# Control de réservoir de carburant

**Gnebehi Bagre**

**Ingénierie des logiciels**

**Ouriachi Khadir**

# I-Objectif du modèle

Produire un logiciel qui contrôle un système de control de réservoir qui répond aux règles de sûreté de fonctionnement:

Le but étant de maintenir à un très haut débit le carburant (température basse et pression haute)

# II-Définir les besoins

R1 :Des capteurs de températures et de pressions pour mesurer ces grandeurs

R2 :Des Régulateurs de températures et de pression pour corriger en temps réél l'écart entre les valeurs fixées pour le système et les valeurs mesurées en temps réel par les capteurs.

R3 : La température basse est comprise entre 6 et 9 degrés selon le type de réacteur

R4 : La pression haute est comprise entre 19,8 et 2,1 bars selon le type de réacteur

# III-Analyser le comportement fonctionnel du système

## III-1 Quels sont les classes

### Modèle Classique

-classe Capteur pour modéliser les capteurs

-classe CapteurPression, et classe CapteurTemperature des sous classes de capteurs pour modeliser les capteurs de pression et de température

-classe Controleur

-classe Environnement : l'environnement dirige le flux de contrôle

-Regulateur : pour modéliser les régulateurs

-RegulateurTemperature, RegulateurPression : modeliser les régulateurs de pression et de température

-SystemeRTP : classe qui configure le système

-Horloge : modélise l'horloge de base

-Application : pour faire intérargir les composants du système

## Modèle Concurrent

-classe Capteur pour modéliser les capteurs

-classe CapteurPression, et classe CapteurTemperature des sous classes de capteurs pour modeliser les capteurs de pression et de température

-classe Controleur

-classe Environnement : l'environnement dirige le flux de contrôle

-Cible

-ThreadDeBase : pour créer un thread générique

-TimeStamp

-Regulateur : pour modéliser les régulateurs

-RegulateurTemperature, RegulateurPression : modeliser les régulateurs de pression et de température

-SystemeRTP : classe qui configure le système

-Animation

## Modèle Temps Réel

-classe Capteur pour modéliser les capteurs

-classe CapteurPression, et classe CapteurTemperature des sous classes de capteurs pour modeliser les capteurs de pression et de température

-Controleur

-classe Environnement : l'environnement dirige le flux de contrôle

-RegulateurTemperature, RegulateurPression : modeliser les régulateurs de pression et de température

-SystemeRTP : classe qui configure le système

-Actionneur : pour modéliser le régulateur

-Application : pour faire intérargir les composants du système

# III-3 Production du modèle classique

## III-3-1 Diagramme Modèle classique (Voir Dossier)

### Classe Capteur

```
class Capteur

instance variables

protected ID          : nat; -- identité des capteurs

protected Type        : Controleur`typeActeur;  --type des capteurs

protected Valeur      : nat;   -- valeur mesurée par le capteur

protected Env         : Environnement;  -- environnement du capteur

operations

public GetID: () ==> nat

            GetID() ==

                        return ID;

public GetType: () ==> Controleur`typeActeur

            GetType() ==

                        return Type;

public ReadValeur: () ==> nat

            ReadValeur() ==

                        return Valeur;

public Action: () ==> ()

            Action() ==

                        is subclass responsibility

end Capteur
```

# Classe CapteurPression

```
class CapteurPression is subclass of Capteur

operations

public CapteurPression: nat * Controleur`typeActeur * nat ==> CapteurPression

        CapteurPression(id, type, val) ==

                (ID := id;

                Type := type;

                Valeur := val;

                );

public Action: () ==> ()

        Action() ==

                (Valeur := Application`env.ReadPress();

                );

end CapteurPression
```

# Classe CapteurTemperature

```
class CapteurTemperature is subclass of Capteur

--modelise un capteur de temperature

operations

public CapteurTemperature: nat * Controleur`typeActeur * nat ==> CapteurTemperature

        CapteurTemperature(id, type, val) ==

                (ID := id;

                Type := type;

                Valeur := val;

                );

-- action du CapteurTemperature:  "mesurer la température de l'environnement"

public Action: () ==> ()

        Action() ==

                (Valeur := Application`env.ReadTemp();

                );

end CapteurTemperatur
```

# Classe Horloge

```
class Horloge

instance variables

dateActuelle: nat := 0;

values

periode : nat = 1;

operations

public StepTime : () ==> ()

StepTime() ==

  dateActuelle:= dateActuelle + periode;

public GetTime : () ==> nat

GetTime() ==

  return dateActuelle;

end Horloge
```

