



Projet LO41

Une chaîne de montage en anneau

Yoann CAPLAIN

9 janvier 2014
Semestre A2013

Table des matières

1	Introduction	2
1.1	Rappel du sujet : La chaîne de montage en anneau	2
1.2	Analyse du sujet	2
1.2.1	Les composants	2
1.2.2	Les opérations	2
1.2.3	Les produits	3
1.2.4	L'anneau	3
1.2.5	Les robots	3
2	Analyse et conception	5
2.1	UML : Langage de modélisation	5
2.1.1	Le diagramme de classe	6
3	Réalisation du projet	7
3.1	Structure générale	7
3.2	Description des fichiers sources	7
3.2.1	Anneau	7
3.2.2	Robot	9
3.2.3	Operation	13
3.2.4	Composant	13
3.2.5	Produit	13
3.2.6	SortieProduit	14
3.2.7	PileFIFO	14
3.2.8	voidBuf	15
3.3	Le choix de l'utilisation des threads	15
3.3.1	La synchronisation entre les différents threads	15
3.4	Le Factory Pattern : patron de conception	15
3.4.1	Implémentation dans le projet	16
3.4.2	Conclusion du Pattern Factory et son utilité	17
3.5	Le producteur et les "lecteurs"	17
3.5.1	ProducteurComposant	17
4	Modélisation en Réseau de Pétri	18
5	Conclusion	19
A	Documentation	20
A.1	Factory Pattern	20
A.2	Mes réalisations (github)	20
B	Exemples de code	21

Chapitre 1

Introduction

Dans le cadre de l'unité de valeur "LO41 : Système d'exploitation : Principes et Communication", il nous est demandé de réaliser un projet permettant d'appliquer les différents concepts étudiés tout au long des cours, travaux pratiques et travaux dirigés. Le projet "Chaîne de montage en anneau" qui nous a été soumis permet de mettre en oeuvre ces concepts.

1.1 Rappel du sujet : La chaîne de montage en anneau

Le but du projet est de réaliser un approvisionnement de composants matériels et la gestion du processus de fabrication d'une gamme de produits.

On doit mettre en oeuvre une chaîne de montage "en anneau" pilotée par des robots, sa particularité est que la chaîne a 16 sections pouvant contenir soit des composants servant à la fabrication de produit, des produits en cours de fabrication ou des produits finaux.

Ces robots sont au nombre de 6 et ont deux modes différents (mode normal et dégradé), ils ont chacun leurs caractéristiques propres (opération normal et opérations en mode dégradé).

Les composants servent à créer le produit (1ère opération du produit).

Le projet doit être codé en C et utilisé l'une (ou plusieurs) des méthodes de synchronisation vu en cours :

- Mutex
- Mémoire partagée
- Flux (tubes)
- Sémaphore
- etc

1.2 Analyse du sujet

1.2.1 Les composants

Un composant sert juste à créer un produit et ils sont utilisés lors de la 1ère opération, ils sont donc liés aux opérations et un seul type de composant est nécessaire pour créer un produit.

On a 4 composants différents :

- C1
- C2
- C3
- C4

1.2.2 Les opérations

Les opérations sont des actions réalisées par les robots.

En mode normal une opération n'est faite que par un seul robot contrairement au mode dégradé où une

opération peut être liée à plusieurs robots.

On a 6 opérations différentes :

- Op1
- Op2
- Op3
- Op4
- Op5
- Op6

1.2.3 Les produits

Pour créer un produit il faut les composants nécessaires pour la 1ère opération du produit, les opérations suivantes n'ont pas besoin d'utiliser de composants.

On a 4 produits différents (et leur séquence) :

- Prod1 (Op1, Op2, Op3, Op5)
- Prod2 (Op2, Op4, Op1, Op6)
- Prod3 (Op3, Op1, Op5, Op1, Op3)
- Prod4 (Op4, Op6, Op1)

1.2.4 L'anneau

L'anneau est circulaire et possède 16 sections qui peuvent stocker des composants et des produits. L'anneau est accédé par les robots (6 robots), la chaîne d'approvisionnement des composants et la chaîne de sortie des produits finis.

