AGREGATION D'INFORMATIQUE – Session 2001

Admissibilité

Epreuve d'application
Option : Informatique des systèmes industriels

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

A/ PRESENTATION

1 Description générale

Il s'agit d'étudier l'automatisation d'un système de transport d'hydrocarbures par pipelines[1]. Ce système sert à transporter divers types de carburants fournis par la raffinerie de Bizerte jusqu'aux réservoirs des clients qui se trouvent au port de Radès.

Ce système est composé essentiellement :

- d'une station d'expédition à Bizerte,
- d'une station de distribution à Radès,
- d'un pipeline long de 72 Km, entre Bizerte et Radès, utilisé pour tous les types de carburants,
- d'un centre de dispatching, situé à Tunis, qui coordonne les activités des deux stations.

Le schéma de la figure 1 illustre sommairement les différentes parties du système.

[1] Le procédé décrit dans ce sujet est inspiré de celui d'une entreprise tunisienne, la SOTRAPIL.

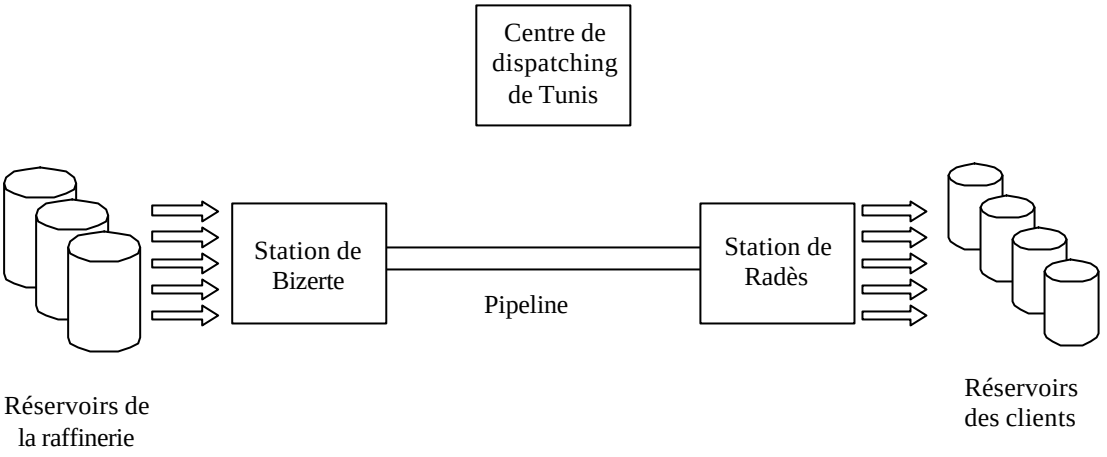


Figure 1 – Description générale du système de transport

La raffinerie fournit 5 types de produits pétroliers :

- Gasoil – GO
- Essence Normale – EN
- Essence Super – ES
- Essence Super Sans Plomb – SSP
- Pétrole Lampant – PL

Les différents types de carburants sont reçus par la station de Bizerte et sont envoyés à travers le pipeline vers la station de réception et de distribution de Radès qui se charge de leur distribution à travers des canalisations vers les réservoirs des clients. Le pipeline a un débit nominal de 300 m^3/h.

Pour des raisons économiques, le même pipeline transporte les différents types de carburants selon un ordre bien déterminé. Lors du passage d'un type de carburant à un autre, le produit intermédiaire qui résulte du mélange des deux types de carburants est appelé contaminât.

Le contaminât est recueilli au niveau de la station de Radès et mis dans des réservoirs particuliers. Il est réinjecté dans le carburant livré aux clients, dans de très faibles proportions, en veillant à ne pas dégrader la qualité de ce carburant au delà des normes reconnues.

Les produits transportés étant très corrosifs et très polluants, de temps à autre un racleur de nettoyage est envoyé dans le pipeline de la station de Bizerte et récupéré à la station de Radès. Le racleur est formé par une boule métallique de même diamètre que le pipeline et recouverte par des morceaux de brosses métalliques qui raclent les parois internes du pipeline pour le nettoyer.

1.1 La station de Bizerte

La station de Bizerte a pour rôle d'expédier les différents types de produits dans le pipeline en direction de la station de Radès. La figure 1 de l'annexe A1 montre les différentes installations de la station de Bizerte. Cette station comprend essentiellement :

- Des filtres, des électrovannes et des groupes de pompes qui permettent de filtrer et d'expédier les produits à travers le pipeline.

- Un banc de mesure qui permet de comptabiliser la quantité de carburant envoyée dans le pipeline.
- Une gare de racleur qui permet d'injecter un racleur dans le pipeline.
- Un système d'échantillonnage qui procède au prélèvement d'un certain nombre d'échantillons étalés sur toute la durée de réception du produit. Le volume total des échantillons prélevés sur chaque type de produit de chaque expédition est de 5 litres. Il est stocké et sert de référence pour une éventuelle réclamation.

1.2 La station de Radès

La station de Radès a pour rôle de réceptionner les produits acheminés à travers le pipeline, de déterminer leurs quantités, d'évaluer leur qualité et de les diriger soit vers les bacs de contaminât soit vers les réservoirs des clients. Pour cela, la station est équipée des installations suivantes (voir figures 2 et 3 de l'annexe A1) :

- Une gare de racleur, qui permet de réceptionner le racleur et de le récupérer.
- Un groupe de filtres S40 et S41 qui permettent d'éliminer les déchets dans les produits. L'un fonctionne et l'autre permet de le secourir.
- Un réservoir de secours R41.
- Un ensemble de réservoirs R42, R43, R44, R45 qui permettent de stocker temporairement les différents produits contaminât.
- Un banc de mesure qui permet de comptabiliser la quantité de produit livrée aux clients ou bien stockée en tant que contaminât.
- Un ensemble de canalisations, d'électrovannes et de pompes permettant de diriger les produits vers les réservoirs clients, de diriger les différents produits contaminât vers les réservoirs correspondants en vue de les réinjecter ultérieurement.
- Un analyseur en ligne qui permet de détecter le type de produit qui passe par la station en examinant ses caractéristiques physico-chimiques.
- Les réservoirs des différents clients de la société.

1.3 Le Centre de Dispatching

Le centre de dispatching de Tunis a pour rôle de réceptionner les commandes des clients et d'assurer le suivi de leur exécution par les stations de Bizerte et de Radès. Il dispose d'une vue globale de tout le système.

1.4 Séquences de transport des produits

Comme décrit plus haut, le contaminât est le résultat du mélange de deux produits expédiés successivement à travers le pipeline.

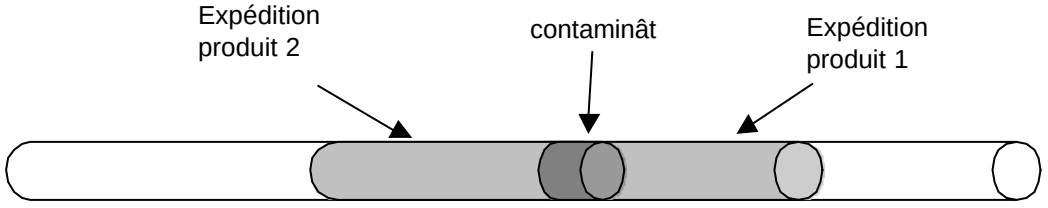
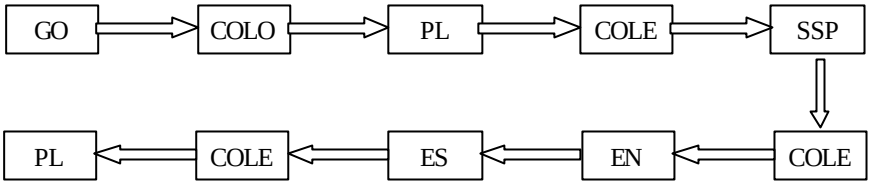


Figure 2 – Transport des divers produits dans le pipeline

- Certains mélanges n'étant pas autorisés, l'expédition des différents types de carburants doit obéir à la séquence de base : GO – COLO – PL – COLE – SSP – COLE – EN – ES – COLE – PL



COLE : Contaminât Léger dû au mélange de produits légers (Essences et Pétrole)
COLO : Contaminât Lourd dû au mélange de produits lourds (Gasoil et Pétrole)

Figure 3 – Séquence de base d'expédition des produits

Notons que le mélange entre l'essence normale et l'essence super ne constitue pas un produit contaminât, il est considéré comme étant de l'essence normale car l'ajout de l'essence super à l'essence normale ne peut que améliorer ce dernier produit.

Pour chaque type de carburant il y a une quantité minimale en dessous de laquelle la société de transport considère que le transport par pipeline n'est pas rentable vu la quantité de contaminât générée. Cependant, le Pétrole Lampant doit être utilisé pour séparer les autres types de carburants, ainsi une quantité minimale de 50 m3 doit être envoyée dans chaque expédition. Le pétrole étant un produit suffisamment demandé, ce minimum ne constitue pas une contrainte.

En cas d'absence d'un des types de carburants (non commandé), il est possible d'effectuer des séquences réduites à partir de cette séquence de base. Il existe un nombre fixe de séquences préétablies parmi lesquelles on peut citer :

- GO – COLO – PL – COLE – EN – ES – COLE – PL
- GO – COLO – PL – COLE – SSP – COLE – ES – COLE – PL
- GO – COLO – PL – COLE – SSP – COLE – EN – COLE – PL
- SSP – COLE – EN – ES – COLE – PL

1.5 Stratégie de transport et de distribution

La société de transport traite uniquement avec six clients (Mobil, Elf, Total, SNDP, Esso et Shell) disposant chacun de ses propres réservoirs à la station de Radès.

Les clients émettent des commandes de carburant (voir Annexe A6) à la raffinerie et des commandes de transport, pour les mêmes quantités, au centre de dispatching. Ils indiquent dans leurs commandes la quantité désirée de chaque type de carburant et le réservoir de destination.

La société de transport organise son activité par cycles de 8 jours. Un cycle comprend deux parties de 4 jours : une première partie "réception et validation des commandes" et une seconde partie, "transport et distribution".

Pendant les 4 premiers jours de chaque cycle, elle réceptionne les commandes de ses clients. Au fur et à mesure, les commandes sont vérifiées auprès de la raffinerie pour s'assurer de la disponibilité des produits commandés et auprès de la station de Radès pour s'assurer de la disponibilité des réservoirs destinataires.

A la fin des 4 premier jours, le centre de dispatching établit un plan d'expédition (voir Annexe A6). Ce plan totalise les commandes de tous les clients par produit et détermine la séquence d'expédition appropriée (cf. § 1.4). Il est envoyé à la station de Bizerte pour exécution. De même, un plan de distribution détaillant les quantités de chaque type de carburant à livrer à chaque client ainsi que les réservoirs correspondants est envoyé à la station de Radès.

Le transport et la distribution du carburant sont effectués durant les 4 derniers jours du cycle. Parallèlement, la première partie du cycle suivant (réception et validation des commandes) est mise en œuvre (voir figure 4).

Signalons que toute commande non honorée dans un cycle est reportée au cycle suivant. Par ailleurs, il n'y a pas de limite maximale pour les quantités commandées puisque la capacité de transport du pipeline durant un cycle d'expédition est nettement supérieure à la capacité de stockage de l'ensemble des réservoirs des clients.

La station de Bizerte puise successivement dans les réservoirs de la raffinerie (supposés toujours alimentés) les quantités désirées de chaque type de carburant. Les différents partenaires conviennent que c'est le banc de mesure homologué de la station de Bizerte qui fait foi pour les facturations..

