METHODE SA-RT

1 INTRODUCTION

La méthode SA-RT (Structured Analysis for Real Time) est une méthode d'analyse fonctionnelle et opérationnelle des applications temps réel (TR). Cette méthode permet de réaliser une description graphique et textuelle de l'application en termes de besoins.

Deux groupes ont élaboré la méthode SA-RT avec différences notables en termes de représentation.

- la méthode établie par Ward et Moller (WM) associe les modèles fonctionnel et événementiel dans un même digramme.

- la méthode de Harley et Pirbhai (HP) sépare l'aspect fonctionnel et l'aspect événementiel.

Dans l'approche de Ward et Moller (que nous traitons dans ce cours), la spécification est appelée *modèle essentiel* (modèle de besoins pour HP) et la conception porte le nom de *modèle d'implémentation* (modèle d'architecture pour HP).

Le modèle essentiel se compose d'un modèle d'environnement et d'un modèle comportement. Modèle de d'environnent contient le diagramme de contexte du système. Le modèle de comportement contient les modèles des aspects fonctionnel, événementiel et informationnel.

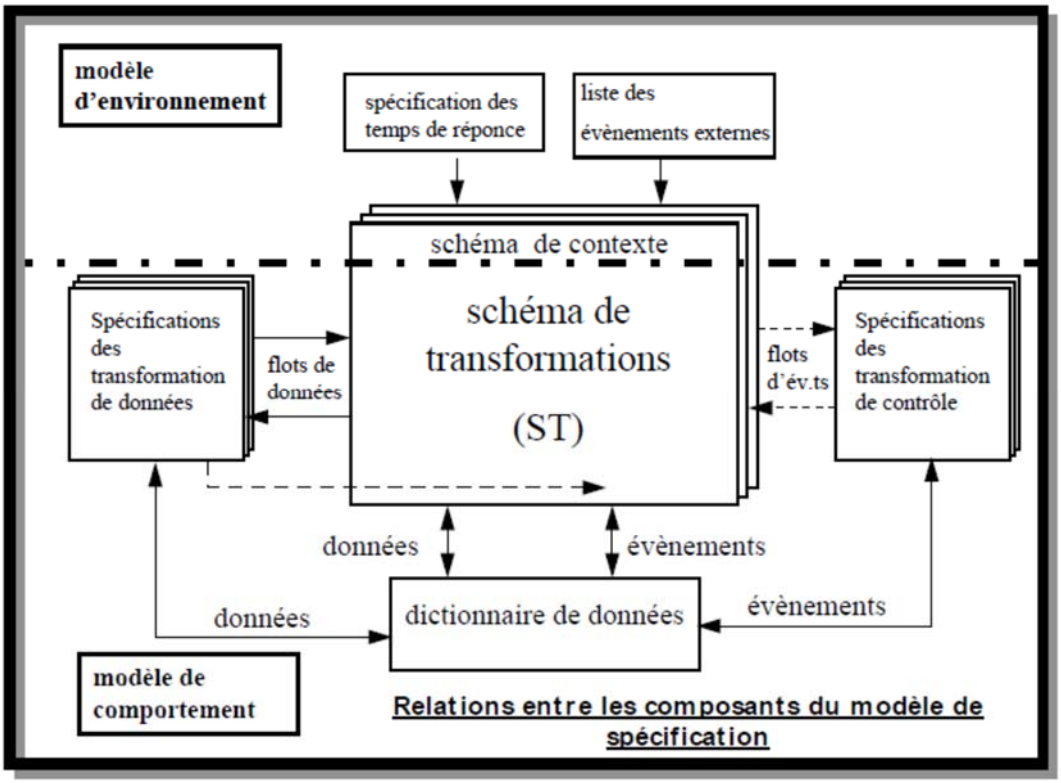


Figure 1

2 DIGRAMME DE CONTEXTE

Le digramme de contexte est un diagramme très abstrait qui représente le système à modéliser. Il ne contient qu'un seul processus dont le nom traduit la fonction d'usage du système. Il est le seul digramme

dans lequel sont représentées les interfaces entre le système et l'environnement (les bords ou les terminaisons). La Figure 2 illustre un exemple de régulation d'amplitude d'un signal sonore.