Algorithme de l'anneau

L'algorithme décrit ci-dessous montre principalement comment ça se passe pour le code, il y a certaines parties du code omises volontairement ici.

```
1 Tant que continuer
  Lock mutex anneau
3  Tant que qqChoseAccedeAnneau > 0
    Attendre sur condition 'Acces a anneau'
5  Fin Tant que
    Tourner l'anneau
7  Signaler tous ceux qui attendent sur la condition que 'Anneau a tourne'
  Unlock mutex anneau
9 Fin Tant que
```

Listing 1.1 – Algorithme de l'anneau simplifié

1.2.5 Les robots

Les robots accèdent à l'anneau pour récupérer des composants ou des produits non finis, ils y déposent seulement des produits.

Ils créent des produits à partir des composants stockés.

Les modes des robots

Les robots ont deux modes qui sont Normal et Dégradé. En mode normal les robots n'ont qu'une seule opération disponible, en mode dégradé ils ont la possibilité de réaliser plusieurs opérations.

Algorithme des robots

L'algorithme décrit ci-dessous montre principalement comment ça se passe pour le code, il y a certaines parties du code omis volontairement ici.

Dans cet algorithme les mutex ne seront pas décrit.

```
1 Tant que continuer
  Attendre sur condition que 'Anneau a tourne'
3 Si le robot peut creer le produit Alors
  Creer le produit
5 Sinon
  Si l''emplacement de l''anneau pour le robot est un composant Alors
7    Si le composant est utile pour le robot Alors
      Prendre le composant
9    Fin si
  Sinon si l''emplacement de l''anneau pour le robot est un produit Et que le robot ne
  possede pas deja un produit Alors
11    Si l''operation suivante peut etre faite Alors
      On prend le produit et on fait l''operation
13    Fin si
  Fin si
15
  Si l''emplacement de l''anneau pour le robot est vide
17    Si le robot a besoin de poser un produit
      Poser le produit sur l''anneau
19    Fin si
  Fin si
21 Fin si
  Signaler Anneau sur condition 'Acces a anneau'
23 Fin Tant que
```

Listing 1.2 – Algorithme des robots simplifié

Chapitre 2

Analyse et conception

2.1 UML : Langage de modélisation

L'analyse d'un projet est primordiale, ça permet de clarifier les aspects techniques et ce que l'on veut faire sur notre projet. C'est pourquoi cette analyse est très importante et qu'il faut qu'elle soit correct avant le début de la conception du projet car selon l'erreur d'analyse celle-ci peut soit être résolu avec quelques modifications ou nécessite de recommencer la conception du projet.

L'analyse qui a été faite sur le projet Chaîne de montage en anneau a permis de mettre en place une structure du programme futur.

En sachant que l'UML est orientée objet il a été utilisé que pour clarifier le projet et voir quels vont être les fichiers sources du programme.

De plus cette analyse a été faite de façons à avoir un code modulable grâce aux factory et une abstraction de la représentation des produits, des composants et des opérations.

2.1.1 Le diagramme de classe

Le diagramme de classe a été la base du projet, certaines classes vu ci-dessous ne sont pas dans le projet étant donné que c'était une 1ère approche du projet et qu'une adaptation en C (non orientée objet) était nécessaire, aussi les "factory" ne sont pas représentées.

