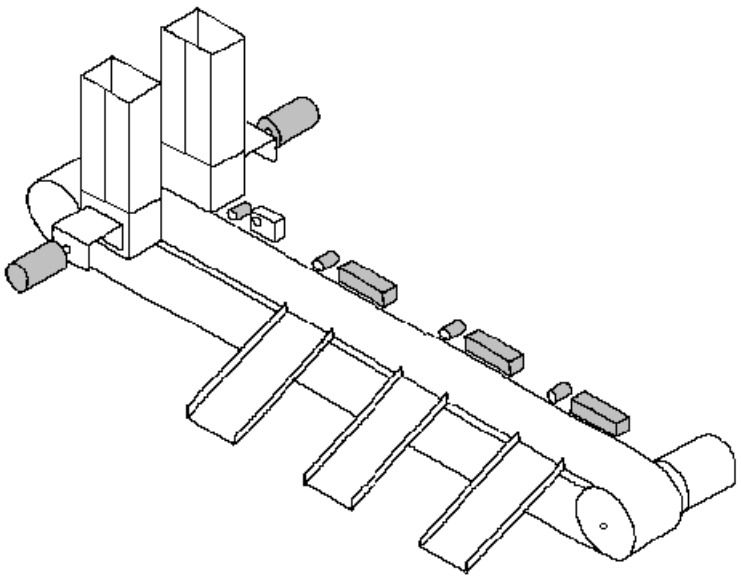
PILOTAGE D'UNE TRIEUSE DE PELLICULE
VERSION CAN TINI



*Etudiant1:*__Boufous Rachid__
*Etudiant2:*__Samur Nihat__
*Etudiant3:*__De Carvalho David__

Sommaire

II.Partie personnelle

Etudiant1:*Boufous Rachid* p.21

III.Partie personnelle

Etudiant2:*Samur Nihat* p.45

IV.Partie personnelle

Etudiant3:*De Carvalho David* p.81

1)Présentation du système

Un système de tri de boites de pellicules est utilisé dans une chaine de production de

matériel photographique.

L'entreprise les emballe en lot. Les lots sont constitués de trois pellicules au maximum afin de pouvoir les exposer à la vente.

A l'origine, les boites de pellicules sont empilées a l'intérieur des magasins **A** et**B***. Lorsque l'opérateur donne l'ordre de marche, ces boites sont transférées une par une sur le convoyeur ou elles sont d'abord identifiées puis dirigées vers l'une des trois goulottes **A**, **B** ou **C.** Celles qui ne conviennent pas restent sur le convoyeur et sont évacuées quand elles arrivent en fin de parcours.

***** Pour simuler une arrivée aléatoire de produits il est indispensable de choisir des pellicules de différentes sortes et de veiller à ce qu'elles ne soit pas classées

L'opérateur va également intervenir sur la machine grâce à un pupitre de commande composé d'un interrupteur CYCLE et de bouttons poussoirs: MARCHE, ACQUITEMENT et ARRET d'URGENCE.

=>Le boutton d'ARRET d'URGENCE va permettre l'arrêt immediat du système et ainsi immobiliser les actionneurs.

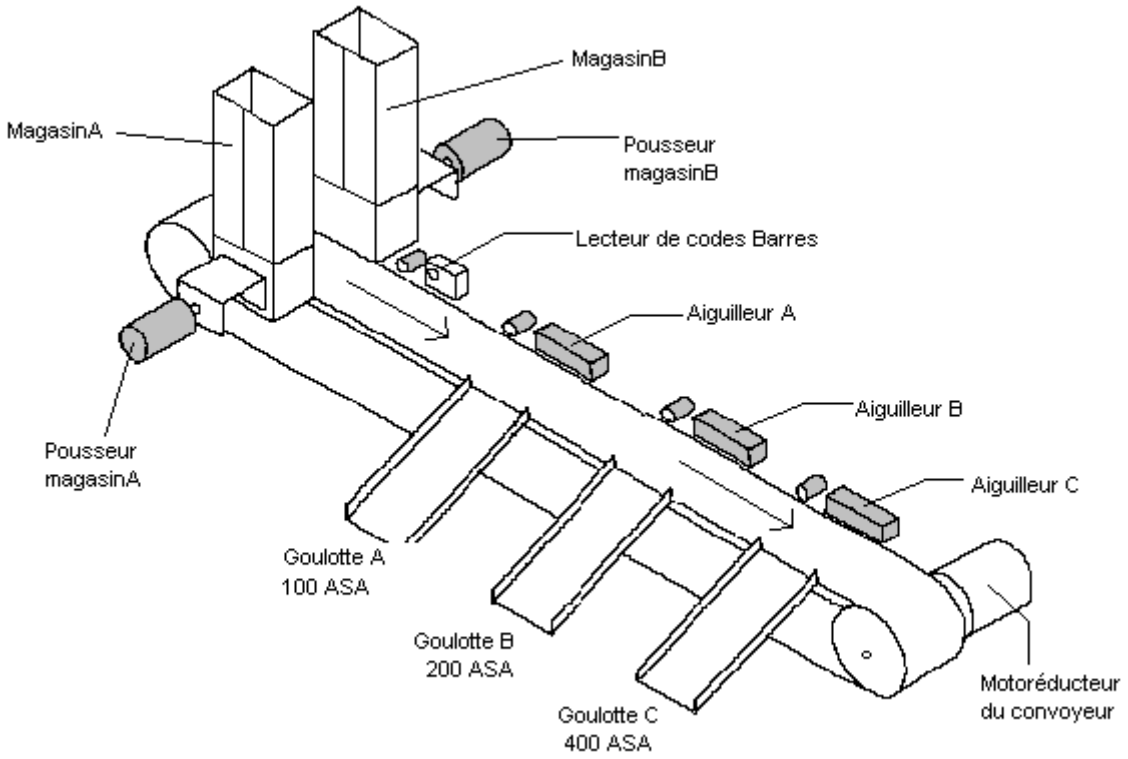
=>L'ACQUITEMENT va permettre de remmettre les actionneurs dans leur état initial suite à un arrêt d'urgence.

=>Le boutton MARCHE va, comme son nom l'indique, mettre en route le convoyeur.

=>Le boutton CYCLE continu va trier les pellicules jusqu'à destockage des magasins.

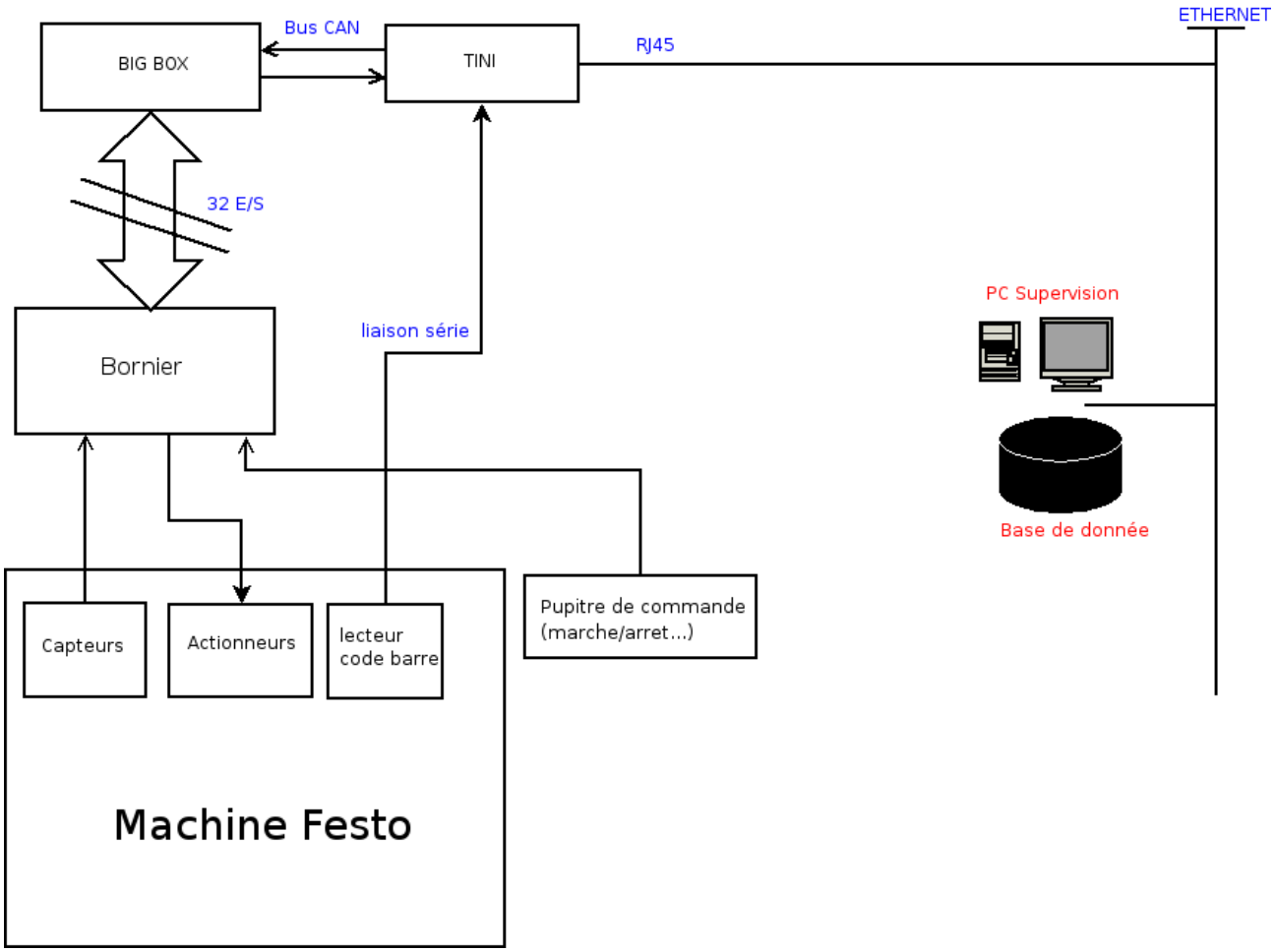
=>Le boutton CYCLE par cycle va trier pellicule par pellicule après chaque intervention de l'opérateur.

2)Principe de fonctionnement



- Le convoyeur est entrainé par le motoréducteur **M**. .
- Le déstockage des boites est réalisé au moyen de deux pousseurs. Ces derniers, ainsi que les volets des aiguillages, sont actionnés par des vérins pneumatiques.
- Les capteurs de présence pièce **PPA, PPB, PPC** et **PPM** sont des détecteurs de présence de proximité à commande photoélectrique.
- Les capteurs installés sur les verins sont des détecteurs de proximité à commande magnétique.
- Le systéme d'identification est constitué d'un lecteur de codes à barres et d'un décodeur.
- Toutes les informations, à l'exception de l'identification des pellicules, sont de type «tout ou rien».
- La commande est réalisée par automate programmable.
- La programmation de l'automate se fait à partir d'un terminal PC.

3)Schéma structurel



La machine festo est composée de capteurs, d'actionneurs et du lecteur code barre.

Les capteurs et les actionneurs sont reliés à un bornier qui permettent la liaison avec les 32 E/S de la Big Box.

Le pupitre de commande composé des différents boutons permettant le début du séquencement du système est également présent sur le bornier.

Le lecteur code barre est relié par l'intermédiaire d'une liaison série à un microcontroleur de type TINI.

La TINI est reliée à la Big Box par l'intermédiaire du bus CAN. Elle est également reliée à un réseau de type éthernet par un câble de type RJ45 permettant la communication avec un serveur Web.

4)Contraintes matérielles

- *TINI*

Il commande la partie opérative du système c'est à dire la Big Box.

Ce composant est de très petite taille et possède un environnement temps réel qui se programme en JAVA.

Il est possible de communiquer avec la TINI par différents moyens :

- TCP/IP
- CAN
- Liaison série

- *BIG BOX*

Ce boîtier Big Box équipé d'un contrôleur de bus CAN va permettre d'activer les 32 Entées/ Sorties reliées au pupitre de commande de la machine.

Ces Entrées/Sorties déportées seront reliées à la TINI à travers le bus CAN.

5)Contraintes logicielles

- Le choix du système d'exploitation: Windows XP pour sa simplicité d'utilisation, son interfaçage, sa compatibilité.

- Une machine de supervision permettant de donner:

    =>l'état du système par l'intermédiaire d'images sur une page HTML raffraichit

    =>les statistiques : nombre de pellicules triées, cadence journalière, temps d'utilisation de la machine.

    Cette machine pourra être consultable par un opérateur par l'intermédiare de pages HTML.

- Le serveur Web sera de type TOMCAT(spécifique JAVA). Il va permettre l'archivage des applications web qui y seront déployées. L'utilisation d'un serveur va permettre une visualisation du système à distance.

- L'utilisation des technologies liées à JAVA(application développée à l'aide de JSP) le tout contenu dans une page HTML

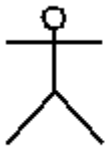
- L'utilisation de la technologie MySql pour crér une base de données contenant différentes statistiques qui seront afficher sur la page HTML.

6)Analyse UML

L'analyse UML permet de repésenter par l'intermédiaire de graphiques, la description et le fonctionnement d'un système.

a)Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation est une représentation UML permettant d'illustrer les différentes fonctionnalités du système ainsi que les différents acteurs qui y intéragissent.



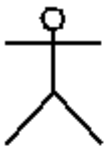
acteur

Un acteur permet l'intéraction avec les fonctionnalités du système

Un cas d'utilisation permet de représenter une fonctionnalité du système. Cette fonctionnalité doit etre un verbe à l'infinitif.

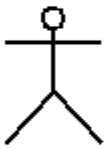
Cas d'utilisation général

Acteurs:



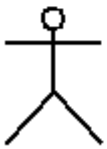
pellicules

les pellicules sont triées selon leur sensibilité



superviseur

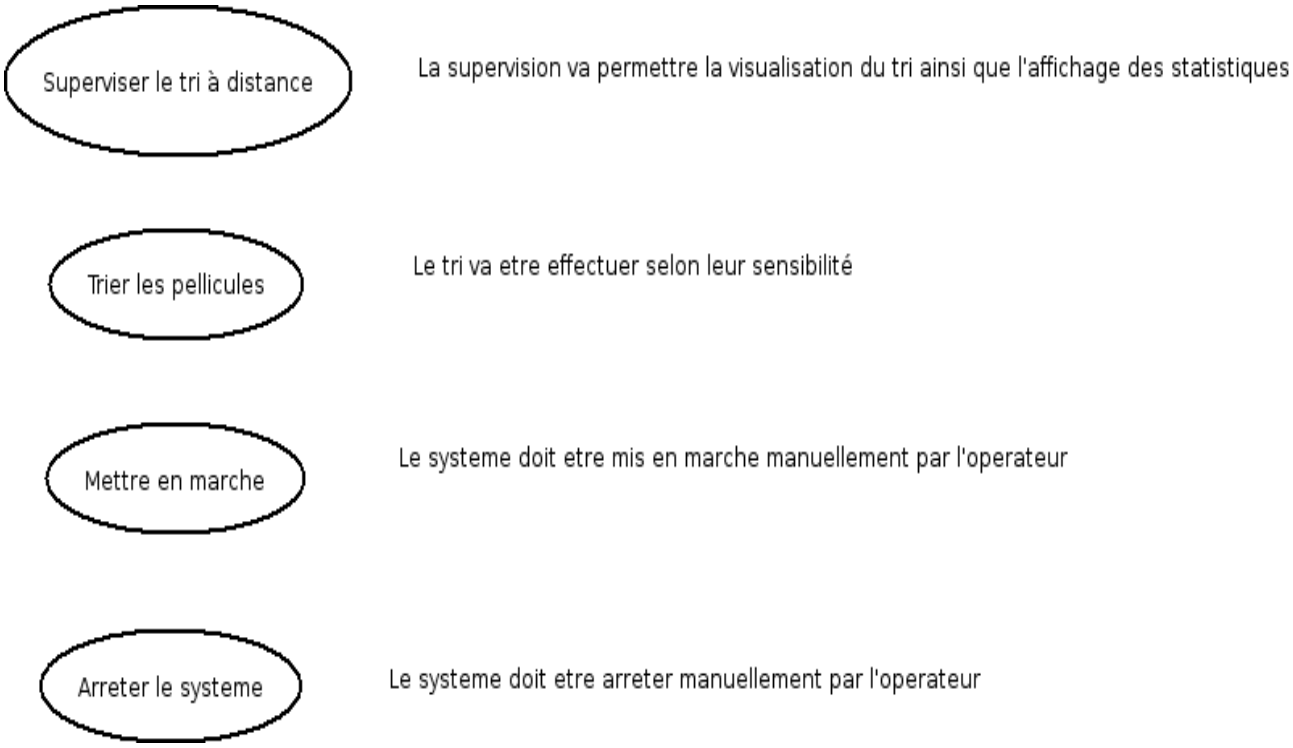
le superviseur va superviser le tri à distance grace à une connection web



opérateur

l'opérateur va permettre le tri des pellicules, la mise en marche ainsi que l'arret du système

Cas d'utilisation:



Représentation UML:

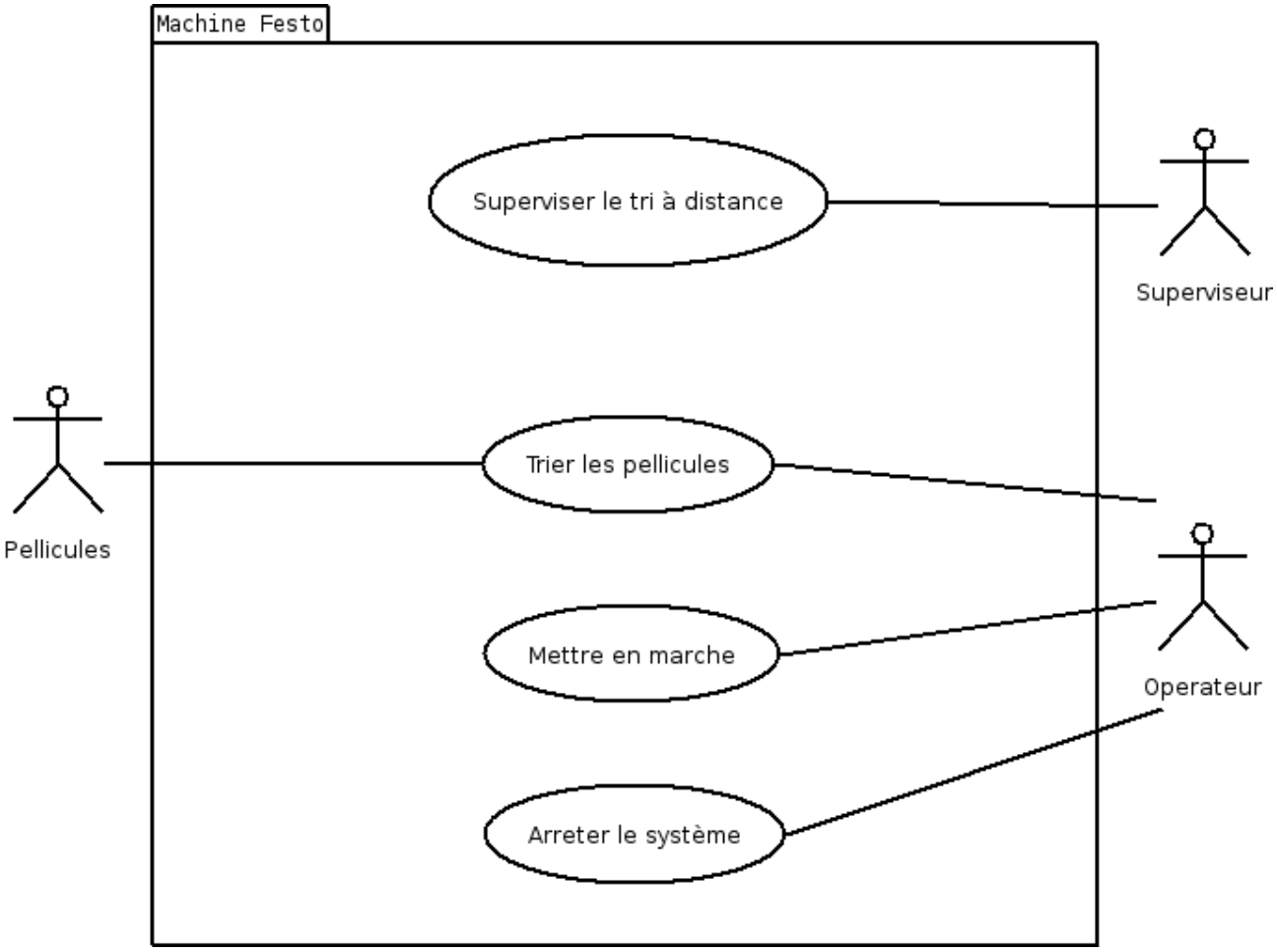
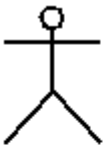


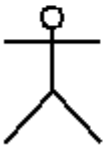
Diagramme de cas d'utilisation : trier les pellicules

