, pour *Classless Inter-Domain Routing*, est historiquement introduite après la notion de classe d'adresse IP (cf. section sur les classes). Elle s'inscrit dans une intention d'outrepasser la limite implicitement fixée par la notion de classe en termes de plages d'adresses disponibles dans les réseaux IPv4.

La notation initiale non CIDR considère pour un réseau donné le couple formé par l'adresse et le masque dudit réseau. En notation CIDR, une forme d'adressage équivalente est construite – ou obtenue, si l'on part de l'adresse en notation initiale non CIDR – par

l'association de l'adresse du réseau (à l'instar de la notation initiale) et de la longueur du préfixe binaire déterminant ledit réseau. Le préfixe binaire de la notation CIDR correspond au nombre des premiers bits à 1 dans la forme binaire du masque du réseau de la notation initiale non CIDR.

En adressage IPv4, cela se concrétise par une forme décimale de 4 octets suivie d'un entier compris entre 0 et 32. En pratique, cette plage peut s'étendre de 1 à 31 afin de permettre un adressage des hôtes (*host-id*) par les bits différentiels (en effectif non nul).

Exemples

On considère le réseau d'adresse (décimale) 150.89.0.0 et de masque (décimal) 255.255.0.0 en notation initiale non CIDR. Ledit masque comporte 16 bits à 1 ; ces 16 bits sont les 16 premiers bits du masque. En notation CIDR, ce réseau est identifié par la forme décimale suivante : 150.89.0.0/16.

De la même manière, le réseau d'adresse (décimale) 200.89.67.0 et de masque (décimal) 255.255.255.0 pourra être identifié par la notation CIDR 200.89.67.0/24.

Pour un réseau d'adresse (décimale) 192.168.144.0 et de masque (décimal) 255.255.240.0, la notation CIDR sera 192.168.144.0/20.

## 2.7 Adresse de diffusion (broadcast)

Cette adresse permet à une machine d'envoyer un datagramme à toutes les machines d'un réseau. Cette adresse est celle obtenue en mettant tous les bits de l'host-id à 1. Le réseau de l'exemple 3 a comme adresse de broadcast 200.150.17.255.

## 2.8 Deux adresses interdites

Il est interdit d'attribuer à une machine d'un réseau IP, l'adresse du réseau et l'adresse de broadcast.

Ce qui, pour le réseau 192.168.1.0/24, nous donne :

- adresse du réseau : 192.168.1.0
- adresse de broadcast : 192.168.1.255

## 2.9 Les classes A, B et C (obsolète)

Historiquement, le réseau Internet était découpé en classes d'adresses :

### 2.9.1 Classe A :

1. Le premier bit de ces adresses IP est à 0.
2. Le masque décimal associé est 255.0.0.0, soit les 8 premiers bits à 1.
3. Les adresses de ces réseaux ont la forme décimale a.0.0.0 avec *a* variant 0 à ($2^7$-1 =)

127.

4. Cette classe détermine ainsi (127 - 0 + 1 =) 128 réseaux.
5. Le nombre de bits restant pour l'adressage des hôtes est de (32 - 8 =) 24.
6. Chaque réseau de cette classe peut donc contenir jusqu'à $2^{24}$-2 = 16 777 214 machines.

**2.9.2 Classe B :**

1. Les 2 premiers bits de ces adresses IP sont à 1 et 0 respectivement.
2. Le masque décimal associé est 255.255.0.0, soit les 16 premiers bits à 1.
3. Les adresses de ces réseaux ont la forme décimale a.b.0.0 avec *a* variant de ($2^7$ =) 128 à ($2^7 + 2^6$-1 =) 191 et *b* variant de 0 à 255.
4. Cette classe détermine ainsi ( (191 - 128 + 1) × (255 - 0 + 1) =) 16 384 réseaux.
5. Le nombre de bits restant pour l'adressage des hôtes est de (32 - 16 =) 16.
6. Chaque réseau de cette classe peut donc contenir jusqu'à $2^{16}$-2 = 65 534 machines.

**2.9.3 Classe C :**

- Les 3 premiers bits de ces adresses IP sont à 1, 1 et 0 respectivement.
- Le masque décimal associé est 255.255.255.0, soit les 24 premiers bits à 1.
- Les adresses de ces réseaux ont la forme décimale a.b.c.0 avec *a* variant de ($2^7 + 2^6$ =) 192 à ($2^7 + 2^6 + 2^5$-1 =) 223, *b* et *c* variant de 0 et 255 chacun.
- Cette classe détermine ainsi ( (223 - 192 + 1) × (255 - 0 + 1) × (255 - 0 + 1) =) 2 097 152 réseaux.
- Le nombre de bits restant pour l'adressage des hôtes est de (32 - 24 =) 8.
- Chaque réseau de cette classe peut donc contenir jusqu'à $2^8$-2 = 254 machines.

**2.9.4 Classe D :**

- Les 4 premiers bits de ces adresses IP sont à 1, 1, 1 et 0 respectivement.
- Le masque décimal associé par défaut est 240.0.0.0, soit les 4 premiers bits à 1
- Les adresses de cette classe ont la forme décimale a.b.c.d avec *a* variant de ($2^7 + 2^6 + 2^5$ =) 224 à ($2^7 + 2^6 + 2^5 + 2^4$-1 =) 239, *b*, *c* et *d* variant de 0 et 255 chacun.
- Cette classe est spéciale : elle est réservée à l'adressage de groupes de diffusion multicast.

**2.9.5 Classe E :**

- Les 4 premiers bits de ces adresses IP sont (tous) à 1.

- Le masque décimal associé par défaut est 240.0.0.0, soit les 4 premiers bits à 1
- Les adresses de cette classe ont la forme décimale a.b.c.d avec *a* variant de ($2^7 + 2^6 + 2^5 + 2^4$ =) 240 à ($2^8$-1 =) 255, *b*, *c* et *d* variant de 0 et 255 chacun.
- Cette classe est également spéciale : elle est actuellement réservée à un adressage de réseaux de recherche.

La notion de classe d'adresses a été rendue obsolète pour l'adressage des nœuds du réseau Internet car elle induisait une restriction notable des adresses IP affectables par l'utilisation de masques spécifiques. Les documents RFC 1518 et RFC 1519 publiés en 1993 spécifient une nouvelle norme : l'adressage CIDR (cf. supra). Ce nouvel adressage précise qu'il est possible d'utiliser un masque quelconque appliqué à une adresse quelconque. Il organise par ailleurs le regroupement géographique des adresses IP pour diminuer la taille des tables de routage des principaux routeurs du réseau Internet.

**Exemple**

Une machine possède l'adresse IP 134.214.80.12 : elle appartient au réseau de classe B 134.214.0.0 de masque 255.255.0.0. Dans ce réseau, une machine peut avoir une adresse IP comprise entre 134.214.0.1 et 134.214.255.254. L'adresse de broadcast est 134.214.255.255**.[18]**

## 2.10 Adresses privées (non routables sur l'Internet)

Un certain nombre de ces adresses IP sont réservées pour un usage interne aux entreprises (RFC 1918) Elles ne doivent pas être utilisées sur l'internet où elles ne seront de toute façon pas routées. Il s'agit des adresses :

8. de 10.0.0.0 à 10.255.255.255
9. de 172.16.0.0 à 172.31.255.255
10. de 192.168.0.0 à 192.168.255.255
11. les adresses de 127.0.0.0 à 127.255.255.255 sont également interdites.

Les adresses 127.0.0.0 à 127.255.255.255 s'appelle l'adresse de boucle locale (loopback en anglais) et désigne la machine locale (localhost).

## 2.11 Distribution des adresses IP

Sur l'internet, l'organisme IANA est chargé de la distribution des adresses IP. IANA a délégué la zone européenne à un organisme : le RIPE NCC. Cet organisme distribue les adresses IP aux fournisseurs d'accès à l'internet

## 2.12 Découpage d'un réseau IP

Un réseau IP de classe A, B ou C peut être découpé en sous-réseaux. Lors d'un découpage le nombre de sous-réseaux est une puissance de 2 : 4, 8, 16, 32… ce qui est naturel si l'on pense à la représentation binaire d'une adresse IP. Chaque sous-réseau peut être découpé en sous-sous-réseaux et ainsi de suite On parle indifféremment de réseau IP pour désigner un réseau, un sous-réseau, … Chaque sous-réseau sera défini par un masque et une adresse IP.

### 2.12.1 Exemple de découpage

On considère le réseau d'adresse 134.214.0.0 et de masque 255.255.0.0. On veut découper ce réseau en 8 sous-réseaux. Pour chaque sous-réseau, on veut obtenir le masque et l'adresse.

- **Calcul du masque**

On veut découper le réseau en 8. Or $8 = 2^3$. En conséquence, le masque de chaque sous-réseau est obtenu en ajoutant 3 bits à 1 au masque initial. L'ancien masque 255.255.0.0 comprend 16 bits à 1 suivis de 16 bits à 0. Le nouveau masque comprendra donc 16 + 3 = 19 bits à 1 suivis de 13 bits à 0. Il correspond à 255.255.224.0.

**Calcul du net-id de chaque sous réseau**

Le net-id de chaque sous-réseau sera constitué de 19 bits :

Les 16 premiers bits seront ceux de l'écriture binaire du préfixe d'adresse 134.214 ;

Les 3 bits suivants seront constitués du numéro du sous-réseau : 000 (0), 001 (1), 010 (2), 011 (3), 100 (4), 101 (5), 110 (6) ou 111 (7).

**Calcul de l'adresse de chaque sous-réseau**

Pour obtenir l'adresse réseau, tous les bits du host-id sont positionnés à 0. On obtient donc comme adresse pour chaque sous-réseau :

- 134.214.(000 00000).0 soit 134.214.0.0
- 134.214.(001 00000).0 soit 134.214.32.0
- 134.214.(010 00000).0 soit 134.214.64.0
- 134.214.(011 00000).0 soit 134.214.96.0
- 134.214.(100 00000).0 soit 134.214.128.0
- 134.214.(101 00000).0 soit 134.214.160.0
- 134.214.(110 00000).0 soit 134.214.192.0
- 134.214.(111 00000).0 soit 134.214.224.0.

**Obtention des adresses de broadcast**

Pour obtenir l'adresse de broadcast, on met à 1 tous les bits du host-id. Les adresses de broadcast sont donc :

- 134.214.(000 11111).255 soit 134.214.31.255
- 134.214.(001 11111).255 soit 134.214.63.255
- 134.214.(010 11111).255 soit 134.214.95.255
- 134.214.(011 11111).255 soit 134.214.127.255
- 134.214.(100 11111).255 soit 134.214.159.255
- 134.214.(101 11111).255 soit 134.214.191.255
- 134.214.(110 11111).255 soit 134.214.223.255
- 134.214.(111 11111).255 soit 134.214.255.255.

La pile TCP/IP

**3 Le datagramme IP (version 4)**

## Lorsque deux machines communiquent en utilisant le protocole IP, elles s'échangent des datagrammes IP qui ont le format ci-dessous :

| 32 bits (= 4 octets) | | | |
|---|---|---|---|
| Numéro de version | Longueur en-tête | Type de service | Longueur totale du datagramme |
| Identificateur (recopié dans chaque segment) | | | Drapeaux + position du segment |
| Durée de vie | | Protocole couche 4 | Somme de contrôle de l'en-tête |
| Adresse IP source | | | |
| Adresse IP destination | | | |
| Options | | | |
| Données | | | |

### 3.1.1 Version *(4 bits)* :

le champ version indique la version utilisée du protocole IP. Début 2006, la version de IP la plus fréquemment utilisée est la version 4. La version 6 commence à apparaitre : il n'y aura pas de version 5. Les 4 bits de ce champ sont donc 0100 (codage en binaire de la valeur décimale 4).

### 3.1.2 IHL = IP Header Length - longueur de l'en-tête IP *(4 bits)* :

Ce champ indique la longueur de l'entête IP. L'unité est le nombre de mots de 32 bits. Pour la version 4 la longueur de cette entête est de 20 octets soit 5 fois 32 bits : ce champ vaut donc 0101.

### 3.1.3 Type of service *(8 bits)* :

Ce champ permet d'indiquer que certains datagrammes IP ont une priorité supérieure à d'autres. Il est peu utilisé sauf par quelques routeurs spécialisés dans la transmission de voix sur IP.

### 3.1.4 Total length *(16 bits)* :

Ce champ indique le nombre d'octets du datagramme, en-tête IP comprise. La longueur maximale du datagramme en octets est $2^{16}$-1 = 65 535 octets = 64 ko -1 octet

### 3.1.5 ID *(16 bits)* :

Ce champ est un identifiant du datagramme IP (le numéro du datagramme).

### 3.1.6 F = Flags - les drapeaux *(3 bits)* :

Le premier bit est inutilisé

Le deuxième bit DF (don't fragment) permet d'interdire ou d'autoriser la fragmentation. positionné à 1, il est interdit de fragmenter ce datagramme IP.

Le troisième bit MF (more fragment) est utilisé lors de la fragmentation : il indique si le fragment est le dernier fragment du datagramme (MF=0) ou non (MF=1).

### 3.1.7 TTL = Time to live - temps restant à vivre *(8 bits)* :

Il s'agit d'une valeur initialisée par l'émetteur et qui est décrémentée de 1 à chaque fois que le datagramme traverse un routeur. Si le TTL arrive à la valeur 0, le datagramme est détruit : ce mécanisme assure la destruction des datagrammes qui se perdent sur le réseau. Ainsi ces datagrammes perdus n'encombrent pas indéfiniment le réseau.

### 3.1.8 Protocole *(8 bits)* :

Ce champ indique la nature des données transportées par ce datagramme IP. 3 protocoles sont principalement utilisés au-dessus de IP : ICMP (code 1), TCP (code 6) et UDP (code 17).

### 3.1.9 Header Checksum - somme de contrôle de l'en-tête *(16 bits)* :

Il s'agit d'un code détecteur d'erreurs qui ne porte que sur l'entête : la somme des octets de l'entête regroupé par paquets de 16 bits (header checksum compris) doit valoir $2^{16}$-1 modulo $2^{16}$. En cas d'erreur sur l'entête le datagramme est détruit. IP n'est pas un protocole fiable puisqu'on ne garantit pas que les données arrivent, ni de leur fiabilité.

### 3.1.10 IP source *(32 bits)* :

Adresse IP de l'expéditeur.