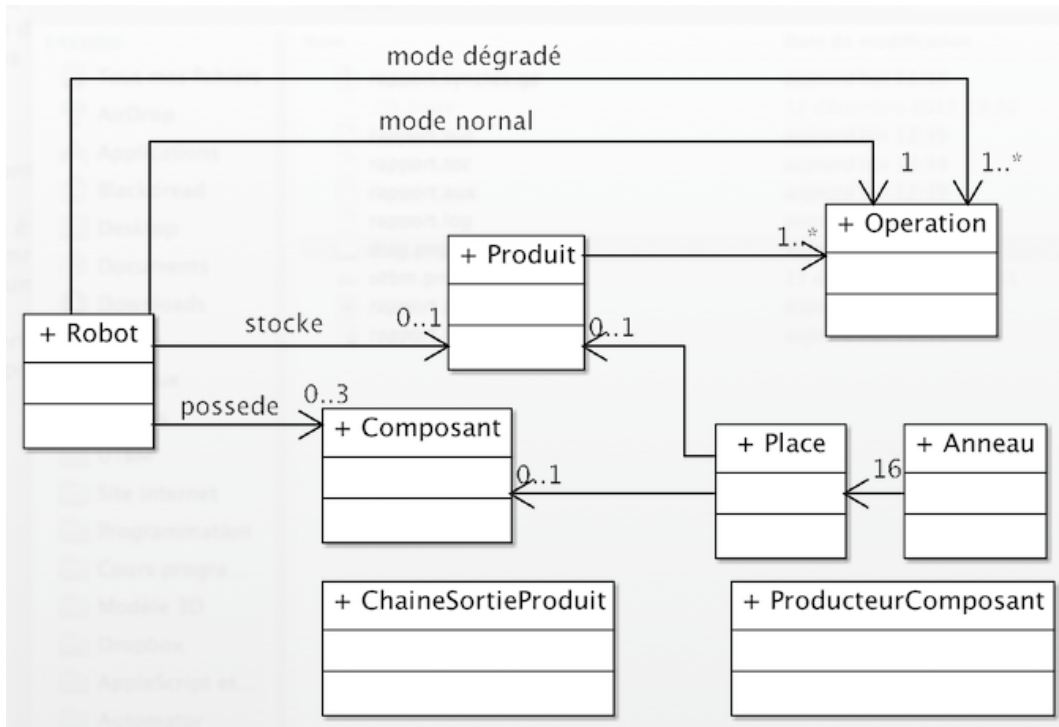


FIGURE 2.1 – Diagramme de classe

Chapitre 3

Réalisation du projet

3.1 Structure générale

Le programme est réalisé en C et la synchronisation est faite avec des mutex sur les thread.

3.2 Description des fichiers sources

La plupart des fichiers sources sont commentés donc nous ne ferons pas forcément de descriptions complètes pour certaines de ces sources.

De plus les fichiers .c ne seront pas dans le rapport étant donné le volume de ligne que cela représente, voir le code source donné à part du rapport (ou cf A.2 page 20).

3.2.1 Anneau

Il y a un mutex (mutex_anneau) et deux conditions (cond_anneauATourner, cond_anneauAttenteAcces). Le mutex permet la synchronisation sur l'anneau pour tout les éléments qui veulent y accéder. La 1ère condition permet de notifier ceux qui attendent que l'anneau est tourné. La 2ème condition permet de réveiller l'anneau dès qu'un élément à fini d'accéder à l'anneau, celui-ci est réveillé et vérifie s'il y a encore des éléments qui accède à l'anneau si oui il se rendort sinon il tourne.

La définition des emplacements des robots, chaîne entrée et chaîne sortie se trouvent ici. L'initiation de l'anneau se fait avec la fonction "initAnneau".

```
1 #ifndef LO41__anneauCircu
2 #define LO41__anneauCircu
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <pthread.h>
7 #include "Robots.h"
8 #include "Produit.h"
9 #include "ProducteurComposant.h"
10
11 #define TAILLE_ANNEAU_MAX 16
12
13 #define POS_C_ENTREE 1
14 #define POS_C_P_SORTIE 3
15 #define POS_ROBOT_1 5
16 #define POS_ROBOT_2 7
17 #define POS_ROBOT_3 9
18 #define POS_ROBOT_4 11
19 #define POS_ROBOT_5 12
20 #define POS_ROBOT_6 14
21
22 #define TYPE_PROD 50
23 #define TYPE_COMP 51
```



```

#define TYPE_RIEN 0
25 pthread_mutex_t mutex_anneau;
27 // Permet de reveiller robots et autres en attente
pthread_cond_t cond_anneauATourner;
29
// l'anneau attend tant que les elements continue d'accéder a l'anneau
31 pthread_cond_t cond_anneauAttenteAcces;

33 void initAnneau();

35 int bufComposantOuProduit[TAILLE_ANNEAU_MAX]; // 51 c et 50 Prod <= ce n'est pas forcément
// utile car on peut supposer que tout les pointeur de type produit auront une valeur
// superieur a 1000 donc on pourrait juste tester la valeur du pointeur, mais qui a faire
// bien je mets ce 2eme tableau
void *bufferAnneau[TAILLE_ANNEAU_MAX];

37 void* prendreElement(int positionBuffer);
39 void ajouterElement(void *elem,int positionBuffer,int type);
41 int isPositionBufLibre(int pos);
int getPositionType(int pos);

43 // renvoie la position du robot sur le buffer
int getPositionDuRobot(int numRobot);

45 // Ne pas trop utiliser cette fonction
47 void* getPointeur(int pos);

49 int continuerAnneau;

51 int nbElementDansBuffer;

53 int nbElementQuiAccedeAnneau;
//int nbElementQuiVeulentAccedeAnneau;

55 void printAnneau();

57 void th_tournerAnneau();

59 #endif