La station de Radès récupère les différentes quantités, analyse la qualité pour déterminer si le produit est livrable ou non. Si le produit est livrable (qualité acceptable) il est distribué directement aux clients selon le plan de distribution (quantités et réservoirs de destination), si le produit est considéré comme contaminât il est dérouté vers l'un des réservoirs de stockage du contaminât.

Au fur et à mesure de l'exécution du plan de distribution, la station de Radès envoie des rapports au centre de dispatching sur les quantités effectivement livrées aux différents clients et sur les quantités stockées comme contaminât.

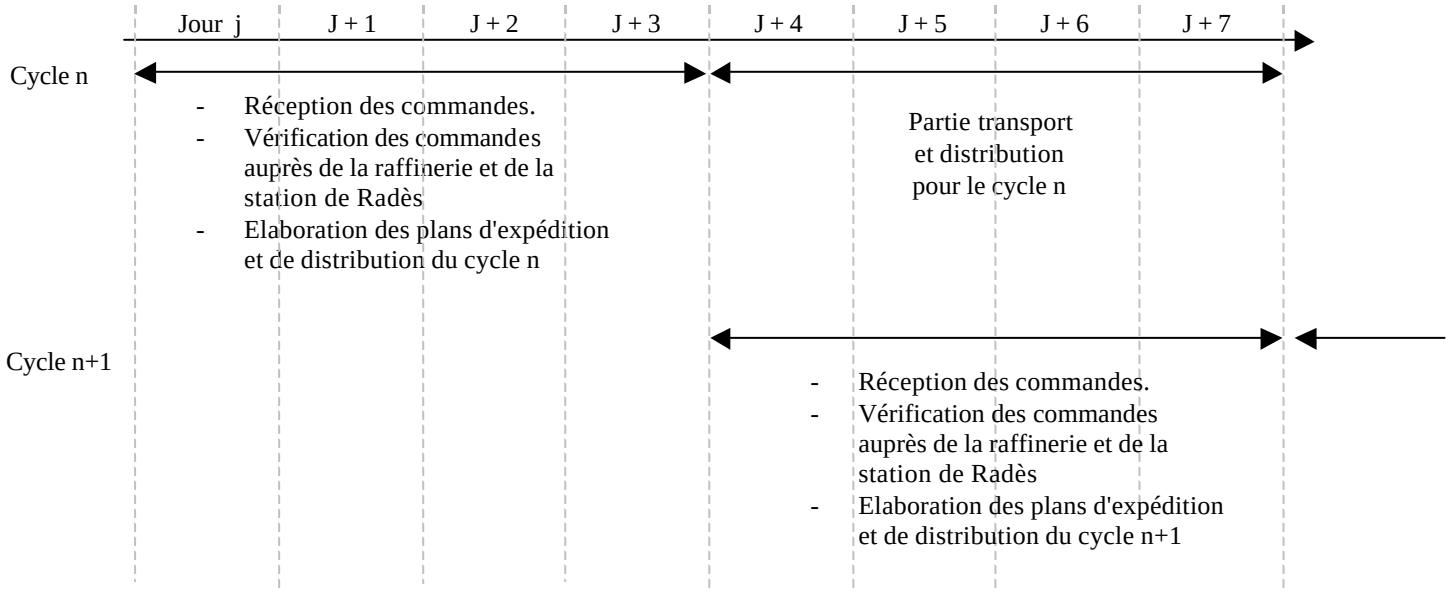


Figure 4 – Cycles d'expédition

La figure 5 récapitule les échanges d'informations entre les différents acteurs interagissant avec le système.

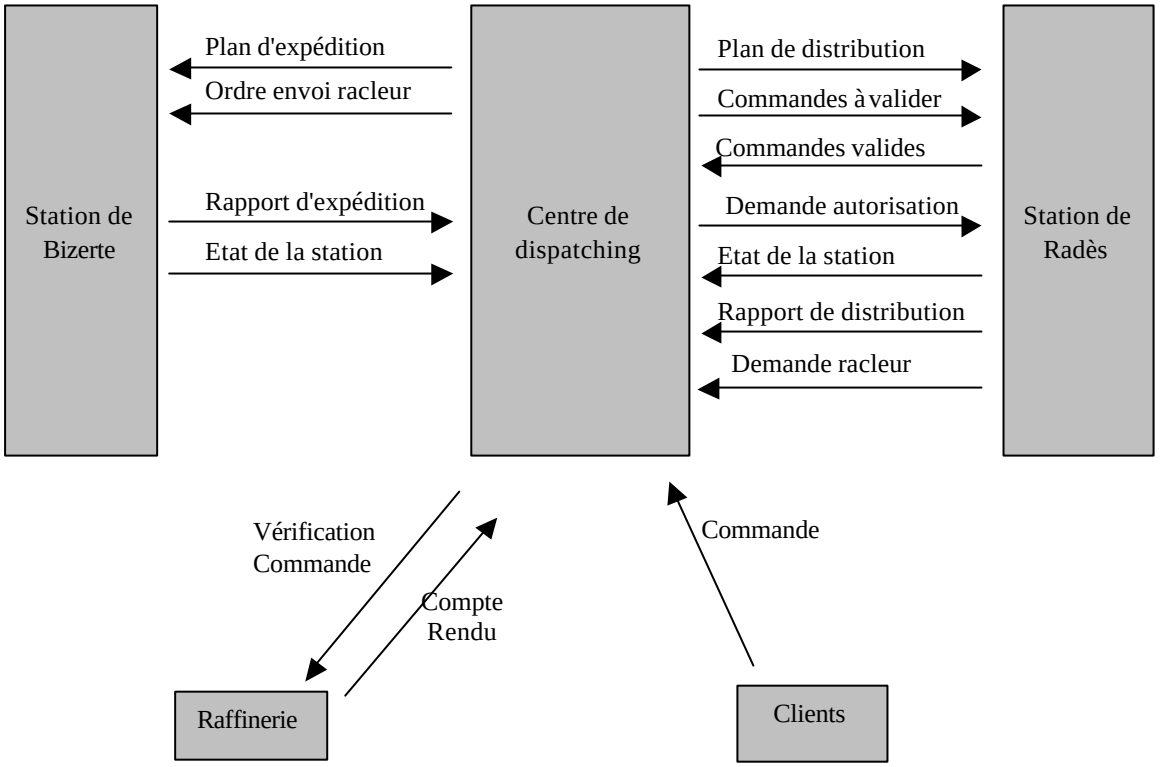


Figure 5 - Schéma récapitulatif des échanges

1.6 Description du système informatique

Le système informatique de la société de transport est constitué des sous-systèmes suivants :

- Le Sous-système de Supervision Centrale (SSC), situé au centre de dispatching de Tunis. Il gère la base de données centrale du système d'information, possède une vision globale des états de fonctionnement des deux stations et assure les interactions éventuelles entre celles-ci.
- Les Sous-systèmes de Commande de Bizerte et de Radès (resp. SCB et SCR), chacun implanté sur un calculateur industriel fonctionnant sous UNIX. Ils s'occupent de la commande en temps réel du procédé et de l'exécution du plan de travail (plan d'expédition pour Bizerte et plan de distribution pour Radès).
- Les Sous-systèmes de Supervision Locale de Bizerte et de Radès (resp. SSLB et SSLR), chacun implanté sur un micro-ordinateur. Ils s'occupent de l'échange d'informations avec le superviseur central, de la visualisation en temps réel de l'état du procédé, de la gestion des alarmes et de la prise en compte des interventions de l'opérateur.

Dans chaque station, le sous-système de commande est relié au sous-système de supervision locale à travers un réseau local. Les deux sous-systèmes de supervision locale de Radès et de Bizerte sont reliés au sous-système de supervision central (SSC) par des lignes spécialisées (voir figure 6).

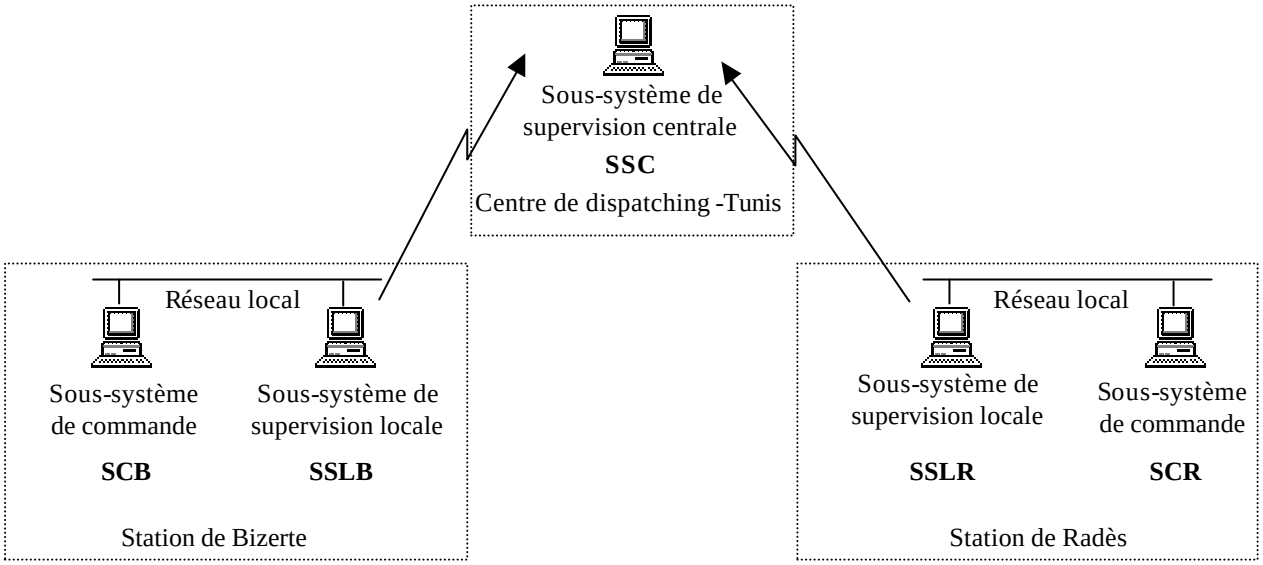


Figure 6 – Architecture du système informatique

L'étude portera sur :

- le sous-système de commande de la station de Radès SCR,
- le sous-système de supervision locale de la station de Radès SSLR,
- le système d'information du SSC.

2 Etude du sous-système de commande de la station de Radès

2.1 Description des installations de la station de Radès (figures 2 et 3 de l'annexe A1)

- La gare du racleur : elle permet de récupérer le racleur envoyé par la station de Bizerte pour nettoyer le pipeline. Elle dispose de deux capteurs : XA401, à 300 m de la station, et XA402, dans la loge d'arrivée du racleur, qui permettent de détecter le passage du racleur. Les électrovannes EV401, EV402 et EV403 permettent de dévier le racleur vers sa loge à son arrivée.
- Les filtres S40 et S41 qui permettent de nettoyer le produit. Un seul est utilisé pour filtrer le produit, quand il est colmaté le produit est dévié vers le second au moyen des électrovannes EV404 et EV405. Chaque filtre dispose d'un capteur qui signale son passage à l'état colmaté.
- L'analyseur en ligne permet, sur demande, de mesurer les caractéristiques physico-chimiques du produit.
- Deux bancs de mesure homologués permettent de mesurer la quantité de produit circulant dans la ligne principale de distribution. Un seul est nécessaire pour mesurer la quantité, le second est mis en service en cas de défaillance du premier. Chaque banc de mesure est équipé d'une hélice qui tourne sous l'effet du passage du produit. A chaque tour d'hélice, une impulsion est générée. Le nombre d'impulsions permet au

système de calculer la quantité brute qui passe par le banc de mesure. Selon la nature du produit, cette quantité doit être corrigée et ramenée à des valeurs de référence de température.