### 3.1.11 IP destination *(32 bits)* :

adresse IP du destinataire.**[19]**

## 4 La fragmentation IP

### 4.1 Notion de MTU

**La plupart des réseaux imposent une limite physique à la taille des données qu'ils peuvent transporter. Un datagramme IP peut avoir une taille maximale de 65535 octets, ce qui est trop grand pour la plupart des réseaux. Le MTU (Maximum Transfert Unit) correspond à la taille maximale des données transportables par le réseau. Le datagramme IP (entête comprise) aura comme taille maximale le MTU du réseau.**

#### 4.1.1 Quelques valeurs du MTU

Ethernet : MTU = 1500 octets (fixé à 1492 pour optimiser sa connexion). FDDI : 4470 octets.

#### 4.1.2 Découpage par le routeur

Si un routeur route des données d'un réseau A vers un réseau B et si les 2 réseaux ont des valeurs différentes de MTU (par exemple le réseau A est un réseau FDDI de MTU 4470 et le réseau B est un réseau Ethernet de MTU 1500 octets), alors il peut être amené à découper un datagramme IP en plusieurs datagrammes plus petits.

### 4.2 Le rôle des flags et du champ FO

Si un datagramme a été découpé, il faut être capable de réunir les différents morceaux dans le bon ordre à l'arrivée : les flags (3 bits) et le champ FO (Fragment Offset sur 13 bits) vont contenir les informations nécessaires à cette reconstruction.

#### 4.2.1 Les flags

**premier bit : inutilisé. La valeur est toujours à 0.**

•deuxième bit : DF (Don't fragment) permet d'autoriser ou d'interdire la fragmentation du datagramme. Si le bit DF est à 0, la fragmentation est autorisée et s'il vaut 1, elle est interdite. Si le routeur doit découper un datagramme et que le bit DF est à 1, alors le datagramme IP sera détruit.

•troisième bit : MF (More Fragment) permet d'indiquer si le datagramme est le dernier ou

non. Si MF est à 0, alors le fragment est le dernier, s'il vaut 1, alors il n'est pas le dernier.

•

**4.2.3 Le champ FO**

**Le nombre d'octets de données de chaque fragment, sauf le dernier, doit être un multiple de 8. Ainsi la position du premier octet de chaque fragment dans le datagramme d'origine sera un multiple de 8. La valeur du champ FO de chaque fragment sera le quotient de cette position par 8, codé en binaire sur 13 bits.**

**Exemple**

## Données de départ :

•Un routeur doit router un datagramme IP dont la taille totale fait 4470 octets sur un réseau Ethernet.

•Le datagramme initial comprend 20 octets d'entête et 4450 octets de données. Numérotons de 0 à 4449 ces octets.

•Chaque fragment aura une taille maximale de 1500 octets, soit 20 octets d'entête et 1480 octets de données.

**4.3 Répartition des données dans les fragments**

On mettra dans le premier fragment les octets de 0 à 1479 du datagramme initial.

On mettra dans le deuxième fragment les octets de 1480 à 2959 du datagramme initial.

On mettra dans le troisième fragment les octets de 2960 à 4439 du datagramme initial.

On mettra dans le quatrième fragment les octets de 4440 à 4449 du datagramme initial.

La position du premier octet du fragment dans le datagramme initial sera donc :

•premier fragment : 0

•deuxième fragment : 1480

•troisième fragment : 2960

•quatrième fragment : 4440

On s'aperçoit que la condition **Chaque fragment sauf le dernier contient un nombre d'octets multiple de 8** a pour conséquence que dans tous les fragments y compris le dernier la position du premier octet du fragment dans le datagramme initial est un multiple de 8.

**4.3.2 Description des fragments :**

•premier fragment : FO=0/8=0 et MF=1

•deuxième fragment : FO=1480/8=185 et MF=1

•troisième fragment : FO=2960/8=370 et MF=1

•quatrième fragment : FO=4440/8=555 et MF=0

Le protocole ARP

**4.3.3 Encapsulation des protocoles**

**Lorsqu'un datagramme IP est envoyé sur un réseau Ethernet alors sur le réseau il ne va circuler que des trames Ethernet. Ces trames Ethernet contiendront dans leur champ de données un datagramme IP. On dit que le protocole IP est encapsulé dans Ethernet.**

•L'encapsulation peut être plus complexe. Lorsque une machine envoie un datagramme ICMP, celui-ci sera encapsulé dans un datagramme IP qui sera encapsulé dans une trame Ethernet.

**4.3.4 Format de la trame Ethernet**

**Une trame Ethernet contient les champs suivants :**

•un préambule sur 64 bits : 31 fois 01 suivi de 11. Le préambule sert à la synchronisation ;

•l'adresse MAC du destinataire ;

•l'adresse MAC de l'expéditeur ;

•un code sur 16 bits indiquant le protocole utilisé au dessus d'Ethernet. IP aura comme code 0800 (en hexadécimal) et ARP aura comme code 0806 ;

•les données ;

•un code détecteur d'erreur de type CRC permettant de savoir si la trame a été altérée ou non.

**Le problème**

**Imaginons qu'une machine *A* veuille envoyer un datagramme IP à une machine *B* via un réseau Ethernet. Le datagramme IP sera encapsulé dans une trame Ethernet. *A* va avoir besoin des informations suivantes :**

Sa propre adresse IP. *A* connait son adresse IP car l'administrateur l'a configurée ;

L'adresse IP de *B*. *A* la connait : un serveur DNS lui a fourni l'adresse IP de *B* ;

Sa propre adresse MAC : *A* la connait car elle est écrite sur sa carte Ethernet ;

L'adresse MAC de *B*. *A* ne possède pas cette information ! *A* est incapable d'envoyer une trame Ethernet à *B* sans cette information. Le protocole ARP va permettre à *A* de récupérer cette information.

### 5. Rôle du protocole ARP

## Le protocole ARP, pour Address Resolution Protocol, permet d'obtenir l'adresse MAC d'une machine à partir de son adresse IP.

### 5.1 Le format de la trame ARP

| s (= 4 octets) | | |
|---|---|---|
| p 1 | | p 2 |
| p 3 | p 4 | p 5 |
| p 6 | | |
| p 6 (suite) | | p 7 |
| p 7 (suite) | | p 8 |
| p 8 (suite) | | |
| p 9 | | |

•**champ 1** *(2 octets)* : type de réseau physique. Il s'agit d'un code indiquant la nature du réseau physique. Pour Ethernet le code sera 01 (en hexadécimal)
•**champ 2** *(2 octets)* : il s'agit du code du protocole réseau utilisé. IP aura comme code 0800 (en hexadécimal).
•**champ 3** *(1 octet)* : longueur de l'adresse physique. Le protocole Ethernet utilise des adresses de 48 bits soit 6 octets. Ce champ vaudra donc 6 pour un réseau Ethernet.
•**champ 4** *(1 octet)* : longueur de l'adresse protocole. Le protocole IP utilise des adresse de 32 bits soit 4 octets. Ce champ vaudra donc 4 pour un réseau IP.
•**champ 5** *(2 octets)* : opération indique la nature de l'opération demandée. Une demande ARP aura comme code 01 et une réponse ARP le code 02.
•**champ 6** *(6 octets)* : adresse physique de l'expéditeur.
•**champ 7** *(4 octets)* : adresse protocole de l'expéditeur.
•**champ 8** *(6 octets)* : adresse physique du destinataire. Pour les demandes ce champ a la valeur hexadécimale FF:FF:FF:FF:FF:FF.
•**champ 9** *(4 octets)* : adresse protocole du destinataire.

### 5.2 La table ARP

**Les réponses des différentes demandes ARP sont mémorisées dans une table ARP qui contient les correspondances entre les adresses MAC et IP de différentes machines.**

#### 5.2.1 Un exemple d'échange de trames ARP

**On considère 2 machines *A* et *B* sur un même réseau. *A* souhaite connaître l'adresse MAC de *B* dont il connait l'adresse IP.**

- *A* envoie en broadcast une demande ARP.
- *B* notifie dans sa table ARP la correspondance entre les adresses IP et MAC de *A*.
- *B* répond à *A* en lui transmettant son adresse MAC.
- *A* mémorise les correspondances entre les adresses MAC et IP de *B* dans sa table ARP.

*Les données contenues dans la table ARP ont une validité de 20 minutes.* Une fois ce delai dépassé, il faut refaire une demande ARP.**[20]**

### 6. Le protocole ICMP

**ICMP, pour *Internet Control Message Protocol*, est un protocole de contrôle au niveau de la couche 3 du modèle OSI. À l'instar du protocole IP, il se décline actuellement en versions 4 et 6.**

#### 6.1 Le rôle de protocole ICMP

**Ce protocole assure les rôles suivants :**

- éprouver la connectivité réseau. Par exemple, dans sa version 4, le datagramme ICMP de type 8 (demande d'écho) invite le destinataire à une réponse par un datagramme ICMP de type 0 (réponse à une demande d'écho) ;
- optimiser le réseau, à une certaine échelle ;
- gérer les messages d'erreurs de réseau.

Il n'assure pas d'échange de données proprement dit.

#### 6.2 Le format du paquet ICMP

**Les données du protocole ICMP sont encapsulées dans un paquet IP. Dans ce sens, un paquet ICMP désigne un paquet IP dont la charge utile**

**correspond aux données ICMP.**

Les champs spécifiques à ICMP sont les suivants :

- **Type :** sur 8 bits. Ce champ détermine la nature (ou le type ou la catégorie) du datagramme ICMP.
- **Code :** sur 8 bits. Il indique un sous-type du datagramme ICMP.
- **Checksum - somme de contrôle :** sur 16 bits. Il correspond à un code détecteur d'erreurs pour les données ICMP.
- **Données :** sur 32 bits ou plus. Elles figurent des informations, optionnelles, liées aux type et sous-type de paquet ICMP.

### 6.3 Types de datagramme ICMP

**Les RFC de référence, 792 pour la version 4, 4443 et 4884 pour la version 6, spécifient plusieurs types de paquet ICMP.**

Les paquets ICMP les plus couramment rencontrés et utilisés sont les suivants :

- demande d'écho : Type 8 en version 4, Type 128 en version 6 ;
- réponse à une demande d'écho : Type 0 en version 4, Type 129 en version 6.

## 7. Le protocole UDP

**Le protocole UDP (*User Datagram Protocol*) utilise le protocole IP (adresses source et destinataire) pour l'envoi et la réception de trames de données (*Datagram*).**

Ce protocole n'est pas "fiable" pour différentes raisons :

- Les paquets peuvent être reçus dans un ordre différent de celui utilisé lors de leur envoi. Ceci s'explique par le fait qu'ils peuvent suivre des routes différentes, subir des traitements différents,
- Il n'y a pas d'acquittement ou de retransmission des paquets de données.
- La connexion n'est pas maintenue entre le serveur (émetteur) et le client (récepteur), la transmission des paquets est ponctuelle.

Ce manque de "fiabilité" doit donc être compensé au niveau supérieur (application) en résolvant les points précédents de la manière suivante (par exemple) :

- Afin de reconstituer les données sources, inclure un numéro de paquet pour reconstituer les

données dans l'ordre, ou ajouter une adresse dans le flux de donnée global,

- Afin de vérifier la bonne transmission de chaque paquet, ajouter une somme de vérification (CRC, code de hashage),
- Afin de vérifier la bonne transmission de tous les paquets, le client envoi un paquet d'acquittement pour chaque paquet reçu correctement, l'émetteur qui n'obtient pas cet acquittement dans un temps raisonnable retransmet à nouveau le paquet correspondant.

### 8. Le protocole TCP

**Le protocole TCP (*Tranmission Control Protocol*) résout les problèmes de "fiabilité" du protocole UDP, et permet la transmission de données sous la forme d'un flux d'octets plutôt que sous la forme de paquets.**

#### 8.1 Administration sous Windows

Une station sous Windows NT peut comporter une ou plusieurs interfaces réseau *(intégrées à la carte mère ou pas)*. TokenRing, Ethernet, Wifi...

Les interfaces sont normalement installées physiquement au démarrage par reconnaissance automatique et implantation du driver adéquat *(plug & Play)*. Un ou plusieurs voyants permettent de vérifier si l'interface est bien connectée au système d'interconnexion extérieur *(le plus souvent un commutateur ou switch)*. Si ce n'est pas le cas, rien ne fonctionnera.

Par défaut, une interface réseau comporte son propre identificateur unique, une adresse de couche 2 : adresse MAC (Medium Access Control) sous la forme d'un nombre de 6 octets attribué par le constructeur. L'autosynchronisation est choisie par défaut mais des vitesses de transmission différentes peuvent être appliquées au niveau du driver *(cas particulier)*.

#### 8.2 Configuration TCP/IP

**Pour utiliser les protocoles de la pile TCP/IP, l'administrateur réseau doit attribuer à l'interface une adresse IP unique *(non redondante)* sur le réseau logique auquel elle appartient. c'est l'adressage STATIQUE. Un nombre de 4 octets choisi dans un plan d'adressage prédéfini. Il fournira aussi un masque de sous réseau qui permettra à l'interface de connaître le**

**sous réseau auquel elle appartient. L'adresse de diffusion *(broadcast)* sera la dernière @IP de la plage du sous réseau.**

Pour que cette machine puisse émettre des datagrammes vers d'autres réseaux *(logique et/ou physique)*, l'administrateur devra lui fournir une adresse de**passerelle par défaut** ou default gateway *(une porte de sortie, un routeur connecté au même segment de réseau physique et adressé dans le même réseau logique que la machine qui prendra en charge les datagrammes afin de les envoyer à leur destination)*.

Pour l'accès à Internet par résolution de nom, l'adresse IP d'un **serveur DNS** devra aussi être fournie. *(Plusieurs adresses peuvent être fournies)*

Dans des cas particuliers, on pourra fournir à l'interface plusieurs adresses IP *(sur le même réseau logique ou pas)*.

Exemples d'adressage IpV4 privé sur les trois classe A, B, C

| **sse** | **ue** | **relle par défaut** | |
|---|---|---|---|
| ).1 | .0.0 | 5.255.254 | DNS local ou externe |
| 6.0.1 | 55.0.0 | 6.255.254 | DNS local ou externe |
| 68.0.1 | 55.255.0 | 68.0.254 | DNS local ou externe |

**8.3 Adressage dynamique par DHCP**

**Pour éviter de paramétrer l'interface manuellement, l'adressage dynamique peut aussi être choisi à condition qu'il existe sur le segment de réseau physique au moins un serveur DHCP (Dynamic Host Configuration Protocol) pour répondre à la demande de la machine.**

Dans le cas contraire, c'est par l'intermédiaire d'APIPA (Automatic Private IP Addressing) qu'une adresse IP entre 169.254.0.1 et 169.254.255.254 avec un masque de sous-réseau de 255.255.0.0 sera automatiquement assignée si aucune configuration alternative n'a été spécifiée. APIPA est conçu pour fournir un adressage IP automatique sur des réseaux à segment unique.

