

AGREGATION D'INFORMATIQUE – Session 2001

Epreuve d'application
Option : Informatique des systèmes industriels

Eléments de correction

GESTION INFORMATISEE DES ACTIVITES D'UNE SOCIETE DE TRANSPORT D'HYDROCARBURES PAR PIPELINES

1^{ère} partie : Etude du sous-système de commande de la station de Radès

Spécification

Question 1.1

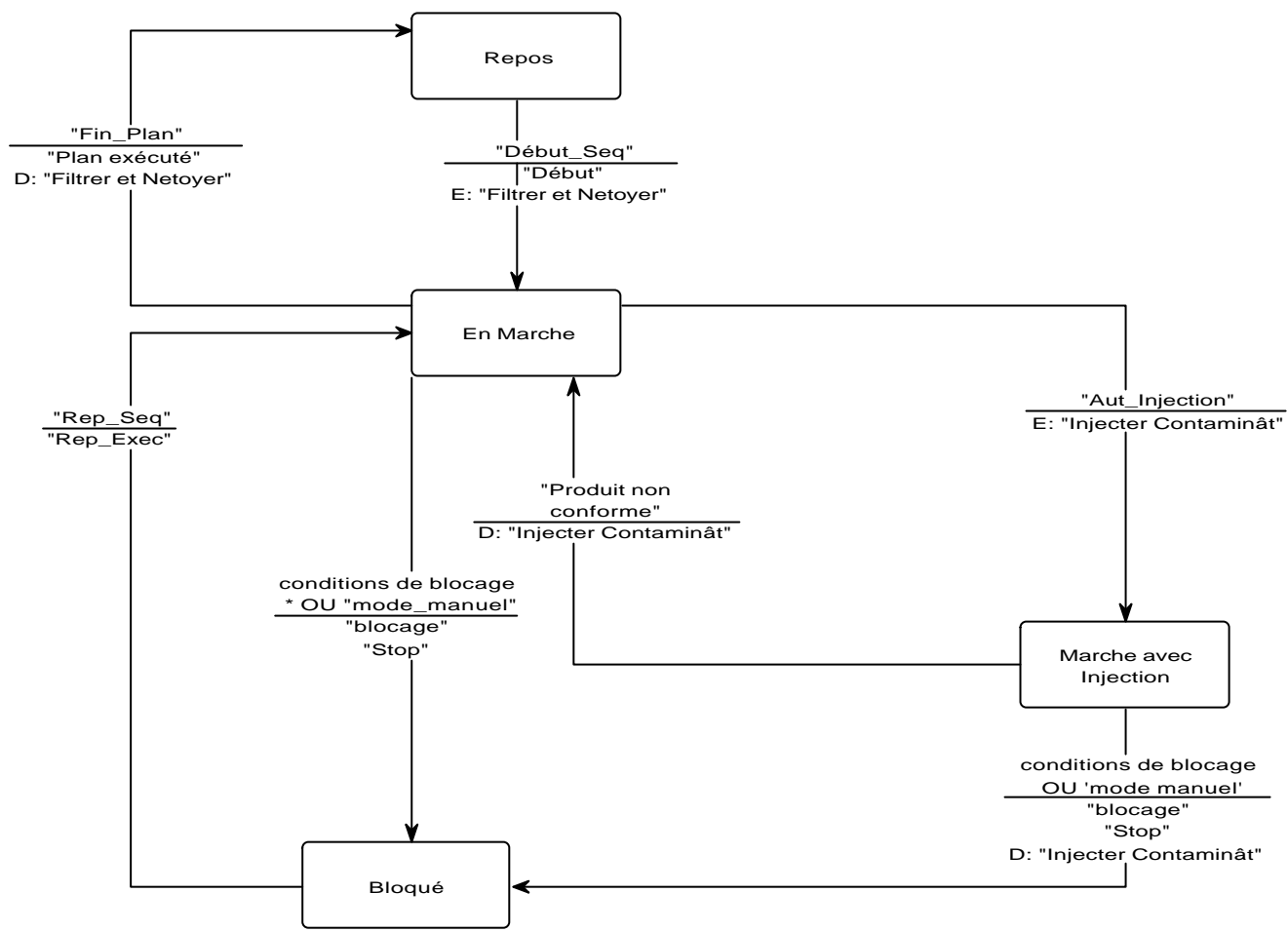
Dans le schéma de transformation ST3.0 "Gérer mode Auto" l'événement "Aut_Injection" (autorisant le début de l'injection du contaminât) est directement traité par la transformation 3.4 "Injecter Contaminât".

Reprendre le diagramme états-transitions spécifiant la transformation de contrôle "contrôler mode auto" de façon à ce que ce soit elle qui traite l'événement "Aut_Injection".

Il faut que la transformation de contrôle prenne en compte l'événement : "Aut_injection" pour activer la transformation 3.4 "Injecter Contaminât" et l'événement "Produit non conforme" indiquant le changement de type de produit pour désactiver la transformation 3.4 "Injecter Contaminât".

La transformation de données 3.4 "Injecter Contaminât" peut aussi gérer elle même par la biais de sa transformation de contrôle l'arrêt de l'injection du contaminât.

Une proposition possible du nouveau diagramme états-transitions est donné par la figure suivante.

