UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

ECOLE DOCTORALE SCIENCES POUR L'INGENIEUR

Doctorat

Automatique et Informatique Industrielle

par

Julien DE JONCKHEERE

METHODOLOGIE ORIENTEE OBJET POUR LE TRAITEMENT TEMPS REEL DU SIGNAL

APPLICATION AU MONITORAGE MEDICAL

Soutenue publiquement le 17 décembre 2007 devant la commission d'examen :

Rapporteurs	**N. NOURY, Maître de Conférence HDR, Université de Grenoble I**
	L. BARVAIS, Professeur, Université Libre de Bruxelles, Belgique
Examinateur	**F. CABESTAING, Professeur, LAGIS, Université de Lille I**
	A. EL HAJJAJI, Professeur, Université de Picardie Jules Verne
Directeur	**C. VASSEUR, Professeur, LAGIS, Université de Lille I**
Co-directeur	**R. LOGIER, Ingénieur Hospitalier en Chef, ITM, CHR&U de Lille**

Cette thèse a été préparée au Laboratoire d'Automatique Génie Informatique et Signal (CNRS UMR 8146)

Remerciements

Ce mémoire est le résultat de travaux menés en collaboration entre le Laboratoire d'Automatique Génie Informatique et Signal (LAGIS CNRS UMR 8146) et l'Institut de Technologie Médicale (ITM EA1049) du Centre Hospitalier & Universitaire de Lille (CHR&U.

Je remercie Messieurs Norbert Noury, Docteur HDR à l'Université de Grenoble I, Luc Barvais, Professeur à l'Université Libre de Bruxelles, de m'avoir fait l'honneur d'accepter la lourde tâche de rapporteur.

Je témoigne ma reconnaissance à Monsieur Cabestaing du Laboratoire d'Automatique Génie Informatique et Signal et à Monsieur El Hajjaji de l'université de Picardie Jules Verne de me faire l'honneur de participer à ce jury.

Je tiens à témoigner ici ma reconnaissance à Monsieur Christian Vasseur, Professeur au Laboratoire d'Automatique Génie Informatique et Signal, et Monsieur Régis Logier, Docteur Ingénieur au CHR&U de Lille, Co-directeurs de thèse, pour l'encadrement dynamique et chaleureux de ce travail.

Les applications médicales décrites dans ce mémoire ont fait l'objet de collaborations étroites avec différents services cliniques du CH&U de Lille. J'exprime ma plus profonde reconnaissance à l'ensemble des personnes des différents services impliqués qui ont manifesté de l'intérêt pour ces travaux et plus particulièrement Messieurs les Docteurs Mathieu Jeanne, Michel Delecroix, Yvon Rioux, Bruno Marciniak, Richard Matis, Emmanuel Ardiet qui sont les initiateurs de nombreux projets et qui ont consacré beaucoup de temps à leur mise en œuvre. Qu'il me soit permis de leur exprimer toute mon amitié.

Ma reconnaissance va également à tous ceux qui par leur amitié ou leur gentillesse ont contribué à la réussite de ces travaux et notamment à Messieurs Guillaume Delmar, Alain Dassonneville et Pascal Chaud.

Table des matières

Introduction.....**9**

1ère partie : Définitions générales

Chapitre 1
Acquisition et traitement du signal

1 – Introduction.....14
2 – Acquisition du signal :.....16
2.1 Descriptif :.....16
2.1.1 Le capteur classique :.....16
2.1.2 Le capteur Numérique :.....17
2.2 Exemple : le capteur ECG 3 dérivations :.....19
3 - Traitement du signal :.....21
3.1 Descriptif :.....21
3.2 Exemple : Calcul de la fréquence cardiaque :.....22
4 - Transmission du signal :.....25
4.1 – Descriptif :.....25
4.2 – Exemple : transmission des informations issues de l'ECG :.....25
5 – Cas particulier des systèmes temps réel :.....26
5.1 – Modélisation des systèmes temps réel :.....26
5.2 – Notion d'ordonnancement :.....27
5.3 – Notion de contrôle :.....29
6 – Conclusion :.....29

Chapitre 2
La programmation Orientée Objet

1 – L'approche objet 32
1.1 – Pourquoi l'approche objet ?.....32
1.1.1 – Encapsulation 34
1.1.2 – Héritage 35
1.1.3 – Polymorphisme 37
1.2 –Les objets 37
1.2.1 – Caractéristiques 37
1.2.2 – Classes d'objet 40
1.3 – Relations entre les objets 40
1.4 – Framework : la bibliothèque de classes 45
1.5 – Les composants 45
2 – Application au domaine du traitement du signal : les architectures flot de données 46
2.1 – Communication entre les modules 48

2.1.1 – Messages 49
2.1.2 – Transmission des messages 50
2.2 – Représentation 52
2.3 – Composition statique 54
3 - Conclusion 55

Chapitre 3
Définition d'une architecture logicielle pour l'acquisition et le traitement du signal

1 – Introduction 58
2 – Définition des différentes classes 58
3 – Communication entre les classes 60
4 – Communication avec la source de données 64
4.1 – Principe général 64
4.2 – Extraction de l'information 65
4.3 – Distribution des données 66
5 – Spécificité des systèmes temps réel 67
6 – Implémentation : création des classes 69
6.1 - Les classes TSource et TVoie 69
6.2 – La classe TTraitements 72
6.3 - La classe TSortie 72
7 – Contrôle 73
7.1 - Les états locaux 73
7.2 - Règles de transition 75
7.3 – exemple 76
8 - Conclusion 77

2ème partie : Application au monitorage médical

Chapitre 4
Architecture matérielle du réseau d'acquisition

1 - Introduction :....80
2 - Présentation générale : 81
3 – La tête de réseau : 82
4 - Les modules d'acquisition :....84
4.1 - Traitement numérique :....84
4.2 - Conditionnement :....84
4.3 - Les modules développés à ce jour :....85
5 - Communication entre les éléments du réseau :....87
5.1 - Flot des données d'acquisition : 87
5.2 - Principe de transfert des données des modules vers le PC :....88
5.3 - Emission de données du PC vers les modules :....89
6 – Conclusion :....90

Chapitre 5
Développement d'une bibliothèque de composants logiciels pour le monitorage médical

1 – Introduction :....92
2 - Les éléments de communication avec le réseau :....92
2.1 – Création :....93
2.2 - Initialisation :....93
2.3 – Commandes :....96
2.4 – Lecture / Extraction / Distribution des données :....97
2.5 – Destruction :....97
3 – Messages et ports de communication :....97
3.1 – Les types de messages :....97
3.1.1 – TAcqData :....97
3.1.2 – TacqDataBuf :....98
3.1.3 – TAcqDataMat :....98
3.1.4 – TacqDataBool :....99
3.2 – Les ports de communication :....99
4 - Les éléments de traitement de signal :....100
4.3 – Les composants de base :....101
4.4 – Les composants de détection :....102
4.5 – Les composants de mesures cycliques :....102
4.6 – Les composants de mesures sur fenêtre glissante :....103
4.7 – Les composants d'analyse avancé :....104
4.8 – Les composants spécialisés :....104
5 – Interface de programmation graphique :....105
5.1 – programmation du traitement :....105
5.2 – Visualisation des résultats :....106
6 - Conclusion :....107

Chapitre 6
Exemple d'application : le monitorage du niveau d'analgésie

1 – Introduction : 110
2 – Le Système Nerveux Autonome 110
2.1 - La neurotransmission : 110
2.2 - Effets du SNA sur le Système Cardiovasculaire (SCV) : 111
2.3 – Régulation du système cardio-vasculaire (SCV) par le SNA : 112
2.3.2 – Principe : 112
2.3.2 Modèle cybernétique : 115
2.4 – Caractéristiques spectrales du rythme cardiaque : 116
3 – Prise en charge du niveau d'analgésie: 117
3.1 – Principe : 117
3.2 - Acquisition de la série RR : 119
3.3 – Traitement de la série RR : 120
3.3.1 – Filtrage : 120
3.3.2 – Prétraitement : 121
3.3.3 – Détection du pattern respiratoire : 121
3.4 – Calcul des paramètres : 122
3.5 – Mesure en continu : 124
4 - Réalisation de l'algorithme : 124
4.1 – Décomposition en modules élémentaires : 124
4.1.1 – Acquisition de l'ECG : 125
4.1.2 – Construction de la série RR : 125
4.1.3 – Filtrage de la série RR : 125
4.1.4 – Ré-échantillonnage : 126
4.1.5 – Normalisation : 126
4.1.6 – Filtrage par ondelette : 127
4.1.7 – Calcul des paramètres : 127
5 – Résultat : 128
6 – Conclusion : 129

Conclusion et perspectives 131

Annexes

Annexe 1 : Fonctionnement pratique du réseau : protocole 135
Annexe 2 : Les différentes classes de composants 146
Annexe 3 : Filtres numériques 157
Annexe 4 : Allures des différents types de fenêtres de pondérations 166
Annexe 5 : Transformée de Fourier Rapide 167
Annexe 6 : Transformé en Ondelette 171
Annexe 7 – RR series filtering 179

Références bibliographiques 185

Introduction

Le monitorage trouve des applications dans de nombreux secteurs d'activité du domaine médical (surveillance et aide au diagnostic [SENH 02], recherche expérimentale, essais cliniques...). Pour chacun de ces domaines, le besoin constant de nouvelles méthodes de monitorage [BLOC 86] se traduit par l'émergence de nouvelles techniques d'acquisition et de traitement numérique des signaux physiologiques. Ainsi, il est de plus en plus difficile pour les intervenants du monde médical de mettre en place d'une manière simple des systèmes de monitorage adaptés à leurs besoins sans faire appel à des techniques de programmation et de traitement du signal avancées. Le développement d'une plateforme simple, évolutive et interactive pour la conception de logiciels de traitement des signaux physiologiques constitue donc un bon moyen de répondre à ces besoins.

Les différentes techniques, plus ou moins complexes, de traitement numérique du signal font toutes appel à des méthodes connues et peuvent donc être considérées comme une implémentation modulaire de ces différentes méthodes [LOGI 95]. Ces différentes tâches sont donc interdépendantes et doivent être synchronisées pour un bon fonctionnement de l'outil de traitement final. Cette notion prend toute son importance lorsqu'il s'agit, comme c'est souvent le cas dans les problèmes de traitement numérique du signal et notamment dans le domaine du monitorage médical, de traiter les informations continûment.

Par ailleurs, la conception d'applications de monitorage médical implique certaines contraintes [CANA 01] tant au niveau de la qualité de l'information issue du traitement [SRLF 00] que de la méthodologie de développement. La conception d'algorithmes de traitements numériques par assemblage de briques logicielles (tâches) élémentaires répondant à ces contraintes est donc un moyen efficace d'obtenir une information de qualité en sortie de la chaîne de traitement [HEIN 01].

La création d'une bibliothèque de composants logiciels de traitements numériques du signal interconnectables indifféremment fournit au concepteur un bon outil pour la réalisation de systèmes complexes. Ainsi, cette bibliothèque est initialement constituée de composants de base correspondant aux différentes tâches élémentaires pouvant intervenir dans un système de

traitement complet (filtrage, opérations arithmétiques, seuillage...). La réutilisation et l'assemblage de ces composants élémentaires doit permettre de créer d'une manière simple des composants de traitement du signal plus complexes venant à leur tour enrichir la bibliothèque dans le but d'une réutilisation.

La première partie constitue une définition générale des différents concepts et outils utilisés pour la réalisation de ce travail.

Ainsi, le premier chapitre introduit le concept d'instrumentation de mesures en décrivant les différents éléments pouvant intervenir dans une application classique d'acquisition et de traitement du signal.

Le second chapitre constitue quant à lui une introduction à la programmation orientée objet et présente ses concepts et ses avantages par rapport au méthodes « conventionnelles ». Nous présentons ensuite le paradigme de programmation par flot de données, application du modèle objet au domaine du traitement du signal.

Enfin, pour terminer cette première partie, dans le troisième chapitre, nous présentons une architecture logicielle générique pouvant servir de base au développement de tout type d'application d'acquisition et de traitement du signal.

La seconde partie de ce mémoire présente l'application des concepts et outils précédemment introduits au domaine particulier du monitorage médical.

Ainsi, le quatrième chapitre constitue une présentation générale du réseau d'acquisition ITM. Ce dispositif constituant la source de données de l'application à réaliser, nous donnons ici une description globale de son architecture matérielle et de son protocole de communication.

Dans le cinquième chapitre, nous nous intéressons à la mise en œuvre de la bibliothèque de composants conformément aux spécifications des chapitres 3 et 4. Ainsi, nous décrivons différents types de composants logiciels pour l'acquisition et le traitement des signaux physiologiques.

Enfin, dans le dernier chapitre, les outils développés sont mis en œuvre pour une application particulière du monitorage médical : la mesure du niveau d'analgésie pendant l'anesthésie générale.

Chapitre 1

Acquisition et traitement du signal

1 – Introduction 14
2 – Acquisition du signal 16
2.1 Descriptif 16
2.1.1 Le capteur classique 16
2.1.2 Le capteur numérique 17
2.2 Exemple : le capteur ECG 3 dérivations 19
3 - Traitement du signal 21
3.1 Descriptif 21
3.2 Exemple : calcul de la fréquence cardiaque 22
4 - Transmission du signal 25
4.1 – Descriptif 25
4.2 – Exemple : transmission des informations issues de l'ECG 25
5 – Cas particulier des systèmes temps réel 26
5.1 – Modélisation des systèmes temps réel 26
5.2 – Notion d'ordonnancement 27
5.3 – Notion de contrôle 29
6 – Conclusion 29

1 – Introduction

Les systèmes d'acquisition et de traitement du signal sont utilisés dans le but d'observer un processus, un système ou un phénomène physique et d'en donner une représentation à un observateur. Le fonctionnement d'un processus se traduit par un ensemble de manifestations extérieures telles que bruits, déplacements, tensions électriques, pressions, températures... permettant à l'observateur d'entrer en communication avec le processus. L'objectif d'un système d'acquisition et de traitement du signal est donc de développer des méthodes et des moyens permettant d'améliorer la communication entre le processus et l'observateur. Une telle démarche, qui vise à la conception d'une instrumentation dite « intelligente », ajoute à la notion de perception (capteur) la notion d'aide à la communication et à la compréhension. Dans ce sens, l'instrument n'est plus seulement l'objet qui permet à l'homme d'appréhender son environnement, mais il devient un outil d'aide à la réflexion et à la décision.

On peut décomposer le processus d'acquisition et de traitement du signal en plusieurs phases, les deux phases de base étant l'*acquisition* qui permet la collecte des signaux sur différentes sources de données et la *représentation de l'information* qui permet à un observateur d'obtenir une vue de l'état du processus, système ou phénomène traité.

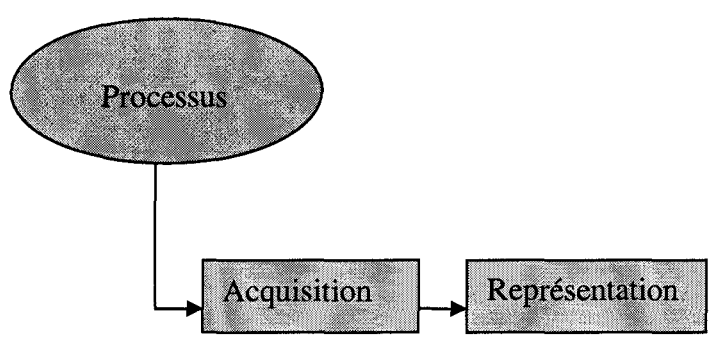


Figure 1.1 : chaîne d'acquisition et de traitement du signal

L'acquisition est le plus souvent réalisée par un élément sensible au phénomène physique (transducteur). La représentation de l'information dépend quant à elle de l'observateur. Ainsi,

dans le cas d'un observateur humain, elle se fait par l'intermédiaire d'une interface homme/machine.

En plus de ces deux fonctions essentielles, on trouve également dans de tels dispositifs des unités de *traitement* de l'information qui permettent, soit de calculer différents paramètres à partir de l'information issue de la source, soit de traiter les données de base pour les rendre compatibles avec le reste des équipements. Les données traitées sont alors transmises par l'intermédiaire d'éléments de *transmission* qui permettent de faire parvenir ces données à d'autres systèmes et/ou périphériques. Ces différents périphériques peuvent être des organes de *représentation* (afficheurs, traceur papier...), de *mémorisation* (disque dur, CDROM...) qui permettent de stocker les données de la source ainsi que les données calculées ou tout simplement les données issues d'autres systèmes (centrale de données, réseau TCP/IP, matériel à commander...).

Ainsi, on peut représenter une chaîne d'acquisition et de traitement du signal selon le schéma de la figure 1.2.

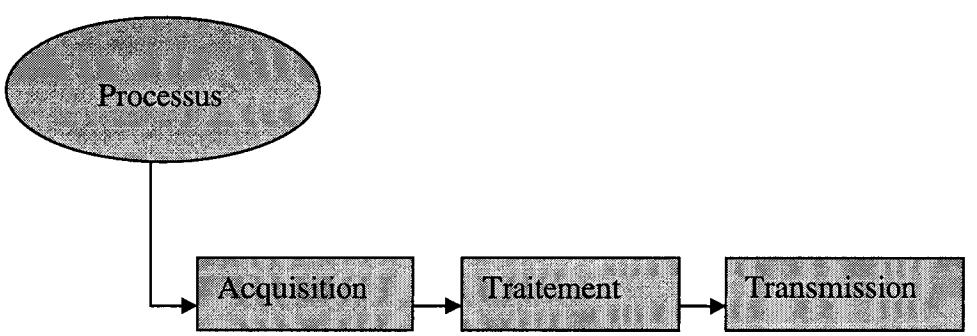


Figure 1.2 : schéma de base d'un système d'acquisition et de traitement du signal

Par exemple, dans le monde médical, la surveillance des patients nécessite la plupart du temps l'acquisition et l'analyse de différents signaux physiologiques comme l'électrocardiogramme (ECG), la pression artérielle (PA), les paramètres respiratoires... Dans un tel système d'acquisition et de traitement du signal, le patient est alors considéré comme un processus dont l'état à l'instant t peut être caractérisé par la mesure de ces différentes grandeurs physiques. Par exemple, un cardiotachymètre est un instrument permettant la

mesure du nombre de battements cardiaques par minute (ou fréquence cardiaque FC) à partir du signal électrocardiographique prélevé à la surface de la peau. L'information transmise au médecin est alors la fréquence cardiaque exprimée en Battements Par Minute (BPM). Le système complet est alors constitué d'un élément d'*acquisition* permettant le relevé du signal ECG, d'un élément de *traitement* permettant le calcul de la fréquence cardiaque et d'un élément de *transmission* permettant au médecin de visualiser l'information fréquence cardiaque.

2 – Acquisition du signal

Appréhender le comportement d'un système ou d'un processus nécessite l'acquisition des grandeurs physiques représentatives de son état. On associe à la phase d'acquisition la notion de capteur. Le capteur est chargé de récupérer l'information à la source, de la coder, c'est-à-dire de la rendre compréhensible pour l'élément de traitement et de la transmettre à ce dernier.

2.1 Descriptif :

2.1.1 Le capteur classique :

On peut représenter un capteur par l'association d'un *transducteur*, d'un *conditionneur*, et d'un *transmetteur*, l'ensemble de ces éléments constituant la partie acquisition de la chaîne de mesure [ASCH 87]. Une telle représentation est donnée sur la figure 1.3.

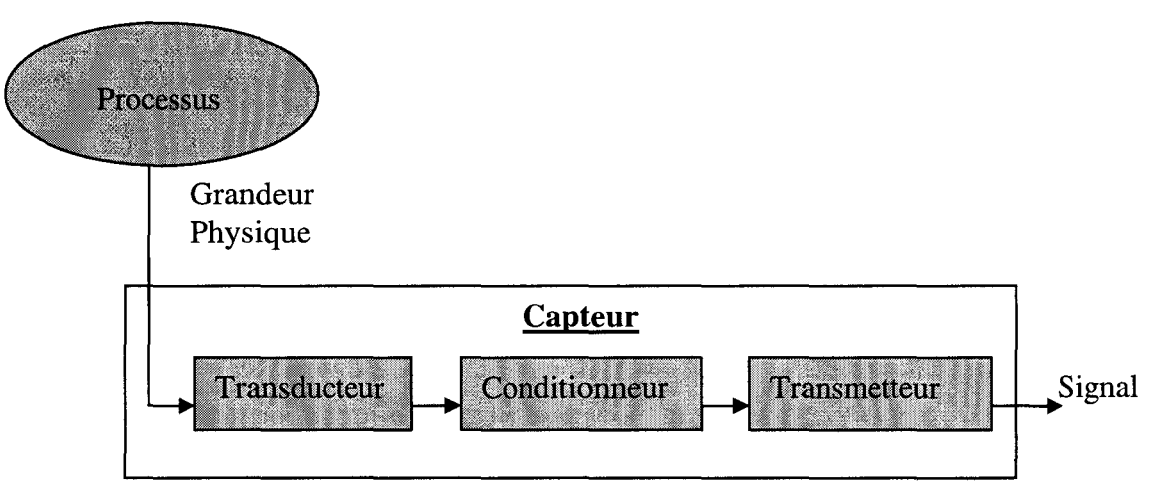


Figure 1.3 : le capteur classique

Le *transducteur* est un élément sensible aux phénomènes physiques. Il traduit l'information captée en une grandeur exploitable par les autres constituants du capteur.
Le *conditionneur* est chargé de la mise en forme du signal renvoyé par le transducteur (filtrage, gain, offset). Il le transforme le plus souvent en grandeur électrique (analogique ou numérique), optique, mécanique...
Le *transmetteur* code l'information issue du conditionneur en l'adaptant au format utilisé par le consommateur et en la rendant compatible avec la ligne de transmission qui dépend du moyen de communication utilisé.

2.1.2 Le capteur numérique :

Le capteur numérique possède au minimum un transducteur, un conditionneur et un organe numérique (figure 1.4).

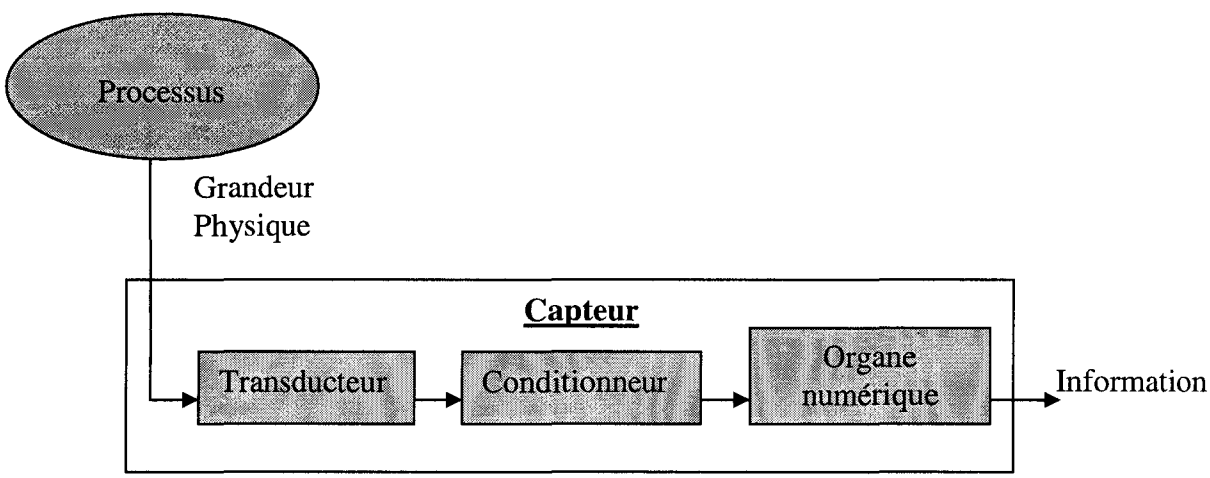


Figure 1.4 – Le capteur numérique

Le *transducteur* reste indispensable pour l'acquisition analogique de la grandeur physique.
Le *conditionneur* reste lui aussi présent. Dans la plupart des cas, sa fonction est de transformer le signal en accord avec les caractéristiques de l'organe numérique (gain, offset, filtre anti-repliement). Cependant, il peut être réduit à sa plus simple expression du fait de la

prise en charge par l'organe numérique de bon nombre de fonctions qui étaient dévolues au capteur classique.

L'*organe numérique* est constitué d'une structure classique de microcalculateur (microprocesseur, mémoires, entrées/sorties...). Cette structure est chargée de la conversion analogique/numérique de l'information issue du conditionneur. La plupart des microcalculateurs étant équipés d'un circuit de communication numérique, la fonction de transmission du capteur est, la plupart du temps, elle aussi, prise en charge par l'organe numérique.

Contrairement aux systèmes analogiques qui traitent l'information de manière continue, les systèmes numériques les traitent de manière séquentielle. En effet, de par leur conception, les systèmes numériques effectuent les différentes tâches qui leur sont dévolues au rythme d'une horloge. Un laps de temps est donc nécessaire entre deux opérations successives. Pour cette raison, le signal source issu du conditionneur se doit d'être numérisé avant traitement.

La numérisation (conversion Analogique/Numérique) des signaux fait appel à deux notions essentielles : l'échantillonnage et la quantification.

- **l'échantillonnage :**

Cette étape est obligatoire pour passer du monde analogique (continu) au monde numérique (discret). En effet, un signal analogique évoluant continûment dans le temps ne peut être connecté à un organe numérique qui, par définition, travaille sur des nombres à un rythme imposé. L'élément numérique ne reçoit donc que des fragments de signal. Cette fragmentation s'appelle l'échantillonnage. L'échantillonnage consiste en fait à convertir le signal continu en une suite de nombres où chaque nombre représente l'amplitude du signal à un instant donné [FUND 01]. Afin de conserver le maximum d'informations sur le signal observé, il convient d'échantillonner à un rythme suffisamment élevé. Le théorème de Shannon énonce qu'un signal continu est correctement échantillonné s'il ne contient pas de composantes fréquentielles au-delà de la moitié de la fréquence d'échantillonnage. Dans la pratique, on choisit une fréquence d'échantillonnage largement supérieure (par-exemple dix fois supérieure) à la fréquence maximale du signal à observer.

- **la quantification :**

Après l'échantillonnage temporel vient la quantification d'amplitude ayant pour but de faire correspondre à la valeur de l'amplitude au moment de l'échantillonnage un nombre x(k), habituellement codé sous forme binaire et appartenant à un ensemble fini de valeurs discrètes.

$$\mathbf{X(t) \rightarrow x(k.Ts) \text{ avec } k.Ts = t \text{ puis } x(k)}$$

Ainsi, on exprimera la précision de la quantification en nombre de bits. Par exemple, un signal continu évoluant entre 0 et 1 Volt quantifié sur 8 bits aura une précision de $1/2^8$ volts. La quantification a donc pour but de transformer un signal x(t) continu en une suite de valeurs discrètes x(k).

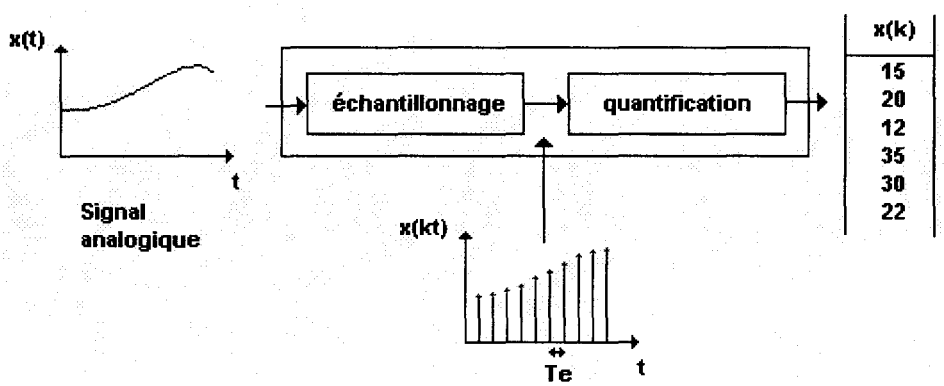


Figure 1.5 – Principe de numérisation du signal

2.2 Exemple : le capteur ECG 3 dérivations :

L'électrocardiogramme (ECG) est la représentation du potentiel électrique généré par l'activité musculaire du cœur (figure 1.6).

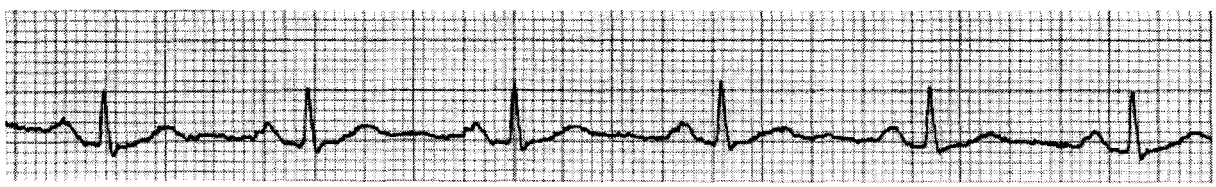


Figure 1.6 : l'ECG

- **Transducteur :**

L'ECG possède 12 dérivations électriques. L'acquisition de l'ensemble de ces dérivations permet d'avoir une idée tridimensionnelle de l'activité électrique du cœur. Cependant, pour certaines utilisations comme par exemple la mesure de la fréquence cardiaque, 3 dérivations suffisent. Dans ce cas, l'ECG est recueilli par le biais de 3 électrodes collées sur la surface de la peau. Ces trois électrodes sont placées sur le bras droit (RA), sur le bras gauche (LA) et sur la jambe droite (RL).

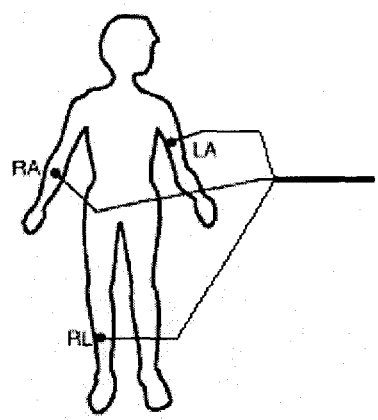


Figure 1.7 : Le capteur ECG

- **Conditionneur :**

La tension électrique véhiculée par l'ECG est extrêmement faible (de l'ordre du millivolt) et donc très susceptible aux énergies rayonnées qui se trouvent dans l'environnement. Cela ce caractérise par un bruit aléatoire qui se superpose au signal et altère l'information initiale. Les sources principales d'interférences proviennent du réseau électrique, des rayonnements électromagnétiques environnants ou des éventuels mouvements du patient (artefacts). Afin de restituer au mieux l'information initiale, le conditionneur se doit donc de filtrer l'information issue du transducteur. Une chaîne possible de conditionnement de l'ECG est décrite figure 1.8. Cette figure décrit le principe de la mesure de l'ECG par amplificateur différentiel.

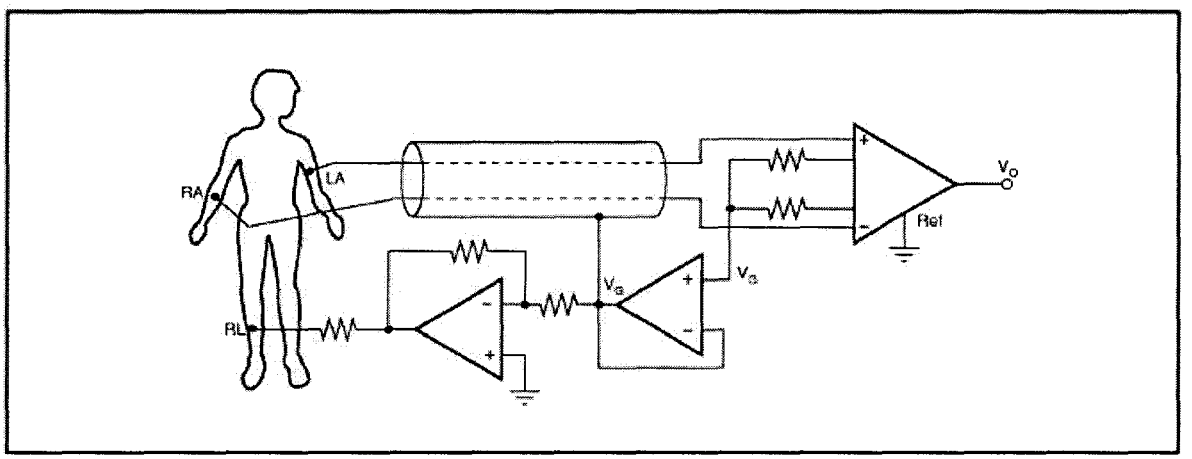


Figure 1.8 : conditionnement de l'ECG

- **Organe numérique :**

L'organe numérique est en charge de l'échantillonnage, de la quantification et de la transmission de l'information. Communément, dans le cadre d'une application de monitoring classique, on échantillonne le signal ECG analogique à 250 Hz avec une précision de 8 bits. Cependant, pour d'autres applications nécessitant une analyse plus fine de l'ECG, on utilisera une fréquence d'échantillonnage et une précision plus élevées.

3 - Traitement du signal

L'élément de traitement est chargé de récupérer l'information issue du capteur, d'effectuer différentes opérations afin d'extraire les données utiles, de coder ces données, c'est-à-dire de les rendre compatibles avec les autres éléments du système et de les transmettre à ces derniers.

3.1 Descriptif :

Ce type d'élément possède donc un organe *récupérateur*, un organe *opérateur* et un organe *transmetteur* (figure 1.9). Les informations sont transmises du capteur à l'élément de traitement via un canal de transmission.

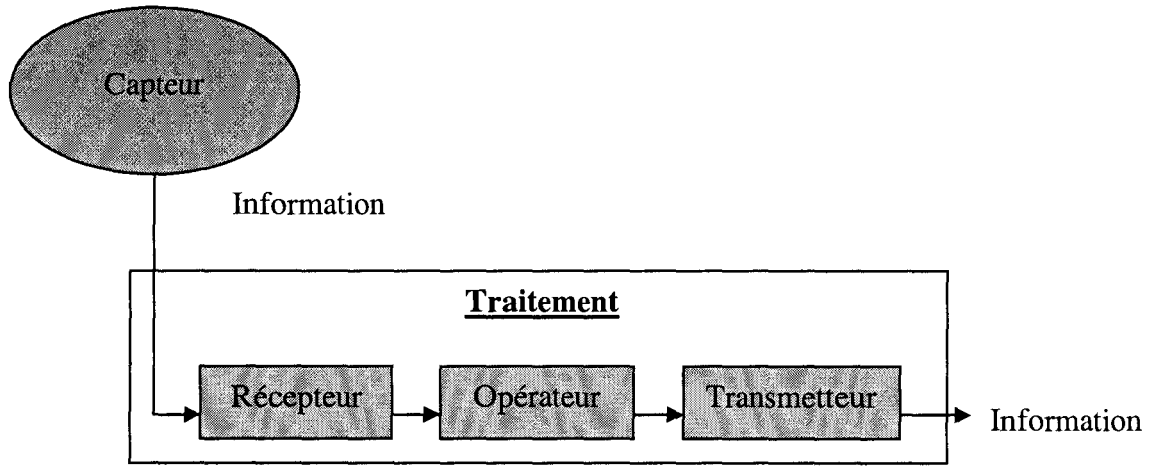


Figure 1.9 – Les éléments de traitement

Le *récepteur* reçoit, via le canal de transmission, les informations issues du capteur et les décode afin de les rendre compréhensibles par les autres constituants de l'élément de traitement. Cet élément effectue, en quelque sorte, les fonctions inverses de l'élément *transmetteur* du capteur.

L'*opérateur* est chargé d'effectuer différentes opérations sur les signaux issus du récepteur pour en extraire des données utiles et exploitables pour l'utilisateur.

Le *transmetteur* code l'information issue de l'opérateur en l'adaptant au format utilisé par le consommateur (sorties) et en la rendant adaptée avec la ligne de transmission qui dépend du medium de communication utilisé.

3.2 Exemple : calcul de la fréquence cardiaque :

Reprenons l'exemple précédent traitant de l'acquisition du signal électrocardiographique. L'analyse morphologique du signal ECG peut être d'une aide précieuse pour le diagnostic clinique d'un grand nombre de pathologies (ischémies, troubles du rythme, malformations cardiaques...). Cependant, un grand nombre d'informations issues de ce signal ECG peuvent également aider le clinicien à élaborer son diagnostic. La fréquence cardiaque, par exemple, le renseigne sur l'état physiologique du patient dans des domaines aussi variés que l'anesthésie, la réponse à l'effort, l'obstétrique...

- **Récepteur :**

La première phase de ce calcul concerne la réception et la mise en forme de l'information issue du capteur ECG. En effet, ce capteur émet une suite de données numériques codées qui ne fournit pas directement d'information quant à l'amplitude du signal et qui ne retranscrit pas complètement l'information temporelle essentielle à notre calcul. Cependant, connaissant les paramètres de quantification et d'échantillonnage du capteur ECG, il est alors aisé de calculer l'amplitude du signal physique et de calculer la date de chaque échantillon.
On obtient donc l'amplitude en effectuant le calcul **Amp[i] = EE[i]*A+B** où **A** et **B** représentent les facteurs de quantification du capteur et **EE[i]** la valeur de l'échantillon reçu. De la même façon, l'information temporelle est obtenue en effectuant le calcul **T[i] = T[i-1] + Te** où **Te** représente la période d'échantillonnage du capteur.

- **Opérateur :**

Une fois le signal remis en forme, il reste à effectuer les différents calculs pour extraire l'information utile du signal source. Le signal ECG peut être considéré comme la succession de différentes ondes, les plus significatives étant les ondes P, Q, R, S et T (figure 1.10).

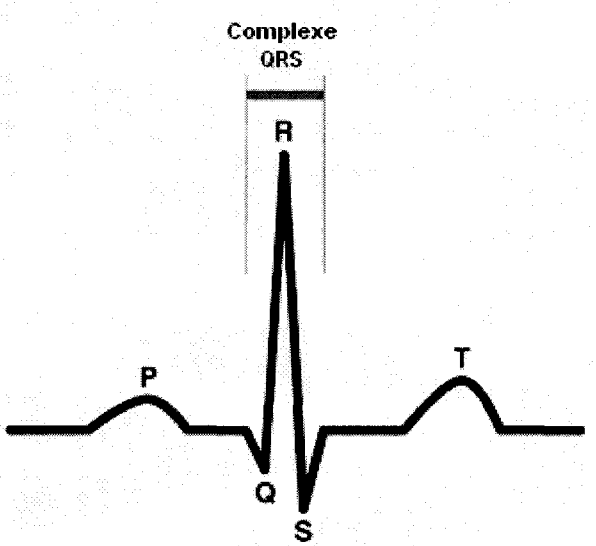


Figure 1.10 : les différentes ondes de l'ECG

Le calcul de la fréquence cardiaque consiste alors, dans un premier temps, à calculer l'intervalle de temps entre deux ondes identiques successives. On parle alors d'intervalles

P-P, Q-Q, R-R, S-S ou T-T. Par souci de facilité de détection, on choisit, la plupart du temps, de calculer l'intervalle R-R pour déterminer la fréquence cardiaque. Pour cela, la méthode consiste à isoler le complexe QRS de l'ECG, à détecter le pic R puis à calculer le temps qui le sépare du dernier pic R détecté.

Une des méthodes de détection de l'onde R consiste à comparer chaque échantillon à la valeur d'un seuil (par exemple 50 % de l'amplitude de l'onde R précédente). Une fois ce seuil atteint, il convient d'observer la dérivée du signal pour détecter l'onde R au moment de son changement de signe. On peut alors déterminer la valeur de l'intervalle RR en millisecondes comme étant le temps écoulé entre les deux derniers pics détectés.

$$\mathbf{RR = T(R[i])–T(R[i-1])}$$

Ensuite, on calcule la fréquence cardiaque en battements par minute (bpm).

$$\mathbf{FC = 60000/RR}$$

On calcule également une nouvelle valeur de seuil.

S = 0.5*Amp[i] où Amp[i] représente l'amplitude de l'échantillon détecté.

- **Transmetteur :**

L'information FC est ensuite transmise à un élément de sortie (afficheur) ou à un autre élément de traitement comme par exemple un dispositif de détection de seuil (haut ou bas) permettant de déceler une éventuelle bradycardie (diminution de la FC) ou une tachycardie (augmentation de la FC).

4 - Transmission du signal

Les éléments de transmission regroupent tous les éléments chargés de la *représentation*, de la *mémorisation* et de la *transmission* de l'information à d'autres systèmes.

4.1 – Descriptif :

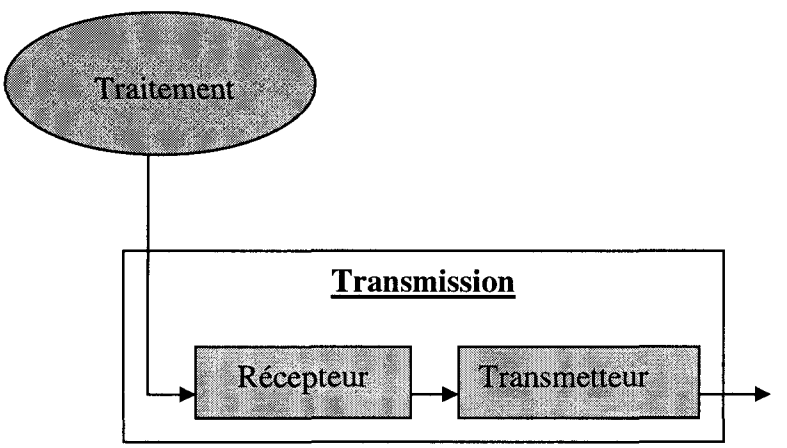


Figure 1.11 – Les éléments de transmission

Le *récepteur* reçoit, via le canal de transmission, les informations issues du capteur et les décode afin de les rendre compréhensibles par les autres constituants de l'élément de traitement. Cet élément effectue, en quelque sorte, les fonctions inverses de l'élément transmetteur du traitement.

Le *transmetteur* code l'information issue du récepteur en l'adaptant au format utilisé par le consommateur et en la rendant compatible avec la ligne de transmission qui dépend du moyen de communication utilisé.

4.2 – Exemple : transmission des informations issues de l'ECG :

Dans le cadre d'un service de réanimation et pour l'exemple particulier de l'ECG, la phase de transmission équivaut, par exemple, à l'affichage du signal ECG et de la fréquence cardiaque sur le moniteur de chevet du patient et la transmission de la fréquence cardiaque moyenne sur un serveur de données via un protocole TCP/IP.

5 – Cas particulier des systèmes temps réel

5.1 – Modélisation des systèmes temps réel :

Le système de traitement que nous envisageons est un système temps réel, c'est-à-dire qu'il doit effectuer un certain nombre de tâches en un temps donné. Cette contrainte implique des exigences rigoureuses en matière de performance et de rapidité [SAKS 99].

Les systèmes informatiques temps réel se différencient des autres systèmes informatiques par la prise en compte de contraintes temporelles, dont le respect est aussi important que l'exactitude du résultat. Autrement dit, le système ne doit pas simplement délivrer des résultats exacts, il doit aussi les délivrer dans des délais imposés. Les systèmes informatiques temps réel sont aujourd'hui présents dans de nombreux secteurs d'activités : dans l'industrie de production par exemple, au travers des systèmes de contrôle de procédé (usines, centrales nucléaires), dans les salles de marché au travers du traitement des données boursières en « temps réel », dans l'aéronautique au travers des systèmes de pilotage embarqués (avions, satellites), ou encore dans le secteur de la nouvelle économie au travers du besoin toujours croissant du traitement et de l'acheminement de l'information (vidéo, données, pilotage à distance, réalité virtuelle, etc.). Le développement d'un système temps réel nécessite donc que *chacun* des éléments du système soit lui-même temps réel, c'est-à-dire permette de prendre en compte des contraintes temporelles.

Un système temps réel est associé à des contraintes de temps et de ressources. Ainsi, afin de modéliser le système temps réel, on définit différentes variables pour chaque tâche Ti :

- Ai : Date d'arrivée de la tâche au processeur (date de création).
- Ri : Date à laquelle la tâche peut commencer son exécution.
- Di : Date à laquelle la tâche peut terminer son exécution (au plus tard).
- Ei : Durée d'exécution de la tâche.
- Pi : Période de la tâche.
- Si : Date à laquelle la tâche accède au processeur.
- Ci : Date à laquelle la tâche termine son exécution.
- TRi : Temps de réponse de la tâche ; TRi = Ci-Ri.
- TLi : Temps de latence de la tâche (période pendant laquelle une tâche peut être retardée sans que son exécution ne dépasse son échéance); TLi = Di-Ri-Ei.

- Ui : Taux d'utilisation du processeur pour la tâche (Ei/Pi).

Pour un bon fonctionnement, il convient donc de respecter certaines conditions :

Ai < Ri < Si < Ci-Ei < Di –Ei.

Ui<1.

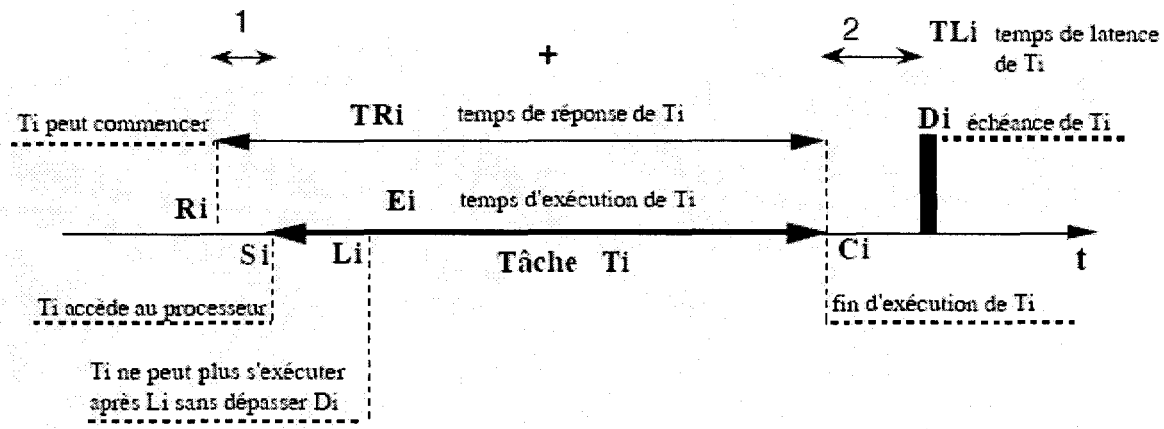


Figure 1.12 : exécution temporelle d'une tâche.

5.2 – Notion d'ordonnancement :

Un système d'ordonnancement des tâches possède trois fonctions principales :

- Calculer les priorités de chacune des tâches et définir l'ordre d'exécution.
- Vérifier la possibilité d'exécution d'une tâche (peut-on accepter une nouvelle tâche).
- Se mettre à jour et s'adapter en cas d'arrivée de nouvelles tâches.

Pour ordonnancer un système temps réel on associe à chacune des tâches les notions de priorité et de périodicité.

- **Priorité :**

  La principale fonction d'un algorithme d'ordonnancement est de définir la priorité de chacune des tâches. On distingue trois niveaux de priorité :

  - **Temps réel critique** (niveau de priorité le plus élevé), dont le non respect équivaut souvent à une catastrophe.
  - **Temps réel essentiel.**
  - **Temps réel non essentiel** (niveau de priorité minimale) qui ne comporte pas trop de contraintes quant à l'exécution ou à la date de déclenchement.

- **Périodicité :**
  - **Les tâches périodiques :** Une tâche Ti est dite périodique de période Pi si elle est exécutée chaque Pi unités de temps. Une telle tâche a ses paramètres Ri et Di connus. Pour ordonnancer une tâche périodique, le système doit être capable de garantir toutes les futures occurrences de la tâche.

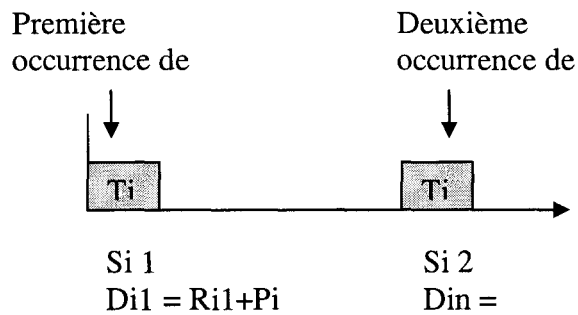


Figure 1.13 : Périodicité.

Dans la plupart des systèmes, on considère qu'une occurrence de tâche périodique a son échéance égale à sa période.

  - **Les tâches apériodiques :** Leur arrivée au processeur est aléatoire ; on ne connaît donc pas leurs paramètres à priori.
  - **Les tâches sporadiques :** Si les occurrences d'une tâche apériodique sont espacées d'au moins q unités de temps, alors cette tâche est considérée comme sporadique de période q. Dans la plupart des modèles d'ordonnancement, ces tâches sont converties en tâches périodiques de période q.

Une application temps réel telle que la nôtre est majoritairement composée de tâches périodiques et on considère souvent ce type de tâches comme étant critique. Autrement dit, toutes ces tâches ont la même priorité.

De la diversité de type de tâches découle une diversité des mécanismes d'ordonnancement de tâches (définition de l'ordre d'exécution des tâches) :

- **Ordonnancement statique :** L'ordre d'exécution des tâches est déterminé avant le début de l'exécution.
- **Ordonnancement dynamique :** La séquence déterminée au préalable est mise à jour et réordonnancée en fonction des nouvelles tâches créées.

La principale fonction du mécanisme d'ordonnancement est donc de vérifier qu'une tâche peut être ordonnancée sans remettre en question les tâches précédemment acceptées et de déterminer un nouvel ordre d'exécution incluant la nouvelle tâche.

5.3 – Notion de contrôle :

Le contrôle représente le jeu d'activités et de mécanismes nécessaires pour définir un état opérationnel optimum pour un système et pour maintenir le système dans cet état en cas de perturbations [GULL 96].
Une structure de contrôle inclut les mécanismes et activités suivants :

- L'installation « on-line » (chargement) de nouvelles structures logicielles et/ou matérielles,
- l'activation et la désactivation du système,
- la détection et la prise en compte des erreurs,
- la maintenance préventive (détection des problèmes avant qu'ils ne deviennent critiques),
- l'évaluation des performances,
- la synchronisation avec d'autres systèmes de contrôle (si le système fait partie d'un système plus large).

6 – Conclusion

Dans ce chapitre, nous avons défini la décomposition des algorithmes de traitement du signal en différentes tâches que sont l'acquisition, le traitement et la transmission. Une telle décomposition pose les bases de la bibliothèque de composants à réaliser ce qui amène à la création de trois grands types de briques logicielles : les sources pour la tâche « acquisition », les traitements pour la tâche « traitement » et les sorties pour la tâche « transmission ».

Dans un second temps, nous avons donné les bases de la définition des systèmes temps réel en termes d'ordonnancement et de contrôle. De même que la mémoire d'un système de traitement du signal peut se trouver saturée dans le cas d'un trop grand nombre de données (saturation spatiale), un système peut saturer en terme de temps processeur dans le cas d'un

trop grand nombre de tâches à réaliser (saturation temporelle). Il est alors important de donner au système la structure d'ordonnancement et de contrôle nécessaire pour palier ce problème.