Acteurs:



pellicules

les pellicules sont destockées des magasins
puis envoyer sur le tapis pour etre triées

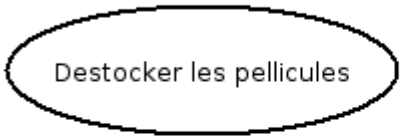


opérateur

l'opérateur va approvisionner les 2 magasins en pellicules pour le tri

Cas d'utilisation:

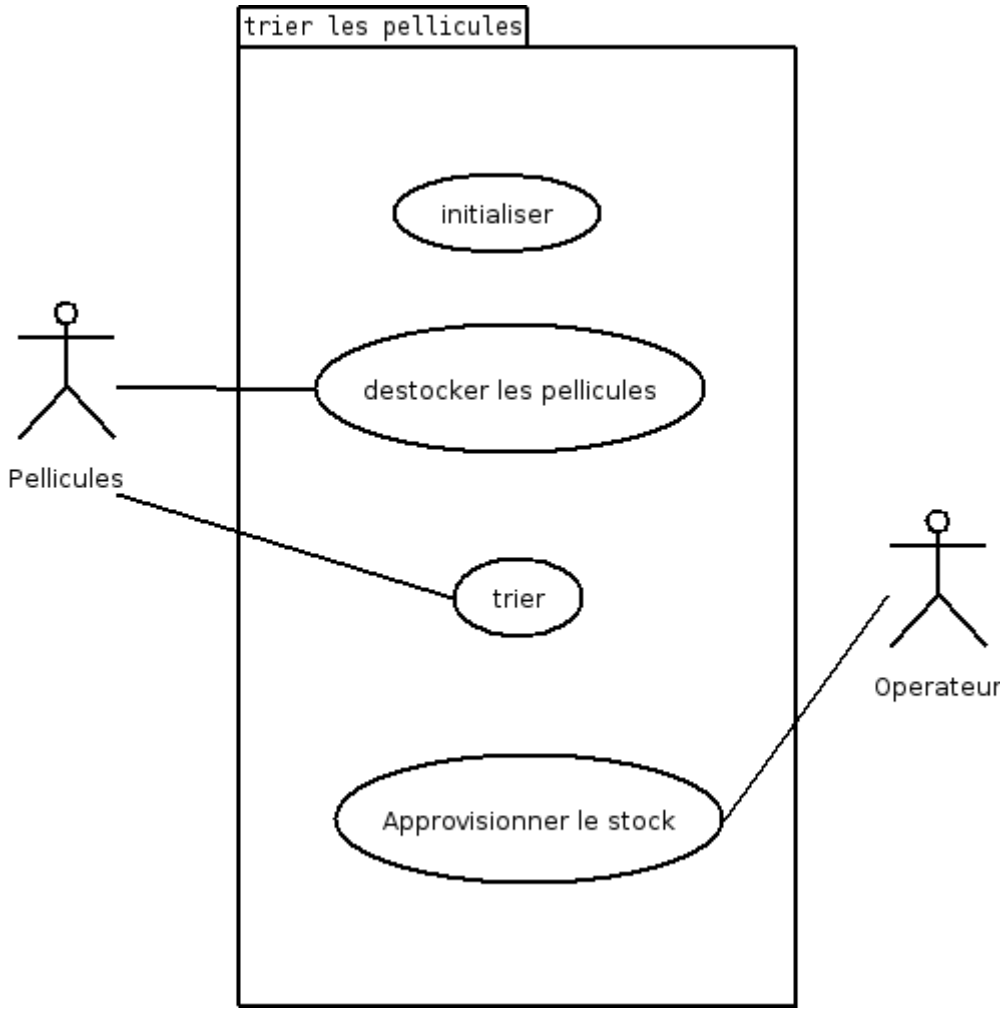
Le tri doit etre initialiser pour permettre le debut du cycle



Le destockage est effectuée par les pousseurs des 2 magasins
contenant les pellicules

Le tri doit etre effectuer selon une sensibilité

Les magasins doivent etre approvisionner
quand les magasins sont vides



Représentation UML:

b)Diagramme de contexte

Le diagramme de contexte est une schématisation faisant le lien entre le diagramme de cas d'utilisation et le diagramme de classe.

Ce diagramme est représenté de deux façons distinctes: simple et élaboré.

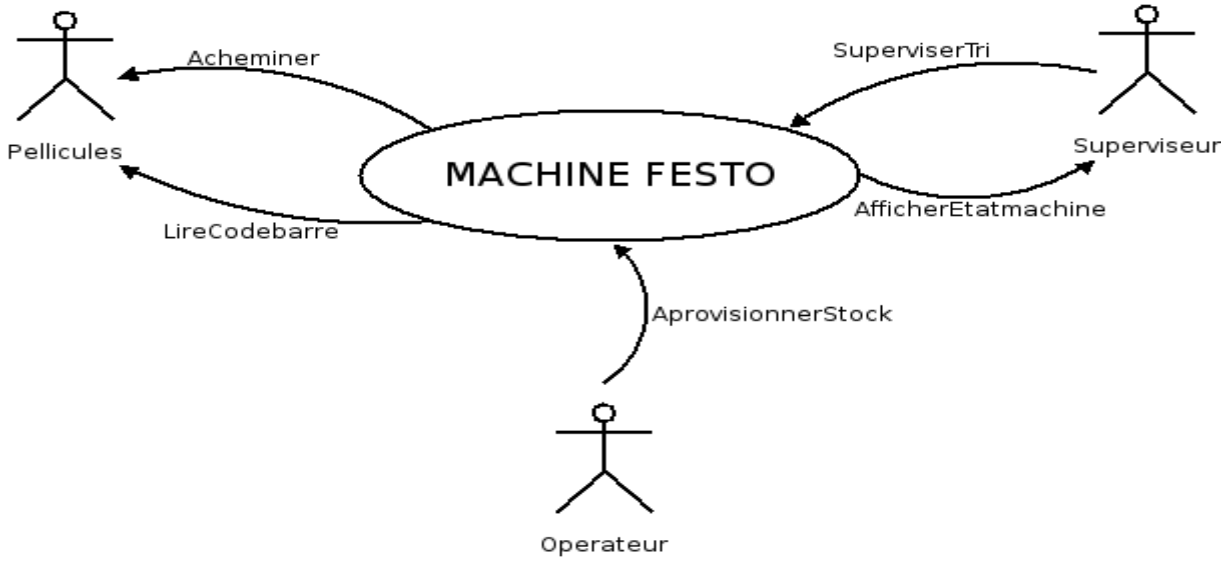
Diagramme de contexte simple:

Dans ce diagramme on y représente les acteurs intéragissant avec le système(Machine Festo) ainsi que les messages faisant la liaison entre le système et les acteurs.

Les pellicules vont etre acheminer par la machine festo. Le système va ensuite lire le code barre des pellicules permettant le tri.

L'opérateur va approvisionner les magasins en pellicules.

Le superviseur va superviser le tri du système et celui-çi va lui envoyer les informations sur l'état de la machine ainsi qu'une liste de statistiques.



Représentation graphique:

Diagramme de contexte : (élaboré)

Dans ce diagramme on peut retrouver tous les composants de notre système. On y illustre donc schématiquement le fonctionnement du système par les objets du système ainsi que les différents acteurs.

Un opérateur va fournir des pellicules à trier, dans les magasins, selon leur sensibilités. Il va également agir sur le pupitre de commande permettant la mise en marche et l'arrêt du système.

Les pellicules vont être détecter par différents capteurs puis convoyer vers les goulottes représentants la sensibilité de la pellicule.

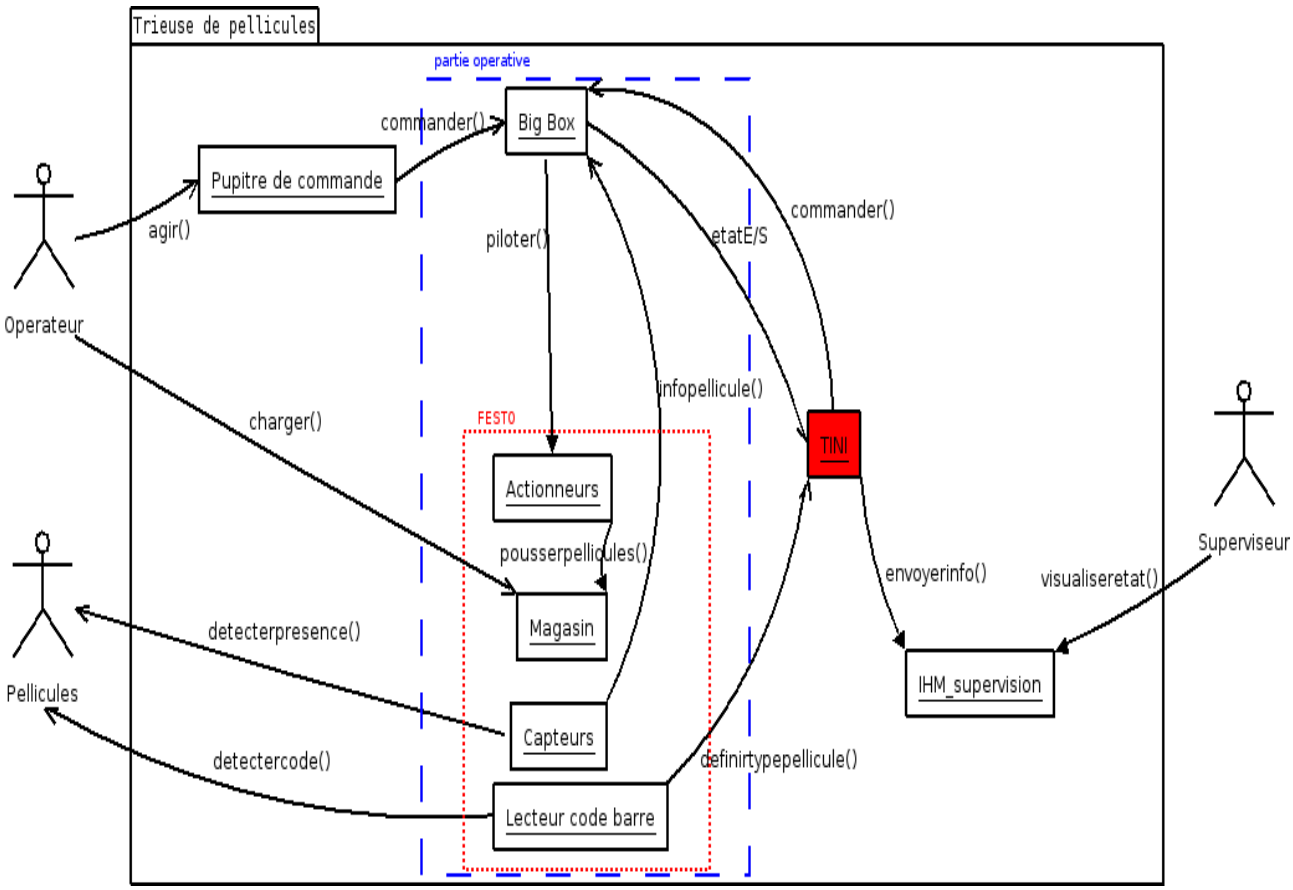
La machine festo est composée d'actionneurs, de capteurs, des magasins ainsi que le code barre. La machine fait partie de zone opérative du projet. La Big Box faisant l'intermédiaire entre la machine festo et le microcontrôleur TINI.

Le micocontôleur TINI est le composant principal du projet.

En effet, il va commander les E/S de la Big Box grâce au bus CAN, obtenir l'état des E/S de la Big Box pour ensuite faire la liaison avec la supervision.

La supervision est faîte pour visualiser l'état du fonctionnement de la machine à travers une Interface Homme-Machine(IHM). Cette IHM dynamique va donc informer son utilisateur sur le déroulement des processus en cours d'exécution à l'aide de statistiques.

Représentation graphique:



PARTIE ETUDIANT 1

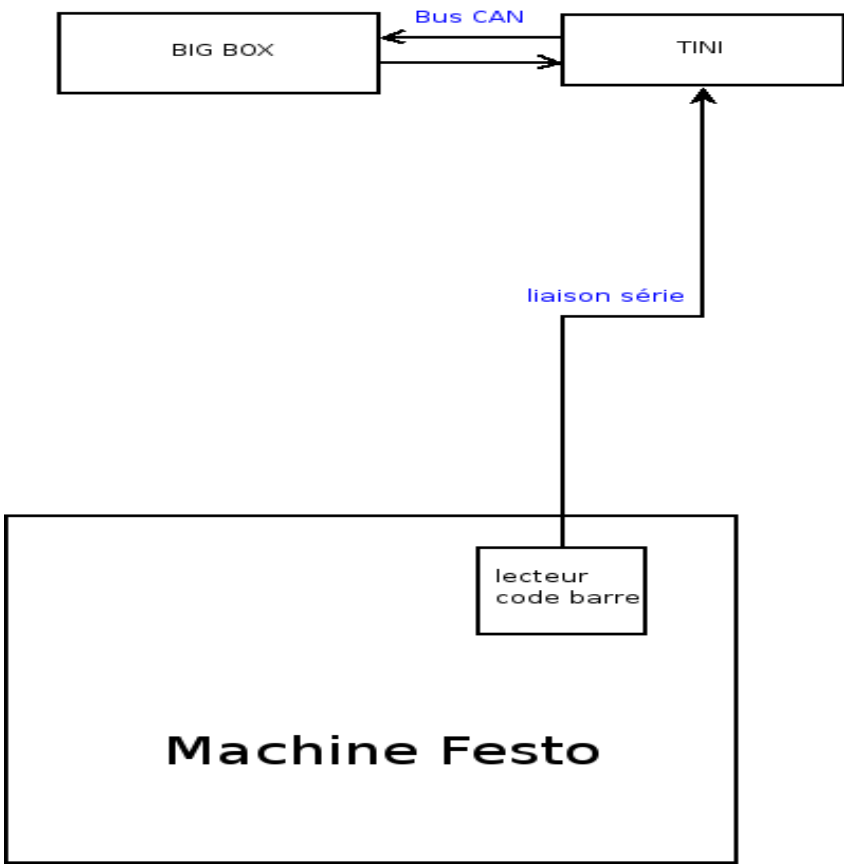
1)Presentation partie personnel

L'objectif de cette partie consiste a s'occuper du :

- module de communication bus CAN entre la TINI et la BIGBOX
- module de communication lecteur code barre

2)Analyse

a)Schéma structurel

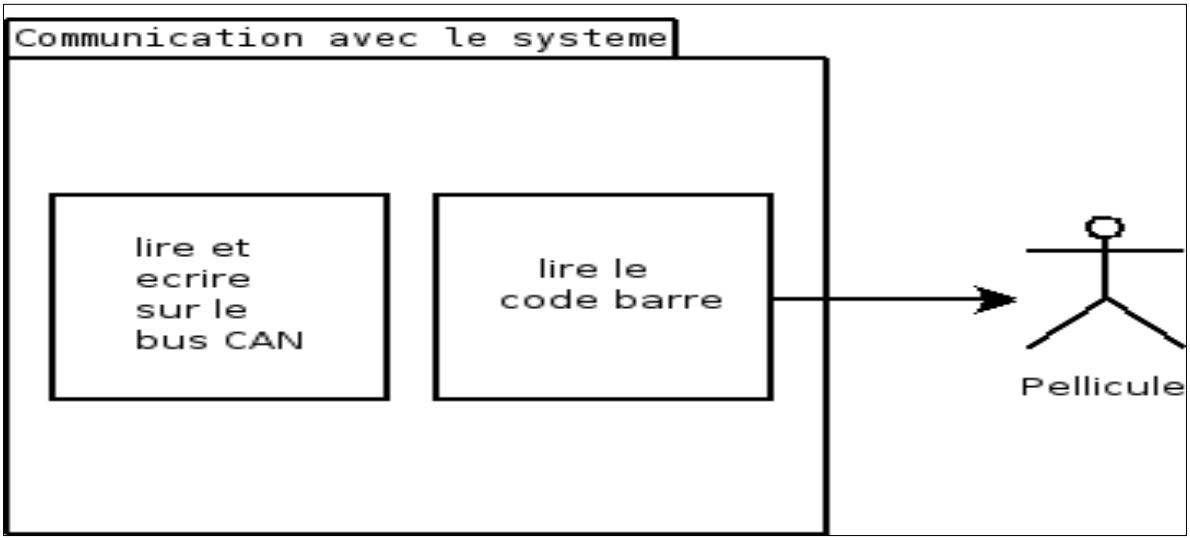


Description :

Le bus CAN permet de communiquer entre le micro-côntroleur et la BIGBOX
Le lecteur code barre et relié au micro-côntroleur via une liaison série

b)Use case personnel

Cette représentation UML permet de comprendre la fonctionnalité de ma partie personnelle

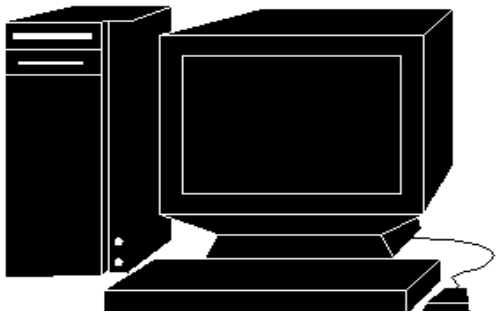


L'acteur pellicule contient un code barre qui doit être lu par le système.
Pour communiquer avec le système on utilise le bus CAN.

3)Contraintes matérielles et logicielles

Dans le but de pouvoir mener a bien le projet il nous faut respecter le cahier des charges. Il y a donc des contraintes a respecter tant au niveau matérielles que logicielles.

- Un PC sous Windows XP



- Une BIGBOX

- Le micro-côntroleur « TINI »

- Logiciel de developpement jEdit

4)Le Bus CAN

a)Description

Le CAN (Controller Area Network) est un bus de communication série qui supporte des systèmes temps réel avec un haut niveau de fiabilité.

b)Principe de fonctionnement

Le protocole est basé sur le principe de diffusion générale : lors de transmission, aucune station n'est adressée en particulier, mais le contenu de chaque message est explicité par une identification reçu de façon univoque par tous les abonnés. Grâce à cet identificateur, les stations, qui sont en permanence à l'écoute du réseau, reconnaissent et traitent les messages qui les concernent, elles ignorent simplement les autres.

L'identificateur indique aussi la priorité du message, qui détermine l'assignation du bus lorsque plusieurs stations émettrices sont en concurences. Dans notre cas, c'est un nombre de 11 bits, ce qui permet de définir jusqu'à 2048 messages plus ou moins prioritaires sur le réseau. Chaque message peut contenir jusqu'à 8 octets de données.

5)La BIGBOX

a)Presentation

voir annexe1 pour les E/S déportées

Le CAN bigbox de janz computer comporte 8 entrées et 8 sorties isolées ainsi qu' une socket MODULbus. Cette socket apporte à l'ensemble de la structure diverses possibilités d'application en fonction du module qui lui est raccordé. Par mis eux, nous pourrons trouver un module permettant de contrôler un moteur local, un module d'acquisition ou encore un module permettant d'obtenir 4 ports rs232 supplémentaire.

Dans notre cas nous utiliserons un module spécifique permettant d'obtenir 8 entrée supplémentaire, soit au total 16 entrées et 8 sorties. L'emploi de cette prothèse est justifiée par le nombre important de composant de la machine Festo : capteur de présence, capteur d'état etc.…

Spécifications

- Contrôleur MC68332
- 82C200 CAN device
- 128Ko Flash EPROM (jusqu'à 512Ko)
- 256Ko RAM (jusqu'à 1Mo)

- Débit de 1Mbit maximum
- Interface port série RS232
- Compatible CAN Spec 2.0
- Socket MODULbus
- 8 sorties 6-28V/1A ou TTL/50mA
- 8 entrées 5-28V

Grâce au contrôleur 16bit Mc68332, au 128Ko de flash rom et au 256Ko de RAM, le CAN bigbox est capable de gérer un large type d'application. L'emploi de Flash EPROM offre la possibilité à l'utilisateur de télécharger ses propres firmware via le port RS232. Les entrées et sorties sont dites isolées car elles sont optocouplée du contrôleur. La connection au bus can de la bigbox se fait via un connecteur 9 broches D-SUB.

b)Principe de fonctionnement

A l'allumage de la BigBox, le logiciel ICANOS lira par défaut les deux switch **SW1** et **SW2.** Le switch SW1 valide une base pour les identifiant CAN //pour distinguer le nœud CAN et le réseau CAN.

c)Règles de lecture et d'écriture

ECRITURE SUR LES SORTIES DIGITALES

Les sorties digitales du bus CAN-Bigbox sont activées par envoie d'un message CAN en utilisant (SW1*8)+221 de longueur 1.