- Des réservoirs pour stocker temporairement les produits contaminât : R42 et R43 pour le contaminât lourd (COLO) et R44 et R45 pour le contaminât léger (COLE).
- Un réservoir de secours R41 permet de stocker le produit arrivant en cas de surpression dans la station. La vanne de sécurité (VS) s'ouvre automatiquement sous l'effet d'une pression supérieure à 18 bars et dévie le produit vers le réservoir de secours en attendant la fermeture des vannes d'entrée de la station. La vanne de sécurité est équipée d'un capteur qui signale son ouverture.
- Les pompes EP46 et EP47 permettent de réinjecter le contenu des réservoirs R41 à R45 dans la ligne principale de distribution. Chaque pompe dispose d'un thermostat qui signale la surchauffe de la pompe.
- Les réservoirs clients : chaque client dispose de plusieurs réservoirs par type de produit.
- Des équipements de mesure de niveau : chaque réservoir dispose d'une jauge de mesure du niveau du produit dans le réservoir et de deux capteurs des niveaux vide et plein.
- Un ensemble d'électrovannes permettent l'acheminement du produit aux réservoirs destinataires. Chaque électrovanne peut être commandée en ouverture ou en fermeture. Un capteur permet de détecter son état effectif (ouverte, fermée).
- Une vanne régulée autonome PCV1 permet de maintenir un débit fixé par consigne.
- Un capteur de surpression permettant de détecter une surpression anormale dans les canalisations de la station.

2.2 Fonctionnement du SCR

Le SCR reçoit les requêtes de validation des commandes des clients provenant du SSC, il vérifie la disponibilité des réservoirs destinataires et informe le SSC de la possibilité d'exécution de ces commandes.

Avant le début de la partie transport et distribution d'un cycle n, le SSC envoie au SCR le plan de distribution et une demande de début d'expédition. Ce plan (voir Annexe A6) contient les différentes phases de distribution. Chaque phase spécifie un type de produit attendu, le réservoir destinataire et la quantité à livrer.

Le SCR vérifie que le système est prêt à fonctionner et envoie au SSC une autorisation d'expédition et se met en attente du carburant : le système est alors en mode automatique.

Le carburant arrivant à la station passe d'abord par l'un des filtres, puis par l'analyseur en ligne, ensuite par l'un des bancs de mesure, enfin il est acheminé vers l'un des réservoirs clients ou vers un réservoir contaminât en fonction de sa nature et des différentes phases du plan de distribution.

En mode automatique, le SCR exécute les différentes phases du plan de distribution, gère les filtres, le racleur, la réinjection du contaminât et prend en compte l'intervention de l'opérateur.

Exécution du plan de distribution

L'exécution du plan de distribution se fait par phases successives.

La première phase du plan de distribution commence par l'arrivée du premier produit. Pendant la distribution, le SCR envoie périodiquement à l'analyseur en ligne une demande de mesure des caractéristiques physico-chimiques du produit pour s'assurer qu'il est dans les normes et commande les électrovannes de façon à acheminer ce produit vers le réservoir client spécifique à cette phase. Le SCR comptabilise au fur et à mesure la quantité de produit livrée au client.

Une phase se termine dans les deux cas suivants :

- la quantité de produit livré atteint la valeur spécifiée dans le plan de distribution. Dans ce cas, le SCR émet vers le SSC à travers le SSLR un compte rendu d'exécution de cette phase (la quantité réelle livrée et le réservoir destinataire) et il passe à la phase suivante : il actionne les électrovannes de telle façon à acheminer le produit vers le réservoir client de cette nouvelle phase ;

- la détection du passage d'un type de produit à un autre. Dans ce cas, c'est en comparant les paramètres fournis par l'analyseur en ligne et ceux du produit normalement attendu que le SCR se rend compte que la phase s'achève. Le SCR émet vers le SSLR un compte rendu d'exécution de cette phase. Si le prochain produit attendu est un produit contaminât, le SCR commande les différentes électrovannes de façon à diriger le produit contaminât vers l'un des réservoirs de récupération du contaminât. Sinon (ex : passage EN-ES), il actionne les électrovannes de telle façon à acheminer le produit vers le réservoir client de cette nouvelle phase.

Gestion des filtres

Si le filtre en cours d'activité est colmaté, le SCR commande les électrovannes de telle façon à faire passer le produit par le second filtre. Il envoie également une demande de racleur pour nettoyer le pipeline.

Gestion du racleur

Comme décrit plus haut, le racleur est utilisé pour nettoyer les parois internes du pipeline, il est injecté à la gare du racleur de la station de Bizerte, en cours d'expédition du carburant, et récupéré à la gare du racleur de la station de Radès. Le racleur est poussé par la pression du carburant jusqu'à la station de Radès.

Lors de la réception du carburant, l'électrovanne EV402 est ouverte tandis que les électrovannes EV401 et EV403 sont fermées. Dès la détection du passage du racleur par le premier capteur XA_401, les opérations suivantes sont effectuées :
- ouvrir l'électrovanne EV401,
- ouvrir l'électrovanne EV403,
- fermer l'électrovanne EV402.

Ensuite, à la détection de l'arrivée du racleur par le capteur XA_402 :
- ouvrir l'électrovanne EV402,
- fermer les électrovannes EV401 et EV403.

Réinjection du contaminât

Le contenu des réservoirs R41 à R45 peut être réinjecté dans le produit en cours de livraison dans de très faibles proportion. Cette proportion est décidée par l'opérateur en fonction du type et de la qualité du carburant arrivant à la station et en fonction de la nature du contaminât. Il indique au SCR (à travers le SSLR) une valeur correspondant au pourcentage d'injection du contaminât dans le produit en cours de livraison et donne l'autorisation d'injection. En fonction de cette valeur et du débit d'arrivée du carburant à la station, le SCR commande la vanne régulée PCV1 avec la consigne de débit qui convient. Le SCR arrête cette réinjection dès la détection de la fin de livraison du type de produit en cours.

Prise en compte de l'intervention de l'opérateur

L'opérateur intervient au niveau du SSLR pour donner les ordres suivants au SCR :

- passage en mode manuel
- reprise
- injection de contaminât avec le pourcentage autorisé.

Commutation entre modes

Le passage du mode automatique au mode manuel peut s'effectuer soit à la demande de l'opérateur soit lors de l'apparition de certains problèmes bloquants. Le SCR décide alors d'arrêter la distribution (fermeture des électrovannes et arrêt des pompes) et commute en mode manuel.

Les problèmes qui provoquent ce passage sont :

- les deux filtres sont colmatés,
- un capteur de niveau plein de l'un des réservoirs est actionné,
- le SCR s'aperçoit que l'une des électrovannes n'a pas exécuté la commande qui lui a été envoyée (ouvrir, fermer) au bout de 6 secondes,
- le déclenchement de la vanne de sécurité,
- la surchauffe de l'une des pompes,
- la détection d'une surpression dans les canalisations.

Dans le mode manuel, qui n'est pas pris en compte dans la présente étude, l'opérateur agit sur les installations pour les remettre en marche et fournit au SCR le signal « reprise ». Le SCR repasse alors en mode automatique et continue l'exécution du plan de distribution.

3 Etude du sous-système de supervision locale de la station de Radès

3.1 Fonctionnalités

Le SSLR offre deux fonctionnalités majeures :

- La visualisation en ligne de l'état du système : cette fonction permet de suivre l'état du procédé à travers une interface graphique appropriée (synoptique de la station). Elle permet aussi à l'opérateur de réaliser des interactions simples avec le SCR comme par exemple : la mise en marche et l'arrêt et le changement de mode (manuel ou automatique). Elle permet enfin de fixer les consignes (paramètres d'injection du contaminât).
- La gestion des messages d'alarme : cette fonction a en charge l'identification des situations d'alarmes (réservoirs pleins, surchauffe, surpression, etc. …) et le traitement des messages correspondants.

Dans cette étude nous nous limiterons au développement de la fonction de gestion des alarmes qui sera désignée dans la suite de l'étude par "Application de Gestion d'Alarmes (AGA)"

3.2 Gestion des messages d'alarmes

Il existe deux sources de messages d'alarmes :

- Les messages issus des capteurs associés aux équipements (capteurs de pression, thermostats, capteurs de niveau,… cf. § 2.1) .
- Les messages issus du SCR qui concernent le blocage éventuel de l'une des électrovannes.

En ce qui concerne le premier type de messages. L'AGA doit contrôler périodiquement les différents capteurs du procédé à une fréquence qui leur est spécifique et générer les affichages appropriés.

Pour le deuxième type de messages, l'arrivée du message doit provoquer son affichage par l'AGA.

Les messages d'alarmes se caractérisent par des priorités différentes selon leur degré d'urgence (par exemple, les messages bloquants sont plus prioritaires que les messages non bloquants).

4 Etude du système d'information

Les informations persistantes du système sont stockées dans une base de données centralisée implantée sur le micro-ordinateur du SSC. Celui-ci inclut un système de gestion de base de données relationnel. Il gère dans cette base de données un ensemble d'informations se rapportant essentiellement aux commandes et aux plans de livraison et de distribution. La base de données est constituée à partir de trois documents circulant au sein de la société de transport de carburant. Une description de ces documents est donnée en Annexe A6.

4.1 Commandes

La société reçoit régulièrement de la part de ses clients des commandes pour le transport (voir Annexe A6). Elle procède au fur et à mesure et après validation, à la saisie des commandes. Chaque commande se voit attribuer une référence constituée des deux derniers chiffres de l'année en cours concaténés à un numéro d'ordre. Le client précise dans sa commande les quantités par réservoir d'accueil pour chaque produit qu'il a commandé. Il est à noter que la société s'assure que les réservoirs (désignés par leurs numéros) appartiennent bien au client et sont bien destinés aux produits commandés.

4.2 Plan de distribution

Avant la mise en œuvre de la partie transport et distribution d'un cycle, la société édite un plan de distribution (voir annexe A6). Pour cela, le SSC totalise les quantités commandées par type de produit pour l'ensemble des clients. Il élimine les produits qui n'ont pas été commandés en quantités suffisantes (cf. § 1.4), consulte un fichier qui définit toutes les séquences autorisées pour déterminer une séquence permettant d'expédier les produits commandés retenus. Il est à noter que ce fichier ne fait pas partie de la base de données.

Le plan de distribution obtenu, impose l'ordre des différentes phases de distribution pour un cycle de transport donné et précise le réservoir destinataire par phase.

4.3 Plan d'expédition

Le plan d'expédition (cf. Annexe A6) est édité en même temps que le plan de distribution. Il est envoyé à la station de Bizerte. Il est obtenu à partir du plan de distribution : il totalise les différentes quantités par produit et les édite en respectant la séquence retenue.

B/ QUESTIONS

Le travail demandé est décomposé en 3 parties qui peuvent être traitées de façon indépendante.

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

La spécification du SCR est conduite à l'aide de la méthode SA-RT selon l'approche Ward et Mellor. Une spécification partielle est fournie en annexe A2.

Dans les réponses aux questions, on respectera les identifiants précisés dans le dictionnaire de données.

L'implémentation du SCR sera réalisée sous l'environnement UNIX. Une tâche du SCR sera implémentée par un processus UNIX. La gestion parallèle des processus UNIX et la synchronisation entre ces processus seront réalisées à l'aide des primitives UNIX fournies en annexe A3.

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".

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.

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" qui a pour rôle d'extraire la phase suivante du plan de distribution et de communiquer aux autres transformations les informations nécessaires à son exécution et éventuellement du type de contaminât attendu.
- "Mesurer Quantité Produit" qui mesure la quantité de produit en fonction des impulsions reçues des hélices et émet un contrôle dès que la quantité à livrer est

atteinte. Cette transformation peut être interrompue par la transformation de contrôle, auquel cas elle émet la quantité mesurée et réinitialise son compteur interne.
- "Informer Qtté livrées" qui fournit au niveau supérieur un compte rendu de l'exécution de la phase en cours.
- "Traiter contaminât" qui gère la récupération du 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.