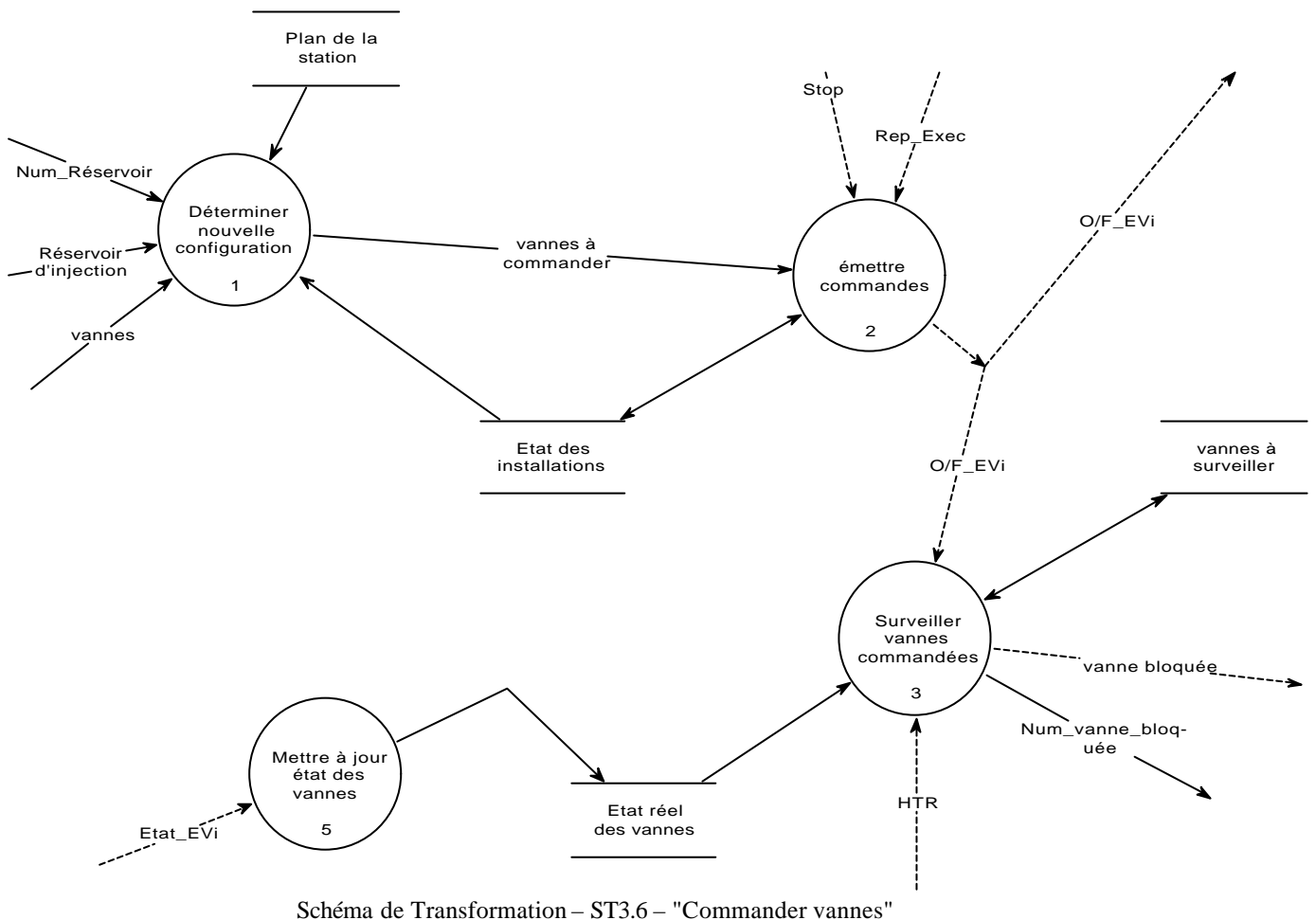


* conditions de blocage =
 VS_ouverte OU Ri_plein OU vanne bloquée OU
 suppression OU Surchauffe EP_i OU filtres bloqués

Diagramme Etats-transitions de la Transformation 3.2 – "contrôler mode auto"

Question 1.2

La transformation "commander vannes" du schéma de transformation ST3.0 n'est pas une transformation primitive. Compléter sa spécification en fournissant le niveau suivant de sa décomposition et les spécifications des transformations terminales obtenues.



Nouvelles entrées au dictionnaire de données :

Name: Etat des installations

Type: Store

Bnf: 0{Num_vanne + dernière_commande}n

@dernière_commande = [ouvrir | fermer]

Comment: la liste de toutes les vannes de la station avec la dernière commande qui lui a été envoyée

Name: Etat réel des vannes

Type: Store

Bnf: 0{Num_vanne + Etat_vanne}n

@Etat_vanne = [ouverte | fermée]

Comment: L'état réel (mesuré) de toutes les vannes de la station

Name: Plan de la station

Type: Store

Comment: pour chaque réservoir, la liste des vannes qui permettent de l'alimenter.

Name: vannes à surveiller

Type: Store

Bnf: 0{Num_vanne + Etat_visé + compteur}n

@Etat_visé = [ouverte | fermée]

Comment: c'est une liste comprenant les numéros des vannes auxquelles une commande a été envoyée, l'état à atteindre (ouverte ou fermée et un timer (compteur) initialisé à 6s.