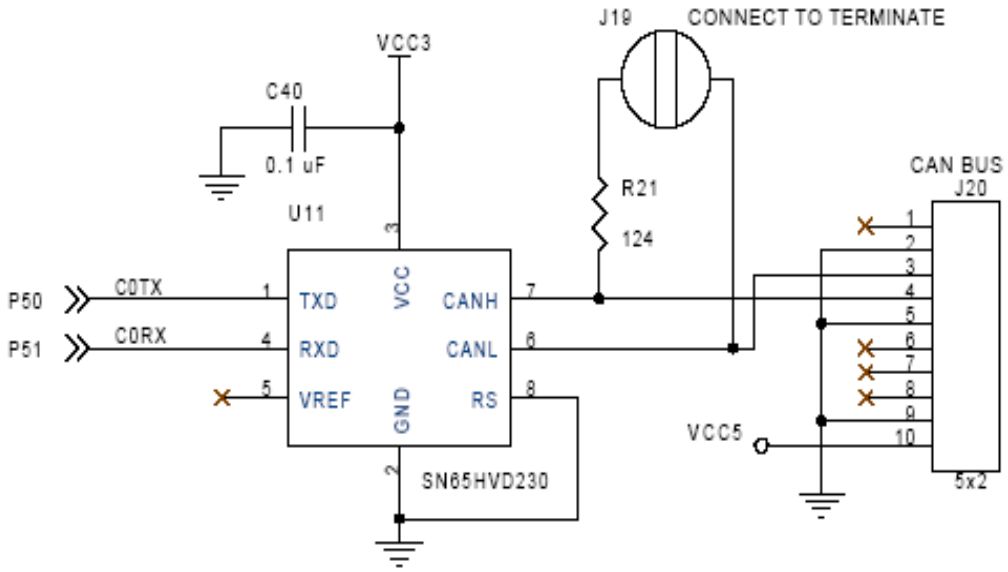
LECTURE SUR LES ENTREES DIGITALES

Adresse specifique pour les données en entrée

Pour lire une entrée digitale depuis le CAN-Bigbox, une requête de données utilisant **(SW*8)+222** doit être envoyée au CAN Bigbox.

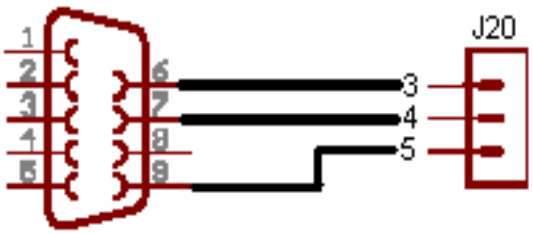
6)Cablage du bus CAN

a)Schéma



Ce schéma représente la liaison entre la connectique J20 et le CAN. En fonction de ce schéma nous pouvons donc fabriquer le câble qui nous permet de LIRE et ECRIRE sur le bus CAN.

Voici le bus CAN:



L'assignation des broches est la suivante:

BROCHES	DESIGNATION
6	CANL
7	CANH
8	GND

Le module de communication bus CAN (Controller Area Network) est un bus de communication série. C'est avantage sont les suivants:

-bus peu couteux

-un débit important jusqu'à 1 Mbits/s

-un très faible taux d'erreur

Par contre, il faut que les deux systemes est la meme vitesse.

b)Réglages de la vitesse

Le principal problème de la liaison CAN c'est le parametrage de la vitesse. La vitesse de transmission de l'emetteur doit être identique à la vitesse d'acquisition du récepteur. Ces vitesses sont exprimées en BAUDS (1 baud correspond à 1 bit/seconde).

Pour ce faire, nous devons calculer la vitesse de la façon suivante:

baud rate (bit/s) ==>vitesse

tqu ==> horloge interne du micro-controleur

freq ==> fréquence de l'horloge

baud rate = freq * 10^6 / BRP/ (1 + TSEG1 + TSEG2)

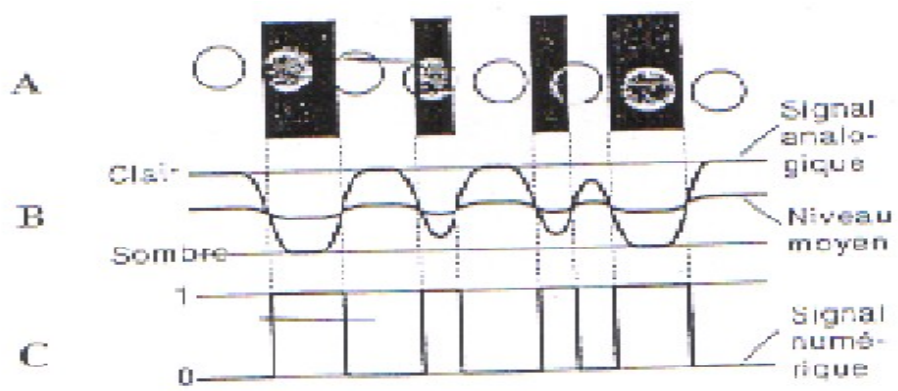
freq = 7*1/tqu

7)Le module lecteur code barre

• Principe de fonctionnement

Dans le système que nous allons mettre en œuvre, nous utiliserons des pellicules comportant un code barre de norme EAN13, celui de la grande distribution. Le principe de lecture de code barre repose sur la projection d'une source lumineuse sur le code. Les barres absorbent la lumière tandis que les espaces la réfléchissent. Ces variations lumineuses sont captées par un élément photo sensible puis amplifiées et traitées.

Le spot A en se déplaçant génère un signal électrique B exploitable :



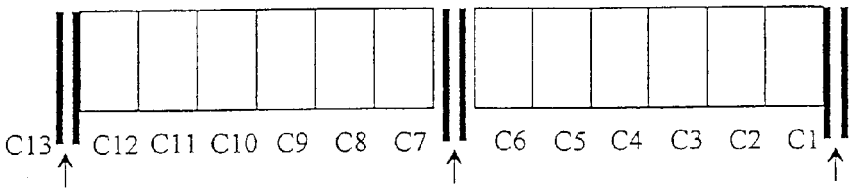
Le code EAN13 est un code :

- **Numérique** : chaque caractère est un chiffre de 0 à 9
- **Contenu** : les caractères sont représentés les uns à la suite des autres sans séparation
- **Bidirectionnel** : sa lecture peut être faite dans les deux sens
- **De format fixe** : il comprend 13 chiffres

Description du code EAN13

Code barre de norme EAN13

Le code est délimité par deux marques de terminaison. Entre celles-ci, le code est structuré en deux zones de même taille séparées par une marque centrale. Six chiffres sont représentés dans chaque zone, intitulés de C1 à C6 pour la zone de droite et de C7 à C12 pour la zone de gauche. Le chiffre C13 se déduit du codage des six chiffres précédents.



Marque de terminaison (101) — Marque centrale (01010) — Marque de terminaison (101)

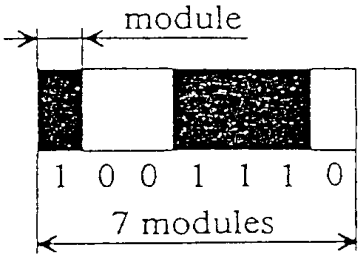
Les chiffres sont codés sur 7 bits selon l'un des trois codes A, B ou C représentés ci-dessous :

Chiffre	Code A	Code B	Code C
0	0001101	0100111	1110010
1	0011001	0110011	1100110
2	0010011	0011011	1101100
3	0111101	0100001	1000010
4	0100011	0011101	1011100
5	0110001	0111001	1001110
6	0101111	0000101	1010000
7	0111011	0010001	1000100
8	0110111	0001001	1001000
9	0001011	0010111	1110100

La largeur des barres ou des espaces est multiple d'une unité de largeur appelée module. A chaque module correspond un bit : les modules clairs sont des 0, les modules sombres sont des 1. A

chaque chiffre correspond donc un intervalle de 7 modules.

Le graphique ci-dessous représente le codage du chiffre 5 en code C.



Règles de codage

Les chiffres de C1 à C6 sont codés selon le code C
Les chiffres de C7à C12 sont codés selon le code A ou B. La configuration de ces codes détermine le treizième chiffre, conformément au tableau ci-dessous :

Signification du code

Le code à barres est la carte d'identité du produit.

- Les chiffres C12 et C13 identifient le pays d'origine (France : 30 à 37, Japon : 49, Grande Bretagne : 50, Allemagne : 40 à 43)
- Les chiffres C8 à C12 identifient l'entreprise dans le pays
- Les chiffres C2 à C7 identifie le produit dans l'entreprise
- Le premier chiffre C1 est une clé de contrôle calculée de la manière suivante :

$$C1 = 10 - [3(\sum_{i=1}^{6} C_{2i}) + \sum_{j=1}^{6} C_{2j+1}]_{\text{modulo}10}$$

(modulo 10 est le reste de la division par 10 : [75]modulo10 = 5)

Le lecteur code barre que nous utiliserons dans le système sera connecté directement au port RS232 du micro-côntroleur (TINI). Cependant, le lecteur est un organe autonome. C'est dire que nous n'aurons pas à nous soucier de la logique interne du lecteur. Une simple lecture sur le port RS232, via le PC, nous permettra de récupérer les trois derniers caractères du code barre.

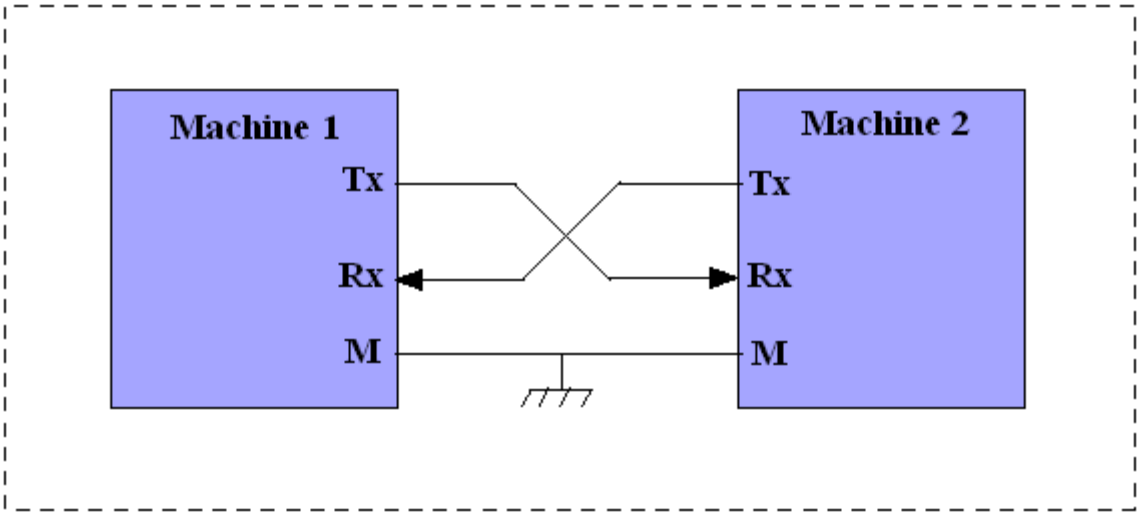
Remarque :

Le lecteur code barre retourne en réalité 4 caractères. Les trois premiers sont les trois derniers chiffres du code barre et le dernier est un caractère nul ('\0') qui marque la fin de lecture.

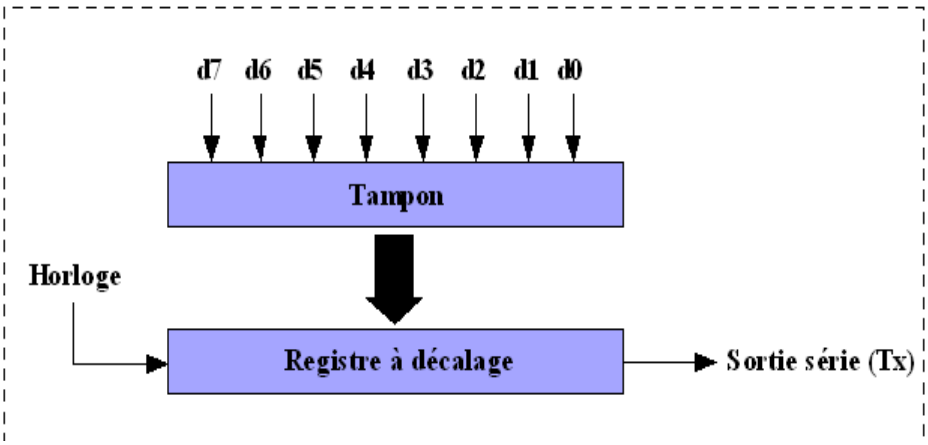
8)Liaison série

Une liaison série est une ligne où les bits d'information (1 ou 0) arrivent successivement, soit à intervalles réguliers (transmission synchrone), soit en groupe et à des intervalles aléatoires, (transmission asynchrone). La liaison RS232 est une liaison série asynchrone.

Schéma d'une liaison RS232 entre le lecteur code barre et le micro-côntroleur :



L'octet à transmettre est envoyé bit par bit (poids faible en premier) par l'émetteur sur la ligne Tx, en direction du récepteur (ligne Rx) qui le reconstitue. L'opération de sérialisation est réalisée de la façon suivante :



La vitesse de transmission de l'émetteur doit être identique à la vitesse d'acquisition du récepteur. Ces vitesses sont exprimées en BAUDS (1 baud correspond à 1 bit / seconde, dans notre cas). Il existe différentes vitesses normalisées: 9600, 4800, 2400, 1200... bauds.

La communication peut se faire dans les deux sens (duplex), soit émission d'abord, puis réception ensuite (half-duplex), soit émission et réception simultanées (full-duplex).

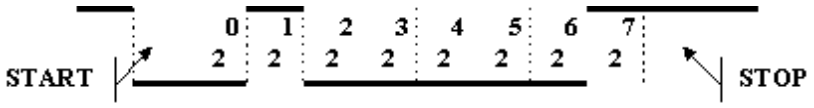
La transmission étant du type asynchrone (pas d'horloge commune entre l'émetteur et le récepteur), des bits supplémentaires sont indispensables au fonctionnement: bit de début de mot (start), bit(s) de fin de mot (stop).

D'autre part, l'utilisation éventuelle d'un bit de parité, permet la détection d'erreurs dans la transmission.

Exemple :

Transmission du code $82 avec 1 bit de stop, sans bit de parité.

$82 ⇒ % 1000 0010

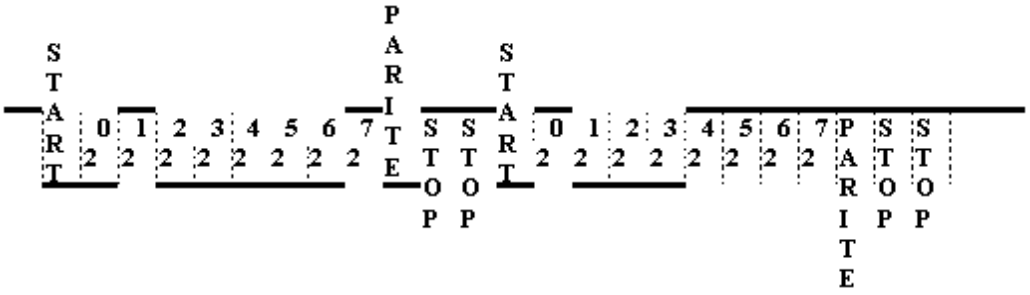


Parité :

La parité est une technique qui permet de vérifier que le contenu d'un mot n'a pas été changé accidentellement lors de sa transmission. L'émetteur compte le nombre de " 1 " dans le mot et met le bit de parité à " 1 " si le nombre trouvé est impair, ce qui rend le total pair : c'est la parité paire. On peut aussi utiliser la parité impaire.

Exemple:

Transmission de $82, puis $F1, avec parité paire et 2 bits de " stop ".



Format des données :

Le format de la donnée peut aller de 5 à 8 bits (très souvent 7 bits : caractère ASCII).

Intérêt de la communication série :

Un des principaux intérêts de la liaison série est l'utilisation d'un nombre réduit de fils. En effet, la communication la plus simple peut être établie sur seulement 3 fils : Tx, Rx et la Masse.

9)Application

a)Installation logicielles

a)jEdit

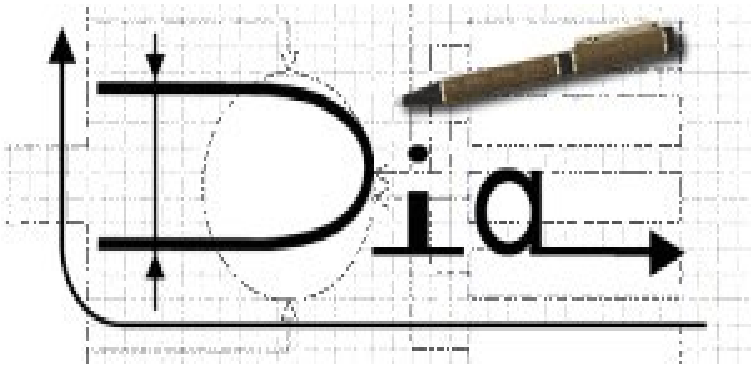
Ce logiciel est un éditeur de texte en java. Pour l'installer, il faut télécharger le logiciel sur le site suivant : http://www.jedit.org/

L'installation s'effectue en plusieurs étapes:

- *téléchargement de jEdit*
- *installation du logiciel*
- *ajouter au système la variable JAVA-HOME dans Propriétés->Avancé->Variables environnement*
- *la valeur du chemin d'accès JDK est:*
  ==>C:\j2sdk1.4.2_06

Désormais l'éditeur de texte est installé et prêt à l'usage..

b)Dia



Dia est un outil permettant de créer des diagrammes. Pour l'installer, il faut télécharger le logiciel sur le site suivant : http://www.dia.org/

*b)*Codage du module de communication bus CAN avec la BIGBOX

. *Bus CAN*

On utilise le bus CAN pour la communication entre le micro-contrôleur et la BIGBOX. Pour ce faire il faut régler la vitesse, et en fonction de la vitesse trouvé on se reporte au tableau suivant:

Baud Rate	BRP	TSEG1	TSEG2
10 Kbps	88	13	7
20 Kbps	71	5	7
20 Kbps	44	13	7
50 Kbps	41	3	5
50 Kbps	23	10	5
125 Kbps	7	9	7

Application:

baud rate = freq * 10^6 / BRP/ (1 + TSEG1 + TSEG2)

freq = 7*1/tqu

tqu = 7*1/14.745

tqu = 475ns

baud rate = 14.745*10^6/7/(1+9+7)

baud rate = 125Kbps

En obtenant, la vitesse on regle la vitesse sur la BIGBOX en modifiant le switch SW2 en se conformant au tableau suivant :

SW2 Position	CAN baud rate
0	1 Mbit/sec
1	500 Kbit/sec
2	250 Kbit/sec
3	125 Kbit/sec
4-F	1 Mbit/sec

Donc la on a reglé la vitesse sur la BIGBOX maintenant il nous faut le reglé sur le micro-côntroleur a l'aide de ce programme

voir annexe 2 pour l'explication du code

```
//Reglage de la vitesse
        public LiaisonCan(in div, int tseg1, int tseg2, int sjw, int id)

                /* 125Kbit/s with crystal of 14.745MHz */
                int CAN_DIVISOR = div;
                int CAN_TSEG1 = tseg1;
                int CAN_TSEG2 = tseg2;
                int CAN_SJW = sjw;
               this.ID = id;
      try
        {
                BusCan = new CanBus(CanBusNum);
                BusCan.setBaudRatePrescaler(CAN_DIVISOR);
                BusCan.setTSEG1(CAN_TSEG1);
                BusCan.setTSEG2(CAN_TSEG2);
                BusCan.setSynchronizationJumpWidth(CAN_SJW);

       // On valide le controleur
                BusCan.enableController();
      catch (Throwable e)
      {
       System.out.println("Erreur! " +e);
```

```
            System.exit(-1);
        }
    }
```

Après avoir reglé la vitesse pour la communication il faut maintenant le testé en écrivant sur le bus. Voici le programme d'écriture

```
import com.dalsemi.comm.*;
import com.dalsemi.system.*;


public class ecrire
{
/* 125Kbit/s with crystal of 14.745MHz */
static final int CAN_DIVISOR = 7;
static final int CAN_TSEG1 = 9;
static final int CAN_TSEG2 = 7;
static final int CAN_SJW = 1;
static final byte CANBUSNUM = CanBus.CANBUS0;


static void doTest() throws Exception
{
CanBus a = new CanBus(CANBUSNUM);
a.setBaudRatePrescaler(CAN_DIVISOR);
a.setTSEG1(CAN_TSEG1);
a.setTSEG2(CAN_TSEG2);
a.setSynchronizationJumpWidth(CAN_SJW);

// Now, we tell the CAN Controller to jump on the bus.
a.enableController();


a.setMessageCenterTXMode(1);
```

```
byte[] data = new byte[8];


data[0] = (byte)0xAA;

// Send a frame using standard (11 bit) ID, block until frame is ACKed
a.sendDataFrame(0xdd, false, temp);

static void main(String args[])
{
try
 {
if (TINIOS.isCurrentTaskInit())
Debug.setDefaultStreams();

System.out.println("CAN Transmit tester");

doTest();

System.out.println("Normal Exit");
}
catch (Throwable e)
{
System.out.println("Exception");
System.out.println(e);
}
}
```

Nous allons donc tester ce bout de programme en l'envoyant au micro-côntroleur. Mais avant il faut le compiler et le convertir en « **.**TINI ». Pour ce faire il faut créer un fichier **.BAT**
Le fichier « ecrire.bat » à pour intérêt de nous éviter d'avoir à retaper tout le chemin nécessaire à

l'exécution d'une requête de compilation.