```

Listing 3.1 – Représentation de l’anneau

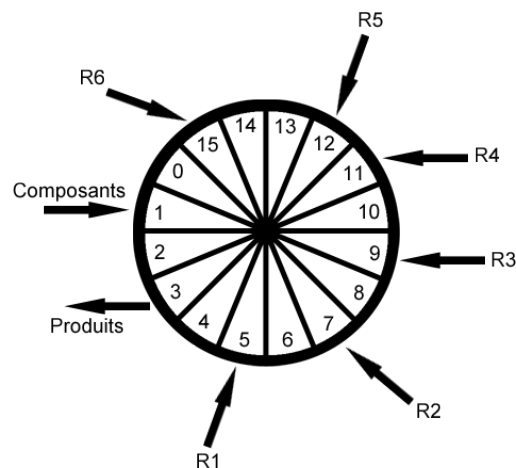


FIGURE 3.1 – Aperçu de l’anneau

L'anneau est l'élément critique du projet :

- Attendre qu'aucun élément accède l'anneau
- Tourner (déplacer les éléments d'un cran)
- Notifier ceux qui attendent que l'anneau est tourné
- Recommencer

```
void th_tournerAnneau() {
2   while(continuerAnneau == 1){

4       pthread_mutex_lock(&mutex_anneau);
       while(nbElementQuiAccedeAnneau != 0){
6           printf("Anneau mis en attente\n");
           pthread_cond_wait(&cond_anneauAttenteAcces, &mutex_anneau);
8       }

10      // tourne les elements
       int i;
12      int tmp = bufComposantOuProduit[TAILLE_ANNEAU_MAX-1];
       void* tmp2 = bufferAnneau[TAILLE_ANNEAU_MAX-1];

14      for (i=TAILLE_ANNEAU_MAX-1; i>0; i--){
16          bufComposantOuProduit[i] = bufComposantOuProduit[i-1];
          bufferAnneau[i] = bufferAnneau[i-1];
18      }
       bufComposantOuProduit[0] = tmp;
       bufferAnneau[0] = tmp2;
20      //printf("Anneau a tourner\n");

22      pthread_cond_broadcast(&cond_anneauATourner);
       pthread_mutex_unlock(&mutex_anneau);
24  }
26  printf("Thread Anneau terminer\n");
}
```

Listing 3.2 – Fonction du thread de l'anneau

3.2.2 Robot

La définition du robot est une structure, on peut ainsi créer plusieurs robots à partir de cette structure. De plus la création est simplifiée et centralisée avec FactoryRobot.

```
1  #ifndef LO41_Robots_h
   #define LO41_Robots_h
3
   /* ca se trouve ici car sinon dans FactoryRobot j'avais une inclusion circulaire , a
      corriger
5  #define R1 1
   #define R2 2
7  #define R3 3
   #define R4 4
9  #define R5 5
   #define R6 6
11 /* */
   #define TAILLE_MAX_NB_OP_DEGRADE 4
13 #define TAILLE_MAX_NB_COMP_GARDE 3

15 #define MODE_NORMAL 1
   #define MODE_DEGRADE 2
17
   #include <stdio.h>
19 #include <stdlib.h>
   #include "Produit.h"
21 #include "Operation.h"
   #include "Composant.h"
23 #include "anneauCircu.h"
   #include "FactoryProduit.h"
```

```

25
27 typedef struct _robot{
28     int continuer; // ajouter a la fin , c'est juste pour arreter le thread a partir du main
29     int mode;
30
31     int opNormal;
32
33     // Nb d'element => Ou remplacer par un pointeur et garder le nb d'element de ce tableau
34     int nbOpDegrade;
35     int opDegrade[TAILLE_MAX_NB_OP_DEGRADE]; // taille statique ou pointeur
36
37     // peut posseder un produit
38     produit* produit;
39     // peut posseder des composants => supposer identique car dans le sujet les produits
40     // commencent avec un seul type de composant
41     int composant[TAILLE_MAX_NB_COMP_GARDE];
42 }robot;
43
44 void changerModeRobot(int mode, robot *robot);
45
46 // Regarde si le produit passer en parametre peut etre utiliser par le robot (si le robot
47 // peut faire l'operation)
48 int isProduitNextOpCanBeDone(produit *prod, robot *robot);
49
50 // renvoie 1 si oui. Regarde si ce composant est utile au robot ce qui signifie si ce robot
51 // peut creer un produit a partir de ce composant, ne prends que les composants d'un meme
52 // type si 1 type a deja etait pris.
53 int isComposantUsefulToBeTaken(int composant, robot *robot);
54
55 // renvoie le nb de composant
56 int countNbComposant(robot *robot);
57
58 // renvoie 1 si le robot possede un produit a remettre sur l'anneau
59 int isRobotNeedToGiveProduit(robot *robot);
60
61 // renvoie 1 si le robot est plein de composants
62 int isRobotFullOfComposant(robot *robot);
63
64 // ajoute le composant -> attention a la limite en taille -> si ca depasse le composant est
65 // perdu
66 void ajouterComposant(int composant, robot *robot);
67
68 // supprime les composants
69 void enleverLesComposants(robot *robot);
70
71 // supprime un composant de type donne (part de la fin a 0)
72 void enleverLeComposant(int composant, robot *robot);
73
74 // Num robot et arg[1] un pointeur sur un robot
75 void th_robot(void* arg[]);
76 //void th_robot(void *numRob, robot *robot);
77
78 // la fonction qui enleve les composants, creer le produit puis...
79 void procederALaCreationDuProduit(int numProd, robot *robot);
80
81 // fonction qui fais les operations sur le produit (faire l'op suivante de la suivante si le
82 // robot peut le faire ? (robot en mode degrade donc))
83 void procederOpSuivanteDuProduit(robot *robot);
84
85 // On remet sur l'anneau le produit (toute verification faite cette fonction
86 void procederRemiseDuProduitSurAnneau(int positionSurAnneau, robot *robot);