Mini-spécifications procédurale :

Surveiller vannes commandées :

SI O/F_Evi reçu ALORS

AJOUTER (Num_vanne+ Etat_visé+compteur=6s) pour Evi au stock "vannes à surveiller"

FIN SI

SI HTR reçu ALORS

ENLEVER de "vannes à surveiller" les vannes qui ont atteint leur Etat visé selon "Etat réel des vannes".

DECREMENTER compteur pour chaque vanne de "vannes à surveiller"

Si compteur = 0 ALORS

EMETTRE "vanne bloquée" et "num_vanne_bloquée"

FINSI

FINSI

Commentaire : "Surveiller vannes commandées" compare l'état réel des vannes avec l'état des vannes à surveiller pour voir si l'une des vannes n'aurait pas exécuté la commande qui lui a été envoyée. Dans ce cas elle signale le blocage de cette vanne.

Mettre à jour état des vannes :

SI Etat_Evi reçu ALORS

Mettre à jour "Etat réel des vannes" en fonction de Etat_Evi

FINSI

Commentaire : "Mettre à jour état des vannes" met à jour le stock de données "Etat réel des vannes" en fonction des comptes rendu reçus du procédé.

Emettre commandes :

GENERER "O/F_Evi" en fonction de "vannes à commander" reçue

Si "Stop" reçue

EMETTRE F_Evi pour toutes les vannes

FINSI

SI "Rep_Exec" reçue

EMETTRE O/F_Evi selon "Etat des installation" mémorisé

FINSI

Commentaire : "Emettre les commandes" doit envoyer les commandes aux différentes vannes à ouvrir ou à fermer et doit mettre à jour le stock de données "Etat projeté des vannes" qui contient l'état des différentes vannes selon les commandes issues par le système (différent de l'état réel des vannes) ainsi que l'instant d'émission de la commande". A la réception de l'ordre "top" elle doit fermer toutes les vannes et à la réception de l'ordre "Rep_Exec" elle doit remettre les vannes à la même configuration précédant l'arrêt".

Déterminer nouvelle configuration :

SI "Num_réservoir" reçu

EXTRAIRE de "Plan de la station" la liste des vannes à ouvrir

COMPARER avec "Etat des installations"

EMETTRE les "vannes à commander" en ouverture ou en fermeture

FINSI

SI "vannes" reçu

EMETTRE les "vannes à commander" en ouverture ou en fermeture correspondant.

FINSI

SI "Réservoir d'injection" reçu

EXTRAIRE de "Plan de la station" la liste des vannes à ouvrir

COMPARER avec "Etat des installations"

EMETTRE les "vannes à commander" en ouverture ou en fermeture

FINSI

Commentaire : "Déterminer nouvelle configuration" doit déterminer quelles sont les vannes à ouvrir ou à fermer en fonction des demandes qui lui parviennent ("vannes", "réservoir d'injection" ou "Num_réservoir"). Elle possède une connaissance détaillée des différentes vannes de la station par le biais du stock de données "Plan de la station". Elle doit s'assurer que les commandes n'ont pas été déjà faites en vérifiant dans le stock de données "Etat des installations". Elle émet La liste des vannes à commander en ouverture ou en fermeture

D'autres solutions peuvent être envisagées.

Question 1.3

La décomposition de la transformation "Exécuter le plan" du schéma de transformation ST3.0 fait apparaître les 4 transformations suivantes :

- "Suivre phases".
- "Mesurer Quantité Produit".
- "Informer Qtté livrées".
- "Traiter contaminât".

a- Sur le document réponse N°1, compléter le schéma de transformation ST 3.1 détaillant la transformation "Exécuter le plan" en matérialisant les flots de contrôle.