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

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.

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.

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.

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

On adopte une approche objet pour réaliser l'application de gestion d'alarmes (AGA) du SSLR. Pour cela, on associe à chaque capteur un objet qui le matérialise au niveau de l'AGA. L'architecture matérielle impose que les capteurs se trouvent physiquement connectés au calculateur industriel sur lequel est implanté le SCR. Il en résulte que l'AGA est en réalité une application répartie sur les deux calculateurs de la station de Radès : les objets capteurs résident sur le calculateur industriel alors que le reste des objets de l'application de gestion des messages résident sur le micro-ordinateur du SSLR. L'environnement de développement de l'application offre un modèle de répartition des objets approprié (tel que CORBA, ou autre). Ainsi, pour la suite de l'étude nous considérons que l'ensemble des objets de l'application résident virtuellement sur le micro-ordinateur du SSLR.

Une analyse préalable de l'AGA a permis d'identifier les objets suivants :

- Objets *Capteurs:* Il s'agit de l'ensemble des objets capteurs introduits ci-dessus. Ils offrent tous une opération *Effectuer_controle* qui permet de lire la valeur du capteur et d'en déduire les messages d'alarmes éventuels. Il convient de noter que ces objets matérialisent des capteurs de natures différentes (pression, température,…) et de marques et de modèles différents. En conséquence, l'implantation de cette opération peut différer d'un objet à un autre.

- Objet *Planificateur* : Le planificateur doit avoir connaissance de tous les objets *Capteurs*. Il a pour fonction de demander à chacun d'entre eux, à tour de rôle et en respectant les priorités et les fréquences respectives, d'effectuer le contrôle qui lui incombe. Le *Planificateur* doit donc gérer une liste (triée sur deux clés : les priorités et les fréquences) de tous les objets capteurs. Il offrira au moins les deux opérations suivantes :

  - *Inscrire*: qui permet de rajouter un capteur à la liste des objets gérés. Cette opération est utilisée lors de l'activation du système, elle permet à l'ensemble des objets capteurs de s'inscrire auprès du planificateur en précisant leurs fréquences et leurs priorités.

  - *Planifier*: Cette opération fait entrer le planificateur dans une boucle infinie dont l'algorithme est le suivant :

    ```
    Répéter
    ```
    - Déterminer quel est le prochain objet qui doit effectuer le contrôle cyclique.
    - Attendre que la période d'attente éventuelle pour cet objet se soit écoulée (selon sa fréquence de contrôle).
    - Demander à l'objet d'effectuer le contrôle et obtenir le message d'alarme éventuel.
    - Demander à l'*Afficheur* d'afficher le message d'alarme.

    ```
    FRépéter
    ```

- Objets *Messages* : Les messages d'alarme et leurs algorithmes d'affichage diffèrent suivant les objets *Capteurs* qui les ont émis et la nature du message. Certains messages sont simplement des avertissements textuels à afficher, d'autres peuvent nécessiter une animation particulière accompagnée d'un signal sonore sur

le synoptique du procédé : faire clignoter une icône en émettant un beep, etc. … Il en résulte que les messages d'alarme seront considérés comme des objets qui offrent une opération *Afficher*. Chaque objet *Capteur* devra donc savoir créer les objets correspondant aux différents messages qu'il doit générer suite à la détection d'une alarme.

- Objet *Afficheur* (introduit dans l'algorithme de l'objet *Planificateur*) : cet objet a en charge l'affichage des messages. Les messages à afficher ont deux sources :

  - Les messages obtenus à partir des objets *Capteurs*, (cf. l'algorithme de l'opération *Planifier* ci-dessus).
  - Les messages issus du SCR (cf. L'objet *Ecouteur* ci-après).

  Ces messages sont gérés dans une file d'attente. L'afficheur devra donc offrir une Opération *Poster* qui sera utilisée pour mettre des messages dans la file en fonction de leurs priorités. Parallèlement, l'*Afficheur* devra procéder à l'affichage de tous les messages en attente dans la file.

- Objet *Ecouteur* : cet objet a pour rôle de construire les objets messages correspondants à des messages de blocage d'électrovannes émis par le SCR . Le SCR n'a pas connaissance de la structure objet des messages. Il se contente d'émettre sur le réseau le numéro de la vanne bloquée. L'AGA devra inclure un objet *Ecouteur* qui se charge de collecter les messages émis par la partie commande, de construire les objets *Message* correspondants et de les ajouter à la file d'attente grâce à l'opération Poster de *l'Afficheur*. Il est à noter que tous les objets messages construits par l'objet *Ecouteur* ont une même priorité prédéfinie.

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).

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*.

Question 2.3

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

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).

3ème 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.

Question 3.3

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

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

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

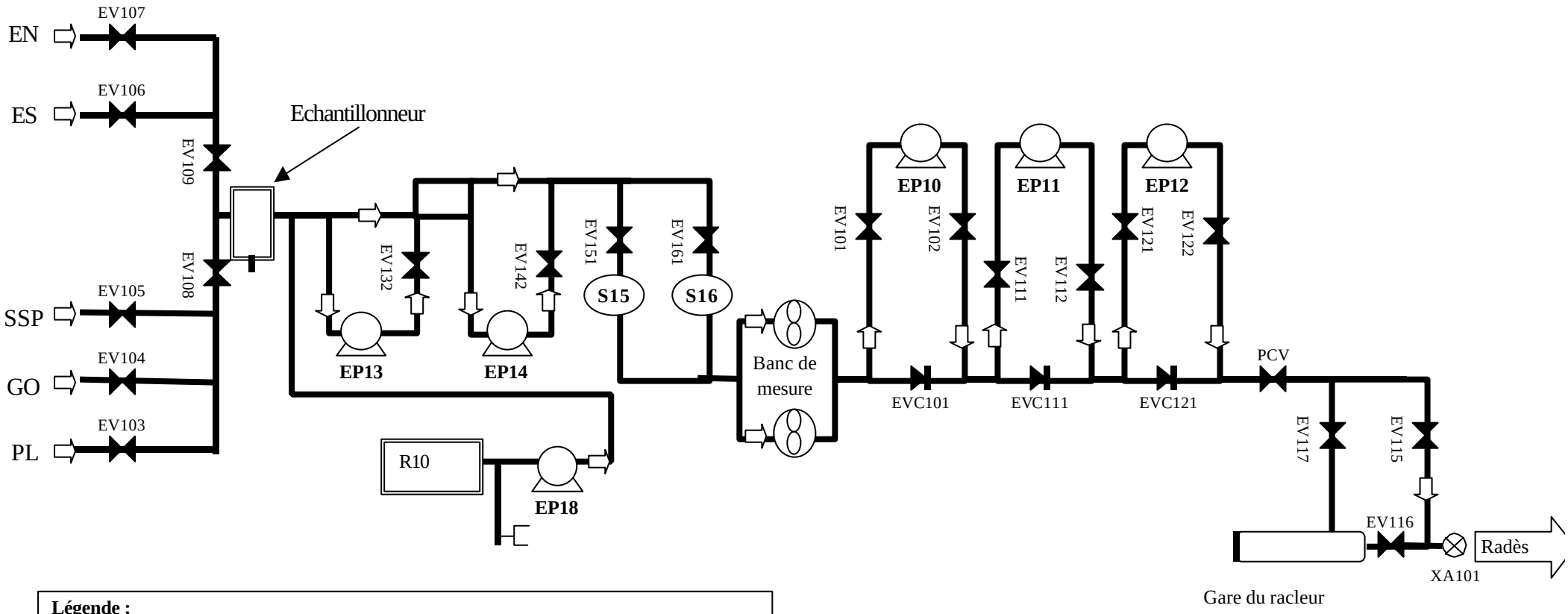
c. Pour les commandes contenant des produits qui n'ont pas été distribués, les numéros de ces commandes et les références de ces produits.

C/ Annexes

Annexe A1 – **Synoptiques des stations**
Annexe A2 – **Spécification partielle par SA-RT**
Annexe A3 – **Primitives UNIX**
Annexe A4 – **Notations HOOD**
Annexe A5 - **Extrait d'une bibliothèque C++**
Annexe A6 - **Documents relatifs au système d'information**

Annexe A1 – Synoptique des stations

Figure 1 - Schéma synoptique de la station de Bizerte



Légende :

- EV401 — Electrovanne
- Hélice du banc de mesure
- S41 — Filtre
- Sens de circulation du carburant
- EVC101 — Electrovanne avec clapet anti-retour
- EP48 — Pompe
- Capteur
- PCV — Vanne Régulée
- R10 — Réservoir