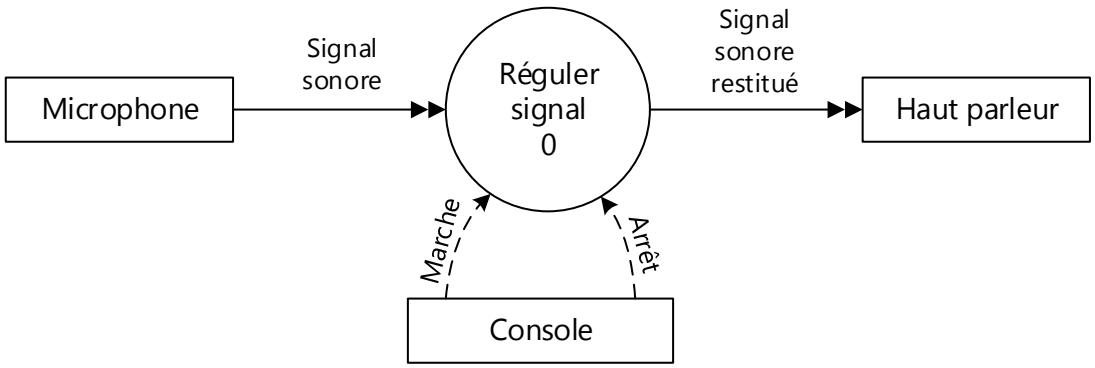


Figure 2: digramme de contexte de régulation d'un signal sonore

3 ASPECT FONCTIONNEL

Le modèle fonctionnel prend en considération :

- Des données porteuses de traitements (leur provenance, leur destination et leur stockage intermédiaire)
- Les transformations qu'elles subissent, entre leur obtention en entrée et leur production en sortie.

3.1 Digramme préliminaire (Digramme de Flots de Données « DFD »)

C'est la première décomposition du processus présenté dans le diagramme de contexte. La Figure 3 illustre le DFD de l'exemple précédent ; le signal sonore est amplifier s'il est faible (inférieur à un seuil MIN) et atténuer s'il dépasse le seul MAX.

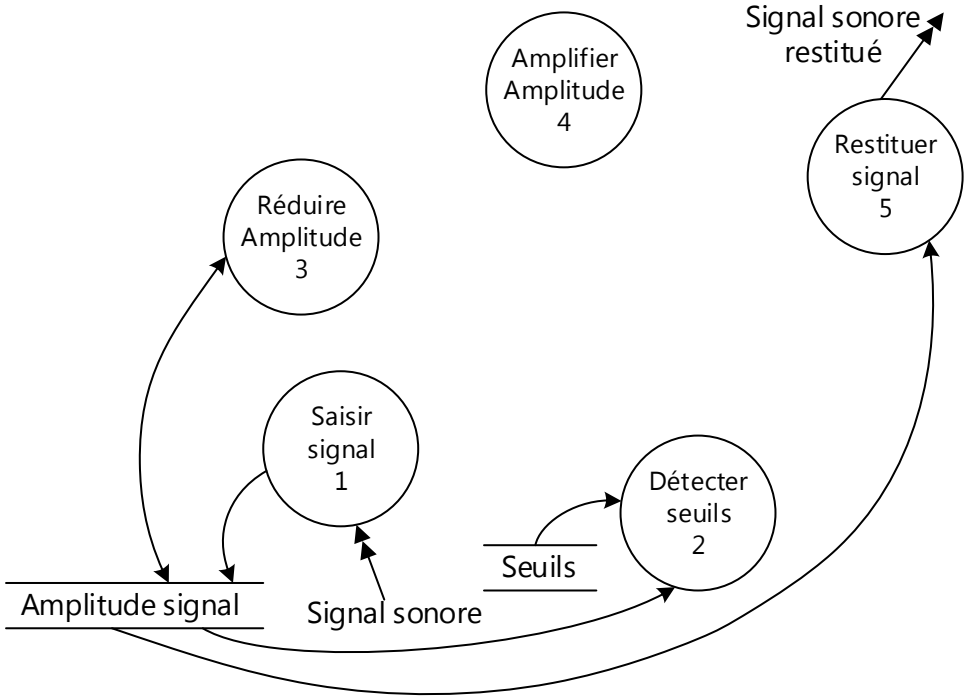
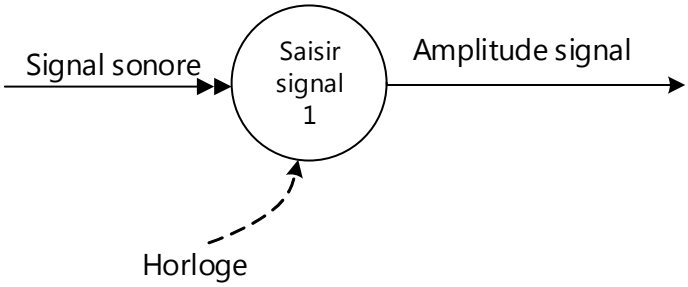