# Classe Regulateur

```
class Regulateur

instance variables

protected ID      : nat;   -- identité

protected Type  : Controleur`typeActeur;  --type d'Regulateur

protected Corr   : Controleur`correction;   -- type de correction

protected Env    : Environnement;         -- environnement du regulateur

operations

public GetID: () ==> nat

            GetID() ==

                        return ID;

public GetType: () ==> Controleur`typeActeur

            GetType() ==

                        return Type;

public Action: () ==> ()

            Action() ==

                        is subclass responsibility

end Regulateur
```

# Classe RegulateurPression

```
class RegulateurPression is subclass of Regulateur

operations

public RegulateurPression: nat * Controleur`typeActeur ==> RegulateurPression
        RegulateurPression (id, type) ==
                (ID := id;
                Type := type;
                Corr := <NUL>
                );
        public Action: () ==> ()
        Action() ==
                (if (Corr = <INC>)    then Application`env.IncPress()
                elseif (Corr = <DEC>) then Application`env.DecPress();
                );
public SetCorrection: Controleur`correction ==> ()
        SetCorrection(cor) ==
                Corr := cor
pre (cor = <INC>) or (cor = <DEC>) or (cor = <NUL>);
public GetCorrection: () ==> Controleur`correction
        GetCorrection() ==
                return Corr;
end RegulateurPression
```

# Classe RegulateurTemperature

```
class RegulateurTemperature is subclass of Regulateu

operations

public RegulateurTemperature: nat * Controleur`typeActeur ==> RegulateurTemperature
        RegulateurTemperature (id, type) ==
                (ID := id;
                Type := type;
                Corr := <NUL>
                );
public Action: () ==> ()
        Action() ==
                (if (Corr = <INC>)    then Application`env.IncTemp()
                elseif (Corr = <DEC>) then Application`env.DecTemp();
                );
public SetCorrection: Controleur`correction ==> ()
        SetCorrection(cor) ==
                Corr := cor
pre (cor = <INC>) or (cor = <DEC>) or (cor = <NUL>);
public GetCorrection: () ==> Controleur`correction
        GetCorrection() ==
                return Corr
post (Corr = <INC>) or (Corr = <DEC>) or (Corr = <NUL>);
end RegulateurTemperature
```

# Classe SystemeRTP

```
class SystemeRTP

instance variables

public static  Pilote      : Controleur := new Controleur(8, 70);  -- un seul système de pilotage qui regule la temperature (autour de 8°) et la pression (autour de 70% de la pression nominale)

public static CapteurTemperature1     : CapteurTemperature := new CapteurTemperature(1, <CAPTEUR_TEMPERATURE>, 0);  -- un seul capteur de température

public static CapteurPression1            : CapteurPression := new CapteurPression(2, <CAPTEUR_PRESSION>, 0);  -- un seul capteur de pression

public static RegulateurTemperature1            : RegulateurTemperature := new RegulateurTemperature(3, <REGULATEUR_TEMPERATURE>); -- un seul regulateur de temperature

public static RegulateurPression1       : RegulateurPression := new RegulateurPression(4, <REGULATEUR_PRESSION>); -- un seul régulateur de pression

operations

public SystemeRTP: () ==> SystemeRTP

        SystemeRTP() ==

                (

                Pilote.AjouterActeur(CapteurTemperature1.GetID(), CapteurTemperature1.GetType());

                Pilote.AjouterActeur(CapteurPression1.GetID(), CapteurPression1.GetType());

                Pilote.AjouterActeur(RegulateurTemperature1.GetID(), RegulateurTemperature1.GetType());

                Pilote.AjouterActeur(RegulateurPression1.GetID(), RegulateurPression1.GetType());

                );

end SystemeRTP
```