**8.4 Paramétrage des interfaces réseaux**

**Le paramétrage sous Windows s'effectue par l'intermédiaire d'une interface graphique dont les modifications seront enregistrées dans la base de registre. Il faut des droits d'administrateur local à la machine pour modifier ces paramètres. Cette interface se trouve dans Menu démarrer >**

## Paramètres > Panneau de configuration > Connexions réseau > *Interface à modifier*

On peut choisir de montrer l'état des interfaces dans la barre de lancement rapide. Les interfaces réseau peuvent être nommées pour en faciliter le repérage.

Une fois les modifications validées, le système ne demande pas de redémarrage comme sous W95, W98.

### 8.5 Test des interfaces

**Pour tester la configuration, on dispose de plusieurs commandes en mode console. Toutes les commandes qui suivent comportent des paramètres d'entrée spécifiques. Pour afficher une aide en ligne tapez la commande suivi de /h pour help.**

```
commande /h
```

•Ouvrir une console en ligne de commande par Menu démarrer > Exécuter > « cmd »

**ipconfig**

**Visualisation des paramètres de toutes les interfaces réseau**

```
ipconfig /all
```

C:>ipconfig /all

Configuration IP de Windows

Nom de l'hôte . . . . . : **MonPC**
Suffixe DNS principal . . . . . :
**Type** de noeud. . . . . : **Hybride**
Routage IP activé . . . . . : **Non**
Proxy WINS activé . . . . . : **Non**

Carte Ethernet Connexion au réseau local :

**Suffixe** DNS propre à la connexion. . . :
**Description.** . . . . . : **Realtek** PCIe GBE Family Controller
Adresse physique . . . . . : **60-EB-69-48-68-C6**
DHCP activé. . . . . : **Oui**
Configuration automatique activée. . . : **Oui**
Adresse IPv6 de liaison locale. . . . .: fe80::89c9:7d53:f73:c8c4%11(préféré)

Adresse IPv4. . . . . .: 192.168.1.2(préféré)
Masque de sous-réseau. . . . . : **255.255.255.0**
Bail obtenu. . . . . : **samedi** 8 septembre 2012 10:30:46
Bail expirant. . . . . : **mardi** 18 septembre 2012 10:30:45

Passerelle par défaut. . . . . : **192.168.1.1**
Serveur DHCP . . . . . : **192.168.1.1**
IAID DHCPv6 . . . . . : **234890910**
DUID de client DHCPv6. . . . . : **00-01-00-01-15-E2-70-88-60-EB-69-48-68-C6**
Serveurs DNS. . . . . : **212.27.40.240**
212.27.40.241
NetBIOS sur Tcpip. . . . . : **Activé**

**ping**

**Test de l'interface de bouclage**

```
ping 127.0.0.1
```

•Test de la passerelle

```
ping "@ip_passerelle"
```

•Test de la connectivité Internet

```
ping "@ip_externe"
```

**route**

**Visualisation de la table de routage**

```
route print
```

```
C:>route print
===========================================================================
===========
Liste d'Interfaces
 11...60 eb 69 48 68 c6 ......Realtek PCIe GBE Family Controller
  1...........................Software Loopback Interface 1
 17...00 00 00 00 00 00 00 e0 Carte Microsoft ISATAP
 12...00 00 00 00 00 00 00 e0 Carte Microsoft 6to4
 16...00 00 00 00 00 00 00 e0 Teredo Tunneling Pseudo-Interface
===========================================================================
===========

IPv4 Table de routage
===========================================================================
===========
Itinéraires actifs :
Destination réseau    Masque réseau  Adr. passerelle   Adr. interface Métrique
          0.0.0.0          0.0.0.0      192.168.1.1      192.168.1.2     20
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    306
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    306
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    306
      192.168.1.0    255.255.255.0         On-link       192.168.1.2    276
      192.168.1.2  255.255.255.255         On-link       192.168.1.2    276
    192.168.1.255  255.255.255.255         On-link       192.168.1.2    276
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    306
        224.0.0.0        240.0.0.0         On-link       192.168.1.2    276
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    306
```

```
  255.255.255.255  255.255.255.255        On-link       192.168.1.2    276
===========================================================================
===========
Itinéraires persistants :
  Aucun

IPv6 Table de routage
===========================================================================
===========
Itinéraires actifs :
 If Metric Network Destination      Gateway
 16     58 ::/0                     On-link
  1    306 ::1/128                  On-link
 16     58 2001::/32                On-link
 16    306 2001:0:5ef5:73b8:863:2cbf:a75b:7680/128
                                    On-link
 11    276 fe80::/64                On-link
 16    306 fe80::/64                On-link
 16    306 fe80::863:2cbf:a75b:7680/128
                                    On-link
 11    276 fe80::89c9:7d53:f73:c8c4/128
                                    On-link
  1    306 ff00::/8                 On-link
 16    306 ff00::/8                 On-link
 11    276 ff00::/8                 On-link
===========================================================================
===========
Itinéraires persistants :
  Aucun

C:>
```

•Ajout

```
route add
```

•Suppression

```
route delete
```

**tracert**

**Test de routage**

```
tracert "@ip_externe"
```

```
C:>tracert fr.wikibooks.org

Détermination de l'itinéraire vers wikibooks-lb.esams.wikimedia.org [91.198.174.228] avec
un maximum de 30 sauts :

  1    <1 ms    <1 ms     1 ms  192.168.1.1
  2    22 ms    21 ms    22 ms  88.164.137.254
```

```
 3    21 ms    23 ms    20 ms  78.254.2.190
 4    22 ms    25 ms    25 ms  sto93-1-v816.intf.nra.proxad.net [78.254.251.133]
 5    25 ms    25 ms    24 ms  cbv-6k-2-po11.intf.nra.proxad.net [78.254.255.85]
 6    25 ms    24 ms    24 ms  th2-crs16-1-be1013.intf.routers.proxad.net [212.27.59.10]
 7    30 ms    30 ms    30 ms  strasbourg-crs16-1-be2000.intf.routers.proxad.net
[212.27.50.10]
 8    32 ms    35 ms    34 ms  francfort-6k-1-po100.intf.routers.proxad.net [212.27.56.30]
 9     *       38 ms     *     amsterdam-6k-1-po100.intf.routers.proxad.net [212.27.56.38]
10    38 ms    41 ms    38 ms  xe-1-1-0.cr2-knams.wikimedia.org [195.69.145.176]
11    39 ms    40 ms    41 ms  ve7.te-8-1.csw1-esams.wikimedia.org [91.198.174.250]
12    41 ms    41 ms    40 ms  wikibooks-lb.esams.wikimedia.org [91.198.174.228]

Itinéraire déterminé.

C:>
```

**netstat**

•visualisation des services actifs

```
netstat /a

C:>netstat -a

Connexions actives

  Proto  Adresse locale        Adresse distante       État
  TCP    0.0.0.0:7             MonPC:0        LISTENING
  TCP    0.0.0.0:9             MonPC:0        LISTENING
  TCP    0.0.0.0:13            MonPC:0        LISTENING
  TCP    0.0.0.0:17            MonPC:0        LISTENING
  TCP    0.0.0.0:19            MonPC:0        LISTENING
  TCP    0.0.0.0:80            MonPC:0        LISTENING
  TCP    0.0.0.0:135           MonPC:0        LISTENING
  TCP    0.0.0.0:443           MonPC:0        LISTENING
  TCP    0.0.0.0:445           MonPC:0        LISTENING
  TCP    0.0.0.0:554           MonPC:0        LISTENING
```

**[21]**

Etude et élaboration d'une plate-forme télé-ophtalmologique

# Chapitre V : Elaboration de l'application sous Visual Basic

# I- Visual Basic

## 1. Historique du langage

- En 1965, John Kemeny et Thomas Kurtz, développent, au Collège Darmouth, le langage **Basic** ***(Beginners All-Purpose Symbolic Instruction Code)*** dans le but d'enseigner à leurs étudiants, un langage simple, facile à apprendre, mais assez puissant.
- En 1975, Bill Gates et Paul Allen développent une version de **Basic** spécialement pour les ordinateurs personnels **« Altair »**. Suite au succès de cette version de **Basic**, Bill Gates et Paul Allen forment la compagnie **Microsoft Corporation**.

  En mai 1991, la version 1.0 de **Visual Basic** fait son apparition sur le marché.

  En 1992, la version 2.0 fait son apparition.
- En 1993, la version 3.0, déjà plus stable, assure le succès de ce langage de programmation.
- En 1995, Microsoft sort la version 4.0 en même temps que son nouveau système d'exploitation Windows 95.
- En 1997, la version 5.0 fait son apparition.
- En 1998, la version 6.0 fait son apparition.
- En 2001, développe sa 7e version sous le nom de VisualBasic.net. Cette nouvelle version offre de nombreux changements comparativement aux méthodes précédentes de programmation.
- En 2005, Microsoft sort la version Visual Basic 2005, une version qui intègre la technologie de VisualBasic.net.
- En 2006, afin de permettre à un plus grand nombre d'amateur de programmation de développer des programmes en utilisant les logiciels de Microsoft, cette compagnie développe une version moins complète mais disponible gratuitement sur Internet, la version Visual Basic 2005 Edition Express.
- En 2008, Microsoft sort la version Visual Basic 2008. **[22]**

## 2. Introduction a Visual Basic

Microsoft Visual Basic est un moyen rapide et facile de créer des programmes pour Microsoft Windows. Même si vous débutez en programmation Windows, Visual Basic vous fournit un jeu complet d'outils qui simplifie le développement.

Mais qu'est-ce que Visual Basic ? Le terme « Visual » (visuel) fait référence à la méthode utilisée pour créer ce que l'utilisateur voit : l'*interface utilisateur graphique*, ou GUI (Graphical User Interface). "Basic" (de base) fait référence au langage de programmation BASIC (Beginners All-Purpose Symbolic Instruction Code), le langage le plus utilisé dans l'histoire de l'informatique. Vous pouvez créer des programmes utiles en apprenant simplement quelques-unes de ses fonctionnalités. Les liens suivants vous initieront à la programmation en Visual Basic ; chacun d'eux inclut des exemples et donne accès à des informations supplémentaires.

Visual Basic est conçu pour générer des applications de type sécurisé et orientées objet de manière productive. Visual Basic permet aux développeurs de cibler des périphériques Windows, Web et mobiles. Comme avec tous les langages ciblant le Microsoft .NET Framework, les programmes écrits en Visual Basic bénéficient de la sécurité et de l'interopérabilité entre les langages.

Visual Basic prend en charge l'opération Modifier & Continuer et propose des fonctionnalités pour le développement rapide d'applications. L'une de ces fonctionnalités, appelée **My**, donne un accès rapide aux tâches courantes proposées par le .NET Framework, ainsi qu'aux informations et instances d'objet par défaut liées à l'application et à son environnement d'exécution. Les nouvelles fonctionnalités de langage comptent notamment la continuation de boucle, la mise à disposition de ressources garanties, la surcharge d'opérateur, les types génériques et les événements personnalisés. Visual Basic intègre également l'intégralité du .NET Framework et du Common Language Runtime, qui offrent ensemble l'interopérabilité entre les langages, le garbage collection, une sécurité améliorée et une meilleure prise en charge du versioning.

### 3. Types de données

Les types de données en Visual Basic déterminent le type de valeurs ou de données qui peut être stocké dans une variable, ainsi que la manière dont ces données sont stockées.

Pourquoi existe-t-il différents types de données ?

Envisagez la question de la manière suivante : supposons trois variables, dont deux stockent des nombres tandis que la troisième stocke un nom : si les deux premières vous permettent d'appliquer une opération arithmétique, la troisième ne vous le permet pas. Affecter un type

de données à une variable permet de déterminer plus facilement la manière dont la variable peut ou non être utilisée.

La plupart des logiciels traitent des nombres d'une manière ou d'une autre. Sachant qu'il existe plusieurs façons d'exprimer des nombres, Visual Basic dispose de plusieurs types de données numériques pour traiter plus efficacement les nombres.

Le type de données numérique que vous utilisez le plus est le type **Integer**. Il sert à représenter un nombre entier (un nombre sans partie décimale). Lors du choix d'un type de données pour représenter des nombres entiers, utilisez de préférence le type de données **Long** si votre variable est amenée à stocker des nombres supérieurs à environ deux milliards ; dans la négative, un type **Integer** est plus efficace.

Tous les nombres ne sont pas des entiers ; par exemple, lorsque vous divisez deux nombres entiers, le résultat est souvent un nombre entier suivi d'une valeur décimale (9 divisé par 2 est égal à 4,5). Le type de données **Double** est utilisé pour représenter des nombres qui ont une partie décimale.

**Remarque**

Il existe d'autres types de données numériques, tels que **Decimal**, **Short**, **SByte** et **UInteger** ; ceux-ci sont généralement utilisés dans les programmes de très grande envergure où l'exploitation de la mémoire ou la vitesse pose problème. Pour le moment, les types de données numériques de base répondront à tous vos besoins.

La plupart des programmes traitent également du texte, qu'il s'agisse de présenter des informations à l'utilisateur ou de capturer le texte saisi par celui-ci. Le texte est généralement stocké dans le type de données **String** qui peut contenir une série de lettres, de nombres, d'espaces ou d'autres caractères. Un type **String** n'est pas limité en longueur : il peut s'agir d'une phrase, d'un paragraphe ou d'un caractère unique, voir de l'absence de caractère (une *chaîne NULL*).

Dans le cas d'une variable qui représentera systématiquement un seul caractère, il existe également le type de données **Char**. Si vous avez besoin qu'une variable unique ne stocke qu'un seul caractère, utilisez le type de données **Char** plutôt que le type **String**.

En plus du texte et des nombres, les programmes sont parfois amenés à stocker d'autres types d'informations, tels qu'une valeur vraie ou fausse, une date, ou encore des données ayant une signification particulière pour le programme.

Pour les valeurs à représentation de type vrai/faux, oui/non, ou activé/désactivé, Visual Basic dispose du type de données **Boolean**. Une variable **Boolean** peut renfermer une valeur sur deux possibles : **True** ou **False**.

Bien que vous puissiez représenter des dates ou des heures sous la forme de nombres, le type de données **Date** facilite le calcul des dates ou des heures, notamment le nombre de jours jusqu'à votre anniversaire ou le nombre de minutes jusqu'au déjeuner.