Chapitre 2

La programmation orientée objet

1 – L'approche objet 32
1.1 – Pourquoi l'approche objet ? 32
1.1.1 – Encapsulation 34
1.1.2 – Héritage 35
1.1.3 – Polymorphisme 37
1.2 –Les objets 37
1.2.1 – Caractéristiques 37
1.2.2 – Classes d'objet 40
1.3 – Relations entre les objets 40
1.4 – Framework : la bibliothèque de classes 45
1.5 – Les composants 45
2 – Application au domaine du traitement du signal : les architectures flot de données 46
2.1 – Communication entre les modules 48
2.1.1 – Messages 49
2.1.2 – Transmission des messages 50
2.2 – Représentation 52
2.3 – Composition statique 54
3 - Conclusion 55

1 – L'approche objet

La Programmation Orientée Objet (P.O.O.) est particulièrement bien adaptée à la description et au développement des systèmes de traitement du signal. Dans ce sens, ce chapitre fournit une introduction à cette méthodologie et présente ses concepts et avantages par rapport aux méthodologies « conventionnelles ».

1.1 – Pourquoi l'approche objet ?

D'une manière générale, toute méthode de conception logicielle doit prendre en compte l'organisation, la mise en relation et l'articulation de structures pour faire émerger le système à réaliser [MULL 97]. Dans ce type de développement, il faut diviser le système en sous systèmes et réunir les sous-systèmes pour construire le système complet. Dans l'approche fonctionnelle, les différentes fonctions du système sont identifiées puis divisées en sous-fonctions et ce itérativement, jusqu'à l'obtention de fonctions suffisamment simples pour être codées dans les langages de programmation conventionnels sous forme de fonctions et de procédures. Ces méthodes s'inspirent directement de la structure des ordinateurs. La séparation des données et du code telle qu'elle existe physiquement dans les ordinateurs a été transposée à ces méthodes. C'est ainsi que les informaticiens ont appris à raisonner en termes de fonctions du système. Le schéma de la figure 2.1 illustre le principe de la décomposition fonctionnelle.

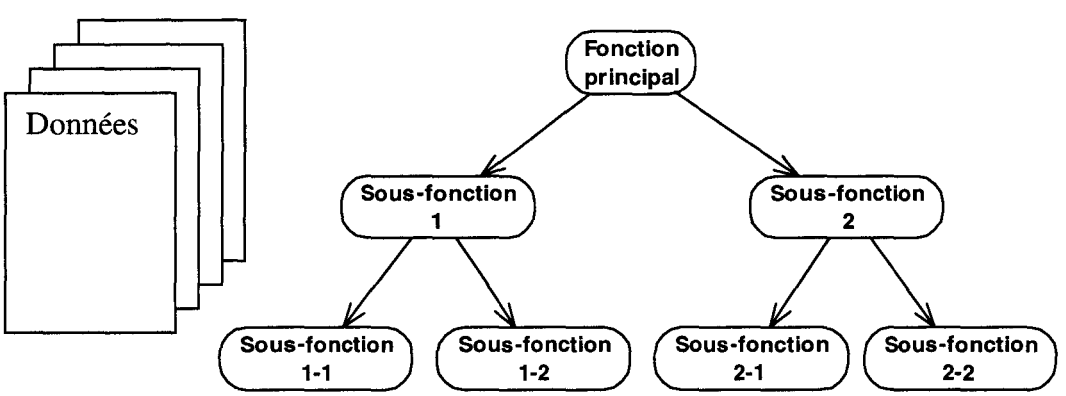


Figure 2.1 : Décomposition fonctionnelle

Tout logiciel est donc composé d'une hiérarchie de fonctions qui, ensemble, fournissent les services attendus. Cette méthode peut apporter des résultats satisfaisants si les fonctions et sous-fonctions sont suffisamment bien identifiées. Cependant, les méthodes fonctionnelles impliquent que la structure même du logiciel dépende du découpage fonctionnel et une modification fonctionnelle peut donc engendrer de lourdes modifications structurelles. En effet, ce type de découpage consiste souvent à factoriser les comportements communs du logiciel c'est-à-dire que pour réaliser une fonction du logiciel, on utilise d'autres fonctions. La factorisation pose des problèmes de maintenance, car la modification, ou la mise à jour d'une fonction va influer en cascade sur une multitude d'autres fonctions.

L'approche objet permet de palier cette difficulté. En effet, cette approche considère le système comme un ensemble organisé dont les éléments solidaires ne peuvent être définis que les uns par rapport aux autres. Dans cette approche, on identifie les différents éléments du système pour en faire des objets que l'on cherche à faire collaborer pour accomplir la tâche voulue. Contrairement à l'approche fonctionnelle, cette approche se base à la fois sur ce que le système fait et sur ce que le système est. Elle intègre donc à la fois la structure et le comportement.

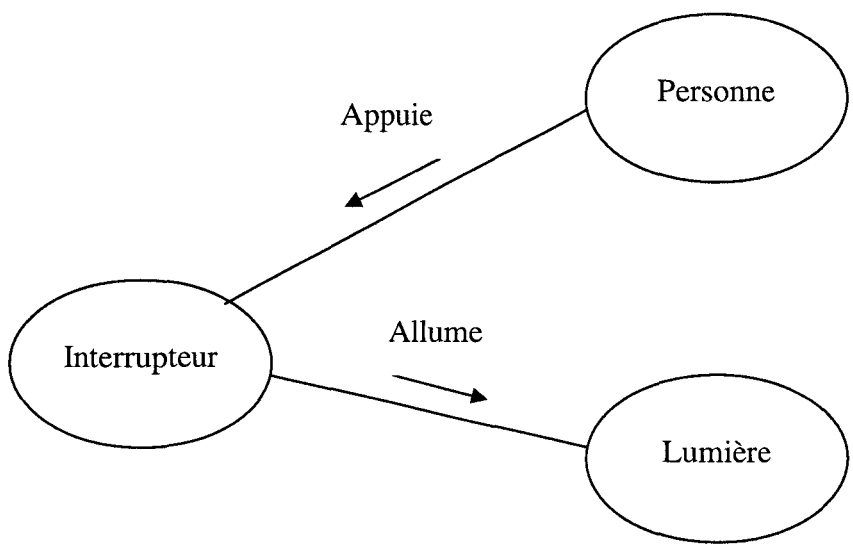


Figure 2.2 : décomposition objet intégrant la structure et le comportement

Dans le langage courant un objet représente une entité physique qui peut être vue ou touchée ou d'une manière plus générale un élément qui peut être perçu par l'esprit. Dans le domaine de l'informatique, un objet est toujours considéré comme étant une entité, mais cette entité évolue dans un monde logiciel et non pas dans le monde réel. Ainsi, les objets logiciels peuvent représenter un objet physique mais également un concept ou une idée plus abstraite. L'approche objet permet donc de formaliser notre perception du monde et des phénomènes qui s'y déroulent en préservant la structure et le comportement du système à analyser. Les fonctions sont alors représentées par des formes de collaboration entre les objets qui composent le système, le couplage devient dynamique et les évolutions fonctionnelles ne remettent plus en cause la structure du logiciel.

La programmation orientée objet est régie par 3 principes fondamentaux : l'***encapsulation***, l'***héritage*** et le ***polymorphisme***.

1.1.1 – Encapsulation :

Le terme d'***encapsulation*** exprime à lui seul le concept même d'objet : réunir sous la même entité les données et les moyens de les gérer. Il ne s'agit donc plus de déclarer des données générales puis un ensemble de fonctions et de procédures destinées à les gérer de manière indépendante, mais de réunir sous une même entité la structure et le comportement : ce que le système est et ce que le système fait. Lorsque l'on passe de l'approche fonctionnelle à l'approche objet, on ne parle plus de *variables*, de *fonctions* ou de *procédures* mais d'*attributs* et de *méthodes*.

- Les *attributs* sont à l'objet ce que les variables sont à un programme : ce sont eux qui ont en charge les données à gérer. Tout comme n'importe quelle autre variable, un *attribut* peut posséder un type quelconque défini au préalable : nombre, caractère, ..., ou même un type objet.

- Les *méthodes* sont les éléments d'un objet qui servent d'interface entre les données et le programme. Sous ce nom obscur se cachent simplement des procédures ou fonctions destinées à traiter les données.

La figure 2.3 illustre le principe d'encapsulation.

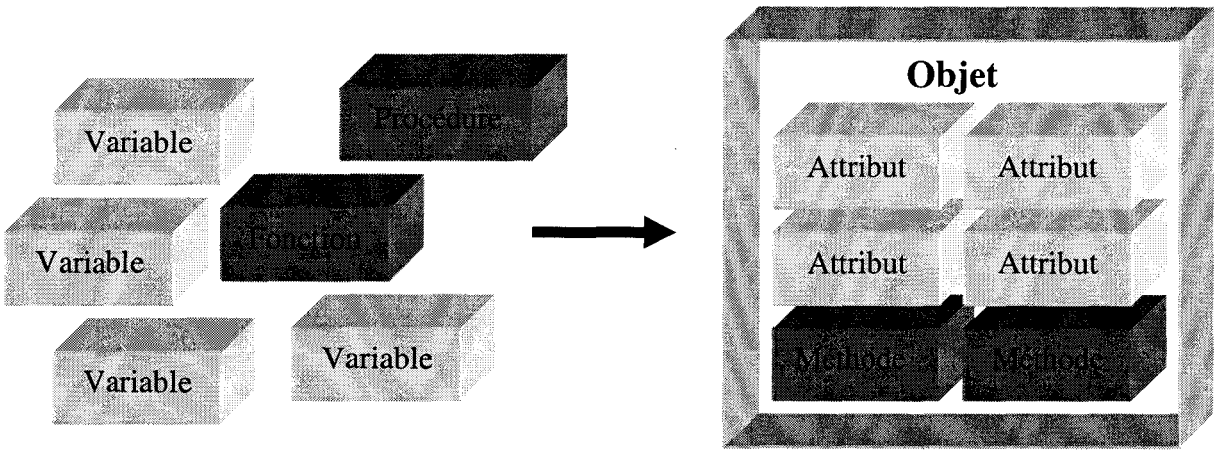


Figure 2.3 : le principe d'encapsulation

Le concept d'encapsulation permet également de masquer aux yeux d'un programmeur utilisant l'objet tous les rouages de l'objet et donc l'ensemble des données, fonctions et procédures destinées à la gestion interne de l'objet. En effet, l'utilisateur de l'objet ne doit pas nécessairement savoir de quelle façon sont structurées les données dans l'objet. Cela signifie qu'un utilisateur n'a pas à connaître l'implémentation. L'encapsulation permet donc au concepteur de l'objet de masquer un certain nombre d'attributs et de méthodes tout en en laissant visible d'autres, interdisant ainsi à l'utilisateur de l'objet de modifier directement l'information et de mettre en péril ses données et ses propriétés comportementales.

1.1.2 – Héritage :

L'héritage constitue l'une des plus grandes forces de l'approche objet. L'héritage permet entre autre la ré-utilisabilité et l'adaptabilité des objets par la constitution de nouveaux types d'objets à partir d'objets existants. Son principe est en quelque sorte le même que celui d'un arbre généalogique ; les objets « descendant » héritent des caractéristiques des objets « parents » auquelles viennent s'ajouter de nouvelles caractéristiques. Ainsi, les attributs et méthodes déclarés dans la classe parent sont accessibles dans la classe enfant comme si ils avaient été créés localement.

Par ce moyen, on crée une hiérarchie d'objets de plus en plus spécialisés. La figure 2.4 illustre le principe d'héritage.

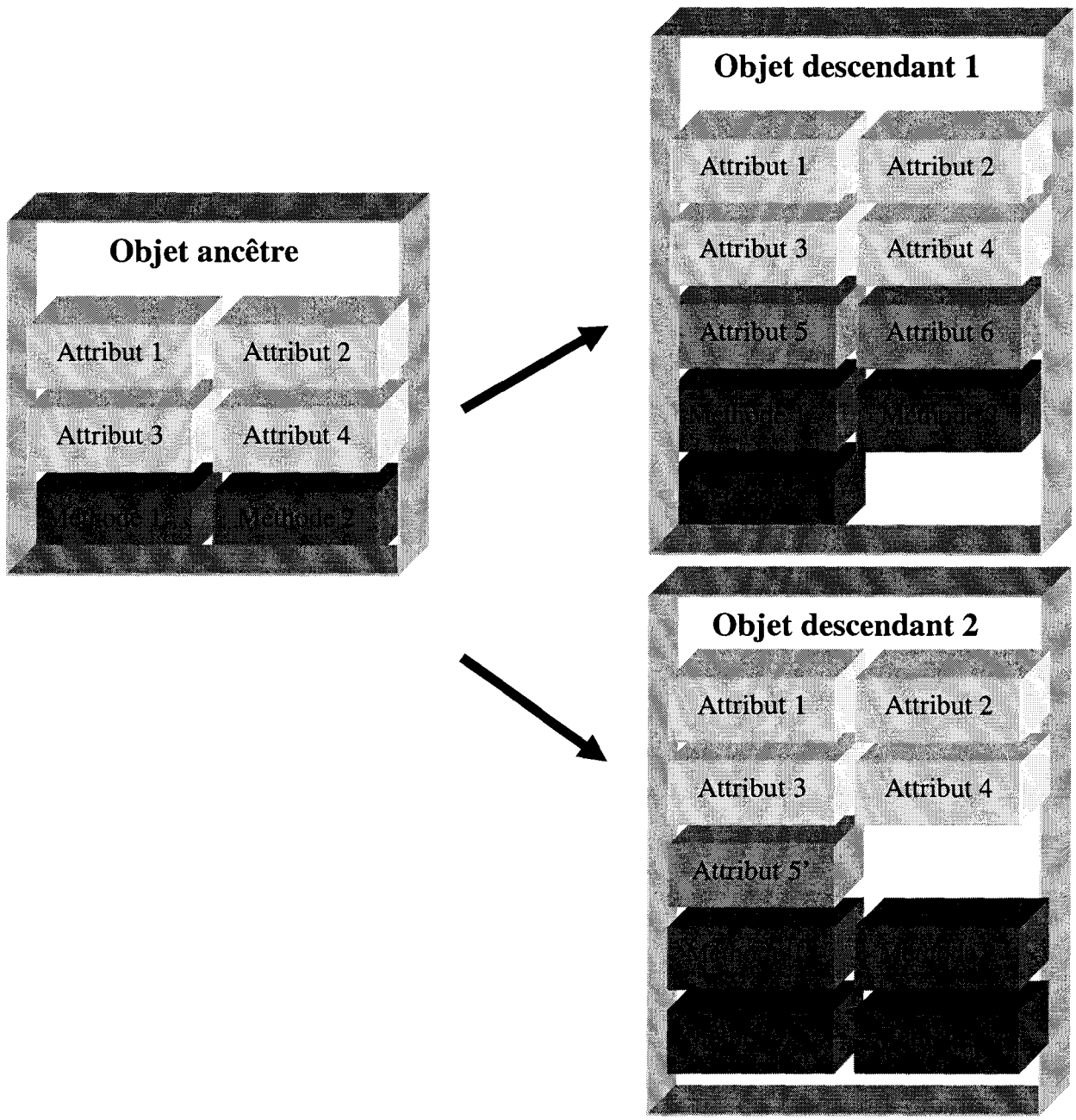


Figure 2.4 : le principe d'héritage

1.1.3 – Polymorphisme :

Le terme **polymorphisme** est certainement le plus difficile à appréhender. Le **polymorphisme** traite de la capacité de l'objet à posséder *plusieurs formes*. Cette capacité dérive directement du principe d'héritage. En effet, tout objet qui hérite des attributs et méthodes de ses ancêtres objets garde toujours la capacité de pouvoir **redéfinir une méthode** soit en la réécrivant soit en la complétant. Le comportement de l'objet devient donc modifiable à volonté.

Le **polymorphisme**, en d'autres termes, est donc la capacité du système à choisir dynamiquement la méthode qui correspond au type réel de l'objet en cours. Ainsi, si l'on considère un objet *Véhicule* et ses descendants *Bateau*, *Avion*, *Voiture* possédant tous la méthode *Avancer*, le système appellera la fonction *Avancer* spécifique suivant que le véhicule est un *Bateau*, un *Avion* ou bien une *Voiture*.

1.2 –Les objets :

1.2.1 – Caractéristiques :

Selon l'Object Management Group (OMG) un objet est « une entité qui possède une identité unique, un ensemble d'opérations qui peuvent lui être appliquées et un état modifié par ces différentes opérations ». Concrètement, un objet est une structure de données valuées répondant à un ensemble de messages. Un objet est donc constitué de trois composantes de base : une identité, un état et un comportement.

OBJET = IDENTITE + ETAT + COMPORTEMENT

L'identité permet de distinguer deux objets qui auraient un comportement et un état identiques. Elle est unique et n'autorise ni doublons (deux objets ne peuvent pas posséder la même identité) ni changement (l'identité d'un objet ne peut pas changer au cours du temps). Elle permet donc à un objet d'être référencé par d'autres, de façon unique et constante pendant toute sa durée de vie.

L'état valorise l'objet à un instant précis. Il est souvent variable et définit les différentes conditions que l'objet peut prendre durant sa durée de vie. L'état regroupe les valeurs instantanées de tous les attributs de l'objet. L'exemple suivant montre un objet voiture qui contient les attributs couleur, poids, puissance fiscale et son volume de carburant.

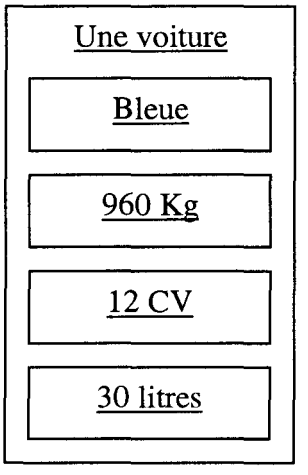


Figure 2.5 : l'état associe les valeurs des différents attributs qui caractérisent l'objet

L'état évolue au cours du temps. Si nous reprenons l'exemple précédent, si une voiture roule, son volume de carburant diminue. Cependant, certaines composantes de l'état peuvent être constantes : la couleur de la voiture et sa puissance fiscale ne varient pas en fonction du nombre de kilomètres parcourus. Le schéma suivant illustre cet exemple.

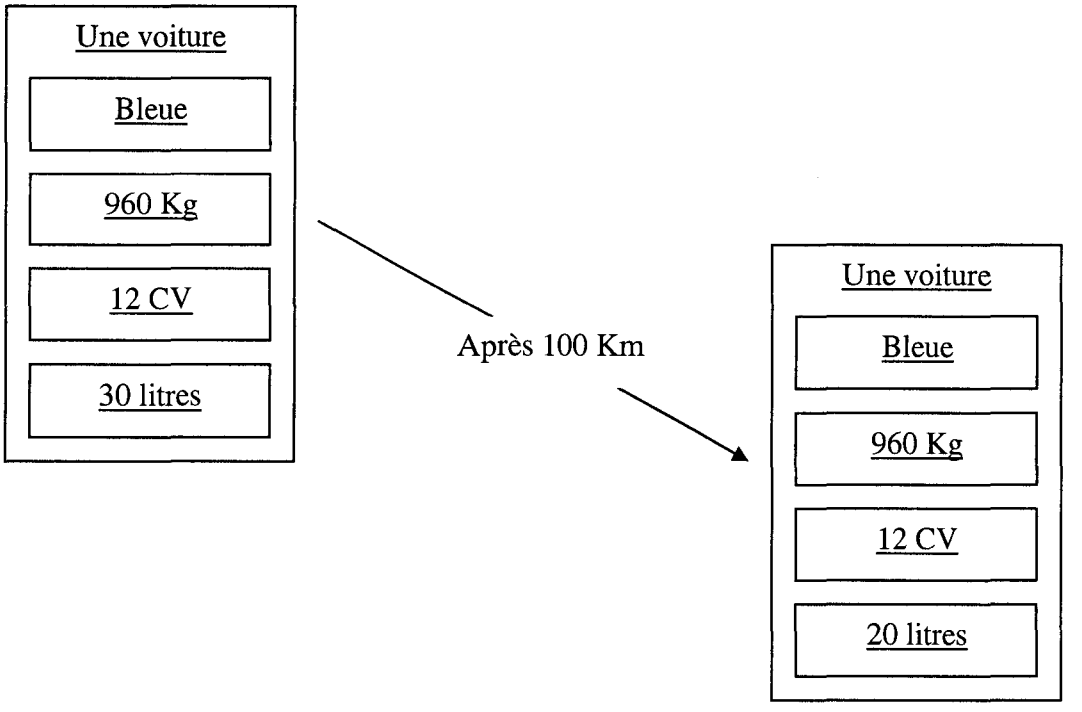


Figure 2.6 : évolution de l'état au cours du temps

Le comportement représente la façon dont un objet particulier réagit à la réception de certains messages. Il définit donc l'ensemble des opérations (méthodes) que peut effectuer l'objet. Le comportement et l'état dépendent l'un de l'autre : le comportement d'un objet dépend de son état à un instant donné et l'état de l'objet peut être affecté par son comportement. Reprenons l'exemple de la voiture. Il n'est possible de démarrer la voiture que si celle-ci est arrêtée. Le comportement « ***Démarrer*** » n'est donc valide que si l'information « ***Arrêtée*** » est valide. Après le démarrage, l'information « ***Arrêtée*** » devient invalide et l'opération « ***Démarrer*** » n'a plus de sens. L'exemple suivant illustre les liens entre l'état et le comportement : un conducteur ne peut démarrer une voiture que si elle est arrêtée et ne peut l'arrêter que si elle est démarrée.

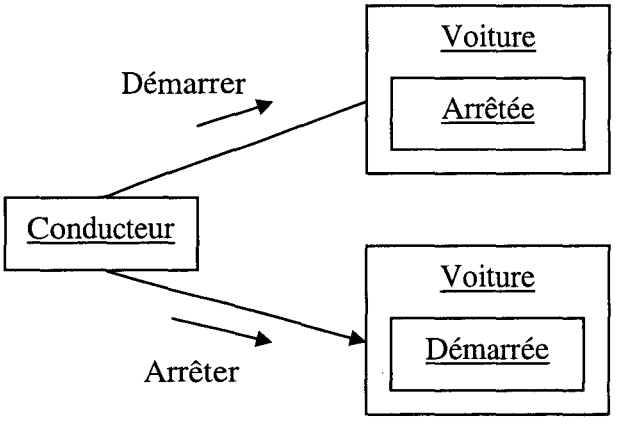


Figure 2.7 : Liaison entre l'état et le comportement

1.2.2 – Classes d'objets :

Une classe d'objets décrit une abstraction d'objets ayant des propriétés similaires, un comportement commun, des relations identiques avec les autres objets et une sémantique commune (OMG). Une classe est donc un type de données abstrait, caractérisé par des propriétés (attributs et méthodes) communes à plusieurs objets possédant ces propriétés. Tous les objets sont donc des « instances » de classe. C'est-à-dire qu'une classe peut être considérée comme un moule à partir duquel on peut créer des objets.

1.3 – Relations entre les objets :

Un système orienté objet peut être vu comme un ensemble d'objets s'échangeant des messages et collaborant entre eux pour la réalisation d'un but commun correspondant aux exigences fonctionnelles du système.

La façon dont les objets sont liés pendant l'exécution dépend directement de la façon dont les différentes classes d'objet sont liées dans la représentation logique du système. Ces relations entre différentes classes sont exprimées à l'aide de diagrammes de classes. On peut ainsi observer différents types de relations : l'association, l'agrégation et la généralisation.

- **L'association** :

L'association décrit une relation structurelle et précise que les objets instances d'une classe sont reliés aux objets instances d'une autre classe.

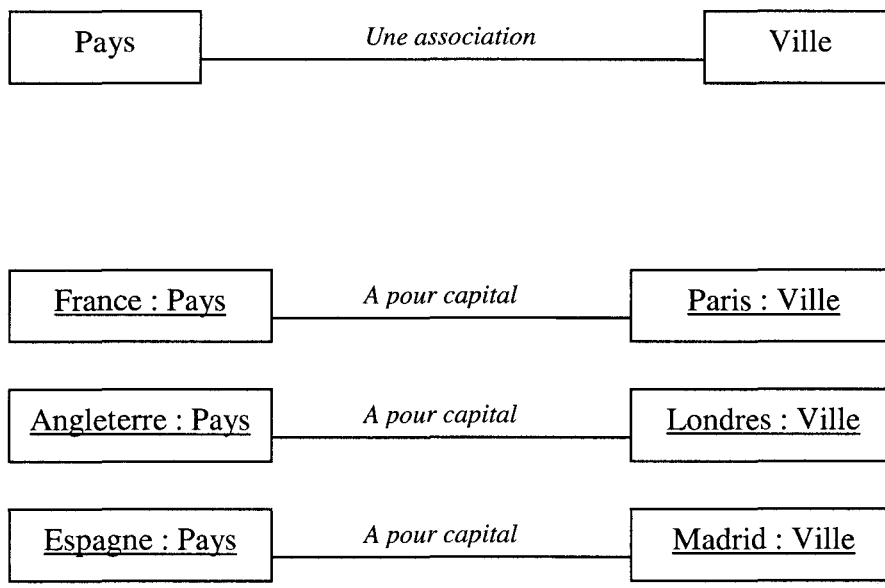


Figure 2.8 : les liens entre les Pays et les Villes sont tous des instances de l'association entre la classe « Pays » et la classe « Ville ».

L'association est bidirectionnelle. Pour une meilleure lisibilité, on peut la préciser par une forme verbale active ou passive et on indique le sens de lecture par les signes < ou >.

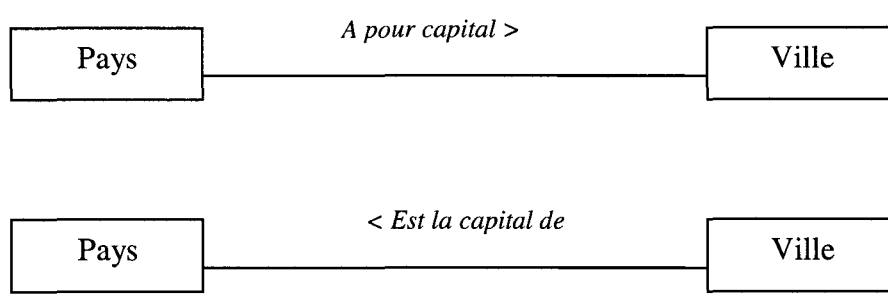


Figure 2.9 : clarification d'une association par forme verbale.

L'association peut également contenir une information de multiplicité. Le tableau suivant exprime différentes valeurs possibles de multiplicité :

1	Une et une seule
0..*	Plusieurs (0 ou plus)
0..1	0 ou 1
1..*	1 ou plus
m..n	De m à n

Figure 2.10 : valeur de multiplicité conventionnelle

Le diagramme suivant exprime le fait qu'une personne possède aucun ou plusieurs véhicules et qu'un véhicule appartient à une seule personne.

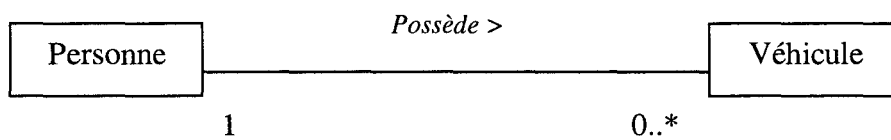


Figure 2.11 : exemple de représentation de multiplicité

- **L'agrégation** :

L'agrégation est une forme particulière d'association avec une sémantique plus forte. Elle représente des relations de type maître / esclaves ou 'composé' et 'composants'. L'exemple suivant montre qu'un Micro ordinateur est constitué d'un clavier, d'une unité centrale de 0 ou plusieurs moniteurs et de 0 ou une souris. La classe Micro-ordinateur joue ici le rôle d'agrégat.

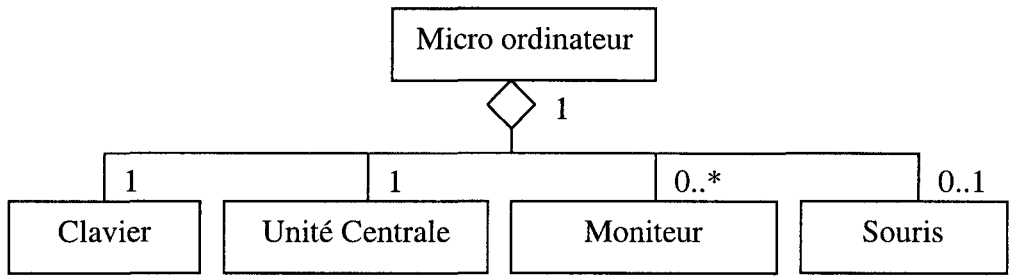


Figure 2.12 : exemple d'agrégation

Lorsque la multiplicité vaut 1 du côté de l'agrégat, la destruction de l'agrégat entraîne la destruction des composants. Dans l'exemple précédent, la destruction complète du Micro-ordinateur entraîne la destruction du clavier, de l'unité centrale, des éventuels moniteurs et de l'éventuelle souris.

- **La généralisation** :

La relation de généralisation est utilisée pour représenter une relation d'héritage. La généralisation consiste à factoriser les éléments communs de plusieurs classes en une classe plus générale. Cette méthode, consistant à hiérarchiser les classes, est délicate car les arbres de classes ne poussent pas à partir de la racine mais se déterminent en partant des feuilles qui appartiennent au monde réel alors que les niveaux supérieurs ne sont que des abstractions créées pour ordonner et comprendre.

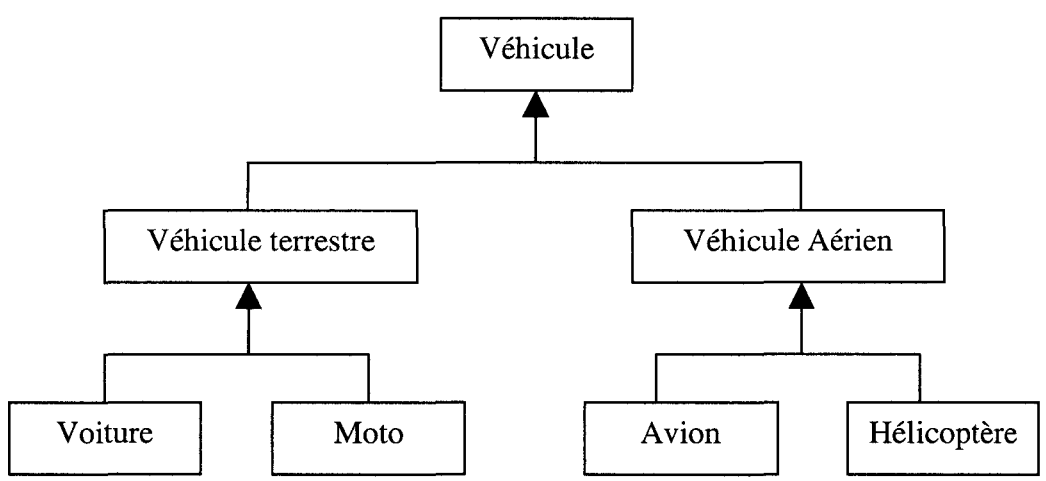


Figure 2.13 : exemple de hiérarchie de classes par généralisation

La généralisation n'autorise pas de multiplicité : dans l'exemple précédent, la voiture est un véhicule terrestre. Il ne lui est pas possible d'être plusieurs fois un véhicule terrestre.
La généralisation ne porte pas de nom particulier. Elle signifie toujours « est un » ou « est une sorte de ».
La généralisation est une relation non réflective : une classe ne peut pas dériver d'elle-même.
La généralisation est une relation non symétrique : si une classe **A** dérive d'une classe **B**, la classe **B** ne peut pas dériver de la classe **A**.
La généralisation est transitive : si une classe **A** dérive d'une classe **B** qui dérive elle-même d'une classe **C**, alors **A** dérive également de **C**. Dans l'exemple précédent, la voiture est un véhicule terrestre et, donc, un véhicule.

- Classe abstraite :

Une classe qui ne peut pas être instanciée est appelée « classe abstraite ». Le but de ces classes est de définir un cadre de travail pour les classes dérivées en proposant un ensemble de méthodes que l'on retrouvera tout au long de l'arborescence (cf chapitre 1.1.2 – Héritage).

Reprenons le diagramme de classe précédent.

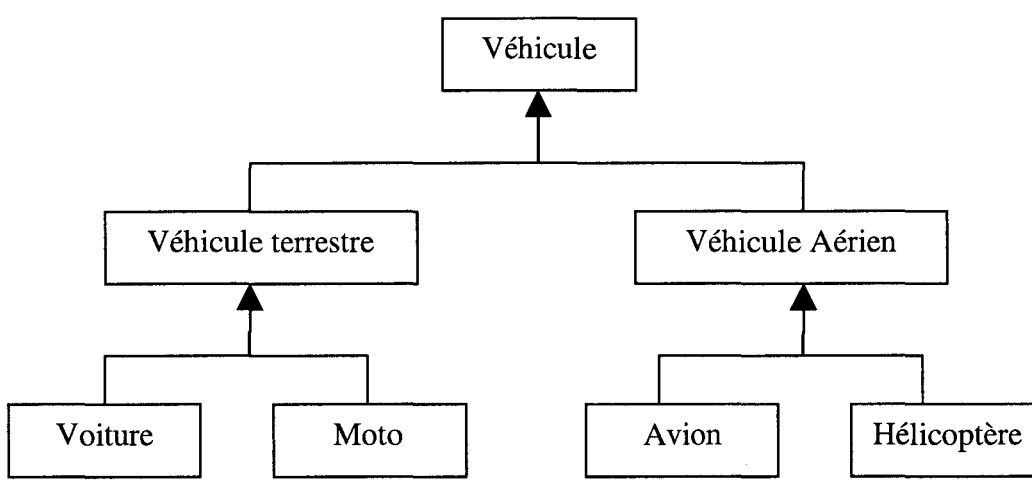


Figure 2.14 : exemple de hiérarchie de classes par généralisation

Si on considère la classe Véhicule, il parait tout à fait naturel qu'elle ne puisse pas avoir d'instance. Un véhicule au sens large du terme ne correspond en aucun cas à un objet concret

mais plutôt au concept d'un objet capable de démarrer, ralentir, accélérer ou s'arrêter que ce soit une voiture, une moto, un avion ou un hélicoptère.

1.4 – Framework : la bibliothèque de classes :

Un cadre d'application (en anglais, Application Framework) informatique ou cadriciel est un ensemble de classes abstraites collaborant entre elles pour faciliter la création de tout ou partie d'un système logiciel. Il fournit suffisamment d'éléments logiciels pour produire une application aboutie. Ces éléments sont organisés pour être utilisés en interaction les uns avec les autres et sont, en principe, spécialisés pour un type particulier d'applications. Un framework fournit un guide architectural en partitionnant le domaine visé en classes abstraites et en définissant les responsabilités de chacune ainsi que les collaborations entre classes. Le déploiement à grande échelle de bibliothèques d'objets exige un framework. Les principaux avantages des cadriciels sont la réutilisation des classes et la standardisation du cycle de vie[1] du logiciel. Ils permettent de formaliser une architecture adaptée aux besoins de l'application. Ils tirent partie de l'expérience des développements antérieurs.

1.5 – Les composants :

Dans l'interface de développement d'application Borland Delphi, le terme "composant" est employé pour désigner une classe qui hérite de la classe TComponent. On peut donc créer un composant Delphi grâce à l'héritage.
La figure suivante explique la hiérarchie de base de tous les composants :

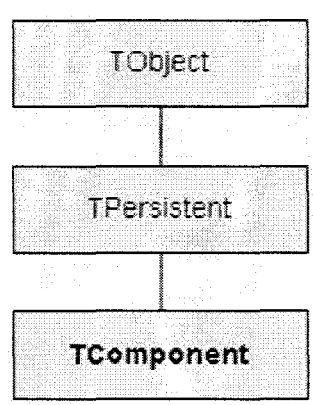


Figure 3.2 : hiérarchie de base des composants Delphi

[1] *Le cycle de vie d'un composant représente la période qui sépare sa création de sa destruction*

TObject est l'ancêtre implicite ou explicite de tous les objets Delphi. TObject encapsule le comportement fondamental commun à tous les objets en intégrant les méthodes qui effectuent les fonctions de base comme la création (Create), la maintenance ou la destruction (Destroy) d'une instance d'un objet.

TPersistent introduit la possibilité de sauvegarder l'état d'un objet. Les classes issues de TPersistent prennent en charge l'envoi de données aux flux et permettent l'affectation des classes.

TComponent spécialise TPersistant en y ajoutant la possibilité de gérer les noms et l'appartenance des composants. Cette classe inclut également la méthode Loaded qui permet de positionner du code après rechargement des propriétés.

Pratiquement, un composant est un bloc de construction que le programmeur peut adopter pour concevoir l'interface graphique ou fournir de nouvelles fonctionnalités à ses applications. Pour le concepteur de composants, un composant est en fait une classe en langage Pascal Objet qui réunit des données et des actions (propriétés et méthodes) permettant à l'utilisateur de traiter ses informations numériques.

2 – Application au domaine du traitement du signal : les architectures flot de données.

Un système de traitement numérique du signal a pour objet d'appliquer une série de transformations à un flot de données. Plutôt que de traiter le problème dans sa globalité comme dans les systèmes de type « boîte noire » habituellement utilisés, la POO organise le problème en plusieurs sous-problèmes traités séparément.

En effet, les différentes techniques, plus ou moins complexes, de traitement numérique du signal font appel à des méthodes connues et peuvent donc être considérées comme une implémentation modulaire de ces différentes méthodes [LOGI 95]. Selon ce principe, une méthodologie de conception d'algorithmes de traitement du signal reproduit le schéma d'une méthode d'analyse descendante et consiste à produire des graphes de tâches successives

utilisant des niveaux d'abstraction différents [CLOU ??]. Dans cette méthodologie, l'algorithme est envisagé selon deux axes :

- un axe horizontal qui décrit le problème sous forme de différentes tâches séquentielles ayant le même niveau d'abstraction.
- un axe vertical représentant un affinement de chacune des tâches à plusieurs niveaux d'abstraction.

Prenons l'exemple d'un algorithme de calcul d'une FFT ; un tel traitement peut être organisé en plusieurs tâches de la manière suivante :

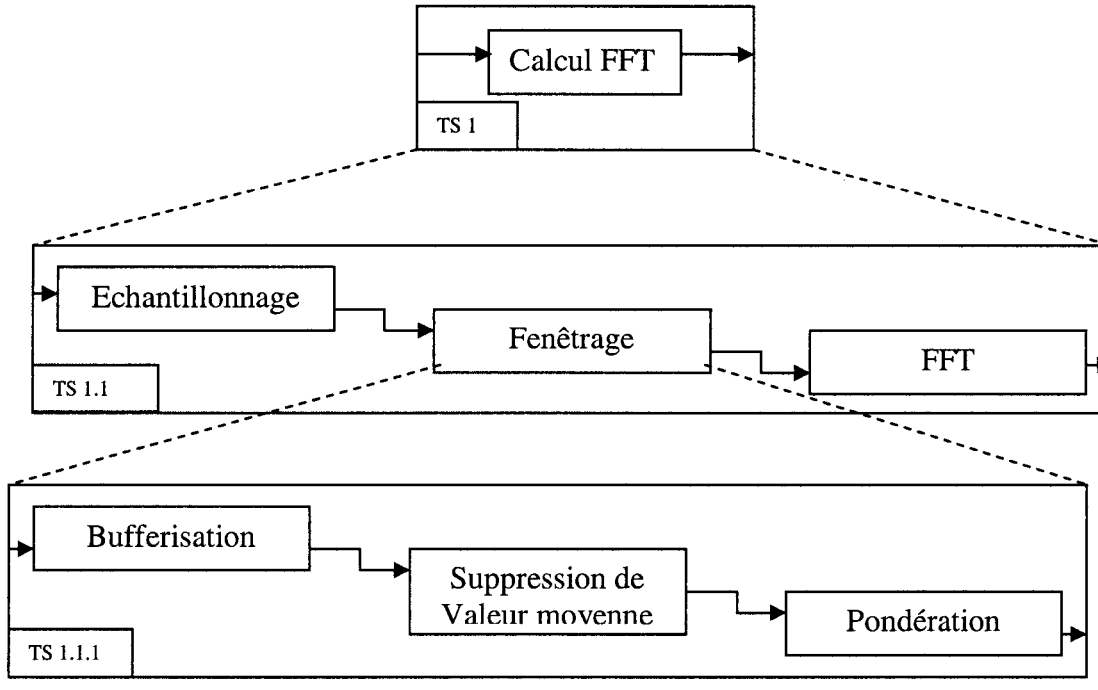


Figure 2.15 : décomposition d'un algorithme de calcul de FFT.

Ainsi, on obtient la description complète du calcul au niveau d'abstraction le plus haut et la définition de ces algorithmes en tâches puis en « sous – tâches ».
Dans l'exemple ci-dessus, les tâches élémentaires permettant de réaliser la tâche ***Calcul FFT*** sont les tâches ***Echantillonnage***, ***Bufferisation***, ***Suppression de la valeur moyenne***, ***Pondération*** et ***FFT***.

Ces différentes tâches sont donc interdépendantes et doivent être synchronisées pour un bon fonctionnement de l'outil de traitement final. Cette notion prend toute son importance

lorsqu'il s'agit, comme c'est souvent le cas dans les problèmes de traitement numérique du signal et notamment dans le domaine du monitorage médical, de traiter les informations continûment.

Ce principe de définition de la transformation, organisé en un réseau de modules de traitement développés indépendamment de leur utilisation, permet d'obtenir un grand nombre de combinaisons possibles et donc d'augmenter le nombre de réutilisations possibles de chaque module élémentaire. Les fonctionnalités de l'application sont alors déterminées par :

- Les types de modules connectés,
- les interconnections entre les modules.

2.1 – Communication entre les modules :

La décomposition d'un système complexe en plusieurs sous-systèmes implique la mise en place de procédures d'échange d'informations entre ces sous-systèmes. Généralement, la communication entre ces entités (modules) s'effectue par passage de messages entre des ports de communication unidirectionnels. On distingue deux types de ports de communication :

- **Les ports d'entrée (récepteurs)** qui permettent à un module d'acquérir un message,
- **les ports de sortie (émetteurs)** qui permettent à un module de transmettre un message.

Le paradigme de flot de données met l'accent sur la détermination des entrées en exigeant que les sorties d'un module soient explicitement liées aux entrées d'un (des) autre(s) module(s).

Ainsi, un port d'entrée ne peut être connecté qu'à un et un seul port de sortie. En revanche, un port de sortie peut transmettre des messages à plusieurs ports d'entrée.

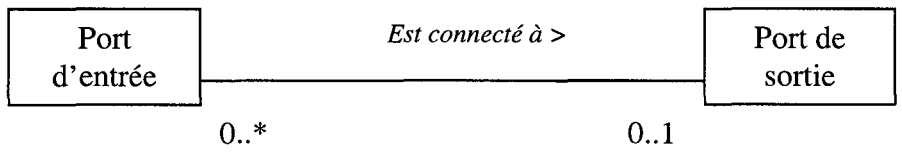


Figure 2.16 : connexion entre les ports

Pour pouvoir être connectés, un port d'entrée et un port de sortie doivent être compatibles, c'est-à-dire que le message émis par le port de sortie doit être du même type que le message attendu par le port d'entrée.

2.1.1 – Messages :

Dans la plupart des cas, les messages possèdent une partie descripteur contenant des informations générales sur le message - comme le type, la taille ou le nom de la source - et une partie données contenant les données à traiter. Certains messages, comme les messages de contrôle, peuvent encapsuler toute l'information nécessaire dans le descripteur. Dans ce cas précis le message contient uniquement le descripteur.

La figure 17 donne un exemple de structure pour les deux types de messages.

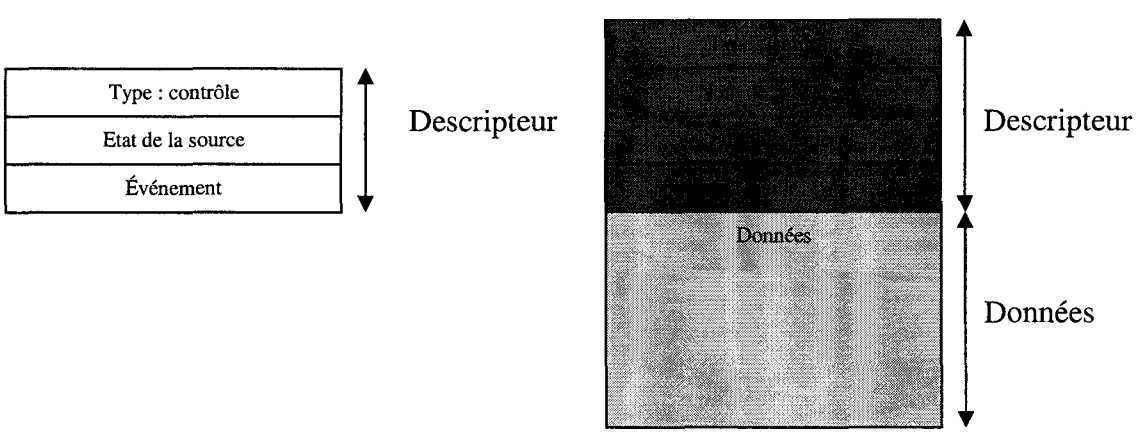


Figure 2.17 : exemples de message.

2.1.2 – Transmission des messages :

Le passage de messages d'un composant à l'autre, via les ports de communication, impose le choix d'une méthode adaptée. On distingue pour cela deux grands types de modèles :

- Le modèle PULL (orienté demandes) :

Pour ce type de communication, le récepteur envoie une requête à l'émetteur à l'aide d'un appel de procédure ou de fonction qui retourne le message comme résultat. Dans ce type de modèle, le transfert de message est donc initialisé par le récepteur qui, par conséquent, détermine le contrôle du flot de données. Ce type de modèle est peu adapté aux communications faisant intervenir des événements asynchrones ou priorisés.

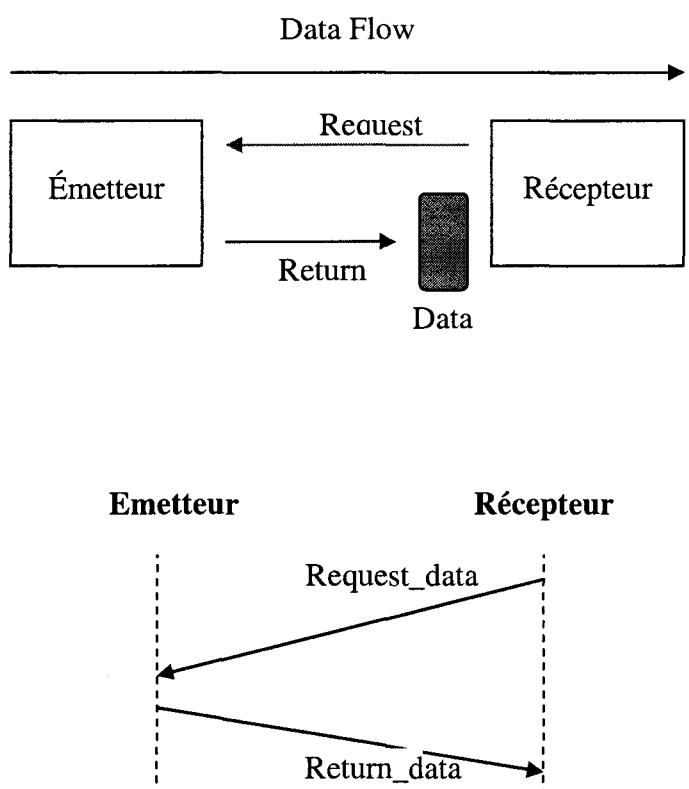


Figure 2.18 : Modèle Pull pour la communication entre ports

- Le modèle PUSH (orienté événements) :

Pour ce type de communication, l'émetteur envoi un message au récepteur dès qu'une nouvelle valeur est disponible. Habituellement, l'émetteur ne sait pas si le récepteur est prêt à recevoir le message. Pour palier cette difficulté et éviter la perte de données, on ajoute un tampon (buffer) de réception à l'entrée du récepteur. Si l'émetteur délivre des messages asynchrones ou prioritaires, le récepteur doit être capable de les identifier et de les positionner en tête du tampon de réception en décalant les messages de plus faible priorité. Cette dernière particularité implique la mise en place d'un dispositif d'ordonnancement des messages.

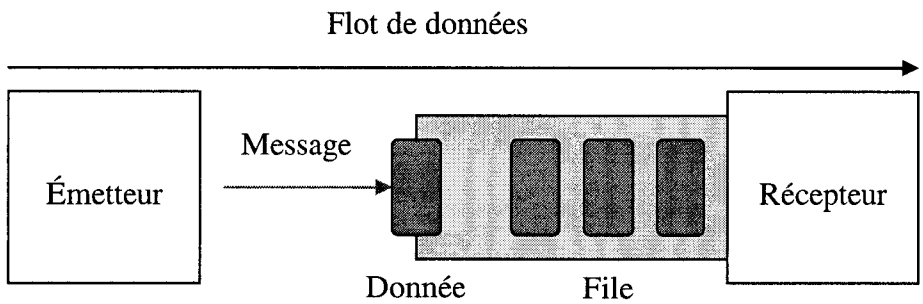


Figure 2.19 : modèle Push pour la communication entre ports

Pour ce dernier type de modèle il convient donc de modifier le message de la figure 18 en ajoutant dans le descripteur, son niveau de priorité. De cette façon, le récepteur pourra aisément réorganiser son tampon et ainsi traiter les messages dans l'ordre imposé par leur priorité.

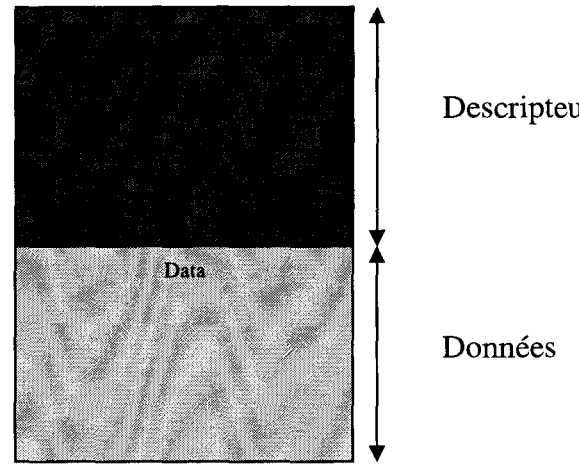


Figure 2.20 : exemple de message.

2.2 – Représentation :

Un tel paradigme peut être représenté sous la forme d'un graphique composé de nœuds et d'arcs, où les nœuds de type actions représentent un calcul, une source de données, une sortie... et les arcs représentent les flots de données c'est-à-dire les transferts de données entre les noeuds.