# Les Classes Application , Contoleur , Environnment sont consultable dans le dossier fourni.

# III-4 Production du modèle Concurrent

## III-3-1 Diagramme Modèle Concurrent(Voir Dossier)

**Les Classes Capteur, Regulateur sont les même que dans le modèle classique.**

## Classe CapteurPression

```
class CapteurPression is subclass of Capteur, ThreadDeBase

instance variables
fini    : bool := false;

operations

public CapteurPression: nat * Controleur`typeActeur * nat * Cible * nat1 * bool ==> CapteurPression
CapteurPression(id, type, val, envir, p, isP) ==
 (ID := id;
  Type := type;
  Valeur := val;
  Env := envir;
  periode := p;
  estPeriodique := isP;
 );

public Finir: () ==> ()
Finir() ==
  fini := true;

public estFini: () ==> ()
estFini() ==
  skip;

-- Action du Capteur:  "mesurer le taux de pression nominale de l'environnement"
protected  Action: () ==> ()
        Action() ==
        (
                Valeur := Env.ReadPress();
 -- IO`print("\n*************************************************************************************************************");
 --IO`print("\n  COTE CAPTEUR PRESSION:    Pression(p)  du carburant  l'instant: d=  ");
 --IO`print(Animation`horloge.GetTime());  IO`print("\n");
 --IO`print(" est: t= ");
 --IO`print(Valeur);
 --IO`print("\n*************************************************************************************************************");

        )

sync
  per estFini => fini;


end CapteurPression
```

# Classe CapteurTemperature

```
class CapteurTemperature is subclass of Capteur, ThreadDeBase

instance variables

fini    : bool := false;

operations

public CapteurTemperature: nat * Controleur`typeActeur * nat * Cible * nat1 * bool ==> CapteurTemperature
CapteurTemperature (id, type, val, envir, p, estP) ==
 (ID := id;
  Type := type;
  Valeur := val;
  Env := envir;
  periode := p;
  estPeriodique := estP;
 );

public Finir: () ==> ()
Finir() ==
  fini := true;

public estFini: () ==> ()
estFini() ==
  skip;

protected Action: () ==> ()
Action() ==
  (Valeur := Env.ReadTemp();

 --IO`print("\n*************************************************************************************************");
 --IO`print("\n COTE CAPTEUR TEMPERATURE:     Temperature(t)  du carburant  l'instant: d=  ");
 --IO`print(Animation`horloge.GetTime());  IO`print("\n");
 --IO`print(" est: t= ");
 --IO`print(Valeur);
 --IO`print("\n*************************************************************************************************");

  )

sync

  per estFini => fini;

end CapteurTemperature
```

# Classe Cible

```
class Cible
instance variables
private envTemp  : nat;
private envPression : nat;

operations
public Cible: int * int  ==> Cible
Cible ( temperature0, pression0) ==
 (envTemp := temperature0;
  envPression := pression0;
 );
public SetTemp: nat ==> ()
SetTemp(t) ==
  envTemp := t;

public SetPress: nat ==> ()
SetPress(p) ==
  envPression:= p;

public IncTemp: () ==> ()
IncTemp() ==
  envTemp := envTemp + 1;

public DecTemp: () ==> ()
DecTemp() ==
  envTemp := envTemp - 1;

public IncPress: () ==> ()
IncPress() ==
  envPression := envPression + 1;

public DecPress: () ==> ()
DecPress() ==
  envPression := envPression - 1;

public ReadTemp: () ==> nat
ReadTemp() ==
  return envTemp;
public ReadPress: () ==> nat
ReadPress() ==
  return envPression;
sync
  mutex(IncTemp);
  mutex(DecTemp);
  mutex(SetTemp);
  mutex(ReadTemp, IncTemp, DecTemp, SetTemp);
  mutex(IncPress);
  mutex(DecPress);
  mutex(SetPress);
  mutex(ReadPress, IncPress, DecPress, SetPress);
end Cible
```

# Classe RegulateurPression

```
class RegulateurPression is subclass of Regulateur, ThreadDeBase
instance variables
fini    : bool := false;
operations
public RegulateurPression: nat * Controleur`typeActeur * Cible * nat1 * bool ==> RegulateurPression
RegulateurPression (id, type, envir, p, isP) ==
 (ID := id;
  Type := type;
  Corr := <NUL>;
  Env := envir;
  periode := p;
  estPeriodique := isP;
 );
public Finir: () ==> ()
Finir() ==
  fini := true;
public estFini: () ==> ()
estFini() ==
  skip;
protected Action: () ==> ()
        Action() ==
                (if (Corr = <INC>)    then Env.IncPress()
                elseif (Corr = <DEC>) then Env.DecPress()
                elseif (Corr = <NUL>) then skip;
         -- IO`print("\n*****************************************************************************************************************");
    --IO`print("\n   COTE REGULATEUR PRESSION:   Pression(p)  du carburant  l'instant: d=  ");
   -- IO`print(Animation`horloge.GetTime());  IO`print("\n");
   -- IO`print(" est: t= ");
   -- IO`print(Env.ReadPress());
   -- IO`print(" après la correction =  ");
   -- IO`print(Corr);
   -- IO`print("\n*****************************************************************************************************************");
        Corr := <NUL>;
        );
public SetCorrection: Controleur`correction ==> ()
        SetCorrection(cor) ==
                Corr := cor
pre (cor = <INC>) or (cor = <DEC>) or (cor = <NUL>);

public GetCorrection: () ==> Controleur`correction
        GetCorrection() ==
                return Corr;
-- (--World`horloge.RegisterThread();

--  while true
--  do
--   (if (GetCorr() = <OPEN>)
--    then (HA`Env.DecHumid();
--          HA`Env.DecTemp();
--         );
--    World`horloge.WaitRelative(5);--World`horloge.stepLength);
--   )
-- )
sync
  per estFini => fini;
end RegulateurPression
```