Figure 3: diagramme de flots de données

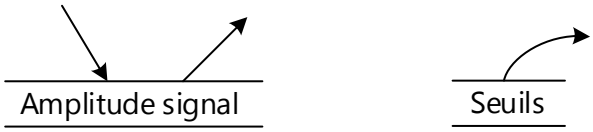
3.2 Composants d'un DFD

- Flot de donnée : indique le chemin suivi par une donnée qui circule entre les processus. Il est représenté par une flèche avec l'identificateur de la donnée. Une donnée continue est représentée par une flèche à double pointure.

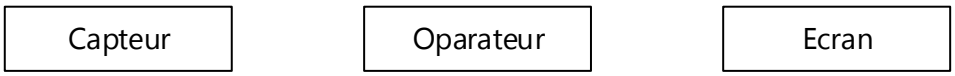
- Processus : un processus est une unité d'activité réalisée par le système. Il est représenté par un cercle entourent son identificateur et son numéro.



- Stockage de donnée : modélise le besoin de mémorisation d'une donnée ou d'un groupement de donnée qui peut être utilisé par tout processus. Il est représenté par deux lignes parallèles encadrant l'identificateur de la donnée.

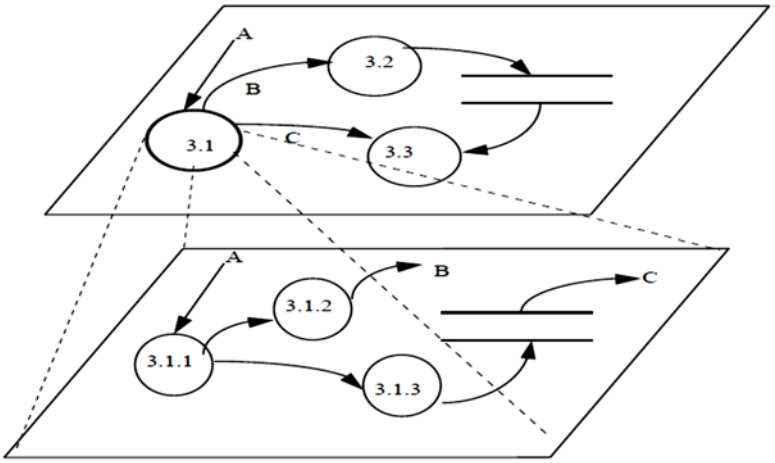


- Bord du modèle : Entité située dans l'environnement qui échange les données avec le système. Il est représenté par un rectangle encadrant l'identificateur du bord.



3.3 Hiérarchisation du modèle fonctionnel

Pour réduire la complexité du système en phase de modélisation, on opte en hiérarchiser le modèle de façon descendante par niveau de détail croissant.



3.4 Spécification des processus

La décomposition d'un processus s'arrête quand sa représentation par un DFD ne peut plus rendre compte des détails supplémentaires, un processus en fin de décomposition est appelé processus primitif. A ce niveau, il doit être décrit par une spécification textuelle.