Fichier ecrire.bat

cls

javac -bootclasspath c:\tini1.15\bin\tiniclasses.jar;c:\tini1.15\bin\modules.jar C:\festo_programme\modules\ecrire.java

c:\tini1.15\bin\ :représente le fichier dans lequel se trouvent toutes les classes utilisées dans le programme ecrire.java

Après la compilation nous devons convertir le programme *ecrire.class* obtenu lors de la compilation en un programme appelé *ecrire.tini* car le microcontrôleur TINI utilise le format .TINI
Donc dans le meme fichier on écrie cette ligne de commande

java -classpath c:\tini1.15\bin\tini.jar;%classpath% BuildDependency -f C:\festo_programme\modules\ecrire.class -o C:\festo_programme\modules\ecrire.tini -d c:\tini1.15\bin\tini.db -add CANAll -x c:\tini1.15\bin\owapi_dep.txt -p c:\tini1.15\bin\modules.jar

C:\festo_programme\modules\ecrire.class : Fichier d'entré avant la conversion
C:\festo_programme\modules\ecrire.tini : Fichier qu'on obtient après la conversion

REMARQUE : les classes utilisées sont :

modules.jar
tini.jar
tiniclasses.jar.

Transfert du fichier cantransmit.tini dans le microcontrôleur TINI

Pour se connecter au micro-côntroleur :

- on ouvre un inviter de commande
- on tape

T:\>ftp 192.168.109.235
Connecté à 192.168.109.235.
220 Welcome to slush. (Version 1.15) Ready for user login.
Utilisateur (192.168.109.235:(none)) : root
331 Password Required for root
Mot de passe :tini
230 User root logged in.

Transférer le fichier ecrire.tini :

On doit spécifier qu'on veut faire un transfert de fichier de type *Binaire*

- ftp> bin
  200 Type set to Binary
  ftp> put C:\festo_programme\modules

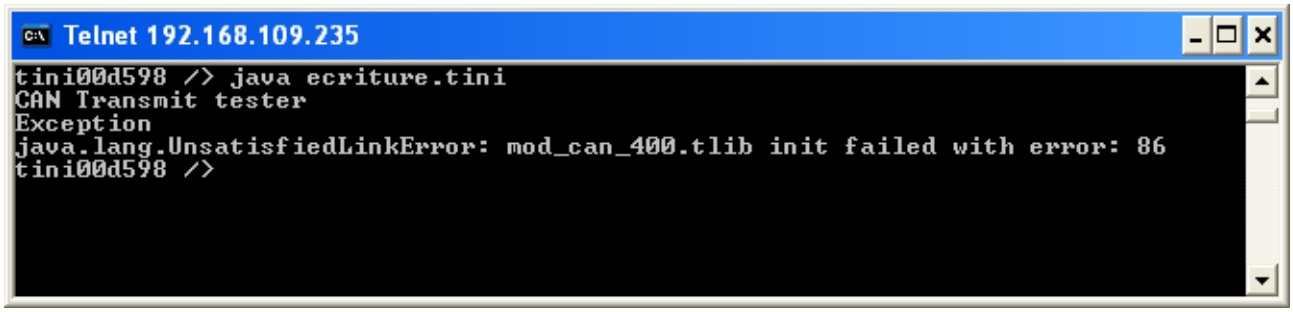
Execution du fichier

Pour executer le fichier on utilise telnet

T:\>telnet 192.168.109.235
Welcome to slush. (Version 1.15)
tini00d598 login: root
tini00d598 password:tini
tini00d598 /> java ecrire.tini

java ecrire.tini : la commande java permet d'executer le fichier *ecrire.tini*

Après l'execution on obtient :



Mais on a une exception qui est dû a un problème avec le bus CAN. Pour résoudre ce problème il suffit soit:

-de brancher la BIGBOX à l'alimentation

-vérifier que le bus CAN soit bien brancher

-verifier la configuration du bus CAN

Après avoir écrie sur le bus CAN il faut lire les états des entrées sorties.

Voir annexe 3 pour le code de lecture

c)Codage du module lecteur code barre

Voir annexe 4 pour le code de lecture de code barre

PARTIE ETUDIANT 2 : PILOTAGE DU TRI

1. Introduction

L'objectif de cette partie est d'assurer le pilotage de la partie opérative pour effectuer le tri des pellicules. Pour cela nous disposons de grafcets. Il faut donc automatiser le tri selon les grafcets donnés.
Cela revient à ***transcrire les grafcets en code JAVA***.

Parallèlement à cela on doit aussi envoyer l'état du tri à un poste de supervision (serveur web).

Contraintes matériels :
Carte micro-contrôleur TINI de chez Dallas Semiconductor :

- environnement temps-réel
- se programme en JAVA
- communications avec l'extérieur par TCP/IP, CAN, liaison série

2. Plate-forme de développement

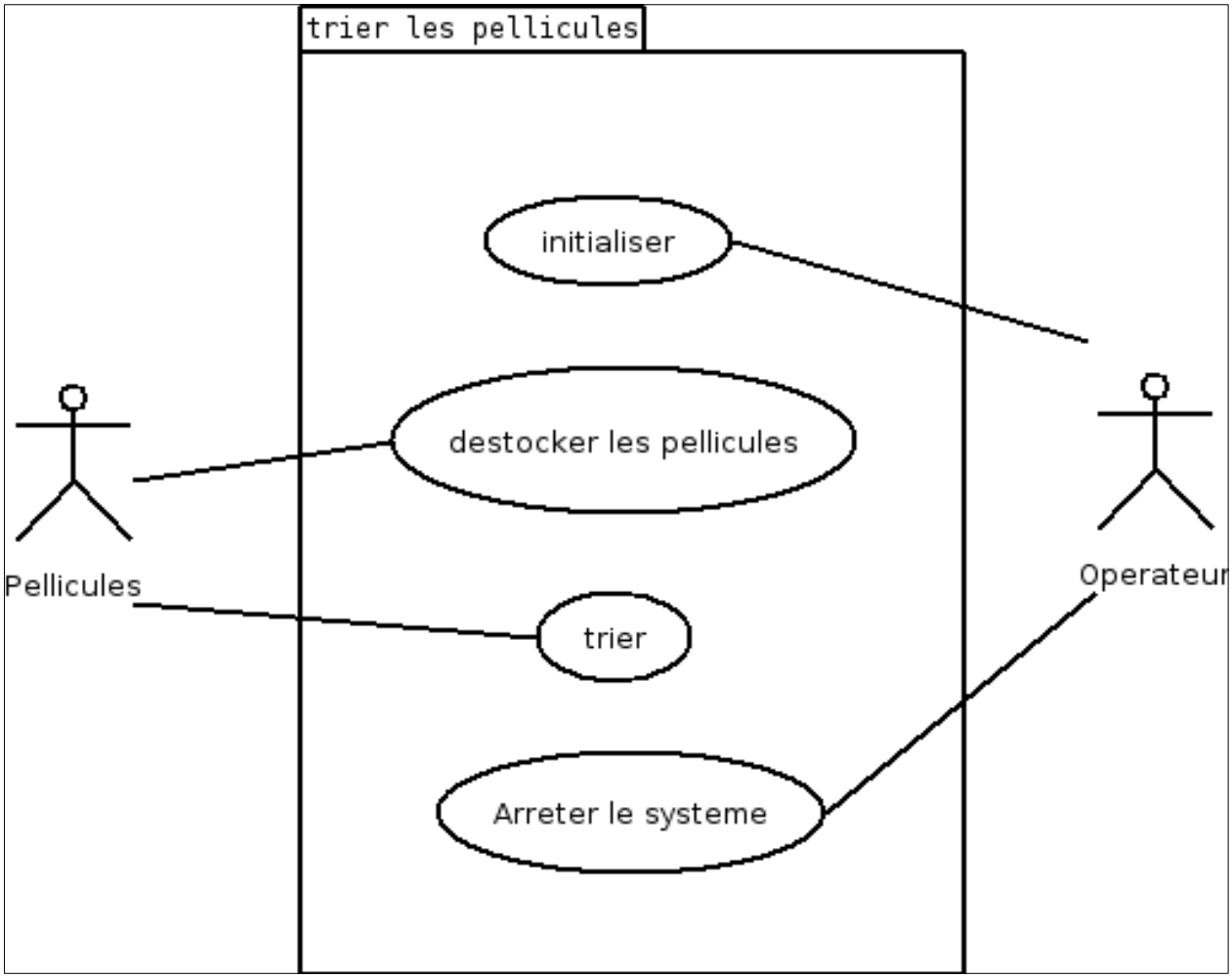
- PC sous Windows XP professionnel sur lequel était installé :

  - un environnement de développement JAVA : **JDK**1.5.0 qui contient :
    - les API JAVA : ce sont les classes qui constituent la base du langage JAVA
    - un compilateur
    - la machine virtuelle JAVA

  - un éditeur de texte : **JEDIT** sur lequel on peut rajouter une console.

3. Analyse UML

a). Diagramme du cas d'utilisation trier les pellicules



b) Description textuelle CU « Trier les pellicules »

Titre: Trier les pellicules

Acteurs: Opérateur, Pellicules

Résumé: Les pellicules sortant des magasins sont dirigées vers les différentes goulottes selon leur codes barre.

Scénario nominal

1. L'opérateur approvisionne le magasin en pellicules.
2. La pellicule est dirigée vers le convoyeur.
3. La pellicule se présente devant le capteur du code barre.
4. La pellicule passe devant le code barre.
5. L'information relative au code barre est envoyée à la Tini pour déterminer son type.
6. La pellicule est présentée devant le capteur de présence.
7. La pellicule est captée.
8. L'aiguilleur va diriger la pellicule vers la goulotte.
9. La pellicule est triée selon son type.

Enchaînements d'erreurs

E1. plus de pellicules

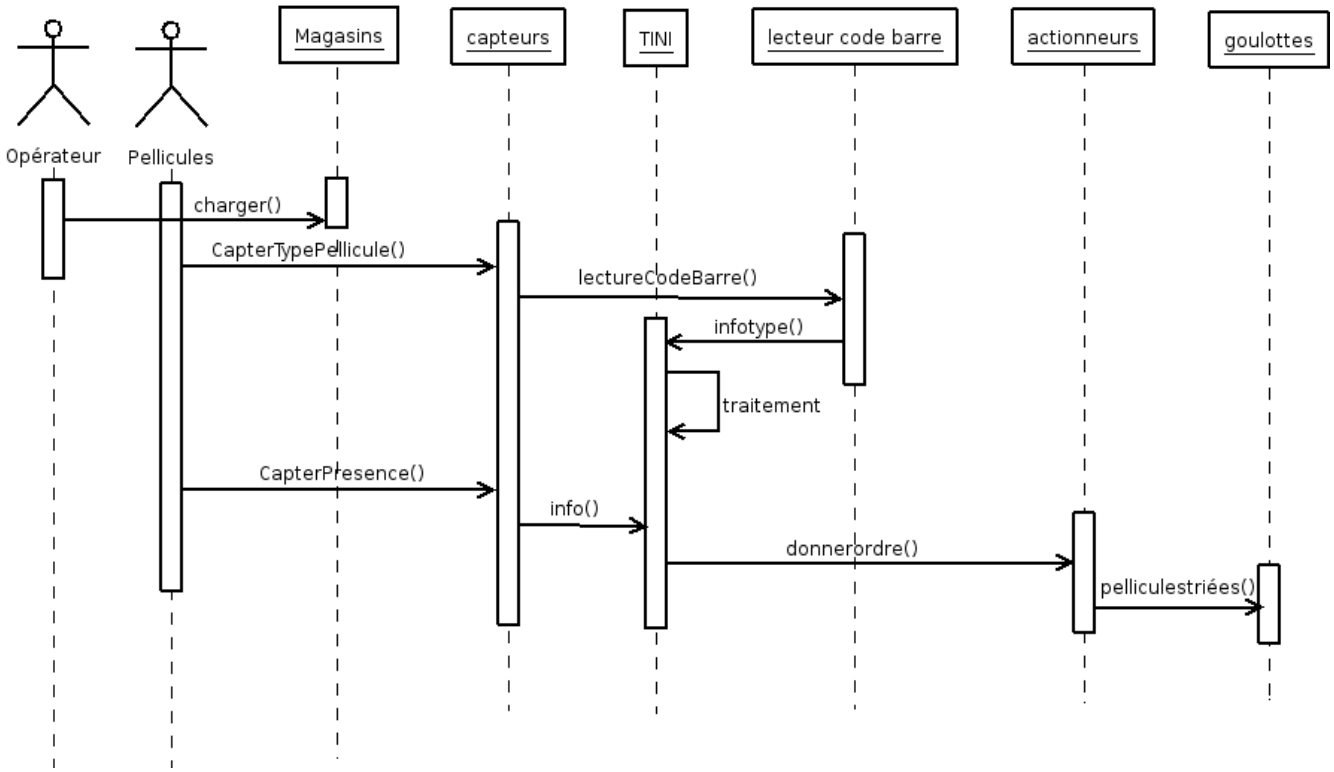
1. Il n'y a plus de pellicules pour approvisionner les magasins.

Le CU est terminé

E2. mauvais code barre.

6. L'information relative au code barre ne détermine aucun type de pellicules.
7. La pellicule est acheminée jusqu'au panier sans être trier.

c) Diagramme de séquence du CU « Trier les pellicules »



4. Grafcets

L'automatisation du tri doit se faire en suivant 3 grafcets :

Init :

- initialisation du système, c'est-à-dire la mise au repos des actionneurs ainsi que du convoyeur.
- lancement du tri (grafcets « Product » et « Destock ») jusqu'à l'activation du bouton d'arrêt d'urgence qui va immobiliser le système jusqu'à activation du bouton acquittement qui va réinitialiser le système.

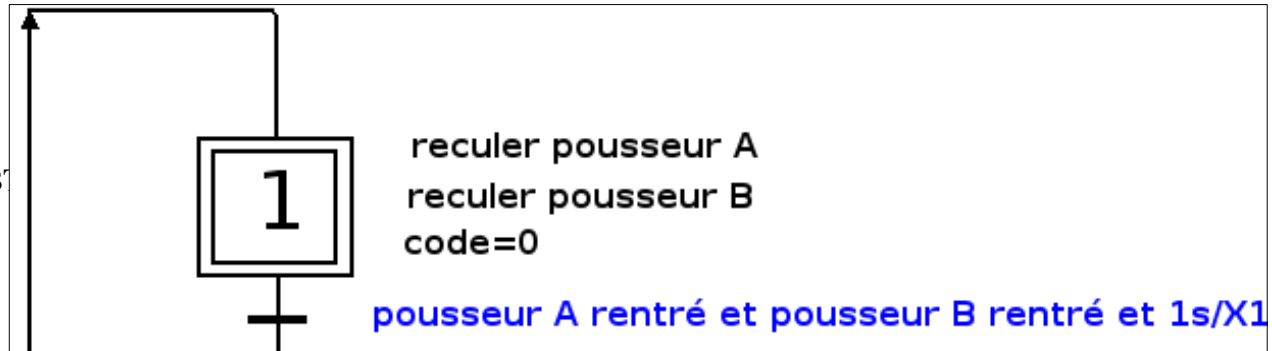
Product :

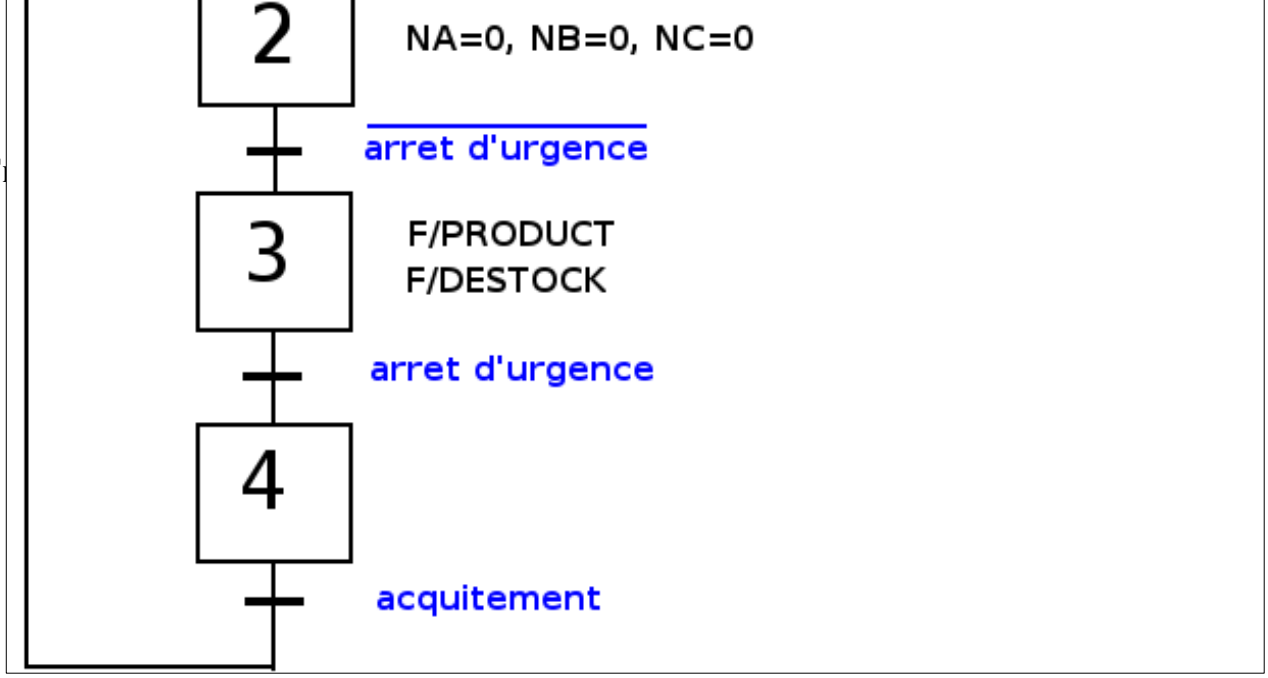
- mise en marche du convoyeur
- lancement du déstockage d'une pellicule (grafcet « Destock »).
- détermination du type de pellicule et acheminement de celle-ci vers la goulotte appropriée.

Destock :

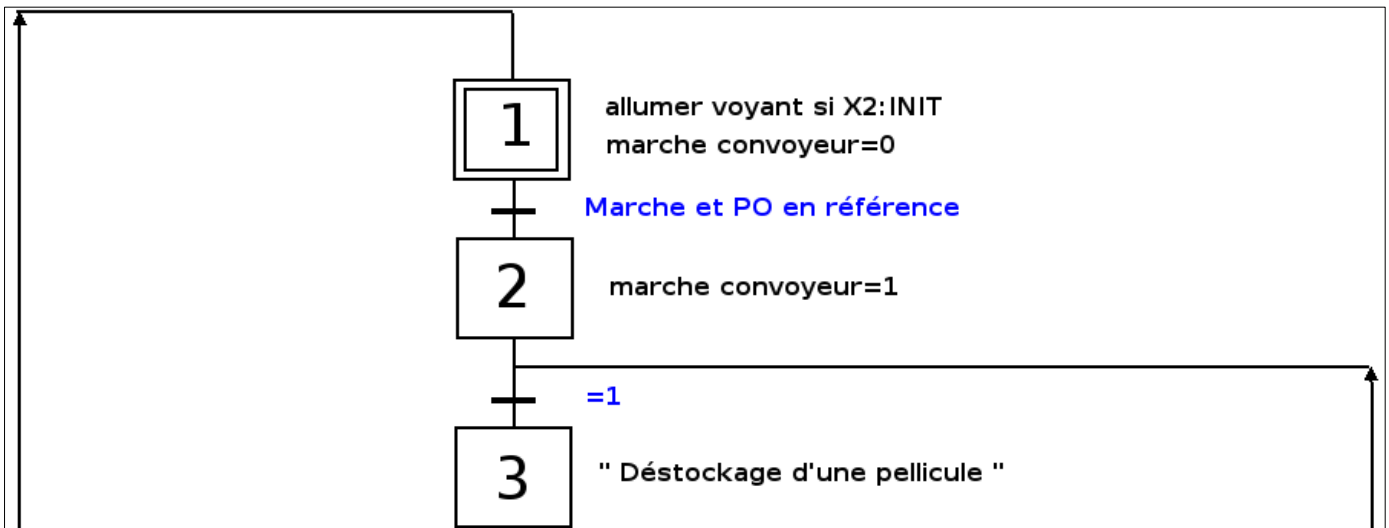
- déstockage des pellicules du magasin A puis celles du magasin B.