Lorsque vous devez stocker plusieurs types de données dans une seule variable, vous pouvez utiliser un type de données *composite*. Les types de données composites comprennent des *tableaux*, des *structures* et des *classes*. Vous en saurez plus sur ces éléments dans les leçons ultérieures.

Enfin, le type des données que vous devez stocker peut être différent selon les circonstances. Le type de données **Object** vous permet de déclarer une variable, puis d'en définir le type de données ultérieurement.

### 4. Utilisation de variables chaîne pour organiser des mots

Une *chaîne* (en anglais, string) est une série de caractères de texte, tels que lettres, nombres, caractères spéciaux et espaces. Les chaînes peuvent être expressions ou des phrases explicites, telles que « Rapide, le renard brun saute sur le chien paresseux », ou des combinaisons d'apparence inintelligible, telles que « @#fTWRE^3 35Gert ».

Les variables **String** sont créés comme les autres variables : elles sont d'abord déclarées puis se voient affecter une valeur, comme illustré ci-dessous.

VB

```
Dim aString As String = "This is a string"
```

Lors de l'affectation du texte proprement dit (également appelé *littéral de chaîne*) à une variable **String**, celui-ci doit figurer entre guillemets (""). Vous pouvez également utiliser le caractère = pour affecter une variable **String** à une autre variable **String**, comme l'illustré dans cet exemple.

VB

```
    Dim aString As String = "This is a string"
...
    Dim bString As String = ""
    bString = aString
```

Le code précédent attribue une valeur identique à `bString` et `aString` (`This is a string`).

Vous pouvez utiliser le caractère de perluète (`&`) pour combiner séquentiellement plusieurs chaînes au sein d'une nouvelle chaîne, comme indiqué ci-dessous.

VB

```
Dim aString As String = "Across the Wide"
Dim bString As String = "Missouri"
Dim cString As String = ""
cString = aString & bString
```

L'exemple précédent déclare trois variables **String** et affecte respectivement les valeurs "Across the Wide" et "Missouri" aux deux premières, puis affecte les valeurs combinées des deux premières à la troisième. D'après vous, qu'elle sera la valeur de `cString` ? Vous serez peut-être surpris d'apprendre que la valeur est `Across the WideMissouri`. En effet, il n'y a aucun espace ni à la fin de `aString`, ni au début de `bString`. Les deux chaînes sont simplement mises bout à bout. Si vous souhaitez intercaler un élément entre les deux chaînes, notamment un espace, vous devez le faire avec un *littéral de chaîne*, tel que `" "`, comme indiqué ci-dessous.

```
Dim aString As String = "Across the Wide"
Dim bString As String = "Missouri"
Dim cString As String = ""
cString = aString & " " & bString
```

Le texte contenu dans `cString` présente désormais l'aspect suivant : `Across the Wide Missouri`.

## 5. Ecriture de procédures

Si Visual Basic intègre de nombreuses procédures pour exécuter des actions communes, il y a toujours des cas où vous souhaitez que votre programme exécute une action qu'une procédure intégrée ne peut pas gérer. Par exemple, la fonction **MsgBox** ne peut afficher aucune boîte de dialogue dotée d'une image. Pour effectuer cette tâche, vous devez écrire votre propre procédure.

**Qu'est-ce qu'une procédure ?**

Une procédure est un bloc de code autonome qui peut être exécuté à partir d'autres blocs de code. En général, les procédures contiennent chacune le code nécessaire pour accomplir une

tâche. Par exemple, vous pouvez disposer d'une procédure appelée **PlaySound** qui contient le code nécessaire à la lecture d'un fichier .wav. Même si vous pouvez écrire le code pour lire un son chaque fois que votre programme doit en émettre un, il est plus rationnel de créer une procédure unique que vous pouvez appeler depuis n'importe quel emplacement dans votre programme.

Une procédure s'*exécute* en *l'appelant* dans un code. Par exemple, pour exécuter la procédure **PlaySound**, vous devez simplement ajouter une ligne de code à votre programme avec le nom de votre procédure, comme indiqué ci-dessous.

```
PlaySound
```

Et c'est tout ! Lorsque le programme atteint cette ligne, il accède à la procédure **PlaySound** et exécute le code qu'il trouve à son emplacement. Le programme revient ensuite à la ligne qui suit celle de l'appel à **PlaySound**.

Vous pouvez appeler autant de procédure que vous le souhaitez. Les procédures s'exécutent dans l'ordre d'appel. Par exemple, si vous disposez également d'une procédure appelée **DisplayResults**, l'exécuter après avoir exécuté la procédure **PlaySounds**, appelle ces procédures conformément au code ci-dessous.

```
PlaySounds
DisplayResults
```

**Fonctions et sous-routines**

Il existe deux types de procédures : les *fonctions* et les *sous-routines* (parfois appelées *sub*). Une fonction retourne une valeur à la procédure qui l'a appelée, tandis qu'une sous-routine se contente d'exécuter un code. Les sous-routines sont appelées lorsqu'une ligne de code qui contient leurs noms est ajoutée au programme, comme dans l'exemple suivant.

```
DisplayResults
```

Les fonctions sont différentes, car non seulement elles exécutent le code, mais elles retournent également une valeur. Par exemple, imaginez une fonction appelée **GetDayOfWeek** qui retourne un **Integer** qui indique le jour de la semaine. Vous appelez cette fonction en déclarant d'abord une variable pour stocker la valeur retournée, puis en assignant la valeur retournée à la variable pour l'utiliser ultérieurement, comme illustré ci-dessous.

```
Dim Today As Integer
Today = GetDayOfWeek
```

Dans cet exemple, la valeur retournée par la fonction est copiée dans la variable nommée `Today`, puis enregistrée à des fins d'utilisation ultérieure.

**Écriture de procédures**

Vous écrivez des procédures en écrivant en premier lieu une *déclaration de procédure*. Une déclaration de procédure a plusieurs objectifs : elle déclare si la procédure est une fonction ou une sous-routine, nomme la procédure, et détaille tous les paramètres éventuels de la procédure (les paramètres seront examinés en détail dans cette leçon). L'exemple suivant est un exemple de déclaration de procédure simple.

VB

```
Sub MyFirstSub()
End Sub
```

Le mot clé `Sub` explique au programme que cette procédure est une sous-routine et ne retournera aucune valeur. Le nom de la sous-routine (`MyFirstSub`) vient ensuite. Les parenthèses vides indiquent l'absence de *paramètres* dans cette procédure. Enfin, le mot clé **End Sub** indique la fin de la sous-routine. Tout le code qui sera exécuté par cette sous-routine se place entre ces deux lignes.

Déclarer des fonctions est similaire, à ceci près que vous devez spécifier le type de la valeur retournée (par exemple, **Integer** ou **String**). Par exemple, une fonction qui a retourné un **Integer** peut ressembler à ce qui suit.

VB

```
Function MyFirstFunction() As Integer
End Function
```

Les mots clés `As Integer` indiquent que cette fonction retourne une valeur de type **Integer** . Pour retourner une valeur à partir d'une fonction, utilisez le mot clé **Return**, comme illustré dans l'exemple suivant.

VB

```
Function GetTheNumberOne() As Integer
    Return 1
End Function
```

Cette procédure retourne le nombre 1.**[23]**

# II. VB.NET, un langage objet

## 1. C'est quoi, un langage objet ?

VB.NET est un langage objet. Ce n'est pas le seul : C# (prononcez "*écharpe*"), C++ ou Java sont eux aussi des langages objets. Ce type de langage, apparu dans les années 1990, s'oppose en particulier aux langages de la génération précédente, qui étaient dits "procéduraux" (comme le Cobol, le Fortran, le Pascal, le C et bien d'autres, on ne va pas tous les citer, on n'a pas que cela à faire non plus).

La première chose à dire des langages objet, c'est qu'ils permettent de faire tout, absolument tout, ce qu'on pouvait faire avec un langage procédural.

Donc, et c'est la bonne nouvelle du jour, rien de ce que vous avez appris n'est perdu ! Les langages objet, comme leurs ancêtres les langages procéduraux, manipulent des variables, des tableaux et des structures, et organisent des instructions d'affectation, d'entrée-sortie, de tests ou de boucles dans des procédures et des fonctions. Donc, à la limite, on peut (presque) programmer en langage objet exactement comme on programmait en langage procédural. Mais évidemment, ce serait bien dommage, car on passerait alors à côté de ces petits riens que les langages objet possèdent en plus, petits riens qui vont nous changer la vie...

Les langages objet, tout en intégrant l'ensemble des capacités des langages procéduraux, possèdent deux aptitudes supplémentaires, aptitudes liées mais distinctes. Ces langages peuvent en effet :

- gérer, en plus des variables, des choses appelées objets.
- exploiter, via certains de ces objets, l'interface graphique de Windows.

On verra plus tard que par contamination, les langages objet ont tendance à tout assimiler à des objets, même des choses qui n'en sont pas spécialement.

### 1.1 Qu'est-ce qu'un objet ? Parlons comme les gens normaux

Dans la vie de tous les jours, nous sommes entourés d'objets. Certains très simples, comme un peigne, ou une matraque de CRS. D'autres très complexes comme une automobile, un ordinateur ou une navette spatiale. Pour faire l'analogie avec les objets des langages objet, mieux vaut penser à un objet un peu compliqué qu'à un objet trop simple. Et pour cette analogie, rien ne nous oblige à penser à un objet inanimé. Des plantes, des animaux ou des êtres humains peuvent tout à fait être assimilés aux objets que manipulent les langages.

Dans les objets, il y a deux sortes de choses qui intéressent les informaticiens.

- ils ont des caractéristiques (une couleur, une taille, un poids, un compte en banque, un nombre de pattes, bref, que sais-je encore). Un objet très simple peut avoir une seule caractéristique (voire aucune, mais là, on peut raisonnablement se demander s'il existe - je laisse cette question aux philosophes désoeuvrés). Un objet complexe peut posséder des centaines, voire des milliers de caractéristiques.
- ils peuvent accomplir certaines tâches, ou subir certaines modifications : pour une automobile, on peut ouvrir une porte, démarrer le moteur, freiner, accélérer, etc. Pour un chat, on peut le caresser, le faire marcher, le faire courir, le faire miauler, le faire cuire, etc. Pour un employé, on peut le changer de poste, de qualification, l'augmenter (c'est rare), le licencier (c'est fréquent), etc.

Les langages objet, ce sont des langages qui permettent, en plus des traditionnels variables, tableaux et structures, de gérer et de manipuler des objets (donc des classes). C'est-à-dire que ces langages permettent notamment :

- de créer des classes, en définissant leurs caractéristiques et leurs actions
- de fabriquer des objets à partir de ces classes
- de manipuler les caractéristiques individuelles de chacun de ces objets, et de leur faire accomplir les actions dont leur classe est capable.

Tout ce que nous avons vu là peut être traduit dans des termes techniques propres aux informaticiens.

## 1.2 Qu'est-ce qu'un objet ? Parlons comme les informaticiens

Nous savons que tout programme informatique a pour but de manipuler des informations. Et nous savons que ces informations relèvent toujours, au bout du compte, de l'un des trois grands types simples : numériques, caractères (alphanumériques), ou booléens.

Lorsque nous disons qu'un objet (ou une classe) possède des caractéristiques comme la taille, le poids, la couleur, etc., nous disons en fait qu'un objet (ou une classe) regroupe un certain nombre de variables : une variable numérique pour stocker le poids, une autre variable numérique pour stocker la taille, une variable caractère pour stocker la couleur, etc.

Donc, première chose, un objet regroupe un ensemble de variables qui peuvent être de différents types. Et dans ce cas, on ne parlera pas des "variables" d'un objet, ni des ses "caractéristiques", mais de ses propriétés (ou encore, de ses attributs).

**Définition :**

Une propriété d'un objet est une des variables (typée) associées à cet objet.

**Théorème :**

Par conséquent, toute instruction légitime avec une variable est légitime avec une propriété d'un objet. Et lycée de Versailles, tout ce qui n'avait pas de sens avec une variable n'a pas non plus de sens avec une propriété d'un objet.

Mais jusqu'à maintenant, allez-vous dire, nous n'avons fait que faire de la mousse avec du vocabulaire nouveau autour d'une notion que nous connaissons déjà comme notre poche. Parce qu'enfin, quoi, un bidule constitué de différentes variables de différents types, on n'a pas attendu les objets pour en avoir. Tous les langages procéduraux connaissent cela, mis à part qu'on ne parle pas de classe mais de structure, qu'on ne parle pas d'objets, mais de variables structurées, et qu'on ne parle pas de propriétés, mais de champs. Mais pour le reste, c'est du pareil au même.

Les esprits chagrins qui feraient cette remarque auraient parfaitement raison... si les objets n'étaient effectivement qu'un ensemble de caractéristiques (de propriétés). Mais les objets ne sont pas que cela : ils incluent également, comme on l'a vu, des actions possibles. Autrement dit, un objet (une classe) est un assemblage d'informations (les propriétés) mais aussi d'instructions (les actions, que l'on appelle les méthodes). C'est cette présence des méthodes qui différencie une classe d'une simple structure, et un objet d'une simple variable structurée. On peut donc dire qu'un objet, c'est une structure plus des procédures.

**Définition :**

Une méthode est une procédure (donc un ensemble d'instructions) associée à un objet (et à sa classe).

Notons dès à présent que les classes (donc, les objets) peuvent aussi être associés à ce que l'on appelle des événements, événements qui joueront un rôle essentiel dans le déroulement de notre application.

Nous mettons donc dès à présent le doigt sur la différence profonde qui existe entre un langage procédural traditionnel et un langage objet : il s'agit de la manière dont sont articulées les informations et les traitements permettan de les modifier.

- dans un langage procédural, il existe une **séparation** rigoureuse entre les informations et les traitements. Au cours de l'exécution d'un programme, les informations

sont stockées des conteneurs spécifiques (variables, tableaux, structures) et les traitements (instructions) agissent sur ces informations, si j'ose dire, « du dehors ».

- dans un langage objet, on a **regroupé** dans une même entité (la classe, donc les objets qui en sont issus) les propriétés et les méthodes, c'est-à-dire les informations et les instructions qui permettent de les manipuler.