```

87 #endif

Listing 3.3 – Représentation des robots

On a une seule fonction qui permet de gérer n'importe quel robot, c'est donc modulable et le changement ne se fait que dans cette fonction, on n'a pas de code copier.
Et la création des différents robots se fait grâce au FactoryRobot.

```
1 void th_robot(void* arg []) {
2 //void th_robot(void *numRob, robot *robot){
3     robot *robot = arg[1];
4     int numRobot = (int) arg[0];
5
6     while(robot->continuer == 1){
7         pthread_mutex_lock(&mutex_anneau);
8         // On attend tjr que l'anneau est tournee
9         pthread_cond_wait(&cond_anneauATourner, &mutex_anneau);
10        nbElementQuiAccedeAnneau++;
11
12        // ***** ALGO *****
13        // regarder si on veut prendre des composants
14        // regarder le composant -> on le prends ?
15        // on prend le composants si on a en a besoin pour creer un produit (produit qui a
16        // besoin de plusieurs compo)
17        // Besoin que de 1 composant -> ce composant nous est presente => on le prends et on
18        // creer le produit
19
20        // regarder le produit -> possible de faire l'action ?
21        // possible mais pas les composants, on attends les prochains composants
22        // possible et ce n'est pas la 1ere op donc on fait l'op
23        // ***** ALGO *****
24
25        int nbCompo = countNbComposant(robot);
26        int numProdACreer = 0;
27        //if(nbCompo > 0)
28        numProdACreer = hasEnoughComposantToDoProd(nbCompo, robot->composant[0]);
29        if(nbCompo > 0 && numProdACreer > 0 && isRobotNeedToGiveProduit(robot) == 0){
30            // supposition -> un seul type de composant
31            nbElementQuiAccedeAnneau--;
32            pthread_cond_signal(&cond_anneauAttenteAcces);
33            pthread_mutex_unlock(&mutex_anneau);
34
35            printf("Robot%d procede creation produit\n", numRobot);
36            procederALaCreationDuProduit(numProdACreer, robot);
37
38            continue;
39        } else {
40
41            pthread_mutex_unlock(&mutex_anneau);
42            int typePos = getPositionType(getPositionDuRobot(numRobot));
43
44            if(typePos == TYPE_COMP){
45                int composantSurAnneau = (int) getPointeur(getPositionDuRobot(numRobot));
46
47                // On ne peut pas encore creer de produit => on cherche a avoir les
48                // composants necessaires
49                if(isRobotAlreadyHaveComposant(robot) == 1){
50                    // Un produit n'a besoin que d'1 type de composant donc on ne va pas
51                    // chercher a stocker differents composant
52                    if(composantSurAnneau == robot->composant[0] && isRobotFullOfComposant(
53                        robot) == 0){
54                        // on prend le composant
55                        ajouterComposant((int) prendreElement(getPositionDuRobot(numRobot)),
56                            robot);
57                        typePos = TYPE_RIEN;
58                        printf("Robot%d pris composant %d 1er nb=%d\n", numRobot,
59                            composantSurAnneau, countNbComposant(robot));
```

```

55         }
56     } else {
57         // on prend le composant seulement si utile pour ce robot