Figure 2 - Schéma synoptique de la station de Radès

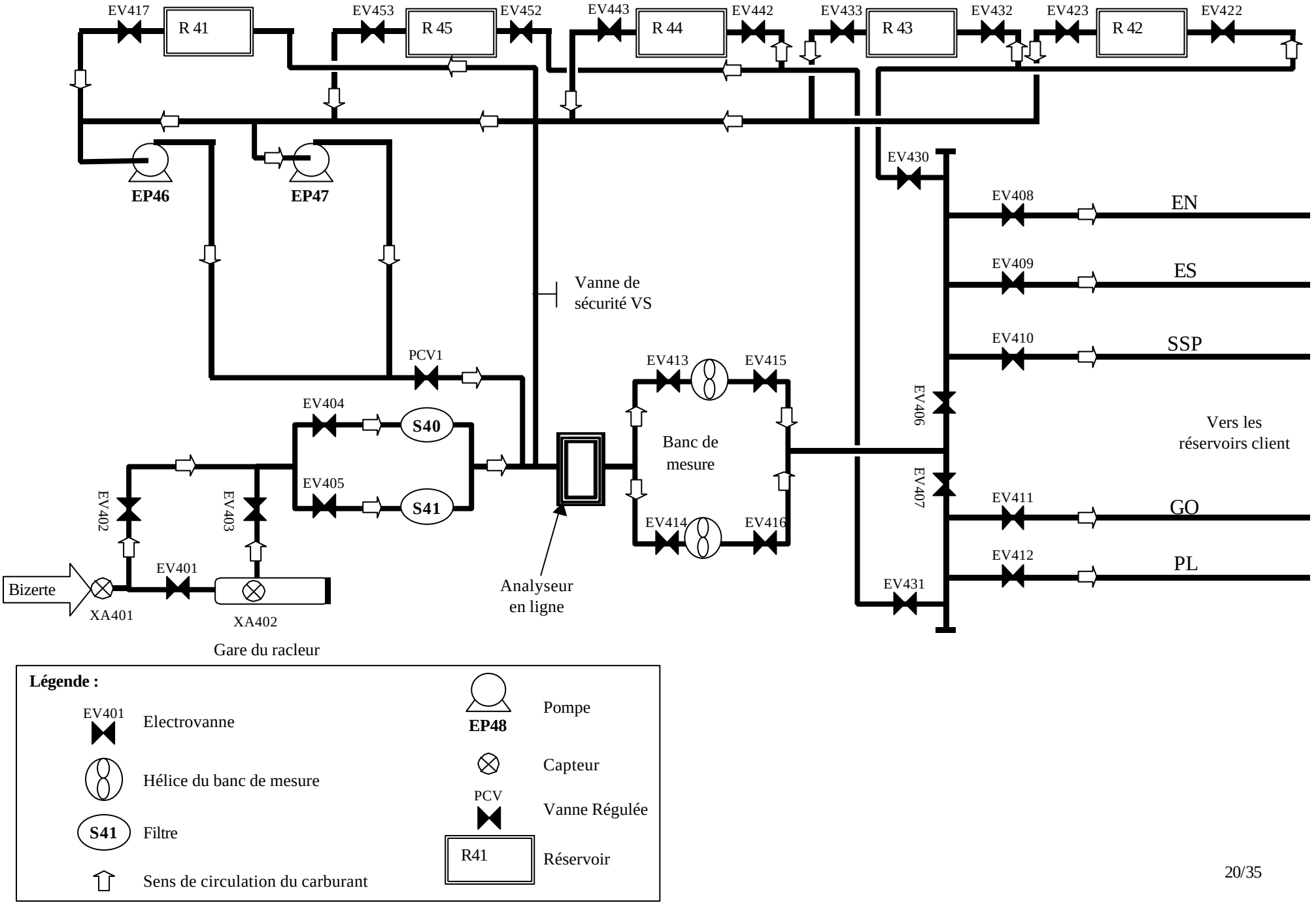
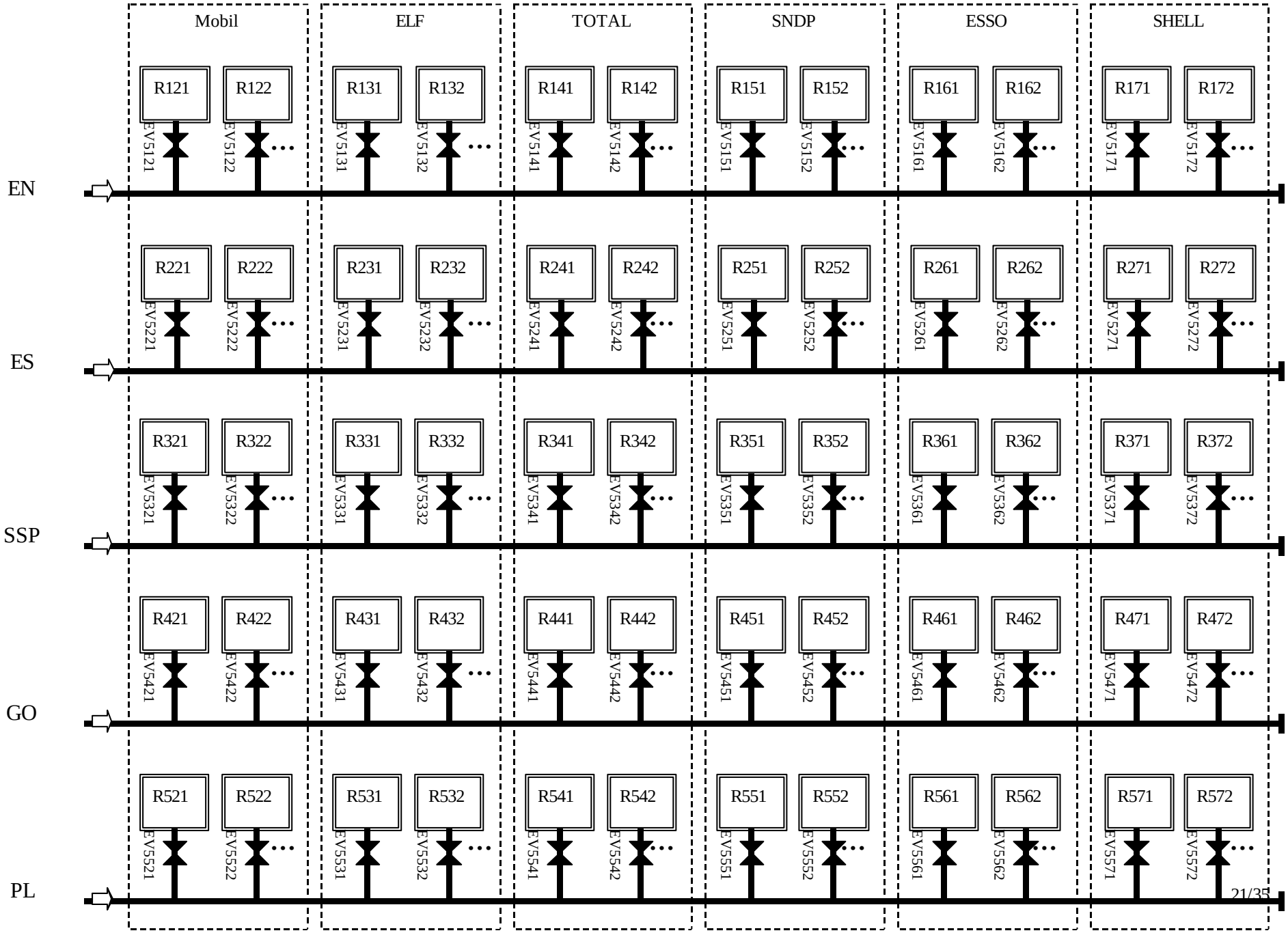


Figure 3 - Schéma synoptique de l'alimentation des réservoirs clients à la station de Radès



Annexe A2 - Spécification partielle par SA-RT

Cette annexe fournit la spécification partielle du SCR par la méthode SA-RT selon l'approche Ward et Mellor.

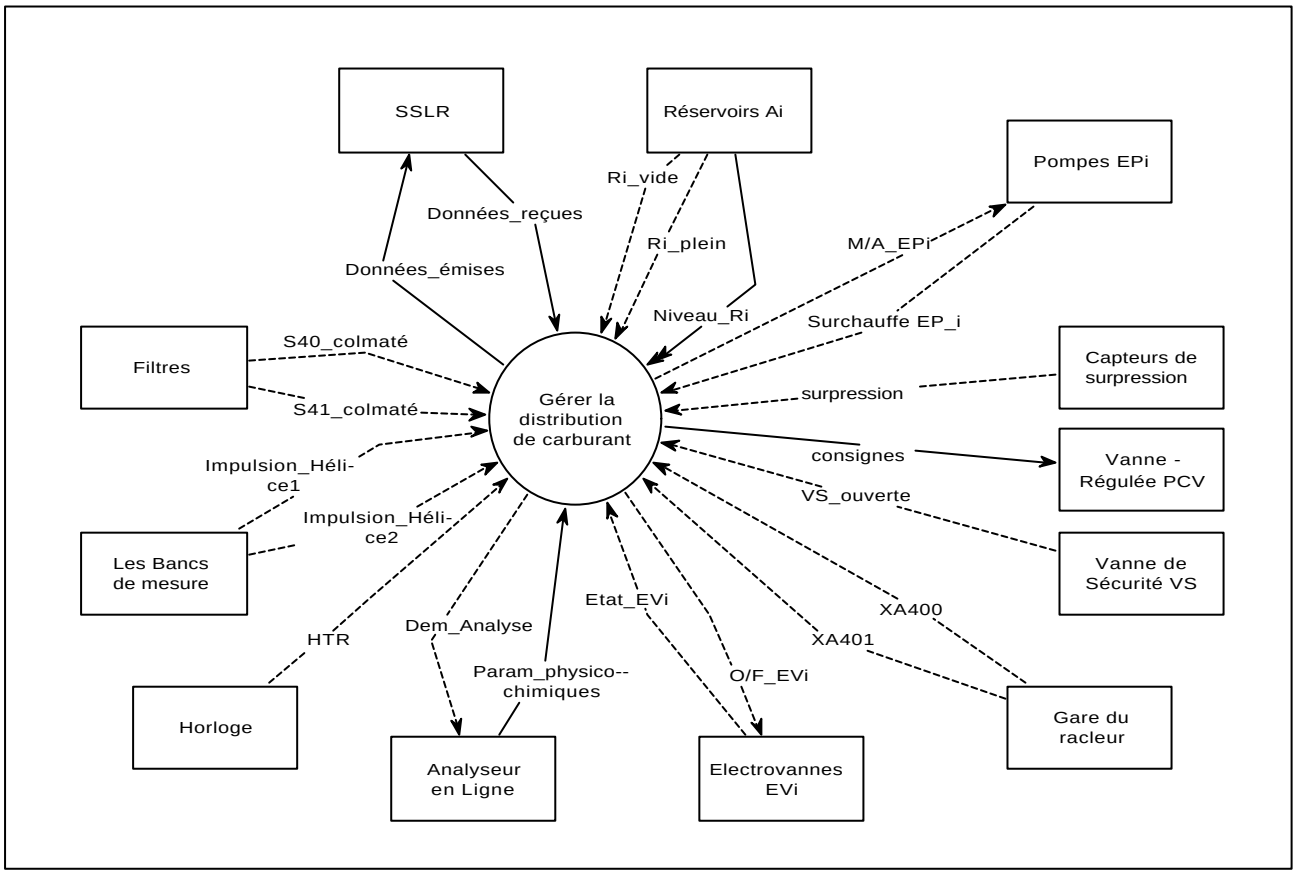


Schéma de contexte

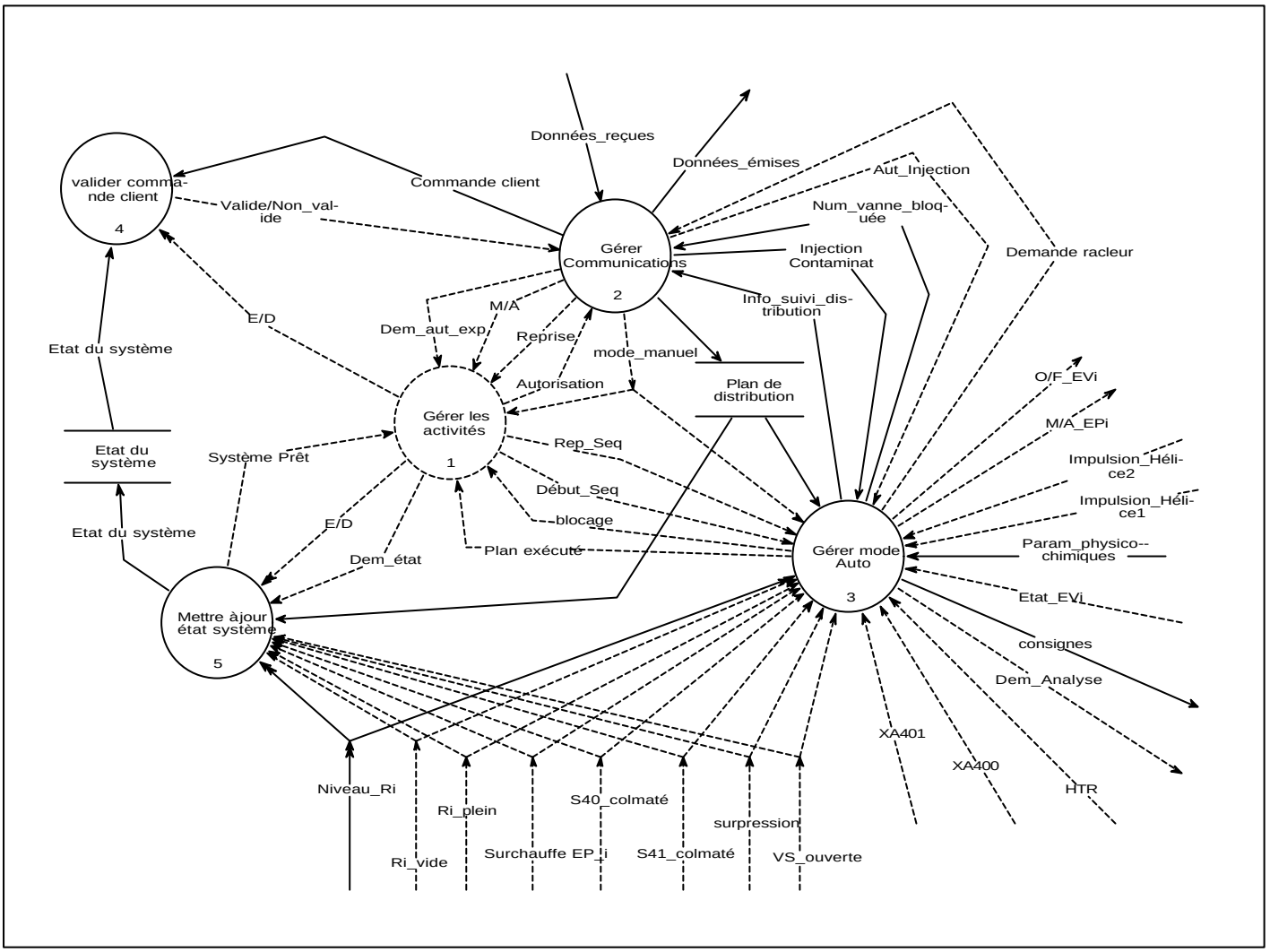


Schéma préliminaire – ST0 – "Gérer la distribution de carburant"