## 2. La syntaxe objet

### 2.1 Généralités

Lorsque nous fabriquions (et nous en fabriquerons encore) une variable, nous utilisions un « moule » à variable, qui s'appelle un type. Pour créer une nouvelle variable, il fallait lui donner un nom, par lequel on pourrait la désigner tout au long du programme, et préciser le type utilisé pour la fabrication de cette variable. D'où la syntaxe du genre :

Dim Toto As Integer

Eh bien, en ce qui concerne les objets, le principe est exactement le même, hormis qu'on ne les fabrique pas avec un type, mais avec une classe. Si nous disposons par exemple d'une classe **Chien**, nous allons pouvoir créer autant de chiens individuels que nous voulons. Voilà par exemple un nouveau compagnon :

Dim Rantanplan As New Chien

On remarque la présence du mot clé New, propre aux déclarations d'objets, et qui différencie celles-ci des déclarations de variables. Dans le jargon du langage objet, cette instruction New s'appelle un constructeur.

Vocabulaire savant :

Créer un objet d'après une classe s'appelle instancier la classe.
Un objet peut aussi être appelé une instance, ou une occurrence, de la classe.

Notre classe **Chien** a forcément été créée avec un certain nombre de propriétés et de méthodes. Pour chaque objet instancié selon cette classe, nous pouvons accéder à ces propriétés et à ces méthodes par la syntaxe suivante, universelle :

Nom-d'objet.Nom-de-propriété

Et également :

Nom-d'objet.Nom-de-Méthode

**Remarque** :

Dans un langage objet, on ne peut trouver que les deux syntaxes ci-dessus.

Sauf dans des cas exceptionnels, il est impossible de désigner un objet sans le faire suivre d'une propriété ou d'une méthode. C'est une faute de syntaxe.
De même, on ne trouvera jamais une propriété ou une méthode seules, non précédées de l'objet auxquelles elles se réfèrent.

## 2.2 Propriétés

Admettons pour les besoins de la cause que notre classe **Chien** possède entre autres propriétés :

- **Taille** (numérique)
- **Poids** (numérique)
- **Couleur** (caractère)
- **Vacciné** (booléen)

Alors,les règles d'utilisation de ces propriétés sont très exactement les mêmes que celles des variables. Sachant que le signe d'affectation, en Visual Basic, est le signe d'égalité, je peux donc écrire :

Rantanplan.Poids = 14

Et je viens de fixer le poids de mon toutou à 14 kg. Pour le faire grossir de 5 kg, rien de plus simple :

Rantanplan.Poids = Rantanplan.Poids + 5

...et le tour est joué. Si je veux torturer la pauvre bête, je peux aussi la rendre aussi lourde que haute, en écrivant :

Rantanplan.Poids = Rantanplan.Taille

L'utilisation de propriétés non numériques ne pose pas davantage de problèmes :

Rantanplan.Couleur = "Beige"

Rantanplan.Vacciné = True

### 2.3. Méthodes

Si les propriétés sont en réalité des variables (et, pour être, encore plus précis, des champs de structures), les méthodes sont quant à elles des procédures. Utiliser convenablement une méthode suppose donc de respecter la manière dont la procédure a été écrite. En particulier, comme à chaque fois en pareil cas, il va falloir respecter le nombre et le type des paramètres dont la méthode (la procédure) a besoin pour s'exécuter. Ainsi, certaines méthodes peuvent-elles être utilisées sans passer de paramètres. D'autres en exigeront un, d'autres deux, etc.

Dans tous les cas, cependant, une méthode étant un appel de procédure, elle représente donc en elle-même une instruction à part entière. Il serait donc parfaitement absurde d'affecter une méthode comme on affecte une propriété. Les appels aux méthodes pouront donc donner lieu à des lignes du genre :

Rantanplan.Aboyer

...si la méthode **Aboyer** s'emploie sans arguments,

Rantanplan.Courir(15)

...si la méthode **Courir** nécessite un argument numérique, celui fixant la vitesse de course,

Rantanplan.DonnerlaPapatte("gauche")

...si la méthode **DonnerlaPapatte** réclame un argument de type caractère, etc.

### 3. La programmation événementielle

Rappelons-nous comment notre code était organisé dans un langage procédural traditionnel. Plutôt qu'une immense procédure fait-tout, comportant des redites de code, nos applications, dès lors qu'elles devenaient un peu joufflues, étaient organisées en modules séparés : sous-procédures et fonctions (lesquelles, je le rappelle, ne sont jamais qu'un cas particulier de sous-procédures).

Parmi ces diverses procédures, il s'en trouvait une qui jouait un rôle particulier : la procédure principale (appelée **Main** dans la plupart des langages). C'est elle qui était exécutée lors du lancement de l'application. Et c'est elle qui tout au long du programme, telle un chef d'orchestre, déclenchait les autres procédures et les autres fonctions.

Le point important dans cette affaire, c'est qu'une fois l'application lancée, une procédure donnée ne pouvait s'exécuter que si une ligne de code, quelque part, en commandait le déclenchement. L'utilisateur n'avait aucune espèce de moyen d'influencer directement l'ordre dans lequel les procédures allaient être exécutées.

Tout ceci va être remis en question avec les langages objet. Non qu'on ne puisse plus créer encore des procédures et des fonctions, et appeler les unes via les autres par des instructions adéquates ; je le rappelle, il n'y a rien qu'on pouvait faire avec un langage procédural qu'on ne puisse faire avec un langage objet. Mais avec un langage objet, on va pouvoir faire certaines choses véritablement inédites. En l'occurrence, la grande nouveauté, c'est qu'on va pouvoir organiser le déclenchement automatique de certaines procédures en réponse à

certaines actions de l'utilisateur sur certains objets. Et ces actions, ce sont les fameux événements dont je parlais tout à l'heure.

Pour comprendre cela, le plus simple est de penser que certains objets ont une caractéristique particulière : ils se voient à l'écran. Un système d'exploitation tel que Windows est truffé de ce genre d'objets ! Les boutons, les menus, les fenêtres, les cases à cocher, tout cela constitue une armada d'objets visibles, et surtout, capables de réagir à diverses sollicitations de l'utilisateur via le clavier ou la souris. Les programmeurs qui ont écrit Windows ont donc prévu, et écrit, des myriades de morceaux de code (des procédures) qui se déclenchent à chaque fois que l'utilisateur accomplit telle action sur tel type d'objet, par exemple, le clic sur un bouton, le duble-clic sur un raccourci, etc. Donc, je le répète, concevoir une application événementielle, c'est concevoir des procédures qui se déclencheront automatiquement à chaque fois que l'utilisateur effectuera telle action sur tel objet qu'on aura mis à sa disposition.

## 3.1 Le diagramme T.O.E.

Lors de la conception d'un programme objet, une des premières tâches du développeur va donc être de concevoir l'interface utilisateur. C'est-à-dire de prévoir quels objets vont être mis à sa disposition à l'écran, quelles seront les actions qu'il pourra effectuer sur ces objets (c'est-à-dire les événements qu'il pourra déclencher) et de prévoir les tâches que le programme devra accomplir lors de ces événements.

**Remarque incidente**

Pour le moment, nous nous contenterons de considérer que nos applications emploient uniquement l'arsenal des objets utilisés par Windows. Ces objets (plus exactement, ces classes) seront donc tout prêts à l'emploi. Leurs propriétés, leurs méthodes et les événements qu'ils sont capables de gérer ont été définis par Windows, c'est-à-dire par Visual Basic (qui nous sert en quelque sorte d'intermédiaire vis-à-vis de Windows, qui parle un langage trop abscons pour nous).

En réalité, dans un programme Visual Basic, on peut utiliser des tas d'autres objets (classes) et on peut même en fabriquer soi-même. Nous verrons cela... plus tard. Pour le moment, on se contentera de la trousse à outils que Visual Basic met à notre disposition, ce qui suffira déjà à nous procurer de longues heures de bonheur.

Ainsi, nous pouvons prévoir qu'il faudra que la fenêtre de notre application se présente comme ceci ou comme cela, qu'il devra y avoir ici un bouton, là des boutons radios, ici une zone de liste, etc. Et il faut également prévoir qu'un clic sur tel bouton déclenchera tel ou tel calcul, qu'un choix dans telle liste devra mettre à jour telle ou telle zone, etc.

Le document synthétique, sous forme de tableau, qui reprend toutes ces informations, s'appelle un diagramme T.O.E., pour Tâche, Objet, Événement. C'est un document préalable indispensable à la programmation d'une application événementielle. Il comporte trois colonnes. Dans la première, on dresse la liste exhaustive des tâches que doit accomplir l'application. Dans la seconde, en regard de chaque tâche à accomplir, on porte, le cas échéant, le nom de l'objet qui en sera chargé. Et dans la troisième, en regard de chaque objet, on écrit le cas échéant, mais oui, l'événement correspondant.

**Remarque importante**

Rien, absolument rien, n'impose qu'un objet soit associé à un événement et à un seul. Certains objets peuvent fort bien n'être associés à aucun événement. De même, d'autres objets peuvent très bien gérer plusieurs événements différents, qui correspondront donc à autant de procédures différentes.

### 3.2 La syntaxe

Commençons par le commencement. Un évènement, en Visual Basic, est formé de la réunion, séparée par un point, des deux éléments suivants :

Nom-d'objet.Nom-d'action

Par exemple, l'événement « l'utilisateur clique sur le bouton du nom de BoutonOK » correspond à :

BoutonOK.Click

Bien sûr, a priori, le nom des évènements ne s'invente pas. Lorsqu'on utilise des classes toutes prêtes, on repère dans l'aide du langage quelles actions peuvent être associées à quelles classes, et donc aux objets instanciés d'après ces classes. Par exemple, dans Windows, la classe des boutons ne gère pas l'action « double clic ». Essayez, dans n'importe quel logiciel sous Windows, de double-cliquer sur un bouton, et vous verrez que c'est à peu près aussi facile que d'éternuer sans fermer les yeux. Donc, si nous employons dans un programme un objet de classe bouton, il nous sera impossible de lui associer une action

double-clic pour gérer l'événement correspondant. Consolez-vous néanmoins, il y a largement, très largement, de quoi faire avec les classes et les actions existantes, et nous serons très loin de pouvoir en épuiser les possibilités !

Passons maintenant au deuxième étage de la fusée. Comment la machine établit-elle la liaison entre une procédure et l'événement qui la déclenche ? C'est extrêmement simple : cette liaison figure en toutes lettres dans l'en-tête de la procédure, via le mot-clé Handles (que l'on peut traduire par "gère"), suivi de l'évènement concerné. Ainsi, une procédure non événementielle, toute bête, toute simple, nommée **Calcul**, aura la tête suivante :

```
Private Sub Calcul()

                                                                    ...
  instructions
                                                                    ...
End Sub
```

Pour que cette procédure s'exécute à chaque fois que l'on clique sur le bouton nommé **Résultat**, il suffira qu'elle se présente ainsi :

```
Private Sub Calcul() Handles Résultat.Click

                                                                    ...
  instructions
                                                                    ...
End Sub
```

**Remarque :**

La **seule** chose qui compte dans la liaison évènement / procédure, c'est... **l'évènement** (qui figure après le mot-clé **Handles**).

Le **nom** de la procédure (ce qui suit **Private Sub**), lui, ne joue **aucun rôle**. On peut le modifier à loisir, cela ne change strictement rien.

Comprendre ceci est d'autant plus important que VB, dans sa bonté naturelle, a tendance à donner aux procédures un titre sur le modèle « NomObjet_NomAction » (par exemple, BoutonOK_Click). On a tôt fait de prendre le leurre pour la proie, et de se mélanger complètement les crayons entre ce qui a une importance vitale (l'évènement qui suit Handles) et ce qui n'en a aucune (le nom de la procédure).

Si d'aventure, une même procédure doit être déclenchée par plusieurs événements différents, ce n'est pas du tout un souci. Il suffit de séparer tous les événements concernés

par des virgules après le mot clé Handles. Si, par exemple la remise à zéro doit être effectuée en cas de clic sur le bouton **Résultat**, mais aussi de clic sur le bouton **Annulation**, on aura :

```
Private Sub RemiseAZero() Handles Résultat.Click, Annulation.Click
                                                                    ...
  instructions
                                                                    ...
End Sub
```

En réalité, il y a juste une petite subtilité supplémentaire. C'est que lorsqu'un événement déclenche une procédure, Visual Basic prévoit automatiquement un passage de paramètres depuis l'événement jusqu'à la procédure, passage de paramètres sans lequel on serait parfois bien embêté.

La syntaxe complète d'une procédure événementielle sera donc typiquement :

```
Private Sub RemiseAZero(ici des paramètres) Handles Résultat.Click
                                                                    ...
  instructions
                                                                    ...
End Sub
```

**[24]**

## III. Le composant Winsock en VB (transmission)

### 1. Winsock en VB

Le contrôle Winsock VB de Microsoft permet d'envoyer des datagrames facilement à un autre ordinateur ou tout autre équipement Ip. Ce contrôle ne supporte que deux mode de connexion qui sont Tcp et Udp.

On peut utiliser ce contrôle sous deux formes. La première comme composant et la seconde comme référence. Quand vous voulez utiliser Winsock dans votre application, vous devez l'ajouter comme composant "Microsoft Winsock Control 6.0" ou comme référence "%WinDir%\System\mswinsck.ocx".

**2 - Les propriétés, les procédures et les évènements de Winsock**

Voici la liste des propriétés ayant rapport à la configuration de Winsock. Leurs effets et les valeurs qu'ils peuvent prendre et évidemment, elle peuvent aussi en retourner.

| Nom | Effet | Valeurs acceptées |
|---|---|---|
| Name | *(Quand il est sur une feuille) Le nom du contrôle | Une chaîne de caractères correcte pour un nom d'objet |
| Index | *(Quand il est sur une feuille) Son index si il fait partit d'une collection de contrôles | rien=pas dans une collection;un chiffre de 0 à (limite du type entier) |
| Left | *(Quand il est sur une feuille) La position du côté gauche du contrôle | un chiffre |
| LocalPort | Port à écouter pour recevoir des paquets ou des demandes de connexion | un chiffre de 0 à 65 000 |
| Protocol | Protocole à utiliser | 0 pour TCP et 1 pour UDP |
| RemoteHost | L'adresse de l'hôte auquel il faut envoyer des paquets (UDP) | une adresse ip: 192.168.0.1, 125.124.368.14, ... ou un DNS. |
| RemotePort | Le port de l'hôte auquel il faut envoyer des paquets (UDP) | un chiffre de 0 à 65 000 |
| Tag | *(Quand il est sur une feuille) Une propriété pour stocker des données, ce que l'on veut | ce que l'on veut |
| Top | *(Quand il est sur une feuille) La position du haut du contrôle | un chiffre |

Voici la liste des propriétés de Winsock retournant une valeur en lecture seule.

| Nom | Résultat |
|---|---|
| BytesReceived | Nombre de BYTES reçus |
| LocalHostName | Nom d'hôte de cet ordinateur |
| RemoteHostIP | Adresse IP de l'hôte auquel on est connecté |
| SocketHandle | Le handle du socket qui est utilisé (la section que l'API winsock a dû lui réserver) |
| State | L'état du socket (connecté, erreur, attente de connexion, ...) |

Voici la liste des évènements que Winsock peut déclencher. Normalement, tout va bien dans une feuille, mais pour la classe, vous devez utiliser Withevents puis lui créer un instance et il n'y aura pas de problème.