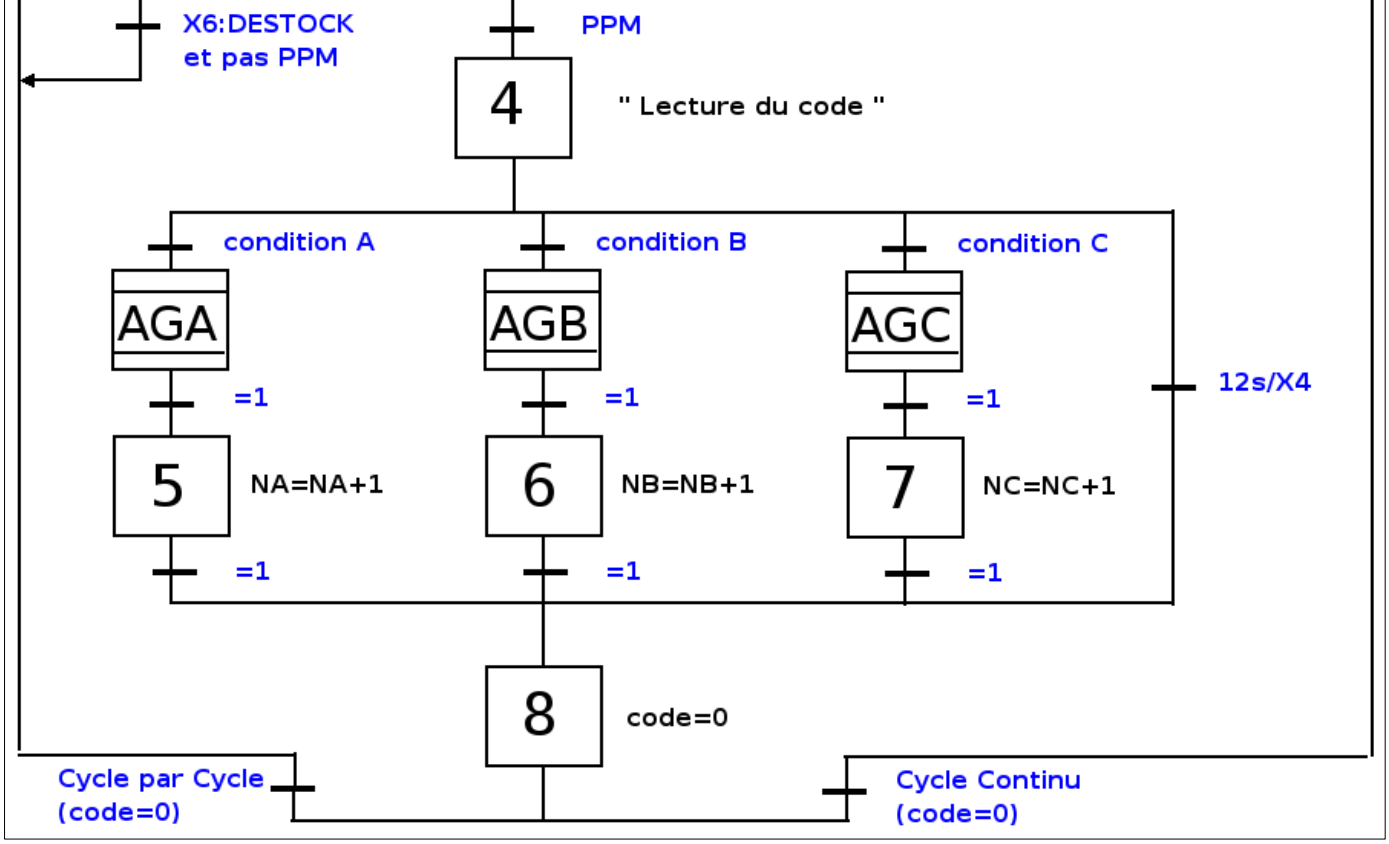
*a) Grafcet **Init***





*b) Grafcet **product***





condition A = (100 ASA) et (NA < 3)

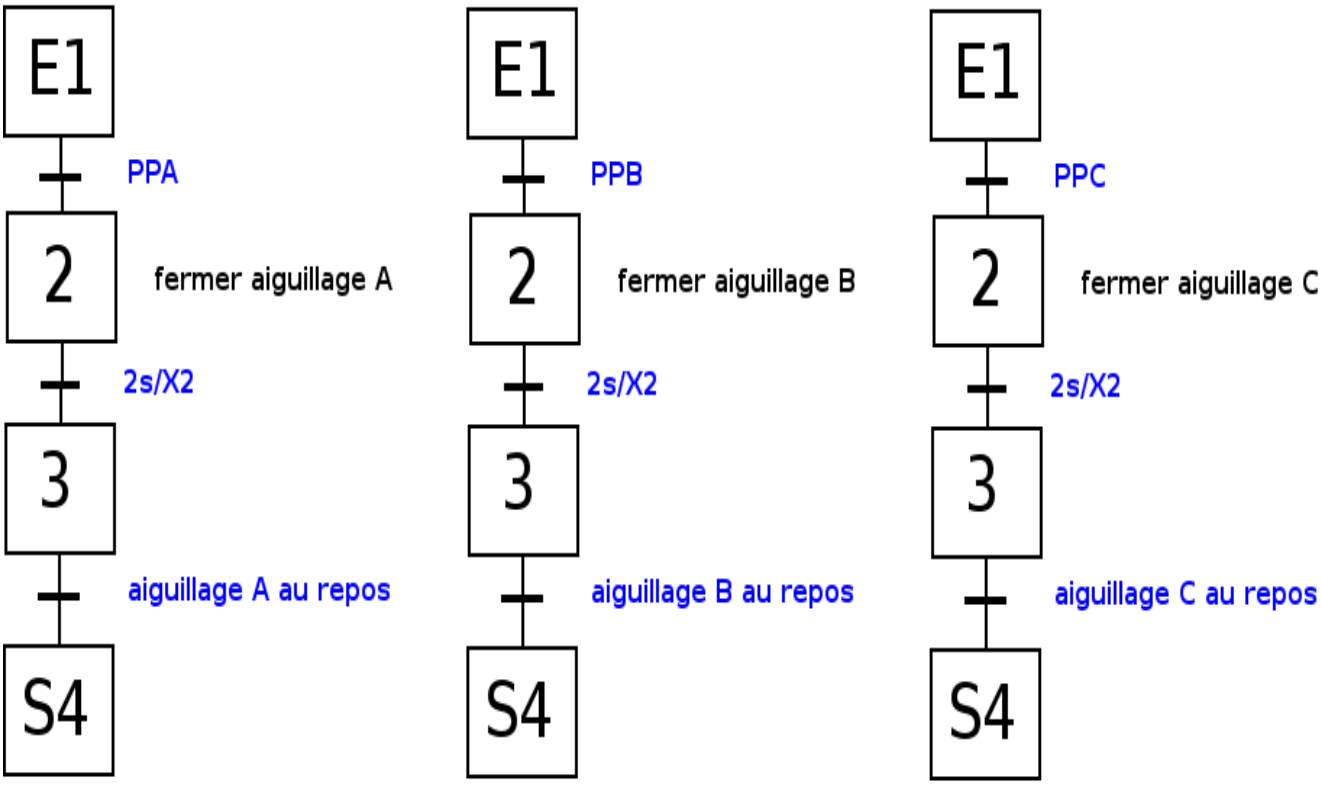
condition B = (200 ASA) et (NB < 3)

condition A = (400 ASA) et (NC < 3)

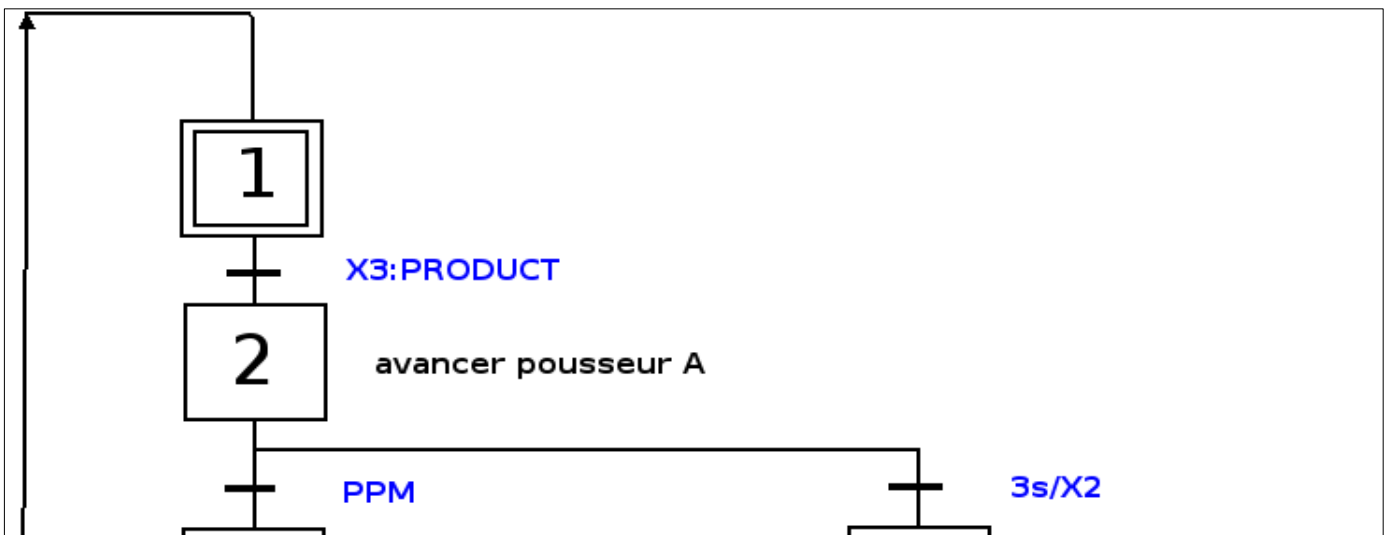
PO en référence = pousseurs A et B rentrés et aiguilleurs A, B et C au repos

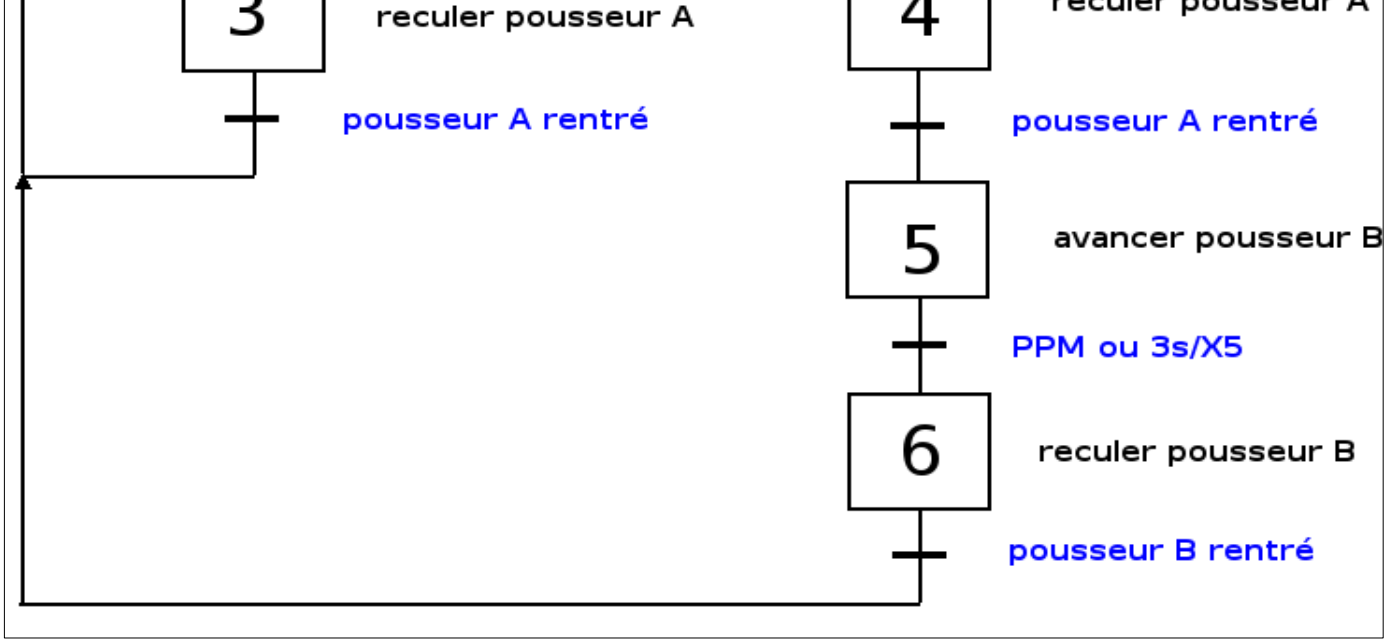
*Expansion des macro-étapes d'aiguillage **AGA, AGB, AGC***

Ces macro-étapes consistent à fermer pendant 2 secondes l'aiguilleur associé suite à une activation du capteur associé.



*c) Grafcet **Destock***





5. Présentation de la carte micro-contrôleur TINI

a) Introduction

La carte micro-contrôleur TINI (Tiny InterNet Interface) est un module « intelligent » fonctionnant sous JAVA et permettant de dialoguer avec le monde extérieur.
Elle se branche sur une carte support comportant toute la connectique nécessaire à la

communication et à son fonctionnement (alimentation, connecteur ethernet, série ...).

b) Composition

micro-contrôleur DS80C400 de chez DALLAS SEMICONDUCTEUR

ROM flash de 1 Mo de type EEPROM (ROM programmable et effaçable électriquement),
contenant:

-le système d'exploitation de la TINI.
-la machine virtuelle JAVA.
-les API JAVA
-les applications JAVA.

RAM statique de 2 Mo contenant:

-la mémoire système.
-le système de fichier.

Communication avec le monde extérieur :

-port série RS232 : pour charger l'environnement temps-réel à partir d'un PC et configurer la carte TINI (attribution d'une IP, d'un masque de sous-réseaux etc....).
-port série DCE : permet à la carte TINI de contrôler un périphérique série.
-port parallèle
-contrôleur 1-Wire : permet à la carte TINI de se connecter à un réseau 1-Wire.
-contrôleur Ethernet : permet à la carte TINI de se connecter à un réseau Ethernet, elle est donc équipée d'une adresse MAC.
-contrôleur CAN : elle en possède 2
-contrôleur I2C

Composition de la carte support

Connecteur 72 broches : pour brancher la TINI

Connecteur RJ45 : port Ethernet

Connecteur RJ11 : port 1-WIRE

Connecteur série DB9 mâle : port DCE

Connecteur série DB9 femelle : port RS232

Connecteur pour l'alimentation 5V

+

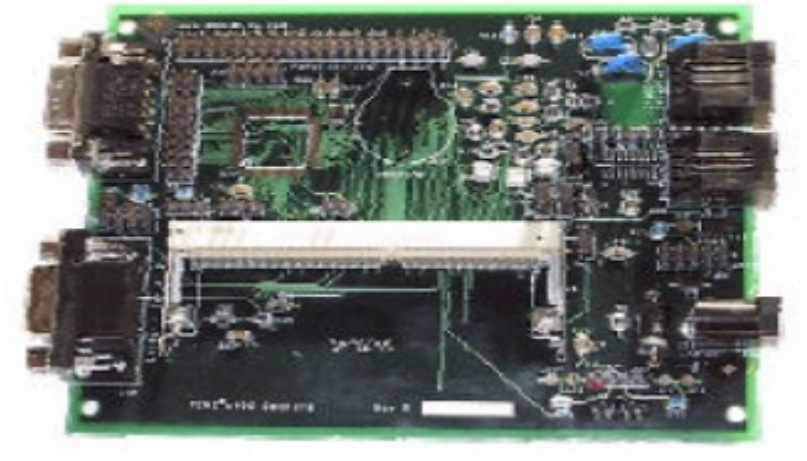
Circuits intégrés et support pour brancher *:*

Connecteur parallèle

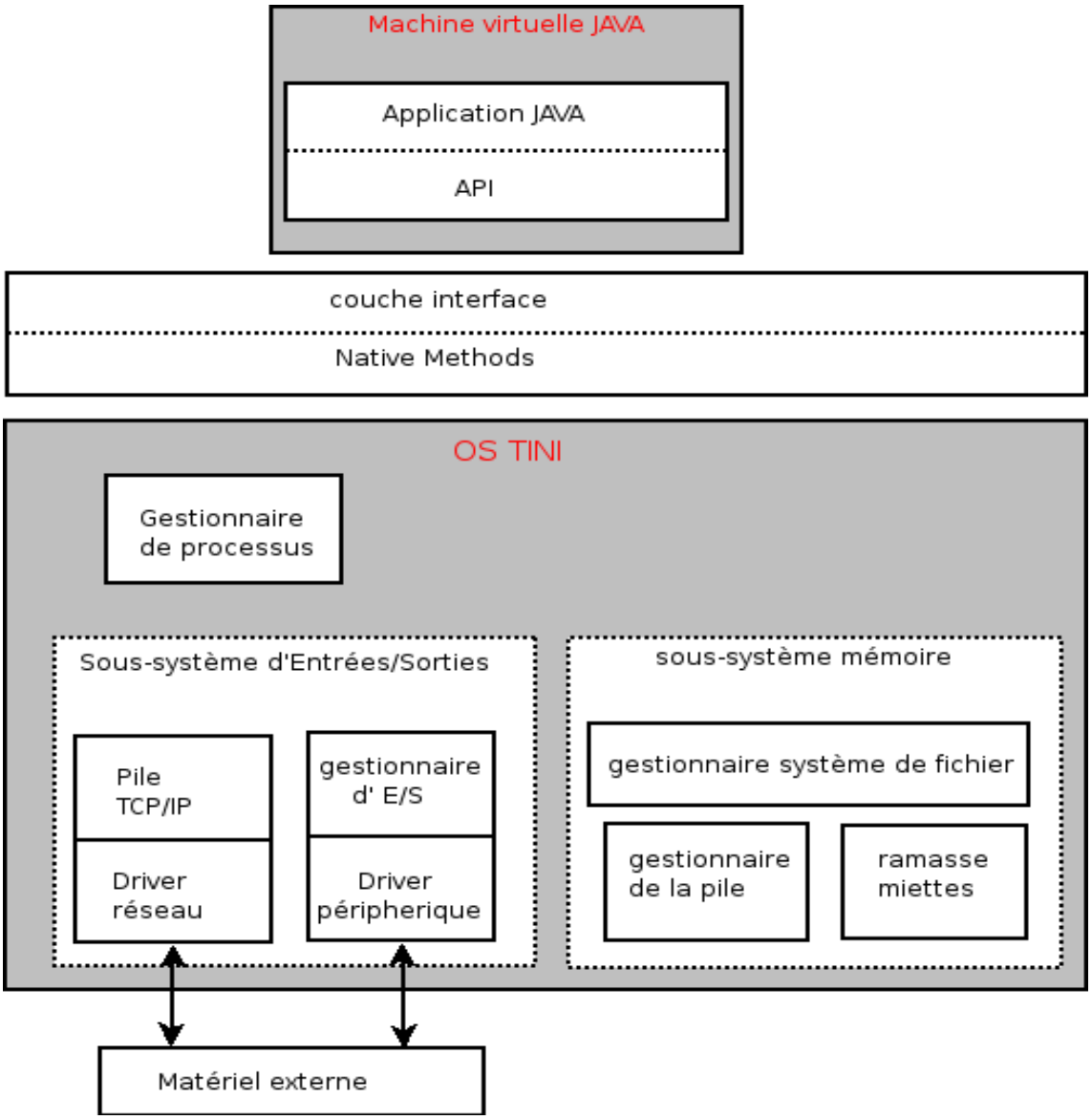
Connecteur CAN

Autre connecteur série (ex I2C...)