Remarque :

E/D : Enable/Disable (Autoriser/Inhiber)
T : Trigger (Déclencher)

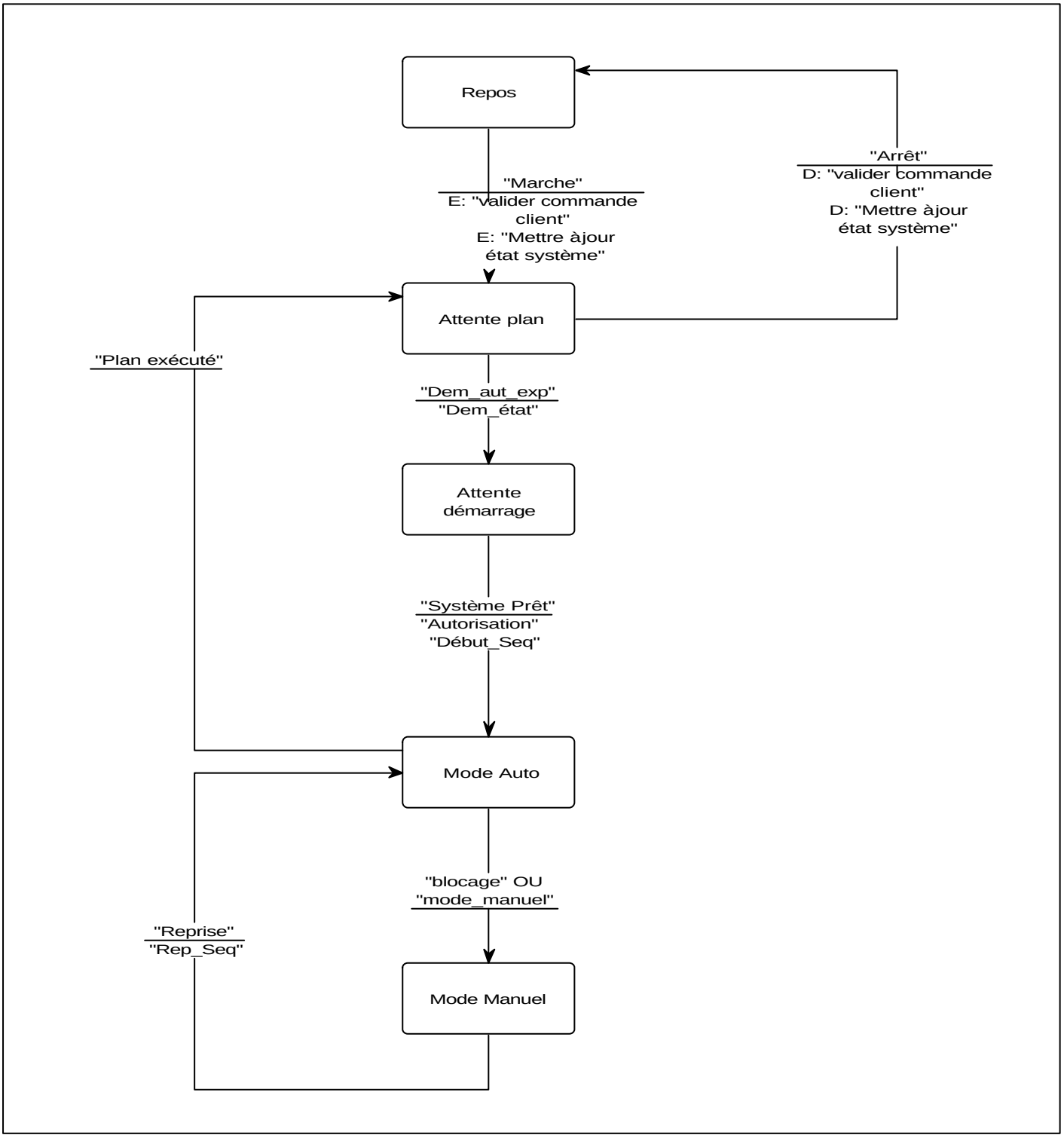


Diagramme Etats-transitions de la Transformation 1.0 – "Gérer les activités"

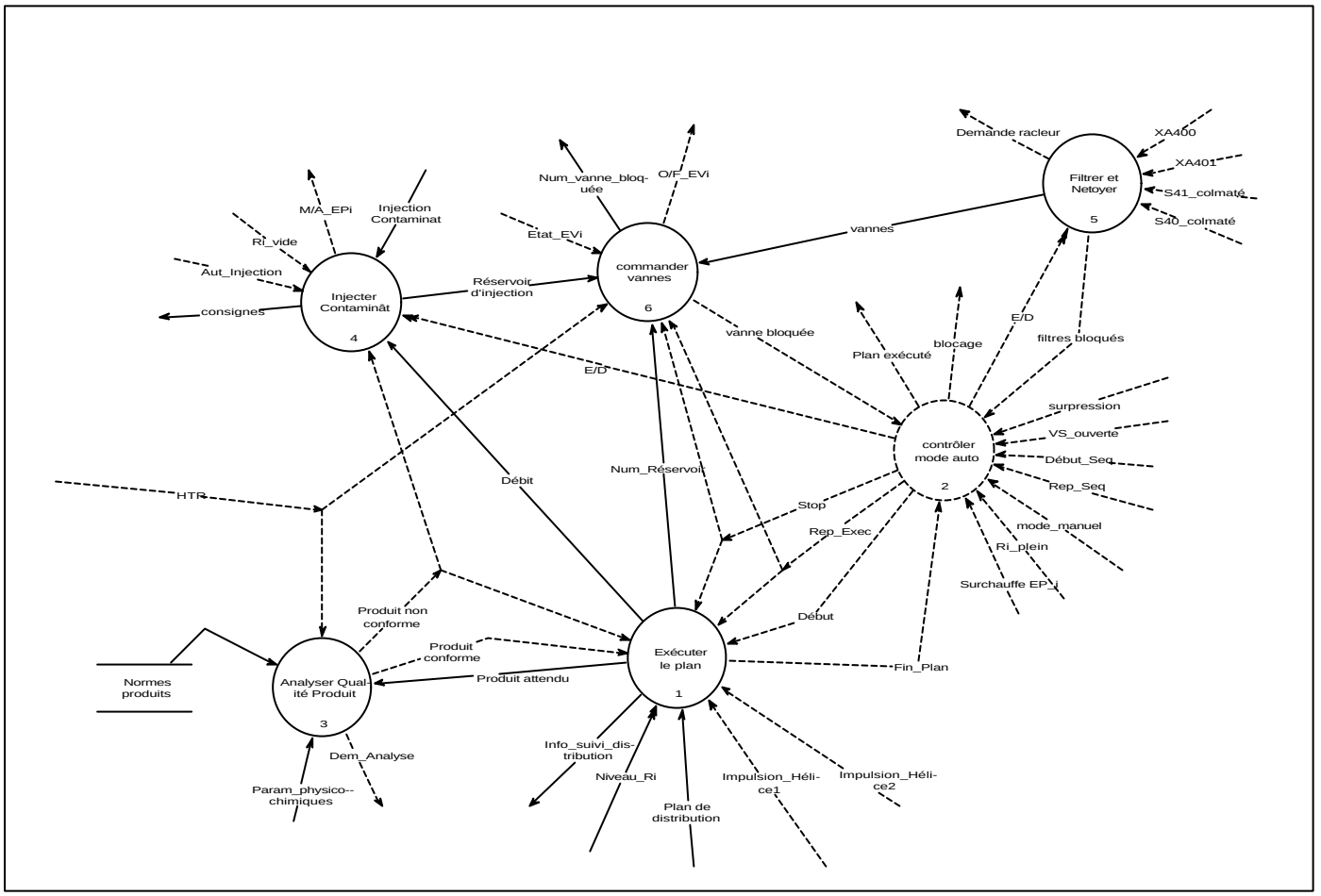


Schéma de Transformation – ST3.0 – "Gérer mode Auto"

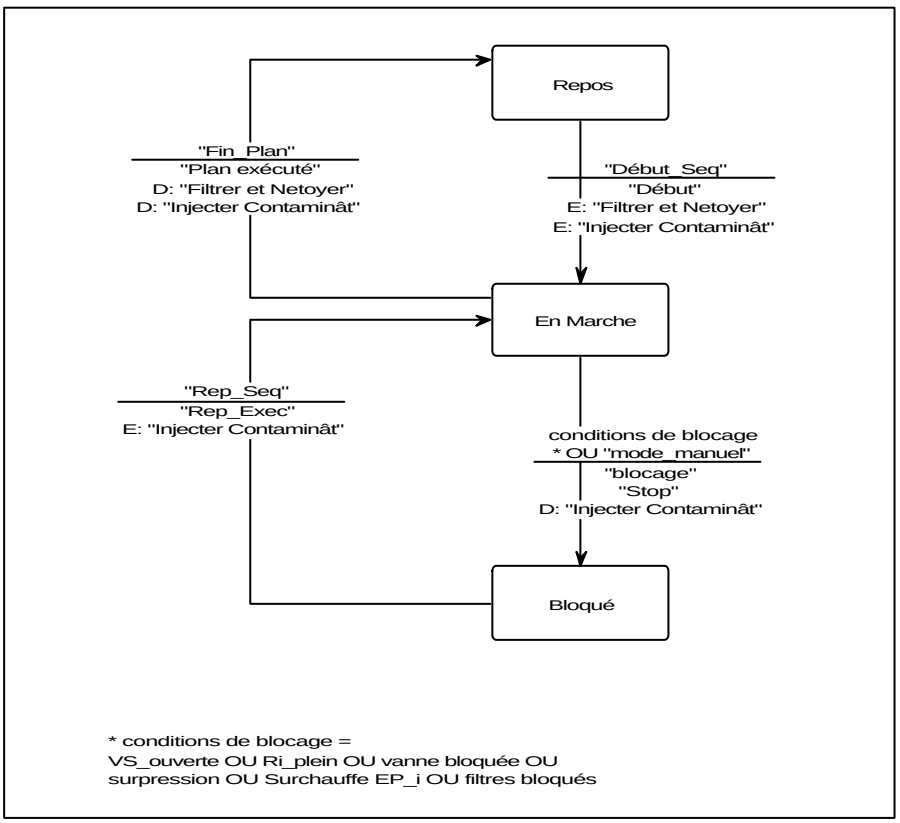


Diagramme Etats-transitions de la Transformation 3.2 – "contrôler mode auto"

Dictionnaire de données

Name: Arrêt
Type: Control flow

Name: Arrêt_EPi
Type: Control flow

Name: Autorisation
Type: Control flow
Comment: Autorisation envoyée au SSC pour commencer l'expédition