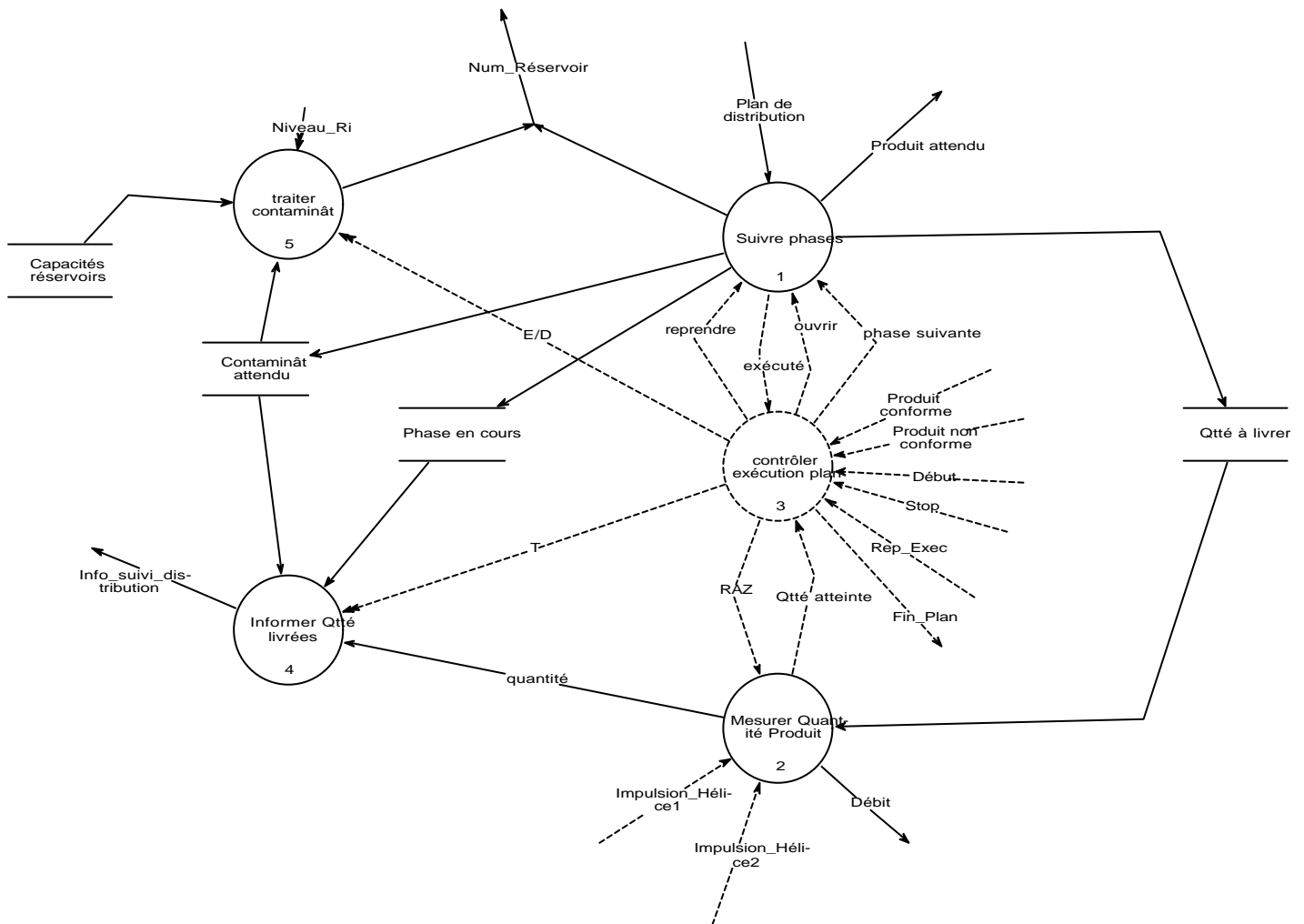


Schéma de Transformation – ST3.1.3 – "Exécuter le plan"

b- Sur le document réponse N°2, compléter le diagramme états-transitions spécifiant la transformation de contrôle "contrôler exécution plan".