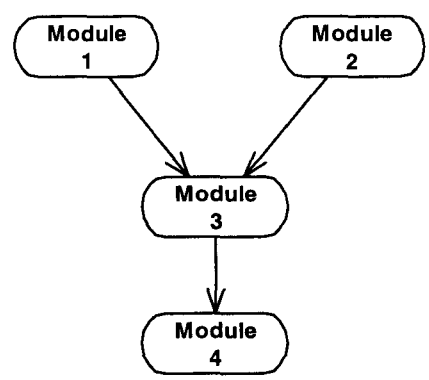


Figure 2.21 : représentation du paradigme « flot de données ».

Dans le modèle d'exécution du paradigme flot de données, l'exécution d'un nœud se produit dès que toutes ses entrées (récepteurs) sont disponibles ; c'est-à-dire que la tâche associée à un module est effectuée si et seulement si tous les récepteurs du module contiennent au moins une donnée. Ce type d'application est alors considéré comme orientée « données » (Data

Driven Application). Ainsi, les émetteurs récupèrent les paramètres de sortie d'un module et les transmettent immédiatement aux récepteurs d'autres modules via les arcs de flots de données.

Pour l'exemple précédent, le module 3 ne sera actif que lorsque ses récepteurs auront reçu les messages émis par les modules 1 et 2.

Les récepteurs reçoivent les valeurs des paramètres d'entrée correspondants. On distingue deux types de récepteurs :

- **Les récepteurs déclenchant :** pour de tels récepteurs, la donnée est consommée dès que le nœud est exécuté ce qui nécessite l'arrivée d'une nouvelle valeur pour une nouvelle exécution du nœud.
- **Les récepteurs non-déclenchant :** pour de tels récepteurs la donnée est conservée après exécution du nœud et remplacée à la réception d'un nouveau message. Ce type de récepteur nécessite une valeur par défaut et n'est pas décisif dans l'activation d'un nœud puisqu'il contient toujours une donnée.

Pour une meilleure compréhension du modèle de représentation, on précise également les ports de communication.

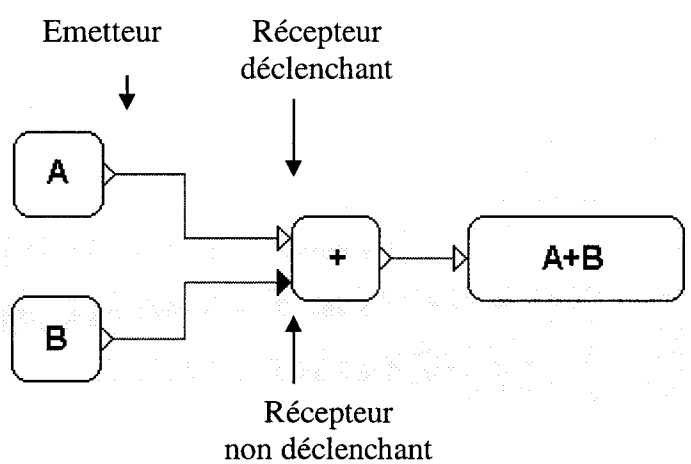


Figure 2.22 : l'exemple ci-dessus représente une addition. Ce schéma est donc composé de deux sources de données (A et B), d'un additionneur (+) et d'une sortie (A+B).

Etant donné que l'additionneur possède un récepteur déclenchant et un récepteur non déclenchant, la somme (A+B) sera recalculée à chaque fois qu'une donnée sera disponible sur le premier récepteur (la donnée du second étant toujours disponible) et envoyée à la sortie.

Ce type de programmation étant très proche de la programmation orientée objet, il est possible de créer plusieurs occurrences d'un même module.

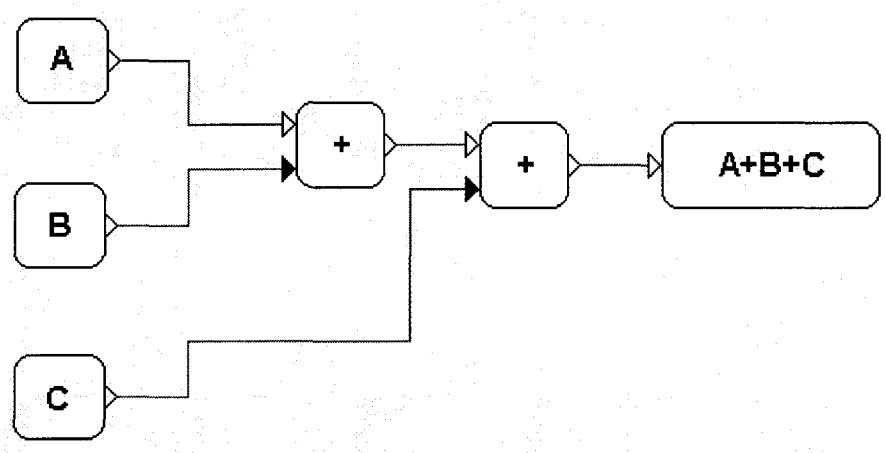


Figure 2.23 : l'exemple ci-dessus représente une double addition. Ce schéma est donc composé de trois sources de données (A, B et C), de deux additionneurs (+) et d'une sortie (A+B+C).

Cette représentation constitue une interface de création graphique simple permettant aux créateurs d'applications non spécialistes de concevoir simplement des algorithmes complexes.

2.3 – Composition statique :

De la même façon que la chaîne de modules, de l'entrée à la sortie, fournit une application complète de traitement du signal, l'interconnexion de différents modules indépendamment des entrées / sorties peut fournir, par composition statique (agrégation), de nouveaux modules de traitement plus élaborés et plus spécifiques, ainsi que l'illustre la figure 2.24.

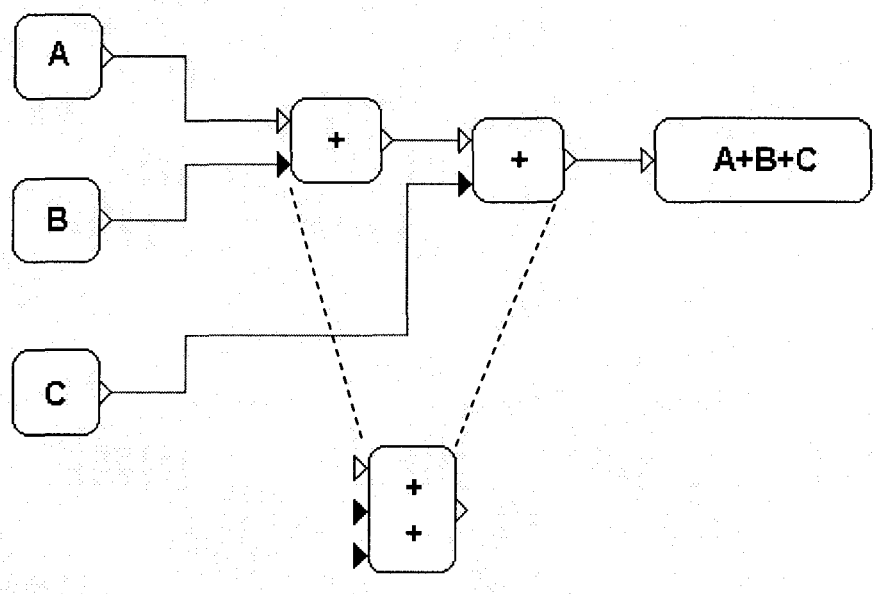


Figure 2.24 : composition statique.

En effet, deux modules de traitement de fonctions f(x) et g(y) peuvent être compactés, par composition statique, en un seul module de traitement de fonctions h = g(f(x)) = g¤f(x). Cette particularité permet la redéfinition d'une séquence de modules en un module primaire réduisant ainsi le nombre de communications entre les modules (augmentation des performances).

3 - Conclusion

Dans ce chapitre, nous avons présenté la programmation orientée objet et ses avantages par rapport à la programmation fonctionnelle ainsi que le paradigme de programmation par flots de données. Ce paradigme de programmation fournit un outil particulièrement adapté aux développements d'applications d'acquisition et de traitement du signal. Ainsi, la création d'une bibliothèque de composants logiciels de traitements numériques du signal, interconnectables indifféremment, permet la réalisation de systèmes complexes. Cette bibliothèque est donc initialement constituée de composants correspondant aux tâches élémentaires réalisant un système de traitement complet (filtrage, opérations arithmétiques, seuillage...). La réutilisation et l'interconnexion de ces composants élémentaires permettent également de créer, par composition statique, des composants de traitement du signal plus complexes venant à leur tour enrichir la bibliothèque dans le but d'une réutilisation.

Chapitre 3

Définition d'une architecture logicielle pour l'acquisition et le traitement du signal

1 – Introduction 58
2 – Définition des différentes classes 58
3 – Communication entre les classes 60
4 – Communication avec la source de données 64
4.1 – Principe général 64
4.2 – Extraction de l'information 65
4.3 – Distribution des données 66
5 – Spécificité des systèmes temps réel 67
6 – Implémentation : création des classes 69
6.1 - Les classes TSource et TVoie 69
6.2 – La classe TTraitements 72
6.3 - La classe TSortie 72
7 – Contrôle 73
7.1 - Les états locaux 73
7.2 - Règles de transition 75
7.3 – exemple 76
8 - Conclusion 77

1 – Introduction

Dans ce chapitre nous présentons une architecture logicielle pouvant servir de base au développement de tout type d'application d'acquisition et de traitement du signal. Basée sur le paradigme de programmation orientée objet présenté précédemment, cette architecture générique constitue un ensemble de classes et de sous-classes qui, par généralisation, donneront naissance à différents composants logiciels spécialisés pour une application de traitement du signal particulière.

2 – Définition des différentes classes

L'**architecture logicielle** décrit d'une manière symbolique et schématique les différents composants d'un ou de plusieurs programmes informatiques ainsi que leurs interrelations (association, agrégation, généralisation...) et leurs interactions. Le travail de définition de l'architecture du système consiste, dans un premier temps, à définir l'ensemble des classes de modules constituant le dispositif à réaliser (dans notre cas un dispositif d'acquisition et de traitement du signal). Conformément à la méthode de décomposition de chaînes de traitements en tâches élémentaires (cf § II.1), tout algorithme de traitement numérique du signal peut se décomposer en trois phases :

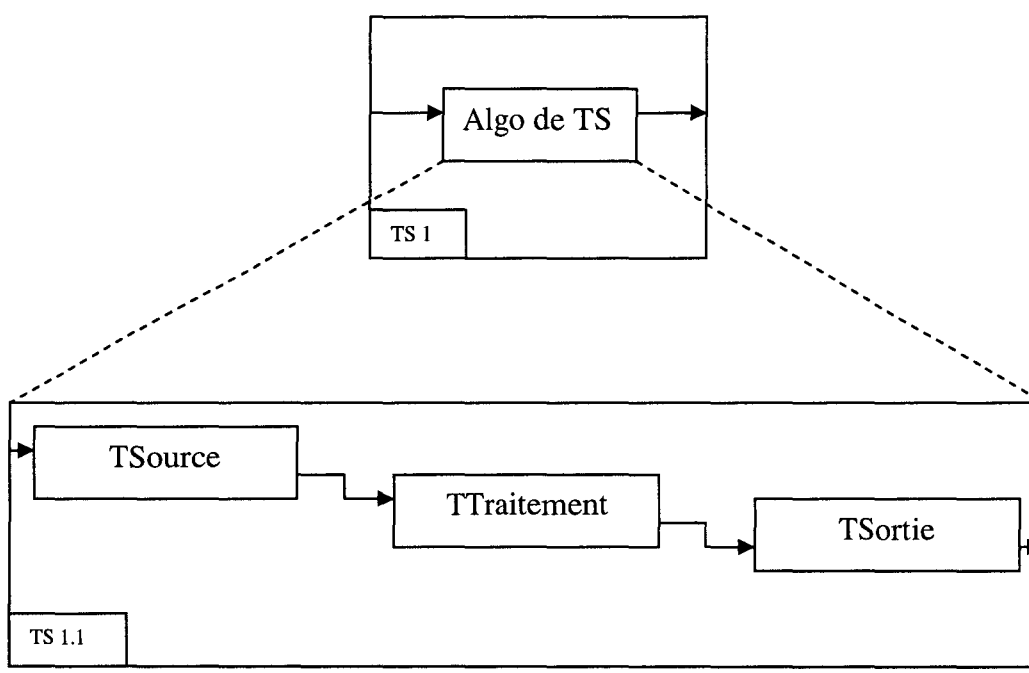


Figure 3.1 : décomposition d'un algorithme de TS.

On distingue donc trois classes de modules (figure 3.1) :

- **Les Sources** qui permettent d'acquérir le signal à partir de différents supports (fichiers, connexion USB ou Ethernet...), de le coder et enfin de le transmettre aux autres modules.

- **Les Traitements** qui effectuent une ou plusieurs transformations sur les données d'entrées et transmettent les résultats aux modules suivants.

- **Les Sorties** qui permettent d'afficher, de sauvegarder, de transmettre à un dispositif externe les données après transformations.

Les Sources sont chargées de l'interfaçage avec la source de données au sens physique. Ces sources de données peuvent être des fichiers mais également un matériel distant comme des modules d'acquisition temps réel ou des moniteurs d'anesthésies.

Les Traitements correspondent à des séries de transformations appliquées aux flux de données, telles que, par exemple, les opérations de filtrage, de détection de cycle, de mesure... **Les Traitements** constituent la part la plus importante de la bibliothèque de composants.

Enfin, **Les Sorties** correspond à l'interfaçage avec les sorties au sens physique (affichage, sauvegarde, transmission...).

Chacun des composants de cette bibliothèque est donc nécessairement une classe descendante de l'une des classes que nous venons de définir.

- TSource
- TTraitement
- TSortie

3 – Communication entre les classes

La figure 3.2 peut se décomposer de la manière suivante :

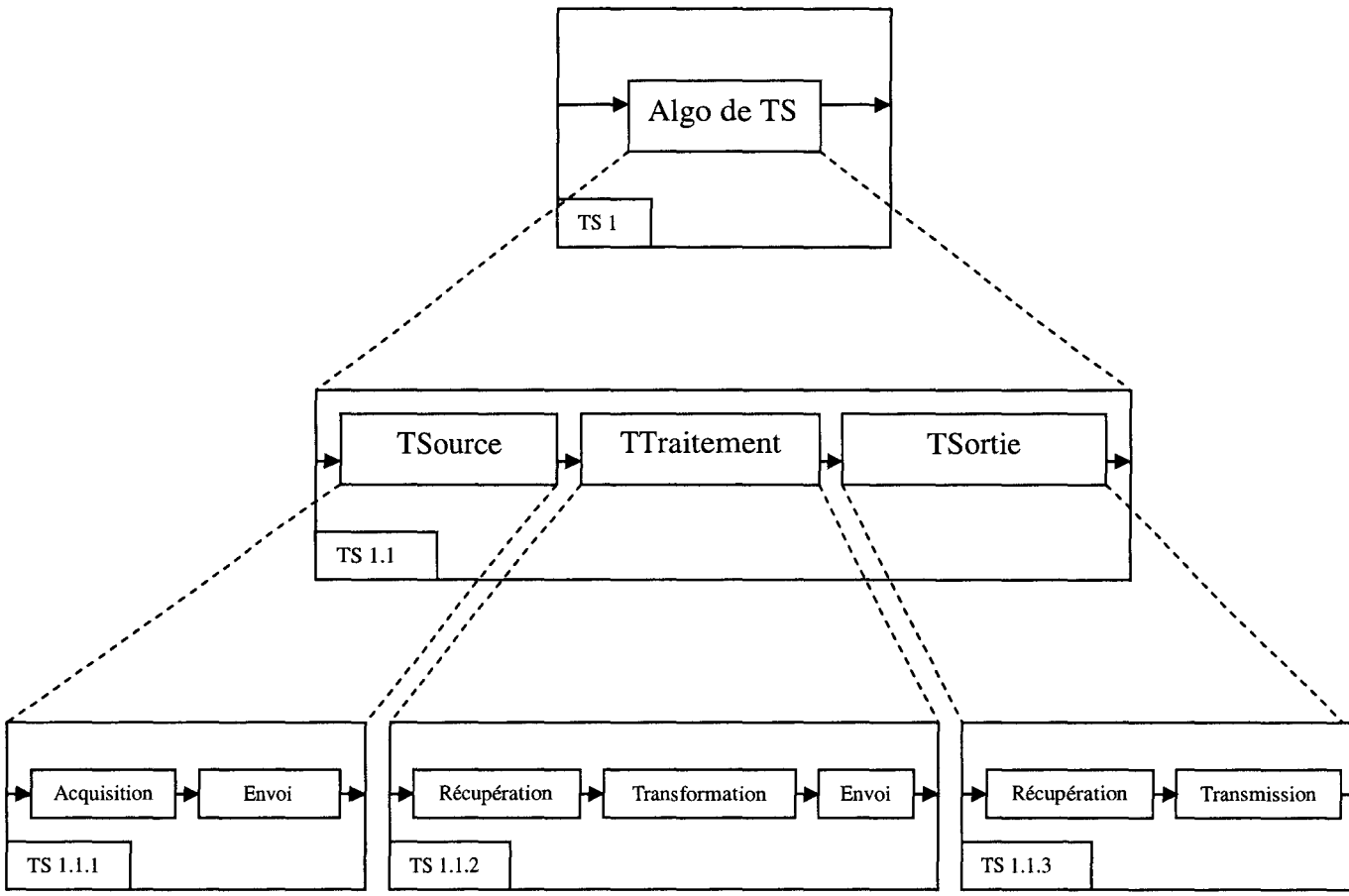


Figure 3.2 : décomposition d'un algorithme de TS.

Les sous-tâches « Envoi » et « Récupération » impliquent la conception de modules logiciels de communication (ou Port). Comme nous l'avons présenté dans le chapitre précédent, ces modules permettent la transmission unidirectionnelle de messages d'une classe à l'autre.

Les ports réalisent la communication (envoi / récupération) entre deux modules de la chaîne de traitement du signal. On observe ainsi deux types de ports de communication : Les ports d'entrée (classe ***TintputPort***) et les ports de sortie (classe **TOutputPort**).

Comme nous l'avons vu dans le chapitre précédent, l'architecture flot de données implique qu'un port d'entrée ne peut être connecté qu'à un et un seul port de sortie. En revanche, un port de sortie peut transmettre des messages à plusieurs ports d'entrée.

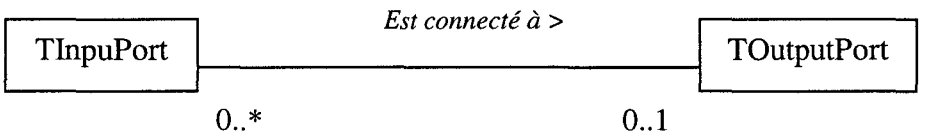


Figure 3.3 : connexion entre les ports

Pour intégrer cette particularité, la classe ***TInputport*** possède un attribut ***OutputPort*** lui indiquant le ***TOutputPort*** qui lui envoie les messages.

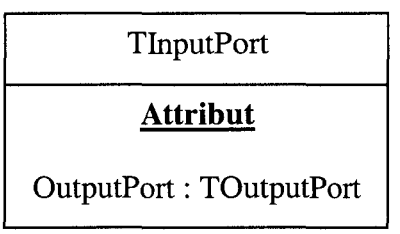


Figure 3.4 : définition de l'attribut TOutputPort

De la même façon, la classe ***TOutputport*** doit intégrer la liste des ***TInputPort*** à laquelle il doit transmettre. Il possède pour cela un attribut PortList instanciation d'une nouvelle classe ***TInputPortList***.

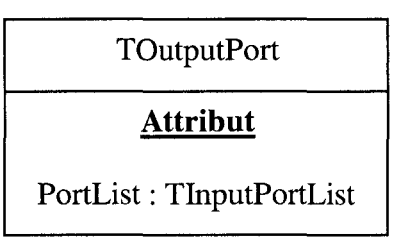


Figure 3.5 : définition de l'attribut TInputPortList

Ainsi, le port de sortie envoie chacun des messages dans l'ordre définit par la liste de ports d'entrée. Par ailleurs, on se donne la possibilité d'ajouter ou de supprimer des éléments de cette liste l'implémentation des méthodes ***AddInputPort*** et ***RemoveInputPort...***

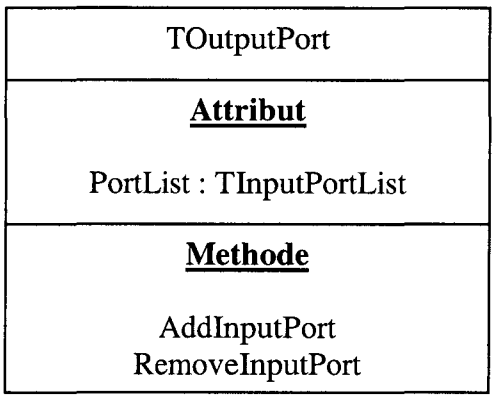


Figure 3.6 : Définition des méthodes d'ajout et de suppression de port d'entrée.

A tout moment, on peut modifier la séquence d'émission d'un port de sortie en ajoutant ou en supprimant un ou plusieurs ports d'entrée de la liste de ports (**TInpuPortList**) à l'aide des méthode ***AddInputPort*** et ***RemoveInputPort***. L'appel de ces deux méthodes se fait généralement au chargement ou à la destruction d'un composant ***TInputPort*** mais peut également être fait dynamiquement pendant l'exécution.

En effet, si le composant ***TInputPort*** et « connecté » à un ***TOutputPort*** ; c'est-à-dire si son attribut « OutputPort » est non nul, au chargement du composant on lance la méthode ***AddInputPort*** du composant ***OutpoutPort*** relié pour ajouter le nouveau port d'entrée dans sa liste.

La figure 3.7 illustre le déroulement de la connexion d'un nouveau port d'entrée à un port de sortie. Initialement (étape (1)), le port de sortie n'est connecté qu'à un seul port d'entrée. Lorsqu'on lui connecte un nouveau port d'entrée (étape (2)), le port d'entrée appelle la méthode ***AddInputPort*** du port de sortie pour mettre à jour la liste (étape (3)).

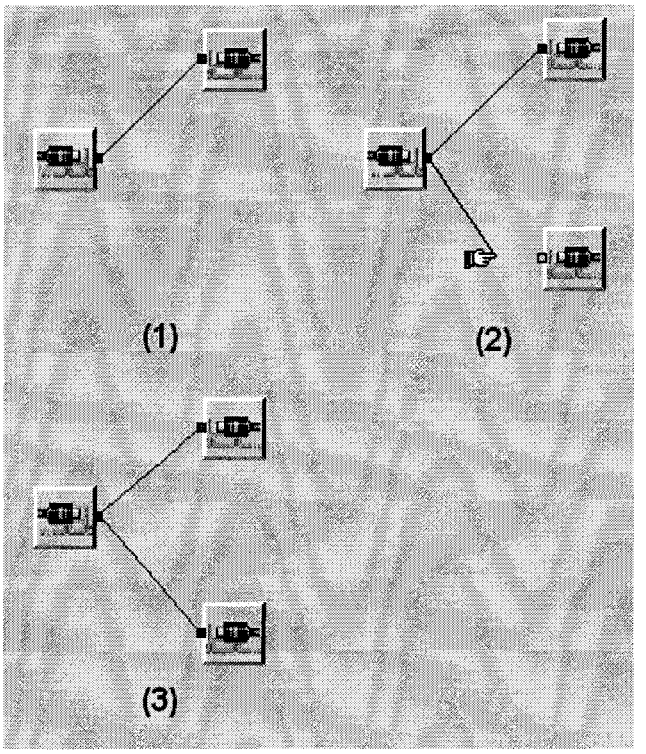


Figure 3.7 : ajout d'un port.

De la même façon, lors de la destruction, la méthode ***RemoveInputPort*** du composant ***OutpoutPort*** relié est appelée pour retirer de sa liste le port d'entrée détruit.

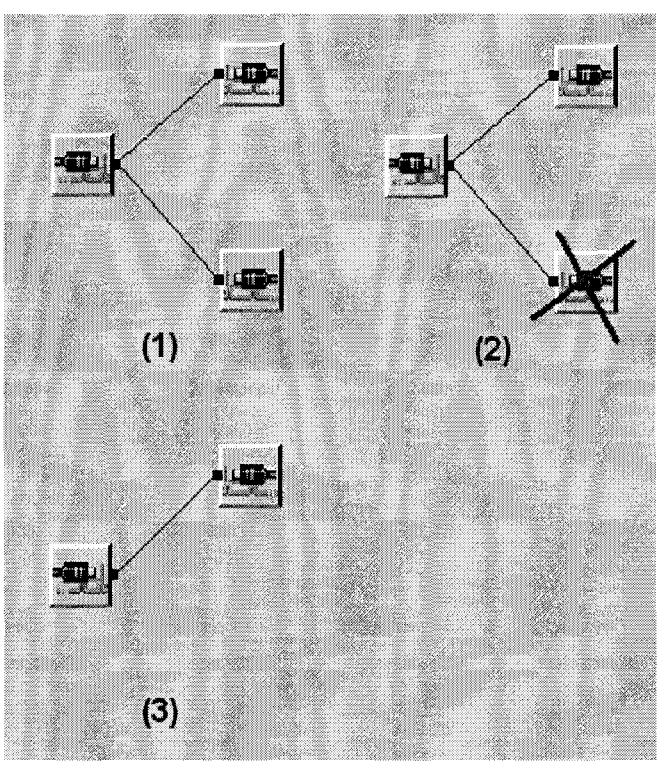


Figure 3.8 : suppression d'un port.

La figure 3.8 décrit le déroulement de la suppression d'un port d'entrée dans la liste d'un port de sortie. Initialement (étape (1)), le port de sortie est connecté à deux ports d'entrée. A la suppression du composant (étape (2)), le lien entre le port d'entrée du composant et le port de sortie est détruit également par l'appel de la méthode ***RemoveInputPort*** du port de sortie. La liste de port est alors automatiquement mise à jour (étape (3)).

Cette particularité permet de connecter ou de déconnecter les ports de communication et de supprimer ou d'ajouter de nouveaux composants de manière complètement dynamique.

4 – Communication avec la source de données

4.1 – Principe général :

La communication avec une source de données, qu'elle soit matérielle ou logicielle, pose le problème de l'interopérabilité. L'interopérabilité est la capacité que possède un système à fonctionner avec d'autres systèmes existants ou futurs. Il est donc important d'intégrer l'interopérabilité de la réalisation des éléments de communication avec les différentes sources de données.

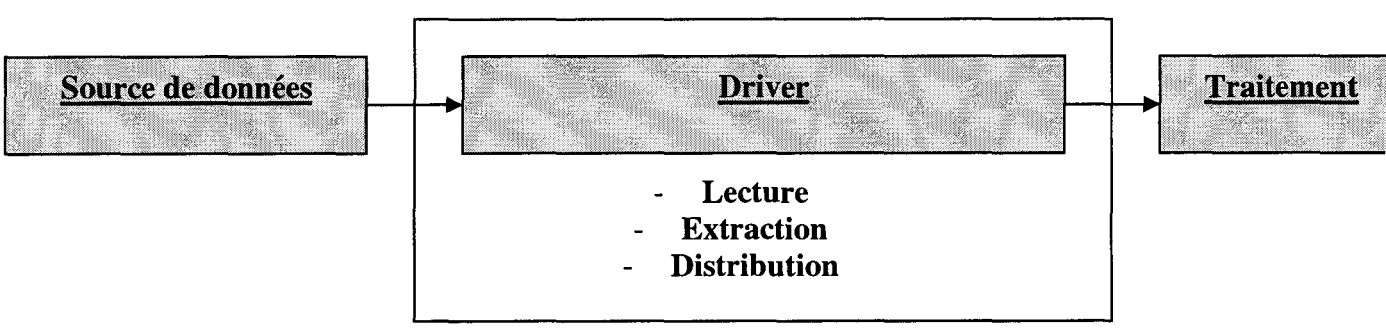


Figure 3.9 : Synoptique d'un système pilote (driver)

Le driver a donc pour objet de faire la « traduction » entre la source et l'application en lisant, décodant, réordonnant et distribuant les données de cette suite d'octets pour la transformer en information.

4.2 – Extraction de l'information :

Une source de données peut être un support soit matériel (réseau d'acquisition temps réel, sortie numérique d'un appareil...), soit logiciel (fichier, base de données...). Dans la plupart des cas, la source fournit des données codées et organisées (ou trame de données) selon un protocole spécifique. Il est donc important de développer un pilote (ou driver) permettant d'interpréter ce protocole.

D'une manière générale, un protocole de communication est une spécification de plusieurs règles pour un type de communication particulier. Communiquer consiste à transmettre des informations mais tant que les interlocuteurs ne peuvent pas attribuer un sens au message transmis, il ne s'agit que de données et pas d'information. Les interlocuteurs doivent donc non seulement parler un langage commun mais aussi maîtriser des règles minimales d'émission et de réception des données. : c'est le rôle d'un protocole. D'un point de vue logiciel, une source de données ne constitue qu'une suite d'octets.

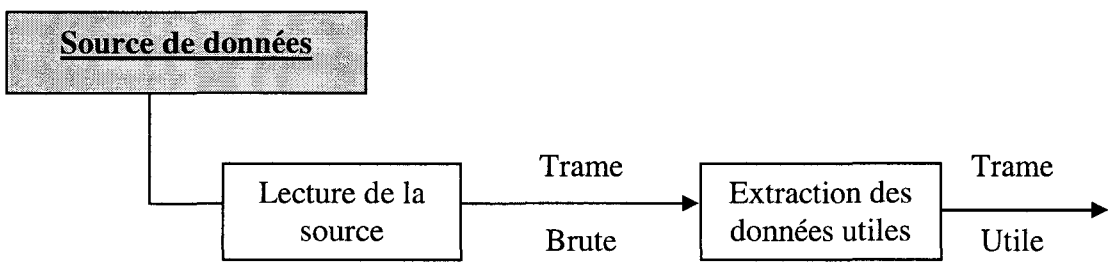


Figure 3.10 : extraction des données utiles

A titre d'exemple considérons une source de données comportant 4 signaux S1, S2, S3 et S4 avec S1, S3 et S4 ayant une résolution de 1 octet et S2 une résolution de 2 octets. La trame de données correspondant à un cycle de lecture se présente de la manière suivante :

| ? | ? | S1 | ? | S2 | S2 | ? | S3 | ? | S4 | (? -> Paramètres spécifiques du protocole inutiles pour l'application).

Les données utiles se trouvent aux positions 3, 5, 6, 8, 10 de la trame. L'extraction des informations utiles impose donc, dans ce cas, de conserver uniquement les cinq octets constituant la nouvelle trame « utile »:

| S1 | S2 | S2 | S3 | S4 |

4.3 – Distribution des données :

A partir de la trame utile il est nécessaire de séparer clairement les voies d'acquisition d'un point de vue logiciel : c'est le rôle d'un algorithme de distribution des voies d'acquisition.

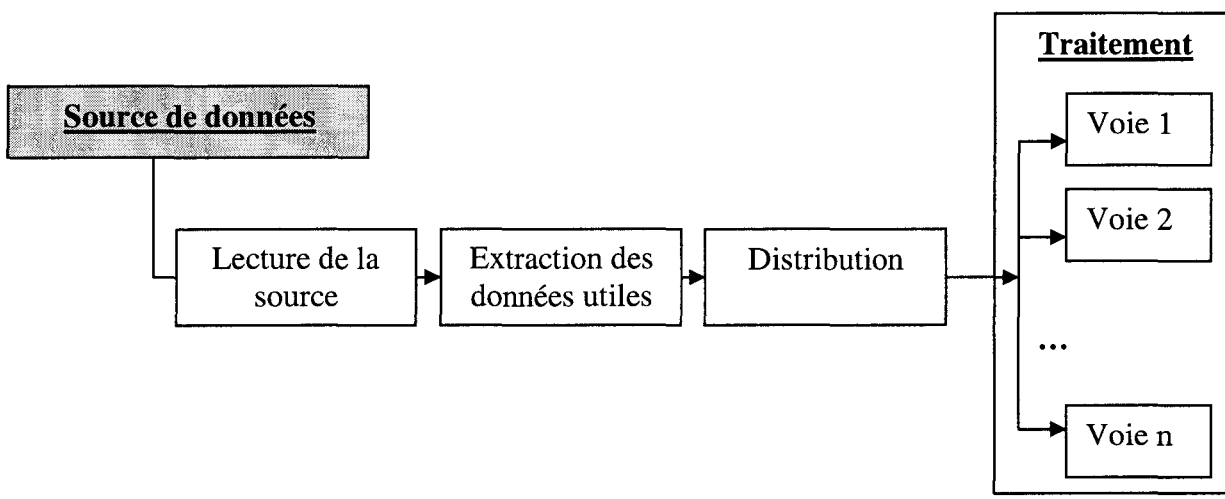


Figure 3.11 : extraction et distribution des données utiles

Cet algorithme de distribution logiciel constitue en fait un multiplexage des voies.

Il est à noter que l'algorithme de distribution ne se contente pas de retranscrire l'ordre apparaissant dans la trame utile mais il a aussi pour objet de redéfinir un ordre d'utilisation des signaux pour l'application.

Reprenons l'exemple précédent.

En sortie de l'algorithme d'extraction on obtenait la trame utile :

| S1 | S2 | S2 | S3 | S4 |

Soit une application imposant la configuration suivante :

- **Voie logicielle 1 : Numéro de signal = 1 (résolution = 1).**
- **Voie logicielle 2 : Numéro de signal = 3 (résolution = 1).**
- **Voie logicielle 3 : Numéro de signal = 2 (résolution = 2).**
- **Voie logicielle 4 : Numéro de signal = 4 (résolution = 1).**

Dans ce cas, le protocole de distribution envoie les données extraites aux différentes voies comme suit :

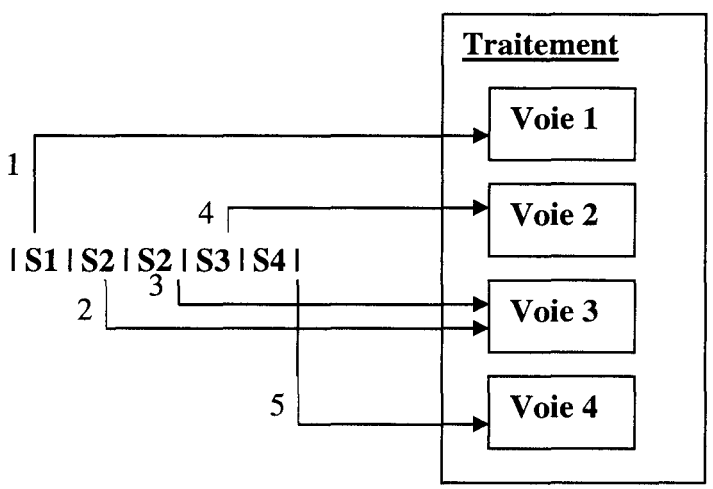


Figure 3.12 : distribution des données extraites.

5 – Spécificité des systèmes temps réel

La figure 3.11 illustre le principe d'extraction / distribution des informations à partir d'une source quelconque. Toutefois, pour appliquer ce principe à une source temps réel, il est nécessaire d'introduire une étape supplémentaire. En effet, par nature, les sources temps réel envoient continûment leurs données aux systèmes destinataires. Le plus souvent, le système destinataire est un ordinateur dont le système d'exploitation n'est pas un système temps réel. Dans ce cas, les différentes tâches nécessaires à l'extraction et à la distribution des données

d'entrées ne sont effectuées que lorsque le système est inoccupé. Il en résulte qu'on ne peut garantir ni le traitement continue des données ni leur intégrité. Pour palier cette difficulté, on introduit, en sortie de la source temps réelle, un tampon (buffer) de type « file accordéon » qui se remplit tant que le système est occupé et se vide lorsque le système est libéré. Le tampon doit être dimensionné de telle sorte que le temps correspondant au nombre d'échantillons dans le tampon soit supérieur au temps d'exécution de l'ensemble des tâches.

Afin de permettre la commande des systèmes d'acquisition temps réel (configuration, instructions de début et de fin de transmission), nous introduisons également un système d'envoi de commandes de l'ordinateur vers la source.

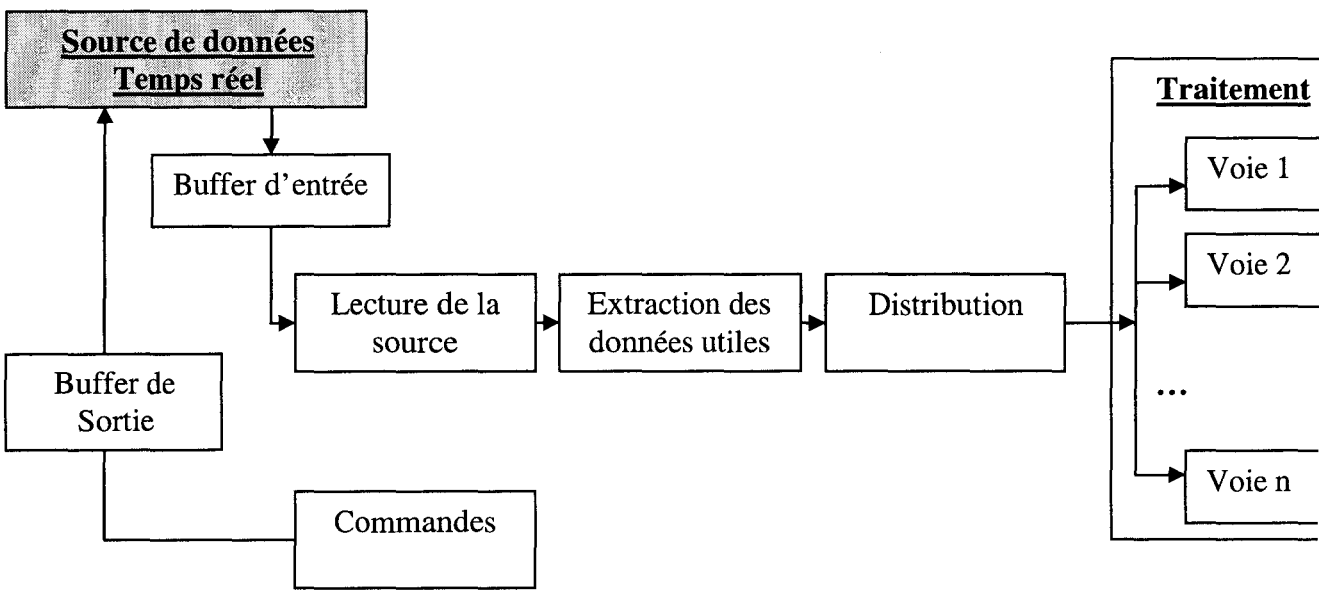


Figure 3.13 : Application aux systèmes temps réel

6 – Implémentation : création des classes

6.1 - Les classes TSource et TVoie :

La classe TSource regroupe un ensemble de composants logiciels capables de lire, comprendre, extraire et distribuer les signaux issus de tous types de sources de données. Quelle que soit la source, de tels composants nécessitent l'encapsulation des attributs et des méthodes permettant la lecture de la source, l'extraction et la distribution des données utiles à l'algorithme de traitement numérique du signal. Cependant, la plupart de ces fonctions dépendant directement du type de source de données à traiter, ces différentes méthodes et attributs appartiennent aux classes descendantes de la classe ***TSource*** spécifiques des sources de données.

La classe TVoie récupère chaque information envoyée par la classe ***TSource*** sous forme d'une suite d'octets et effectue les opérations nécessaires pour transformer cette suite d'octets en une valeur correspondant en une grandeur physique. Pour cela, il est nécessaire que cette classe intègre dans ses attributs les paramètres de quantification et d'échantillonnage du signal source. Nous avons choisi de réaliser cette quantification par une transformation linéaire. Il suffit donc de connaître le minimum et le maximum de la donnée envoyée par la source, le minimum et le maximum dans l'échelle de la grandeur physique, ainsi que le nombre d'octets sur lequel l'information est codée pour reconstituer la grandeur physique.

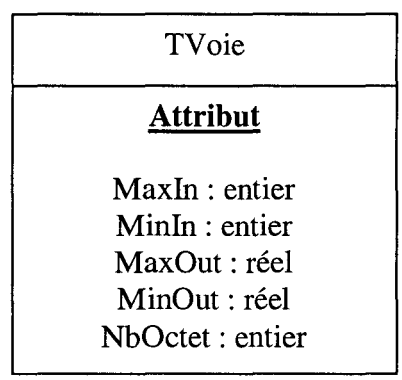


Figure 3.14 : définition des attributs de quantification

Le lien entre ***TVoie*** et ***Tsource*** est analogue au lien entre un ***TOutputPort*** et un ***TInputPort***. En effet, une voie d'acquisition ne peut être connectée qu'à une et une seule source de données. En revanche, une source de données peut transmettre des messages à plusieurs voies d'acquisition.

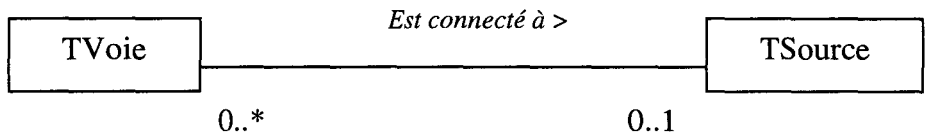


Figure 3.15 : connexion entre la source et les voies

La communication entre les classes ***TSource*** et ***TVoie*** est approximativement la même que celle entre les classes ***TOutputPort*** et ***TInputPort***.

Ainsi, chaque ***TVoie*** possède un attribut ***Source*** lui indiquant le ***TSource*** qui lui envoit les messages.

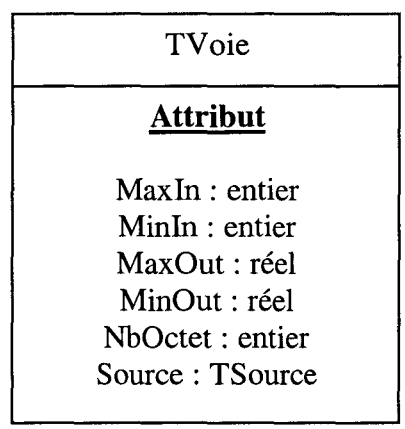


Figure 3.16 : Définition de l'attribut Source

D'un autre coté, chaque ***TSource*** doit intégrer la liste des ***TVoie*** à laquelle il doit transmettre. Ainsi, le composant source envoit chacun des messages dans l'ordre imposé par la liste.

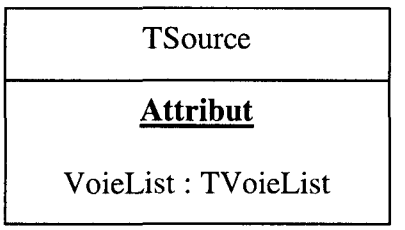


Figure 3.17 : définition de l'attribut VoieList

Tout comme pour les ports de communications, cette structure de type liste chaînée implique l'implémentation de méthode d'ajout et de suppression de ***TVoie*** pour chaque ***TSource***.

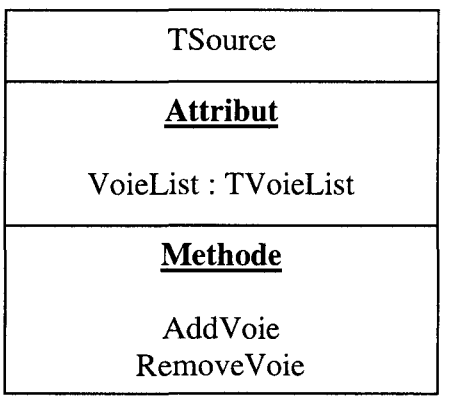


Figure 3.18 : définition des méthodes d'ajout et de suppression de Voie.

Le composant ***TVoie*** contient également un Port de sortie ***TOutputPort*** pour la communication de la donnée aux composants de traitement.

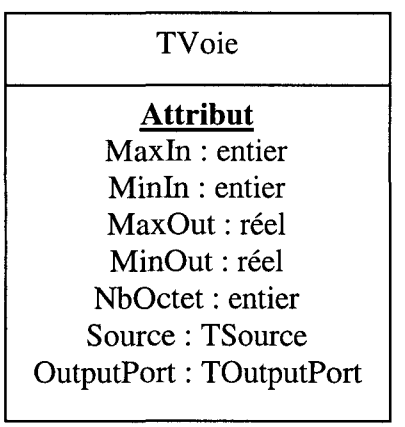


Figure 3.19 : définition de l'attribut OutputPort

6.2 – La classe TTraitements:

Les composants issus de la classe ***TTraitement*** permettent d'effectuer différentes transformations sur le flot de données. Ils possèdent un ou plusieurs ports d'entrée (***TInputPort***) et un ou plusieurs ports de sortie (***TOutputPort***).

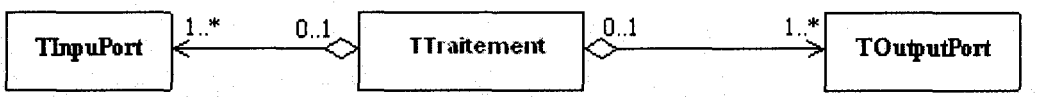


Figure 3.20 : diagramme de la classe TTraitement.

Dès qu'une donnée est disponible sur chacun des ports d'entrée du composant, celui-ci effectue les transformations permettant le calcul des données de sortie (méthode ***Calcul***). Une fois le calcul effectué, les données sont transmises aux autres composants par l'intermédiaire des ports de sortie.

6.3 - La classe TSortie :

Les composants issus de la classe TSortie réalisent l'affichage (scope, display...) ou la transmission (fichiers, base de données, réseau TCP/IP...) des données traitées. Ils utilisent donc un protocole de sortie (TOutputProtocol) et possèdent un ou plusieurs ports d'entrée (TInputPort).

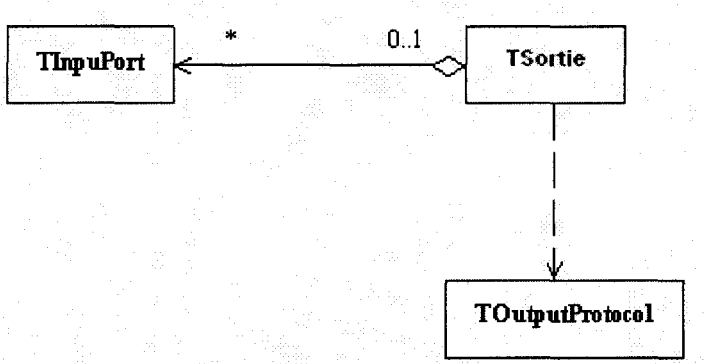


Figure 3.21 : diagramme de la classes TOutput.

Cette classe regroupe un ensemble de composants logiciels capables de coder et de transmettre l'information à tout type de sortie (système de base de données, afficheur, imprimante papier etc...). Quelle que soit la sortie, de tels composants nécessitent l'encapsulation des attributs et des méthodes permettant le codage et la distribution des données utiles à la sortie. Cependant, la plupart de ces fonctions dépendant directement du type de sortie connectée, ces différentes méthodes et attributs appartiennent aux classes descendantes de la classe ***TSortie***.

Pour pouvoir interfacer avec des systèmes externes (ex : base de données), ces composants doivent fournir une information compréhensible par la plupart des dispositifs médicaux. Il convient donc d'adopter un formalisme pour la création des messages [OMG 04] fournis par ces composants comme, par exemple, le formalisme HL7 [HL7 99] spécialement conçu pour l'interopérabilité des dispositifs médicaux.

7 – Contrôle

Les spécificités du temps réel et de l'interactivité nécessitent la mise en place d'une structure de contrôle de la chaîne de traitements ([GULL 96], [BENO 03]), afin d'augmenter la robustesse du système en analysant, en temps réel, son état et de fournir les moyens de réagir à chaque détection de défaut.

7.1 - Les états locaux :

La figure 3.22 illustre le schéma de base des différents états que peut prendre un composant de la bibliothèque au cours de son cycle de vie.

A sa création, un composant est ***inactif***. En effet, celui-ci a nécessité un certain nombre d'actions (configuration, mise en route...) avant de devenir ***opérationnel***. Après réception d'un message de configuration (***config***), le composant passe à l'état configuré. Pour être actif et passé dans l'état ***opérationnel***, le composant attend alors l'arrivée d'un événement ***start***.

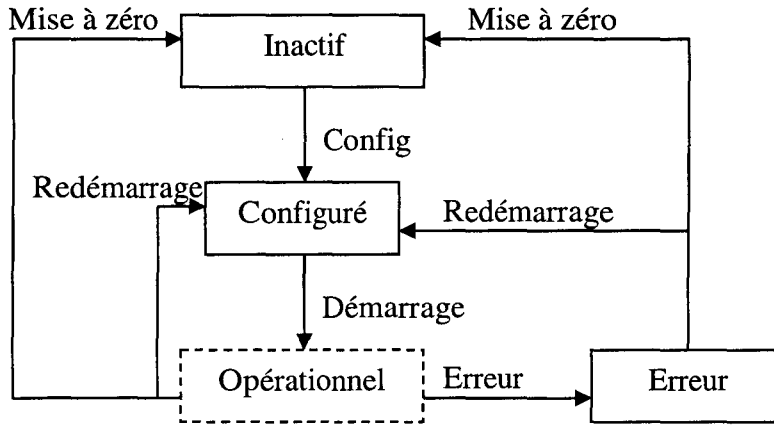


Figure 3.22 : Diagramme d'états - transitions.

L'état « ***Opérationnel*** » du composant signifie que celui-ci est apte à recevoir, traiter et transmettre des messages. Cependant, cet état se divise en plusieurs sous-états (figure 3.23). En effet, durant cette étape, le composant peut attendre la réception d'une donnée sur l'un de ses ports d'entrée, effectuer différents calculs sur les données reçues ou transmettre ces données.

Le mode Opérationnel se divise alors en trois sous-états : « ***Réception*** », « ***Calcul*** » et « ***Envoi*** ».

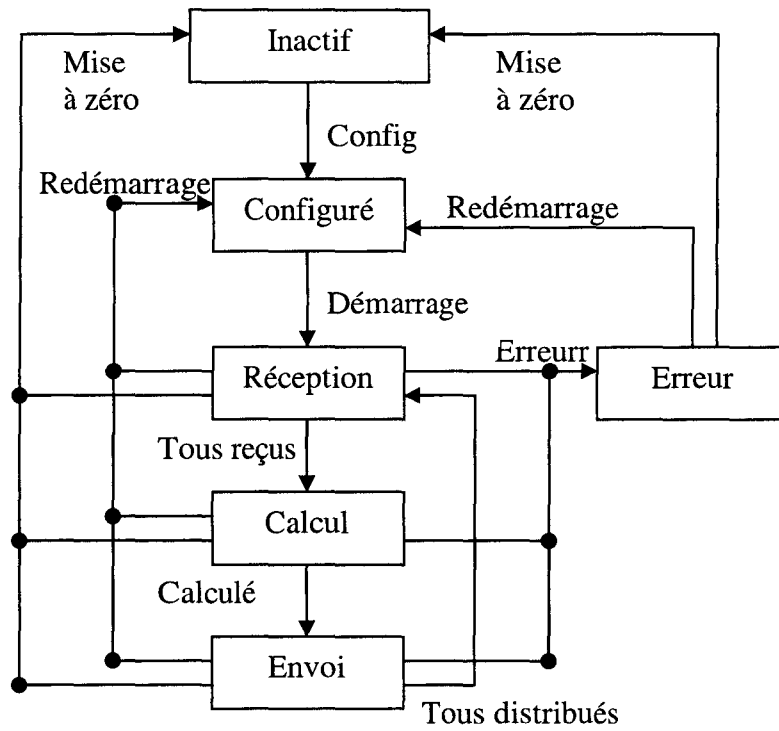


Figure 3.23 :diagramme d'états - transitions.