Carte Support



c) Environnement logiciel



L' OS TINI est le système d'exploitation de la TINI, il est principalement constitué de :

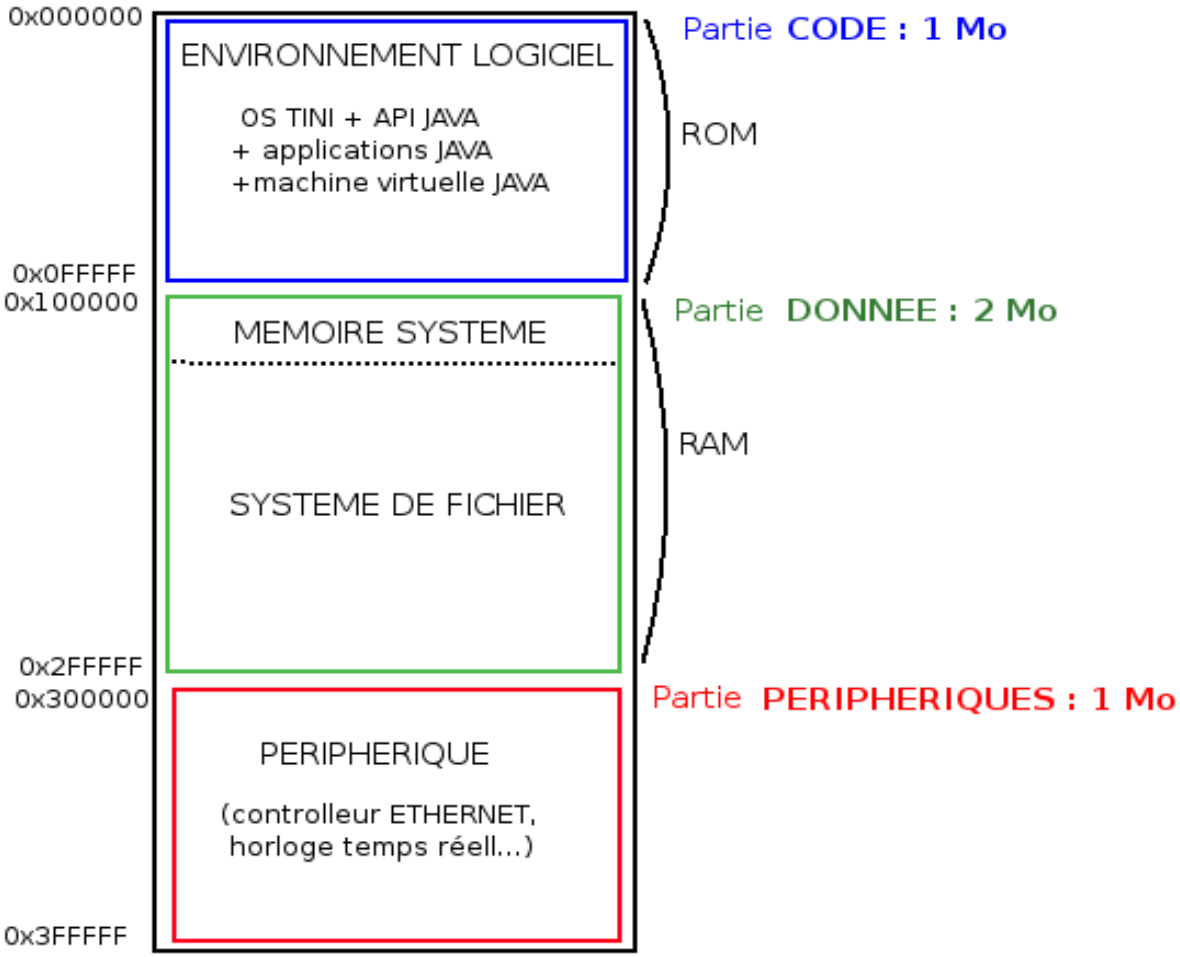
- le sous système mémoire : qui regroupe le gestionnaire du système de fichier, le gestionnaire de la pile ainsi que le ramasse-miettes (garbage collecteur en anglais).

- le sous système d'Entrées/Sorties : il est divisé en deux parties:
  - une pour le réseau Ethernet qui contient la pile TCP/IP
  - une pour les autres périphériques qui contient le gestionnaire d'Entrées/Sorties.

- le gestionnaire de processus.

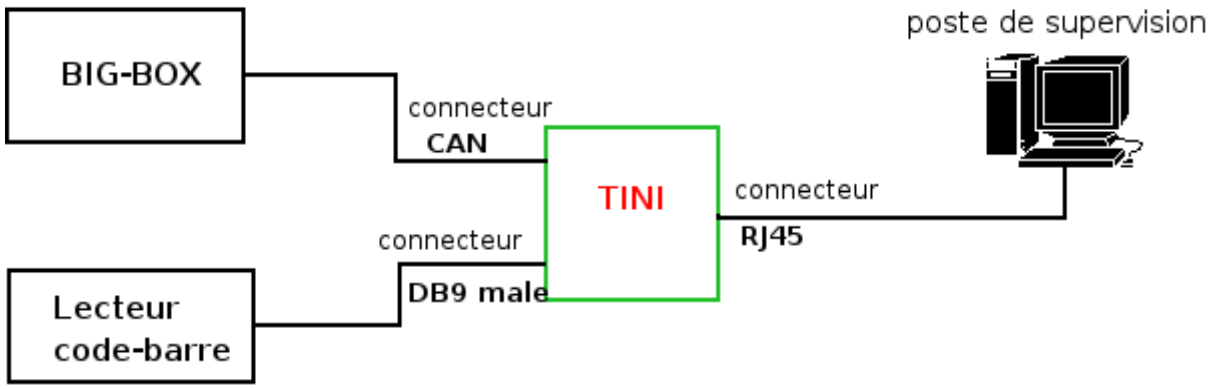
Mais les applications JAVA ne sont pas directement compréhensible par l'OS TINI.
IL existe des « Native Methods » pour permettre au API d'exploiter le matériel. En effet le code généré par la machine virtuelle JAVA n'est pas compatible avec la partie Hardware, nous avons donc besoin de ces « Native Methods ». Elles permettent aussi d'accéder et de configurer les ressources systèmes telle que le watchdog ou l'horloge temps-réelle.
Le lien entre les « Native Methods » et la machine virtuelle JAVA est fait au niveau de la couche interface.

d) Espace adressable :



e) Utilisation dans le projet

Dans notre projet la carte micro-contrôleur TINI constitue le cerveau du système. Elle est reliée à la BIG-BOX, au lecteur code-barre et au poste de supervision (serveur web) de la manière suivante :



***remarque** : Le connecteur CAN à du être rajouté sur la carte support*

C'est elle qui, grâce à ces composant, va gérer le tri :

Le module BIG-BOX est une carte d'entrées sorties qui pilote les capteurs et les actionneurs et qui est branché au pupitre de commande. Cette carte est elle même commandé par le micro-contrôleur TINI qui lui envoie via le bus CAN les consignes à acomplire pour le tri. Elle reçoit aussi l'état des boutons du pupitre de commande, et le code-barre de la pellicule grâce au lecteur-barre.

En même temps elle communique, via le port Ethernet, avec le poste de supervision (serveur web). Elle lui envoie grâce à une socket JAVA l'état du tri

6. INCREMENT 1

Configuration de la carte micro-contrôleurs TINI

BUT

Configurer la carte TINI pour pouvoir ensuite lui transférer un programme et exécuté.

PRINCIPE

Pour pouvoir utiliser la carte TINI il ne suffit pas d'avoir le matériel, il faut aussi avoir les composants logiciel nécessaires à son fonctionnement.
Il y en a trois :

-**Un environnement de développement JAVA** : il contient la machine virtuelle JAVA , le compilateur, ainsi que les API qui sont en fait les classes de base du langage JAVA.

- Les API communication JAVA (Comm API): ils permettent d'utiliser le port RS232 de la TINI, nécessaire au chargement de l'environnement logiciel.
On peut les téléchargés sur le site : http://java.sun.com/products/javacomm/index.html.

- Le SDK (Software Development Kit) : il contient les programmes nécessaires au fonctionnement et à la configuration de la carte TINI tel que le Slush, qui est en fait sont shell (interpréteur de commande)..
On peut le téléchargé sur le site : http://ibutton.com/TINI/software/index.html.

Une fois ces composants installés il faut charger l'environnement logiciel dans la carte TINI.Cela consiste en 2 étapes :
- charger les fichiers **tini.tbin** et **slush.tbin** qui se trouvent dans c:\< *SDK* >\bin.
- Initialiser la pile.

Ensuite il faut lui attribuée une adresse IP et un masque de sous réseau.On pourra donc ouvrir une session telnet sur la carte TINI et on n'aura donc plus besoin de la liaison RS232.

APPLICATION

Pour l'environnement JAVA on choisit le **JDK**1.5.0 (Java Development Kit) de Sun Microsystems, téléchargeable sur le site http://java.sun.com .

Remarque : *avec JDK il est nécessaire d'ajouter le répertoire c:\<répertoire du jdk>\bin au chemin d'exécution, c'est-à-dire à la variable PATH.*

Ensuite on installe les CommAPI ainsi que le SDK.
Un fois ces composants installés il faut faire une série de copie de fichier :

- copié le fichier **c:\commapi\comm.jar** dans le répertoire **c:\< JDK >\jre\lib\ext** , le fichier **c:\commapi\javax.comm.properties** dans le répertoire **c:\< JDK >\jre\lib** et le fichier **c:\commapi\win32comm.dll** dans le répertoire **c:\< JDK >\jre\bin** .

On peut maintenant charger l'environnement logiciel dans la carte TINI.
Pour cela on doit utiliser un programme, JavaKit, qui se trouve dans le dossier :

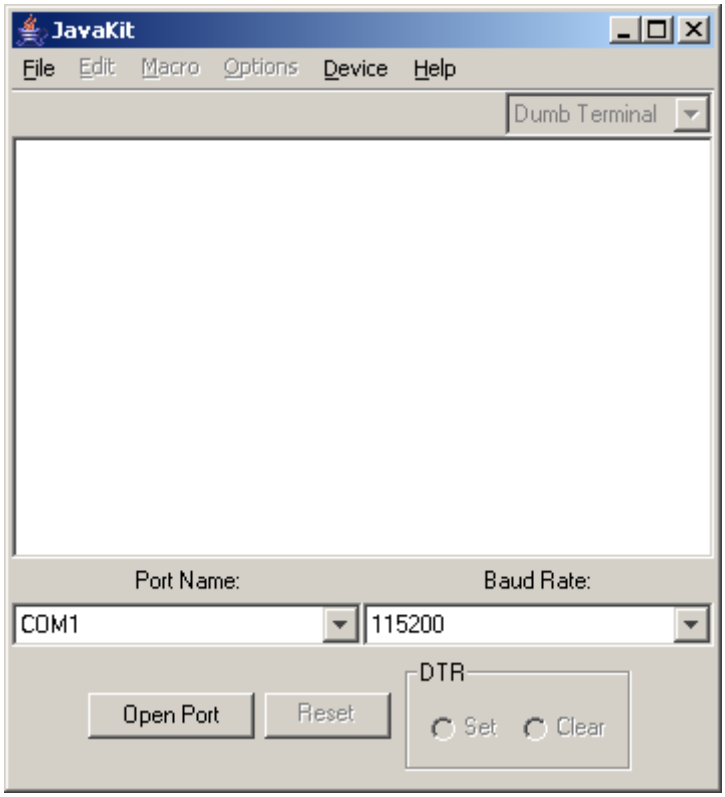
c:\< SDK >\bin\tini.jar . Mais pour utiliser JavaKit on doit ajouter ce dossier au chemin d'exécution, c'est-à-dire à la variable PATH. On peut le faire en ouvrant un invite de commande DOS et taper la commande suivante :

SET CLASSPATH=c:\< SDK >\bin\tini.jar;%CLASSPATH%

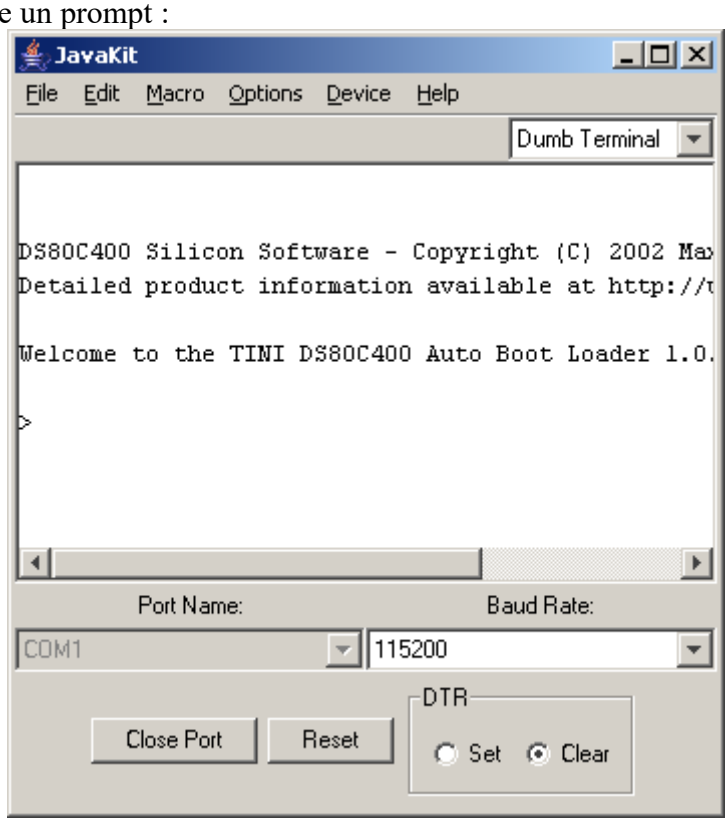
Ensuite on branche la carte TINI au PC grâce à un câble RS232 et on lance JavaKit à l'aide de la commande :

c:\< JDK >\bin\javaw -classpath c:\< SDK >\bin\tini.jar JavaKit

La fenêtre suivante s'affiche :



On sélectionne le port sur lequel est branché la carte TINI (COM1 par exemple)et on clique sur « Open Port », on vérifie que la vitesse de transmission est bien à 115200 et on clique sur *Reset.*
Puis on voit apparaître un prompt :

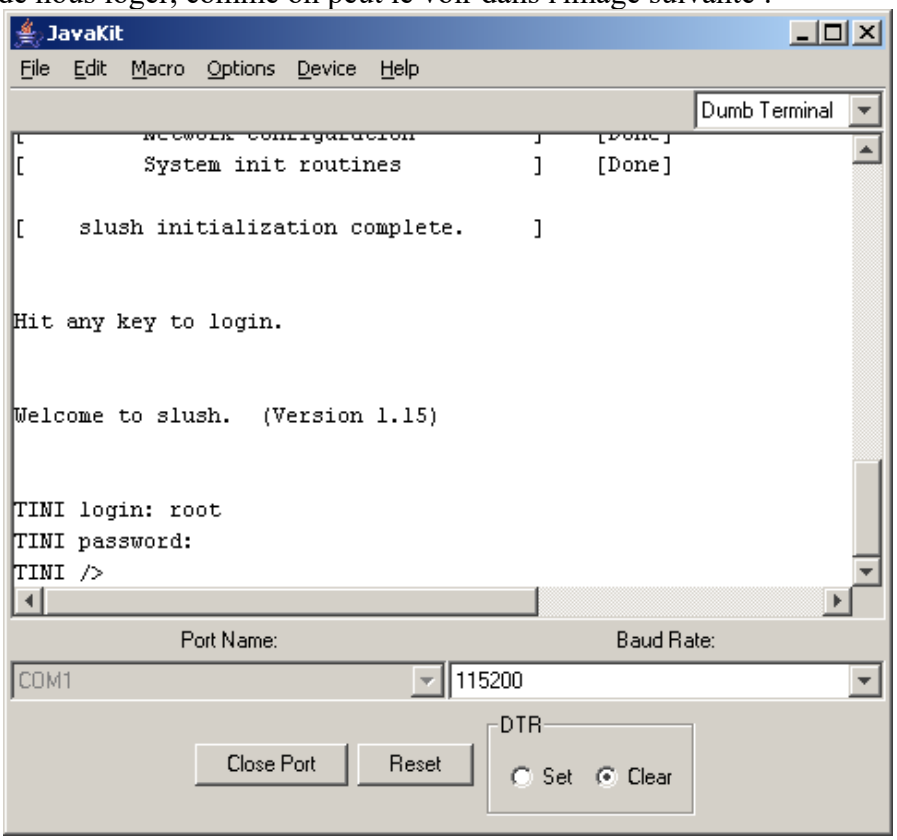


Une fois JavaKit lancé et connecté à la carte TINI, on peut charger l'environnement logiciel. On clique sur *File* et *Load File,* et on sélectionne les fichiers tini.tbin et Slush.tbin. Cela prend quelques minutes.

Il faut maintenant initialiser la pile grâce aux deux commande suivantes, tapés successivement :

- BANK 18
- FILL 0

Maintenant il faut lancer le système : on doit tapé la commande EXIT, le Slush se lance donc et nous demande de nous loger, comme on peut le voir dans l'image suivante :

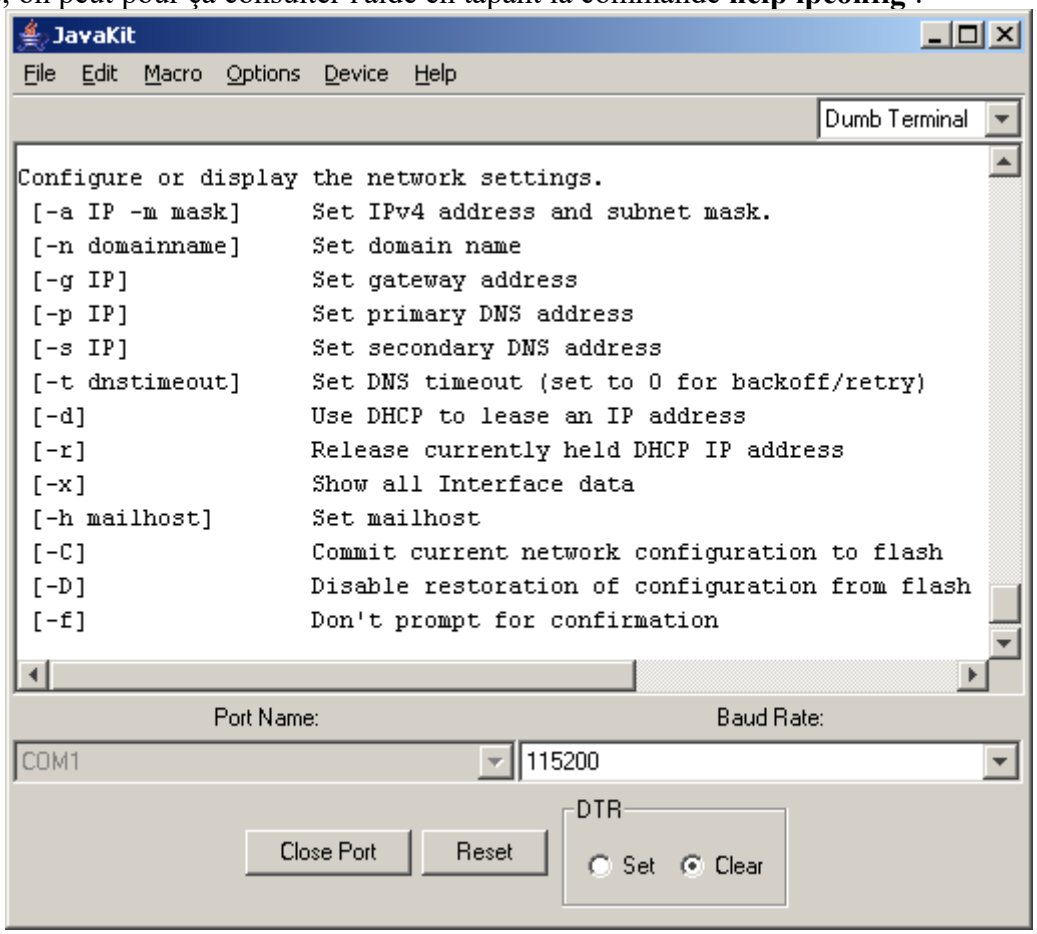


On se log en root (mot de passe *tini*), pour configurer la carte TINI.
La commande **help** affiche toutes commande du Slush et si elle est suivie d'une commande elle affiche la description de celle-ci.
Il existe la commande **ipconfig** pour voir et configurer les connections réseau. Ainsi les options **-a** et **-m** permettent respectivement d'attribuer à la TINI une adresse IP et un masque de sous-réseau.
On tape donc la commande suivante pour attribué l'adresse IP 192.168.109.235, à la carte TINI :

ipconfig -a 192.168.109.235 -m 255.255.255.0

Remarque *: Il faut vérifié au niveau des adresses IP que le PC et la carte TINI sont sur le même réseau.*

Il est aussi possible de lui donné d'autres attributs tel que le nom de domaine ou l'IP de la passerelle, on peut pour ça consulter l'aide en tapant la commande **help ipconfig** :



La CARTE TINI a maintenant une adresse IP, on peut donc enlever le câble RS232 et la connecté au PC avec un câble RJ45 croisé pour ouvrir une session telnet depuis ce PC.

7. INCREMENT 2

Exécution d'un programme « HelloWorld »

BUT

Concevoir un programme qui ne fait rien d'autre qu'afficher une chaîne de caractère à l'écran, et exécuté sur la carte micro-contrôleur TINI en vue de la testé.

PRINCIPE

La conception d'un programme pour être exécuté sur la carte TINI consiste en quatre phases :

- Édition du programme : cela peut être fait avec un simple éditeur de texte en sauvegardant le programme en *.java.*

- Compilation du programme : Il faut compilé le programme avec les classes de l'API JAVA de la carte TINI et non celles du JDK. Elles se trouvent dans :

c:\<SDK>\bin\tiniclasses.jar

- Conversion du programme : en effet le (ou les) fichier *.class* d'un programme ne sont pas interpréter par la machine virtuelle JAVA de la carte TINI. Il faut pour ça le (ou les) convertir en un fichier **.tini.**

Cela se fait grâce à un programme du JDK : **TINIConvertor.**

Il se trouve dans ***c:\< SDK >\bin\tini.jar***, tout comme JavaKit.