Name: Aut_Injection
Type: Control flow
Comment: Ordre fournie par l'opérateur à travers le SSLR pour commencer l'injection de contaminât

Name: blocage
Type: Control flow
Comment: l'une des conditions bloquantes est détectée

Name: Capacités réservoirs
Type: Discrete flow

Name: Commande client
Type: Discrete flow
Comment: détail de la commande d'un client destinée pour vérifier la si le réservoir spécifié peut l'accueillir

Name: consignes
Type: Discrete flow
Comment: consigne de débit envoyée à la vanne régulée PCV1

Name: Contaminât attendu
Type: Discrete flow

Name: Demande racleur
Type: Control flow

Name: Dem_Analyse
Type: Control flow

Name: Dem_aut_exp
Type: Control flow
Comment: Demande envoyée par le SSC pour demander l'autorisation de commencer l'expédition du carburant

Name: Dem_état
Type: Control flow
Comment: Demande si le système est prêt

Name: Données_reçues
Type: Discrete flow
Comment: trames de données reçues sur le bus local

Name: Données_émises
Type: Discrete flow
Comment: trames de données envoyées sur le bus local

Name: Débit
Type: Discrete flow

Name: Début
Type: Control flow
Comment: Début d'exécution du plan de distribution

Name: Début_Seq
Type: Control flow
Comment: Début d'exécution du plan de distribution

Name: Etat du système
Type: Discrete flow

Name: Etat_EVi
Type: Control flow
Bnf: [EVi Ouverte | EVi Fermée]

Name: EVi Fermée
Type: Control flow

Name: EVi Ouverte
Type: Control flow

Name: exécuté
Type: Control flow

Name: Fermer EVi
Type: Control flow

Name: filtres bloqués
Type: Control flow

Name: Fin_Plan
Type: Control flow
Comment: fin d'exécution du plan de distribution

Name: HTR
Type: Control flow
Comment: Horloge temps réel

Name: Impulsion_Hélice1
Type: Control flow
Comment: signal Généré à chaque tour d'hélice du banc de mesure 1

Name: Impulsion_Hélice2
Type: Control flow
Comment: signal Généré à chaque tour d'hélice du banc de mesure 2

Name: Info_suivi_distribution
Type: Discrete flow
Comment: rapport émis à chaque fin de phase récapitulant la quantité réelle livrée ou la quantité de contaminât recueillie

Name: Injection Contaminat
Type: Discrete flow
Comment: pourcentage d'injection du contaminât fournie par l'opérateur à travers le poste du SSLR

Name: Marche
Type: Control flow

Name: Marche_EPi
Type: Control flow

Name: mode_manuel
Type: Control flow

Name: M/A
Type: Control flow
Bnf: [Marche | Arrêt]

Name: M/A_EPi
Type: Control flow
Bnf: [Marche_EPi | Arrêt_EPi]

Name: Niveau_Ri
Type: Continuous flow
Comment: niveau donné par la jauge de mesure du réservoir Ri

Name: Normes produits
Type: Discrete flow

Name: Num_Réservoir
Type: Discrete flow
Comment: Numéro de réservoir vers lequel le carburant doit être dirigé

Name: Num_vanne_bloquée
Type: Discrete flow
Comment: Numéro de la vanne bloquée en cas de blocage dû à une vanne

Name: ouvrir
Type: Control flow
Comment: donne l'ordre de commencer la distribution de la phase en cours

Name: Ouvrir EVi
Type: Control flow

Name: O/F_EVi
Type: Control flow
Bnf: [Ouvrir EVi | Fermer EVi]

Name: Param_physico-chimiques
Type: Discrete flow
Comment: Les paramètres physico-chimiques déterminés par l'analyseur en ligne

Name: Phase en cours
Type: Discrete flow

Name: phase suivante
Type: Control flow
Comment: Donne l'ordre de passer à la phase suivante du plan de distribution. S'il s'agit du début de l'exécution, elle est interprétée comme passer à la première phase

Name: Plan de distribution
Type: Discrete flow

Name: Plan exécuté
Type: Control flow

Name: Produit attendu
Type: Discrete flow

Name: Produit conforme
Type: Control flow

Name: Produit non conforme
Type: Control flow

Name: Qtté atteinte
Type: Control flow

Name: Qtté à livrer
Type: Discrete flow

Name: quantité
Type: Discrete flow

Name: RAZ
Type: Control flow

Name: reprendre
Type: Control flow

Name: Reprise
Type: Control flow
Comment: Reprise après passage en manuel

Name: Rep_Exec
Type: Control flow

Name: Rep_Seq
Type: Control flow
Comment: Reprise de séquence après passage en manuel

Name: Ri_plein
Type: Control flow

Name: Ri_vide
Type: Control flow

Name: Réservoir d'injection
Type: Discrete flow
Comment: Numéro de réservoir à raccorder au circuit d'injection

Name: S40_colmaté
Type: Control flow
Comment: Signal généré lorsque le filtre S40 est colmaté

Name: S41_colmaté
Type: Control flow
Comment: Signal généré lorsque le filtre S41 est colmaté

Name: Stop
Type: Control flow

Name: Surchauffe EP_i
Type: Control flow
Comment: La température de la pompe EPi à dépassé la limite tolérable

Name: surpression
Type: Control flow
Comment: Généré lorsque une surpression est détectée dans les canalisations de la station

Name: Système Prêt
Type: Control flow
Comment: Le système est prêt pour le début d'exécution du plan : pas de panne bloquante

Name: Valide/Non_valide
Type: Control flow

Name: vanne bloquée
Type: Control flow

Name: vannes
Type: Discrete flow
Comment: vannes à ouvrir ou à fermer

Name: VS_ouverte
Type: Control flow

Name: XA400
Type: Control flow
Comment: Passage du racleur par le capteur XA_400

Name: XA401
Type: Control flow
Comment: Passage du racleur par le capteur XA_401

Annexe A3 – Primitives UNIX

Les primitives de base :

- **Identification de processus** : les fonctions à valeurs entières getpid() et getppid() fournissent respectivement le numéro du processus appelant et celui du processus père (c'est-à-dire celui qui a donné naissance au processus appelant).
- **La primitive sleep(n)** : suspend l'exécution du processus appelant pour une durée de n secondes.
- **Création de processus** : primitive fork()

```
int n ;
n=fork() ;
```

fork() permet la création dynamique d'un nouveau processus (dit processus fils) qui s'exécute de façon concurrente avec le processus qui le crée (dit processus père). Le processus fils hérite un certain nombre de ses attributs du processus père. Un fils hérite :

- du même code ;
- d'une copie dans l'état courant de la zone de donnée du père ;
- de l'environnement ;
- de la priorité ;
- des descripteurs de fichiers ouverts ;
- du traitement des signaux.

La valeur de retour de fork permet de distinguer le processus fils créé (valeur 0) de son père (valeur égale au numéro du processus fils créé). C'est le seul moyen de différencier le père de son fils. Si la primitive fork() échoue alors sa valeur de retour est –1.

- **La primitive exit :**

```
int n ;
exit(n) ;
```

exit(n) met fin au processus courant avec le code de retour n (le code de retour par défaut d'un processus étant 0).

- **Synchronisation des processus** : primitive wait

```
int m, n ;
m=wait(&n) ;
```

wait provoque la suspension du processus appelant jusqu'à ce que l'un des processus fils se termine ou jusqu'à l'occurrence d'un signal émis vers le processus en attente. Si un processus fils s'est terminé avant l'exécution du wait, le retour est immédiat et la valeur de retour est le numéro du processus fils qui s'est terminé. S'il n'existe aucun processus fils alors le retour est immédiat et la valeur de retour est –1. Dans les autres cas, la valeur de retour est le numéro du processus fils qui a provoqué la fin de l'attente.

Si le processus fils s'est terminé normalement (par l'appel à la fonction exit) alors le code de retour du processus fils est placé dans les 8 bits de poids fort de l'objet pointé par &n, les autres bits étant positionné à 0.

- **Les primitives exec (commutation d'image) :**

Elles sont une famille de primitives permettant le lancement de l'exécution d'un nouveau programme par un processus appelant. Cela n'entraîne pas la création d'un nouveau processus mais seulement la modification du programme exécutable. Les descripteurs de fichiers ouverts avant l'appel d'un execl() le restent, mais il y a écrasement du tampon associé au fichier et donc risque de perte d'information. Pour forcer le vidage de ce tampon avant l'appel à exec() on utilise la fonction fflush() de la bibliothèque standard d'Entrée/Sortie (faire include stdio.h).

```
int n ;
char *ref, *arg0, …, *argn ;
char *argv[ ] ;
n=execl(ref, arg0, arg1, arg2, .., argn, 0);
n=execv(ref, argv);
```

execl execv permettent le lancement de l'exécution du fichier exécutable dont la référence est ref avec les arguments arg0, arg1, …, argn pour execl et les différentes chaînes de caractères pointées par argv pour execv. Le premier paramètre (arg0 ou argv[0]) est obligatoire et doit pointer sur une chaîne identique à celle pointée par ref. Le fichier donné en référence doit exister dans le catalogue de travail et correspondre à un fichier binaire exécutable.

execlp et execvp sont deux variantes respectivement de execl et execv, mais avec recherche du fichier exécutable dans la liste des catalogues décrits par la variable PATH (cas de l'exécution du commande Unix par exemple).

En cas d'échec de l'appel à l'une des fonctions exec, la valeur de retour est –1.

Traitement des signaux :

- **Listes de signaux :** le fichier /usr/include/signal.h contient la liste des signaux accessibles sur un système donné.
- **Emission d'un signal :** la primitive kill

```
int n, pid, sig ;
n=kill(pid, sig) ;
```

kill permet l'émission du signal de numéro sig à destination du processus de numéro pid. Si sig est nul aucun signal n'est envoyé et l'opération est réussi si et seulement si le processus de numéro pid existe. La valeur de retour n indique si l'opération est réussi (n=0) ou non (n=-1).

- **Comportement lors de la réception d'un signal** : la primitive signal

La plupart du temps, la réception d'un signal entraîne la terminaison du processus. Cependant, certains signaux tels que SIGCHDL, SIGCONT et SIGIO sont par défaut ignorés. La réception peut être provoquée par :

- une erreur de programme, et donc être émise par le système ;
- une interruption de l'utilisateur depuis son terminal ;
- kill effectué par un autre processus.
- Il est cependant possible de modifier le comportement d'un processus à la réception d'un signal. De façon générale il est possible de dérouter la réception d'un signal sur l'exécution d'une fonction comme indiqué ci-après :

```
#include<signal.h>
…
int sig, fonc();
signal(sig, fonc) ;
…
```

- le numéro de signal (sig) est alors fourni comme argument à la fonction de traitement (fonc(sig)) ;
- après exécution de fonc(sig), le traitement du processus reprend au point où il avait été interrompu. Le processus est masqué contre le signal sig durant l'exécution de fonc. Deux fonctions particulières sont prédéfinies dans le fichier /usr/include/signal.h.
  SIG_DFL spécifie un comportement par défaut (en général avortement du processus).
  SIG_IGN correspond à l'ignorance du signal.
  Vis-à-vis des signaux, un processus fils hérite du comportement de son père.

- **Primitives de manipulation du masque des signaux :**

- La primitive alarm :

```
unsigned ret, sec ;
ret=alarm(sec);
```

alarm entraîne l'envoi du signal SIGALRM au processus appelant après un laps de temps sec. Des appels successifs à la primitive provoquent une réinitialisation de l'horloge à la valeur de sec. La valeur de retour (ret) est le temps restant dans l'horloge. Après émission du signal l'horloge est réinitialisée. La primitive alarm peut être perturbée par d'autres primitives qui utilisent aussi le signal SIGALRM (par exemple sleep).