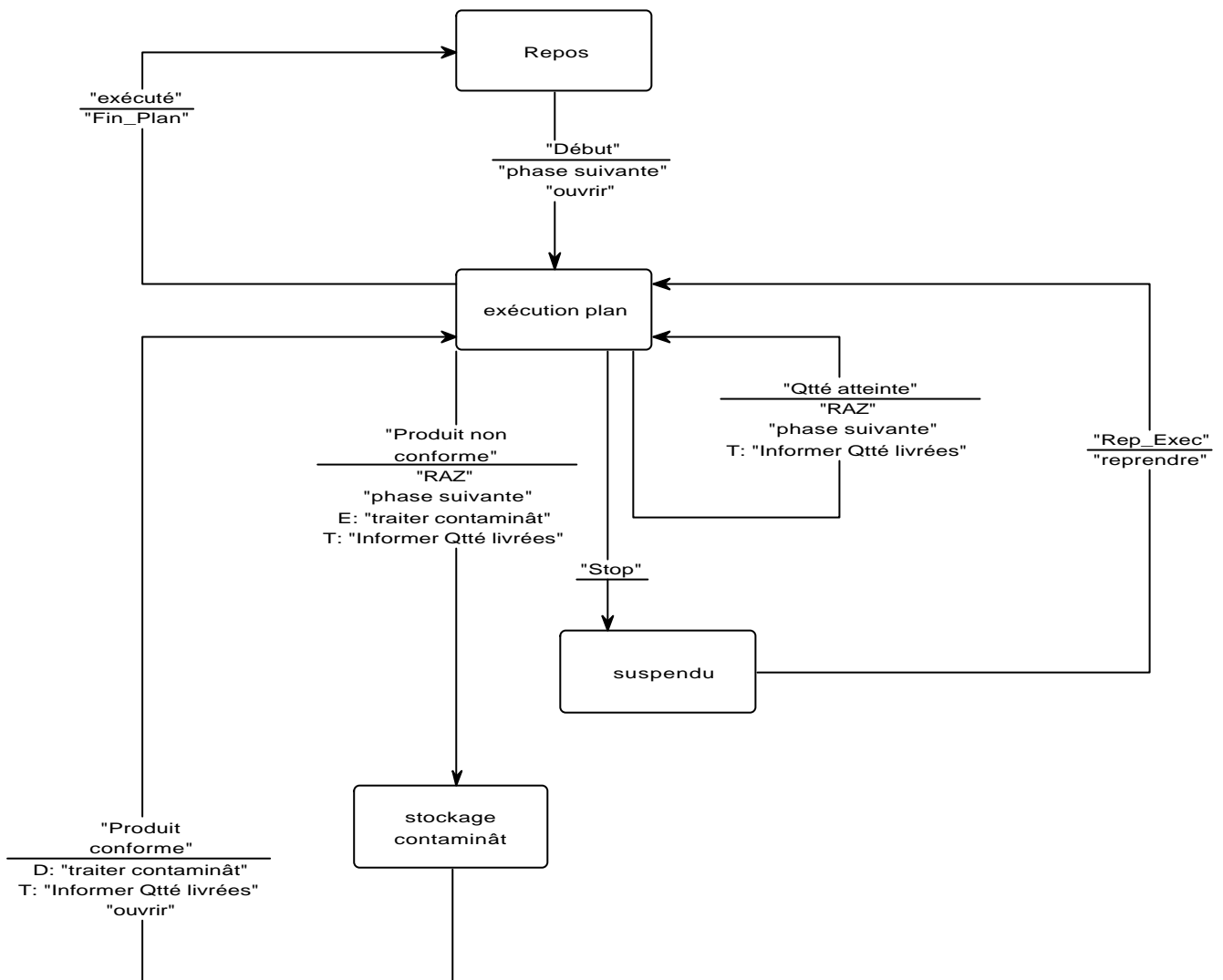


Diagramme Etats-transitions de la Transformation 3.2 - "contrôler exécution plan"

Conception

Question 1.4

Identifier les tâches du SCR et préciser leurs natures (logicielle ou matérielle) et leurs priorités respectives en justifiant vos choix.

L'identification des tâches se fait en regardant les différents traitements concurrents qui sont effectués ainsi que la prise en compte des différents événements "urgents" provenant du procédé. Certaines transformations de données seront décomposées en plusieurs tâches, d'autres transformations de données seront regroupés en 1 seule tâche.

Les tâches peuvent être réparties en groupes de priorité GPi (Proupe de Priorité i) , l'ordre des groupes doit être respecté. Au sein d'un groupe, l'ordre de priorité n'est pas impératif.

Une décomposition possible en tâches est la suivante :

- Timer, tâche matérielle activée par l'horloge GP1.
- La transformation de données commander les vannes sera réalisée à l'aide de 2 tâches: "Surveiller les vannes" qui est une tâche matérielle activée par le Timer (GP2) et une seconde tâche "gérer les vannes" qui est une tâche logicielle(GP4).
- Gérer les communications sera implémentée à l'aide d'une tâche matérielle activée par l'interruption de la carte réseau (GP3).
- "Filtrer et nettoyer" tâche matérielle activée par l'un des signaux lui parvenant (GP2).
- La machine à état fini de "Gérer les activités" sera implémentée par la tâche de fond.
- "Contrôler le mode automatique" sera implémenté par une tâche matérielle activée par l'un des signaux lui parvenant (GP2).
- "Analyser qualité produit" implémentée par une tâche logicielle (GP4).
- "Exécuter le plan" sera implémentée par 2 tâches : "mesurer quantité produit" une tâche matérielle (GP2) et le reste des traitement par une tâche logicielle (GP4).
- " Injecter Contaminât" tâche matérielle déclenchée par Aut_injection (GP3).
- "Valider commande client" tâche logicielle (GP5)
- "Mettre à jour état du système" sera décomposée en 2 parties : la partie qui gère les signaux provenant du système sera intégrée à la tâche "contrôler mode auto" et le reste par une autre tâche logicielle (GP5).

Implémentation

Question 1.5

L'étape de conception nous amène à faire correspondre aux transformations "Gérer communications", "valider commande client" et "mettre à jour état système" trois tâches concurrentes. Celles-ci seront implémentées par des processus UNIX.

- a- *Montrer que l'accès au stockage "Etat du système" (du schéma préliminaire ST0) peut être conflictuel. Proposer une solution à ce problème en utilisant les primitives UNIX.*

- Le stockage « Etat du système » est une information partagée par au moins deux processus : « Valider_commande_de_client » et « Mettre_à-jour_état_système ». Donc, c'est une ressource critique dont l'accès constitue une section critique. Ainsi, l'accès doit se faire en exclusion mutuelle.
- Une solution à ce problème peut reposer sur l'utilisation de l'outils de synchronisation « Sémaphores ».
- Le code en C sous Unix :

```
$ cat utile.c  
#include <stdio.h>  
#include <sys/types.h>  
#include <sys/sem.h>  
#include <sys/ipc.h>
```

```

#define SEMKEY 9001 /* clé associé au semaphore */

/* fonction réalisant une operation sur un semaphore */
static void semcall(sid, op)
int sid, op;
{
struct sembuf sb;
sb.sem_num = 0;
sb.sem_op = op;
sb.sem_flg = 0;
semop(sid, &sb, 1);
}

/* Opération P du semaphore */
void P(sid)
int sid;
{ semcall(sid, -1); }

/* Opération V du semaphore */
void V(sid)
int sid;
{ semcall(sid, 1); }

```

\$ cat application.c

```

#include <utile.c>
main() {
/* structure de contrôle sémaphore */
int UnSem ;
union semun {
    int val ;
    struct sem_ds *buf ;
    ushort *array;
} semctl_arg;

/* création de l'identificateur associé au clé pour le
sémaphore */
UnSem = semget ( (key_t) SEMKEY,1, PERM|IPC_CREAT) ;

/* Initialisation du semaphore */
semctl_arg.val = 0 ;
semctl(UnSem, 0, SETVAL, semctl_arg) ;

... /* traitement non critique */

P(UnSem) ;

... /* section critique */

V(UnSem) ;

```

```

...    /* traitement non critique */
}

```

- b- Comment implémenter la communication entre les 2 processus "Gérer communications" et "valider commande client" ? Pour chacun de ces deux processus, écrire la partie du code qui implémente en C ce mode de communication.