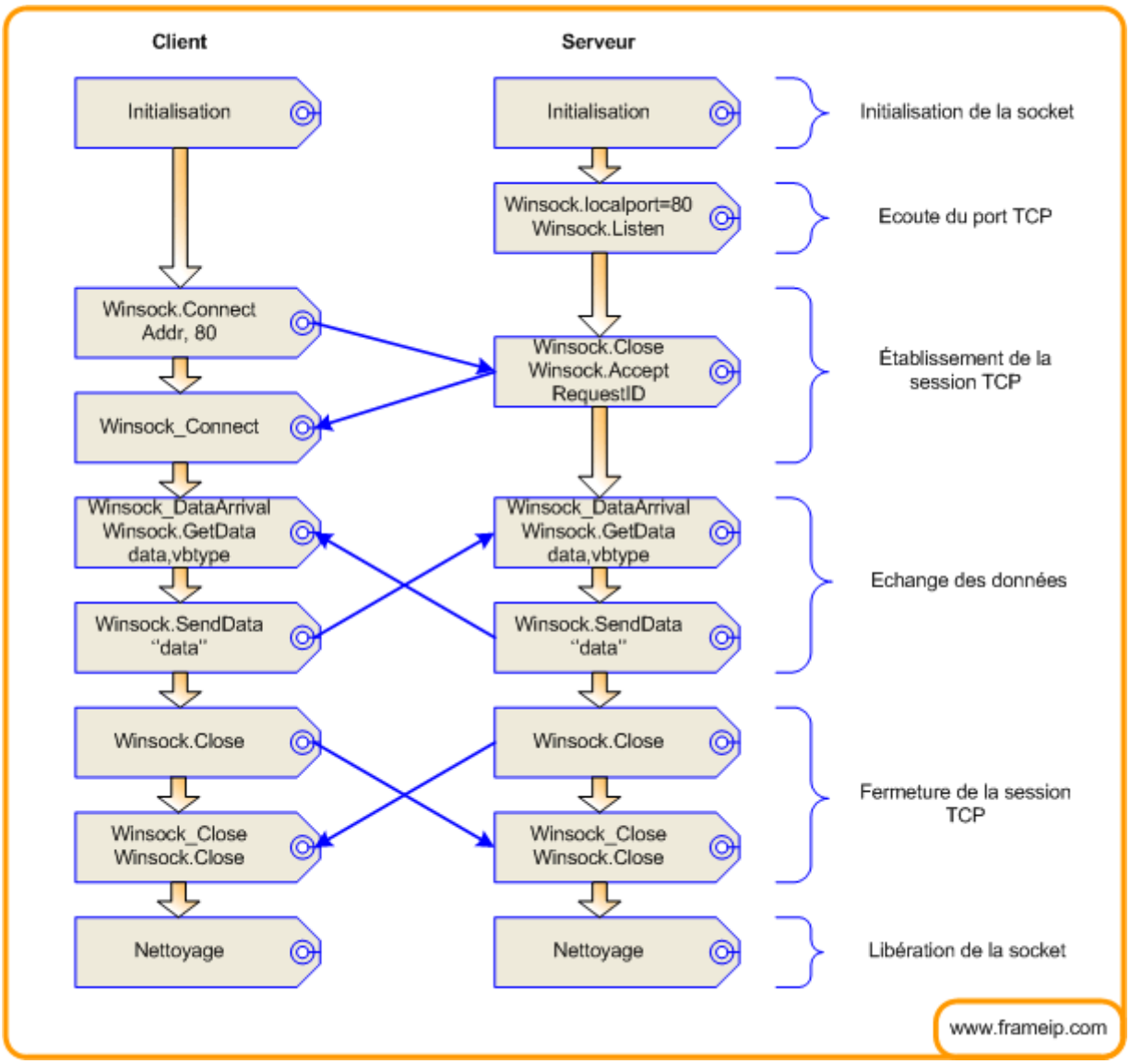
| Nom | Description des arguments | Description |
|---|---|---|
| Close() | | Fermeture d'une connexion (TCP) |

| | | |
|---|---|---|
| Connect() | | Connexion réussie (TCP) |
| ConnectionRequest(ByVal requestID As Long) | l'ID d'une requête de connexion) | Demande de connexion (TCP) |
| DataArrival(ByVal bytesTotal As Long) | Le nombre de bytes qui arrivent | Arrivée de données |
| Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean) | Informations relatives à l'erreur (numéro, description, ...) | Erreur!!!!!! |
| SendComplete() | | Envoi des données terminé |
| SendProgress(ByVal bytesSent As Long, ByVal bytesRemaining As Long) | Nombre de bytes envoyées<br>Nombre de bytes qui reste à envoyer | Envoi des données en cours ... |

Avec ces différentes méthodes et propriétés, vous serez capables de créer des applications utilisant Winsock.

# 3 - Mode connecté

## 3.1 - Schéma d'une relation client serveur




## 3.2 - Le mode client

Voici le code expliqué d'un client en Tcp. Il est bien sur requis une feuille avec un bouton command1 et un contrôle socket.

```
Private Sub Command1_Click() socket.Connect "www.google.com", 80 'On se connecte à www.google.com sur le port 80 End Sub

Private Sub socket_Close() 'Fin de la connexion MsgBox "connexion terminée!" socket.Close 'Fermeture du socket End Sub

Private Sub socket_Connect() 'Connexion établie MsgBox "connexion réessie!" socket.SendData "GET / HTTP/1.0" & vbCrLf & "Accept: */*" & vbCrLf & "User-Agent: Winsock" & vbCrLf & vbCrLf 'On envoie
```

```
des données au serveur, ici c'est une requête HTTP End Sub

Private Sub socket_DataArrival(ByVal bytesTotal As Long) 'Arrivée de données Dim data As String If socket.State <> sckConnected Then Exit Sub 'Juste au cas où ça a buggé socket.GetData data, vbString 'On récupère les données 'DONNÉES DANS DATA MsgBox data End Sub

Private Sub socket_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean) Err.Clear 'On ignore l'erreur End Sub
```

Voici comment créer un client http sans utiliser le contrôle Winsock, mais plutôt l'objet référence.

```
Public WithEvents socket As Winsock 'Pour les évènements

Private Sub Command1_Click() socket.Connect "www.google.com", 80 'On se connecte à www.google.com sur le port 80 End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer) Set socket = Nothing 'On vide l'objet winsock End Sub

Private Sub socket_Close() 'Fin de la connexion MsgBox "connexion terminée!" socket.Close 'Fermeture du socket End Sub

Private Sub socket_Connect() 'Connexion établie MsgBox "connexion réessie!" socket.SendData "GET / HTTP/1.0" & vbCrLf & "Accept: */*" & vbCrLf & "User-Agent: Winsock" & vbCrLf & vbCrLf 'On envoie des données au serveur, ici c'est une requête HTTP End Sub

Private Sub socket_DataArrival(ByVal bytesTotal As Long) 'Arrivée de données Dim data As String If socket.State <> sckConnected Then Exit Sub 'Juste au cas où ça a buggé socket.GetData data, vbString 'On récupère les données 'DONNÉES DANS DATA MsgBox data End Sub

Private Sub socket_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean) Err.Clear 'On ignore l'erreur End Sub

Private Sub Form_Load() Set socket = New Winsock 'On initialise winsock 'With socket '.param1=??? '.param2=??? 'End With End Sub
```

## 3.3 - Le mode serveur

### 3.3.1 - Sur une feuille

Voici le code expliqué d'un serveur avec un seul client Tcp simultané. Il est bien sur requis une feuille avec un bouton command1 et un contrôle Winsock socket.

```
Private Sub Command1_Click() socket.Close 'UN SOCKET DOIT ÊTRE FERMER AVANT D'ÊTRE MIS EN ÉCOUTE!!! socket.LocalPort = 95 'écoute port 95 socket.Listen End Sub

Private Sub socket_Close() 'Fermeture du socket MsgBox "connexion terminée!" socket.Close 'S'assurer que le socket se ferme car sinon, ça peut être très drôle ;-) Command1_Click End Sub
```

```
Private Sub socket_ConnectionRequest(ByVal requestID As Long) 'Demande de connexion MsgBox "Demande de connexion!" socket.Close 'UN SOCKET DOIT ÊTRE FERMER AVANT D'ACCEPTER UNE CONNEXION!!! 'Bon, ya u gros problème. c'est une connexion par socket. QUE FAIRE!!! socket.Accept requestID 'Une autre contrôle winsock peut très bien accepter cette demande... End Sub

Private Sub socket_DataArrival(ByVal bytesTotal As Long) 'Arrivée des données Dim data As String If socket.State <> sckConnected Then Exit Sub 'En cas de problèmes ... on ne sait pas socket.GetData data, vbString MsgBox data 'Ce que le client dit (on s'en fiche tu :-) 'Ce qu'on lui répond : Dim envoie As String envoie = "<html><head><title>Salut</title></head><body>Allo</body></html>" 'La réponse à envoyer socket.SendData "HTTP/1.0 200 OK" & vbCrLf & "Server: Winsock/6.0" & vbCrLf & "Content-Type: text/html" & vbCrLf & "Content-Lenght: " & Len(envoie) & vbCrLf & vbCrLf & envoie 'Une réponse HTTP ^ End Sub

Private Sub socket_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean) Err.Clear 'En cas d'erreur, ignorer End Sub

Private Sub socket_SendComplete() socket.Close 'On ferme End Sub
```

Voici la partie serveur multi-clients. il est bien sur requis une feuille avec un bouton command1 et deux contrôles Winsock socket et écoute.

```
Private Sub Command1_Click() 'Le socket est ouvert et il restera ouvert If ecoute.State = 2 Then Exit Sub 'Si on écoute déjà, on quitte la procédure ecoute.LocalPort = 95 ecoute.Listen End Sub

Private Sub ecoute_ConnectionRequest(ByVal requestID As Long) 'LE TRÈS GROS MORCEAU Dim nb As Integer nb = socket.Count 'Le nombre de socket dans la collection. Comme le 0 compte, pas besoins de rajouter +1. Load socket(nb) 'On ajoute ajoute un autre socket dans la collection de ce contrôle socket(nb).Accept requestID 'CE contrôle fesant partit de la collection accepte la requête Debug.Print "Client ajouté : " & socket(nb).RemoteHostIP End Sub

Private Sub ecoute_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean) Err.Clear 'En cas d'erreur, ignorer End Sub

Private Sub socket_Close(Index As Integer) socket_SendComplete Index 'Si le socket se ferme, fait comme si l'envoie serait fini End Sub

Private Sub socket_DataArrival(Index As Integer, ByVal bytesTotal As Long) 'NOTEZ BIEN L'ARGUMENT INDEX. IL RETOURE L'INDEX DE CE SOCKET DANS LA COLLECTION 'Ce que le client dit (on s'en fiche tu :-) 'Ce qu'on lui répond : Dim envoie As String envoie = "<html><head><title>Salut</title></head><body>a" & String$(10000, "l") & "o</body></html>" 'La réponse à envoyer Debug.Print "Envoie du socket # " & Index & " à " & Now socket(Index).SendData "HTTP/1.0 200 OK" & vbCrLf & "Server: Winsock/6.0" & vbCrLf & "Content-Type: text/html" & vbCrLf & "Content-Lenght: " & Len(envoie) & vbCrLf & "Pragma: no-cache" & vbCrLf & vbCrLf & envoie 'Une réponse HTTP ^ End Sub

Private Sub socket_Error(Index As Integer, ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean) Err.Clear 'En cas d'erreur, ignorer End Sub

Private Sub socket_SendComplete(Index As Integer) socket(Index).Close 'Fermer le socket Debug.Print "Fermeture du socket # " & Index & " à " & Now If Index <> 0 Then Unload socket(Index) 'Si le socket n'est pas le 0 (celui de base), on le FLUSH End Sub
```

### 3.3.2 - Dans une classe

Pour un serveur avec un seul client, c'est la même chose que sur une feuille sauf qu'il faut utiliser Winsock comme référence et le déclarer avec Withevents. Pour un serveur multi-clients, c'est plus complexe.

Voici la partie cliente Classe 1.

```
Public WithEvents socket As Winsock
Public MyNum As Long

Private Sub Class_Initialize() Set socket = New Winsock End Sub

Private Sub Class_Terminate() Set socket = Nothing End Sub

Private Sub socket_Close() Form1.Killsocket MyNum 'S'assurer que le socket se ferme car sinon, ça peut être très drôle ;-) End Sub

Private Sub socket_DataArrival(ByVal bytesTotal As Long) 'Ce qu'on lui répond : Debug.Print "OK" Dim envoie As String envoie = "<html><head><title>Salut</title></head><body>A" & String$(1000, "l") & "o</body></html>" 'La réponse à envoyer socket.SendData "HTTP/1.0 200 OK" & vbCrLf & "Server: Winsock/6.0" & vbCrLf & "Content-Type: text/html" & vbCrLf & "Content-Lenght: " & Len(envoie) & vbCrLf & vbCrLf & envoie 'Une réponse HTTP ^ End Sub

Private Sub socket_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean) Err.Clear 'En cas d'erreur, ignorer End Sub

Private Sub socket_SendComplete() socket.Close 'On ferme End Sub
```

Voici la partie clients Classe 2.

```
Private WithEvents socket As Winsock
Public Sockets As New Collection
Public Num As Long

Private Sub Class_Initialize() Set socket = New Winsock With socket .LocalPort = 95 .Listen End With End Sub

Private Sub Class_Terminate() Set Sockets = Nothing Set socket = Nothing End Sub

Private Sub socket_ConnectionRequest(ByVal requestID As Long) Dim buf As New Client buf.socket.Accept requestID buf.MyNum = Num Num = Num + 1 Sockets.Add buf End Sub

Private Sub socket_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean) Err.Clear End Sub
```

Voici la définition de la feuille que l'on peux, au passage, se passer.

```
Private Clients As Clients
```

```
'JUSTE POUR DÉMARRER ET NETTOYER CAR J'AI PAS ENVIE DE ME CASSER LA TÊTE

Private Sub Form_Load() Set Clients = New Clients End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer) Set Clients = Nothing End Sub

Sub Killsocket(MyNum As Long) For i = 1 To Clients.Sockets.Count If Clients.Sockets.MyNum = MyNum Then Clients.Sockets.Remove i Next End Sub
```

### 3.4 - Exemple complet d'une communication client serveur

C'est très simple, le serveur commence à écouter un port pour attendre une connexion Tcp. Dès qu'un client envoie une demande de connexion, le serveur l'accepte et établit l'ouverture de session avec le client. Le serveur et le client sont maintenant connectés et peuvent donc envoyer et recevoir des données. Quand le serveur arrête la connexion, le client s'en rend compte grâce à l'évènement socket_close. La fermeture de la session TCP peux être demandé par le client ou par le serveur.