# Classe RegulateurTemperature

```
class RegulateurTemperature is subclass of Regulateur, ThreadDeBase

instance variables
fini    : bool := false;

operations

public RegulateurTemperature: nat * Controleur`typeActeur * Cible * nat1 * bool ==> RegulateurTemperature
RegulateurTemperature (id, type, envir, p, isP) ==
 (ID := id;
  Type := type;
  Corr := <NUL>;
  Env := envir;
  periode := p;
  estPeriodique := isP;
 );
public SetCorrection: Controleur`correction ==> ()
SetCorrection(cor) ==
  Corr := cor
pre (cor = <INC>) or (cor = <DEC>) or (cor = <NUL>);

public Finir: () ==> ()
Finir() ==
  fini := true;

public estFini: () ==> ()
estFini() ==
  skip;

protected Action: () ==> ()
        Action() ==
                (if (Corr = <INC>)    then Env.IncTemp()
                elseif (Corr = <DEC>) then Env.DecTemp()
    elseif (Corr = <NUL>) then skip;
    --IO`print("\n*****************************************************************************************************");
    --IO`print("\n COTE REGULATEUR TEMPERATURE:    Temperature  du carburant  l'instant: d=  ");
    --IO`print(Animation`horloge.GetTime());  IO`print("\n");
    --IO`print(" est: t= ");
    --IO`print(Env.ReadTemp());
    --IO`print(" après la correction:  ");
    --IO`print(Corr);
    --IO`print("\n*****************************************************************************************************");
    Corr := <NUL>;

                );

 sync
        per estFini => fini;
end RegulateurTemperature
```

# Classe SystemeRTP

```
class SystemeRTP
-- classe qui configure le système: Pilote + 2 Regulateurs + 2 capteurs
instance variables
public static laCible          : Cible := new Cible(10,71);
--crée des instances "public static" pour permettre l'accès à partir de toutes les classes du modèle
public static  Pilote   : Controleur := new Controleur(8, 70,3,true);
public static CapteurTemperature1      : CapteurTemperature := new CapteurTemperature(1, <CAPTEUR_TEMPERATURE>, 10, laCible, 3, true);  -- un seul capteur de température
public static CapteurPression1  : CapteurPression := new CapteurPression(2, <CAPTEUR_PRESSION>, 71, laCible, 3, true);  -- un seul capteur de pression
public static RegulateurTemperature1    : RegulateurTemperature := new RegulateurTemperature(3, <REGULATEUR_TEMPERATURE>, laCible, 5, true); -- un seul regulateur de
temperature
public static RegulateurPression1      : RegulateurPression := new RegulateurPression(4, <REGULATEUR_PRESSION>, laCible , 5, true ); -- un seul régulateur de pression

end SystemeRTP
```

# Classe ThreadDeBase

```
class ThreadDeBase
        -- pour créer un thread générique: il appelle opération action() laquelle est générique
        --                                - action de capteur
        --                                -action de régulateur
        --                                -action de l'environnement
instance variables
--période du thread
protected periode : nat1 := 1;
protected estPeriodique : bool := true;


operations

protected ThreadDeBase : () ==> ThreadDeBase
ThreadDeBase() ==
 (Animation`horloge.EnregisterThread(self);
  if(not Animation`horloge.EstInitialisé())
  then start(self);
 );

protected Action : () ==> ()
Action() ==
  is subclass responsibility

thread
 (if estPeriodique
  then (while true
        do
         (Action();
         Animation`horloge.Wait(periode);
         )
       )
  else (Action();
       Animation`horloge.Wait(0);
       Animation`horloge.DesenregistrerThread();
       )
 );

end ThreadDeBase
```