Les deux processus en question n'ont pas forcément un lien de parenté d'où l'inadéquation des tubes ordinaires pour les faire communiquer. La communication peut se faire par tubes nommés.

```

$ cat StructDonnees.h

```

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
#define TRUE 1
#define FALSE 0
#define Command "queue1"
#define Answer  "queue2"
struct Requete { int com;
                 int pidC; }
struct Validation {int pids;
                  int val; }

```

```

$cat ValiderComClient.c

```

```

#include "StructDonnees.h"
void HandReveil(int sig) {return; }
void halt(int sig) { unlink(Command); unlink(Answer);
exit(2); }

main() {
int dReq, dValid;
int sig;
sigset_t ens, ensvide;
struct Requete Req;
struct Validation Valid;
mode_t mode = S_IRUSR | S_IRGRP | S_IROTH | S_IWUSR | S_IWGRP |
S_IWOTH;

/* creation des tubes nommés */
if mkfifo(Command,mode)==-1 || mkfifo(Answer,mode)==-1
    {fprintf(stderr, "échec de création des tubes nommés \n");
exit(3) ; }

/* ouverture des tubes nommés */
dReq=open(Command,O_RDONLY) ;
dValid=open(Answer, O_WRONLY);
if ((dReq==-1) || (dValid==-1))

```

```

        {fprintf(stderr, "échec d'ouverture des tubes nommés \n");
exit(4) ; }

/* installation des handlers */
for (sig=1; sig <= 31 ; sig++) signal(sig, halt);
signal(SIGUSR1, HandReveil);

Valid.pidS=getpid() ;
sigemtyset(&ensvide) ; sigemtyset(&ens) ;
sigaddset(&ens , SIGUSR1) ;
sigprocmask(SIG_SETMASK, &ens, NULL) ; /* masquage de SIGUSR1 */

while ( TRUE ) {
    if (read(dReq, (void *)&Req, sizeof(struct Requete)) <= 0)
        { close(dReq); dReq=open(Command,O_RDONLY); continue; }

    /*construction de la réponse */
    ...
    Valid.val = ???

    /* envoi de la réponse */
    if (write(dValid, (void *)&Valid, sizeof(struct Validation)) ==
1)
        {perror("write\n"); halt(); }

    /* envoi d'un signal au client GererCommunication */
    kill(Req.pidC, SIGUSR1) ; sigsuspend(&ensvide) ;
    }
}

```

\$cat GererCommunication.c

```

#include "StructDonnees.h"
void HandReveil(int sig) {return; }

main() {
int dReq, dValid;
int sig;
sigset_t ens, ensvide;
struct Requete Req;
struct Validation Valid;

/* ouverture des tubes nommés */
dReq=open(Command,O_WRONLY) ;
dValid=open(Answer, O_RDONLY);
if ((dReq==-1) || (dValid==-1))
    {fprintf(stderr, "échec d'ouverture des tubes nommés \n");
exit(4) ; }

signal(SIGUSR1, HandReveil);

Req.pidC=getpid() ;

```

```

sigemtyset(&ensvide) ; sigemtyset(&ens) ;
sigaddset(&ens , SIGUSR1) ;
sigprocmask(SIG_SETMASK, &ens, NULL) ; /* masquage de SIGUSR1 */

if (write(dReq, (void *)&Req, sizeof(struct Requete)) == -1)
    { perror("write\n"); exit(5); }
/* attente de la validation */
sigsuspend(&ensvide) ;

if (read(dValid, (void *)&Valid, sizeof(struct Validation)) <=0)
    { fprintf(stderr, "problème de lecture \n"); exit(6);}
kill(Valid.pids, SIGUSR1) ;

/*suite du traitement : exploitation de la validation */
...
}

```

Question 1.6

De même, l'étape de conception nous amène à faire correspondre à la transformation "Analyser qualité produit" une tâche qui sera implémentée par un processus UNIX. Ce processus doit se déclencher toutes les 3 secondes. Implémenter ce mécanisme d'interruption à l'aide des primitives UNIX.

L'implémentation du processus « AnalyserQualitéProduit » peut se faire en s'appuyant sur le mécanisme d'interruption (déclenchement toute les 3 secondes). Pour cela nous pouvons utiliser les signaux UNIX. :

\$cat AnaQualProd.c