- Transfert du programme dans la carte TINI : une fois le programme prêt il faut le transférer à l'aide d'une session FTP sur la carte TINI.

Ensuite il faut ouvrir une session telnet sur la carte TINI et exécuter le programme à l'aide de la commande **java** suivie du nom du programme **.tini.**

APPLICATION:

-Édition du programme :

On édite le programme suivant à l'aide de l'éditeur de texte **JEDIT** :

```
class helloworld{
        public static void main(String[] args) {
                System.out.println("Hello World");
        }
}
```

Ce programme ne fait rien d'autre que d'afficher la célèbre phrase « HelloWorld ». On le sauvegarde en helloworld.java.

-Compilation du programme :

Pour compiler le programme avec L'API de la carte TINI on utilise la commande bootclasspath suivie du dossier contenant les classes, c'est-à-dire c:\<SDK>\bin\tiniclasses.jar

On compile le programme à l'aide de la commande suivante :

javac -bootclasspath c:\<SDK>\bin\tiniclasses.jar helloworld.java

Une fois la compilation terminée on obtient un fichier **helloworld.class**

Conversion du programme :

Pour convertir le programme on utilise **TINIConvertor.**

Pour obtenir de l'aide dessus il suffit de le lancer sans paramètres à l'aide de la commande suivante :

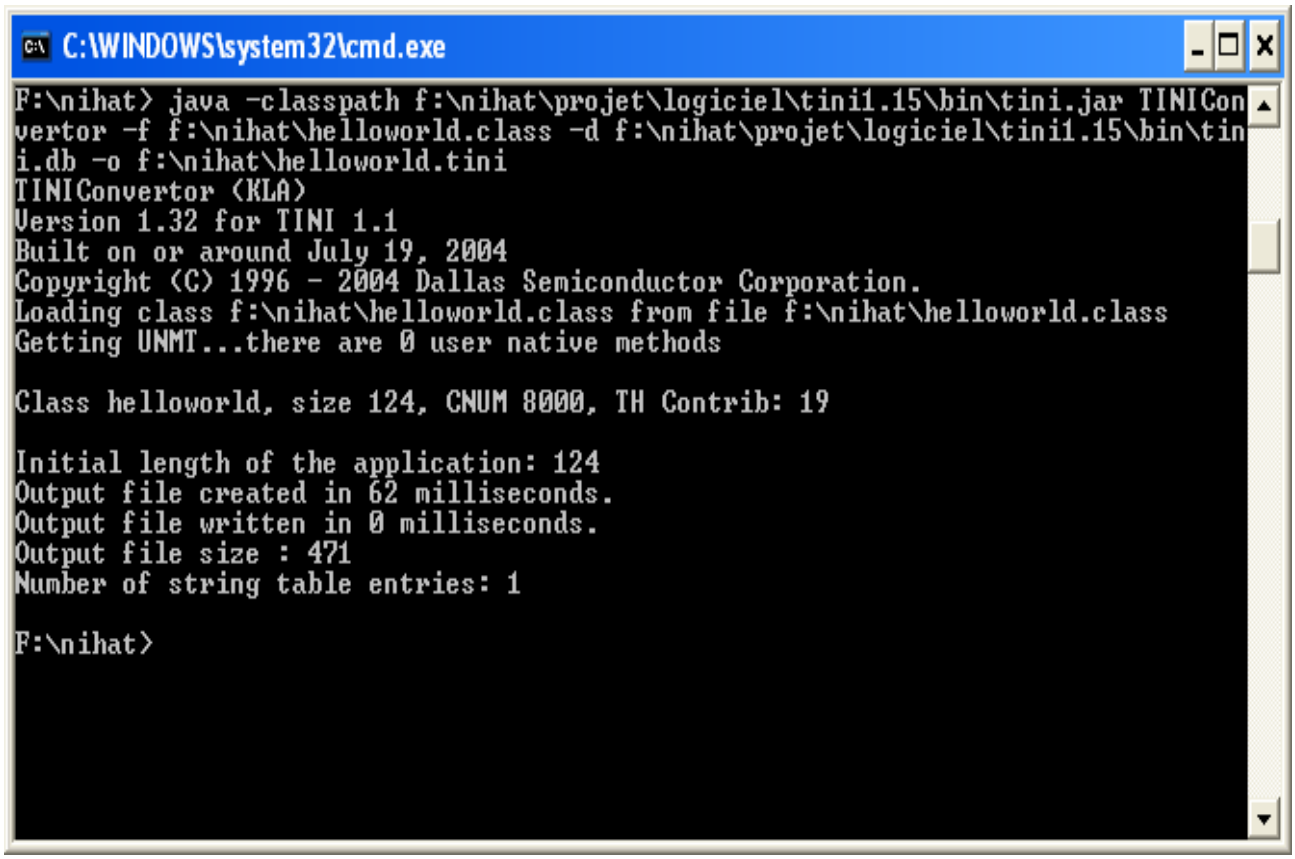
java -classpath c:\< SDK >\bin\tini.jar TINIConvertor

Sinon il se lance avec trois paramètres :

-f *<fichier ou répertoire d'entrée (.class)>* ***-d*** *<base de donée des API>* ***-o*** *<fichier de sortie (.tini)>*

On convertit donc **helloworld.class** avec la commande suivante :

java -classpath c:\< SDK >\bin\tini.jar TINIConvertor -f c:\HelloWorld.class -d c:\< SDK > \bin\tini.db -o HelloWorld.tini



Une fois la conversion terminé on obtient un fichier **helloworld.tini**

- Transfert du programme :

On ouvre une session FTP sur la carte TINI à l'aide de la commande

ftp 192.168.109.235

on se log en root puis on tape la commande ***bin*** pour être sur qu'on est bien en mode binaire et non en mode ASCII.

Ensuite on transfert le programme sur la carte TINI avec la commande :

put helloworld.tini

Pour vérifier que le transfert à bien eu lieu on ouvre une session telnet grâce à la commande

telnet 192.168.109.235

puis on tape la commande ***dir*** qui permet de lister le contenue de la carte TINI.

Maintenant que le programme est bien dans la carte TINI il ne reste plus qu'à exécuté :

on le lance depuis la session telnet avec à l'aide de la commande

java HelloWorld.tini

On peut aussi utilisé l'option « & » (après le nom du fichier) pour lancer le programme en tache de fond.

8. Les threads

a) Introduction

Dans notre projet le pilotage du tri peut se décomposer en plusieurs taches :

Il y a d'abord la tache de tri des pellicules bien-sur , mais pour pouvoir effectivement trié les pellicules il faut à tout moment connaître l'état des capteurs et actionneurs ainsi que des boutons du pupitre de commande.
Il y a aussi la gestion de l'arrêt d'urgence : en effet pour arrêté notre système il faut presser le bouton d'arrêt d'urgence ce qui doit immobiliser les actionneurs jusqu'à pression sur le bouton acquittement qui va réinitialiser le système.

Toute la difficulté réside dans le fait que ces taches doivent se dérouler en parallèles et non l'une à la suite de l'autre.

Nous avons donc 3 taches qui se déroulent en parallèles :

- **le tri des pellicules**
- **scrutation des Entrés/Sorties**
- **gestion de l'arrêt d'urgence**

Pour coder en JAVA plusieurs taches qui s'exécutent en parallèles nous allons utiliser des **THREADS.**

b) Qu'est ce qu'un thread ?

Un thread est une portion de code qui est capable de s'exécuter en parallèle à d'autres traitements (ou d'autres threads). On appelle aussi ça un processus léger, mais un thread et un processus sont deux choses à ne pas confondre :

- un processus a une zone mémoire propre et il ne peut pas empiéter sur la zone mémoire d'un autre processus
- les threads eux se partagent une même zone mémoire : en effet si une application dispose de plusieurs threads alors ces threads se partageront la zone mémoire de l'application qui les a lancés.

Notre application devra donc comporter 3 threads :

- un pour la tache de tri
- un pour la tache de scrutation des E/S
- un pour la tache de gestion de l'arrêt d'urgence

c) Définir un thread en JAVA

En JAVA un thread est une classe qui hérite de la classe **Thread (**qui fait partie des classes de base du langage). Cette classe possède plusieurs méthodes dont les méthode *run( )* et *start( )* :

run( ) : c'est une méthode que l'on doit redéfinir. Elle doit comporter la fonction du thread c'est-à-dire la tache à accomplir en parallèle à d'autre traitements. Par exemple pour le thread du tri des pellicules, la méthode run( ) devra contenir les instructions qui vont piloter les actionneurs pour effectuer le tri.

Start( ) : cette méthode n'est pas à redéfinir. C'est par elle que l'on va lancer le thread. En fait elle lance juste le méthode run( ).

Si on lançait directement la méthode run( ) sans utiliser start( ), on ne pourrait pas faire plusieurs traitements à la fois : en effet lorsqu'on va lancer start( ) elle va elle même lancer run( ) puis on vas ensuite exécuter les instructions suivant start( ), et la méthode run( ) vas s'exécuter en parallèle à ces instructions (qui peuvent éventuellement lancer d'autres threads).

d) L'exclusion mutuelle

Comme on doit coder plusieurs threads, il faut veiller à ce que les différent traitements ne s'entremêle pas.
Il existe pour ça un procédé qu'on appelle l'exclusion mutuelle ou synchronisation.
On vas pour ça utiliser le mot-clés *synchronized* et les méthodes *wait( )* et *notify( )* :

- synchronized :

si on veut qu'un bloc d'instructions (ou une méthode) ne puisse pas s'exécuter en même temps qu'un autre bloc d'instructions alors il faut **synchroniser** ces deux traitements sur un même objet :

```
synchronized(objet) {
        instructions ...
                     ...
}
```

On dit que le thread en cours va poser un verrou sur objet avant d'exécuter les instructions du bloc, puis il va enlevé ce verrou. Ainsi, si pendant l'exécution de ces instructions, un autre bloc synchronisé sur le même objet veut s'exécuter, il va alors se mettre en attente jusqu'à retrait du verrou. Et de même, si un verrou avais déjà été placé sur objet on aurait aussi attendue le retrait du verrou avant d'exécuter les instructions.

- wait( ) :

Cette méthode va permettre de mettre en attente le thread en cours c'est-à-dire le thread qui vas lancer la méthode :
Exemple :

```
class thread1 extends Thread {
        run() {
                instruction.....
                thread2.wait( );
                instruction.....
```

}

Dans cette exemple thread1 est le thread en cours, c'est donc lui qui est mis en attente et non thread2. On dit que thread1 est mis en attente par thread2.

- notify( ) :

Cette méthode va permettre de réveiller un thread qui a été mis en attente par le thread en cours. Dans l'exemple ci-dessus, pour réveiller thread1 , il faut que dans le run( ) de thread2 il y est une instruction **notify( ).** Elle vas réveiller thread1 qui a était mis en attente par thread2.

9. INCREMENT 3

Intégration avec l'étudiant 1

BUT :

Mettre en place un protocole d'échange avec l'étudiant 1 pour pouvoir utiliser :

- son module de communication par bus can qui va permettre de lire l'état des capteurs, actionneurs et boutons poussoirs, et de piloter les actionneurs du système.
- son module de lecture code-barre qui va déterminé la sensibilité des pellicules (100ASA, 200ASA, 400ASA).

PRINCIPE :

L'étudiant 1 réalise une classe **ComCan** qui va permettre d'utiliser le bus can de la carte TINI. Cette classe comporte 2 méthodes principales :

- ***byte[] lire( )*** : comme son nom l'indique cette méthode va permettre de lire sur le bus. Le tableau de 3 octets retourné par la méthode correspond aux 24 entrées de la Big-Box.

  La disposition des capteurs, actionneurs et boutons selon ces 3 octets se fait de la façon suivante :

data[0] :	bit 0 : bouton marche
	bit 1 : cycle continue
	bit 2 : cycle/cycle
	bit 3 : bouton acquittement
	bit 4 : aiguilleur C
	bit 5 : aiguilleur B
	bit 6 : aiguilleur A
	bit 7 : pousseur magasin A
data[1] :	bit 0 : pousseur magasin B
	bit 2 : capteur PPM

bit 4 : capteur aiguilleur A
bit 6 : capteur aiguilleur B

data[2] : bit 0 : capteur aiguilleur C
bit 2 : bouton d'arrêt d'urgence

Chaque capteurs, actionneurs ou boutons est actif à 1.

Une fois la lecture terminé il faut faire un masquage en « & » pour isolé le bit que l'on souhaite connaître.

Exemple : On veut connaître l'état du *capteur de l'aiguilleur A*, qui correspond au bit 4 de data[1].

Supposons que ce capteur est actif, le bit 4 de data[1] est donc égal à 1.

Il faut donc faire un « & » entre data[1] et la valeur hexa 0x10 qui correspond en binaire à 0001 0000 .

Comme on sait que ***0 « ET » n'importe quoi*** donnera toujours 0, on aura donc le résultat suivant :

	0001 0000	0x10
&		
	xxx1 xxxx	data[1]
	0001 0000	

Il faut maintenant faire un décalage à droite de 4 bits pour avoir l'état du bit 4 de data[1] et donc l'état du capteur :

$0001\ 0000 \rightarrow = 1$

- ***ecrire(byte[ ] donnée) :*** C'est la méthode qui va nous permettre, d'écrire sur le bus can et donc d'agir sur les actionneurs du système. Le tableau d'octets passé en paramètre est en fait un tableau d'un seul octet qui correspond au 8 sorties de la Big-Box. Chaque bits de cet octet correspond à un actionneur.

  La disposition des actionneurs selon cet octet se fait de la manière suivante :

  donnée[0] : bit 0 : pousseur magasin A actif
  bit 1 : pousseur magasin A au repos
  bit 2 : pousseur magasin B actif
  bit 3 : pousseur magasin B au repos

bit 4 : aiguilleur A actif
bit 5 : aiguilleur B actif
bit 6 : aiguilleur C actif
bit 7 : convoyeur activé

Pour pouvoir agir sur un seul actionneur sans changer l'état des autres il faudra, avant d'utiliser la méthode, faire un masquage en « OU » logique entre l'actionneur à activé (ou désactivé) et l'ancienne valeur des actionneurs. C'est le résultat de cette opération que l'on va passé en paramètres à la méthode écrire.

Exemple : On veut activé le pousseur du magasin A (bit 0) tout en laissant le convoyeur en marche (bit 7).

L'ancienne valeur des actionneurs est donc en binaire : 1000 0000 et en hexa 0x80 (convoyeur actif)

pousseur magasin A	0000 0001		On sait que ***1 « OU » n'importe quoi*** donnera toujours 1, on vas donc gardé les anciens actionneurs actif .
		« ou »	
convoyeur	1000 0000		
	1000 0001		

On vas donc activé le pousseur magasin A tout en laissant le convoyeur en marche.

Pour la lecture du code-barre, l'étudiant 2 a codé une classé LecteurCodeBarre qui contient une méthode qui renvoi la lettre 'a', 'b' ou 'c' en fonction de la sensibilité de la pellicule (respectivement 100ASA, 200ASA et 400ASA).

APPLICATION

LECTURE : On veut connaître l'état du *capteur de l'aiguilleur A*

```
ComCan bus=new ComCan();          // création d'un objet de type ComCan
byte CA=(byte) 0x10;              // correspond au capteur de l'aiguilleur A : bit 4 de data[1] actif
```

```
byte valCA = (byte) 0x00;;  // correspond à l'état du capteur
byte[] data=new byte[3];    // création du tableau de 3 octets
data=bus.lire( );           //lecture
valCA = data[1] & CA;       // masquage en « & »
valCA = valCA>>4;           // décalage de 4 bits à droite
```

ECRITURE : On veut activé le pousseur du magasin A (bit 0) tout en laissant le convoyeur en marche (bit 7).

```
ComCan bus=new ComCan();        // création d'un objet de type ComCan
byte PMAon=(byte) 0x01;  // correspond a l'activation du pousseur magasin A
byte sauv=(byte) 0x80;      // correspond l'ancien état des actionneurs ( seul le convoyeur est actif)
byte[] donnée=new byte[1]; // création du tableau
donnée[0] = sauv | PMAon; // masquage en « OU »
bus.ecrire(donnée);         //écriture
```

LECTURE CODE-BARRE :

```
LectureCodeBarre LCB=new LectureCodeBarre( );  // Création d'un objet de type
LectureCodeBarre
char type;                  // correspond au type de pellicule ; a, b ou c
type = LCB.type();
```

Partie étudiant 3

1)Présentation

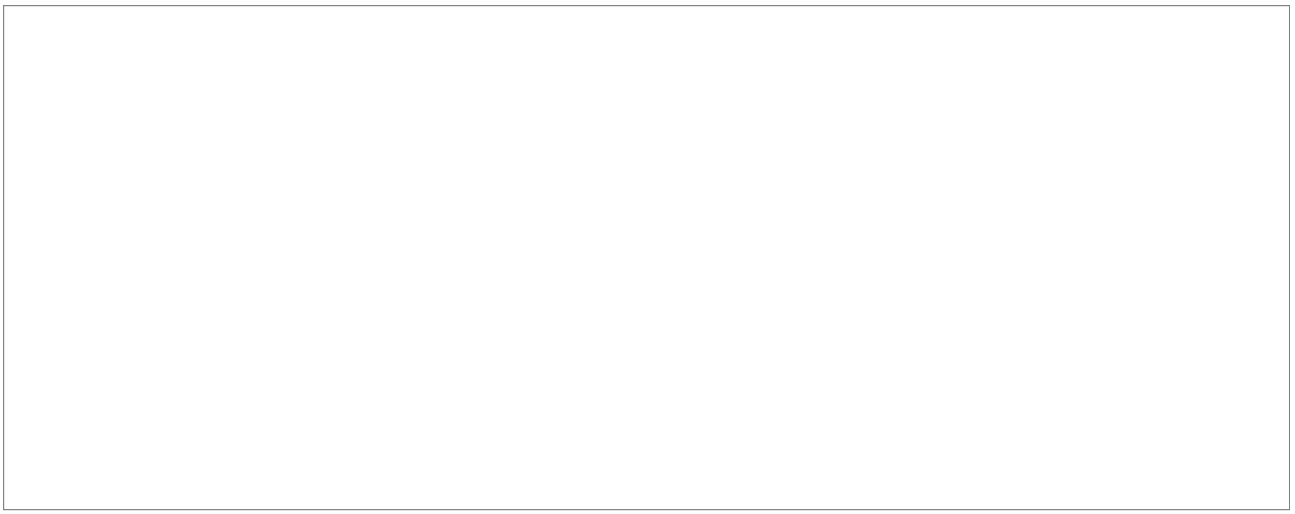
L'objectif de cette partie est de permettre la supervision de l'acheminement des pellicules à travers une Inerface Homme-Machine(IHM).

Le superviseur pourra consulter, à travers la page, les différentes statistiques:

- nombre de pellicules triées par catégorie (100ASA, 200ASA, 400ASA)
- cadence journalière de la machine.

Le superviseur pourra également, grâce à une actualisation, observer le cheminement des pellicules sur la machine avec l'affichage d'imges représentant les actionneurs et les capteurs lors du passage des pellicules sur le tapis.

2)Schéma structurel

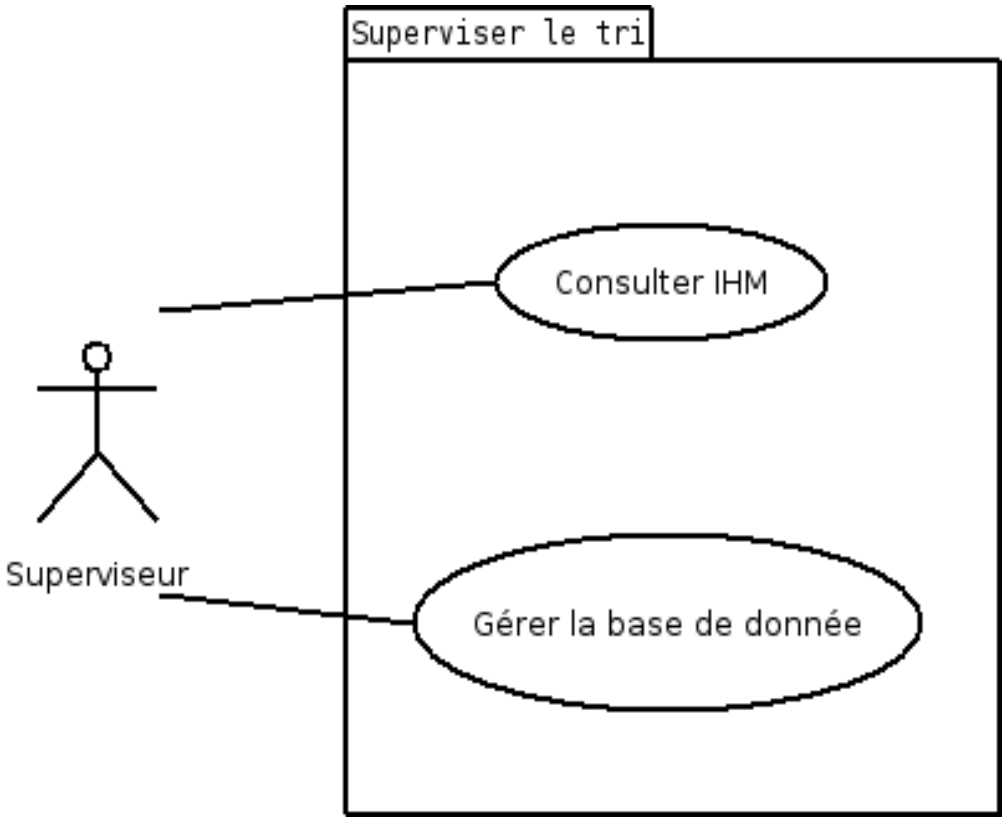


J'utilise pour ma partie un ordinateur sous Windows XP. Cet ordinateur possède un serveur de type Tomcat dont les spécificités seront expliquées plus tard. Le PC de supervision communiquera avec une base de donnée qui va lui fournir les différentes statistiques. La consultation se fera à travers une page HTML contenu dans le serveur web.