# Les Classes Animation , Contoleur , Environnment, TimeStamp sont consultable dans le dossier fourni.

# III-5 Production du modèle Temps Réel

## III-3-1 Diagramme Modèle Temps Réel(Voir Dossier)

### Classe Capteur

```
class Capteur

instance variables

  protected ID    : nat;
  protected Type  : Controleur`typeActeur;
  protected Valeur : int;

operations

public GetID: () ==> nat
GetID() ==
  return ID;

public GetType: () ==> Controleur`typeActeur
GetType() ==
  return Type;

public ReadValeur: () ==> int
ReadValeur() ==
  return Valeur;

public Action: () ==> ()
Action() ==
  is subclass responsibility

end Capteur
```

## Classe CapteurPression

```
class CapteurPression is subclass of Capteur

instance variables

  fini : bool := false;

operations

public CapteurPression: nat * Controleur`typeActeur * nat ==> CapteurPression
CapteurPression (id, type, val) ==
 (ID := id;
  Type := type;
  Valeur := val;
 );

public Action: () ==> ()
Action () ==
  Valeur := Application`env.ReadPression() ;

public estFini: () ==> ()
estFini() ==
  skip;

sync
  --mutex(Action);       -- inutile!
  per estFini => fini;

thread
-- periode du thread (periode, jitter, delay, offset)
periodic(1000E6,0,0,0) (Action)

end CapteurPression
```

## Classe CapteurTemperature

```
class CapteurTemperature is subclass of Capteur

instance variables
  fini : bool := false;
operations

public CapteurTemperature: nat * Controleur`typeActeur * int ==> CapteurTemperature
CapteurTemperature (id, type, val) ==
 (ID := id;
  Type := type;
  Valeur:= val;
 );

public Action: () ==> ()
Action () ==
(
  Valeur := Application`env.ReadTemp();
 -- IO`print("\n valeur := Application`env.ReadTemp()");
 --IO`print(Valeur);
);

public estFini: () ==> ()
estFini() ==
  skip;

sync
  --mutex(Action); -- inutile!
  per estFini => fini;

thread
-- period of thread (period, jitter, delay, offset)
periodic(1000E6,0,0,0) (Action)

end CapteurTemperature
```

## Classe RegulateurPression

```
class RegulateurPression is subclass of Actionneur
instance variables

  fini : bool := false;
operations

public RegulateurPression: nat *Controleur`typeActeur ==> RegulateurPression
RegulateurPression (id, type) ==
 (ID := id;
  Type := type;
  Corr := <NUL>;
 );

public Action: () ==> ()
Action() ==
 (dcl pressionCorr: Controleur`correction := GetCorrection();
  if (pressionCorr = <INC>)
  then Application`env.IncPression()
  elseif (pressionCorr  = <DEC>)
  then Application`env.DecPression()
  elseif(pressionCorr = <NUL>)
  then skip;
  pressionCorr := <NUL>;
 );

--action asynchrone
async public SetCorrection: Controleur`correction ==> ()
SetCorrection(cor) ==
  Corr := cor
pre (cor = <DEC>) or (cor = <NUL> ) or (cor = <INC> );

public GetCorrection: () ==> Controleur`correction
GetCorrection() ==
  return Corr
post (Corr = <DEC>) or (Corr = <NUL> ) or (Corr = <INC> );

public estFini: () ==> ()
estFini() ==
  skip;

sync
  --mutex(Action);
  per estFini=> fini;
  mutex(SetCorrection, GetCorrection);

thread

-- periode du  thread (periode, jitter, delay, offset)
periodic(1000E6,0,0,0) (Action)

end RegulateurPression
```

## Classe RegulateurTemperature

```
class RegulateurTemperature is subclass of Actionneur

instance variables

  fini : bool := false;

operations

public RegulateurTemperature: nat * Controleur`typeActeur  ==> RegulateurTemperature
RegulateurTemperature (id, type) ==
 (ID := id;
  Type := type;
  Corr := <NUL>;
 );

public Action: () ==> ()
Action() ==
 (dcl tempCorr: Controleur`correction := GetCorrection();
  if (tempCorr = <INC>)
  then Application`env.IncTemp()
  elseif (tempCorr = <DEC>)
  then Application`env.DecTemp()
  elseif(tempCorr = <NUL>)
  then skip;
  tempCorr := <NUL>;
 );

async public SetCorrection: Controleur`correction ==> ()
SetCorrection(cor) ==
  --cycles(1E3)
  Corr := cor
pre (cor = <INC>) or (cor = <DEC>) or (cor = <NUL>);

public GetCorrection: () ==> Controleur`correction
GetCorrection() ==
  return Corr
post (Corr = <INC>) or (Corr = <DEC>) or (Corr = <NUL>);

public estFini: () ==> ()
estFini() ==
  skip;

sync
  --mutex(Action); -- à  prévoir
  per estFini=> fini;
  mutex(SetCorrection, GetCorrection);

thread
-- periode du thread (period, jitter, delay, offset)
periodic(1000E6,0,0,0) (Action)

end RegulateurTemperature
```