```

#include <stdio.h>
#include<unistd.h>
#include <signal.h>
#define TEMPS 3
#define TRUE 1
void handler(int sig) {
printf("Analyse de la qualité du produit en cours \n");
    ... /* traitement d'analyse */

alarm(TEMPS) ; /* Armement de l'horloge*/
exit(0) ;
}

main() {
signal(SIGALRM, handler) ; /* installation du handler */
alarm(TEMPS) ; /* Armement de l'horloge*/
while (TRUE)
{
    ...
    /* traitemnt principal */
    ...
}
}

```

2^{ème} partie : Etude de l'application de gestion des alarmes du SSLR

Question 2.1

Quel est la relation entre l'objet capteur, un capteur de pression et le capteur de pression de la marque X (Capteur_Pression_X).

Capteur, Capteur de pression et capteur de pression de la marque X vont donner naissance à trois classes qui seront organisés selon une hiérarchie de spécialisation: Capteur_Pression est une sous-classe de Capteur et Capteur_Pression_X est une sous-classe de Capteur_Pression.

Question 2.2

En adoptant l'approche HOOD, il est demandé de fournir les ODS (Object Description Skeleton dont la syntaxe est fournie en annexe A4) des objets Planificateur, Capteur_Pression_X et Afficheur.

OBJECT Capteur_Pression_X IS

DESCRIPTION

Le capteur de pression X réalise sur demande le contrôle qui lui est assigné: il lit la pression et vérifie que cette pression se trouve dans l'intervalle autorisé. Si c'est le cas, il construit un message "Rien à Signaler", sinon il construit le message d'alarme approprié, il précise lors de cette construction la priorité de cette alarme.

Implementation or synchronisation constraints

Néant

PROVIDED INTERFACE

TYPES

Néant

DATA

Néant

OPERATIONS

EffectuerCtrl

EXCEPTIONS

Néant

REQUIRED INTERFACE

USED OBJECTS

MessageAlarme, MessageRAS

TYPES

Néant

DATA

Néant

OPERATIONS

MessageAlarme.Creer(Msg, priorite)

MessageRAS.Creer

EXCEPTIONS

Néant

INTERNALS

INCLUDED OBJECTS

TYPES

DATA

Minima, Maxima : Real;
PrioriteBasse, PrioriteHaute : Integer

OPERATIONS

Déclaration des opérations internes (syntaxe Ada)

OBJECT CONTROL STRUCTURE

OPERATION CONTROL STRUCTURE

```
Pression:=Lire;  
If Pression < Minima then  
    Resultat:=MessageAlarme.Creer("Pression Basse", PrioriteBasse)  
Elseif Pression > Maxima then  
    Resultat:=MessageAlarme.Creer("Pression Haute", PrioriteHaute)  
Else MessageRAS.Créer;
```

END Capteur_Pression_X

OBJECT Afficheur IS

DESCRIPTION

Gère la liste d'attente des messages: Ajoute les messages dans la liste selon leurs priorités et affiche les messages en attente.

Implementation or synchronisation constraints

PROVIDED INTERFACE

TYPES

Message

DATA

OPERATIONS

Poster(Msg : Message)

EXCEPTIONS

QueuePleine

REQUIRED INTERFACE

USED OBJECTS

TYPES

DATA

OPERATIONS

EXCEPTIONS

INTERNALS

INCLUDED OBJECTS

TYPES

DATA

OPERATIONS

AjouterMsg

RetirerMsg

OBJECT CONTROL STRUCTURE

Loop

Select

Accept Poster(Msg : Message) do begin

AjouterMsg(Msg)

-- Ajoute le message à la file selon sa priorité

end Poster

else RetirerMsg -- Retirer le premier message de la liste et l'afficher

OPERATION CONTROL STRUCTURE

* AjouterMsg(Msg : Message)

Insertion triée dans une liste chaînée de Messages sur la base de
Message.Priorité.

* RetirerMsg

Suppression puis affichage du message en tête de liste.

END Afficheur

OBJECT Planificateur IS

DESCRIPTION

Demande cycliquement à chacun des capteurs inscrits d'effectuer sa procédure
de contrôle.

Implementation or synchronisation constraints

PROVIDED INTERFACE

TYPES

DATA

OPERATIONS

EXCEPTIONS

REQUIRED INTERFACE

USED OBJECTS

Capteurs, Afficheur

TYPES

Afficheur.Message

Capteurs.Capteur

DATA

OPERATIONS

Capteurs.EffectuerCtrl(Cptr : Capteur)
Afficheur.Poster(Msg : Message)

EXCEPTIONS

INTERNALS

INCLUDED OBJECTS

TYPES

DATA

LCptr : Liste de Capteurs

OPERATIONS

Inscrire(Cptr : Capteur)
Planifier

OBJECT CONTROL STRUCTURE

OPERATION CONTROL STRUCTURE

```
*Inscrire(Cptr : Capteur)
    -- Rajouter le capteur à la liste LCptr, la liste est triée par rapport au
    -- cycle et à la priorité du capteur
* Planifier
    Courant:=Tete(LCptr)
    Loop
        Delay(DureeAttente(courant))
        Msg:=Capteur.effectuerCtrl(Courant)
        Afficheur.Poster(Msg)
        Courant:=Suivant(Courant,LCptr)
        If Courant=nil then
            Courant:=Tete(LCptr)
        End if
    End loop
```

END Planificateur

Question 2.3

Donner en C++, la définition de l'interface de la classe Capteur.