58         if (isRobotFullOfComposant(robot) == 0 && isComposantUsefulToBeTaken(
composantSurAnneau, robot) == 1) {
59             ajouterComposant((int)prendreElement(getPositionDuRobot(numRobot)),
robot);
60             typePos = TYPE_RIEN;
61             printf("Robot%d pris composant %d 2eme nb=%d\n", numRobot,
composantSurAnneau, countNbComposant(robot));
62         }
63     } else if (typePos == TYPE_PROD && isRobotNeedToGiveProduit(robot) == 0) {
64         produit* produitSurAnneau = (produit*)getPointeur(getPositionDuRobot(
numRobot));
65         if (isProduitNextOpCanBeDone(produitSurAnneau, robot) == 1) {
66             // on peut faire l'operation
67             // regarder si on peut faire plusieurs op si on est en mode degrade, c
bete de remettre le produit si on peut faire les 2 operations qui suivent
68             robot->produit = prendreElement(getPositionDuRobot(numRobot));
69
70             pthread_mutex_lock(&mutex_anneau);
71             nbElementQuiAccedeAnneau--;
72             pthread_cond_signal(&cond_anneauAttenteAcces);
73             pthread_mutex_unlock(&mutex_anneau);
74
75             printf("Robot%d procede OpSuivante\n", numRobot);
76             procederOpSuivanteDuProduit(robot);
77
78             continue;
79         }
80     }
81 }
82
83 // Pas sur de garder ca dans ce else...a voir -> le but est de pouvoir remettre
le produit si il est possible de le mettre car on a pris le composant qui etait sur l'
anneau ou s'il y avait rien
84 if (typePos == TYPE_RIEN) {
85     if (isRobotNeedToGiveProduit(robot) == 1) {
86         // Poser le produit
87         procederRemiseDuProduitSurAnneau(getPositionDuRobot(numRobot), robot);
88         printf("Robot%d produit remis\n", numRobot);
89         //printAnneau();
90     }
91 }
92 }
93 /*
94 if (robot->produit != NULL) {
95     printf("Robot%d stocke un produit%d\n", numRobot, robot->produit->numProduit);
96     if (isRobotNeedToGiveProduit(robot)) {
97         printf("Et a besoin de donner produit / %d\n", nbElementDansBuffer);
98         printAnneau();
99     }
100 }
101 printf("Robot%d fin boucle et mode %d\n", numRobot, robot->mode);
102 pthread_mutex_lock(&mutex_anneau);
103 nbElementQuiAccedeAnneau--;
104 pthread_cond_signal(&cond_anneauAttenteAcces);
105 pthread_mutex_unlock(&mutex_anneau);
106 */
107 }
108 if (robot->produit != NULL)
109     free(robot->produit);
110 printf("Thread robot%d terminer\n", numRobot);
111 }

```

Listing 3.4 – Fonction thread des robots

3.2.3 Operation

La représentation des opérations est juste des "define", on a aucune fonction.

```
1 #ifndef LO41__Operation_h
2 #define LO41__Operation_h
3
4 #define Op1 1
5 #define Op2 2
6 #define Op3 3
7 #define Op4 4
8 #define Op5 5
9 #define Op6 6
10
11 #endif
```

Listing 3.5 – Représentation des opérations

3.2.4 Composant

La représentation des composants est juste des "define", on a aucune fonction.

```
1 #ifndef LO41__Composant
2 #define LO41__Composant
3
4 #define C1 101
5 #define C2 102
6 #define C3 103
7 #define C4 104
8
9 #endif
```

Listing 3.6 – Représentation des composants

3.2.5 Produit

La définition du produit est une structure, on peut ainsi créer plusieurs produit à partir de cette structure. De plus la création est simplifiée et centralisée avec FactoryProduit.