Ainsi, l'évolution de l'état de chaque composant est déterminé par son état actuel et par l'état des composants de son voisinage (c'est-à-dire les composants qui lui sont connectés). On peut alors déterminer des règles de transition locales ou synchronisantes applicables à tous les composants issus de la classe TTraitement :

7.2 - Règles de transition

- **R1 :** à sa création un composant est dans l'état « ***Inactif*** ».
- **R2 :** le passage de l'état « ***Inactif*** » à l'état « ***Configuré*** » (événement « ***Config*** ») se fait si tous les ports d'entrée du composant sont connectés et si tous les composants connectés à ces ports sont opérationnels (état « ***Opérationnel*** »).
- **R3 :** le passage de l'état « ***Configuré*** » à l'état « ***Réception*** » se fait à la réception d'un événement « ***Démarrage*** ».
- **R4 :** le passage de l'état « ***Réception*** » à l'état « ***Calcul*** » se fait dès que le composant dispose de toutes les données nécessaires pour effectuer un cycle de calcul (ce nombre de données est donc spécifique à chaque composant).
- **R5 :** le passage de l'état « ***Calcul*** » à l'état « ***Envoi*** » se fait automatiquement, dès que le calcul a été effectué.
- **R6 :** le passage de l'état « ***Envoi*** » à l'état « ***Réception*** » se fait dès que tous les ports de sortie ont distribué leur messages à tous les ports d'entrée qui leur sont connectés.
- **R7 :** le composant passe en état « ***Erreur*** » si le composant détecte une anomalie dans la réception des données (ex : message invalide) ou dans son calcul (ex : division par 0).
- **R8 :** tout changement des paramètres d'un composant (ex : fréquence de coupure pour un filtre) provoque un événement « ***Redémarrage*** » et le passage à l'état « ***Configuré*** ».
- **R9 :** si un composant passe en état « ***Erreur*** » tout composant connecté à l'une de ses sorties passe en état « ***Erreur*** ».
- **R10 :** si un composant passe en état « ***Inactif*** » tout composant connecté à l'une de ses sorties passe en état « ***Inactif*** » (événement « ***Mise à zéro*** »).
- **R11 :** la suppression d'un composant provoque le passage à l'état « ***Inactif*** » (événement « ***Mise à zéro*** ») de tout composant connecté à l'une de ses sorties.

- **R12 :** la suppression d'un composant provoque le passage à l'état « ***Inactif*** » (événement «***Mise à zéro*** ») de tout composant connecté à l'une de ses entrées. Une fois cet état atteint, chaque composant ainsi modifié détermine à nouveau l'ordre d'envoi des messages (cf §III) et passe automatiquement à l'état « ***Configuré*** » (événement « ***Config*** »).

7.3 – exemple :

A titre d'exemple, l'incidence de l'action « suppression d'un composant » sur les autres composants est détaillée ci-dessous :

La suppression d'un composant de la chaîne de traitement rend la donnée d'entrée inaccessible pour les composants situés en aval dans le flux de données et nécessite un ré-ordonnancement du dispositif d'envoi des messages pour les composants lui fournissant ses données d'entrée.

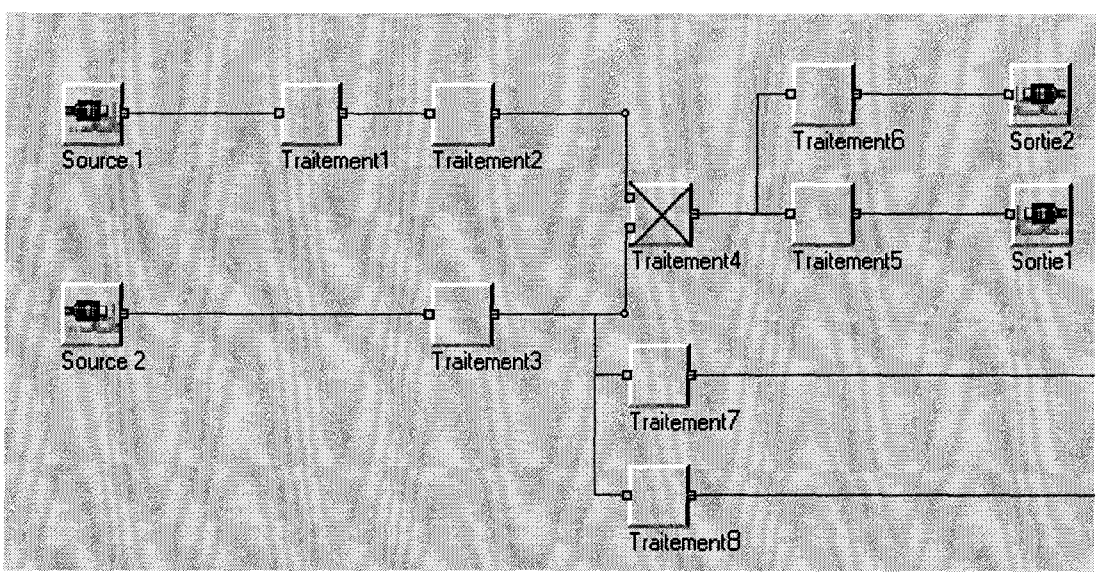


Figure 3.24 : suppression d'un composants.

Dans l'exemple ci-dessus, la suppression du composant « Traitement4 » provoque donc l'envoi d'un événement de type ***« Mise à zéro »*** aux composants « Traitement5 » et « Traitement6 » qui devront attendre la connexion d'un nouveau composant en entrée pour être reconfigurés. Les composants « Sortie1 » et « Sortie2 » reçoivent également un message de type ***« Mise à zéro »*** de la part des composants « Traitement5 » et « Traitement6 » précédemment passés dans l'état « ***Inactif*** ».

Enfin, la suppression du composant « Traitement 4 » provoquant la destruction de ses ports d'entrée, les listes de ports (***PortList***) à transmettre des ports de sortie des composants « Traitement 2 » et « Traitement 3 » sont mises à jour.

8 - Conclusion :

Dans ce chapitre nous avons présenté les différentes classes de composants à la base de tous types d'applications de traitement et d'acquisition temps réel des signaux.

Ainsi, nous avons défini les grandes classes de composants (***TSource***, ***TVoie***, ***TTraitement*** et ***TSortie***) permettant l'acquisition, le traitement et la transmission des signaux conformément aux spécifications du chapitre 1.

Par ailleurs, nous avons également défini les principes de communication et de contrôle relatif à chacune des classes de composants. Ainsi, il est possible d'envisager une spécification de ces classes pour tenir compte des particularités du domaine du monitorage médical (signaux physiologiques).

Chapitre 4

Architecture matérielle du réseau d'acquisition

1 - Introduction 80
2 - Présentation générale 81
3 – La tête de réseau 82
4 - Les modules d'acquisition 84
4.1 - Traitement numérique 84
4.2 - Conditionnement 84
4.3 - Les modules développés à ce jour 85
5 - Communication entre les éléments du réseau 87
5.1 - Flot des données d'acquisition 87
5.2 - Principe de transfert des données des modules vers le micro ordinateur 88
5.3 - Emission de données du PC vers les modules 89
6 – Conclusion 90

Ce chapitre présente l'architecture matérielle et le protocole de communication du réseau d'acquisition développé à l'ITM. Ce dispositif constitue la source de données temps réel alimentant l'application logicielle de traitement du signal présentée au chapitre 5.

1 - Introduction :

La recherche biomédicale, qu'elle soit expérimentale ou clinique, nécessite la mesure et l'analyse de différents signaux physiologiques comme l'électrocardiogramme (ECG), la pression artérielle, les paramètres respiratoires, etc....

En général, ces paramètres sont fournis par des moniteurs médicaux équipés de capteurs spécifiques. Cependant, ces appareils effectuent des traitements locaux sur les signaux sources afin de fournir des paramètres immédiatement interprétables en routine clinique (fréquence cardiaque moyenne, pression artérielle moyenne ou fréquence respiratoire). Il en résulte que de tels appareils ne permettent pas d'accéder aux signaux primaires essentiels pour la recherche biomédicale.

De plus, le grand nombre d'informations à collecter impose des contraintes de centralisation et de synchronisation des données.

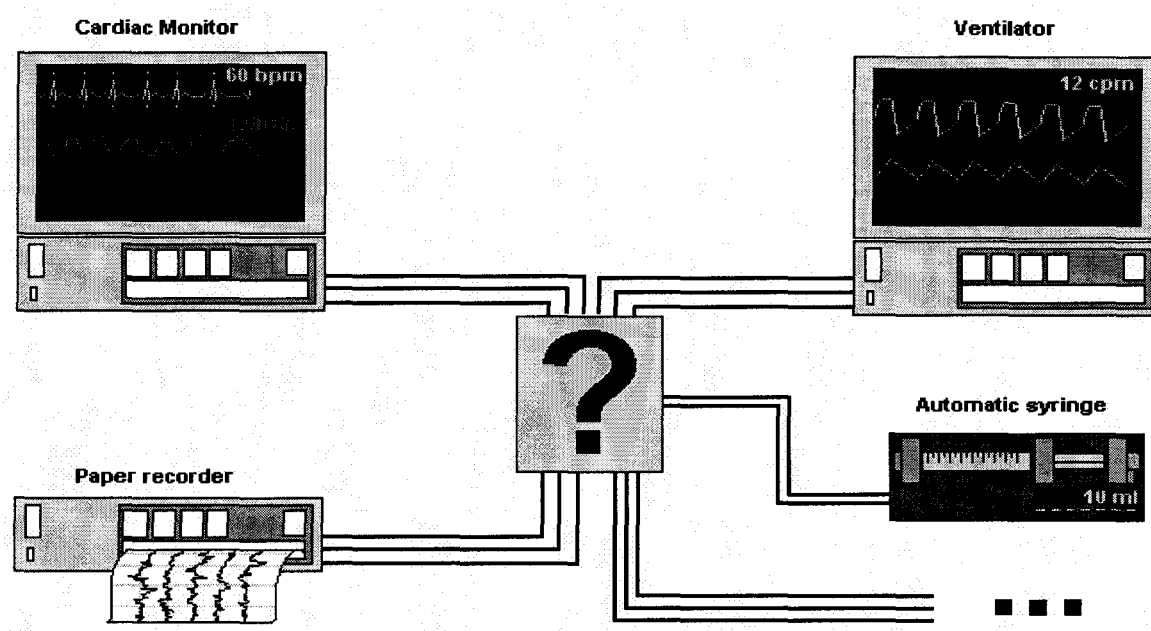


Figure 4.1 : centralisation des données biomédicales

2 - Présentation générale

La station d'acquisition et de traitement des signaux biomédicaux de l'ITM ([LOGI 03]) a été conçue comme une plateforme multimodale traitant tout type d'information biomédicale qu'elle soit analogique ou numérique, représentant un signal ou un paramètre.

La technologie développée vise la mise en oeuvre d'un réseau de terrain pour lequel les signaux sont acquis au moyen de modules dont la structure interne est celle d'un ***Capteur Intelligent*** intégrant les fonctions de conditionnement (mise à niveau, amplification), de validation (contrôle, filtrage), de numérisation et de communication. Les différents modules sont interconnectés indifféremment les uns aux autres (concept *plug and play*) par un réseau numérique lui-même connecté, par l'intermédiaire d'un module « tête de réseau », à un micro-ordinateur utilisé comme console de supervision (Figure 4.2).

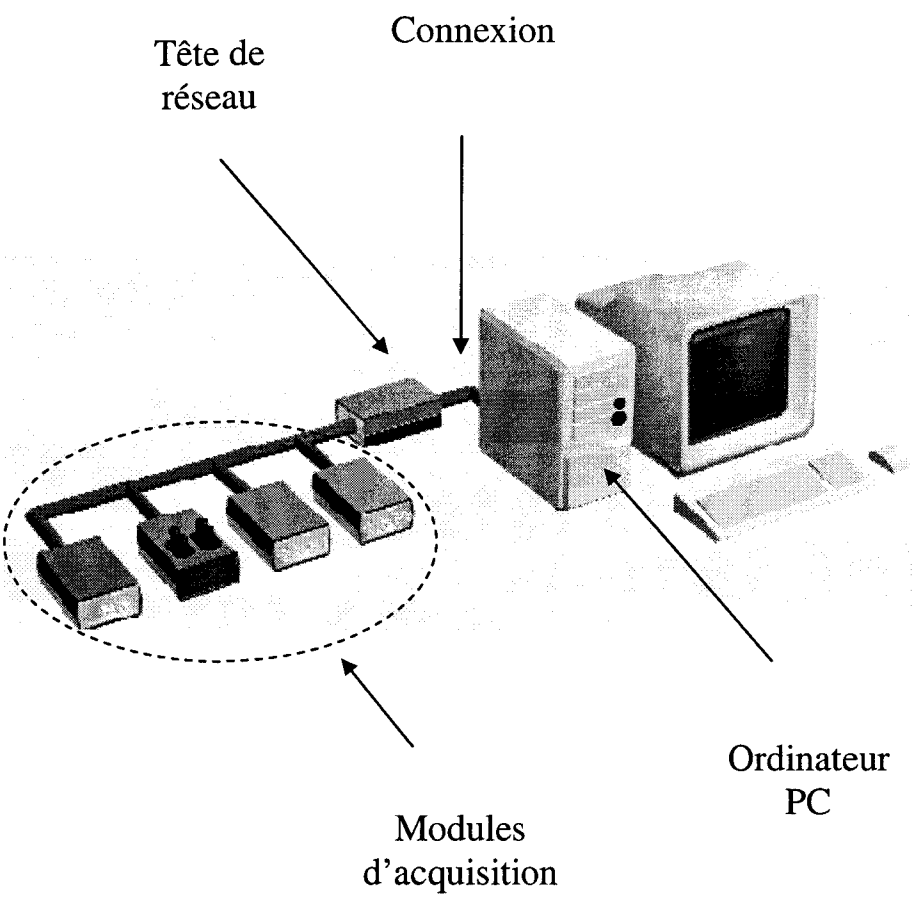


Figure 4.2 : présentation générale de la station

La tête de réseau sert d'interface avec l'ordinateur. Elle possède une horloge temps réel qui assure un échantillonnage synchrone de tous les signaux connectés aux différents modules. Elle permet de rediriger les ordres émanant de l'ordinateur vers les modules concernés. Elle transmet également l'alimentation électrique jusqu'aux modules par le câble de liaison réseau.

D'un point de vue fonctionnel, l'ensemble se compose d'une structure matérielle (modules et tête de réseau) et d'une structure logicielle implantée au niveau de l'ordinateur. Cette structure logicielle intègre un driver dont le rôle est d'effectuer le lien avec le matériel, et une application, interface utilisateur, qui permet de visualiser, de traiter et d'enregistrer les signaux acquis.

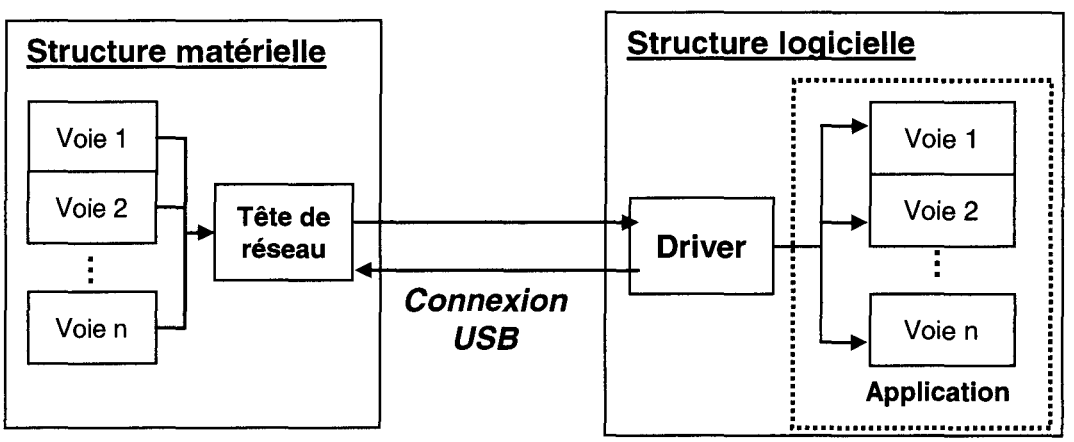


Figure 4.3 : schéma fonctionnel de la station

Un protocole de communication « propriétaire » a été développé pour le dialogue entre les différents éléments de la station. Ce protocole est optimisé pour la transmission en temps réel de signaux échantillonnés. Son implémentation logicielle est partagée entre le driver et les microcontrôleurs intégrés dans la tête de réseau et les différents modules.

3 – La tête de réseau

La tête de réseau occupe un rôle déterminant dans la communication entre les différents modules connectés. Elle est bâtie autour d'un micro contrôleur qui pilote l'ensemble de

l'interface. Ce Micro Contrôleur Tête de Réseau (***MCTR***) est chargé de trois fonctions essentielles dans la gestion du réseau :

- configurer l'architecture matérielle en terme de modules et d'entrées/sorties,
- cadencer l'échantillonnage en mode temps réel,
- commander les modules en mode temps réel.

La figure 4.4 montre le schéma synoptique de la tête de réseau. Toute commande ou information provenant de l'ordinateur est lue sur l'entrée Rx du MCTR qui en décode et traite le contenu. Les commandes ou informations destinées aux modules sont émises par la sortie Tx du MCTR qui est maître du réseau. Une horloge temps réel permet la synchronisation de l'échantillonnage des signaux. A chaque top d'horloge, le MCTR émet une commande d'échantillonnage sur le réseau. Cette commande est interprétée simultanément par tous les modules connectés qui sont en permanence à l'écoute du réseau. Chaque module exécute alors son échantillonnage. Les modules transmettent tour à tour leur donnée échantillonnée dans un ordre qui leur est dicté par le MCTR. Afin d'optimiser le flot de données, le transfert des données échantillonnées vers l'ordinateur s'effectue directement sans intervention du ***MCTR*** par la connexion Rx de l'interface de communication. Ceci est rendu possible grâce à la commande bidirectionnelle du circuit d'interface réseau. Ce dernier aspect qui a fait l'objet d'un brevet ([LOGI 03]) permet notamment d'obtenir des fréquences d'échantillonnage beaucoup plus élevées que sur les systèmes conventionnels.

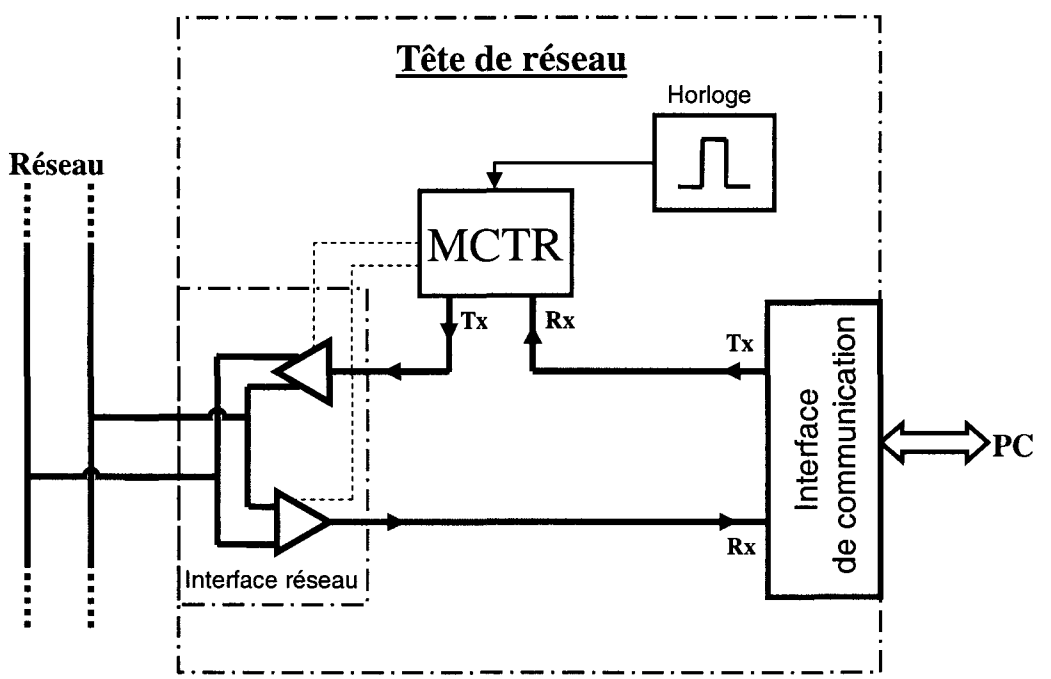


Figure 4.4 : schéma synoptique de la tête de réseau.

4 - Les modules d'acquisition

Tous les modules d'acquisition ont la même architecture matérielle constituée de deux parties distinctes : « conditionnement » et « traitement numérique » (figure 4.5).

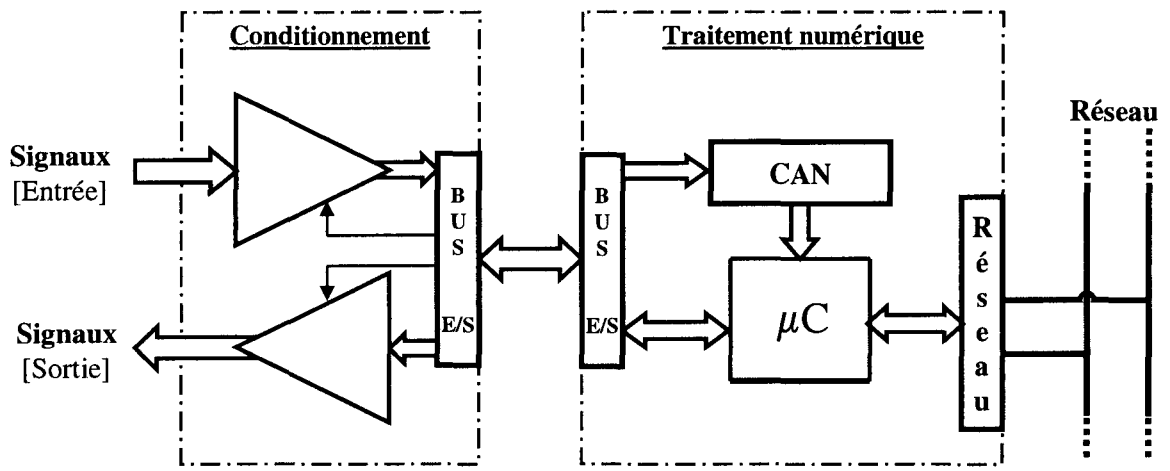


Figure 4.5 : structure générale d'un module

4.1 - Traitement numérique :

La partie numérique est bâtie autour d'un microcontrôleur. Elle est identique pour tous les modules. Seul le programme implanté au niveau du microcontrôleur est spécifique de l'utilisation du module. Cette partie intègre également un convertisseur analogique – numérique et un port de communication série qui permet le dialogue sur le réseau. Les échanges avec la partie conditionnement se font par un bus parallèle.

4.2 - Conditionnement :

Bien que l'utilisation principale de la station concerne l'acquisition de signaux analogiques, la structure matérielle des modules est adaptée à de multiples besoins. Le bus parallèle communiquant avec la partie numérique comporte trois types d'entrées / sorties : analogique, logique et numérique sérielle. Ainsi il est possible de concevoir tout type de conditionnement pour des utilisations variées telles que :

- Acquisition de signaux analogiques, logiques, numériques.
- Restitution de signaux analogiques, logiques, numériques.

De plus, chaque module peut intégrer, le cas échéant, un transducteur en entrée (par exemple un capteur de pression) ou un actionneur en sortie (par exemple une électrovanne).

4.3 - Les modules développés à ce jour :

Les différents types de modules réalisés à ce jour sont :

1. **ECG** : acquisition du signal électrocardiographique à partir d'électrodes cutanées standards.
2. **Pouls / SPO2** : acquisition du signal de pouls et calcul de la saturation partielle en oxygène. Ce module utilise un capteur rouge/infrarouge annulaire standard.
3. **Pressions sanglantes** : acquisition de deux signaux de pression sanglante utilisant des capteurs standards.
4. **Fonctions respiratoires** : ce module est muni de deux capteurs de pression différentielle. Il a été conçu spécialement pour la mesure de la pression et du débit des voies respiratoires. Il utilise un pneumotachographe standard.
5. **Température** : acquisition de la température corporelle à partir d'un capteur standard.
6. **4 voies analogiques (4VA)**: acquisition simultanée de quatre signaux analogiques provenant de dispositifs annexes.
7. **Numérique RS232** : échange de données numériques avec des équipements biomédicaux dotés d'une connexion de type RS232.
8. **Commandes numériques** : sorties logiques programmables pour la commande d'organes actionneurs.

A titre d'exemple, la figure 4.6 donne le schéma du module 4 voies analogiques. La figure 4.7 donne, quant à elle, un exemple d'assemblage de modules.

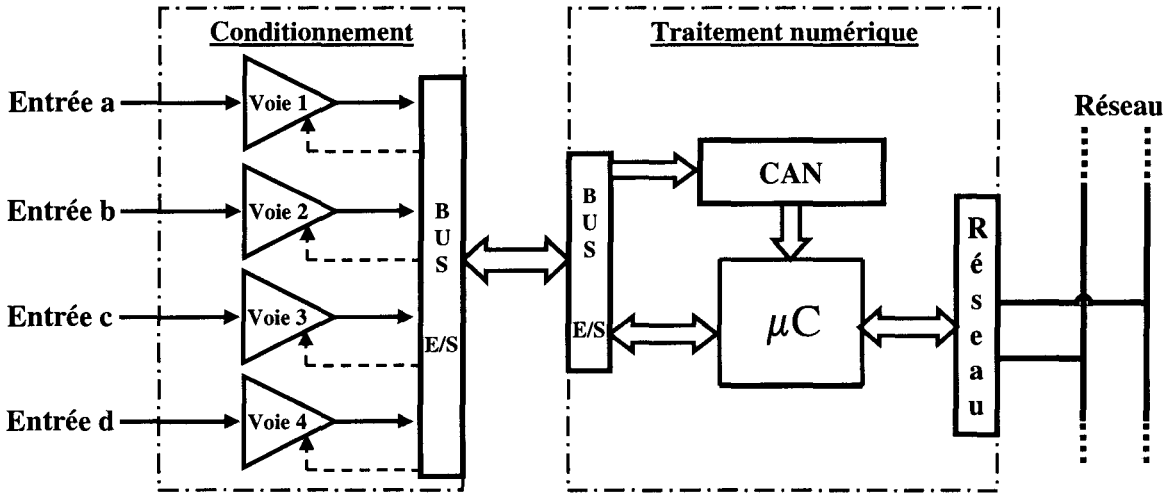


Figure 4.6 : exemple : le module 4VA.

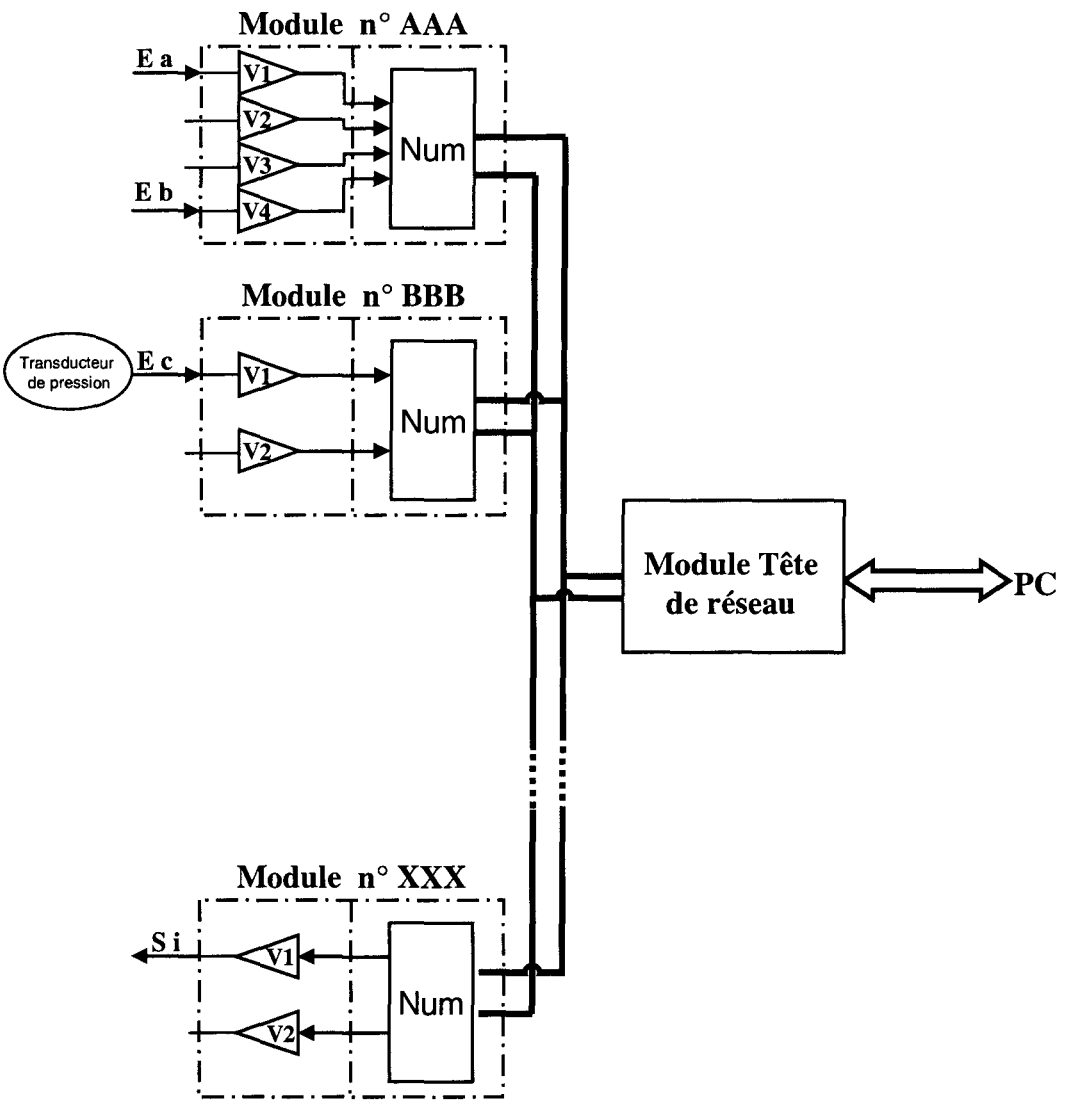


Figure 4.7 : exemple d'assemblage de modules en réseau

5 - Communication entre les éléments du réseau

5.1 - Flot des données d'acquisition :

Comme indiqué sur la figure 4.4, le transfert des données du réseau vers l'ordinateur s'effectue directement sans intervention du MCTR. Le MCTR n'a donc pas la possibilité de contrôler les données en provenance des modules connectés au réseau. Toutefois, si l'échantillonnage des signaux est synchronisé par la commande du MCTR, l'acheminement des données échantillonnées vers l'ordinateur se fait, quant à lui, de façon séquentielle. Il est donc important de respecter une séquence de transmission qui soit connue de tous les modules. Ainsi, chaque module est capable de contrôler son flot de données vers le PC en réponse à l'ordre d'échantillonnage du MCTR.

Le flot de données d'acquisition se présente de la façon suivante :

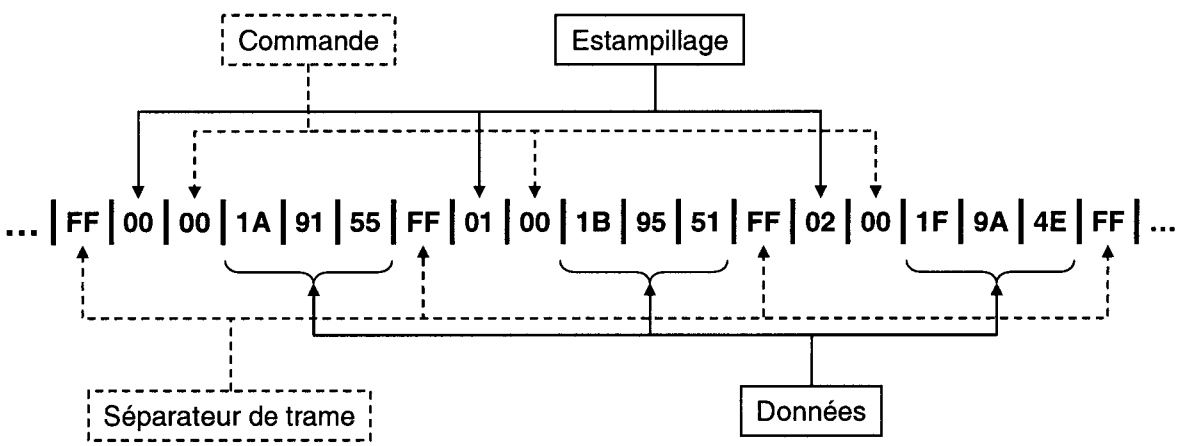


Figure 4.8 : forme du flux de données d'acquisition

Les trames d'octets sont séparées par le mot réservé FF. Chacune de ces trames comprend donc :

Octet 1 : séparateur FF
Octet 2 : estampillage
Octet 3 : commande
Octets 4 à n : données

L'estampillage est un octet, généré par le MCTR, variant de 00 à FE qui est incrémenté à chaque commande d'échantillonnage. Cette technique de datation permet, lors du traitement, de contrôler la continuité des échantillonnages, afin de détecter d'éventuels problèmes de communication entre le réseau et l'ordinateur.

La commande est un octet généré par MCTR destiné à l'activation de fonctions particulières au niveau des modules. La commande la plus commune (00) est la commande d'échantillonnage, mais le protocole prévoit d'autres possibilités comme par exemple le changement du niveau d'amplification d'une entrée.

5.2 - Principe de transfert des données des modules vers le micro ordinateur :

Le transfert des données d'acquisition du réseau vers l'ordinateur est ordonné. Cela implique pour l'ordinateur de connaître à la fois la séquence d'émission des voies d'acquisition et le nombre d'octets que chaque voie transmet. La séquence des émissions peut être imposée par l'ordinateur lors de la configuration du réseau. Par ailleurs, le nombre d'octets transférés par une voie est directement lié à la résolution choisie pour la voie (8 ou 12 bits) et / ou au type de modules dont la voie est issue.

L'ordre dans lequel les données sont émises doit être implicitement connu de l'ensemble des modules d'acquisition. Cela signifie que tout module doit connaître, pour chacune de ses voies utilisées, le moment auquel il doit émettre les données d'acquisition. Pour cela, on utilise le fait que les octets qui circulent sur le réseau sont vus par tous les modules. Ainsi, pour qu'un module connaisse exactement l'état d'avancement du transfert des données vers l'ordinateur, il lui suffit de compter les octets qu'il voit passer. Ainsi, pour envoyer ses données au bon moment, il est nécessaire qu'il prenne connaissance, à l'initialisation du réseau, du nombre d'octets de transfert à laisser passer avant d'émettre.

Par contre, pour effectuer des opérations plus complexes (par exemple reconfigurer une voie en cours d'acquisition), il est nécessaire d'informer les autres modules du nombre d'octets

qu'elles émettent. Les modules peuvent alors mettre à jour les nombres d'octets à laisser passer avant d'émettre pour chacune des voies.

En conclusion, toute commande de configuration d'une voie sur le réseau nécessite deux arguments :

- le nombre d'octets à attendre avant d'émettre,
- le nombre d'octets à transmettre à l'ordinateur.

Seule la commande de sélection d'une voie non connectée nécessite des arguments supplémentaires :

- le type de module
- son adresse physique (numéro de série)
- le numéro de la voie d'acquisition.

5.3 - Emission de données du PC vers les modules :

Les commandes envoyées par l'ordinateur vers les modules (réglage de gain, d'offset, de fréquence d'échantillonnage...) passent obligatoirement par la tête de réseau qui les renvoie vers les modules en synchronisation avec les tops d'horloge.

Deux solutions se présentent alors pour transférer les commandes en provenance de l'ordinateur :

- Elles sont émises à la place de la commande d'échantillonnage ; auquel cas certaines acquisitions sont perdues.
- Elles sont associées à la commande d'échantillonnage ; auquel cas le nombre d'acquisitions possibles à chaque échantillonnage est réduit du nombre d'octets nécessaires au codage de la commande.

Ces deux solutions coexistent. La deuxième est utilisée avec les commandes dont le nombre d'octets de codage est assez réduit pour que l'ensemble des valeurs d'acquisition puisse être

transmis ; c'est le mode de transfert « **commandes réduites** ». Dans l'autre cas de figure, on emploie le mode de transfert « **commandes évoluées** » correspondant à la première solution. Le seuil pour passer d'un mode à l'autre sera fixé par défaut par la tête de réseau mais pourra également être définie par le PC. Le protocole de communication du réseau d'acquisition est présenté en annexe 1.

6 – Conclusion

Dans ce chapitre, nous avons présenté l'architecture matérielle d'une station permettant l'acquisition et le traitement des signaux physiologiques.

Ce chapitre fournit en particulier une description technique de ce matériel en présentant d'une part le principe de communication entre les différents éléments du réseau et d'autre part, le protocole de communication permettant la commande et le décryptage des données issues de cette source temps réel.

L'intégration d'un driver logiciel intégrant les différentes fonctions et commandes relatives au protocole de communication de ce réseau conformément aux spécifications décrites au chapitre 3 permet de réaliser d'une manière simple l'interface entre le réseau de terrain et l'application logicielle de traitement temps réel des signaux physiologiques présentée au chapitre suivant.

Chapitre 5

Développement d'une bibliothèque de composants logiciels pour le monitorage médical

1 – Introduction 92
2 - Les éléments de communication avec le réseau 92
2.1 – Création 93
2.2 - Initialisation 93
2.3 – Commandes 96
2.4 – Lecture / Extraction / Distribution des données 97
2.5 – Destruction 97
3 – Messages et ports de communication 97
3.1 – Les types de messages 97
3.1.1 – TAcqData 97
3.1.2 – TacqDataBuf 98
3.1.3 – TAcqDataMat 98
3.1.4 – TacqDataBool 99
3.2 – Les ports de communication 99
4 - Les éléments de traitement de signal 100
4.3 – Les composants de base 101
4.4 – Les composants de détection 102
4.5 – Les composants de mesures cycliques 102
4.6 – Les composants de mesures sur fenêtre glissante 103
4.7 – Les composants d'analyse avancé 104
4.8 – Les composants spécialisés 104
5 – Interface de programmation graphique 105
5.1 – Programmation du traitement 105
5.2 – Visualisation des résultats 106
6 - Conclusion 107

1 – Introduction

Comme nous l'avons vu dans le chapitre précédent, d'un point de vue fonctionnel, la station d'acquisition et de traitement des signaux physiologiques se compose d'une structure matérielle (modules et tête de réseau) et d'une structure logicielle implantée au niveau de l'ordinateur. Cette structure logicielle intègre une partie « pilote » (ou driver) dont le rôle est d'effectuer le lien avec le matériel, et une partie « application », interface utilisateur, qui permet de visualiser, de traiter et d'enregistrer les signaux physiologiques (figure 5.1).

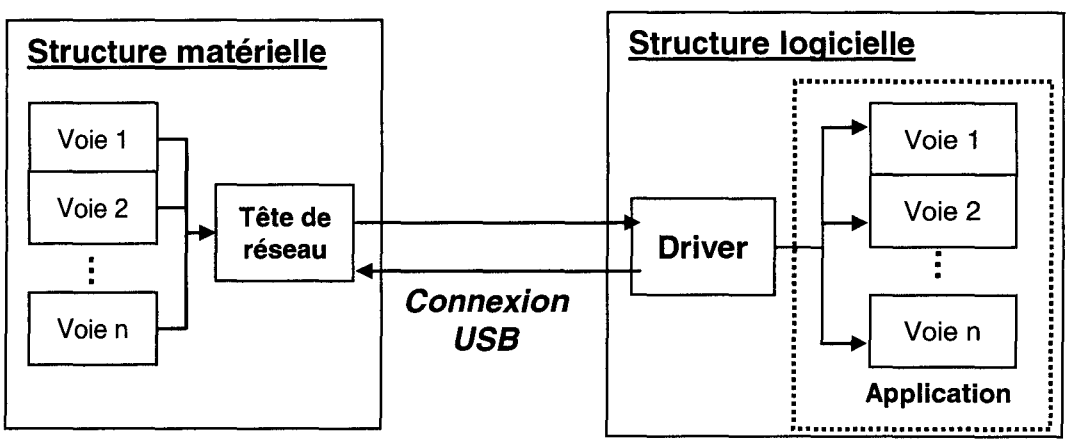


Figure 5.1 : schéma fonctionnel de la station

Ainsi, conformément aux spécifications du chapitre 3, la structure logicielle peut être implémentée sous la forme d'une bibliothèque de composants descendant des classes ***TSource***, ***TVoie***, ***TTraitement*** et ***TSortie***.

2 - Les éléments de communication avec le réseau :

La réalisation du driver implique la création d'une classe descendante de la classe ***TSource***.

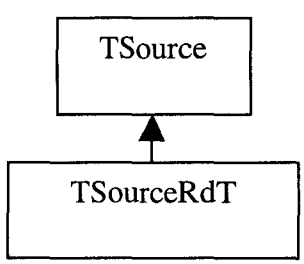


Figure 5.2 : la classe TSourceRdT

Conformément au concept d'héritage en méthodologie orientée objet, cette classe hérite des attributs et méthodes de sa classe parente (attribut ***VoieList*** et méthodes ***AddVoie*** et ***RemoveVoie***).

Elle permet de configurer le réseau de terrain, de récupérer les données issues de ce réseau via le port USB et de redistribuer ces différentes données aux composants de type ***TVoie***. Cette classe intègre toutes les méthodes et attributs nécessaires au fonctionnement du réseau de terrain présenté précédemment.

Ainsi, les différentes actions pouvant intervenir tout au long du cycle de vie d'un composant issu de cette classe sont :

- la création,
- l'initialisation,
- la commande,
- la lecture / extraction / distribution des données,
- la destruction.

2.1 – Création :

Par héritage, la classe ***TSourceRdT*** intègre un constructeur (méthode ***Create***) permettant l'instanciation d'un composant issu de cette classe. A sa création, ce composant n'est pas actif. En effet, il ne connaît pas les paramètres nécessaires à son fonctionnement tesl que la fréquence d'échantillonnage ou la liste des voies d'acquisitions auxquelles il doit transmettre ses données. Par conséquent, l'implémentation d'une méthode d'initialisation est nécessaire.

2.2 - Initialisation :

Par héritage, la classe ***TSourceRdT*** intègre une procédure de chargement (méthode ***Loaded***). Cette méthode intègre généralement toutes les fonctions d'initialisation du composant, selon 3 étapes :

- la connexion,
- le choix de la fréquence d'échantillonnage,
- la sélection des voies matérielles.

La première étape de l'initialisation est la connexion du composant logiciel au composant électronique USB de la tête de réseau (méthode ***Connect***). Une méthode ***TestConnect*** est également implémentée pour tester la validité de la connexion à tout moment (figure 5.3).

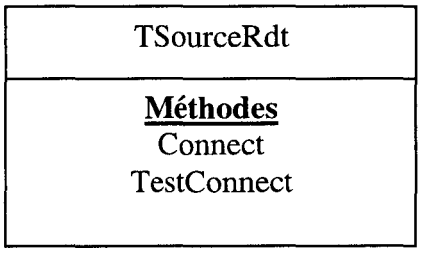


Figure 5.3 : définition des méthodes de connexion

Le chargement d'un composant d'acquisition temps réel passe par la création d'un processus temps réel de lecture/écriture (appelé ***Thread***) qui permet l'envoi de commandes et la réception de données issues d'un périphérique.

La deuxième étape de l'initialisation est le choix de la fréquence d'échantillonnage des signaux. Cela nécessite l'implémentation d'un attribut ***FreqEch*** et d'une méthode de sélection de la fréquence d'échantillonnage ***SelectFreq***. Afin de permettre le changement de fréquence pendant l'exécution du composant, cette méthode peut être appelée à tout moment du cycle de vie du composant (figure 5.4).

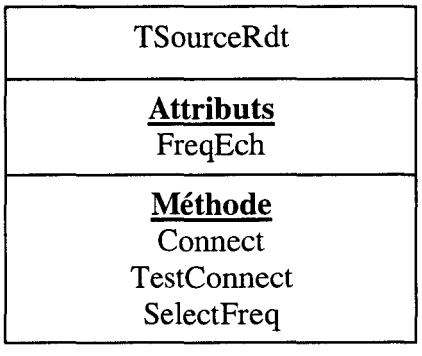


Figure 5.4 : définition des méthodes et attributs de réglages de fréquence

Pour notre application les différentes fréquenced d'échantillonnage retenues sont : 25, 50, 100, 250, 500, 1000, 2500, 5000 ou 10000 Hz.

Enfin, la troisième étape consiste à la sélection des voies d'acquisition. Cela est réalisé par le biais de la méthode ***AddVoie*** du composant ***TSourceRdT*** appelée lors du chargement des composants ***TVoie***. Il est donc indispensable que les composants ***TVoie*** intègrent l'ensemble des attributs nécessaires à la sélection d'une voie d'acquisition. Ainsi, conformément aux spécifications définies au chapitre précédent, cette classe intègre les attributs suivants (figure 5 .5) :

- le numéro de série du module d'acquisition (SNum),
- le type de module (BoxType),
- le numéro de la voie d'acquisition (NumVoie)

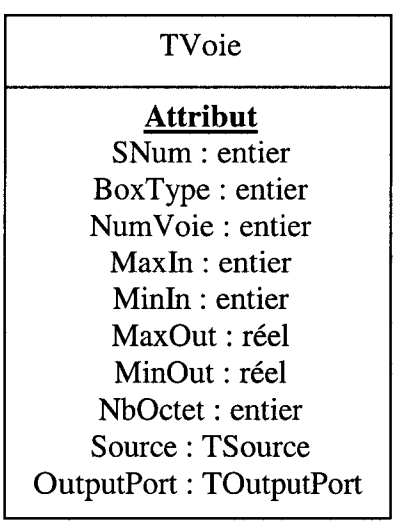


Figure 5.5 : Définition des attributs de sélection d'une voie d'acquisition

Lors du chargement d'un composant ***TVoie***, ***AddVoie*** du composant ***TSourceRdT*** est appelée. Cette méthode calcule, pour la nouvelle voie sélectionnée, le nombre d'octets à recevoir avant d'émettre. Elle envoit ensuite tous les paramètres au réseau de modules pour l'activation de la voie matérielle et la mise à jour de sa séquence d'acquisition. Cette particularité impose que le composant ***TSourceRdT*** soit créé et initialisé avant le chargement des composants ***TVoie***.

2.3 – Commandes :

La classe ***TSourceRdT*** permet également d'envoyer différentes commandes au réseau de modules. Ainsi, au cours de l'acquisition, l'utilisateur a la possibilité de modifier différents paramètres tels que la fréquence d'échantillonnage (cf. paragraphe précédent), le gain, l'offset, etc... ou d'envoyer des commandes à différents actionneurs. Pour cela, on considère les éléments à configurer (potentiomètre de gain, d'offset, électrovanne...) comme étant des récepteurs avec lesquels le driver communique. L'envoi d'une commande du driver vers un récepteur passe donc par différentes étapes :

- l'initialisation de la communication,
- l'envoi des octets de commande,
- la mémorisation des données,
- la destruction de la communication.

Ainsi, la classe ***TSourceRdt*** intègre également les méthodes suivantes (figure 5.6) :

- **InitRecepteur** : Initialisation de la communication avec un récepteur,
- **EcriRecepteur** : Envoi des données au récepteur,
- **MemoRecepteur** : Mémorisation de l'état du récepteur,
- **DetRecepteur** : Destruction de la communication avec le récepteur.

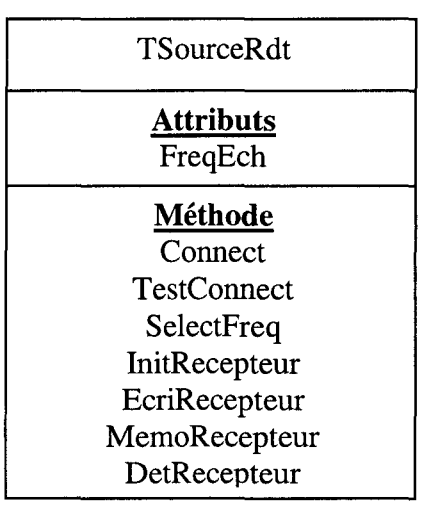


Figure 5.6 : définition des méthodes de commandes

2.4 – Lecture / Extraction / Distribution des données :

La lecture de la source de données est effectuée au sein du thread d'acquisition créé au chargement du composant ***TSourceRdT***. Dès leur arrivée, ces données sont traitées par le composant qui localise les données utiles afin de les distribuer aux différents composants ***TVoie***. Cela est rendu possible par la connaissance préalable de la séquence d'envoi (attribut ***TListVoie***). Enfin, le composant effectue un contrôle sur l'estampillage (datation) afin de localiser une éventuelle perte d'échantillon.

2.5 – Destruction :

Par héritage, les composants ***TSourceRdT*** et ***TVoie*** possèdent un destructeur (méthode **Destroy**) permettant de détruire le composant. La structure des composants décrite précédemment implique que les composants ***TVoie*** soient détruis avant les composants ***TSourceRdt***. Ainsi, à chaque destruction de ***TVoie***, la liste des vois (***VoieList***) du composant ***TSourceRdT*** est mise à jour par appel de la méthode ***RemoveVoie***. Enfin, à la destruction de ***TSourceRdT***, ou le ***thread*** d'acquisition est désactivé et la connexion avec la source de données coupée.

3 – Messages et ports de communication :

Tout composant de la bibliothèque de traitement descend directement de la classe ***TTraitement*** présentée au chapitre 3.

3.1 – Les types de messages :

Comme nous l'avons exprimé au chapitre 2, tout port d'entrée peut être connecté à un port de sortie si est seulement si le type de message est connu des deux ports. Pour notre application, nous avons donc défini différentes classes de messages.

3.1.1 – TAcqData :

Ce message est le message de base utilisé par la plupart des composants de la bibliothèque. Il ne transporte qu'une donnée utile de type réel (figure 5.7). L'entête de ce message intègre le nom de la source (qui fait référence au port de sortie d'un autre composant), l'estampillage

(pour vérifier qu'aucune donnée n'a été perdue) et la référence temporelle qui représente en fait le temps écoulé depuis l'envoi du dernier message.