Les sources et l'exe peuvent être téléchargé ici.

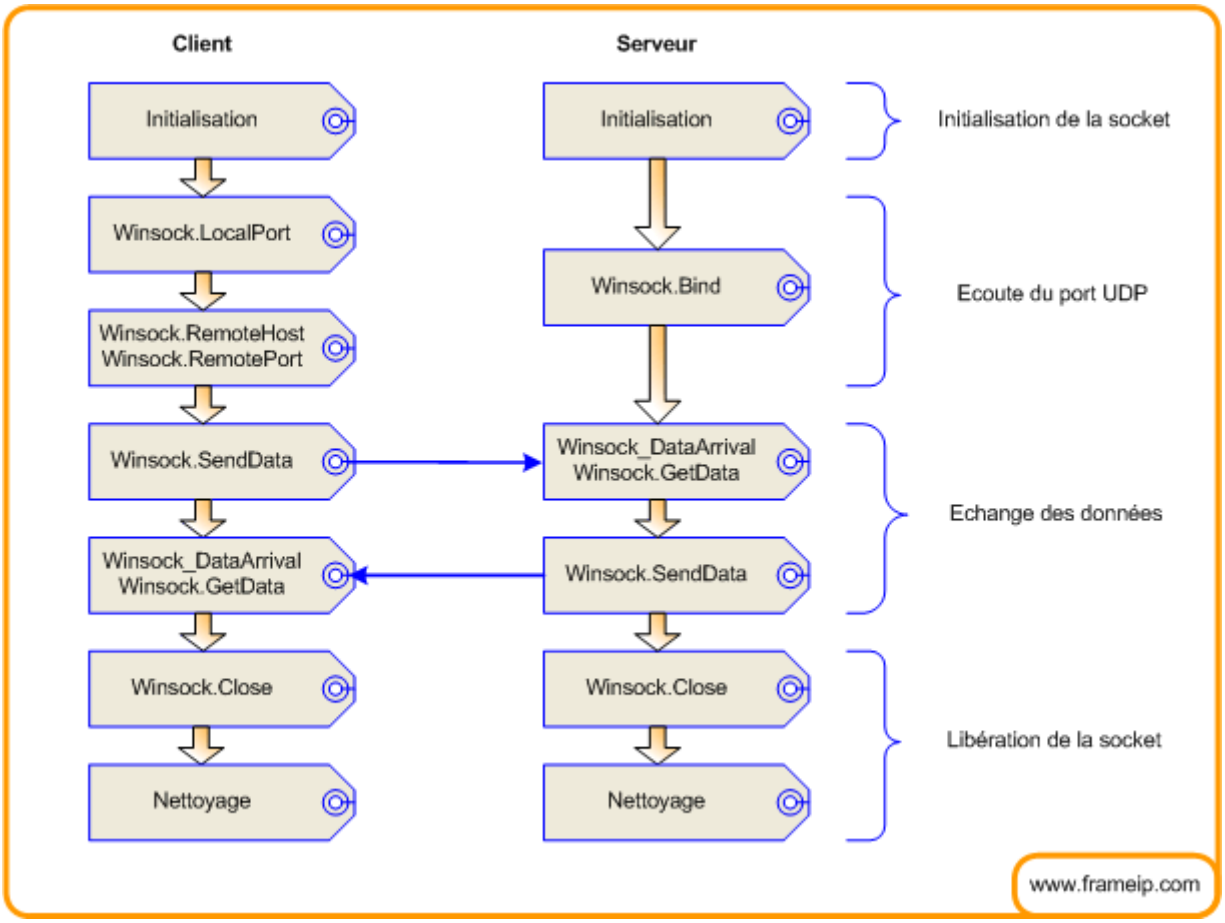
```
Private Sub CheckState_Timer()
If Clients.ListCount = 0 Then send.Enabled = False Else: send.Enabled = True 'Si il n'y a pas de client, pourquoi envoyer un message?
If Ecoute.State = 0 Then Go.Caption = "Écouter" Else: Go.Caption = "Arrêter le serveur" 'Placer le texte du bouton Go
End Sub

Private Sub Clients_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = vbRightButton And Clients.ListIndex <> -1 Then 'Si on clique avec le bouton droit et qu'on a sélectionner un ip dans la liste, on affiche le menu
PopupMenu pop
End If
End Sub

Private Sub Decon_Click() 'ON LE BOOT!!
Dim col
col = Split(Clients.List(Clients.ListIndex), ":") 'On récuprère son # de socket (après le ":")
Socket_Close (CInt(col(1))) 'On fait comme s'il aurait quitté
End Sub

Private Sub Ecoute_ConnectionRequest(ByVal requestID As Long) 'Demande de connexion
Dim Nb As Integer
Nb = Socket.Count '# de la prochaince socket disponible (LIMITE : 65535 CAR TYPE INTEGER = SUR 4 OCTETS (FFFF))
Load Socket(Nb) 'On charge ce socket
Socket(Nb).Accept requestID 'On fait accepter la connexion
recep.Text = recep.Text & recep.Text & "<system> : Connexion établie avec " & Socket(Nb).RemoteHostIP & " !" & vbCrLf & vbCrLf 'On dit que le client c'est connecté
Clients.AddItem Socket(Nb).RemoteHostIP & ":" & Nb 'On met son IP suivit de ":" et son 1 de socket dans la liste
End Sub
```

```
Private Sub Ecoute_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal
Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean)
'Erreur!
MsgBox "Erreur avec le socket! & vbcrlf # " & Number & vbCrLf & Description
Ecoute.Close
End Sub

Private Sub Form_Load()
CheckState_Timer 'On fait faire une première vérification avec le timer
End Sub

Private Sub Go_Click() 'On appuie sur le bouton Go
On Error Resume Next
If Ecoute.State = 2 Then 'Si on écoutait, ben on ferme
Ecoute.Close
For i = 1 To Socket.Count - 1 'On décharge tout les sockets
Socket(i).Close
Unload Socket(i) 'On détruit l'objet créer (0 ne peut pas être unloadé)
Next
Else 'Sinon ben on écoute
Ecoute.LocalPort = Port.Text 'Le port
Ecoute.Listen 'On écoute
If Err.Number <> 0 Then 'ERREUR, POURQUOI ON PEUT PAS ÉCOUTER (sûrement que le port est
occupé)
MsgBox "Erreur avec le socket! & vbcrlf # " & Err.Number & vbCrLf & Err.Description 'On dit pourquoi
Err.Clear 'On nettoie l'erreur
End If
End If
End Sub

Private Sub send_Click() 'Envoie du message, comme avec le client
recep.Text = recep.Text & "<you> : " & vbCrLf & Emission.Text & vbCrLf & vbCrLf
For i = 1 To Socket.Count - 1
Socket(i).SendData Emission.Text
Next
Emission.Text = ""
End Sub

Private Sub Socket_Close(Index As Integer) 'Le client vient de quitter ou il s'est fait booter
Dim col
For i = 0 To Clients.ListCount - 1 'On cherche son # de socket pour l'enlever de la liste
col = Split(Clients.List(i), ":")
If col(1) = Index Then
Socket(col(1)).Close
Unload Socket(col(1)) 'On kill le socket
recep.Text = recep.Text & recep.Text & "<system> : Connexion terminée avec " & col(0) & " !" & vbCrLf &
vbCrLf 'On dit qu'il quitte
Clients.RemoveItem i 'On l'enlève de la liste
Exit For 'Pas besoins de continuer
End If
Next
End Sub

Private Sub Socket_DataArrival(Index As Integer, ByVal bytesTotal As Long) 'Arrivée des données, comme
avec le client
Dim data As String
```

```
Socket(Index).GetData data, vbString
recep.Text = recep.Text & "<" & Socket(Index).RemoteHostIP & "> : " & vbCrLf & data & vbCrLf & vbCrLf
End Sub
```

## 4 - Mode non connecté

### 4.1 - Schéma d'une relation client serveur




### 4.2 - Le mode client et serveur

Udp est orienté sans connexion, donc il nécessite peu de code. C'est le même fonctionnement que Tcp, mais il n'y a pas de session, donc pas de fermeture de connexion. Il est bien sur requis deux Sockets en tant que contrôle ou objet socket1 et socket2 avec le protocole Udp de sélectionné!

```
'SOCKET 1 EST LE "CLIENT" ET LE 2 EST LE "SERVEUR"

Private Sub Form_Load() On Error Resume Next 'UDP PLANTE DESFOIS!!! socket2.Bind 98, "0.0.0.0"
'PORT À ÉCOUTER socket1.RemoteHost = "127.0.0.1" 'Adresse auquel il faut envoyer le(s) datagramme(s)
socket1.RemotePort = 98 'Port auquel il faut envoyer le(s) datagramme(s) socket1.SendData "drole" 'Envoyer
les données End Sub
```

```
Private Sub Form_Unload(Cancel As Integer) socket2.Close 'Arrêter d'écouter le port UDP 98 End Sub

Private Sub socket1_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean) Err.Clear 'Ignorer l'erreur End Sub

Private Sub socket2_DataArrival(ByVal bytesTotal As Long) Dim data As String socket2.GetData data, vbString 'Prendre les données comme avec tcp MsgBox "Message de " & socket2.RemoteHostIP & " : " & data End Sub

Private Sub socket2_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean) Err.Clear 'Ignorer l'erreur End Sub
```

## 4.3 - Exemple complet d'une communication client serveur

Le principe d'Udp est simple. Premièrement, un "serveur" décide de commencer à écouter un port et attend l'arrivée de paquets de données. Un client lui envoie un datagramme en spécifiant à l'adresse ip et ports de destination ainsi que la correspondance de retour. Le client utilise LocalPort=le_#_du_port pour spécifier le port sur lequel il attend une réponse, pus communément appelé port source. Quand le serveur reçoit un paquet, il détermine les coordonnées de l'émetteur et peut ainsi lui répondre. Il n'y a pas de session donc pas de fermeture de connexion, cela permet d'effectuer des échanges courts et rapides.

Les sources et l'exe peuvent être téléchargé ici.

```
Option Explicit

Private Sub CheckState_Timer()
If Socket.State = 1 Then Go.Caption = "Arrêter" Else: Go.Caption = "Écouter"
End Sub

Private Sub Go_Click()
On Error Resume Next
If Socket.State <> 1 Then
Socket.Close
Socket.Bind port.Text, addr.Text
If Err.Number <> 0 Then 'ERREUR, POURQUOI ON PEUT PAS ÉCOUTER (sûrement que le port est occupé)
MsgBox "Erreur avec le socket! & vbcrlf # " & Err.Number & vbCrLf & Err.Description 'On dit pourquoi
Err.Clear 'On nettoie l'erreur
End If
Else
Socket.Close
End If
End Sub


Private Sub send_Click()
On Error Resume Next
If Socket.State <> 1 Then 'On prend 1 port pour recevoir les messages du recépteur
Socket.LocalPort = 0 'N'IMPORTE QUEL PORT LIBRE
End If
Socket.RemoteHost = sendaddr.Text 'L'adresse où ilo faut envoyer les données
Socket.RemotePort = sendport.Text 'Le port où ilo faut envoyer les données
```

```
Socket.SendData Emission.Text 'On envoie (ÇA PEUT BOGUER POUR RIEN!!!)
recep.Text = recep.Text & "<you> : " & vbCrLf & Emission.Text & vbCrLf & vbCrLf 'Pour savoir ce qu'on a envoyer
Emission.Text = "" 'On fait de la place pour un nouveau message
End Sub

Private Sub Socket_DataArrival(ByVal bytesTotal As Long)
Dim data As String
Socket.GetData data, vbString 'On les met dans la variable data
recep.Text = recep.Text & "<" & Socket.RemoteHostIP & "> : " & vbCrLf & data & vbCrLf & vbCrLf 'Pour dire qui à envoyer le message
sendaddr.Text = Socket.RemoteHostIP 'On garde son adresse et on peut lui répondre
sendport.Text = Socket.RemotePort 'On garde son port et on peut lui répondre
Beep
End Sub

Private Sub Socket_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean)
'ERREUR!!! ON AFFICHE L'ERRREUR ET ON FERME LE SOCKET
MsgBox "Erreur avec le socket! & vbcrlf # " & Number & vbCrLf & Description
Socket.Close
End Sub
```

# IV. Traitement d'images (Code source)

Voici notre code source sous visual basic du programme de traitement d'image ce dernier utilise différents filtres pour traiter l'image

```
Option Explicit

Dim Filters() As String

Private Sub Btn_Filter_Click()
Dim Filter As FilterType
With Filter
.Add = Int(Add)
.Div = Int(Div)
If Div = 0 Then
MsgBox "Div ne peut etre nul"
Exit Sub
End If
Dim n As Byte
For n = 1 To 9
.M(n) = Int(Mat(n))
Next n
End With
Screen.MousePointer = vbHourglass
ApplyFilter PicTarget.hdc, 0, 0, PicSrc.ScaleWidth, PicSrc.ScaleHeight, PicSrc.hdc, 0, 0, Filter
Screen.MousePointer = vbDefault
End Sub

Private Sub Add_GotFocus()
With Add
.SelStart = 0
.SelLength = Len(.Text)
End With
End Sub

Private Sub Btn_LoadPicture_Click()
On Error Resume Next
```

```
CommonDialog1.ShowOpen
Set PicSrc.Picture = LoadPicture(CommonDialog1.FileName)
PicTarget.Cls
End Sub

Private Sub Div_GotFocus()
With Div
.SelStart = 0
.SelLength = Len(.Text)
End With
End Sub

Private Sub Form_Load()
With List_Filters
.AddItem "None"
.AddItem "Blur"
.AddItem "Gaussian Blur"
.AddItem "Edge Detection"
.AddItem "Emboss"
.AddItem "Sharpen"
.AddItem "Saturation"
.AddItem "Eclaircir"
ReDim Filters(List_Filters.ListCount - 1)
Filters(0) = "0;0;0;0;1;0;0;0;0;1;0"
Filters(1) = "1;1;1;1;1;1;1;1;1;9;0"
Filters(2) = "0;1;0;1;4;1;0;1;0;8;0"
Filters(3) = "-1;-1;-1;-1;8;-1;-1;-1;-1;4;128"
Filters(4) = "1;0;0;0;0;0;0;0;-1;2;32"
Filters(5) = "0;-1;0;-1;9;-1;0;-1;0;5;0"
Filters(6) = "0;0;0;0;25;0;0;0;0;1;-255"
Filters(7) = "0;0;0;0;25;0;0;0;0;9;0"
End With
End Sub

Private Sub List_Filters_Click()
Dim FilterStr() As String
FilterStr = Split(Filters(List_Filters.ListIndex), ";")
Mat(1) = FilterStr(0): Mat(2) = FilterStr(1): Mat(3) = FilterStr(2)
Mat(4) = FilterStr(3): Mat(5) = FilterStr(4): Mat(6) = FilterStr(5)
Mat(7) = FilterStr(6): Mat(8) = FilterStr(7): Mat(9) = FilterStr(8)
Div = FilterStr(9)
Add = FilterStr(10)

Btn_Filter_Click
End Sub

Private Sub Mat_GotFocus(Index As Integer)
With Mat(Index)
.SelStart = 0
.SelLength = Len(.Text)
End With
End Sub

Private Sub PicSrc_Click()
Set PicSrc.Picture = LoadPicture("C:\TELOPHT\VB_WebCam_1855842212005\myPic\p1.jpg")
PicTarget.Cls
End Sub

Private Sub PicTarget_Click()

End Sub
```