## Classe SystemeRTP

```
system SystemeRTP

instance variables

  ---1 cpu controleur
             cpu1 : CPU := new CPU(<FCFS>, 1E6);
  ---2 cpu capteurs: un par chaque capteur
             cpu2 : CPU := new CPU(<FCFS>, 1E6);
             cpu5 : CPU := new CPU(<FCFS>, 1E6);
  ---2 cpu régulateurs: un pour chaque  régulateur
             cpu3 : CPU := new CPU(<FCFS>, 1E6);
             cpu4 : CPU := new CPU(<FCFS>, 1E6);

  -- bus connexion controleur avec les capteurs et les régulateurs
  bus1 : BUS := new BUS(<FCFS>, 1E3, {cpu1, cpu2, cpu3, cpu4, cpu5 });

  public static Pilote : Controleur:= new Controleur (8, 70);
  public static CapteurTemperature1: CapteurTemperature := new CapteurTemperature(1, <CAPTEUR_TEMPERATURE>, 20);
  public static CapteurPression1 : CapteurPression := new CapteurPression(2, <CAPTEUR_PRESSION>, 75);
  public static RegulateurTemperature1 :  RegulateurTemperature := new RegulateurTemperature(3, <REGULATEUR_TEMPERATURE>);
  public static RegulateurPression1 : RegulateurPression := new RegulateurPression(4, <REGULATEUR_PRESSION>);

--
-- Operations definition section
operations

public SystemeRTP: () ==> SystemeRTP
SystemeRTP() ==
 -- déploiement des threads sur la plate forme cpu comme suit:
         (
-- on déploie le Pilote sur cpu1
         cpu1.deploy(Pilote );
-- on déploie CapteurTemperature1 sur cpu2
         cpu2.deploy(CapteurTemperature1);
-- on déploie RegulateurPression1 sur cpu5
         cpu5.deploy(CapteurPression1);
-- on déploie RegulateurTemperature1 sur cpu3
         cpu3.deploy(RegulateurTemperature1);
-- on déploie RegulateurPression1 sur cpu4
         cpu4.deploy(RegulateurPression1 );
          );

end SystemeRTP
```

## Classe Actionneur

```
class Actionneur
instance variables
  protected ID   : nat;
  protected Type : Controleur`typeActeur;
  protected Corr : Controleur`correction;

operations
public GetID: () ==> nat
GetID() ==
  return ID;

public GetType: () ==> Controleur`typeActeur
GetType() ==
  return Type;

public Action: () ==> ()
Action() ==
  is subclass responsibility
end Actionneur
```

## Classe Application

```
class Application

instance variables
public static env : [Environnement] := nil;
--env: := new Environnement("flotStimuli.txt");

operations
public Application: () ==> Application
Application() ==
 (env := new Environnement("flotSimuli.txt");
  SystemeRTP`Pilote.AjouterActeur(SystemeRTP`CapteurTemperature1.GetID(),SystemeRTP`CapteurTemperature1.GetType());
  SystemeRTP`Pilote.AjouterActeur(SystemeRTP`CapteurPression1.GetID(),SystemeRTP`CapteurPression1.GetType());
  SystemeRTP`Pilote.AjouterActeur(SystemeRTP`RegulateurTemperature1.GetID(),SystemeRTP`RegulateurTemperature1.GetType());
  SystemeRTP`Pilote.AjouterActeur(SystemeRTP`RegulateurPression1.GetID(),SystemeRTP`RegulateurPression1.GetType());

  start(SystemeRTP`CapteurTemperature1);
  start(SystemeRTP`CapteurPression1);
  start(SystemeRTP`RegulateurTemperature1);
  start(SystemeRTP`RegulateurPression1);
  start(SystemeRTP`Pilote);
 );

public Simuler: () ==> ()
Simuler() ==
 (-- lancer (start thread)  environment
  start(env);
  --attendre que l'environnement ait fini de produire des stimuli
  env.estFini();
  -- désactiver le pilote (controleur)
 SystemeRTP`Pilote.Finir();
 );
end Application
```