1- Processus : Saisir signal
E/ donnée : Signal sonore
S/ donnée : Amplitude signal
Acquérir signal
Ranger valeur saisie dans Amplitude signal

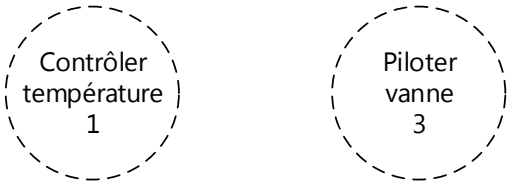
4 ASPECT EVENEMENTIEL

4.1 Introduction

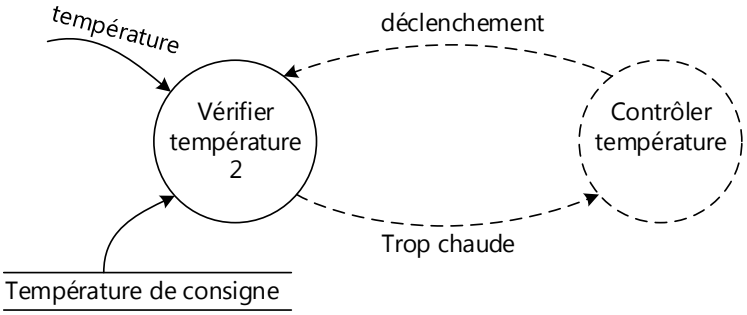
Dans le diagramme de flot de données le déclenchement d'exécution des processus de données peut être lié au rythme d'apparition des données entrantes. Mais dans le cas de la spécification des applications temps réel, il est préférable de piloter ces processus de données par des conditions externes comme l'occurrence d'événements. Cette vue dynamique du modèle impose la mise en place de la partie contrôle, nommé *processus de contrôle* ou *flot de contrôle*.

4.2 Composants de l'aspect événementiel

- processus de contrôle : représente la logique de pilotage des processus fonctionnels. Il gère l'ensemble des événements qui vont activer ou désactiver les processus fonctionnels ; en retour, les processus fonctionnels fournissent au processus de contrôle tous les événements nécessaires aux prises de décision. Le processus de contrôle est représenté par un cercle en pointillé entourant sont identificateur et son numéro.



- flot de contrôle : transporte les événements ou informations qui conditionnent directement ou indirectement l'exécution des processus de transformation de données. Le flot de contrôle est représenté par une flèche en pointillée avec un identifiant.

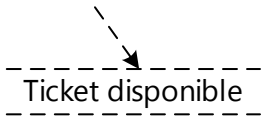


Les événements fournis par les processus de contrôle sont généralement liés à l'activation et à la désactivation des processus fonctionnels, appelés aussi activateurs/désactivateurs des processus. Il existe trois activateurs/désactivateurs principaux :

- *Autorisation* (A) : initie les activités de transformation.
- *Inhibition* (I) : termine les activités d'une transformation.
- *Déclenchement* (D) : qui lance l'activité d'une transformation réalisant une opération ponctuelle dans le temps et se terminant d'elle-même.

Les deux premiers événements sont utilisés ensemble « A/I ».

- stockage d'événements : est une mémorisation d'un ou plusieurs flots d'événements, qui sont maintenus de façon durable et mis à la disposition de toute transformation qui le nécessite. A la différence des stockages des données, un stockage d'événements peut recevoir des flots d'événements en provenance de l'environnement, sans qu'ils soient produits par une transformation. Le stockage d'événements est représenté par deux lignes parallèles en pointillé encadrant l'identificateur d'événement.



4.3 Règles de formation d'un schéma de transformation (ST)

- Un niveau de schéma de transformation peut comporter ou non des processus de contrôle.
- La hiérarchisation de l'aspect événementiel suit celle de l'aspect fonctionnel.
- Les connexions entre les éléments d'un ST sont données par le tableau suivant :

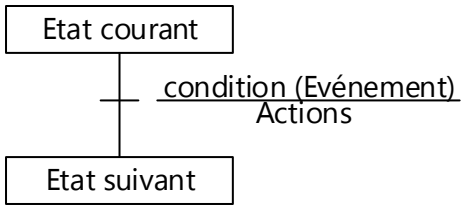
de \ Vers	Transformation de données	Transformation de contrôle	Stockage de données	Stockage de contrôle	bord
Transformation de données	→ ⇉	⇢	→	⇢	⇢ ⇉ →
Transformation de contrôle	⇢ A/I/D	⇢ ⇢ A/I/D	interdit	⇢	interdit
Stockage de données	→	interdit	interdit	interdit	interdit
Stockage de contrôle	interdit	⇢	interdit	interdit	interdit
bord	→ ⇉	⇢	interdit	⇢	interdit

4.4 Diagramme Etat – transition

La représentation de l'aspect comportemental d'une application temps réel peut être faite de diverses manières : diagramme état- transition, table état-transition, matrice état-transition ou un GRAFCET. La représentation la plus courante est le diagramme état-transition.