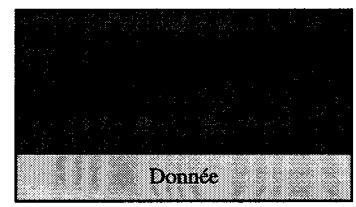


Figure 5.7 : message TAcqData

3.1.2 – TacqDataBuf :

Ce message transporte un buffer de données de type réel. Son entête intègre en plus la taille du buffer (figure 5.8).

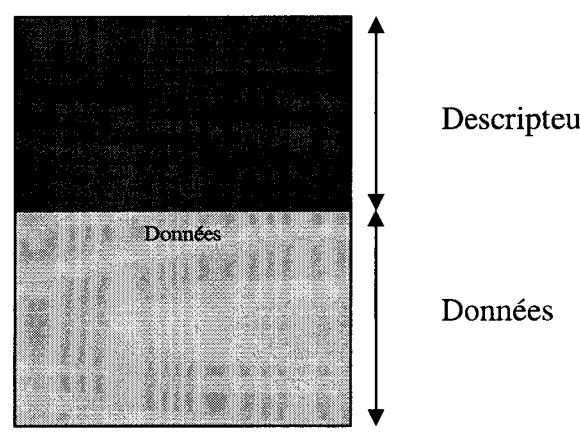


Figure 5.8 : message TAcqDataBuf

3.1.3 – TAcqDataMat :

Ce message transporte une matrice de données de type réel. Son entête intègre en plus les dimensions de la matrice transmise (figure 5.9).

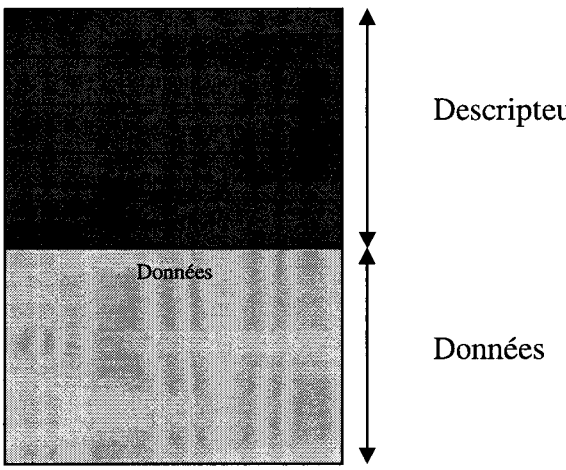


Figure 5.9 : message TAcqDataMat

3.1.4 – TacqDataBool :

Ce message transporte une donnée unique de type booléen (figure 5.10). Son entête est identique à celle du message TAcqData.

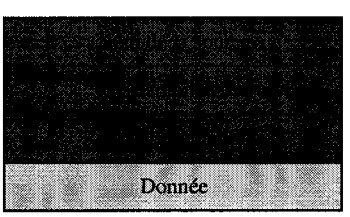


Figure 5.10 : message TAcqDataBool

3.2 – Les ports de communication :

Une classe de ports de communication d'entrée et de sortie est définie pour chaque type de message :

- **TInputPortReal** et **TOutputPortreal** le transfert des messages de type **TAcqData**,
- **TInputPortBuf** et **TOutputPortBuf** le transfert des messages de type **TAcqDataBuf**,
- **TInputPortMat** et **TOutputPortMat** le transfert des messages de type **TAcqDataMat**,
- **TInputPortBool** et **TOutputPortBool** le transfert des messages de type **TAcqDataBool**.

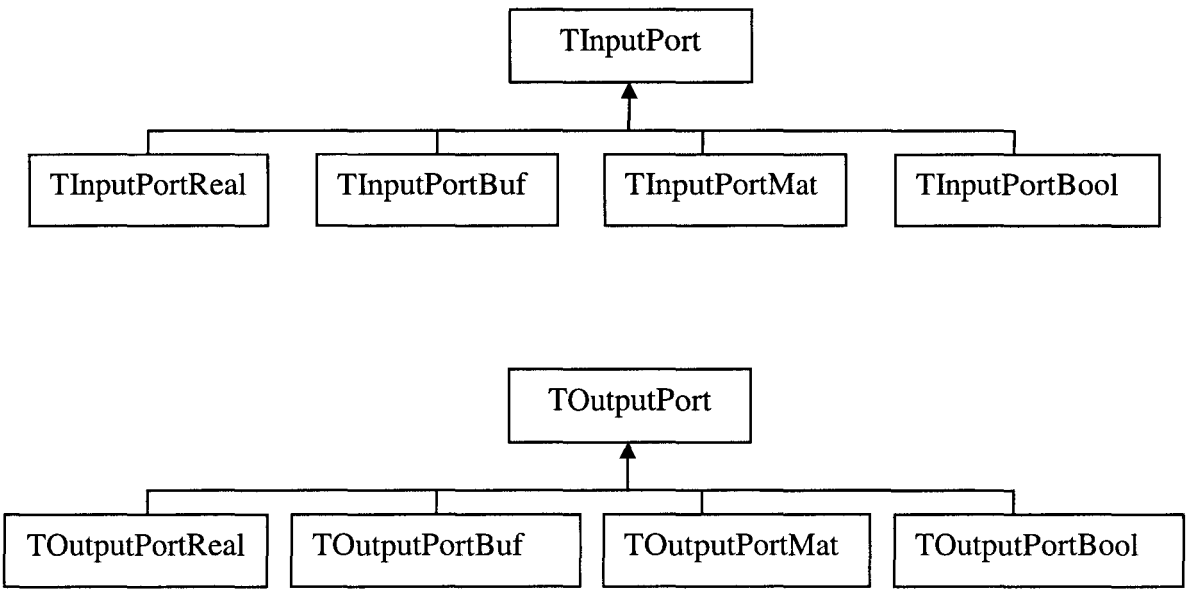


Figure 5.11 : définition des différentes classes de ports d'entrées/sorties

Ainsi, chaque composant ***TTraitement*** de la bibliothèque possède un ou plusieurs de ces ports d'entrée et/ou de sortie.

4 - Les éléments de traitement de signal :

Les composants de traitement sont tous descendants de la classe ***TTraitement*** présentée au chapitre 3. Ils possèdent donc un ou plusieurs ports d'entrée et un ou plusieurs ports de sortie (figure 5.12).

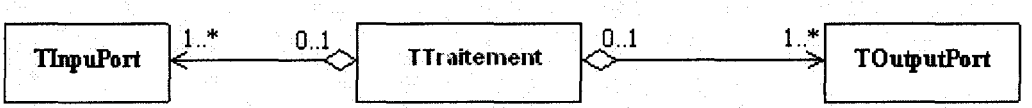


Figure 5.12 : diagramme de la classe TTraitement.

Prenons l'exemple d'un composant permettant d'effectuer un filtrage numérique passe bas sur une donnée d'entrée. Les attributs nécessaires à la configuration d'un tel composant sont :

- le Type de filtre (Bessel,Butterworth, Tchebychev1 ou Tchebychev3) (TypeF),
- le Type de filtrage (Passe haut ou passe bas) (FType),
- la fréquence de coupure (FC),
- l'ordre du Filtre (Ordre).

Afin d'acquérir et de transmettre ses données, ce composant doit également intégrer un port d'entrée et un port de sortie permettant l'acquisition et la transmission de messages de type « donnée réelle » (respectivement ***TInputPortReal*** et ***TOutputPortReal***).

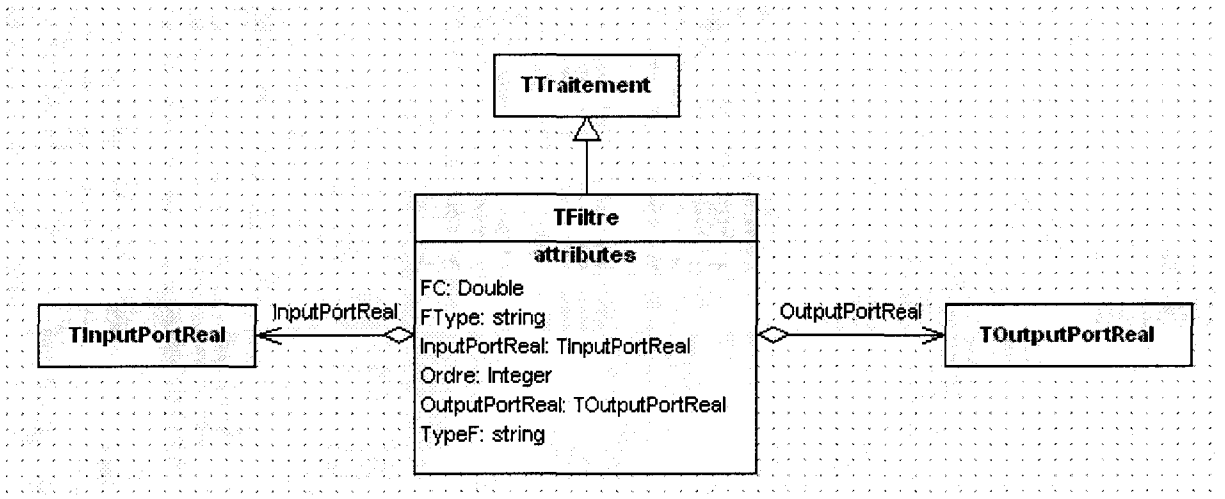


Figure 5.13 : définition du composant TFilter

La généralisation de la classe ***TTraitement*** permet l'obtention d'un grand nombre de composants permettant d'effectuer tout type d'algorithme de traitement numérique du signal.

On obtient ainsi un grand nombre de composants organisés en différents paquets :

- les composants de base,
- les composants de détection,
- les composants de mesures cycliques,
- les composants de mesure sur fenêtre glissante,
- les composants de traitements avancés,
- les composants spécialisés.

L'ensemble de ces composants est décrit en annexe 3.

4.3 – Les composants de base :

Ce paquet de composants représente en fait tous les calculs basiques pouvant intervenir dans une application de traitement du signal. Cette bibliothèque intègre donc des composants de

filtrage, des composants de calculs arithmétiques, trigonométrique ou logiques, des ré-échantillonneurs...

4.4 – Les composants de détection :

Ces composants permettent, à partir d'un signal réel, de détecter l'occurrence d'un événement comme le dépassement d'un seuil ou, dans l'exemple de l'ECG, une onde R. Ces composants produisent donc à leur sortie, un signal binaire (ou booléen) qui prend la valeur « 1 » à l'occurrence de l'événement sinon la valeur « 0 ».

On propose alors deux modes de sortie :

- **impulsion** : le signal binaire prend la valeur « 1 » uniquement lorsque la condition de détection passe de l'état « faux » à l'état « vrai ».
- **échelon** : le signal binaire reste à « 1 » tant que le condition est « vraie ».

De tels composants intègrent un intervalle de tolérance sur la condition de détection ainsi qu'une période réfractaire pendant laquelle le mécanisme de détection est inhibé.

Les sorties des composants de détection peuvent ensuite être connectées à des composants de calculs logiques (***TCombinatoire***) pour l'élaboration de signaux binaires plus complexes pouvant être utilisés par exemple pour l'aide au diagnostique.

4.5 – Les composants de mesures cycliques :

Les composants de mesures cycliques permettent le calcul de paramètres (valeur max, valeur min, valeur moyenne...) entre deux impulsions d'un composant de détection. De tels composants possèdent donc généralement deux entrées :

- une entrée de type réelle correspondant au signal à analyser,
- une entrée de type booléen correspondant au composant de détection fournissant les impulsions.

Tout comme pour les composants de détection, on propose deux types de signaux de sortie.

- **impulsion** : la valeur calculée est envoyée de manière asynchrone dès la réception d'un message de détection.

- **échelon** : la valeur est envoyée continûment et on effectue un nouveau calcul et un changement de valeur dès la réception d'un message de détection.

Prenons l'exemple du calcul de l'intervalle RR à partir de l'ECG (Figuer 5.14).

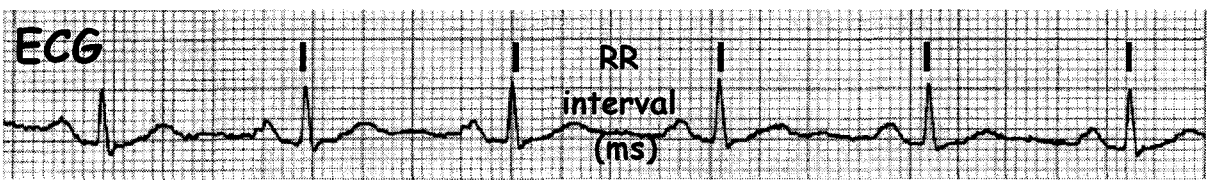


Figure 5.14 : calcul de l'intervalle RR à partir de l'ECG

Soit un composant de mesure cyclique relié à une source de donnée de type ECG sur l'une de ses entrées et à un composant permettant la détection des ondes R de l'ECG sur l'autre.

Dès son initialisation, le composant de mesure cyclique compte le nombre d'échantillons qu'il reçoit sur son entrée « ECG ». A chaque occurrence d'une onde R, le composant de détection fournit une impulsion au composant de mesure cyclique. Le composant de mesure cyclique calcule alors le temps écoulé depuis la dernière détection en multipliant le nombre d'échantillons comptés par la période d'échantillonnage. La valeur calculée est alors transmise par l'intermédiaire du port de sortie et le nombre d'échantillons comptés est remis à zéro. Le composant de mesure cyclique recommence alors à compter jusqu'à l'occurrence d'une nouvelle impulsion.

4.6 – Les composants de mesures sur fenêtre glissante :

Ces composants permettent de calculer plusieurs paramètres tel que le minimum, le maximum, la moyenne... sur une fenêtre glissante. Globalement, de tels composants permettent d'effectuer les mêmes calculs que les composants de mesures cycliques. Cependant, plutôt que d'effectuer ces calculs entre deux impulsions (et donc de manière asynchrone), l'information est ici traitée continûment par décalages successifs d'une fenêtre d'analyse temporelle.

La figure 5.15 illustre ce principe de mesure dans le cadre de la mesure du maximum d'amplitude.

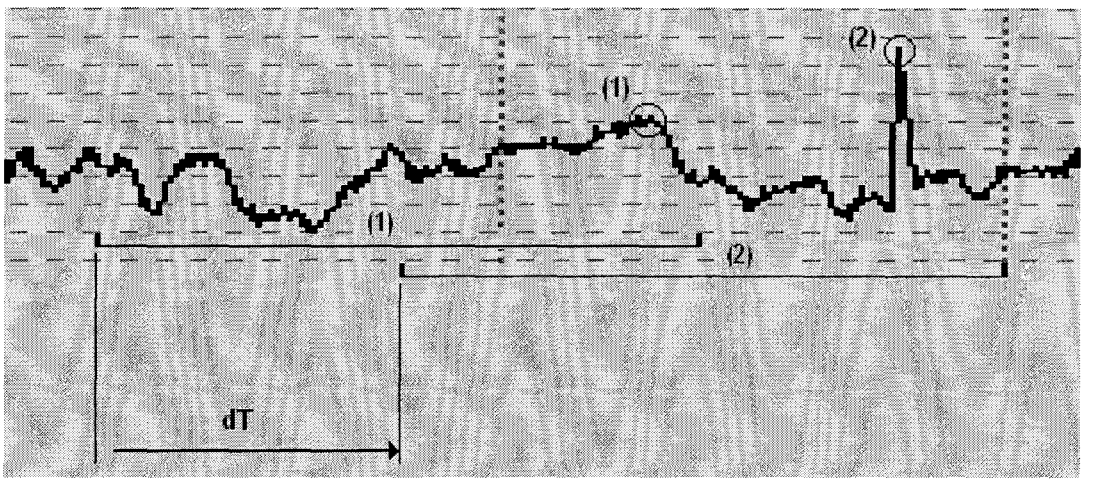


Figure 5.15 : Mesures sur fenêtre glissante

Le composant calcule la valeur du maximum d'amplitude dans la première fenêtre d'analyse (1). Cette fenêtre est ensuite décalée d'un intervalle de temps dT. Enfin, le même calcul est effectué dans la nouvelle fenêtre d'analyse (2) et ainsi de suite. Pour ce composant particulier, la valeur de la cadence d'échantillonnage de la sortie dépend donc directement de dT (f = 1/dT).

4.7 – Les composants d'analyse avancé :

Ce paquet regroupe les composants permettant l'application de techniques de traitement du signal avancées tel que le fenêtrage, la Transformée de Fourier Rapide (FFT) ou la transformée en ondelettes. La description de ces différentes méthodes de calcul est fournie en annexes 3, 4, 5 et 6.

4.8 – Les composants spécialisés :

Ce paquet regroupe les composants permettant l'analyse des signaux physiologiques. Ce paquet intègre, par exemple, les composants de détection des ondes R sur l'ECG, de filtrage de la série RR, de calcul des pressions systolique, diastolique et moyenne sur un signal de pression artérielle…

De tels composants sont créés à partir d'un assemblage particulier (par composition statique) des composants présentés précédement.

5 – Interface de programmation graphique :

5.1 – Programmation du traitement :

Pour permettre aux utilisateurs non spécialistes du développement informatique de créer leurs applications d'acquisition et de traitement des signaux physiologiques, il apparaît important de réaliser une interface de création d'application intégrant un langage de programmation visuel. Basé sur le principe de la programmation par flots de données, ce langage consiste alors à tracer le diagramme fonctionnel de l'algorithme de traitement du signal. La figure 5.27 donne un exemple de ce type de programme.

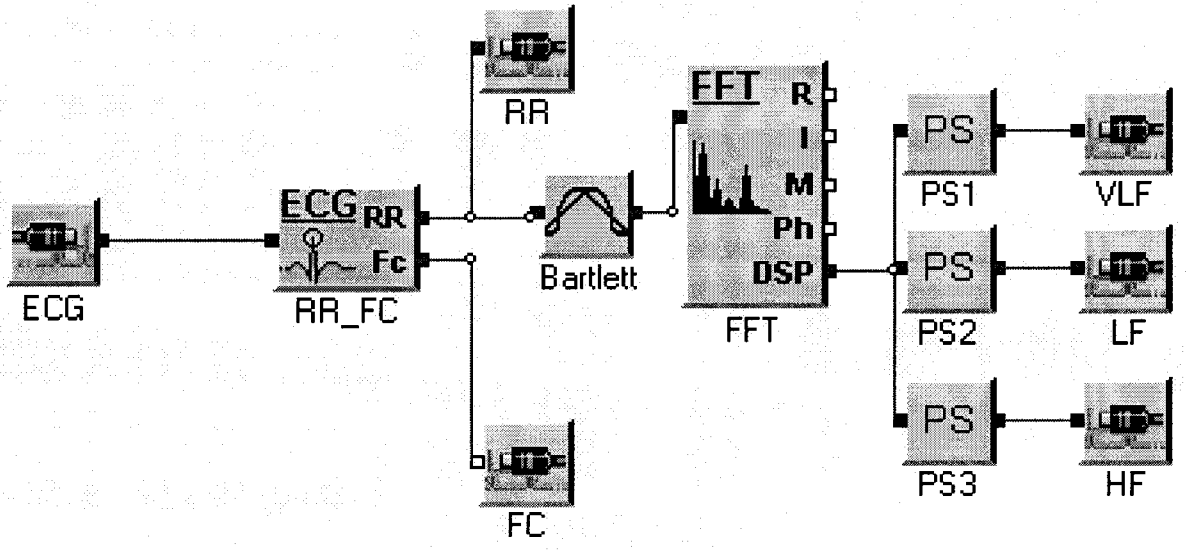


Figure 5.16 : Interface de programmation graphique

Cet exemple montre la représentation d'un algorithme d'analyse de la variabilité de la fréquence cardiaque. Le RR et la fréquence cardiaque (FC) sont calculés à l'aide du composant ECG connecté à la source. Une fenêtre glissante intégrant un fenêtrage de Bartlett est appliquée sur la série RR. Le signal résultant est ensuite traité par Transformée de Fourier Rapide (TFR). Enfin, la puissance spectrale est calculée dans les zones de très basses (TBF), basses (BF) et hautes fréquences (HF).

Sur un tel graphique, les arcs représentent les données et les nœuds, les traitements mathématiques.

5.2 – Visualisation des résultats :

Un système de création d'interface de visualisation a également été développé. Il permet à l'utilisateur de créer un tableau de bord personnalisé pour la visualisation des résultats de l'analyse.

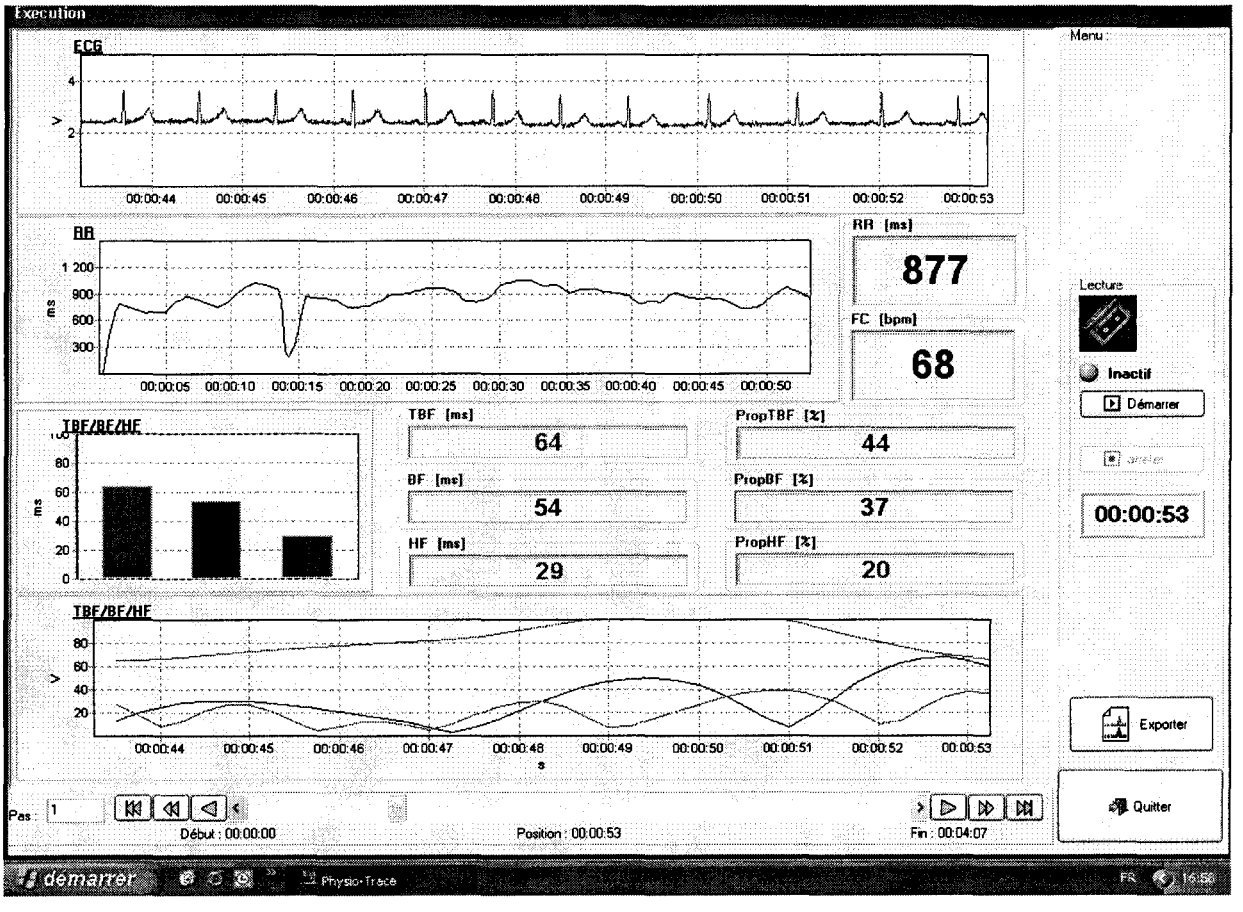


Figure 5.17 : Interface d'exécution

Cet exemple montre le résultat de l'analyse de la variabilité de la fréquence cardiaque. Le signal ECG, la série RR et les valeurs calculées sont représentés dans les différents scopes et afficheurs de l'interface de visualisation.

6 - Conclusion

Par généralisation des classes et sous-classes définies au chapitre 3, nous avons réalisé une bibliothèque de composants logiciels pour l'acquisition et le traitement des signaux physiologiques. Cette bibliothèque, couplée au réseau de terrain présenté au chapitre précédent, constitue une plateforme de conception spécialisée pour la réalisation de dispositifs de monitorage médical.

Cette station d'acquisition et de traitement ([LOGI 04.3], [DEJO 05]) a été testée dans un grand nombre d'études cliniques et de développements de dispositifs innovants pour le traitement des signaux physiologiques. Pour chacune de ces études, l'intérêt principal de cette station réside dans la facilité d'assemblage des modules et l'interface graphique de création d'algorithmes de traitement du signal. Ainsi, couplé au réseau de modules, un tel outil représente un moyen simple et efficace pour le prototypage rapide d'application de monitoring médical.

Le chapitre suivant présente une utilisation particulière de l'outil développé pour le monitorage du niveau d'analgésie au cours de l'anesthésie générale.

Chapitre 6

Exemple d'utilisation : le monitorage du niveau d'analgésie

1 – Introduction 110

2 – Le Système Nerveux Autonome 110

2.1 - LA NEUROTRANSMISSION 110
2.2 - EFFETS DU SNA SUR LE SYSTÈME CARDIOVASCULAIRE (SCV) 111
2.3 – RÉGULATION DU SYSTÈME CARDIO-VASCULAIRE (SCV) PAR LE SNA 113
2.3.1 – Principe: 113
2.3.2 Modèle cybernétique 115
2.4 – CARACTÉRISTIQUES SPECTRALES DU RYTHME CARDIAQUE 116

3 – Prise en charge du niveau d'analgésie 117

3.1 – PRINCIPE 117
3.2 - ACQUISITION DE LA SÉRIE RR 119
3.3 – TRAITEMENT DE LA SÉRIE RR 120
3.3.1 – Filtrage 120
3.3.2 – Prétraitement 121
3.3.3 – Détection du pattern respiratoire 122
3.4 – CALCUL DES PARAMÈTRES 122
3.5 – MESURE EN CONTINU 124

4 - Réalisation de l'algorithme 124

4.1 – DÉCOMPOSITION EN MODULE ÉLÉMENTAIRE : 124
4.1.1 – Acquisition de l'ECG 125
4.1.2 – Construction de la série RR 125
4.1.3 – Filtrage de la série RR 126
4.1.4 – Ré-échantillonnage 126
4.1.5 – Normalisation 127
4.1.6 – Filtrage par ondelette 128
4.1.7 – Calcul des paramètres 128

5 – Résultat 129

6 – Conclusion : 130

1 – Introduction :

Dans ce chapitre, nous présentons la réalisation d'une application innovante de monitorage médical à partir des outils préalablement décrits. Il s'agit d'un dispositif pour la validation de paramètres permettant d'appréhender le niveau d'analgésie lors de l'anesthésie générale. En effet, les différentes études préliminaires effectuées au CHRU de Lille sur des tachogrammes préalablement enregistrés ont permis de mettre en évidence plusieurs paramètres basés sur l'analyse des variations du rythme cardiaque. Cependant, la vérification des hypothèses établies lors de ces études nécessite désormais l'implémentation temps réel de cette méthode de calcul. Dans cette optique, après avoir présenté les principes physiologiques et la méthodologie de calcul des paramètres d'investigation du niveau d'analgésie, nous présentons les différentes étapes nécessaires à la réalisation du dispositif à l'aide de la bibliothèque de composants. Le prototype ainsi développé permettra de réaliser la validation clinique des différents paramètres diagnostiques.

2 – Le Système Nerveux Autonome

Le SNA s'organise en deux pôles à la fois opposés et complémentaires. Le système nerveux ***sympathique*** (SNS) et le système nerveux ***parasympathique*** ou ***vagal*** (SNPS). La stimulation du SNS a pour effet, notamment, de dilater les bronches, d'accélérer l'activité cardiaque et respiratoire ou encore d'augmenter la tension artérielle. La stimulation du SNPS provoque quant à elle le ralentissement général des organes.

Il existe en permanence entre ces deux pôles des interactions complexes responsables de ***l'équilibre sympathovagal*** (ou ***balance vago – sympathique***). Chacun des pôles sympathique et vagal a ses propres ***neurotransmetteurs*** et ***récepteurs***.

2.1 - La neurotransmission :

Chaque neurone sympathique ou parasympathique libère différentes substances (***neurotransmetteurs***) auxquelles les différents récepteurs sont sensibles.

Ainsi, tous les neurones pré-ganglionnaires sympathiques ou parasympathiques libèrent de l'***acétylcholine*** (Ach). Les neurones post-ganglionnaires parasympathiques libèrent eux aussi de l'Ach tandis que la plupart des neurones post-ganglionnaires sympathiques libèrent de la ***noradrénaline*** et de l'***adrénaline***.

On distingue deux types de récepteurs : les ***récepteurs α*** et les ***récepteurs β***. Ces deux types sont eux même divisés en deux et trois types respectivement : ***α1 et α2 et β1, β2 et β3***. L'adrénaline active les deux types de récepteurs de façon équivalente alors que l'effet de la noradrénaline prédomine sur les récepteurs alpha. Ainsi, les effets de l'adrénaline et de la noradrénaline sur chaque organe dépendent de la distribution de ces récepteurs (Figure 6.1).

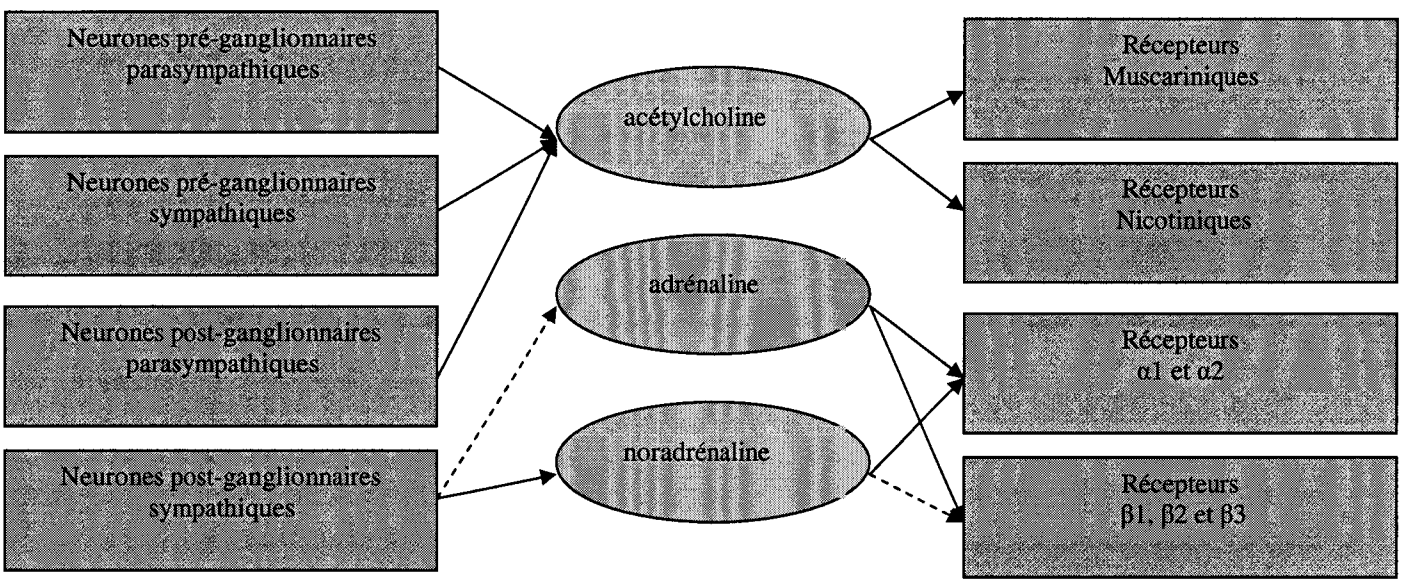


Figure 6.1 : neurotransmition

2.2 - Effets du SNA sur le Système Cardiovasculaire (SCV) :

Le SNS et le SNPS effectuent la régulation du SCV en fonction de l'activation de chacun des récepteurs.

On observe trois types d'effet du SNA sur le cœur :

- **Chronotropisme :**

  Cet effet correspond à une modification de la ***fréquence de décharge*** du nœud sinusal. Ceci se caractérise par une modification de la ***fréquence cardiaque***.

- **Dromotropisme :**

Cet effet correspond à une modification de la ***vitesse de conduction*** entre le nœud sinusal et le nœud auriculo-ventriculaire. Ceci se traduit par une modification de l'***intervalle QT*** sur l'électrocardiogramme (ECG).

- **Inotropisme :**

Cet effet correspond à une modification de la ***contractilité du cœur***. Ceci se traduit par une modification du ***volume*** et des ***pressions d'éjections systoliques***.

Ainsi, les neurotransmetteurs influencent les comportements ***chronotrope***, ***dromotrope*** et ***inotrope*** du cœur par l'intermédiaire des différents récepteurs mais ils modulent également l'***élasticité des artères*** et des veines en provoquant un effet ***vasoconstricteur*** et / ou ***vasodilatateur*** (*figure 6.2*).

L'activation des nerfs du SNS cardiaque induit la libération de noradrénaline tandis que l'activation du SNPS cardiaque induit la libération d'acétylcholine.

	Adrénaline / Noradrénaline	Acétylcholine
Chronotropisme	Augmentation (récepteurs β1β2)	Diminution (récepteurs M2)
Dromotropisme	Augmentation (récepteurs β1)	Diminution (récepteurs M2)
Inotropisme	Augmentation (récepteurs β1β2)	Très peu d'influence
Elasticité des artères et veines	Vasoconstriction par effet direct (récepteur α1) puis vasodilatation par effet indirect (récepteur β2)	Vasodilatation (récepteur M3)

Figure 6.2 : effet des différents neurotransmetteurs sur le SCV

2.3 – Régulation du système cardio-vasculaire (SCV) par le SNA :

2.3.1 – Principe :

Au niveau cardiovasculaire, les composants qui interagissent par le biais du SNA sont le ***cœur*** et le ***système circulatoire*** mais également la ***température*** et la ***respiration***. Chacun de ces composants possède des capteurs qui enregistrent des variations et envoient des informations vers les centres cérébraux qui envoient une réponse aux composants par le biais des deux branches du SNA. On obtient donc la boucle de régulation de la figure 6.3 :

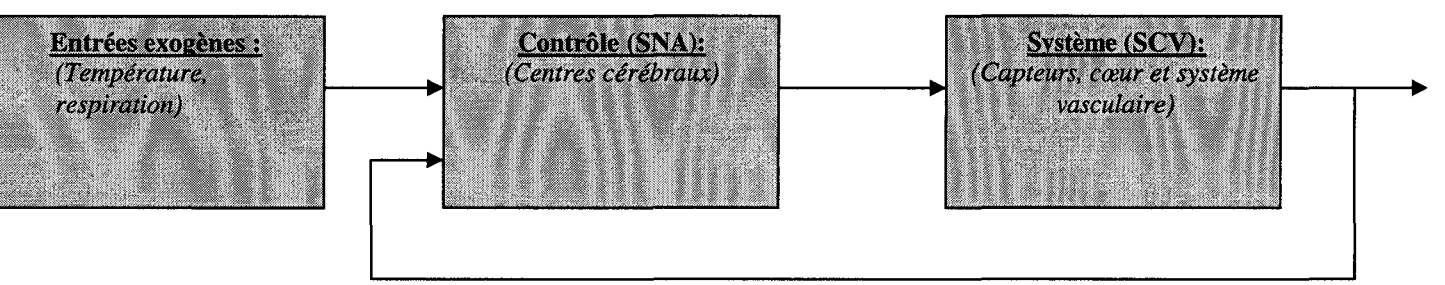


Figure 6.3 : boucle de régulation du SCV par le SNA

Les capteurs :

Chaque composant du SCV possède son propre ensemble de capteurs.
Les capteurs les plus importants pour le système circulatoire sont les ***barorécepteurs***.

- **Barorécepteurs :**

Les barorécepteurs se situent principalement sur l'aorte près du cœur et sur le sinus carotidien dans le cou. Ces capteurs sont sensibles aux ***déformations des veines et artères*** ; ils sont donc fortement impliqués dans la régulation de la pression artérielle. Lors d'un changement de pression artérielle, le capteur est dilaté ou contracté.

- **Chimiorécepteurs :**

Ces récepteurs analysent les ***gaz du sang*** (O_2, CO_2, H^+) et transmettent les informations au centre respiratoire. Par exemple, si une baisse de O_2 intervient, la fréquence et le volume

respiratoire seront augmentés. Ils agissent également sur le système circulatoire afin de dilater ou contracter les vaisseaux.

- **Thermorécepteurs :**

Ces récepteurs se trouvent à l'intérieur du corps et sur la peau. Ils envoient leurs informations sur la ***température*** à l'hypothalamus qui transmet des stimuli au système circulatoire par le biais du SNA. Par exemple, si la température est trop élevée il en résulte une dilatation des vaisseaux. Le temps de réponse de cette boucle est supérieur à 1 min. Dans la mesure où nous nous intéressons à des variations à court terme (1s), l'influence de la température sur le SCV n'est pas prise en compte dans la suite de ce mémoire.

- **Mécanorécepteurs :**

Ces récepteurs sont sensibles à l'***étirement de la cage thoracique***. Cependant, leur mode d'interaction avec les centres cérébraux reste peu connu.

Les entrées exogènes :

La ***température*** et la ***respiration*** influencent elles aussi le comportement du SCV par l'intermédiaire du SNA. En effet, la ***température*** modifie le comportement du SCV par le biais des ***thermorécepteurs*** et de l'hypothalamus. La ***respiration*** influence quant à elle les ***pressions artérielle et veineuse centrale*** par un effet ***mécanique*** lié à la compression des artères et des vaisseaux due aux mouvements respiratoires. La ***respiration*** est également le lieu d'un phénomène appelé ***l'arythmie sinusale respiratoire*** (ASR).

L'ASR se caractérise par une ***accélération*** de la fréquence cardiaque (FC) lors de l'***inspiration*** ([CONS 01]). Ce mécanisme a des origines ***centrales*** (réflexes) dont l'explication serait une interaction directe entre les centres autonomes qui contrôlent le cœur et le centre respiratoire ([CLAI 97]). La ***décélération*** de la FC lors de l'***expiration*** serait quant à elle due à un phénomène ***réflexe*** ayant des origines ***mécaniques*** fortement liées à la compression des vaisseaux due aux mouvements respiratoires ([CLAI 97]).

Dans un but de simplification, on appelle ASR, par abus de langage, l'ensemble des influences de la respiration sur la FC (accélération et décélération).

Les transmetteurs d'information :

Pour transmettre l'information, le SNA utilise deux types de ***nerfs*** : ***myélinisés*** pour le système ***parasympathique*** (réponse à court terme) ou ***non myélinisés*** pour le système ***sympathique*** (changement à long terme – changement de la valeur moyenne).
Les fibres non myélinisées ont un temps de réponse 10 fois plus grand que les fibres myélinisées.

Le contrôleur :

Le contrôle effectue l'intégration des informations des capteurs et des entrées exogènes et génère, vers le SCV, une ***commande vagale*** ou ***sympathique,*** afin de l'adapter à la nouvelle situation.

2.3.2 Modèle cybernétique :

L'ensemble des phénomènes physiologiques décrit précédemment peut être modélisé par le schéma de la figure 6.4.

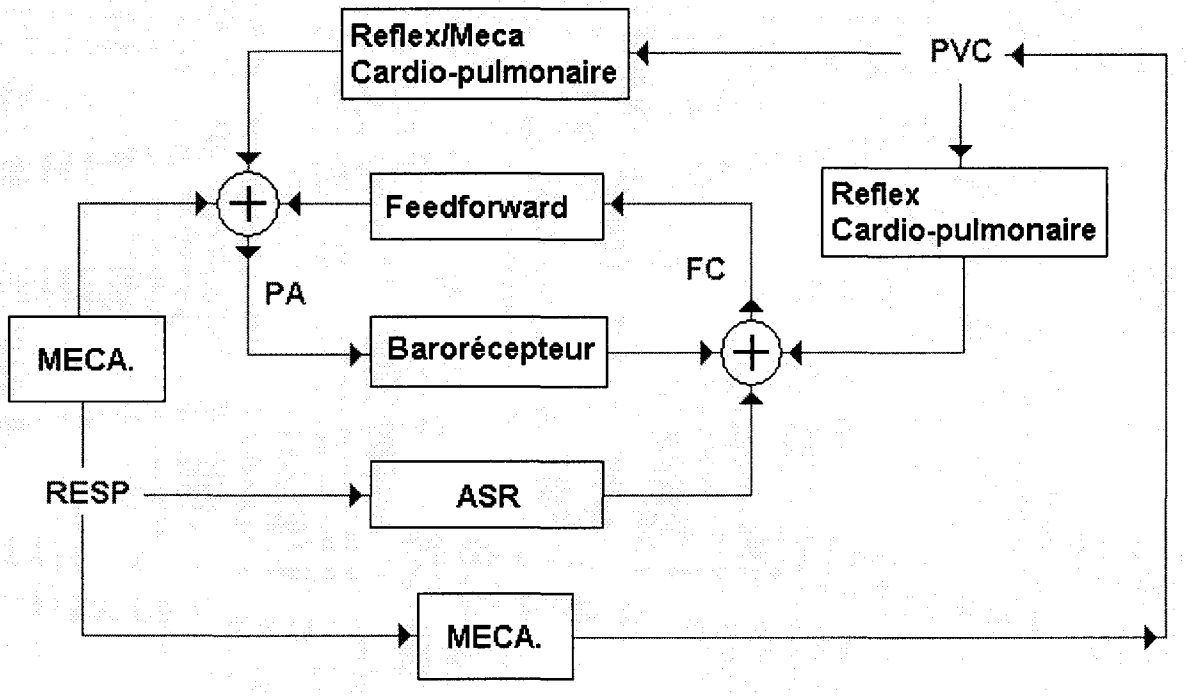


Figure 6.4 : modèle cybernétique

Les différents blocs fonctionnels qui apparaissent sur la figure 6.4 réalisent les mécanismes biologiques qui transforment les différents signaux physiologiques.

Ainsi, la pression artérielle (***PA***) influence la fréquence cardiaque (***FC***) par l'intermédiaire des ***barorécepteurs.*** En retour, la FC influence mécaniquement la PA par l'intermédiaire du cœur et du système vasculaire (***Feedforward***). La respiration (***RESP***), a quant à elle, un effet mécanique (***MECA.***) sur la pression artérielle et un effet réflexe correspondant à l'Arythmie Sinusale Respiratoire (***ASR***) sur la fréquence cardiaque. La respiration influence également la pression veineuse centrale (***PVC***) par un effet mécanique (MECA2). A son tour la PVC influence elle-même la FC par un effet ***réflexe cardio-pulmonaire*** et la PA par un effet ***cardio-pulmonaire*** à la fois ***réflexe et mécanique.***

2.4 – Caractéristiques spectrales du rythme cardiaque :

La méthode de référence pour l'étude du SNA est l'analyse des ***variations du rythme cardiaque*** par transformation de ***Fourier*** ([LOGI 90], [BIAN 97]). Cette analyse est basée sur le principe que les fluctuations du rythme cardiaque sont le reflet de l'activité des systèmes sympathique et vagal, principales composantes du SNA.

Ainsi, le tracé du spectre de Fourier de la FC (*figure 6.5*) et le calcul des puissances spectrales deviennent un outil de référence pour l'analyse du SNA.

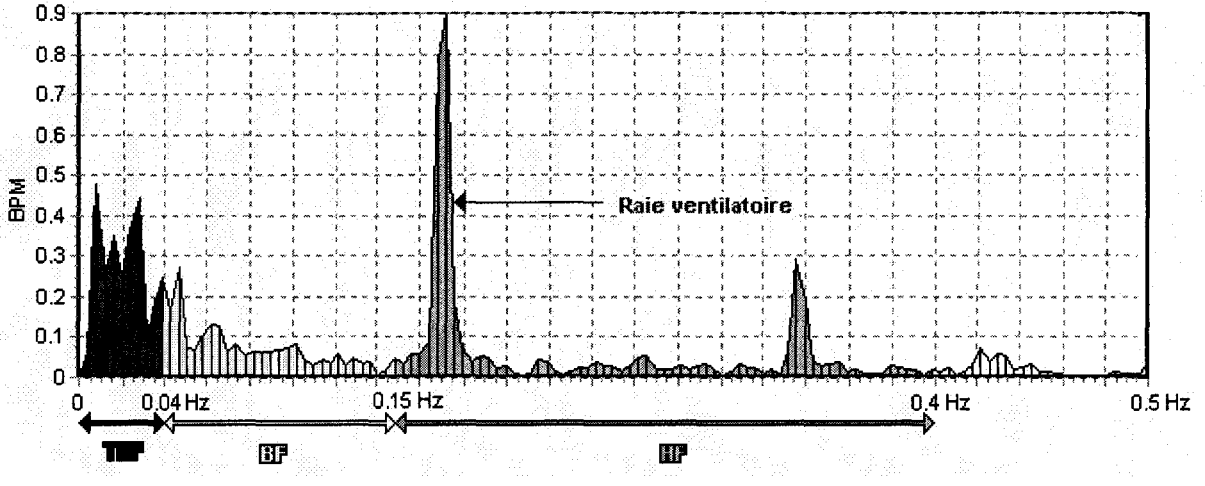


Figure 6.5 : spectre de Fourier pour l'analyse des variation de la Fc.

Les différentes études effectuées sur l'analyse spectrale du rythme cardiaque ont permis de mettre en évidence trois zones de fréquences ([CONS 01]) :

- ***les Très Basses Fréquences (TBF)*** de 0 à 0.04 Hz, correspondant entre autre à la régulation de la FC par la température (rarement étudié),
- ***les Basses Fréquences (BF)*** de 0.04 à 0.15 Hz, correspondant essentiellement à l'influence de la pression artérielle sur la FC,
- ***les Hautes Fréquences (HF)*** de 0.15 à 0.4 Hz, correspondant en grande partie à l'effet de la respiration sur la FC (ASR). Il en résulte que l'analyse spectrale de la FC est un bon moyen d'apprécier l'ASR par isolation de cette dernière zone spectrale.

3 – Prise en charge du niveau d'analgésie

3.1 – Principe :

La prise en charge du niveau d'analgésie durant les procédures médicales représente une priorité pour l'optimisation de la prescription de drogues analgésiques.

Dans le cas de patients adultes conscients, la douleur est évaluée par le biais de l'Echelle Visuelle Analogique (EVA) qui consiste en une règle graduée de dix centimètres sur laquelle le patient déplace un curseur en fonction de son degré de douleur (figure 6.6).

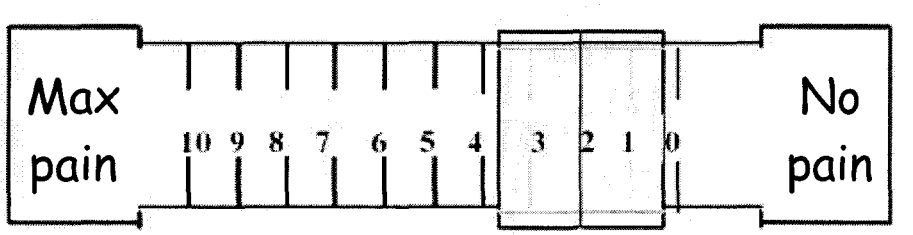


Figure 6.6 : échelle Visuelle Analogique (EVA)

Toutefois, l'EVA ne donne qu'une information subjective et ponctuelle. De plus, une telle mesure demeure inutilisable sur les patients inconscients comme par exemple au cours d'une anesthésie générale ou en soins intensifs.

L'anesthésie générale utilise la combinaison d'une composante hypnotique et d'une composante analgésique associées parfois à une myorelaxation par agents curarisants pour permettre la réalisation d'actes douloureux ou réflexogènes. Cependant, les paramètres cliniques et para cliniques (fréquence cardiaque et pression artérielle) utilisés habituellement pour surveiller le patient sous anesthésie générale manquent de sensibilité et de spécificité et ne permettent pas d'anticiper précisément les besoins en antalgiques des patients. D'autre part, les médicaments puissants utilisés exposent au risque constant de surdosage et imposent de rechercher les doses minimales nécessaires pour la réalisation du geste chirurgical.

De nombreuses études ont montré que l'analyse de la variabilité sinusale du rythme cardiaque reflétait les variations du Système Nerveux Autonome (SNA) au cours de l'induction ainsi que pendant l'entretien anesthésique ([LATS 93], [TASK 96]). Ainsi, la connaissance de l'activité du SNA peut être d'une aide précieuse dans l'élaboration d'un diagnostic dans bon nombre de situations cliniques et notamment dans le domaine de l'anesthésie pour l'appréciation de la ***profondeur d'anesthésie*** et de la ***douleur per-opératoire***.

Si quelques auteurs proposent un index global de profondeur de l'anesthésie, aucun en revanche n'a décrits d'index reflétant la profondeur de la composante antalgique au cours de l'anesthésie générale.

Comme nous l'avons présenté précédemment, La méthode de référence pour l'étude du SNA est l'analyse des ***variations du rythme cardiaque*** par transformation de ***Fourier*** ([LOGI 90], [BIAN 97]). Cette analyse est basée sur le principe selon lequel les fluctuations du rythme cardiaque sont le reflet de l'activité des systèmes sympathique et vagal, principales composantes du SNA. Depuis une dizaine d'années, l'équipe « Biocapteur et Instrumentation » de l'Institut de Technologie Médical (ITM) travaille sur l'étude du SNA ([LOGI 90], [LOGI 01]). Les récents travaux de cette équipe, en collaboration avec les différents services d'anesthésie du CHRU de Lille, ont permis la mise en œuvre de nouvelles méthodes d'analyse non invasives pour l'étude de la variabilité du rythme cardiaque ([DEJO 03]). Ces méthodes d'analyse ont ensuite permis de mettre en évidence des paramètres sensibles à l'influence de la douleur sur le SNA ([JEAN 04], [JEAN 04.1], [LOGI 04.1], [LOGI 04.2], [LOGI 06]).

Notre méthode d'analyse pour évaluer le niveau d'analgésie pendant l'anesthésie générale est basée sur une analyse temporelle de la variabilité de la fréquence cardiaque. L'enregistrement de séries RR (cf chapitre1) nous a permis de mettre en évidence des changements morphologiques du signal induits par des stimulations chirurgicales durant l'anesthésie générale. Notamment, nous avons remarqué que, dans le cas d'une anesthésie stable, la série RR est principalement régulée par l'ASR. Il en résulte qu'une oscillation régulière d'origine respiratoire apparaît clairement sur la série RR (figure 6.7).