# V. Interface de capture (Code source)

Voici notre code source sous visual Basic du programme de capture de l'image ce denier fait aussi la conversion bmp to Jpeg :

```
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal wMsg As
Long, ByVal wParam As Long, lParam As Any) As Long
Private Declare Sub ReleaseCapture Lib "user32" ()
Const WM_NCLBUTTONDOWN = &HA1
Const HTCAPTION = 2

Private m_TimeToCapture_milliseconds As Integer
Private m_Width As Long
Private m_Height As Long
Private mCapHwnd As Long
Private bStopped As Boolean

Private Sub File1_Click()

End Sub

Private Sub Frame1_DragDrop(Source As Control, X As Single, Y As Single)

End Sub

Private Sub lvButtons_H8_click()
Form2.Show
End Sub



Private Sub lvButtons_H7_click()
FrmMain.Show
End Sub

Private Sub Form_Load()
On Error Resume Next
Label1.BackStyle = 0 ' CheckBox
Label4.BackStyle = 0 ' Form Caption
Label5.BackStyle = 0 ' Form Close

m_TimeToCapture_milliseconds = 100
m_Width = 352
m_Height = 288
bStopped = True
mCapHwnd = 0
End Sub

Public Sub Start()
On Error Resume Next
If mCapHwnd <> 0 Then Exit Sub
FrameNum = 0
Timer1.Interval = m_TimeToCapture_milliseconds

' for safety, call stop, just in case we are already running
Me.Timer1.Enabled = False

' setup a capture window
mCapHwnd = capCreateCaptureWindowA("WebCap", 0, 0, 0, m_Width, m_Height, Me.hwnd, 0)
DoEvents
' connect to the capture device
Call SendMessage(mCapHwnd, WM_CAP_CONNECT, 0, 0)
DoEvents
Call SendMessage(mCapHwnd, WM_CAP_SET_PREVIEW, 0, 0)

' set the timer information
```

```
bStopped = False
Me.Timer1.Enabled = True

End Sub
Public Sub StopWork()
On Error Resume Next
' stop the timer
bStopped = True
Timer1.Enabled = False

' disconnect from the video source
DoEvents

Call SendMessage(mCapHwnd, WM_CAP_DISCONNECT, 0, 0)
mCapHwnd = 0

End Sub


Private Sub Label1_Click()
On Error Resume Next
Image2.Visible = Not Image2.Visible
If Image2.Visible = True Then
Image1.Width = 352
Image1.Height = 288
Image1.Stretch = True
Else
Image1.Stretch = False
End If
End Sub


Private Sub Label3_Click()

End Sub

Private Sub Label4_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
On Error Resume Next
Dim lngReturnValue As Long
If Button = 1 Then
'Release capture
Call ReleaseCapture
'Send a 'left mouse button down on caption'-message to our form
lngReturnValue = SendMessage(Me.hwnd, WM_NCLBUTTONDOWN, HTCAPTION, 0&)
End If
End Sub

Private Sub Label5_Click()
' From Close
Call lvButtons_H6_Click
End Sub

Private Sub lvButtons_H1_Click()
On Error Resume Next
If mCapHwnd = 0 Then Exit Sub

Call SendMessage(mCapHwnd, WM_CAP_DLG_VIDEOSOURCE, 0, 0)
DoEvents
End Sub

Private Sub lvButtons_H2_Click()
On Error Resume Next
If mCapHwnd = 0 Then Exit Sub

Call SendMessage(mCapHwnd, WM_CAP_DLG_VIDEOFORMAT, 0, 0)
DoEvents

End Sub
```

```
Private Sub lvButtons_H3_Click()
Start
lvButtons_H1.Enabled = True
lvButtons_H2.Enabled = True
lvButtons_H4.Enabled = True
lvButtons_H3.Enabled = False
End Sub

Private Sub lvButtons_H4_Click()
StopWork
lvButtons_H1.Enabled = False
lvButtons_H2.Enabled = False
lvButtons_H4.Enabled = False
lvButtons_H3.Enabled = True
End Sub

Private Sub lvButtons_H5_Click()
On Error Resume Next
DoEvents
If Dir(App.Path & "\myPic", vbDirectory) = "" Then MkDir (App.Path & "\myPic")
File1.Path = App.Path & "\myPic"
'File1.Pattern = "*.bmp"
File1.Pattern = "*.jpg"
File1.Refresh

Dim Maxnum As Integer, ii As Integer
For ii = 0 To File1.ListCount - 1
If Left(File1.List(ii), 1) = "p" Then
If CInt(Mid(File1.List(ii), 2, Len(File1.List(ii)) - 4)) > Maxnum Then
Maxnum = CInt(Mid(File1.List(ii), 2, Len(File1.List(ii)) - 4))
End If
End If
Next

'SavePicture Image1.Picture, App.Path & "\myPic\p" & Maxnum + 1 & ".bmp"
Picture1.Picture = Image1.Picture
SAVEJPEG App.Path & "\myPic\p" & Maxnum + 1 & ".jpg", 100, Me.Picture1
DoEvents
End Sub

Private Sub lvButtons_H6_Click()
Timer1.Enabled = False
If mCapHwnd <> 0 Then StopWork
Unload Me
End

End Sub



Private Sub Timer1_Timer()
On Error Resume Next

' pause the timer
Timer1.Enabled = False

' get the next frame;
Call SendMessage(mCapHwnd, WM_CAP_GET_FRAME, 0, 0)

' copy the frame to the clipboard
Call SendMessage(mCapHwnd, WM_CAP_COPY, 0, 0)

' For some reason, the API is not resizing the video
' feed to the width and height provided when the video
' feed was started, so we must resize the image here
' Image1.Stretch = True
' get from the clipboard
```

```
Image1.Picture = Clipboard.GetData
' restart the timer
DoEvents
If Not bStopped Then
Timer1.Enabled = True
End If

End Sub
```

## IV. présentation de l'application

Notre application est une fusion de 3 programmes sous visual basic :

Le premier programme sert à capturer l'image d'une webcam et l'enregistrer sous forme d'une image Jpeg

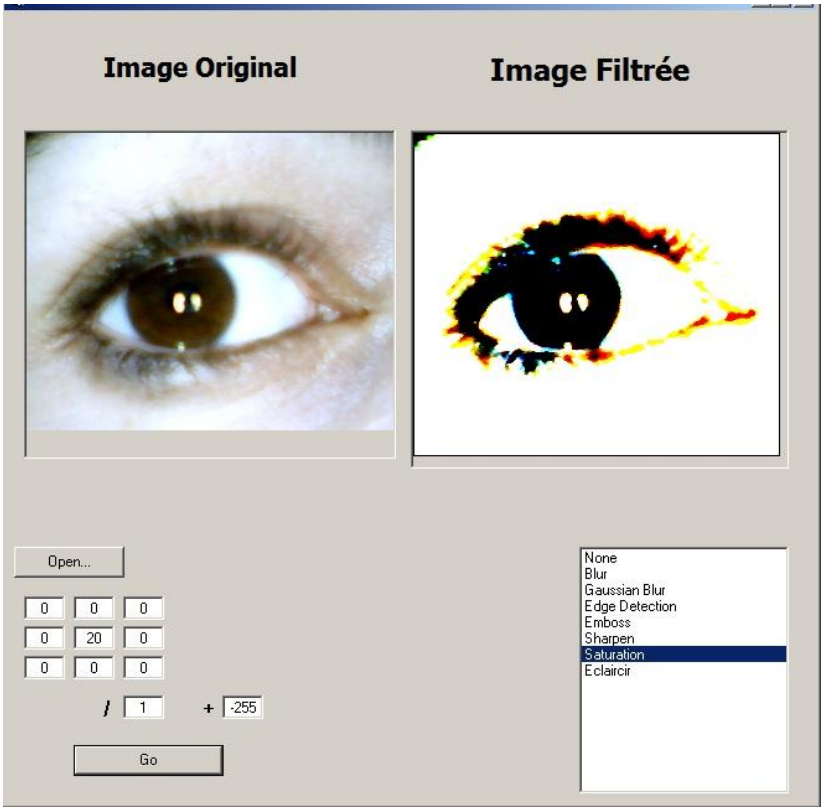
Le deuxième programme sert à traiter cette image cette dernière est chargée d'une manière automatique sur l'interface et ces algorithmes de traitement d'image pour mieux distinguer l'état et l'utilité de l'image capturée.

Le Troisième est un programme nous servira à transférer cette du pc local vers un pc distant via le protocole TCP /IP afin que le spécialiste puisse voir l'image, et si il y a pathologie la déceler.

L'application est conçue sous forme d'un fichier exécutable donc elle est utilisable depuis n'importe quel ordinateur équipé des fichiers OCX requis, l'interface est très simpliste et intuitive a la portée de n'importe quel individu qui sait manier au minimum l'outil informatique les illustrations ci-dessous sont jointes à cette présentation afin de mieux expliquer notre application.

Capture 1 : Interface Globale


Capture 2 : interface traitement




Capture 3 : interface transmission

Etude et élaboration d’une plate-forme télé-ophtalmologique

# Conclusion et perspectives

A partir de cette plate-forme qui est en version préliminaire (béta), nous pouvons envisager une infinité de perspectives tant bien sur l'objectif premier que sur l'ergonomie et la facilité d'utilisation,

C'est-à-dire que nous pouvons envisager pour cette application une adaptation et un portage direct sur les appareils mobiles (Smartphones et tablettes) et que nous pouvons envisager d'utiliser cette application dans le domaine de la dermatologie par exemple, et que nous envisageons aussi d'y intégrer un moyen de communication direct et vocal entre le patient et le spécialiste mais cette adaptation nécessite une longue période de test.

Cette plate-forme pourrait aussi être dédié à un seul médecin qui fournira son propre service de téléconsultation, il suffira juste de fixer l'adresse d'envoie dans la partie transmission,

Cette plate-forme sera donc proposée avec une webcam plus perfectionné et une interface » mieux pensée ce qui garantira une excellente qualité de service.

Ce dispositif pourra aussi être embarqué dans une machine cette dernière sera relié via Smartphone à un médecin de garde et là on parlera d'une dispositif de téléconsultation qui pourra être embarqué en pharmacie ou dans n'importe quel lieu adéquat.

En conclusion ce projet est prêt à suivre l'avancée technologique et à utiliser cette technologie au service de la médecine le schéma capture, traitement et transmission a une infinité d'applications possible.

## Bibliographie

1. Atlas de poche en couleurs Ophtalmologie, Gerhard K. Lang, Edition Maloine 2002

2. Anatomy and Physiology for Nurses, Roger Watson, Twelfth EDITION ELSEVIER 2005

3. Hanyane - Bien voir et mieux vivre au village, Erika Sutter, Allen Foster et Victoria

Francis, ICEH http://www.guide-vue.fr/la-sante-de-vos-yeux/pathologies-adultes

4. http://www.solutionsmedicales.fr/gerer-un-cabinet/la-telemedecine--definition-et-mise-en-oeuvre

5. http://www.chu-sainte-justine.org/pro/page.aspx?id_page=10004135

6. http://www.beyondlogic.org/spp/parallel.htm

7. http://computer.howstuffworks.com/parallel-port.htm

8. http://ctips.com/parallel.htm

9. http://fr.wikipedia.org/wiki/FireWire

10. http://www.macworld.com/article/1158145/thunderbolt_what_you_need_to_know.html

11. http://fr.wikipedia.org/wiki/Traitement_d'images

12. J. Canny, *A Computational Approach to Edge Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 8, No. 6, Nov 1986

13. H. Maître, *La détection de contour dans les images*, http://www.tsi.enst.fr/~bloch/TDI/poly_contours.pdf

14. S. Mavromatis, O. Coulon, *Analyse d'image*, http://pages-perso.esil.univ-mrs.fr/~ocoulon/cours/ESIL3A_contours.pdf

15. F. Devernay, *Détection de contour*, http://devernay.free.fr/cours/vision/pdf/c3.pdf

16. Image Processing *Fundamentals*, http://www.ph.tn.tudelft.nl/Courses/FIP/noframes/fip.html

Les définitions des adresses IP versions 4 et 6, la notion de classe et la notation CIDR sont documentées dans les *Request for comments* suivants (en anglais) :

17. RFC 997 — Internet numbers, mars 1987

18. RFC 791 — Internet Protocol, septembre 1981 (IP).

19. RFC 1519 — Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy, septembre 1993

20. RFC 1918 — Address Allocation for Private Internets, février 1996

21. RFC 1531 — Dynamic Host Configuration Protocol, octobre 1993 (DHCP).

22. VISUAL BASIC 2008 | Cours de M. Jean-Luc Roy
http://www.esct.on.ca/profs/jlroy/visual_basic2008/index.html

23. Microsoft Msdn Library http://*msdn.microsoft.com/fr-fr/library*

24. Cours initiation a Visual Basic, Conçu et enseigné par Christophe Darmangeat (Université Paris 7)

25- Le 25 juillet 2004, par DeadlyPredator, http://www.frameip.com/