Ce digramme est composé de :

- Etat courant : correspond à l'état actuel du système
- Transition : représente le passage d'un état à un autre
- Evénement : condition qui provoque le franchissement d'une transition
- Action : qui autorise ou inhibe l'exécution d'un processus
- Etat suivant : état du système après franchissement de la transition



- Règles de formation
  - Un des états est désigné comme état de départ ou initial, souvent appelé « Repos ».
  - Une transition peut exister entre deux états quelconques du système y compris entre un état et lui-même.
  - Plusieurs conditions peuvent être nécessaires à la réalisation d'une transition et plusieurs actions peuvent y être associées. Les événements sont regroupés par les opérateurs ET, OU et NON.
  - Une condition peut provoquer une transition, sans qu'il y ait d'action associé.

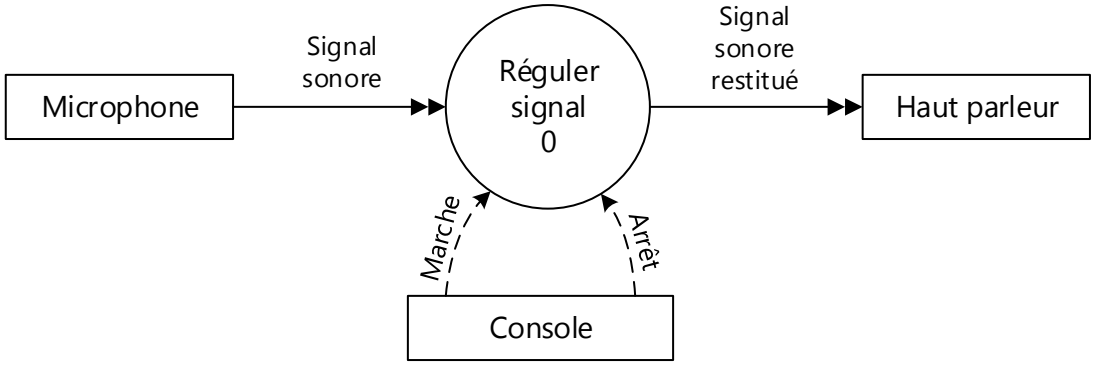
5 ASPECT INFORMATIONNEL

Les données et les événements, qui interviennent à tous les stades du modèle, sont alors réunis dans un dictionnaire de données. Le tableau suivant illustre une spécification de données basée sur la notation de BNF (Backus Naur Form).