```
class Capteur {
public:
    virtual Message *EffectuerCtrl()=0;
    unsigned short GetPriorite()
    unsigned long GetPeriode()
Private:
    unsigned short Priorite;
    unsigned long Periode;
}
```

Question 2.4

Donner en C++, la définition complète (Interface + Implantation) de la classe *Planificateur*. On utilisera pour cela la bibliothèque dont un extrait est fourni en annexe A5 (on ne se préoccupera pas des problèmes de synchronisation entre objets).

```
class Planificateur {
public:
    void Inscrire(Capteur *Cptr);
    void Planifier;
private:
    ListDC<unsigned short,unsigned long,Capteur> LesCapteurs;
    Afficheur Aff;
}

void Planificateur::Inscrire(Capteur *Cptr) {
    LesCapteurs.Insere(Cptr->GetPriorite(),Cptr->GetPeriode(),Cptr);
}

void Planificateur::Planifier() {
    InitBaseTemps();

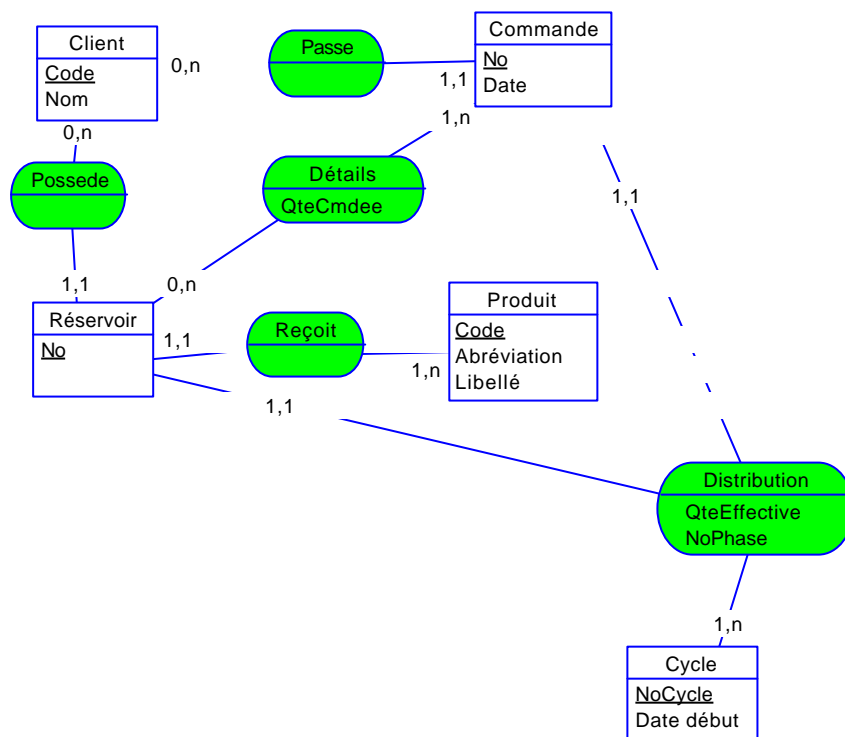
    ItListDC< unsigned short,unsigned long,Capteur> It(LesCapteurs);

    while (1) {
        It.First();
        while (!It.Fin()) {
            Delay(It.GetCleS());
            Aff.Poster((It.getElmnt)->EffectuerCtrl());
            It.Next();
        }
    }
}
```

3^{ème} partie : Etude du Système d'Information du SSC.

Question 3.1

Donner le modèle conceptuel de données (MCD) normalisé (modèle entités-associations) de la base.



Question 3.2

En déduire le modèle relationnel de la base.

Client(Code, Nom)

Commande(No, Date, CodeClient)

Detail(NoCmde, NoReservoir, QteCmdee)

Reservoir(No, CodeClient, CodeProduit)

Produit(Code, Abréviation, Libellé)

Cycle(NoCycle, DateDebut)

Distribution(NoCmde, NoReservoir, NoCycle, QteEffective, NoPhase)

Question 3.3

Donner les requêtes en SQL permettant d'obtenir les informations suivantes :

- a. Les totaux par produit d'une commande donnée.

```
Select P.Code, P.Nom, SUM(D.QteCmdee)
From Produit P, Detail D, Commande C, Reservoir R
Where C.No=x AND D.NoCmde=C.NoCmde AND P.Code=T.CodeProduit
Group By P.Code, P.Nom
```

- b. *Le numéro du cycle d'expédition et la quantité effectivement livrée d'un produit donné apparaissant dans une commande donnée.*

```
Select D.NoCycle, SUM(D.QteEffective)
From Distribution D, Reservoir R
where D.NoCmde=x AND D.NoReservoir=R.No AND R.CodeProduit=y
Group By D.NoCycle
```

- c. *Pour les commandes contenant des produits qui n'ont pas été distribués, les numéros de ces commandes et les références de ces produits.*

```
Select D.NoCmde, P.Code From Detail D, Produits P, Reservoir R
Where D.NoReservoir=R.No AND R.CodeProduit =P.Code
AND 0=(Select Count(R1.NoReservoir)
        From Distribution D1
        Where D1.NoCmde=D.NoCmde AND D.NoReservoir =D1.NoReservoir)
```