La fonction isProduitT0Done renvoie 1 si la 1ère opération a été faite.

```
1 #ifndef LO41__Produit
2 #define LO41__Produit
3
4 #include "PileFIFO.h"
5
6 typedef struct _produit{
7     int numProduit; //juste pour l'identifier
8
9     // Ici on ne cherche pas a stocker differents composant pour creer un produit
10    // car le sujet montre que pour un produit on a besoin que d'un seul type de composant
11    int nbCNecessaire;
12    int composant;
13
14    // le 1er element est l'operation a faire prochainement
15    pileFIFO listeDesOperationRequired;
16
17 }produit;
18
19 int isProduitT0Done(const produit *prod);
20 int isProduitDone(const produit *prod);
21
22 int popNextOp(produit *prod);
23
24 // n'enleve pas l'operation
25 int getNextOp(produit *prod);
```

```
27 #endif
```

Listing 3.7 – Représentation des produits

On peut voir que le code source des produits est simple mais est suffisant pour définir un produit.

```
1 #include "Produit.h"
3 int isProduitT0Done(const produit *prod){
4     if(prod->nbCNecessaire == 0)
5         return 1;
6     return 0;
7 }
8 int isProduitDone(const produit *prod){
9     return isEmpty(&(prod->listeDesOperationRequired));
10 }
11 int popNextOp(produit *prod){
12     if(isProduitDone(prod) == 0){
13         return pop(&(prod->listeDesOperationRequired));
14     }
15     return -1;
16 }
17 }
18 int getNextOp(produit *prod){
19     return getSommetValue(&(prod->listeDesOperationRequired));
20 }
21 }
```

Listing 3.8 – Code source des produits

3.2.6 SortieProduit

```
1 #ifndef LO41__SortieProduit
2 #define LO41__SortieProduit
3
4 #include "voidBuf.h"
5 #include <pthread.h>
6 #include "anneauCircu.h"
7 #include "ProducteurComposant.h"
8
9 int continuerSortie;
10
11 int nbProduitSortie;
12
13 // Pour le moment je choisi de ne faire sortir que des produits
14 voidBuf produitsSortis;
15
16 void initSortieProduit();
17
18 void th_verifSortieProduit();
19
20 #endif
```

Listing 3.9 – Représentation de la sortie des produits finis

3.2.7 PileFIFO

```
1 #ifndef LO41__PileFIFO
2 #define LO41__PileFIFO
3
4 #include <stdio.h>
5 #include <stdlib.h>
6
```

```

8 typedef struct _maillon{
    struct _maillon *next;
    int value; // je pourrais mettre a la place de int -> void* ainsi j'aurais une pile fifo
    pouvant contenir tout
10 }maillon;

12 typedef struct _pileFIFO{
    maillon *somet;
    //int nbElement;
14 }pileFIFO;

16 void initPileFIFO(pileFIFO* pile);

18 void push(pileFIFO* pile , int value);
20 int pop(pileFIFO* pile);
int isEmpty(const pileFIFO* pile);
22 void vider(pileFIFO* pile);

24 int getSometValue(pileFIFO* pile);

26 // renvoie -1 si a atteint la fin
28 int getValueAt(maillon* mail, int pos);

30 #endif

```

Listing 3.10 – Représentation de la pile FIFO

3.2.8 voidBuf

```

1 #ifndef LO41_voidBuf
2 #define LO41_voidBuf

4 #include <stdio.h>
#include <stdlib.h>

6
8 typedef struct {
    int capa;
    int nbElement;
10 void **buffer;
} voidBuf;

12 void initBuf(voidBuf* buf, int capa);
void appendBuf(voidBuf* buf, void* s);
void* readBuf(voidBuf* buf, int pos);
16 void printVoidBuf(voidBuf* buf);

18 void libererBuf(voidBuf* buf);

20 #endif

```

Listing 3.11 – Représentation d'un buffer de void*

3.3 Le choix de l'utilisation des threads

3.3.1 La synchronisation entre les différents threads

3.4 Le Factory Pattern : patron de conception

La fabrique (factory method) est un patron de conception créationnel utilisé en programmation orientée objet. Elle permet d'instancier des objets dont le type est dérivé d'un type abstrait. La classe exacte de l'objet n'est donc pas connue par l'appelant.

3.4.1 Implémentation dans le projet

Malgré le fait que le pattern Factory est supposé être utilisé pour la programmation orientée objet, nous avons pensé que ce pattern convenait très bien à ce que nous voulions faire.

Ce pattern a été utilisé pour FactoryRobot et FactoryProduit, ceci a permis de regrouper tout ce qui doit être fait lors de la création d'un robot ou produit.

De plus ce pattern permet une clarté du code et une meilleure essance dans la création des différents robots/produits, et ça permet aussi de modifier ou ajouter de nouveaux éléments dans les factory, c'est donc modulable.