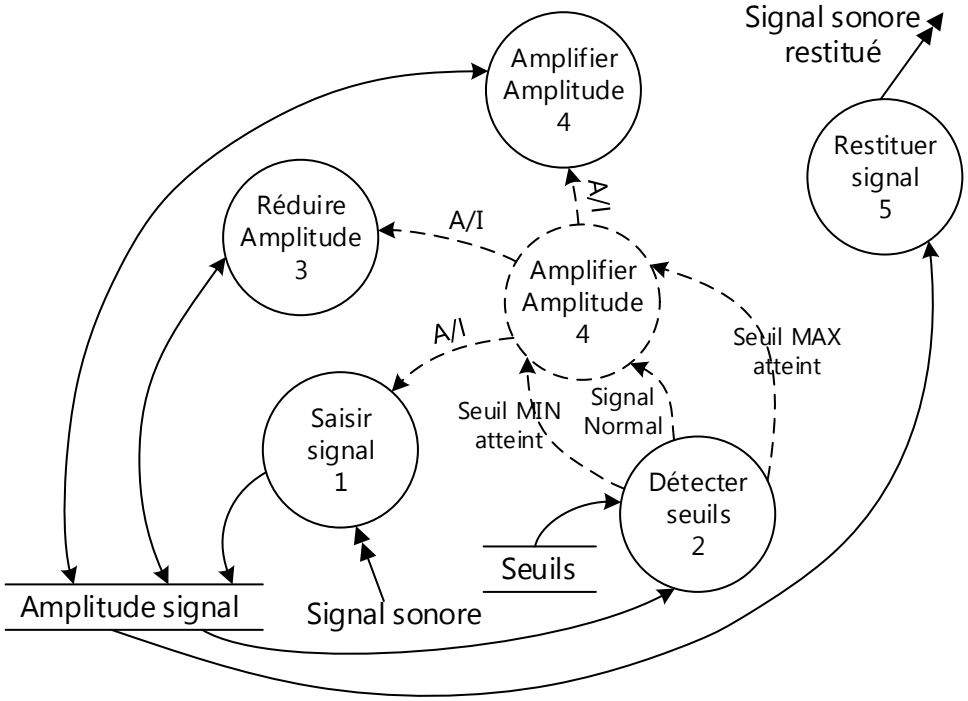
Symbole	Signification
= ...	Composé de
.....	Commentaire
+	Regroupement
{.....}	Itération non bornée
n{.....}p	Itération de n à p
(.....)	Optionnel
".....”	Expression littérale
(|) ou (/)	Sélection ou Exclusif

Exemple d'application : R□gulation d'un signal sonore

1. Rappel du diagramme de contexte



2. Schéma de transformation



3. Spécification des processus

1- Processus : **Saisir signal**

E/ donnée : Signal sonore

S/ donnée : Amplitude signal

Acquérir signal

Ranger valeur saisie dans Amplitude signal

2- Processus : **Détecter seuil**

E/ Seuil MIN, Seuil MAX

E/ Amplitude signal

S/ Evénements : Seuil MIN atteint, Seuil Max atteint, Signal Normal

```
Si Amplitude signal > seuil MAX
    Retourner Seuil MAX atteint
Sinon
    Si Amplitude signal < Seuil MIN
        Retourner Seuil MIN atteint
    Sinon
        Retourner signal Normal
```