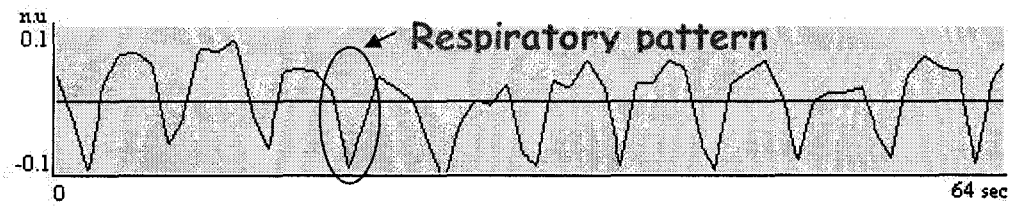


Figure 6.7 :série RR dans le cas d'une anesthésie stabilisée

Cette oscillation devient chaotique dès que l'anesthésie est perturbée par des événements extérieurs. En particulier, nous avons remarqué que les événements douloureux, comme l'incision chirurgicale, provoquent une diminution de l'amplitude des oscillations induites par l'ASR (figure 6.8).

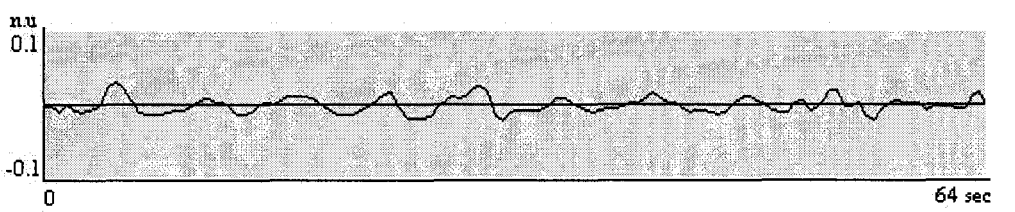


Figure 6.8 : série RR dans la cas d'un événement douloureux

Conformément à ces observations, nous avons développé un algorithme d'évaluation du niveau d'analgésie basé sur l'analyse de l'amplitude des oscillations induites par la respiration (ou pattern respiratoire) sur la signal RR ([LOGI 04.1], [LOGI 04.2], [LOGI 06]).

3.2 - Acquisition de la série RR :

Comme nous l'avons vu dans le premier chapitre, la valeur instantanée du RR représente l'intervalle de temps entre deux ondes R de l'électrocardiogramme (ECG) (figure 6.9). La

série RR est donc obtenue par détection successive des ondes R de l'ECG sur un signal ECG acquis par le biais du réseau de modules et échantillonné à 250 Hz.

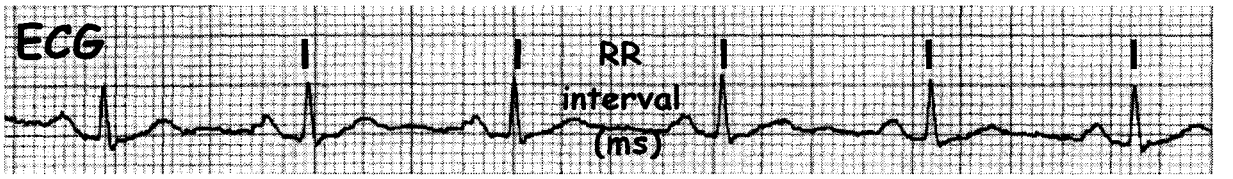


Figure 6.9 : détection des ondes R sur l'ECG

L'algorithme de détection de l'onde R est celui décrit au chapitre 1. On obtient ainsi la série RR (figure 6.10).

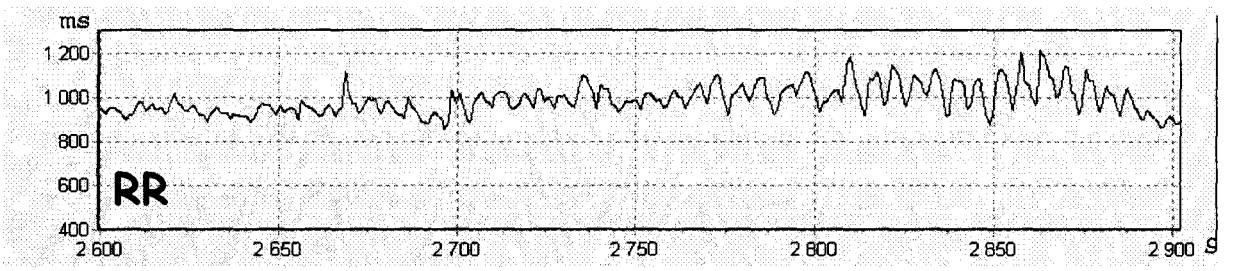


Figure 6.10 : série RR

3.3 – Traitement de la série RR :

3.3.1 – Filtrage :

Sur une longue période d'enregistrement, il est extrêmement difficile d'obtenir une série RR sans aucune perturbation. En effet, le signal ECG est un signal extrêmement bruité et l'effet de chacune des perturbations de l'ECG sur la série RR donne une évaluation erronée de l'analyse de la variabilité du RR. Par conséquent, nous avons développé un dispositif de filtrage intelligent de la série RR ([LOGI 01.1], [LOGI 04]) permettant une analyse continue de la variabilité de la fréquence cardiaque. Le dispositif de filtrage utilisé est décrit en annexe 7.

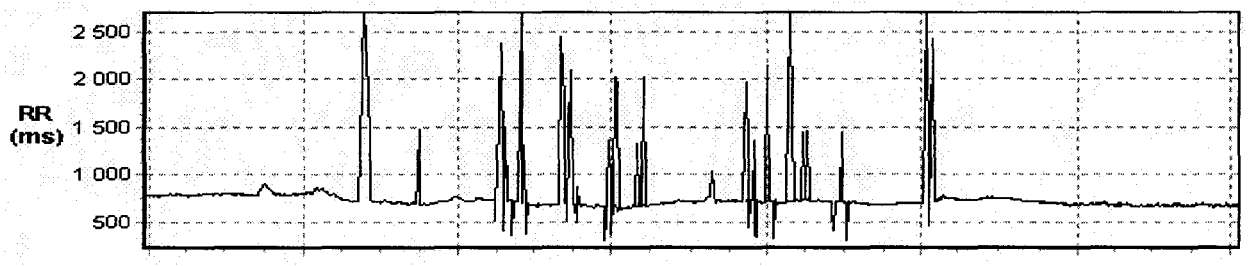


Figure 6.11a : série RR avant filtrage

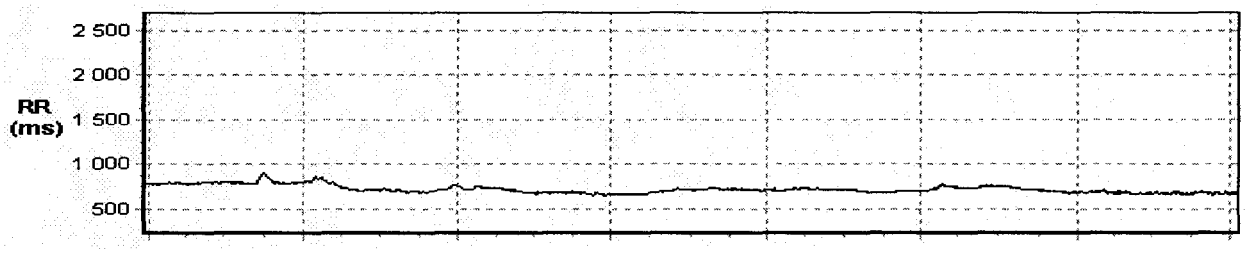


Figure 6.11b : série RR après filtrage

3.3.2 – Prétraitement :

Afin de calculer les paramètres issus du pattern respiratoire, les échantillons de la série RR sont re-échantillonnés à 8 Hz et isolés dans une fenêtre glissante de 64 s (soit 512 échantillons).

Le signal est ensuite centré par suppression de la valeur moyenne.

Afin d'obtenir des valeurs de paramètres exempts de toute variabilité inter patients, le signal est ensuite normalisé dans la fenêtre d'analyse. L'algorithme de normalisation consiste à calculer la norme S du signal selon la formule suivante :

$$S = \sqrt{\sum_{i=1}^{N} (RR_i)^2}$$

Ou RRi représente la valeur de l'échantillon RR et N le nombre de RR dans la fenêtre.
Dans un second temps, chaque échantillon de la fenêtre est divisé par la valeur de S.

3.3.3 – Détection du pattern respiratoire :

La méthode d'analyse se basant sur la mesure des changements du pattern respiratoire, il est nécessaire d'isoler ce pattern de l'ensemble des phénomènes influençant le rythme cardiaque par le biais du SNA. Pour cela, la série RR re-échantillonnée et fenêtrée est ensuite traitée par un filtre passe-bande entre 0.1 et 1 Hz.

Pour cela, on utilise un filtre numérique basé sur l'utilisation des ondelettes (Annexe 6). En effet, les méthodes d'analyses par ondelettes ont déjà montré leurs possibilités dans de nombreuses applications biomédicales et notamment dans le domaine de l'analyse de la variabilité de la fréquence cardiaque [KYRI 07]. De façon générale, de telles méthodes sont utilisées pour calculer la distribution de l'énergie d'un signal. Cependant, les ondelettes peuvent aussi être utilisées comme filtre passe bande lorsque les filtres numériques classiques montrent leurs limites. En particulier, les filtres basés sur la théorie des ondelettes permettent d'isoler une ou plusieurs bandes de fréquences du signal sans aucun déphasage. Pour réaliser notre filtre passe-bande, nous avons donc choisi d'utiliser un filtre utilisant une ondelette de Daubechie à quatre coefficients [MEYE 90]. Le tableau suivant montre les fréquences de coupure du filtre en fonction du niveau d'ondelette.

Niveau	Echelle	Pseudo-Fréquence (bpm)	Pseudo-Fréquence (Hz)	Domain Fréquentiel
1	2	160	2.6667	HF
2	4	80	1.3333	HF
3	8	40	0.6667	HF
4	16	20	0.3333	HF
5	32	10	0.1667	HF
6	64	5	0.0833	LF
7	128	2.5	0.0417	LF
8	256	1.25	0.02085	VLF

Figure 6.12 : fréquences de coupure du filtre

3.4 – Calcul des paramètres :

Après détection du pattern respiratoire, il est nécessaire de déterminer le paramètre de quantification le plus adapté pour la mesure de l'analgésie per-opératoire. Lors des études

cliniques préliminaires ([JEAN 04], [JEAN 04.1]), un grand nombre de paramètres ont été testés (aire sous la courbe total, aire sous l'enveloppe max, aire sous l'enveloppe min, aire sous l'enveloppe total...). Dans ce paragraphe, nous décrivons uniquement les paramètres apparaissant comme étant les plus significatifs (après analyse statistique) pour la quantification du niveau d'analgésie.

Ainsi, nous évaluons les changements du pattern respiratoire en calculant l'aire sous la courbe de la série RR. Pour cela, on détecte chaque minimum de la série RR dans la fenêtre d'analyse (figure 6.13a).

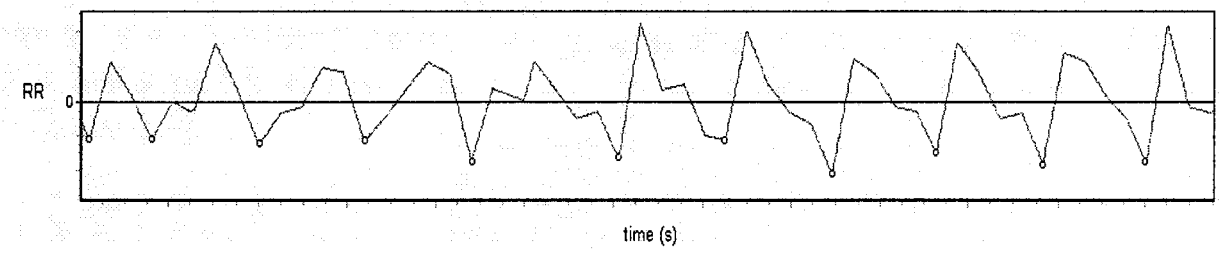


Figure 6.13a : détection des minimum

Ensuite, afin d'obtenir une analyse temporelle compatible avec la dynamique d'un cycle respiratoire, la fenêtre glissante est divisée en quatre sous-fenêtres de seize secondes chacune (figure 6.13b).

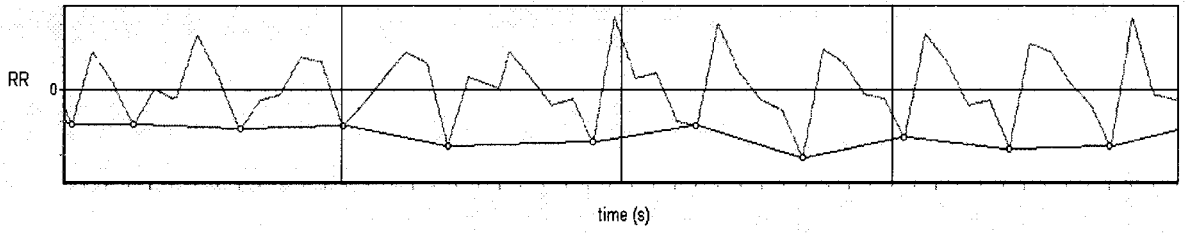


Figure 6.13b : division en sous-fenêtres

Enfin, dans chaque sous fenêtre, on calcule l'aire comprise entre la courbe reliant les minima et l'axe des abscisses obtenant ainsi quatre aires ; A1, A2, A3 et A4 (figure 6.13c).

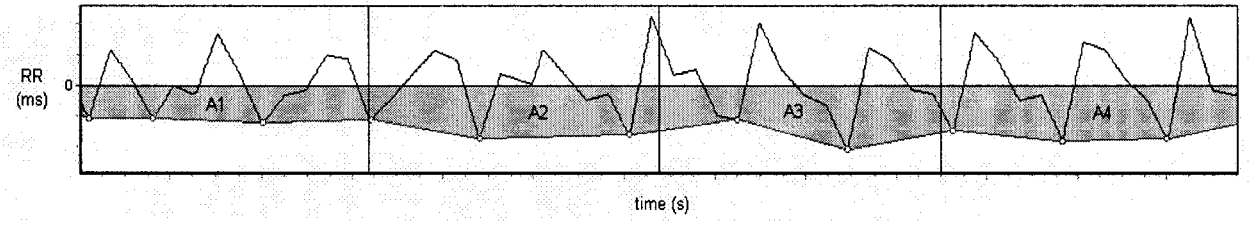


Figure 6.13c : calcul des paramètres

De ces quatre aires, 2 paramètres sont testés.

AUCmoy : qui représente la valeur moyenne des quatre surfaces.

AUCmax : qui représente la valeur maximum des quatre surfaces.

Ces deux paramètres constituent les index représentatifs du niveau d'analgésie.

3.5 – Mesure en continu :

La mesure en continu des paramètres **AUCmoy** et **AUCmax** peut être effectuée en décalant la fenêtre d'analyse de 64 s après chaque calcul. Ainsi, la fréquence d'échantillonnage du paramètre final dépend de la fréquence de décalage de la fenêtre glissante. En pratique, un décalage de 5 s donne des courbes de tendance acceptables pour l'analyse des paramètres.

4 - Réalisation de l'algorithme

4.1 – Décomposition en module élémentaire :

La figure ci-dessous décrit la décomposition de l'algorithme réalisant les taches décrites ci dessus :

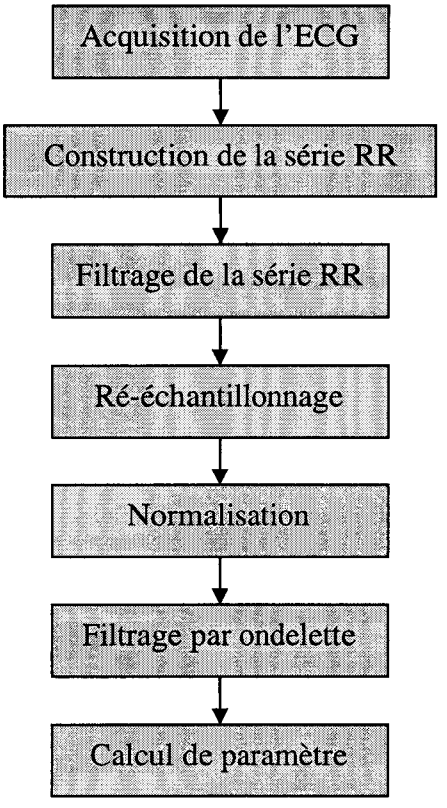


Figure 6.14 : décomposition de l'algorithme

4.1.1 – Acquisition de l'ECG :

L'acquisition de l'ECG se fait par l'intermédiaire du module ECG du réseau de terrain présenté au chapitre 4. D'un point de vue logiciel, l'interface entre l'application et ce matériel passe donc par l'utilisation du composant ***TSourceRdT*** présenté au chapitre précédent.

4.1.2 – Construction de la série RR :

La construction de la série RR, nécessite, dans un premier temps, la détection des ondes R de l'ECG (composant ***TDetecR***). Cependant, avant d'effectuer cette détection, il est préférable de filtrer le signal ECG afin d'éliminer les changements de ligne de base (perturbation basse fréquence) et le bruit (perturbation haute fréquence) pour mieux isoler le complexe QRS à détecter (composants ***TFiltre***). Conformément aux caractéristiques de l'ECG, on utilise pour cela un filtre passe-haut à 15 Hz et un filtre passe-bas à 20 Hz. Enfin, on calcule le temps séparant deux détections d'ondes R pour déterminer l'intervalle RR (composant ***TBloqueur***).

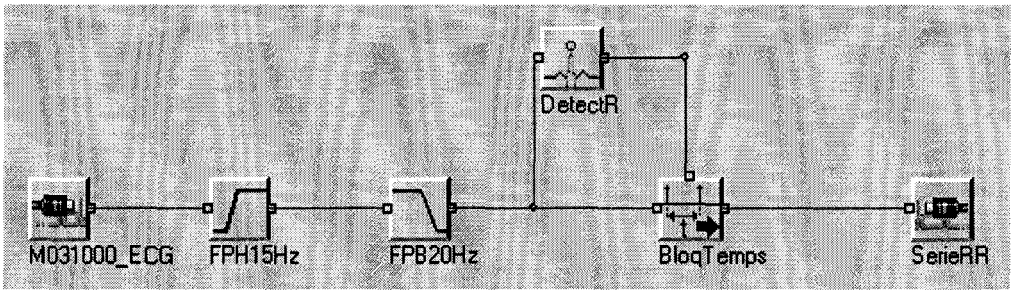


Figure 6.17 : construction de la série RR

Par composition statique, cet assemblage de composants donne naissance au composant ***TECG***.

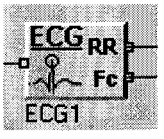


Figure 6.18 : composant ECG

4.1.3 – Filtrage de la série RR :

L'algorithme utilisé pour le filtrage de la série RR est décrit en annexe 6 (composant ***TFiltreRR***).

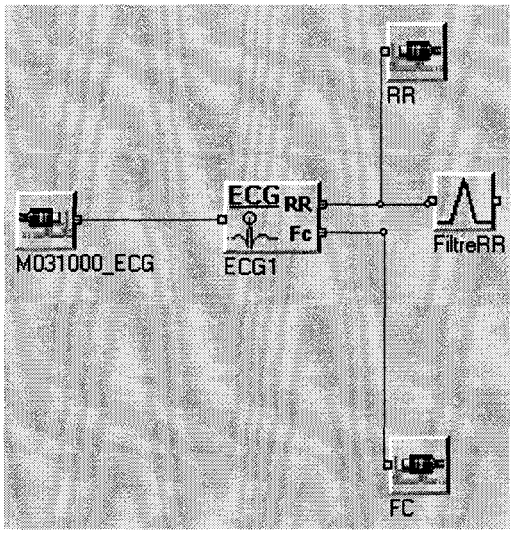


Figure 6.19 : filtrage de la série RR

4.1.4 – Ré-échantillonnage :

Conformément au théorème de Shannon, après filtrage, la série RR est ré échantillonnée à 8 Hz (composant ***TReechantil***).

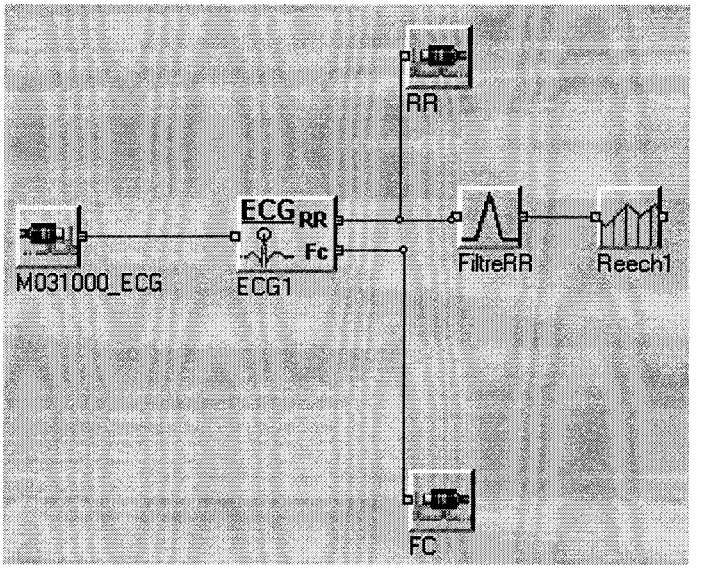


Figure 6.20 : ré échantillonage

4.1.5 – Normalisation :

Le signal est ensuite isolé dans une fenêtre de 64s. C'est également ce composant qui effectue la normalisation du signal (composant ***TFenetrage***).

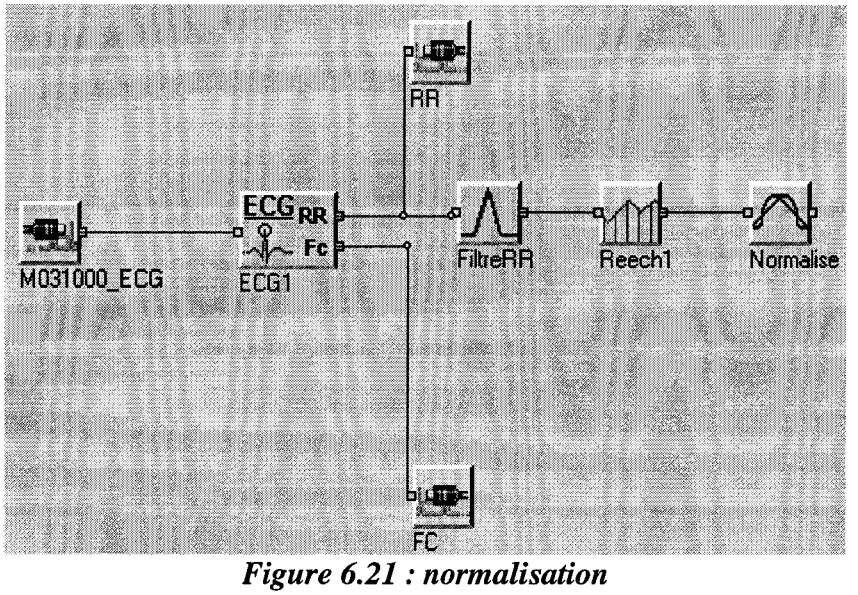


Figure 6.21 : normalisation

4.1.6 – Filtrage par ondelette :

Le filtrage est réalisé en utilisant une ondelette de Daubechie à 4 coefficients sur la fenêtre de 64 s ré échantillonnée à 8 Hz (1024 points). On applique ensuite un algorithme de transformée d'ondelette inverse en ne tenant compte que des itérations 2, 3, 4, 5 et 6.

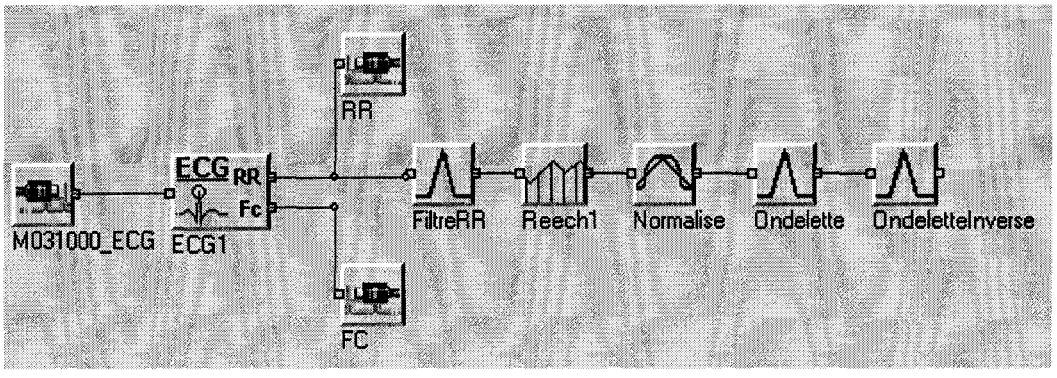


Figure 6.22 : filtrage

4.1.7 – Calcul des paramètres :

Le calcul des paramètres **AUCmoy** et **AUCmax** nécessite le développement et l'intégration d'un nouveau composant dans la bibliothèque. Ce composant effectue les différents calculs présentés au paragraphe 3.4 (détection des minimums sur la série RR filtrées, calcul des aires sous la courbe des minimums dans 4 fenêtres de 16 secondes chacune).

Ce composant possède un port d'entrée (InputPortBuf) et deux ports de sortie (OutputPortReal et OutputPortReal1) pour la transmission des paramètres AUCmoy et AUCmax.

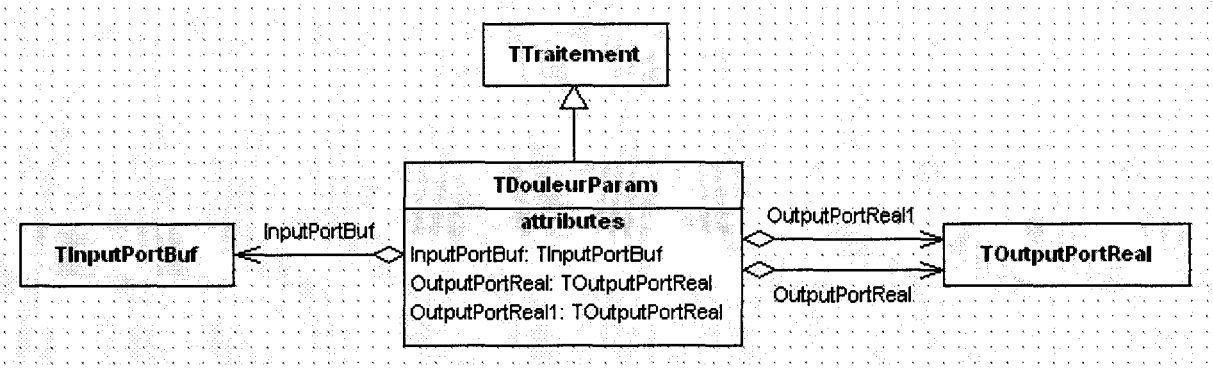


Figure 6.23 : définition du composant TDouleurParam

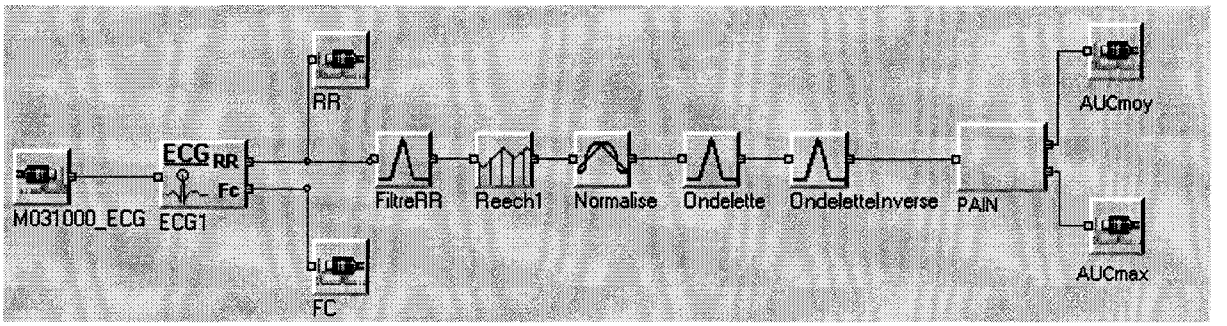


Figure 6.24 : calcul des index de douleur

5 – Résultat

La figure 6.25 montre l'écran du micro-ordinateur au cours du monitorage d'un patient sous anesthésie générale. Sur cet écran, les signaux ECG, RR et le signal RR filtré significatif de l'effet de la respiration sur la série RR sont affichés dans des « scopes » (fenêtres de tracés) réglables en amplitude et en échelle de temps. Dans la partie supérieure du scope ECG, un pointeur permet un contrôle visuel de la qualité de la détection de l'onde R dont dépend la validité des résultats numériques. Sur la droite de l'écran, apparaissent les valeurs instantanées des paramètres calculés : la fréquence cardiaque **FC**, le **RR**, **AUCmoy** et **AUCmax**. Ainsi, l'utilisateur a la possibilité de visualiser l'ensemble de ces paramètres en temps réel tout au long de l'anesthésie.

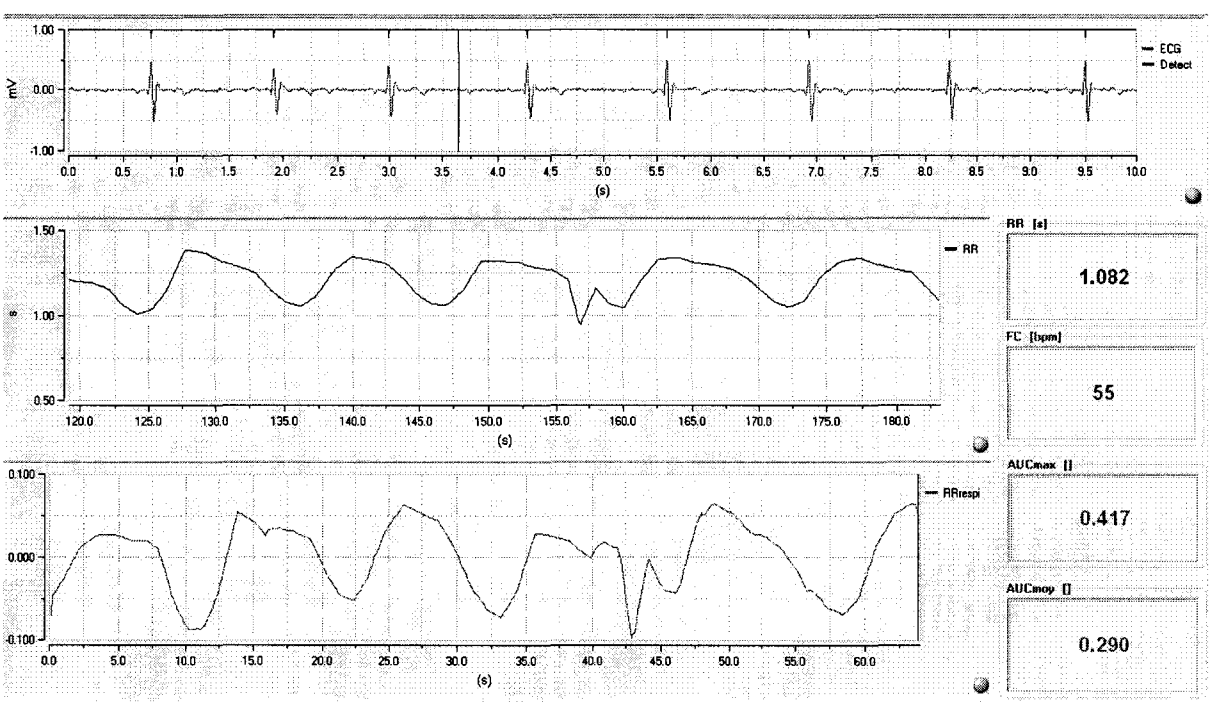


Figure 6.25 : application de monitorage du niveau d'analgésie

6 – Conclusion

Dans ce chapitre, nous avons voulu illustrer les résultats obtenus dans les chapitres précédents et voir leur application sur un cas concret de conception d'application de monitorage médical, à savoir, un dispositif de monitorage du niveau d'analgésie.

Dans un premier temps, nous avons décomposé l'algorithme présenté au paragraphe 3 sous forme de tâches élémentaires à exécuter séquentiellement (paragraphe 4). A partir de cette décomposition, nous avons réalisé chacune des tâches à partir des différents composants de la bibliothèque. Finalement, seul le calcul des paramètres AUCmax et AUCmin a nécessité le développement d'un nouveau composant ***TDouleurParam***. Ce composant est désormais intégré à la bibliothèque dans le but d'une éventuelle réutilisation.

Ainsi, à partir du signal issu du module ECG, nous aboutissons à l'élaboration des paramètres AUCmax et AUCmoy représentatifs du niveau d'analgésie. L'implémentation temps réel de cette méthodologie de calcul jusqu'alors uniquement utilisée et testée sur des enregistrements préalablement acquis permet désormais la validation des paramètres en continu.

Le prototype ainsi réalisé a été mis en évaluation dans les différents services d'anesthésie du CHR&U de Lille et est désormais utilisé dans plusieurs hôpitaux dans le cadre d'une étude clinique multicentrique portant sur la quantification du niveau d'analgésie à l'aide des paramètres AUCmax et AUCmoy. Cet outil fait également l'objet d'un financement dans le cadre d'un appel d'offre ANR sur le thème « Emergence et maturation de projets de biotechnologie et de technologies pour la santé ». .

Conclusion et perspectives

Dans ce travail nous nous sommes intéressés à la mise en œuvre d'une méthodologie orientée objet pour le traitement temps réel du signal et à son application particulière au domaine du monitorage médical.

Le besoin constant de nouvelles méthodes de monitorage se traduit par l'émergence de nouvelles techniques d'acquisition et de traitement numérique des signaux physiologiques. Or, ces différentes techniques peuvent être considérées sous l'angle d'une implémentation modulaire de méthodes connues. Dans un souci de réutilisation et de capitalisation des développements, nous proposons une méthodologie de conception d'application de traitement du signal basée sur l'utilisation d'une bibliothèque de composants logiciels.

Dans la première partie de ce mémoire, nous avons introduit le concept d'instrumentation de mesure ainsi que les bases de la Programmation Orientée Objet. Conformément aux spécifications du type d'applications visées, nous avons réalisé une architecture logicielle générique constituant un ensemble de classes et de sous-classes permettant, par spécification, la création de composants logiciels pour une application de traitement particulière. Ainsi, cette architecture permet la définition de tous types d'applications de traitement du signal quel que soit le domaine visé.

La seconde partie de ce mémoire présente l'application des concepts et outils précédemment introduits au domaine particulier du monitorage médical. Dans cette partie, nous donnons une brève description de l'architecture matérielle du réseau d'acquisition constituant la source de données pour nos applications. Nous décrivons alors différents types de composants logiciels pour l'acquisition et le traitement des signaux physiologiques. En dialogue avec l'architecture matérielle d'acquisition, cette nouvelle bibliothèque de composants constitue une spécification des classes et sous-classes définies au chapitre 3. La bibliothèque, couplée au réseau d'acquisition, constitue une plateforme de conception spécialisée pour le prototypage rapide de dispositifs temps réel pour le monitorage médical. Enfin, le dernier chapitre présente une application de l'outil développé. Il s'agit d'un dispositif innovant pour l'évaluation du

niveau d'analgésie lors de l'anesthésie générale basé sur l'analyse des variations du rythme cardiaque.

En dehors de cette application particulière, la station d'acquisition et de traitement a été testée dans un grand nombre d'études cliniques ([LOGI 01], [LOGI 03.1], [LOGI 04.1], [LOGI 04.2], [LOGI 06]) et de développements de dispositifs innovants pour le traitement des signaux physiologiques ([LOGI 04], [DEJO 07], [PIRO 07]).

L'exemple d'application du chapitre 6 montre l'intérêt de cet outil pour de telles applications. Son utilisation systématique pour la conception de prototypes d'applications de monitorage permet non seulement de réduire considérablement le temps de conception par réutilisation des travaux antérieurs mais également d'enrichir continuellement la bibliothèque de composants obtenant ainsi un outil évolutif.

Concernant les possibles évolutions de cette station, il serait intéressant de continuer le développement de la structure d'ordonnancement et de contrôle du système. Si pour la plupart des applications visées, la gestion du temps de calcul n'est pas réellement critique, il peut arriver, dans le cas de systèmes de traitement du signal complexes, que le temps nécessaire à l'exécution de toutes les tâches soit supérieur au temps processeur disponible. En l'état actuel, le système d'ordonnancement est statique et déterminé par l'ordre de création des composants. Ainsi, c'est à l'utilisateur de déterminer la priorité de chacun des composants de son application et de tenir compte de cet ordre lors de la création de l'algorithme. Afin de permettre le développement d'applications à temps critique, il convient donc d'optimiser le système d'ordonnancement.

Par ailleurs, l'intégration de nouveaux pilotes pour permettre le dialogue avec d'autres types de périphériques augmenterait considérablement les possibilités d'utilisation de cet outil. En effet, la conception d'une bibliothèque de pilotes logiciels pour tout type de matériel médical (moniteur d'anesthésie, respirateur, pousse seringue...) permettrait la centralisation et la synchronisation de données utiles et complémentaires dans le cadre d'analyses multiparamétriques.

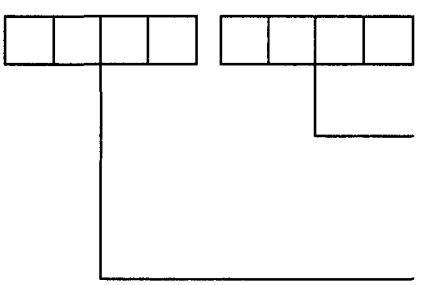
ANNEXES

Annexe1 – Fonctionnement pratique du réseau : protocole

1 - Conventions adoptées :

- Les **modules d'acquisition** comportent des « **voies d'acquisition** »
- Les **modules de restitution** sont vus du réseau comme des « **récepteurs de données** »
- Rappel : le **mot réservé** est un octet qui a été choisi égal à **255 (FFh).**

- L'octet du **mot de commande** est scindé en deux parties de quatre bits :



Les bits de poids faible (LByte) indiquent le nombre d'octets restant à transmettre en argument de la commande.

Les bits de poids fort (HByte) servent au codage de la commande.

- Les deux premiers bits de la commande indiquent sont type :

  - 00 commande de base de la tête de réseau
  - 01 commande de configuration des entrées : voies d'acquisition des boîtiers du réseau
  - 10 commande de configuration des sorties : récepteurs de données du réseau
  - 11 non utilisé

2 - Utilisation des variables dans les micro contrôleurs :

Les variables importantes utilisées par la tête de réseau ou les boîtiers sont définies dans ce chapitre pour permettre une description plus détaillée des commandes dans la suite du document.

- **Tête de réseau :**

- CptAcq (compteur sur 8 bits) : compteur du nombre d'acquisitions pour l'estampillage des séquences. A chaque échantillonnage ce nombre est incrémenté de 1 modulo 255

- MaxTxd (variable de 1 octet) : contient le nombre d'octets maximum transmis en mode de commandes réduites. Initialisé à 4 (suffisant pour les commandes de base du réseau), cette variable peut être définie via le PC.

- **Modules du réseau :**

- AdrRxd (variable de 1 octet) : pour chaque voie d'acquisition, contient le nombre d'octets à recevoir avant d'émettre. Cette variable est connue au début de l'utilisation de la voie.

3 - Commandes du réseau :

Liste des commandes

L'ensemble des commandes définies est listé ci-dessous. Le fonctionnement détaillé de chacune d'entre elles est décrit dans les autres paragraphes.

Hbyte	Lbyte	Commande	octet(s) d'arguments
0000	00xx	top d'échantillonnage (suivie ou non d'une commande)	1 ou 2
0001	0000	initialisation du réseau	0
0010	0001	sélection de la fréquence d'échantillonnage	1
0011	0001	maximum d'octets de commande du mode réduit	1
0100	0100	sélection d'une voie d'acquisition pour utilisation	4
0101	0010	début d'utilisation de la voie sélectionnée	2
0110	0010	fin d'utilisation d'une voie d'acquisition	2
0111	0011	décalage d'une voie d'acquisition en cours d'utilisation	3
1000	0100	sélection d'un récepteur de données pour adressage	4
1001	0001	attribution d'une adresse à un récepteur de données	1
1010	0001	fin d'adressage d'un récepteur de données	1
1011	xxxx	envoie des données à l'adresse d'un récepteur de données	variable
11xx	xxxx	non utilisé	non défini
1111	1111	réservé	non connu

Commandes gérées par la tête de réseau

Elles sont envoyées par la tête de réseau au PC pour qu'il contrôle leurs exécutions et sache à quel moment elles sont transmises aux modules du réseau.

- (00h) Echantillonnage sans commande

octet	signification
11111111	mot réservé
xxxxxxxx	compteur d'échantillons CptAcq
00000000	commande d'échantillonnage

- (01h) Echantillonnage avec le transfert d'une commande réduite

octet	signification
11111111	mot réservé
xxxxxxxx	compteur d'échantillons CptAcq
00000001	commande d'échantillonnage
xxxxxxxx	commande suivie de ses arguments transmis uniquement au réseau

- (xxh) Commande seule en mode transfert de commande évoluées

octet	signification
11111111	mot réservé
xxxxxxxx	compteur d'échantillons CptAcq
xxxxxxxx	commande suivie de ses arguments transmis uniquement au réseau

Commandes de base émises par le PC pour la tête de réseau

- (10h) Initialisation simple du réseau

octet	signification
11111111	mot réservé
00010000	commande d'initialisation simple

Cette commande provoque la réinitialisation complète des modules du réseau ce qui implique que l'ensemble de la configuration est perdue. Elle provoque également la réinitialisation des paramètres de la tête de réseau fixés par le PC.

- (21h) Sélection de la fréquence d'échantillonnage par la tête de réseau

octet	signification
11111111	mot réservé
00100001	commande de sélection de la fréquence d'horloge
xxxxxxxx	octet du diviseur de fréquence d'horloge

La valeur de l'octet du diviseur de fréquence d'horloge dépend directement de la fréquence d'échantillonnage souhaitée :

Fréq. D'éch.		Octet du diviseur de fréq. d'horloge
- 64 Hz	:	70h
- 128 Hz	:	71h
- 256 Hz	:	72h
- 512 Hz	:	73h
- 1024 Hz	:	74h
- 2048 Hz	:	75h
- 4096 Hz	:	76h
- 8192 Hz	:	77h

- (31h) Définition du seuil du mode de transfert des commandes du réseau

octet	signification
11111111	mot réservé
00110001	commande de définition du seuil de transfert des commandes
xxxxxxxx	nombre maximum d'octets pour le transfert de commandes réduites

Commandes de configuration des voies d'acquisition du réseau

La demande d'utilisation d'une voie d'acquisition du réseau s'effectue en deux temps. Cela permet de réduire le nombre d'octets de transfert de la commande puisqu'elle est répartie sur deux tops d'horloge.

- (44h) Sélection d'une voie d'acquisition

Sur le premier top d'horloge, une commande est envoyée au réseau pour sélectionner la voie d'acquisition :

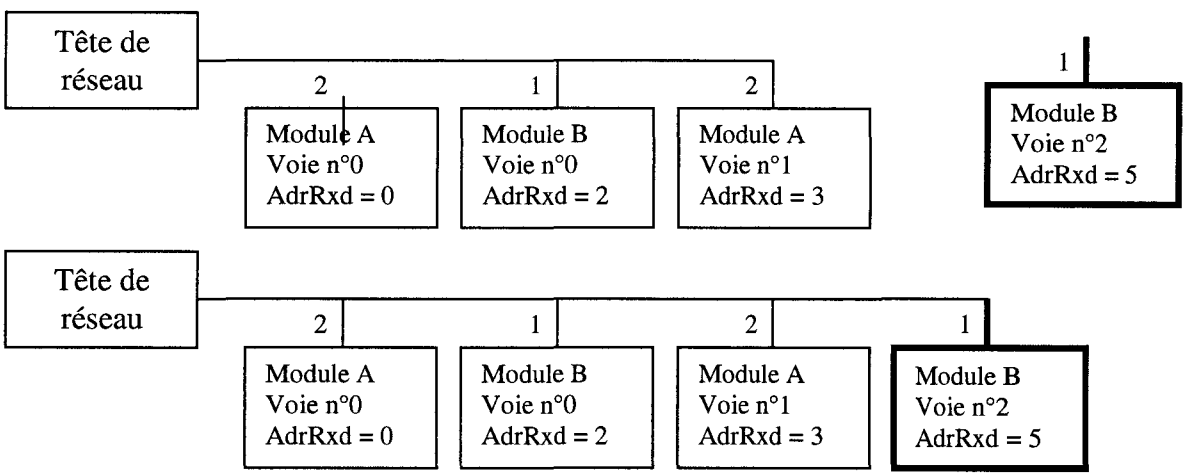
octet	signification
11111111	mot réservé
01000100	commande de sélection d'une voie d'acquisition
xxxxxxxx	type de module
xxxxxxxx	numéro de série du module
xxxxxxxx	(codé sur deux octets)
xxxxxxxx	numéro de la voie d'acquisition à sélectionner

- (52h) Début d'utilisation d'une voie d'acquisition

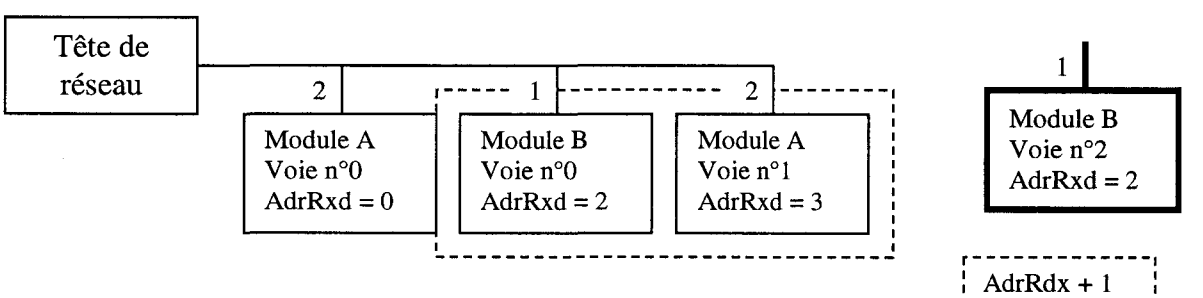
La seconde commande marque le début d'utilisation de la voie sélectionnée :

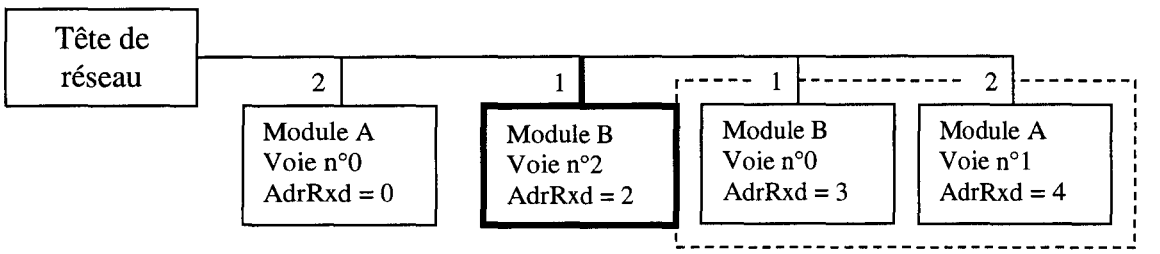
octet	signification
11111111	mot réservé
01010010	commande de début d'utilisation de la voie sélectionnée
xxxxxxxx	nombre d'octets à attendre avant d'émettre
xxxxxxxx	nombre d'octets à émettre

Si le nombre d'octets à attendre avant d'émettre est supérieur à ceux des autres voies déjà connectées alors la nouvelle voie est placée en dernière place.



En revanche, si le numéro d'ordre requis est déjà attribué à une voie en cours d'utilisation, les autres voies connectées doivent se déplacer automatiquement pour réordonner le réseau. Si elles possèdent un nombre d'octets à attendre supérieur ou égal à celui de la nouvelle voie, elles doivent y ajouter le nombre d'octets que la nouvelle voie transmet.





C'est seulement à la séquence suivante que les modifications seront effectives. A la séquence correspondant à l'envoi de la commande, les modules envoient leurs données d'acquisition dans l'état précédent du réseau.

- (62h) Fin d'utilisation d'une voie d'acquisition

octet	signification
11111111	mot réservé
01100010	commande de fin d'utilisation de la voie d'acquisition
xxxxxxxx	nombre d'octets à attendre avant d'émettre
xxxxxxxx	nombre d'octets à émettre

La voie possédant ce nombre d'octets à attendre doit arrêter d'émettre. En pratique, elle prend un nombre d'octets à attendre supérieur au nombre d'octets transmissibles sur le réseau dans l'intervalle d'une période d'échantillonnage. Ainsi, elle ne reçoit jamais le nombre d'octets suffisant pour émettre. Les autre voies utilisées doivent encore une fois se déplacer si elles étaient placées après la voie supprimée : elles doivent maintenant décrémenter le nombre d'octets à attendre du nombre d'octets que la voie supprimée émettait.

- (73h) Décalage d'une voie en cours d'utilisation

octet	signification
11111111	mot réservé

01110011	commande de décalage du numéro d'ordre d'une voie
xxxxxxxx	nombre d'octets à attendre avant d'émettre
xxxxxxxx	nombre d'octets à émettre
xxxxxxxx	nouveau nombre d'octets à attendre avant d'émettre

Une commande de décalage d'une voie peut être considérée comme une **commande de suppression** de la voie à son ancienne place dans l'ordre des émissions défini par son ancien nombre d'octets à attendre avant d'émettre, et une **commande d'ajout** de la même voie (alors sélectionnée) placée en fonction du nouveau nombre d'octets à attendre. C'est ainsi qu'elle est interprétée par les modules, mais en une seule séquence au lieu de trois (suppression, sélection et ajout).

Cette commande est indispensable pour décaler rapidement les voies des modules qui ne répondent plus (modules déconnectés par exemple) et pour ne pas perdre les données des autres voies.

Commandes de configuration des récepteurs de données du réseau

Le principe d'adressage des récepteurs du réseau (électrovannes, potentiomètres numériques...) est beaucoup plus simple que celui d'ordonnancement des voies d'acquisition. En effet, il s'agit simplement d'attribuer de une à quatre adresses (entre 1 et 254) à un récepteur sélectionné pour pouvoir lui envoyer n'importe quelle commande. De plus, on autorisera le PC à attribuer la même adresse à plusieurs récepteurs.

L'absence de limites sur l'adressage ne pose aucun problème de conflit puisque le nombre d'utilisateurs d'une commande n'est pas limité et que les récepteurs n'ont jamais à répondre aux commandes ou à utiliser le réseau pour communiquer et si ce cas d'exception se présente, ce sera au PC de s'assurer de l'unicité de l'adressage.

Par défaut, si les adresses ne sont pas attribuées à un récepteur de données, les variables d'adresses sont initialisées à 0. Cette valeur 0 est aussi mise à profit pour adresser tous les récepteurs présents sur le réseau.