FactoryRobot

```
1 #ifndef LO41__FactoryRobot
2 #define LO41__FactoryRobot
3
4 #include "Robots.h"
5
6 /* Pour le moment, je commente car j'ai une inclusion circulaire en mettant ça la
7 // C'est donc mis dans Robots.h
8 #define R1 1
9 #define R2 2
10 #define R3 3
11 #define R4 4
12 #define R5 5
13 #define R6 6
14 */
15
16 robot * creerRobot(int numRobot);
17
18 #endif
```

Listing 3.12 – Factory des robots

FactoryProduit

```
1 #ifndef LO41__FactoryProduit
2 #define LO41__FactoryProduit
3
4 #include "Produit.h"
5 #include "Composant.h"
6 #include "Operation.h"
7
8 #define Prod1 1
9 #define Prod2 2
10 #define Prod3 3
11 #define Prod4 4
12
13 produit* creerProduit(int numProduit);
14
15 // renvoie le numDuProduit si suffisamment de composant pour le creer / fonction special du
16 // au sujet et un produit n'a besoin que d'un seul type de composant / sinon renvoie 0
17 int hasEnoughComposantToDoProd(int nb, int compo);
18
19 #endif
```

Listing 3.13 – Factory des Produits

3.4.2 Conclusion du Pattern Factory et son utilité

3.5 Le producteur et les "lecteurs"

3.5.1 ProducteurComposant

```
#ifndef LO41__ProducteurComposant
2 #define LO41__ProducteurComposant

4 #include "Composant.h"
#include <pthread.h>
6 #include "anneauCircu.h"

8 int continuerProducteur;

10 // Le nb de composant au total qui seront envoye
int nbC1;
12 int nbC2;
int nbC3;
14 int nbC4;

16 void initProducteur();

18 // Renvoie 1 s'il reste des composants a envoyer
int resteTilComposant();
20 int recupererUnComposant();

22 void th_creerComposantEtAjoutDansAnneau();

24 #endif
```

Listing 3.14 – Représentation du producteur de composant

```
void th_creerComposantEtAjoutDansAnneau(){
2   while(resteTilComposant() == 1 && continuerProducteur == 1){
       pthread_mutex_lock(&mutex_anneau);
4       // On attend tjr que l'anneau est tournee
       pthread_cond_wait(&cond_anneauATourner, &mutex_anneau);
6       while(nbElementDansBuffer >= TAILLE_ANNEAU_MAX-6){
           printf("Producteur composant dormir while\n");
8           // on s'endort pour ne pas remplir totalement la chaine
           pthread_cond_wait(&cond_anneauATourner, &mutex_anneau);
10      }
       nbElementQuiAccedeAnneau++;
12      pthread_mutex_unlock(&mutex_anneau);
       if(isPostionBufLibre(POS_C_ENTREE) == 1){
14         int aa=recupererUnComposant();
           ajouterElement((void*)aa, POS_C_ENTREE, TYPE_COMP);
16         printf("composant ajouter %d\n",aa);
       }else
18         printf("composant non ajouter car case non libre\n");

20         pthread_mutex_lock(&mutex_anneau);
           nbElementQuiAccedeAnneau--;
22         pthread_cond_signal(&cond_anneauAttenteAcces);
           pthread_mutex_unlock(&mutex_anneau);
24     }
       printf("Thread Producteur Composant terminer\n");
26 }
```

Listing 3.15 – Fonction thread du producteur

Chapitre 4

Modélisation en Réseau de Pétri

Chapitre 5

Conclusion

Annexe A

Documentation

Voici des documents qui m'ont été utiles et fut aussi dans le passé (pour d'autres projets) pour m'amener à réaliser ce projet.

A.1 Factory Pattern

[Wikipedia Fabrique : Patron de conception](http://fr.wikipedia.org/wiki/Fabrique_(patron_de_conception)) ([http://fr.wikipedia.org/wiki/Fabrique_\(patron_de_conception\)](http://fr.wikipedia.org/wiki/Fabrique_(patron_de_conception)))
[Une explication plus poussée](http://design-patterns.fr/fabrique) (<http://design-patterns.fr/fabrique>)

A.2 Mes réalisations (github)

Voici une bonne partie de mes réalisations en programmation.
[Mes réalisations sur GitHub](https://github.com/Blackdread/) (<https://github.com/Blackdread/>)

Annexe B

Exemples de code