3- Processus : **Réduire Amplitude**

E/ Donnée : Amplitude Signal

Nécessite : Kr

Ranger Amplitude signal * Kr dans Amplitude signal

4- Processus : **Amplifier Amplitude**

E/ Donnée : Amplitude Signal

Nécessite : Ka

Ranger Amplitude signal * Ka dans Amplitude signal

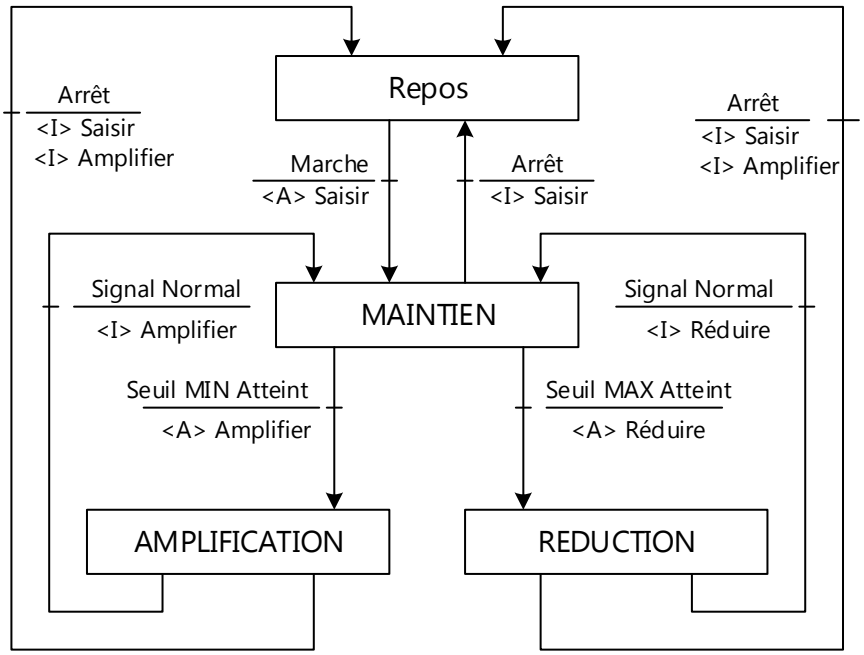
5- Processus : **Restituer signal**

E/ Donnée : Amplitude Signal

S/ Donnée : signal sonore restitué

Appliquer Amplitude signal au signal sonore restitué

4. Diagramme Etat - transition



5. Spécification de données et événements

Signal sonore = * signal analogique [0 .. 5V] *

Signal sonore restituer = * signal analogique [0 .. 5V] *

Marche =* signal logique (vrai ou faux)*

Arrêt = * signal logique (vrai ou faux)*

Amplitude signal = * équivalent numérique du signal sonore [0 .. 255] *

Seuil MIN = * donnée interne de type entier codée sur 8 bits *

Seuil MAX = * donnée interne de type entier codée sur 8 bits *

Seuil MIN atteint = 1 , * événement interne de type entier *

Seuil MAX atteint = 2 , * événement interne de type entier *

Signal normal = 0 , * événement interne de type entier *

Programmation en C

```
#define Seuil_MIN             50
#define Seuil_MAX             200
#define Seuil_MIN_atteint     1
#define Seuil_MAX_atteint     2
#define Signal_Normal         0
#define Ka                    1.25
#define Kr                    0.75
unsigned char Etat            = 0  ;      // 0 : Etat repos
                                          // 1 : Maintien
                                          // 2 : Amplification
                                          // 3 : Reduction
unsigned char  AI_Saisir      = 0 ,
               AI_Amplifier   = 0 ,
               AI_Reduire     = 0 ;
unsigned char  Amplitude_Signal ;

//*********************************************************************
void Saisir ()
{
   Amplitude_Signal = Read_ADC();   // conversion analogique numérique
}
unsigned char detecter( unsigned char data)
{
   if(data > Seuil_MAX)
      return (Seuil_MAX_atteint);
   else
      if(data < Seuil_MIN_atteint)
         return (Seuil_MIN_atteint)
      else
         return (Signal_Normal);
}
```

```
void Amplifier()
{
   Amplitude_Signal = Amplitude_Signal * Ka;
}
void Reduire()
{
   Amplitude_Signal = Amplitude_Signal * Kr;
}
void Resituer()
{
   PORTB = Amplitude_Signal ;
}
void Repos()
{
  if(Marche == 1)
  {
      AI_Saisir   = 1;
      Etat        = 1;
  }
}
void Maintien()
{  unsigned char seuil;
   if(Arret == 1)
   {AI_Saisir = 1; Etat = 0; }
   seuil = detecter (Amplitude_Signal);
   if(seuil == Seuil_MAX_atteint)
   {
      AI_Reduire = 1; Etat = 3;
   }
   if(seuil == Seuil_MIN_atteint)
   {
      AI_Amplifier = 1; Etat = 2;
   }
   if(seuil == Signal_Normal)
      Restituer();
}
```

```
void Amplification ()
{
   unsigned char seuil;
   seuil = detecter (Amplitude_Signal);
   if(Arret == 1)
   {  AI_Saisir = 0;
      AI_Amplifier = 0;
      Etat = 0;
   }
  if(seuil == Signal_Normal)
   {
      AI_Amplifier = 0; Etat = 1;
   }
}
void Reduction ()
{
   unsigned char seuil;
   seuil = detecter (Amplitude_Signal);
   if(Arret == 1)
   {  AI_Saisir = 0;
      AI_Reduire = 0;
      Etat = 0;
   }
  if(seuil == Signal_Normal)
   {
      AI_Reduire = 0; Etat = 1;
   }
}
```

```
void main()
{
   Init();
   while(1)
   {
      switch (Etat)
      {
         case 0 : Repos(); break;
         case 1 : Maintien(); break;
         case 2 : Amplification(); break;
         case 3 : Reduction(); break;
      }
      if(AI_Saisir == 1)      Saisir();
      if(AI_Amplifier == 1)   Amplifier();
      if(AI_Reduire == 1)     Reduire();
   }
}
```