Comme pour l'utilisation d'une voie d'acquisition, deux étapes sont nécessaires pour pouvoir utiliser un récepteur de données :

- (84h) Sélection d'un récepteur de données

octet	signification
11111111	mot réservé
10000100	commande de sélection d'un récepteur de données pour adressage
xxxxxxxx	type de module
xxxxxxxx	numéro de série du module
xxxxxxxx	(codé sur 2 octets)
xxxxxxxx	numéro du récepteur de données à sélectionner

- (91h) Début d'adressage d'un récepteur de données

octet	signification
11111111	mot réservé
10010001	commande d'attribution d'une adresse au récepteur de données sélectionné
xxxxxxxx	adresse attribuée

- (A1h) Fin d'adressage d'un récepteur de données

octet	signification
11111111	mot réservé
10100001	commande de fin d'adressage d'un récepteur de données
xxxxxxxx	adresse attribuée

Cette commande remet à 0 les variables d'adresse de tous les récepteurs de données du réseau ayant l'adresse attribuée.

- (Bxh) Envoi de données à l'adresse d'un récepteur de données

octet	signification
11111111	mot réservé
1011xxxx	commande d'adressage de données à un récepteur de données
xxxxxxxx	adresse attribuée

Le nombre d'octets transmis à la suite de cette commande est indéfini. C'est au PC d'indiquer à la tête de réseau le nombre d'octets de données transférés, en fonction de quoi elle sélectionne le mode de commandes réduites ou avancées.

Annexe 2 : Les différentes classes de composants

1 – Les composants de bases

1.1 – Le composant TFiltre :

Le composant ***TFiltre*** issu de la classe ***TTraitement***, permet d'effectuer un filtrage passe-bas ou passe-haut. Les différents types de filtre utilisés pour la réalisation de ce composant sont présentés en annexe 3.

Ses attributs sont :

- le Type de filtre (Bessel,Butterworth,Tchebychev1 ou Tchebychev3) (TypeF),
- le Type de filtrage (Passe haut ou passe bas) (FType),
- la fréquence de coupure (FC),
- l'ordre du Filtre (Ordre).

Ce composant possède un port d'entrée (InputPortReal) et un port de sortie (OutputPortReal).

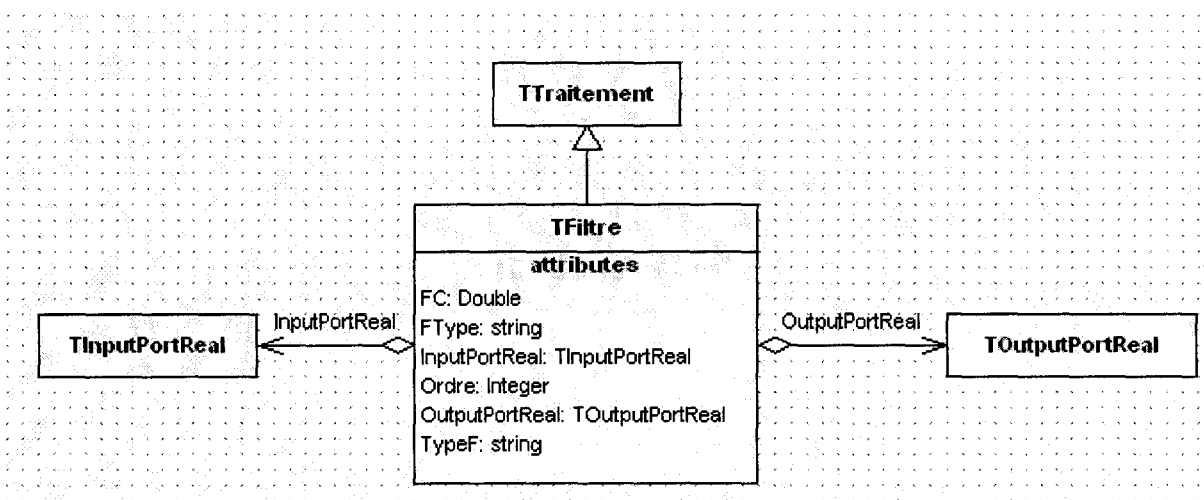


Figure 1 : définition du composant TFilter

1.2 – Le composant TArithmetic1V

Le composant TArithmetic1V permet d'effectuer des calculs mathématiques de type ln, exp, carré, racine... sur un opérande unique.

Il possède un attribut : le choix de l'opération (Operation) ainsi qu'un port d'entrée (InputPortReal) et un port de sortie (OutputPortReal).

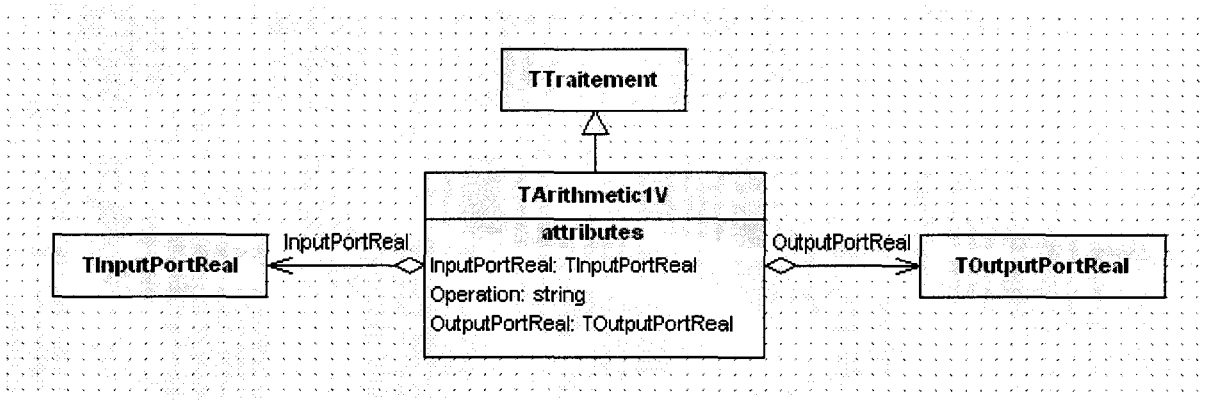


Figure 2 : définition du composant TArithmetic1V

1.3 – Le composant TTriggo1V

Le composant ***TTriggo1V*** permet d'effectuer des calculs trigonométriques de type cos, sin, tan… sur un opérande unique.

Ses attributs sont :

- 1 choix d'opération de sortie (Operation).

Ce composant possède un Port d'entrée (InputPortReal) et un port de sortie (OutputPortReal).

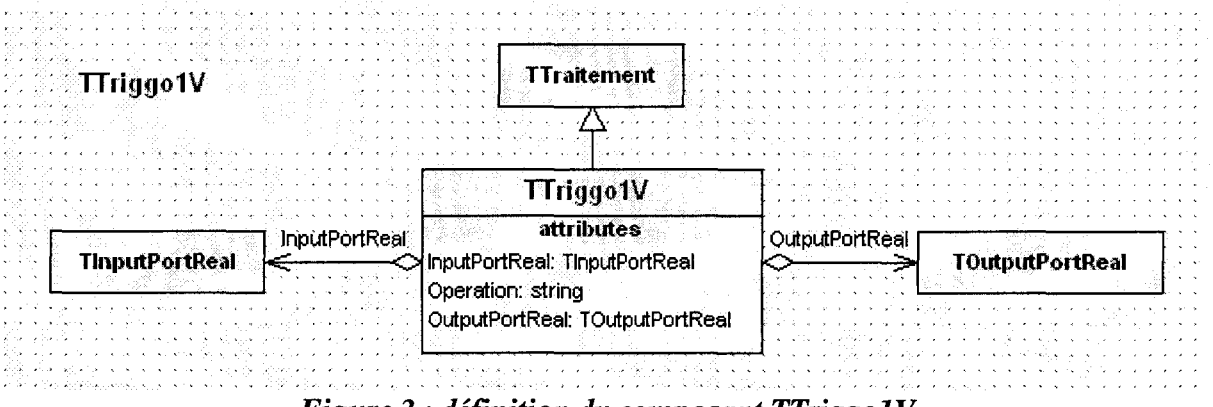


Figure 3 : définition du composant TTriggo1V

1.4 – Le composant TArithmetic2V :

Le composant ***TArithmetic2V*** permet d'effectuer les calculs sur deux opérandes issus de deux ports d'entrées distincts ou d'un port d'entrée et d'une constante.

Ses attributs sont :

- 1 choix d'opérandes (Choix),
- 1 choix d'opération (OpType),
- 1 constante numérique (Ctse).

Les opérations disponibles sont addition, multiplication, soustraction, division, puissance.
Les combinaisons d'opérandes sont voie1_voie2 et voie1_constante.

Ce composant possède donc deux ports d'entrée (InputPortReal1 et InputPortReal2) et un port de sortie (OutputPortReal).

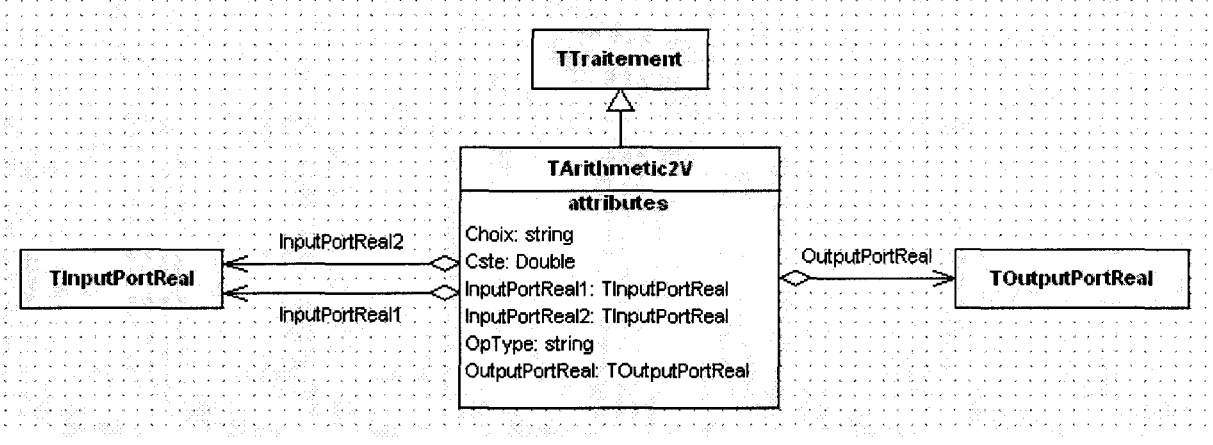


Figure 4 : Définition du composant TArithmetic2V

1.5 – Le composant TCombinatoire :

Le composant TCombinatoire permet d'effectuer des calculs logiques sur des signaux issus de composants de détection.

Ses attributs sont :

- 1 opérateur logique (Operateur).

Les différents opérateur logiques sont : OR, NOR, AND, NAND, XOR, NXOR

Ce composant possède deux ports d'entrée (InputPortBool1 et InputPortBool2) et un port de sortie (OutputPortBool).

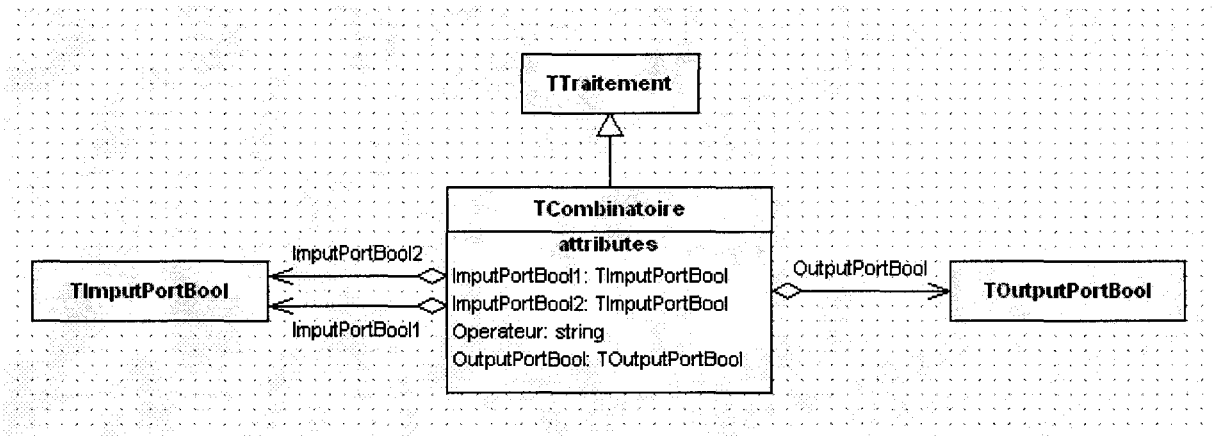


Figure 5 : définition du composant TCombinatoire

1.6 – Le composant TReechantil :

Le composant ***TReechantil*** permet de ré échantillonner un signal.

Ses attributs sont :

- 1 fréquence d'échantillonnage (Freq).

Ce composant possède un port d'entrée (InputPortReal) et un port de sortie (OutputPortReal).

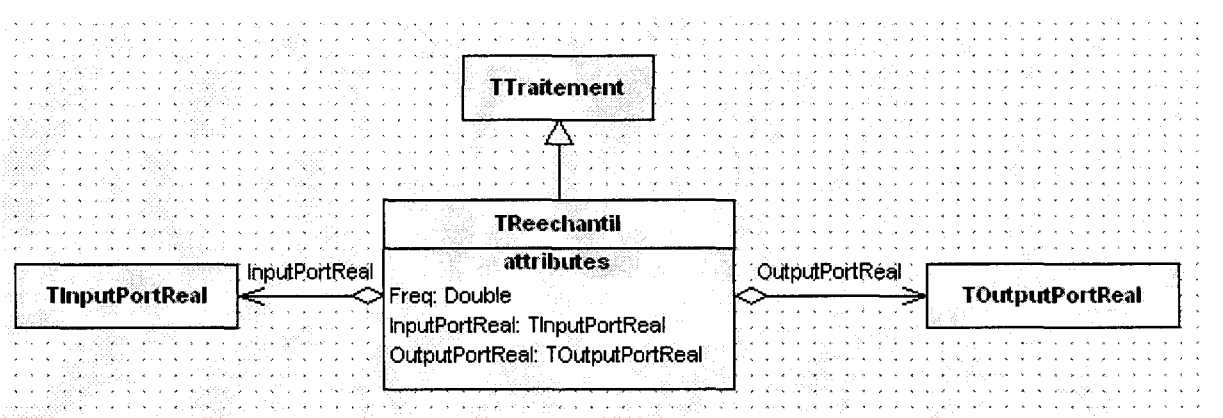


Figure 6 : définition du composant TReechantil

2 – Les composants de détection :

2.1 – Le composant TComparateur :

Le composant ***TComparateur*** permet d'effectuer des comparaisons sur des données issues de deux ports d'entrée distincts ou d'un port d'entrée et d'une constante.

Ses attributs sont :

- 1 choix d'opérande (Choix),
- 1 choix d'opération (OpType),
- 1 constante numérique (Ctse),
- 1 période réfractaire (PRef),
- 1 intervalle de tolérance (Intervalle),
- 1 type de sortie (TrigTyp).

Les différentes opérations sont >, >=, <, <=, =.
Les différents opérandes sont voie1_voie2 et voie1_constante. En fonction du choix de l'opérande, le composant calculera le résultat sur 2 voies ou sur 1 voie et la constante.
Les différents types de sortie sont impulsion ou échelon.

Ce composant possède deux ports d'entrée (InputPortReal1 et InputPortReal2) et un port de sortie (OutputPortBool).

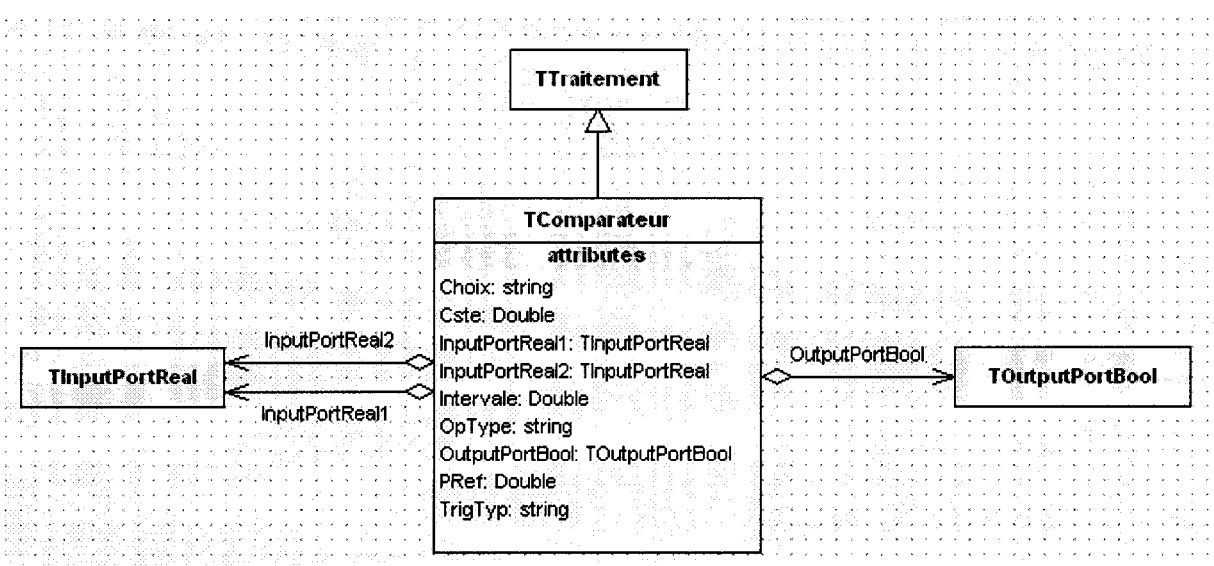


Figure 7 : définition du composant TComparateur

3 – Les composants de mesures cycliques

3.1 - Le composant TBloqueur :

Le composant TBloqueur permet de calculer, sur un signal d'entrée plusieurs paramètres tels que le min, le max, la moyenne… entre deux impulsions d'un composant de détection.

Ses attributs sont :

- 1 type d'opération (Operation)

Les différents types d'opérations sont : valeur instantanée, min, max, valeur crête à crête, moyenne, médiane, écart type, écart moyen, somme, intégrale, temps ou fréquence.

Ce composant possède deux Port d'entrée, un port représentant le signal à analyser (InputPortReal) et l'autre le signal de détection (InputPortBool) et un port de sortie (OutputPortReal).

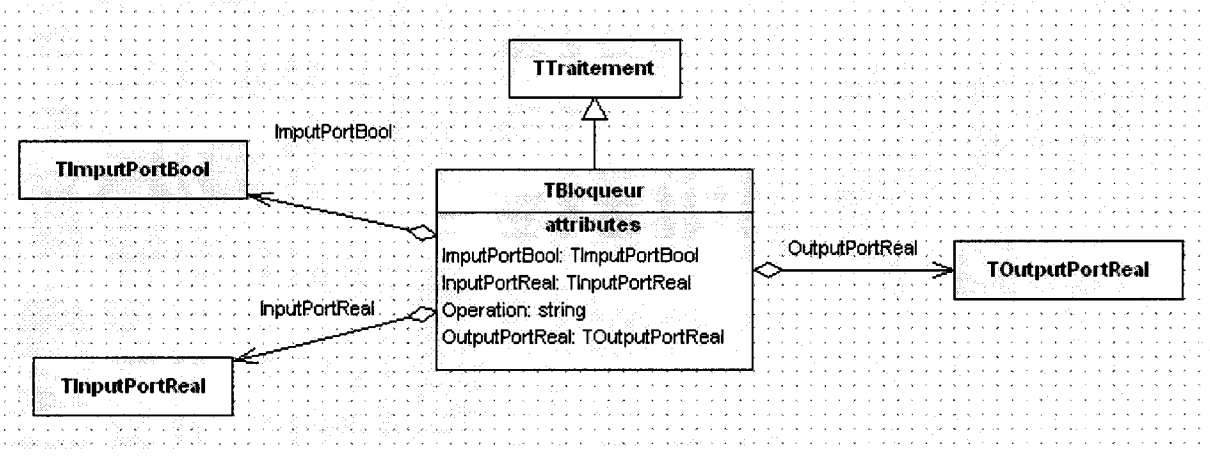


Figure 8 : définition du composant TBloqueur

4 – Les composants de mesures sur fenêtre glissante

4.1 - Le composant TMesFenGlis :

Le composant TMesFenGlis permet de calculer plusieurs paramètres tel que le min, le max, la moyenne… sur une fenêtre glissante.

Ses attributs sont :

- 1 type d'opération (Operation),
- la durée de la fenêtre (Duree),
- un booleen indiquant si on effectue la suppression de la valeur moyenne (SupValMoy),
- un booleen indiquant si on effectue la normalisation (Normalise).

Les différents types d'opérations sont : valeur instantanée, min, max, valeur crête à crête, moyenne, médiane, écart type, écart moyen, somme ou intégrale.

Ce composant possède un port d'entrée (InputPortReal) et un port de sortie (OutputPortReal).

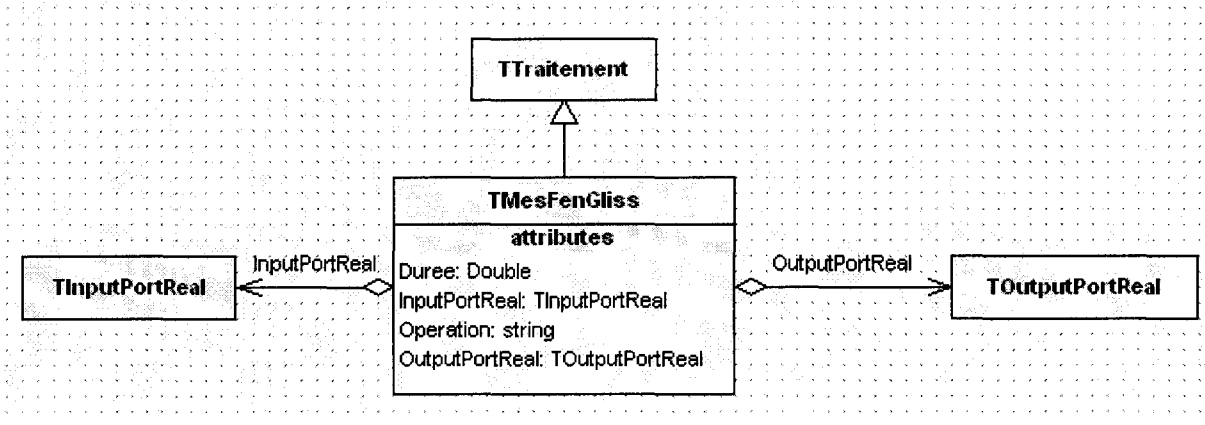


Figure 9 : définition du composant TMesFenGliss

5 – Les composants de traitements avancés

5.1 – Le composant TFenetrage :

Ce composant permet de bufférisers les données en entrée et d'y appliquer une fenêtre de pondération. Les fenêtres disponibles sont les fenêtres rectangulaire, triangulaire, de Blackman, de Hamming, de Hanning et de Kaiser-Bessel (cf. Annexe 4). En plus du type de fenêtre, l'utilisateur a la possibilité de régler d'autres paramètres : la largeur de la fenêtre

d'analyse et le temps de rafraîchissement de cette fenêtre. On peut aussi supprimer la valeur moyenne calculée sur la largeur de la fenêtre ou normaliser le signal.

Ses attributs sont :

- le type de fenêtre (FType),
- la durée de la fenêtre (Duree),
- le temps de rafraîchissement (Rafraichissement),
- un booléen indiquant si on supprime ou non la valeur moyenne (SupValMoy),
- un Booléen indiquant si on normalise ou nom (Normalise).

Ce composant possède un port d'entrée (InputPortReal) et un port de sortie (OutputPortBuf).

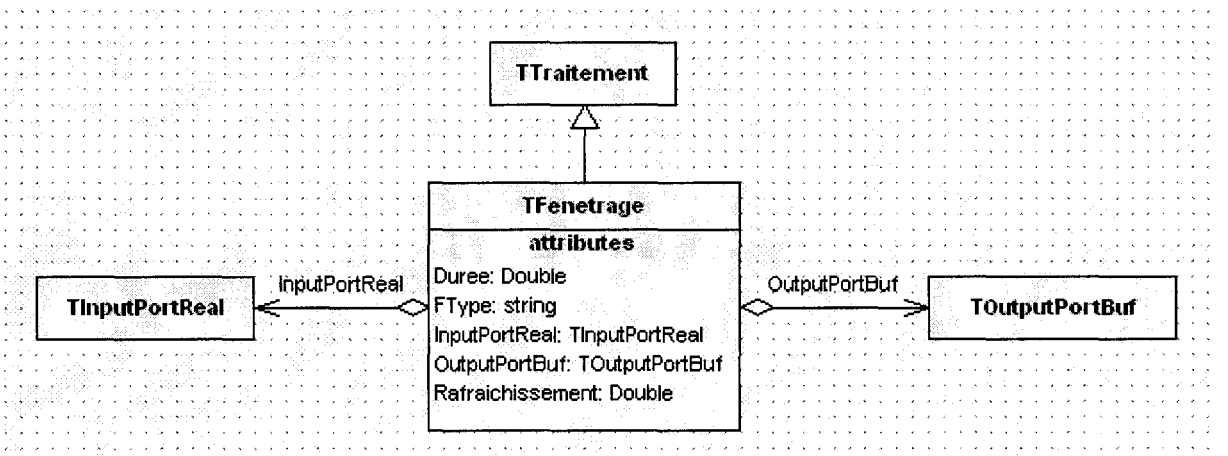


Figure 10 : définition du composant TFenetrage

5.2 - Le composant TFFT :

Le composant TFFT permet d'effectuer une transformation de Fourier rapide sur un signal fenêtré. Ce composant délivre, en sortie, plusieurs tableaux de données correspondant au signal réel, au signal imaginaire et aux spectres d'amplitude et de phase du signal d'entrée. L'algorithme utilisé pour le calcul de la FFT est décrit en annexe 5.

Ses attributs sont :

- la fréquence d'échantillonnage (SampleRate),
- le nombre de point (NbPts),

- le rafraichissement (Raif).

Ce composant possède un port d'entrée (InputPortBuf) et quatre ports de sortie (OutputPortBuf).

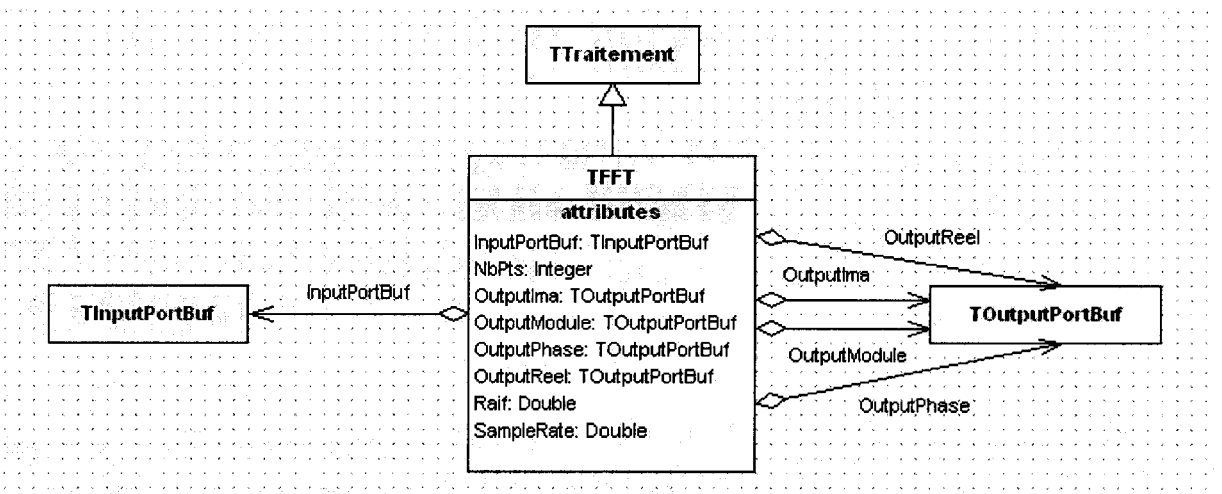


Figure 11 : définition du composant TFFT

5.3 – Le composant TOndelette :

Le composant TOndelette réalise la transformée en ondelette discrète dyadique (DWT : Discrete Wavelet Transform) de type Daubechie 4, 12 ou 20 ou Haar d'un signal quelconque (annexe 6).

Ses attributs sont :

- le nombre de points dans la fenêtre (NbPts),
- le nombre d'itérations (NbItération),
- le type d'ondelette (Ondelette),
- la période de rafraîchissement (Rafr).

Ce composant possède un port d'entrée (InputPortBuf) et un port de sortie (OutputPortMat).

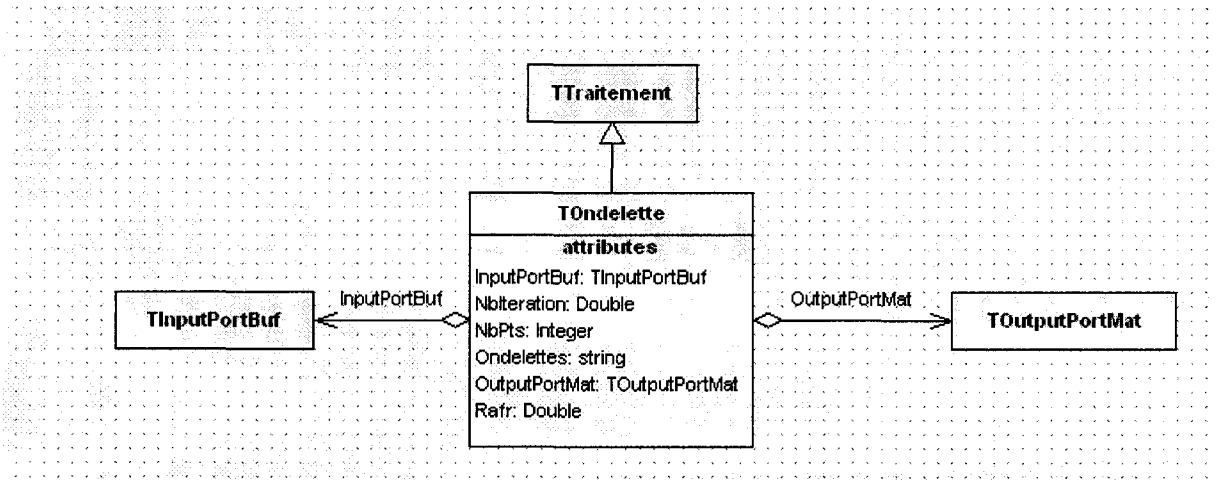


Figure 12 : définition du composant TOndelette

5.4 – Le composant TOndelettelnv :

Le composant TOndeletteInv réalise la transformée en ondelette inverse d'un signal issu d'un composant TOndelette.

Ses attributs sont :

- La liste des itérations utilisées pour la reconstruction (SelectIte).

Ce composant possède un port d'entrée (InputPortMat) et un port de sortie (OutputPortBuf).

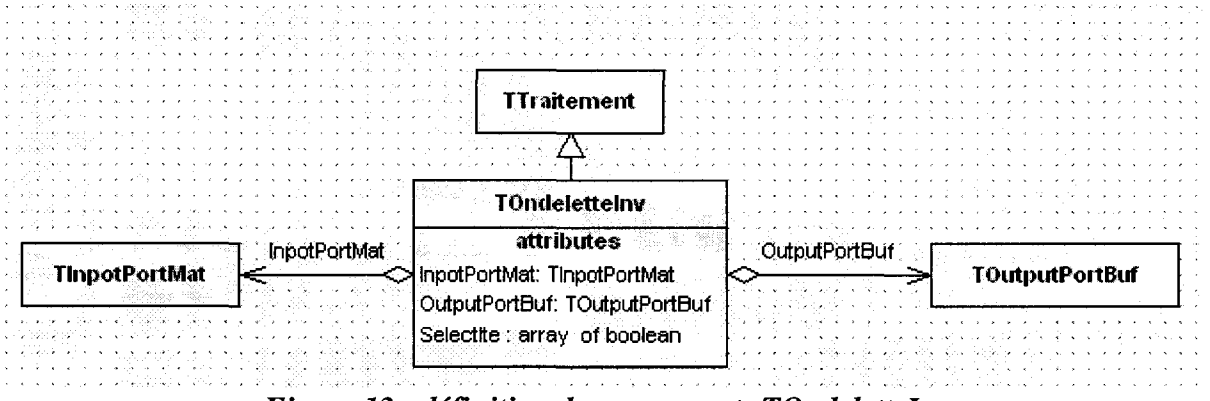


Figure 13 : définition du composants TOndeletteInv

6 – Les composants spécilisés

6.1 – Le composant TECGTrig :

Le composant ***TECGTrig*** permet de réaliser la détection des ondes R de l'ECG selon l'algorithme de détection présenté au chapitre 1.

Ses attributs sont :

- un seuil de détection (Seuil),
- une période réfractaire (PRef).

Ce composant possède un port d'entrée (InputPortReal) et un port de sortie (OutputPortBool).

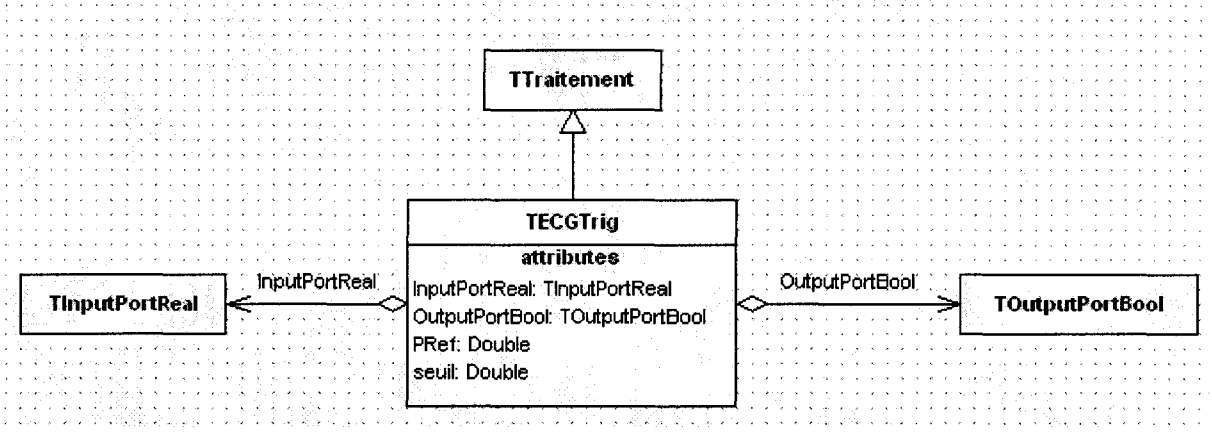


Figure 14 : définition du composant TECGTrig

6.1 – Le composant TFiltreRR

Le composant ***TFiltreRR*** permet de réaliser le filtrage de la série RR selon l'algorithme décrit en annexe 7.

Ce composant possède un port d'entrée (InputPortReal) et un port de sortie (OutputPortReal).

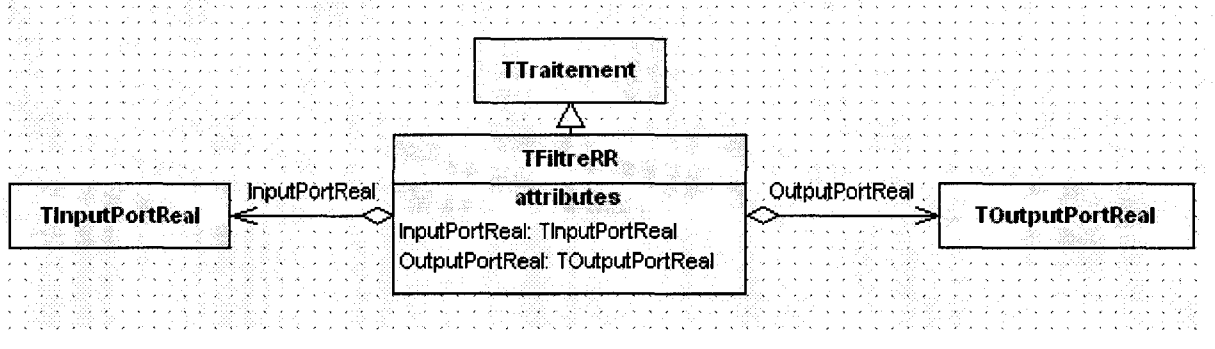


Figure 15 : définition du composant TFiltreRR

Annexe 3 : Filtres numériques

1 – Expression des filtres

Les filtres passe-haut et passe-bande se déduisant facilement des passe-bas, nous donnerons uniquement les paramètres pour ces derniers.

La forme générale de la fonction de transfert d'un filtre passe-bas d'ordre n de fréquence de coupure f_c et de gain 1 dans la bande passante est la suivante:

$$H(s) = \frac{a_0}{a_n s^n + a_{n-1} s^{n-1} + ... + a_1 s + a_0} \quad \text{, où} \quad s = \frac{j\omega}{\omega_c} = \frac{p}{\omega_c} \quad \text{et} \quad \omega_c = 2\pi f_c$$

Calculons maintenant les coefficients a_i des 3 types de filtres qui nous intéressent.

1.1 Filtres de Bessel :

Le dénominateur de H(s) est égal au polynôme de Bessel qui se calcule grâce à la récurrence suivante :

$B_0(s) = 1$

$B_1(s) = s + 1$

$B_n(s) = (2n-1)B_{n-1}(s) + s^2 B_{n-2}(s)$

Ce qui nous donne les coefficients a_i des filtres d'ordre n de Bessel :

n	a0	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
1	1	1									
2	3	3	1								
3	15	15	6	1							
4	105	105	45	10	1						
5	945	945	420	105	15	1					
6	10395	10395	4725	1260	210	21	1				
7	135135	135135	62370	17325	3150	378	28	1			

8	2027025	2027025	945945	270270	51975	6930	630	36	1		
9	34459425	34459425	16216200	4729725	945945	135135	13860	990	45	1	
10	654729075	654729075	310134825	91891800	18918900	2837835	315315	25740	1485	55	1

1.2 - Filtres de Butterworth :

$$H(s) = \frac{1}{\prod_{k=1}^{n}(s - s_k)}$$

avec $s_k = \exp[j\pi(\frac{1}{2} + (\frac{2k-1}{2n}))]$ $\qquad k = 1,2...,n$

Voici un petit programme Matlab permettant de calculer le dénominateur D de H(s) en fonction de l'ordre n choisi :

```
» n= ;
» D=[1,-exp(j*pi*(1/2+1/(2*n)))];
» for I=2:n,
» D=conv(D,[1,-exp(j*pi*(1/2+(2*I-1)/(2*n)))]);
» end
```

Ce qui nous donne les coefficients a_i des filtres d'ordre n de Butterworth :

n	a0	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
1	1	1									
2	1	1.4142	1								
3	1	2	2	1							
4	1	2.6131	3.4142	2.6131	1						
5	1					1					

		3.2361	5.2361	5.2361	3.2361						
6	1	3.8637	7.4641	9.1416	7.4641	3.8637	1				
7	1	4.4940	10.0978	14.5918	14.5918	10.0978	4.4940	1			
8	1	5.1258	13.1371	21.8462	25.6884	21.8462	13.1371	5.1258	1		
9	1	5.7588	16.5817	31.1634	41.9864	41.9864	31.1634	16.5817	5.7588	1	
10	1	6.3925	20.4317	42.8021	64.8824	74.2334	64.8824	42.8021	20.4317	6.3925	1

1.3 - Filtres de Tchebychev :

Un paramètre supplémentaire est à prendre en compte : l'atténuation A voulue à la fréquence de coupure (en dB). Nous prendrons 2 cas : A=1dB et A=3dB

Le paramètre ε (taux d'ondulation) représente cette atténuation et vaut $\sqrt{10^{A/10}-1}$

Les pôles de H(s) sont les suivants :

$$H(s)=\frac{1}{\prod_{k=1}^{n}(s-s_k)}$$

où :

$$s_k=-\sinh(a).\sin(\frac{2k-1}{2n}\pi)+i.\cosh(a).\cos(\frac{2k-1}{2n}\pi) \qquad k=1,2...,n$$

et :

$$a=\frac{1}{n}.\sinh^{-1}\frac{1}{\varepsilon}$$

Voici un programme Matlab permettant de calculer le dénominateur D de la fonction de transfert en fonction de l'atténuation et de l'ordre choisis :

```
»att= ;          //atténuation en dB
»n= ;                    //ordre du filtre

»epsilon=sqrt(10^(att/10)-1);
»a=1/n*asinh(1/epsilon);
»D=[1,sinh(a)*sin(pi/(2*n))-i*cosh(a)*cos(pi/(2*n))];
»for I=2:n,
»D=conv(D,[1,sinh(a)*sin((2*I-1)*pi/(2*n))-i*cosh(a)*cos((2*I-1)*pi/(2*n))]);
»end
```

Ce qui nous donne les coefficients a_i des filtres d'ordre n de Tchebychev :

Cas A=1dB :

n	a0	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
1	1.9652	1									
2	1.1025	1.0977	1								
3	0.4913	1.2384	0.9883	1							
4	0.2756	0.7426	1.4539	0.9528	1						
5	0.1228	0.5805	0.9744	1.6888	0.9368	1					
6	0.0689	0.3071	0.9393	1.2021	1.9308	0.9283	1				
7	0.0307	0.2137	0.5486	1.3575	1.4288	2.1761	0.9231	1			
8	0.0172	0.1073	0.4478	0.8468	1.8369	1.6552	2.4230	0.9198	1		
9	0.0077	0.0706	0.2442	0.7863	1.2016	2.3781	1.8815	2.6709	0.9175	1	
10	0.0043	0.0345	0.1825	0.4554	1.2445	1.6130	2.9815	2.1079	2.9195	0.9159	1

Cas A=3dB :

n	a0	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
1	1.0024	1									
2	0.7079	0.6449	1								
3	0.2506	0.9283	0.5972	1							
4	0.1770	0.4048	1.1691	0.5816	1						
5	0.0626	0.4080	0.5489	1.4150	0.5745	1					
6	0.0442	0.1634	0.6991	0.6906	1.6628	0.5707	1				
7	0.0157	0.1462	0.3000	1.0518	0.8314	1.9116	0.5684	1			
8	0.0111	0.0565	0.3208	0.4719	1.4667	0.9719	2.1607	0.5669	1		
9	0.0039	0.0476	0.1314	0.5835	0.6789	1.9439	1.1123	2.4101	0.5659	1	
10	0.0028	0.0180	0.1278	0.2492	0.9499	0.9211	2.4834	1.2526	2.6597	0.5652	1

2 – Transformation en Z

Pour cela, 3 transformées de base ont été utiles (Te est la période d'échantillonnage) :

$$\frac{1}{p+a} \Leftrightarrow \frac{1}{1-e^{-aTe}\cdot z^{-1}} \tag{1}$$

$$\frac{1}{p^2+2\sigma p+\rho^2} \Leftrightarrow \frac{\frac{1}{\omega}\cdot e^{-\sigma Te}\cdot \sin \omega Te\cdot z^{-1}}{D(z)} \tag{2}$$

$$\frac{p+\alpha}{p^2+2\sigma p+\rho^2} \Leftrightarrow \frac{K\cdot\cos\theta - K\cdot e^{-\sigma Te}\cdot\cos(\omega Te+\theta)\cdot z^{-1}}{D(z)} \tag{3}$$

avec $\omega = \sqrt{\rho^2-\sigma^2}$

$$K = \sqrt{1 + \left(\frac{\alpha - \sigma}{\omega}\right)^2}$$

$$\theta = \tan^{-1}\left(\frac{\alpha - \sigma}{\omega}\right)$$

$$D(z) = 1 - 2e^{-\sigma Te} \cdot \cos \omega Te \cdot z^{-1} + e^{-2\sigma Te} \cdot z^{-2}$$

2.1 - Décomposition en éléments simples

Pour cela on utilise la fonction residue de Matlab dans le programme suivant :

```
»n=;                          //ordre du filtre (à partir de 3)
»h=[ ];                       //coefficients a_i du dénominateur de H(s) en
                              //puissance décroissante de s (cf I)
»[r,p,k]=residue(h(n+1),h)    //décomposition en éléments simples de H(s)
```

La fonction residue donne les pôles complexes conjugués. Il faut donc les regrouper afin d'obtenir des termes du 2ème ordre et de pouvoir utiliser la transformée usuelle (3).

```
»den=poly([p(1),p(2)]);           //qui utilise le résultat p de residue
»a=conv(r(1),[1,-p(2)]);          //qui utilise les résultats p et r de residue

»gain=2*real(a(1))
»alpha=(2*real(a(2)))/gain1  //correspond à α de la transformée (3)
»sigma=den(2)/2                   //correspond à σ
»ro=den(3)                        //correspond à ρ²
```

2.2 - Obtention de la formule de récurrence

On effectue la transformée en z de chaque élément simple du 1er ou 2ème ordre avec l'aide des 3 transformées usuelles puis on réduit au même dénominateur en z.

En pratique :
pour l'ordre 1, on utilise directement la transformée (1)
pour l'ordre 2, on utilise directement la transformée (2)
pour les ordres 4,6,8,10, on utilise plusieurs fois la transformée (3)
pour les ordres 3,5,7,9, on utilise la transformée (1) et plusieurs fois la transformée (3)

Ensuite, sachant que $H(z) = \frac{S(z)}{E(z)}$ et que $S_n \cdot z^{-k} = S_{n-k}$, on peut donner S_n en fonction des E_{n-k} et des S_{n-k} et ainsi obtenir une équation de récurrence du filtre que l'on peut implémenter facilement dans le programme informatique.

Afin d'illustrer le raisonnement ci-dessus, voici un exemple complet pour le filtre d'ordre 3 de Bessel :

$$H(s) = \frac{15}{s^3 + 6s^2 + 15s + 15} = (4.5298)\frac{1}{s + 2.3222} + (-4.5298)\frac{s + 1.3556}{s^2 + 2 \times 1.8389 \cdot s + 6.4594}$$

d'où :

$$H(p) = (4.5298 \cdot \omega_c)\frac{1}{p + 2.3222 \cdot \omega_c} + (-4.5298 \cdot \omega_c)\frac{p + 1.3556 \cdot \omega_c}{p^2 + 2 \times 1.8389 \cdot \omega_c \cdot p + 6.5494 \cdot \omega_c^2}$$

$$H(p) = (g0 \cdot w_c)\frac{1}{p + a0 \cdot \omega_c} + (g1 \cdot \omega_c)\frac{p + \alpha_1 \cdot \omega_c}{p^2 + 2\sigma_1 \cdot \omega_c \cdot p + \rho_1 \cdot \omega_c^2}$$

d'où, avec les transformées usuelles (1) et (3) :

$$H(z) = G0 \cdot \frac{1}{1 + A0 \cdot z^{-1}} + G1 \cdot \frac{A1 + B1 \cdot z^{-1}}{1 + C1 \cdot z^{-1} + D1 \cdot z^{-2}}$$

avec :

$$G0 = g0 \cdot \omega_c$$
$$A0 = -\exp(-a0 \cdot \omega_c \cdot Te)$$
$$G1 = g1 \cdot \omega_c$$
$$A1 = K_1 \cdot \cos\theta_1$$
$$B1 = -K_1 \cdot \cos(\omega_1 \cdot \omega_c \cdot Te + \theta_1) \cdot \exp(-\sigma_1 \cdot \omega_c \cdot Te)$$
$$C1 = -2 \cdot \exp(-\sigma_1 \cdot \omega_c \cdot Te\ ) \cdot \cos(\omega_1 \cdot \omega_c \cdot Te)$$
$$D1 = \exp(-2 \cdot \sigma_1 \cdot \omega_c \cdot Te)$$

et :

$$\omega_1 = \sqrt{\rho_1 - \sigma_1^2}$$
$$K_1 = \sqrt{1 + \left(\frac{\alpha_1 - \sigma_1}{\omega_1}\right)^2}$$
$$\theta_1 = \tan^{-1}\left(\frac{\alpha_1 - \sigma_1}{\omega_1}\right)$$

puis, en mettant au même dénominateur , on obtient :

$$H(z) = \frac{A + B \cdot z^{-1} + C \cdot z^{-2}}{1 + D \cdot z^{-1} + E \cdot z^{-2} + F \cdot z^{-3}} = \frac{S(z)}{E(z)} = \frac{S_n}{E_n}$$

avec :

$$A = G0 + G1A1$$
$$B = G0C1 + G1(A0A1 + B1)$$
$$C = A0G1B1 + G0D1$$
$$D = A0 + C1$$
$$E = A0C1 + D1$$
$$F = A0D1$$

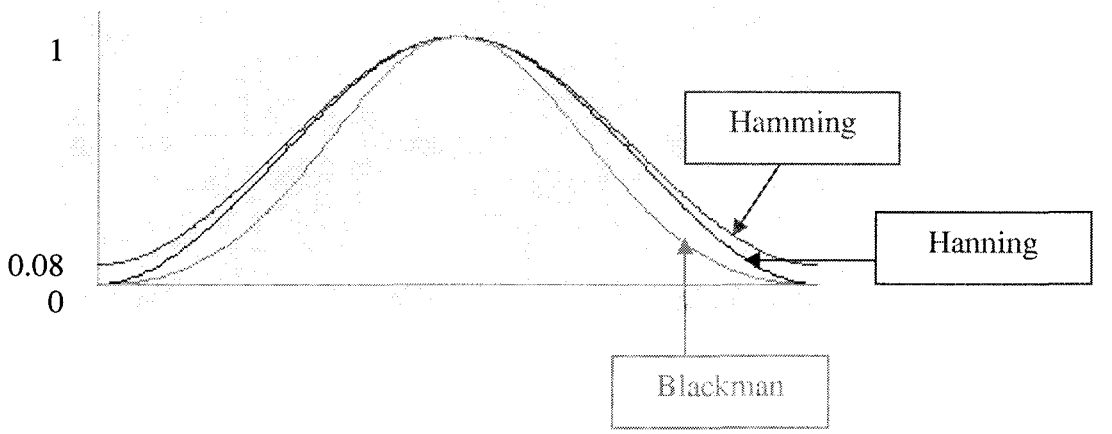
et enfin :

$$S_n = A \cdot E_n + B \cdot E_{n-1} + C \cdot E_{n-2} - D \cdot S_{n-1} - E \cdot S_{n-2} - F \cdot S_{n-3}$$

Annexe 4 : Allures des différents types de fenêtres de pondération

Les différentes fenêtres de pondération disponibles sont les fenêtres rectangulaires, triangulaires, de Blackman, de Hamming, de Hanning et de Kaiser-Bessel.

Hamming, Hanning, Blackman :



Le fenêtrage de Kaiser-Bessel est plus particulier car il dépend d'un paramètre Alpha.