- Une interface standard pour la construction et la manipulation d'ensemble de signaux . Le type `sigset_t` correspond à de tels ensembles. Soit la liste des fonctions suivantes :

```
/*  effet : *p_ens= Φ*/
int  sigemptyset(sigset_t *p_ens) ;
/*  effet : *p_ens= *p_ens ∪ {sig}*/
int sigaddset(sigset_t *p_ens, int sig) ;
```

- L'installation manuelle d'un masque de blocage des signaux est réalisée par un appel à la fonction :

```
int  sigprocmask(int  op,  const  sigset_t  *p-ens,  sigset_t
*p_ens_ancien) ;
```

Le nouveau masque est construit à partir de l'ensemble pointé par p_ens et éventuellement du masque antérieur dont la valeur est récupérée au retour à l'adresse p_ens_ancien (si ce pointeur n'est pas le pointeur NULL). Le paramètre op permet de déterminer le nouvel ensemble (si op=SIG_SETMASK, alors le nouveau masque est celui référencé par *p_ens).

Primitives de communication :

- **Communication par tubes** : la primitive pipe permet la création d'un tube de communication

```
int n, desc[2] ;
n=pipe(desc) ;
```

Au retour de l'appel :
- desc[0] est le numéro du descripteur pour la lecture ;
- desc[1] est le numéro du descripteur pour l'écriture.

La valeur de retour n est 0 si la création s 'est bien passée et –1 en cas de problème. Il y a héritage des tubes créés lors de l'exécution des primitives fork et exec. Deux processus pourront donc communiquer via un tube si ce tube a été créé par un ancêtre commun. Le système assure de plus certains contrôles :
- si un processus tente d'écrire dans un tube alors que personne n'est en mesure d'y lire, le signal SIGPIPE est envoyé au processus. Le processus est alors interrompu s'il ne traite pas le signal en utilisant la primitive signal ;
- la primitive read sur un tube a pour valeur 0 si le tube est vide et si aucun processus n'est en mesure d'écrire dans le tube.

- **Communication par tube nommé**

La création d'un i-noeud de type S_FIFO et d'une entrée dans un répertoire lui correspondant est réalisée dans un programme au moyen de la fonction suivante :

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *ref, mode_t mode);
```

Le paramètre ref définit le chemin d'accès au tube et le paramètre mode les droits d'accès des différents utilisateurs à cet objet, éventuellement masqués par la valeur courante du masque dans le processus. Le paramètre mode est construit comme le paramètre mode de la fonction open sur les fichiers ordinaires.

Synchronisation de processus par ensemble de sémaphores :

- **La primitive semget** : elle a le prototype suivant :

```
#include <sys/sem.h>
int semget(key_t cle, int nb_sem, int option);
```

Il s'agit d'une primitive dont l'appel est un préalable à toute utilisation d'un ensemble de sémaphores par un processus. Elle permet la création d'un ensemble de sémaphores ou la recherche de l'identification d'un ensemble existant à partir de son clé. Le paramètre option est construit comme combinaison (opérateur ou bit à bit |) des constantes IPC_CREAT, IPC_EXCL et des droits d'accès. Le paramètre nb_sem correspond au nombre de sémaphores de l'ensemble. La demande d'acquisition de l'identification d'un ensemble de sémaphores existant suppose que le nombre nb_sem donné en paramètre ne soit pas supérieur au nombre de sémaphore de l'ensemble. Rappelons que les différents sémaphores d'un ensemble en contenant n auront comme identification dans cet ensemble 0, 1, …, n-1.

L'appel de cette primitive renvoie l'identification d'un ensemble de sémaphore ou –1 en cas d'erreur.

- **La primitive semop** : c'est par son intermédiaire qu'un processus connaissant l'identification d'un ensemble de sémaphores pourra réaliser des (ensembles d') opérateurs sur les sémaphores de cet ensemble. La forme générale de la primitive est la suivante :

```
#include <sys/sem.h>
int semop(int semid, struct sembuf *tab_op, int nb_op);
```

Les nb_op opérations placées à l'adresse tab-op sont réalisées atomiquement . La structure sembuf correspond à une opération sur un sémaphore (l'augmenter d'une valeur entière, le diminuer d'une valeur entière ou attendre qu'il devienne nul).

```
struct sembuf {
    unsigned short int sem_num ; /* numéro de sémaphore*/
    short              sem_op; /* opération sur le sémaphore*/
    short              sem_flg ; /* option*/
};
```

C'est le signe du champ sem_op d'un objet qui détermine l'opération :

* négatif pour l'opération Pn (le sémaphore est décrémenter de n),;

* positif pour l'opération Vn (le sémaphore est incrémenter de n) donc tout les processus attendant l'augmentation de ce sémaphore sont réveillés ;

- nulle pour l'opération Z (le processus est bloqué tant que le sémaphore n'est pas nul).

- **La primitive de contrôle semctl** : elle permet de réaliser, outre les opérations de consultation et modification d'une entrée de la table des ensembles de sémaphores et suppression d'une entrée dans cette table, des initialisations de sémaphores et la consultation de différents valeurs (sémaphores, nombre de sémaphores en attente d'augmentation de valeur ou de nullité). La forme générale de la primitive est :

```
#include <sys/sem.h>
int semctl(int semid, int semnum, int op, … /* arg */);
```

Le type du quatrième paramètre optionnel de la fonction dépend de la valeur du paramètre op. De même, la valeur de ce paramètre détermine si le paramètre num est un numéro ou un nombre de sémaphores. Le tableau précédent résume l'effet de la primitive et l'interprétation des différents paramètres en fonction de la valeur du paramètre op.

Voici un exemple d'initialisation à 1 du premier sémaphore de l'ensemble contenant un seul élément:

```
int n;
/* valeur initiale de l'unique sémaphore de l'ensemble identifié par semid */
static ushort init_sem[]={ 1 } ;
…
n= semctl(semid,1,SETALL,init_sem) ;
…
```

Annexe A4 – Notations HOOD

OBJECT Nom_Objet **IS**

DESCRIPTION

Stratégie informelle décrivant le rôle de l'objet.

Implementation or synchronisation constraints

Description informelle de toute synchronisation ou contraintes pesant sur les opérations fournies ou internes

PROVIDED INTERFACE

TYPES

Définition des types fournis et/ou utilisés par les opérations (syntaxe Ada)

DATA

Définition des données fournies aux autres objets

OPERATIONS

Définition des opérations et paramètres fournis aux autres objets (syntaxe Ada)

EXCEPTIONS

Exceptions qui pourraient être fournies aux objets utilisateurs

REQUIRED INTERFACE

USED OBJECTS

Liste des objets utilisés

TYPES

Définition des types requis des objets utilisés (syntaxe Ada)

DATA

Description des données requises des objets utilisés (syntaxe Ada)

OPERATIONS

Liste des opérations requises des objets utilisés sous la forme Objet Opération

EXCEPTIONS

Exceptions qui pourraient être propagées à partir des objets utilisés

INTERNALS

INCLUDED OBJECTS

Liste des objets enfants internes

TYPES

Déclaration des types utiles

DATA

Déclaration des données internes

OPERATIONS

Déclaration des opérations internes (syntaxe Ada)

OBJECT CONTROL STRUCTURE

Description des opérations et de leurs synchronisations. Description (en utilisant la sémantique du Rendez-vous Ada, i.e. Instruction SELECT) des interactions entre les opérations fournies, les opérations internes et les opérations utilisées en explicitant les liens avec l'état de l'objet.

OPERATION CONTROL STRUCTURE

Description de chaque opération interne.

Description de chaque opération fournie.

END Nom_Objet

Annexe A5 – Extrait d'une bibliothèque C++

```
template <class TcleP, class TcleS, class Telmt>
class ListDC {
  // Gère une liste de valeur de type Telmt triées par rapport à deux clés. TcleP est le type de la
  // clé principale et TcleS est le type de la deuxième clé.

public:
  ListDC();
  ListDC(const ListDC<TcleP,TcleS,Telmt> &UneListe);
  ListDC &operator=(const ListDC<TcleP,TcleS,Telmt> &UneListe);
  ~ListDC();

  void Insere(TcleP LaCleP, TcleS LaCleS, Telmt *Elmt);
     // insere Elmt à sa place dans la liste. La liste est triée de façon décroissante par rapport à la
     // clé principale et de façon croissante par rapport à la deuxième clé. La liste autorise la
     // répétition des valeurs pour les deux clés.

  int Supprime(TcleP LaCleP, TcleS LaCleS) ;
    // supprime la première occurrence de l'entrée (LaCleP,LaCleS) dans la liste. Retourne 1
    // s'il y a eu une suppression 0 sinon.

  Telmt *Get(TcleP LaCleP, TcleS LaCleS) ;
    // Retourne  l'élément correpondant à la première occurrence de l'entrée  (LaCleP,LaCleS)
    // dans la liste. Retoune NULL si aucune occurrence n'est trouvée.

  unsigned int Longueur() ;
    // Calcule le nombre d'élements dans la liste.

  friend class ItListDC<TcleP,TcleS,Telmt>;

private:
 …
}

template<class TCP, class TCS, class Telt>
class ItListDC{
   // itérateur sur un objet ListDC.

public :
    ItListDC(ListDC<TCP,TCS,Telt> *LaListe>
    // LaListe doit pointer sur l'objet sur lequel s'effectuera l'itération.

  void First() ; // se positionner au début
  void Last() ; // se positionner à la fin  ;
  int End() ; // retourne 1 si en fin de liste 0 sinon ;
  int Begin() ; // retourne 1 si en début de liste 0 sinon ;
  int Next() // se positionner sur l'entrée suivante de la liste. Retourne 1 si la progression a eu
                // lieu 0 sinon.
```

```
  int Previous() // se positionner sur l'entrée précédente de la liste. Retourne 1 si le
                 // déplacement a eu lieu 0 sinon.
Telt* GetElt() // Retourne l'élément à la position courante de la liste. Peut provoquer
                 // l'exception ElisteVide.
TCP GetCleP() ; // Retourne la clé principale de l'élément à la position courante de la liste.
                 // Peut provoquer l'exception ElisteVide.
TCS GetCleS() ; // Retourne la deuxième de l'élément à la position courante de la liste.
                 // Peut provoquer l'exception ElisteVide.


private :
…
}
```

On supposera aussi disposer des deux fonctions suivantes :

- InitBaseTemps() qui permet de définir un instant de départ ;
- Delay(unsigned long Periode) qui permet de bloquer l'appelant jusqu'à ce que l'instant courant devienne le prochain multiple entier de Période.

Annexe A6 – Documents relatifs au système d'information

Modèle de Bon de commande

No Commande : 01/967 Date Commande : 21/05/2001

Code Client : 07 Nom Client : SHELL

Code produit	Abrév.	Nom Produit	No Rés. Dest.	Qte. commandée
01	EN	Essence Normale	R172	3000
02	ES	Essence Super	R271	5000
…	…	…	…	…

Modèle de plan de distribution

Cycle 01-26			
Phase	**Produit**	**Réservoir**	**Quantité**
1	GO	R421	3000
2	GO	R442	3000
3	GO	R451	2000
4	PL	R522	500
5	PL	R541	1000
6	SSP	R331	1000
7	SSP	R351	1000
8	ES	R261	3000
9	ES	R271	2000
10	EN	R161	500
11	EN	R152	1000
12	PL	R552	1000

Modèle de plan d'expédition

Cycle 01-26	
Produit	**Quantité**
GO	8000
PL	1500
SSP	2000
ES	5000
EN	1500
PL	1000