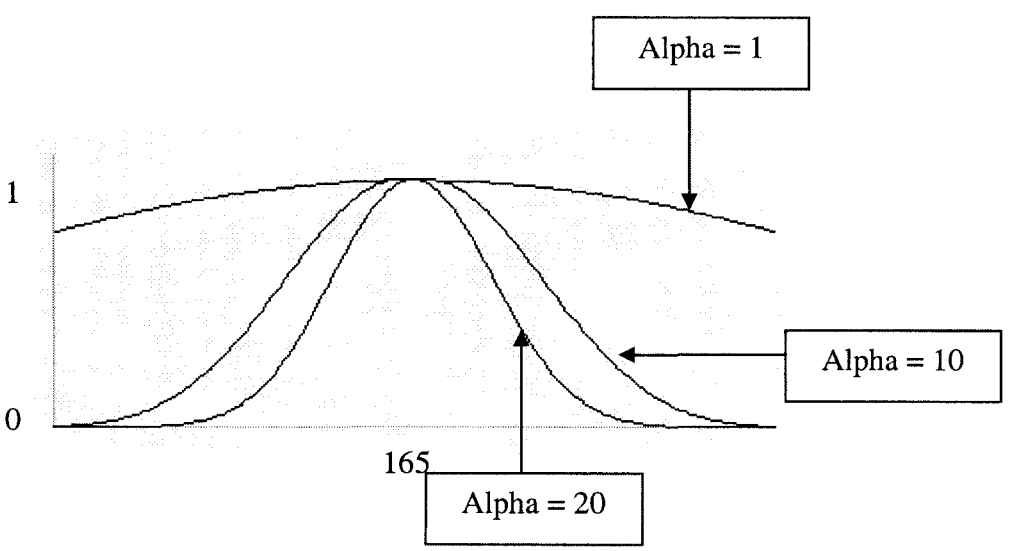
Si Alpha = 0, on obtient une fenêtre rectangulaire.

Si Alpha = 5, on obtient une fenêtre proche de Hamming.

Si Alpha = 6, on obtient une fenêtre proche de Hanning.

Si Alpha = 8.6, on obtient une fenêtre proche de Blackman.

Plus Alpha augmente et plus la fenêtre se resserre comme illustré ci-dessous.



Annexe 5 : Transformée de Fourier Rapide

La transformée de Fourier rapide est simplement un algorithme permettant de réduire le nombre d'opérations pour calculer la DFT (Discrete Fourier Transform).
Rappelons la relation permettant de calculer la DFT (on omet volontairement la pondération 1/N) :

$$S_k = \sum_{n=0}^{N-1} s_n \cdot e^{-j2\pi\frac{kn}{N}} \qquad (1)$$

Par conséquent, les opérations à effectuer pour obtenir les N valeurs de la DFT sont :

N^2 multiplications complexes
N.(N-1) additions complexes

L'algorithme de FFT le plus connu est celui de Cooley-Tukey, également appelé algorithme de réduction à base 2 dans le domaine temporel. Cet algorithme requiert un nombre d'échantillons N qui soit une puissance de 2. Par exemple : N=128, 256, 4096, etc. D'autres algorithmes FFT existent qui requièrent d'autres exigences, mais nous ne les considérerons pas ici.

Dans le cas d'une FFT selon l'algorithme de Cooley-Tukey, le nombre d'opérations est considérablement réduit : $\frac{N}{2}\log_2 N$ multiplications complexes au lieu de N^2 pour la DFT.
Voilà comment on optimise le calcul de le DFT :

Posons :

$$W_N^{nk} = e^{-j2\pi\frac{kn}{N}} \qquad (2)$$

L'équation (1) peut alors être réécrite :

$$S_k = \sum_{n=0}^{N-1} s_k \cdot W_N^{nk} \tag{3}$$

L'équation (2) a les deux caractéristiques suivantes :

$$W_N^{2nk} = e^{-j2\pi \frac{kn}{N/2}} = W_{\frac{N}{2}}^{nk} \tag{4}$$

$$W_N^{(nk+\frac{N}{2})} = e^{-j2\pi \frac{kn+N/2}{N}} = -W_N^{nk} \qquad \text{(pour } n < N/2\text{)} \tag{5}$$

Prenons un exemple avec 8 échantillons. La DFT se présente ainsi :

$$S_0 = s_0 \cdot e^{-j0} + s_1 \cdot e^{-j0} + s_2 \cdot e^{-j0} + s_3 \cdot e^{-j0} + s_4 \cdot e^{-j0} + s_5 \cdot e^{-j0} + s_6 \cdot e^{-j0} + s_7 \cdot e^{-j0}$$

$$S_1 = s_0 \cdot e^{-j0} + s_1 \cdot e^{-j\frac{\pi}{4}} + s_2 \cdot e^{-j\frac{2\pi}{4}} + s_3 \cdot e^{-j\frac{3\pi}{4}} + s_4 \cdot e^{-j\frac{4\pi}{4}} + s_5 \cdot e^{-j\frac{5\pi}{4}} + s_6 \cdot e^{-j\frac{6\pi}{4}} + s_7 \cdot e^{-j\frac{7\pi}{4}}$$

$$S_2 = s_0 \cdot e^{-j0} + s_1 \cdot e^{-j\frac{2\pi}{4}} + s_2 \cdot e^{-j\frac{4\pi}{4}} + s_3 \cdot e^{-j\frac{6\pi}{4}} + s_4 \cdot e^{-j\frac{8\pi}{4}} + s_5 \cdot e^{-j\frac{10\pi}{4}} + s_6 \cdot e^{-j\frac{12\pi}{4}} + s_7 \cdot e^{-j\frac{14\pi}{4}}$$

$$S_3 = s_0 \cdot e^{-j0} + s_1 \cdot e^{-j\frac{3\pi}{4}} + s_2 \cdot e^{-j\frac{6\pi}{4}} + s_3 \cdot e^{-j\frac{9\pi}{4}} + s_4 \cdot e^{-j\frac{12\pi}{4}} + s_5 \cdot e^{-j\frac{15\pi}{4}} + s_6 \cdot e^{-j\frac{18\pi}{4}} + s_7 \cdot e^{-j\frac{21\pi}{4}}$$

$$S_4 = s_0 \cdot e^{-j0} + s_1 \cdot e^{-j\frac{4\pi}{4}} + s_2 \cdot e^{-j\frac{8\pi}{4}} + s_3 \cdot e^{-j\frac{12\pi}{4}} + s_4 \cdot e^{-j\frac{16\pi}{4}} + s_5 \cdot e^{-j\frac{20\pi}{4}} + s_6 \cdot e^{-j\frac{24\pi}{4}} + s_7 \cdot e^{-j\frac{28\pi}{4}}$$

$$S_5 = s_0 \cdot e^{-j0} + s_1 \cdot e^{-j\frac{5\pi}{4}} + s_2 \cdot e^{-j\frac{10\pi}{4}} + s_3 \cdot e^{-j\frac{15\pi}{4}} + s_4 \cdot e^{-j\frac{20\pi}{4}} + s_5 \cdot e^{-j\frac{25\pi}{4}} + s_6 \cdot e^{-j\frac{30\pi}{4}} + s_7 \cdot e^{-j\frac{35\pi}{4}}$$

$$S_6 = s_0 \cdot e^{-j0} + s_1 \cdot e^{-j\frac{6\pi}{4}} + s_2 \cdot e^{-j\frac{12\pi}{4}} + s_3 \cdot e^{-j\frac{18\pi}{4}} + s_4 \cdot e^{-j\frac{24\pi}{4}} + s_5 \cdot e^{-j\frac{30\pi}{4}} + s_6 \cdot e^{-j\frac{36\pi}{4}} + s_7 \cdot e^{-j\frac{42\pi}{4}}$$

$$S_7 = s_0 \cdot e^{-j0} + s_1 \cdot e^{-j\frac{7\pi}{4}} + s_2 \cdot e^{-j\frac{14\pi}{4}} + s_3 \cdot e^{-j\frac{21\pi}{4}} + s_4 \cdot e^{-j\frac{28\pi}{4}} + s_5 \cdot e^{-j\frac{35\pi}{4}} + s_6 \cdot e^{-j\frac{42\pi}{4}} + s_7 \cdot e^{-j\frac{49\pi}{4}}$$

En séparant les échantillons pairs et impairs et en factorisant les nombres impairs, on peut mettre en évidence des similitudes entre les opérations pratiquées sur ces échantillons. Prenons comme exemple la ligne S_1 :

$$S_1 = s_0 \cdot e^{-j0} + s_2 \cdot e^{-j\frac{2\pi}{4}} + s_4 \cdot e^{-j\frac{4\pi}{4}} + s_6 \cdot e^{-j\frac{6\pi}{4}} + \left(s_1 \cdot e^{-j\frac{\pi}{4}} + s_3 \cdot e^{-j\frac{3\pi}{4}} + s_5 \cdot e^{-j\frac{5\pi}{4}} + s_7 \cdot e^{-j\frac{7\pi}{4}} \right)$$

$$S_1 = s_0 \cdot e^{-j0} + s_2 \cdot e^{-j\frac{\pi}{2}} + s_4 \cdot e^{-j\frac{2\pi}{2}} + s_6 \cdot e^{-j\frac{3\pi}{2}} + e^{-j\frac{\pi}{4}} \left(s_1 \cdot e^{-j0} + s_3 \cdot e^{-j\frac{\pi}{2}} + s_5 \cdot e^{-j\frac{2\pi}{2}} + s_7 \cdot e^{-j\frac{3\pi}{2}} \right)$$

Traduction en langage mathématique :

$$S_l = \sum_{i=0}^{3} s_{2i} \cdot W_4^i + W_8^l \cdot \sum_{i=0}^{3} s_{2i+1} \cdot W_4^i$$

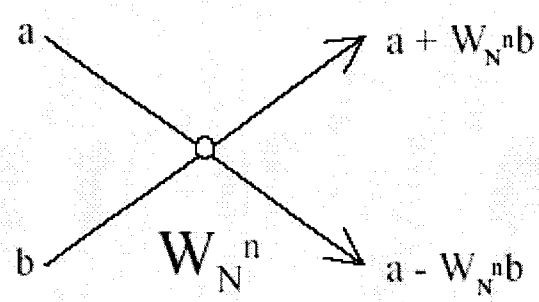
On peut encore subdiviser les 4 échantillons de nos parenthèses :

$$S_l = s_0 \cdot e^{-j0} + s_4 \cdot e^{-j\frac{4\pi}{4}} + e^{-j\frac{2\pi}{4}}\left(s_2 \cdot e^{-j0} + s_6 \cdot e^{-j\frac{4\pi}{4}}\right) + e^{-j\frac{\pi}{4}}\left(s_1 \cdot e^{-j0} + s_5 \cdot e^{-j\frac{4\pi}{4}} + e^{-j\frac{2\pi}{4}}\left(s_3 \cdot e^{-j0} + s_7 \cdot e^{-j\frac{4\pi}{4}}\right)\right)$$

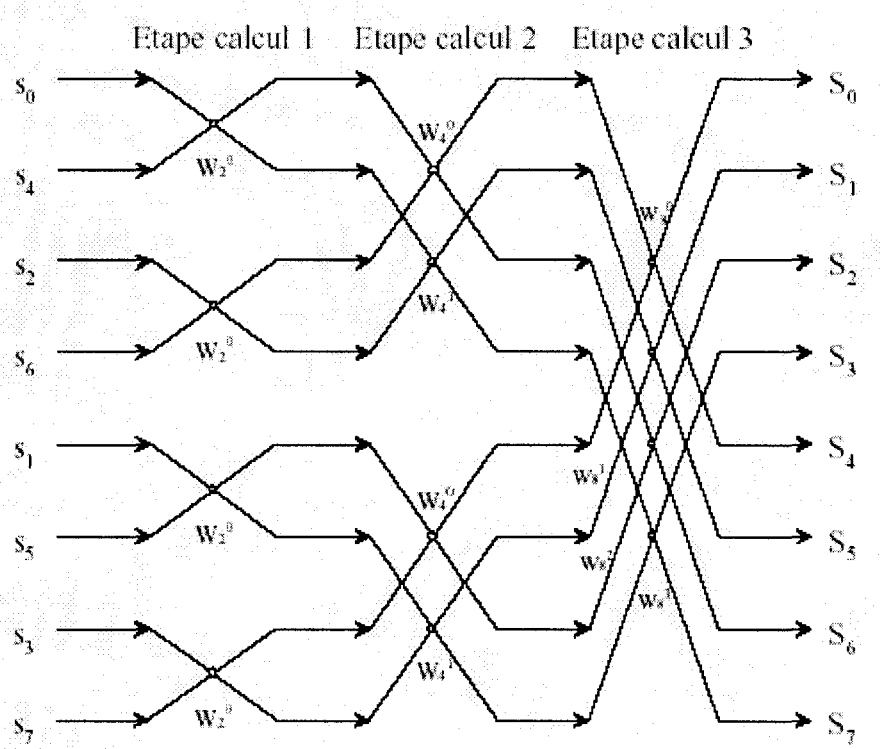
Ainsi, en agissant de même pour les 8 échantillons, on obtient en langage mathématique :

$$\begin{aligned}
S_0 &= s_0 + s_4 W_2^0 + W_4^0 (s_2 + s_6 W_2^0) + W_8^0 [s_1 + s_5 W_2^0 + W_4^0 (s_3 + s_7 W_2^0)] \\
S_1 &= s_0 - s_4 W_2^0 + W_4^1 (s_2 - s_6 W_2^0) + W_8^1 [s_1 - s_5 W_2^0 + W_4^1 (s_3 - s_7 W_2^0)] \\
S_2 &= s_0 + s_4 W_2^0 - W_4^0 (s_2 + s_6 W_2^0) + W_8^2 [s_1 + s_5 W_2^0 - W_4^0 (s_3 + s_7 W_2^0)] \\
S_3 &= s_0 - s_4 W_2^0 - W_4^1 (s_2 - s_6 W_2^0) + W_8^3 [s_1 - s_5 W_2^0 - W_4^1 (s_3 - s_7 W_2^0)] \\
S_4 &= s_0 + s_4 W_2^0 + W_4^0 (s_2 + s_6 W_2^0) - W_8^0 [s_1 + s_5 W_2^0 + W_4^0 (s_3 + s_7 W_2^0)] \\
S_5 &= s_0 - s_4 W_2^0 + W_4^1 (s_2 - s_6 W_2^0) - W_8^1 [s_1 - s_5 W_2^0 + W_4^1 (s_3 - s_7 W_2^0)] \\
S_6 &= s_0 + s_4 W_2^0 - W_4^0 (s_2 + s_6 W_2^0) - W_8^2 [s_1 + s_5 W_2^0 - W_4^0 (s_3 + s_7 W_2^0)] \\
S_7 &= s_0 - s_4 W_2^0 - W_4^1 (s_2 - s_6 W_2^0) - W_8^3 [s_1 - s_5 W_2^0 - W_4^1 (s_3 - s_7 W_2^0)]
\end{aligned} \qquad (6)$$

On peut représenter cet algorithme graphiquement en utilisant l'opération « butterfly » ou papillon :



Ainsi, la FFT de 8 échantillons (6) correspond à la figure suivante :



Enfin, on remarque que l'ordre initial des échantillons n'est pas naturel mais plutôt un ordre entrelacé. On peut obtenir très facilement cet ordre entrelacé à partir de l'ordre naturel en appliquant la technique dite du « bit reversal ». Cette technique consiste à écrire en binaire l'indice de l'échantillon dans l'ordre naturel puis à retourner l'ordre des bits pour obtenir la représentation binaire de l'indice correspondant dans l'ordre entrelacé. Dans notre exemple avec 8 échantillons, voilà comment on procède :

Indice dans l'ordre naturel	Représentation binaire	Représentation retournée	Indice dans l'ordre entrelacé
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Annexe 6 : Transformée en Ondelette.

Du long règne de Fourier aux ondelettes

La transformée de Fourier (1822) joue un rôle prépondérant dans l'analyse et le traitement des signaux. Elle permet en outre de réaliser des opérations de filtrage par simple multiplication dans le domaine fréquentiel (domaine de Fourier).
La transformée de Fourier est une transformation linéaire et inversible permettant de décomposer un signal sur la base des exponentielles complexes. Le spectre obtenu permet alors de rendre compte de la composition fréquentielle du signal original.

$$TF(\omega) = \int_{-\infty}^{+\infty} f(t)\ \exp(j\omega t)\ dt \tag{A.1}$$

Cependant, les signaux possédant des transitoires ou des parties non-stationnaires sont mal décrits ou représentés par une transformée de Fourier.

Par exemple un signal musical, tout simplement parce que l'intégration sur tout le domaine temporel (de $-\infty$ à $+\infty$) va faire disparaître toute relation par rapport à l'indice temporel dans le spectre final. Ainsi dans le spectre de notre exemple du morceau musical qui comprend, imaginons, les trois notes mi, ré, do (dans cet ordre), nous ne saurons à la seule vue du spectre la succession des notes effectivement jouées.

Pour pallier à cette *déficience*, on a eu l'idée d'introduire la transformée de Fourier à fenêtre glissante :

$$TFFG(t,\omega) = \int_{-\infty}^{+\infty} f(s)\ g(s-t)\ \exp(j\omega s)\ ds \tag{A.2}$$

dans laquelle on fait intervenir une fenêtre temporelle dont le rôle est de limiter le domaine d'intégration temporel et d'obtenir ainsi une localisation de l'information.

Cette approche n'est pourtant pas pleinement satisfaisante pour plusieurs raisons :
- premièrement se pose le problème du choix de la fenêtre utilisée et plus précisement du choix de sa largeur temporelle
- deuxièment pour une taille de fenêtre fixée (cf. Figure A.1), on remarque alors que pour des fréquences, hautes ou basses, la résolution obtenue n'est pas optimale :

une exponentielle complexe basse fréquence $\exp(j\omega s)$ modulée par une fenêtre suffisament large $g(s-t)$ permettra l'analyse de basse fréquences dans le signal.
En revanche, une exponentielle haute fréquence modulée par cette même fenêtre ne permettra pas une bonne localisation temporelle de cette fréquence. On aurait le même problème avec une fenêtre courte : bonne localisation en hautes fréquences, mais mauvaise analyse des basses fréquences

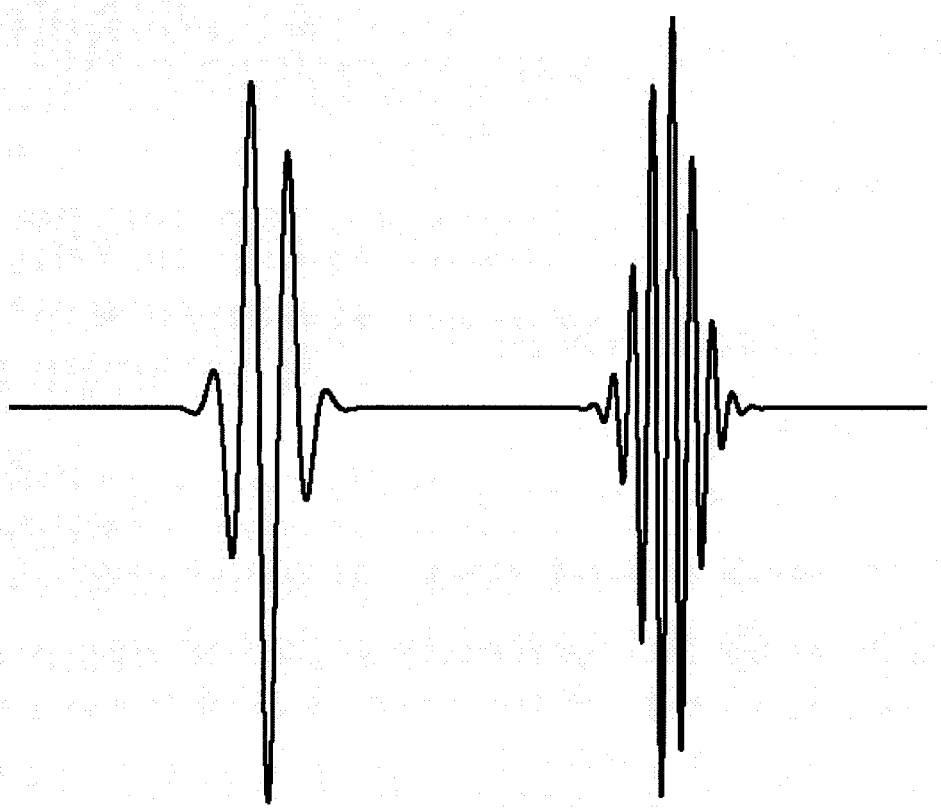


FIG. *A.1* — Fenêtre de la transformée de Fourier à court terme pour deux fréquences différentes

Pour résoudre ces problèmes, aucun compromis sur une taille de fenêtre optimale sans connaissance *a priori* du signal ne semble satisfaisante.

Finalement, le constat qui découle de tout ceci est qu'il faut faire varier la taille de la fenêtre d'analyse pour pouvoir saisir les hautes et basses fréquences à une résolution acceptable. De plus, le nombre d'oscillations contenues dans la fenêtre d'analyse doit demeurer constant à toutes les fréquences.
C'est justement le principe de fonctionnement des ondelettes, qui va être décrit dans les prochaines sections.

Ondelettes : premières propriétés et définitions

De la même façon que la transformée de Fourier peut se définir comme étant une projection sur la base des exponentielles complexes, on introduit la tranformée en ondelettes comme la projection sur la base des fonctions ondelettes.

$$TO(a,b) = \frac{1}{\sqrt{(a)}} \int_{-\infty}^{+\infty} f(t)\, \psi(\frac{t-b}{a})\, dt \quad \text{avec} \quad a,b \in \mathbb{R}, a \neq 0 \tag{A.3}$$

$$TO(a,b) = \int_{-\infty}^{+\infty} f(t)\, \psi_{a,b}(t)\, dt \quad \text{avec} \quad \psi_{a,b}(t) = \frac{1}{\sqrt{(a)}} \psi(\frac{t-b}{a}) \tag{A.4}$$

On obtient des coefficients d'ondelettes ($TO(a,b)$) dépendant de deux paramètres a et b : a étant le facteur d'échelle et b le facteur de translation. Le pas de translation à l'échelle a est $\frac{b}{a}$.
Les fonctions $\psi_{a,b}(t)$ sont obtenues à partir de la dilatation et de la translation de la fonction *mère* $\psi(t)$. Les fonctions $\psi_{a,b}(t)$ sont par conséquent parfois appelées les ondelettes *filles*.

Ces fonctions forment une base, c'est-à-dire que, si l'on note le produit scalaire entre deux fonctions f et g comme étant $< f \mid g >= \int f(t)\, g(t)\, dt$, alors on a :

$$< \psi_{a,b} | \psi_{a',b'} >= \delta_{a,a'} . \delta_{b,b'} \tag{A.5}$$

Inversibilité

Tout comme la transformée de Fourier, la transformée en ondelettes est inversible.

$$f(t) = \frac{1}{C_\psi} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \frac{1}{a^2} < f | \psi_{a,b} > \psi_{a,b}\, da\, db \tag{A.6}$$

où C_ψ est un coefficient dont l'expression est $C_\psi = 2\pi \int_{-\infty}^{+\infty} |\hat{\psi}(\omega)|^2 \frac{d\omega}{\omega}$
(avec $\hat{\psi}(\omega)$ transformée de Fourier de $\psi(t)$).

Condition d'admissibilité

La fonction ondelette doit vérifier un certain nombre de propriétés, la première d'entre elle se nomme *condition d'admissibilité*.

Soit $\psi(t) \in \mathcal{L}^2$, alors

$$\int_{-\infty}^{+\infty} \frac{|\psi(\omega)|}{|\omega|} d\omega < \infty \tag{A.7}$$

Cette condition permet d'analyser le signal, puis de le reconstruire sans perte d'information.

La condition d'admissibilité implique en outre que la transformée de Fourier de l'ondelette à la fréquence du continu (pour $\omega = 0$) doit être nulle. Soit,

$$\hat{\psi}(\omega) \Big|_{\omega=0} = 0 \tag{A.8}$$

Ceci implique en particulier deux conséquences importantes :
- la première est que les ondelettes doivent posséder un spectre de type passe-bande
- la seconde apparaît en réécrivant l'équation A.8 de façon équivalente sous la forme

$$\int_{-\infty}^{+\infty} \psi(t)\, dt = 0 \tag{A.9}$$

et montre que $\psi(t)$ doit être à moyenne nulle

$\psi(t)$ est donc une fonction à largeur temporelle finie (fenêtre temporelle) possédant un caractère oscillatoire. On est donc bien en présence d'une **petite onde** : une ondelette.

Transformée en ondelettes continue - transformée en ondelettes discrète

La formule A.3 dépend des réels a et b, on peut décider d'une infinité de valeur pour ces deux paramètres ; on peut les faire varier continûment : on parle alors de transformée en ondelettes continue.

De par cette aspect, la transformée en ondelettes telle qu'elle est définie est redondante, c'est à dire que l'on obtient plus de coefficients d'ondelettes qu'il n'en est nécessaire pour décrire le signal de manière exhaustive.
En pratique, on a plus souvent affaire à des signaux discrets, mais même sans cela, on a intérêt à discrétiser les valeurs de a et b.

Pour se rendre compte d'une part, de l'intérêt d'utiliser les ondelettes et d'autre part, de la manière dont on va discrétiser les valeurs de a et b, on va regarder comment les ondelettes se *déploient* ou se répartissent en temps et en fréquence, par rapport, par exemple, à une transformée de Fourier à fenêtre glissante.

Pour cela, on va représenter, en les juxtaposant, les supports temporels et fréquentiels des ondelettes dans le plan défini en abscisse par l'axe temporel et en ordonnées par l'axe

fréquentiel. On visualise ainsi comment est découpé le plan temps-fréquence pour chaque type de transformée cf. Figure A.2.

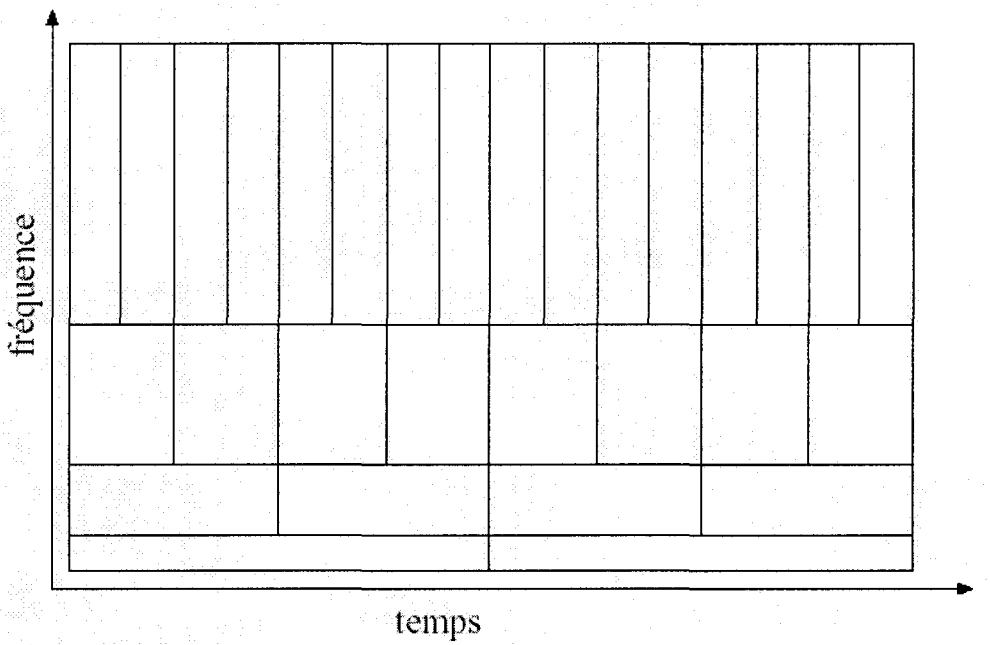


FIG. *A.2* — Plan temps-fréquence ou temps-échelle

Le pavage temps-échelle utilisé sur la figure précédente suggère une méthode de discrétisation exponentielle pour les échelles et pour le temps.

Soit $a = a_0^m$ et $b = b_0^n$ avec $a_0, b_0 \in \mathbb{Z}$. On obtient alors une transformée en ondelettes discrète [1].

$$TO(m,n) \stackrel{\Delta}{=} a_o^{-\frac{m}{2}} \int_{-\infty}^{+\infty} f(t)\, \psi(a_o^{-m}t - nb_0)\, dt \tag{A.10}$$

Si on choisit $a_0 = 2$ et $b_0 = 1$, on parle alors de **transformée en ondelettes dyadique**.

$$TO(m,n) \stackrel{\Delta}{=} (\sqrt{2})^{-m} \int_{-\infty}^{+\infty} f(t)\, \psi(\frac{t}{2^m} - n)\, dt \tag{A.11}$$

[1] il faut noter ici que c'est la transformée qui est discrète, et non l'ondelette qui reste une fonction continue

Régularité

La régularité d'une ondelette [Mal00] est la propriété permettant de localiser les singularités dans un signal. Cette propriété se traduit sur les coefficients d'ondelettes par une amplitude importante, caractérisant une singularité dans le signal, et par la décroissance des valeurs des coefficients avec l'échelle de résolution.

La régularité est une propriété importante pour les applications concernant la compression. En effet, on désire alors obtenir des coefficients d'ondelettes les plus petits possibles (afin de les annuler), pour tout ce qui concerne les détails du signal ; la décroissance des coefficients en fonction de l'échelle est donc primordiale. On peut noter qu'il existe un lien entre la régularité et les moments nuls d'une ondelette. Une ondelette ψ a n moments nuls si :

$$M_k = \int_{-\infty}^{+\infty} t^k \, \psi(t) \, dt = 0 \qquad 0 \leq k < n \tag{A.12}$$

remarque : on a déjà vu que $M_0 = \int_{-\infty}^{+\infty} \psi(t) \, dt = 0$

Pour affiner la compréhension de la régularité, on va partir d'une expression simplifiée d'une transformée en ondelettes.

$$TO(m,q) = \int f(t) \, \psi_{m,q}(t) \, dt \tag{A.13}$$

Posons $q = 0$ dans l'équation A.13 et développons la en série de Taylor autour de $t = 0$

$$TO(m,0) = \frac{1}{\sqrt{(m)}} \left[\sum_{p=0}^{n} f^{(p)}(0) \int \frac{t^p}{p!} \, \psi(\frac{t}{m}) \, dt + O(n+1) \right] \tag{A.14}$$

où $O(n+1)$ désigne les termes d'ordre supérieur à n.

$$TO(m,0) = \frac{1}{\sqrt{(m)}} \left[f(0) \, M_0 \, m + \frac{f^{(1)}(0)}{1!} \, M_1 \, m^2 + \frac{f^{(2)}(0)}{2!} \, M_2 \, m^3 + \ldots + \frac{f^{(n)}(0)}{n!} \, M_n \, m^{n+1} + O(m^{n+2}) \right] \tag{A.15}$$

Si tous les moments $M_0 \ldots M_n$ sont nuls, cela veut dire que les coefficients $TO(m,q)$ vont décroître comme m^{n+2} pour une fonction f suffisamment ***régulière***. Cette propriété de ***régularité-moments nuls*** conditionne l'ondelette à posséder une décroissance rapide. On peut effectuer une mesure de la régularité locale d'une fonction en utilisants les exposants de Lipschitz [MH92].

Implémentation par banc de filtres

On n'évoquera pas ici le cas des ondelettes continues (Morlet, Sombrero) qui s'implémentent directement dans l'espace de Fourier.

Nous avons vu qu'une ondelette possède un spectre de type passe-bande. Que devient ce spectre lorsque l'ondelette est dilatée dans le domaine temporel ?
Pour répondre à cela, on va rappeler une propriété bien connue de la transformée de Fourier :

$$\mathcal{F}f(at) = \frac{1}{|a|}\hat{f}(\frac{\omega}{a}) \tag{A.17}$$

Cette propriété implique que, si nous dilatons notre ondelette d'un facteur 2, cela va compresser la largeur du spectre et décaler le spectre d'un facteur 2.

En répétant la procédure de dilatation, on peut couvrir par décalages successifs l'ensemble du domaine spectral (de la même façon que l'on couvre le domaine temporel par des ondelettes translatées).

Pour obtenir une bonne couverture fréquentielle, les spectres doivent se toucher en *se tenant la main* (cf. Figure A.3) (ceci est possible pour un choix convenable de l'ondelette). Une série d'ondelettes dilatées peut donc être vue comme un ensemble de filtres passe-bande.

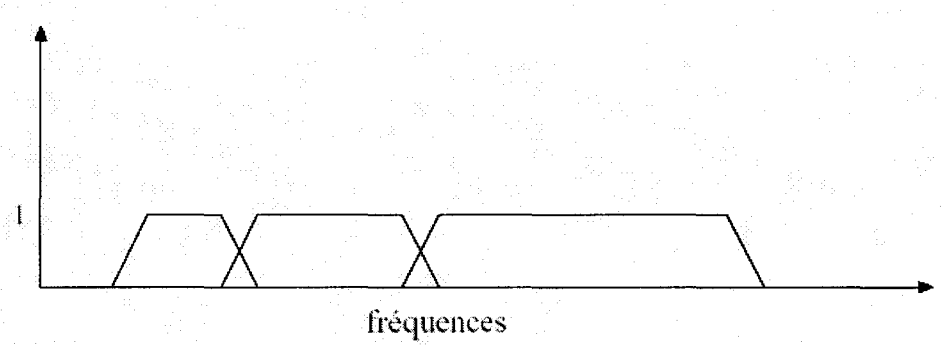


FIG. *A.3* — Les ondelettes forment une série de filtres passe-bande

Annexe 7 – RR series filtering

1 – Principle:

Any artifact on the R-R series is tracking using auto-adaptive thresholds on a moving window. The first threshold overstepping is marked as the beginning of a disturbed period. The end of the disturbed period is marked as the acquisition of a correct sample, that is, which value is inside the thresholds interval. At that time, the erroneous values are replaced by the most probable R-R samples computed by linear approximation using the two correct values on both sides of the disturbed period. At last, the computed samples are integrated into the moving window and taken into account to calculate the new thresholds values.

2 - Artifacts classification:

R wave detection algorithms, used for the R-R intervals calculation, are subject to bad operating due to the relative difficulty to obtain a noiseless ECG signal in long term recording. In practice, we can observe different kind of perturbations such as: baseline wander or amplitude changes which can cause missed detection, power noise, external noise or electrode motion which can cause bad detection, and physiological perturbations such as ectopic beats. Studying the effects of these different perturbations on the R-R series, we have classified the R-R series artifacts in three classes (Fig. 1).

Due to their particular effect on the R-R series, we use the specifications of these classes to elaborate a selective filter which, in complement to the thresholds set, contributes to make the method more robust.

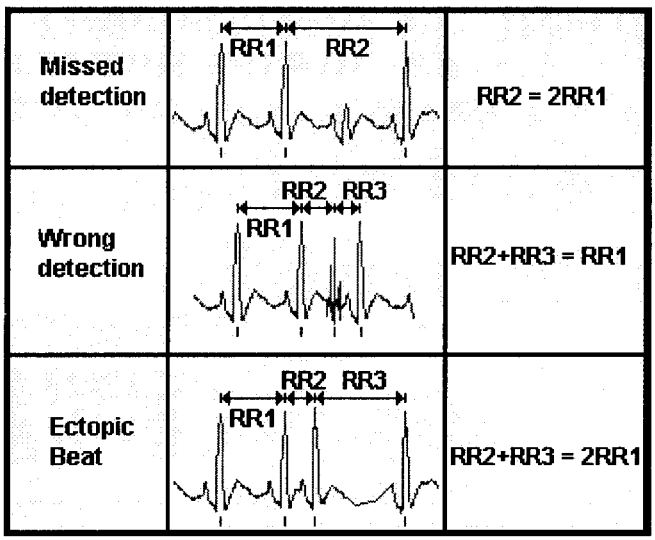


Fig. 1: artifacts classification.

3 - Detection algorithm:

Since the R-R intervals series distribution matches with the normal law, we used its characteristics to elaborate a first auto-adaptive thresholds set. Considering a 20's samples moving window, mean (m_{20}) and standard deviation (σ_{20}) values are used to establish the two thresholds as:

$m_{20} - 2\sigma_{20}$ and $m_{20} + 2\sigma_{20}$.

So, each new sample acquired is compared with the thresholds computed on the previous valid window. According to the normal law characteristics, the $[m_{20} - 2\sigma_{20}, m_{20} + 2\sigma_{20}]$ interval includes 95% of the valid R-R values. Therefore, 5% of valid values, representing the R-R extremes values of the window, will be detected such as perturbations. To mark the difference between valid samples and perturbations, we use a selective filter elaborated from the different artifacts specifications.

Therefore, any sample outside the $[m_{20} - 2\sigma_{20}, m_{20} + 2\sigma_{20}]$ interval is submit to three conditions :

- C1: $RR_i < m_{20} - 2\sigma_{20}$ and $RR_{i+1} > m_{20} + 2\sigma_{20}$

- C2: $RR_i < 0.75\ RR_{i-1}$ or $RR_{i+1} < 0.75\ RR_{i-1}$

- C3: $RR_i > 1.75\ RR_{i-1}$

If the sample agrees with one of these conditions, it is marked as a wrong sample. Otherwise, it is stored as an indeterminate sample. In such a case, all new sample will be stored until the detection of a correct sample (inside $[m_{20} - 2\sigma_{20}, m_{20} + 2\sigma_{20}]$) or a wrong one (agrees with C1, C2 or C3) occurs. If a correct sample is detected, all the stored samples are marked as correct R-R intervals and taken into account for the next moving window thresholds computation. If a wrong sample is detected then all the stored samples are marked as wrong and replaced by rebuilt ones.

4 - Rebuilding algorithm:

After detection, the wrong samples have to be replaced by correct ones while keeping the real recording time and without having any effect on the frequency content. In that way, we chose to rebuild the disturbed area using a linear interpolation between the two correct samples located on both sides of the disturbed area as follow:

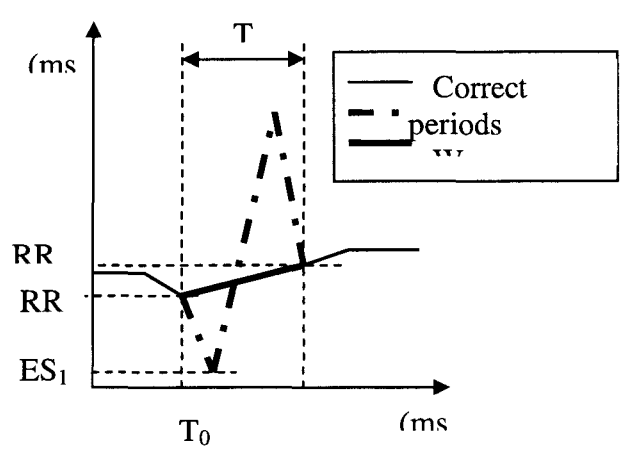


Fig. 2: linear interpolation

The rebuilt area is a straight line which equation is:

$RR_i = aT_i + b$; $a = (RR_m - RR_0) / T$; $b = RR_0$

where RR_i is the computed sample at time T_i, RR_0 is the last sample marked as correct and RR_m is the first sample after the disturbed period. The interpolation duration T is computed as the sum of the wrong samples values added to the first sample value after the disturbed period RR_m.

$$T = Tm\text{-}T0 = \sum_{i=0}^{n} ESi + RR_m$$

where ES_i are the erroneous samples values and n the number of erroneous samples in queue.

So, for each computed sample:

$$RR_k = a^* \sum_{i=0}^{k} RRi + b$$

or

$$RR_k = [a^* \sum_{i=0}^{k-1} RRi + b] / (1 - a) \text{ with } RR_1 = b / (1\text{-}a)$$

Since R-R intervals are expressed in millisecond, the integer values of RR_k are used to replace the wrong samples. But, rounding these values may change the total record time proportionally to the count of rebuilt samples k. According to that, the difference between the disturbed period duration and the interpolation duration is computed and equally distributed to each rebuilt sample to match the time difference. Finally, to valid the rebuilt samples, the first and the last samples of the interpolation (x_0 and x_n) are compared with the two values on both sides of the disturbed period (RR_0 and RR_m). If $\mathbf{x_0 = RR_0 \pm 10\%}$ and $\mathbf{x_n = RR_m \pm 10\%}$ then the computed samples and RR_m are marked as correct and taken into account for the next

moving window thresholds computation. Otherwise, the sample RR_m is notified as a wrong sample and then another linear interpolation is computed using the next sample RR_{m+1}.

5 - Real time implementation:

The main interest of this filtering method rests in the possibility to use it in real time. In that way, we developed a real time R-R series filtering algorithm based on the detection and rebuilding algorithms previously described. However, these algorithms need to be adjusted to be able to detect and replace any wrong sample continuously.

In the real time implementation, the detection algorithm is used to detect the first sample of any disturbed area. Other wrong area samples will be detected thanks to the rebuilding algorithm. Indeed, since a wrong sample RR_i is detected, a rebuilding attempt is performed using the last correct sample and the next sample RR_{i+1}. If the linear interpolation is not correct, RR_{i+1} is marked as a wrong sample and a rebuilding attempt is performed using the next sample RR_{i+2}, and so on... Of course, using this process on a long period would modify the frequency content of the signal because the rebuilt part of the series would differ so much from the real one. That is the reason why we fixed a maximum number of wrong samples in queue (Nw_{max}) equal to 20 and a maximum number of indeterminate sample (Ni_{max}) equal to 5. Therefore, when the number of wrong samples is up to Nw_{max}, we chose to mark all the wrong samples in queue as correct. When the number of indeterminate samples is up to Ni_{max}, the older indeterminate sample in queue is marked as correct and the new one is stored as indeterminate in queue.

Références bibliographiques

[ASCH 87] Asch G. – *Les capteurs en instrumentation industrielle* – Dunod, 1987

[BENO 03] Benoit A.- *Méthodes et algorithmes pour l'évaluation des performances des systèmes informatiques à grand espace d'états* - Thèse de Docteur de l'INPG, Institut National Polytechnique de Grenoble, 2003.

[BIAN 97] Anna M. Bianchi, Luca T. Mainardi, Carlo Meloni, Sergio Chierchia, Sergio Cerutti – *Continuous Monitoring of the Sympatho-Vagal Balance Through Spectral Analysis* – IEEE engineering in medicine and biology volume 16, number 5, September/October 1997, pp 64-73.

[BLOC 86] Block FE Jr. – *Do we monitor enough ? We don't monitor enough* – J. Clin. Monit. October 1986.

[CANA 01] – *Exigences en matière de système qualité pour les dispositifs médicaux* – Industrie Canada, 2001.

[CLAI 97] Clairambault J., Médigue C., Bestel J. – *Le système cardiovasculaire et sa régulation à court terme par le système nerveux autonome* – Journée Automatique et Santé, Montpellier, mai 1997.

[CONS 01] Constant I.- *Physiologie du système nerveux autonome* – Groupe liaison SA, fevrier 2001.

[DEJO 03] J. De jonckheere,– *Modélisation de la régulation du système cardiovasculaire par le système nerveux autonome* – Mémoire de D.E.A., Université des Science et Technologies de Lille, 2003.

[DEJO 05] J. De jonckheere, R. Logier, A. Dassonneville, G. Delmar, C. Vasseur – *Physiotrace : an efficient toolkit for biomedical signal processing* – IEEE EMBC, Shanghai, 2005.

[DEJO 07] J. De jonckheere, , M. Jeanne, A. Grillet, S. Weber, P. Chaud, R. Logier, JL. Weber – *OFSETH: Optical Fibre Embedded into technical Textile for Healtcare, an efficient way to monitor patient under magnetic resonance imaging* – IEEE EMBC. Lyon 2007.

[FUND 01] – *Fundamental Concept of Data Acquisition* – Rev.2, 8/01 Copyright DelSys Inc., 2001.

[GULL 96] – Gullekson G., Selic B. – *Design Pattern for real time software-* Embedded Systems Conference West, 1996.

[HEIN 01] George T. Heineman., William T. - Councill. *Component-based software engineering : Putting the pieces together*, AddisonWesley, 2001.

[JEAN 04] Jeanne M. *Mesure de la variabilité sinusale du rythme cardiaque pour apprécier la composante analgésique de l'anesthésie générale.* DEA Sciences et Technologies (Majeur Génie Biomédical). Université de Technologies de Compiègne. 2004.

[JEAN 04.1] Jeanne M. *Effet de l'analgésie péridurale sur l'activité du système nerveux autonome au cours du travail obstétrical.* Thèse pour le Doctorat en médecine. Université de Lille II. 2004.

[KYRI 07] Kyriaki K., Spyridou and Leontios J. Hadjileontiadis – *Analysis of Fetal Heart Rate in Healthy and Pathologicl Pregnancies Using Wavenlet-based Features* – Proceeding of the 29th Annual International Conference of the IEEE EMBS, Lyon, 2007.

[LATS 93] Latson TW, O'Flaherty D. *Effects of surgical stimulation on autonomic reflex function: assessment of changes in Heart Rate Variability.* Br J Anaesth 1993; 70: 301-5

[LOGI 90] Logier R., Dagano J., Kacet S., Lacroix D., Carles O., Libersa C., Fiévé R., Lekieffre J. – *Analyse spectrale de la variabilité de la fréquence cardiaque : développement d'une station d'acquisition et de traitement* – R.B.M. 1990, 12,1, 32-34.

[LOGI 95] Logier R. – *Contribution à l'étude conceptuelle des instruments intélligents : une méthodologie appliquée au monitorage médical* – Thèse de Docteur Ingénieur, Université des science et Technologie de Lille, 1995.

[LOGI 01] Logier R., Matis R., De jonckheere J. – *Analyse spectrale du rythme cardiaque fœtal pendant le travail obstétrical : étude de faisabilité* – ITBM-RBM 2001 ; 22 : 31-7.

[LOGI 01.1] Logier R., Dassonneville A. – *Procédé et dispositif de filtrage d'une série RR issue d'un signal cardiaque, et plus particulièrement d'un signal ECG* – Brevet - France, N° 01 02760, 28/02/2001- PCT WO 2002/069178 A3, 06/09/2002 – dépôt Europe en cours.

[LOGI 03] Logier R., Dassonneville A. – *Système d'acquisition modulaire et temps réel de signaux, et notamment de signaux biomédicaux* – Brevet - France, N° 03 15130, 22/12/2003.

[LOGI 03.1] Logier R., Dassonneville A. – *Méthode de traitement fréquentiel d'une série RR, procédé et système d'acquisition et de traitement d'un signal cardiaque analogique, et application à la mesure de la souffrance fœtale* – France, N° 02 06676, 31/05/2002 - PCT FR03/01226, 16/04/2003.

[LOGI 04] R. Logier, J. De Jonckheere, A. Dassonneville – *An efficient algorithm for R-R intervals series filtering* - Proceedings of the 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society. 2004.

[LOGI 04.1] Logier R., Jeanne M., Tavernier B. – *Method and device for assessing the depth of analgesia during general anaesthesia* – Provisional Application N° EV 406 076 745, USA, 15/04/2004.

[LOGI 04.2] Régis Logier, Mathieu Jeanne, Benoît Tavernier – *Procédé et dispositif d'évaluation de la douleur chez un être vivant* – Demande de brevet Européen N° 04370029.3, CHRU de Lille, Université de Lille II, 20 Septembre 2004.

[LOGI 04.3] Régis Logier, Julien De jonckheere, Alain Dassonneville – *Physiotrace : a simple way to get physiological parameters during experimental research* – Séminaire Cancer, années croisées france-Chine, Shanghai 13-15 décembre 2004.

[LOGI 06] Logier R., Jeanne M., Tavernier B., De jonckheere J. *Pain / Analgesia evaluation using heart rate variability analysis.* IEEE EMBC. 2006, New York.

[MEYE 90] Muller P.A. – *Modélisation objet avec UML* – Rational Software, 1997.

[MULL 97] Meyer Y. - *Ondelettes et Operateurs* – Hermann éditeur des sciences et des arts, 1990.

[OMG 99] OMG . – *OMG Unified Modeling Language Specification* – www.omg.org, 1999.

[OMG 04] OMG . – *Exploring Interoperability Requirements in Healthcare* – www.omg.org, 2004.

[PIRO 07] F. Pirotte, A. Depre, R. Shishoo, J. De jonckheere, A. Grillet – *Smart textiles embedded with optical fibres sensors for health monitoring of patients.* - Proceedings of the 4th International Conference and Exhibition on Healthcare and Medical Textiles (MedTex07), Bolton, 2007.

[SAKS 99] Saksena M., Selic B. – *Conception de logiciel temps réel – Progrès actuels et défis à venir* – IEEE Canadian Review – Summer / été 1999.

[SENH 02] Senhadgi L., Wodey E., Claude E. – *Monitoring Approaches in general anesthesia : a survey* – Crit Rev Biomed Eng – 2002 ; 30 (1-3) ; 85 – 97.

[SRLF 00] – *Les recommandations des experts de la SRLF* – Réanim Urgence 2000 ; g : 407-12, Edition scientifiques et médicales Elsevier SAS., 2000.

[TASK 96] Task Force of the European Society of Cardiology the North American Society of Pacing Electrophysiology. *Heart Rate Variability. Standards of Measurement, physiological interpretation and clinical use.* Circulation 1996; 93: 1043-1065

[WIKL 97] U. Wiklund, M. Akay, U. Niklasson – *Short-Term Analysis of Heart-Rate Variability by Adapted Wavelet Transforms* - – IEEE engineering in medicine and biology volume 16, number 5, September/October 1997, pp 113-118.

RESUME

Le développement d'applications de monitorage médical implique, pour les concepteurs, l'utilisation d'outils adaptés. Ce mémoire propose une méthodologie orientée objet pour le traitement temps réel des signaux en décrivant, dans un premier temps, une architecture logicielle générique pouvant servir de base au développement de tout type d'application, d'acquisition et de traitement du signal. Basée sur une architecture de type flot de données, ce cadre d'applications constitue un ensemble de classes et de sous classes qui, par généralisation, donnent naissance à différents composants logiciels spécialisés pour une application de traitement du signal particulière. Cette architecture est présentée au chapitre 3. Dans la seconde partie de ce mémoire, nous donnons une brève description de l'architecture matérielle du réseau d'acquisition constituant la source de données pour nos applications. Nous décrivons alors différents types de composants logiciels pour l'acquisition et le traitement des signaux physiologiques. Enfin, le dernier chapitre présente une application possible de l'outil développé. Il s'agit d'un dispositif innovant pour l'évaluation du niveau d'analgésie lors de l'anesthésie générale basé sur l'analyse des variations du rythme cardiaque.

ABSTRACT

Healthcare monitoring applications development implies, for the designers, the use of adapted tools. This research presents an object oriented methodology for real signal processing by describing, in a first step, a generic software architecture which can be use to develop all kind of signal processing application. Based on the data flow concept, this application framework is composed of a set of classes and sub-classes which, by inheritance, allows creating software components specialized for a particular signal processing application. This framework is described in the chapter 3. In the second part of this document, we give a quick description of the data acquisition network hardware which constitutes the data source for our application. Then, we describe a set of software components for physiological signal acquisition and processing. Finally, the last chapter introduces an application of the software components library. This application is a monitoring system for the pain level evaluation during general anesthesia.