3)Analyse UML

Cette analyse va permettre une meilleure compréhension de la partie technique, en utilisant la modélisation UML.

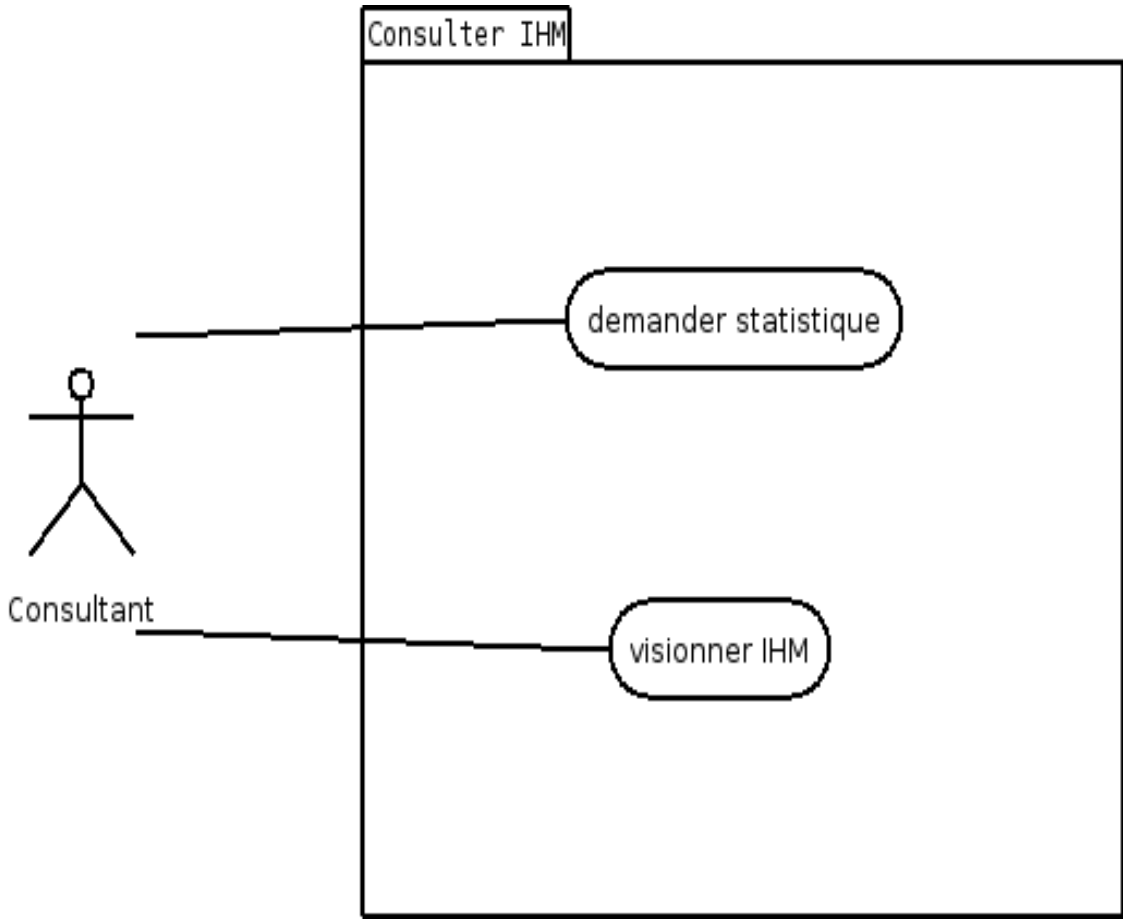
. *Use case personnel*



Le superviseur va pouvoir, à partir d'un poste distant, consulter la page HTML située dans le serveur Web .

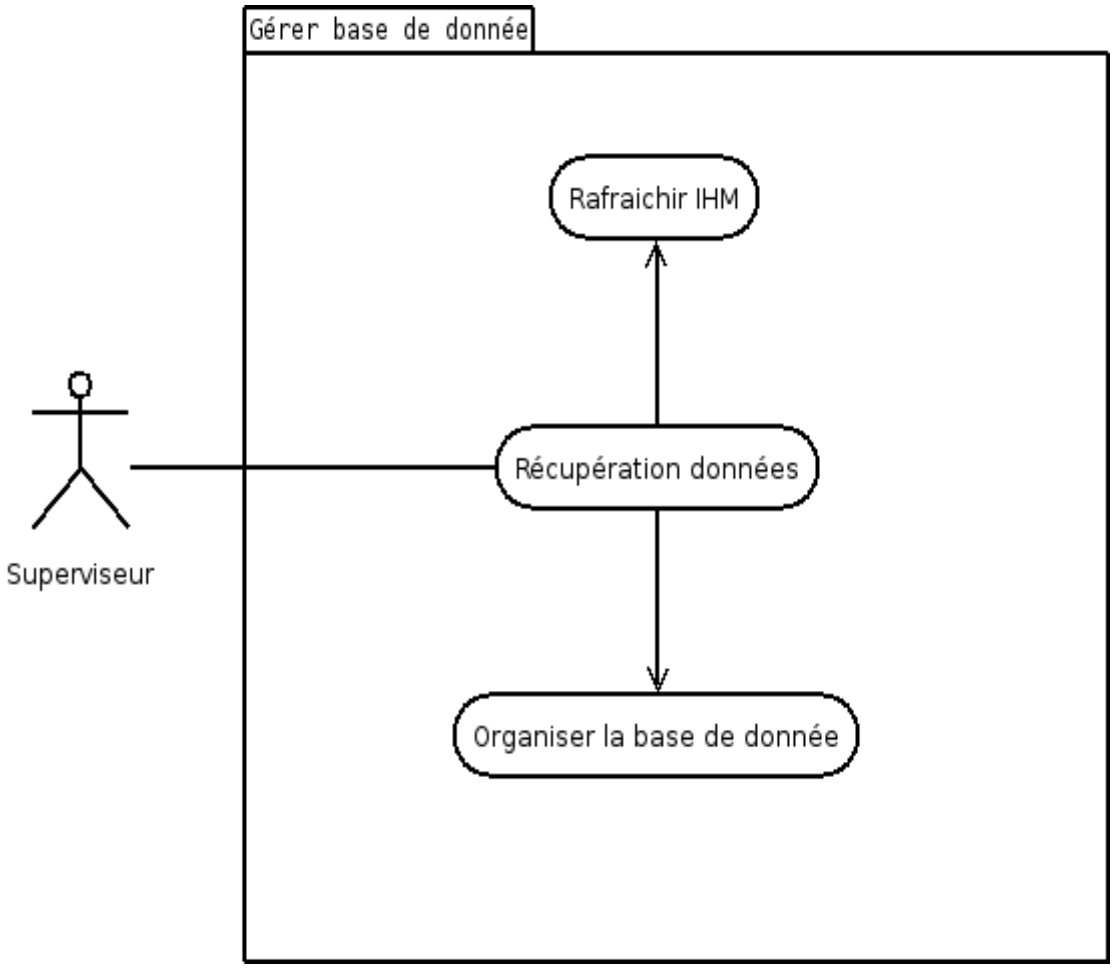
De plus, il pourra gérer la base de donnée en récupérant les statistiques puis en organisant la base.

Use case « consulter IHM »



Un consultant va, à partir d'une page HTML située sur le serveur, visionner l'IHM à distance sur un poste distant ainsi que d'interroger la base de donnée sur les différentes statistiques repésentant le stock de pellicules ainsi que sa cadence.

Use case « gérer base de donnée »

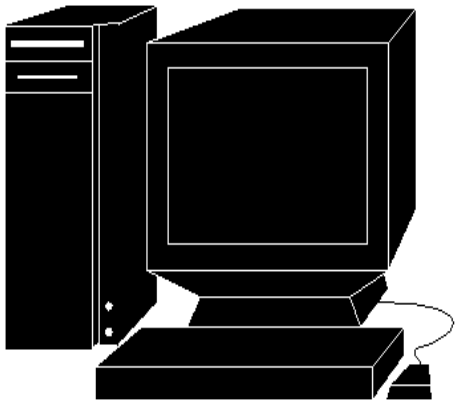


Le superviseur va récupérer les données sur l'IHM puis grâce à celle-çi permettre un rafraichissement de celle-çi tout en organisant les données à l'intérieur de la base de donnée.

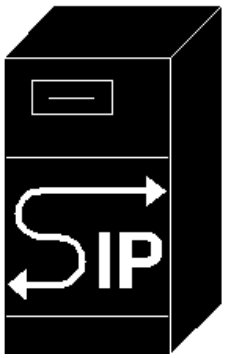
4)Contraintes logicielles et matérielles

Dans le but de réaliser le projet, nous devons nous fixer des contraintes tant au niveau matérielles que logicielles.

- Un PC sous Windows XP



- Un serveur web TOMCAT



- Technologies JAVA(JSP)

Le projet doit totalement être constituer du langage Java ou technologies qui s'y approchent. Pour cela l'utilisation du JSP dans les pages HTML va permettre de réaliser dynamiquement l'IHM.

- Base de donnée MySql

La réalisation d'une base de donnée est nécessaire dans ce projet car elle va permettre au superviseur d'observer à distance l'existence d'anomalies sur le système. La base de donnée va informer son consultant sur le nombre de pellicules triées, la cadence de la machine. Ainsi le superviseur pourra par la suite informer les techniciens sur les solutions à apporter au système.

5)Serveur Tomcat

But:

Fournir des pages Web à des clients distants.

Principe:

Web Service : application utilisable via un réseau du type Internet.

Principale technique : **J2EE** (SUN, IBM, Apache Group).

TOMCAT fournit le support pour tous les services J2EE. On utilise en particulier des **JSP** (Java Server Page), pages HTML qui embarquent des instructions Java ; et des **servlets**, programmes Java qui génèrent à la volée le code HTML. Au départ module conçu pour le serveur Apache, Tomcat est devenu un serveur autonome, tout en Java, développé par le projet Jakarta.

Par défaut, Tomcat démarre sur le port 8080 (http://localhost:8080).

Architecture du serveur:

/bin:
==>Scripts de démarrage, arrêt et autres.

/common/lib:
==>Contient les bibliotèques (**"jar"**) utilisées par Tomcat pour toute Webapp.

/common/classes :
==>Contient les programmes (**"class"**) qui ne sont pas dans un "jar".

/conf:
==>Fichiers de configuration, notamment le fichier **server.xml**

/logs :
==>Fichiers journaux (accès et erreurs)

/webapps:
==>Emplacement où mettre les applications

Sous **/webapps** on range les applications. Tomcat 5.0 fournit une application ROOT avec une documentation et des exemples.

6)JSP
Java Server Pages

1)*Présentation*

Les JSP sont un standard permettant de développer des applications web intéractives, dont le contenu est dynamique.

Il s'agit en réalité d'un langage de script puissant (un langage interprété) exécuté du côté du serveur (comme le CGI,PHP,ASP,...) et non du côté client (les scripts écrits en JavaScript ou les applets Java s'exécutent dans le navigateur de la personne connectée à un site).

Les JSP sont intégrables au sein d'une page Web en HTML à l'aide de balises spéciales permettant au serveur Web de savoir que le code compris à l'intérieur de ces balises doit être interprété afin de renvoyer du code HTML au navigateur du client.

2)*Fonctionnement*

Une page utilisant les Java Server Pages est exécutée au moment de la requête par un serveur web. Lorsqu'un utilisateur appelle une page JSP, le serveur Web appelle le moteur de JSP qui crée un code source Java à partir du script JSP. Elle compile la classe afin de fournir un fichier compilé (d'extension **.class**), cela constitue une servlet à partir du script JSP.

La plupart des technologies de pages actives (ASP, PHP, ...) reposent sur un code interprété, cela demande beaucoup de ressources pour fournir la réponse HTTP. Les JSP sont compilées, elles sont beaucoup plus rapides à l'exécution.

En incluant dans des balises spécifiques, le code JSP au sein du fichier HTML, elles fournissent une technologie rapide afin de créer des pages dynamiques.

De plus, les JSP étant basées sur Java côté serveur, elles possèdent toutes les caractéristiques faisant la force de Java :

- les JSP sont multithreadées
- les JSP sont portables
- les JSP sont orientées objet
- les JSP sont sûres

3)*Petit exemple de démonstration*

Ce petit exemple très simple de code JSP va permettre l'affichage de l'heure de système. L'utlisateur ouvrira un navigateur web puis entrera l'adresse du serveur ainsi que le nom de l'application pour faire apparaître l'affichage de l'heure du système.

Pour cela, il faut donc créer dans le répertoire **/webapps**, le fichier **Test.jsp** contenant le code suivant:

```
<html>                                  //ouverture de la balise de la page HTML
<head><title>JSP de test</title></head> //en tête de la page HTML
<body>                                  //ouverture du corps de la page HTML
<h1>JSP Test</h1>                       //affichage d'un titre
Time: <%= new java.util.Date() %>       //appel de la fonction Date() dans les balises
                                        //spécifiques JSP <%.....%>
</body>                                 //fermeture du corps de la page HTML
</html>                                 //fermeture de la page HTML
```

Utilisation:

http//@serveur:8080/Test.jsp

Affichage de l'heure sur le navigateur.

7)INCREMENT 1

Installation et configuration du serveur Tomcat

BUT

Le but de cet incrément est d'installer le serveur web Tomcat qui va permettre l'affichage des pages HTML.

Pour cela, il suffit de télécharger le serveur sur le site http://jakarta.apache.org.

PRINCIPE

1)*Installation Tomcat*

L'installation du serveur s'effectue en plusieurs étapes:

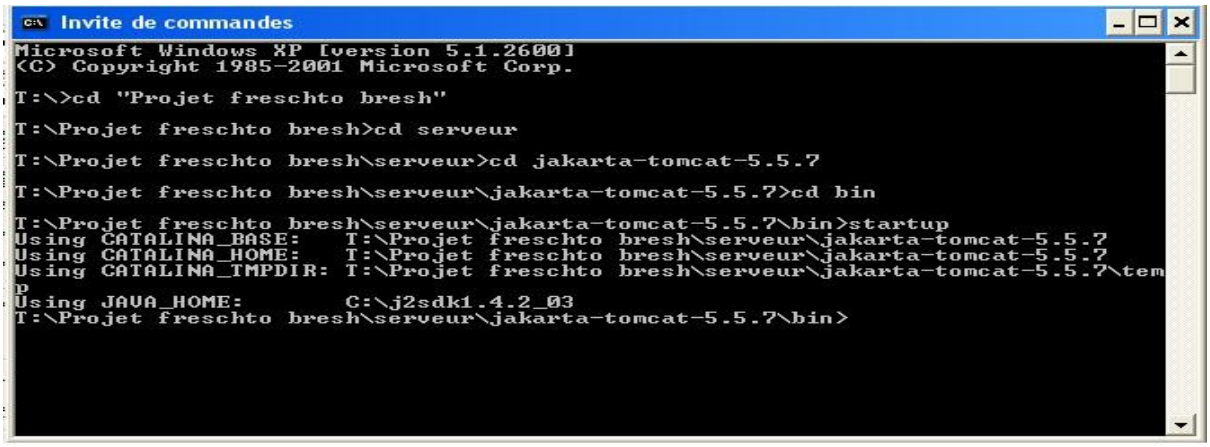
- *téléchargement du fichier jakarta-tomcat-5.5.7.zip*
- *décompression du fichier dans le sous-dossier jakarta-tomcat-5.5.7*
- *ajouter au système la variable JAVA-HOME dans Propriétés->Avancé->Variables environnement*
- *la valeur du chemin d'accès JDK est:*
  ==>C:\j2sdk1.4.2_03

Désormais le serveur web Tomcat est installé et prêt à être configurer.

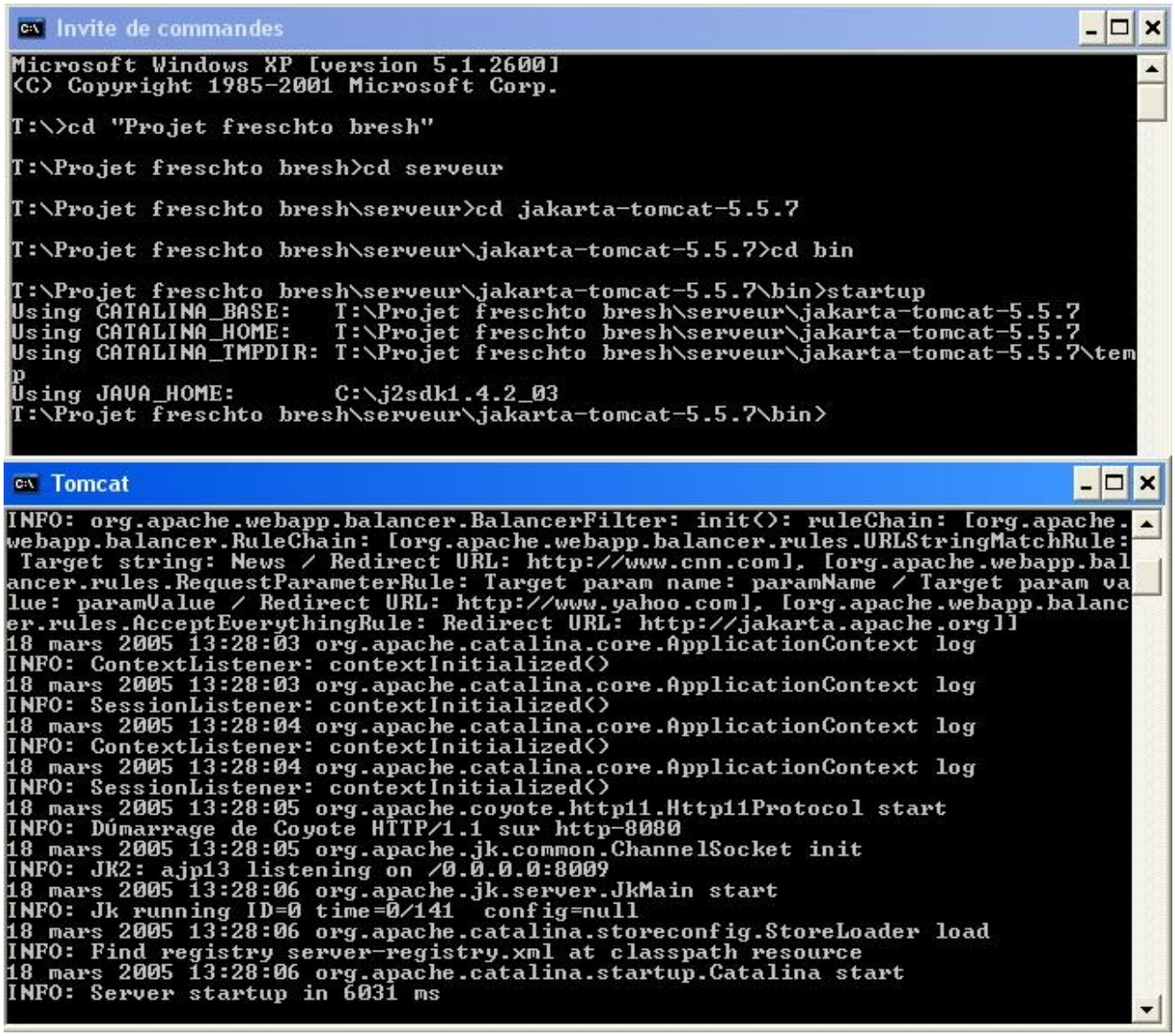
2)*Lancement du serveur*

Pour vérifier le bon fonctionnement du serveur, il suffit d'ouvrir un navigateur web et d'entrer l'adresse http://127.0.0.1:8080.

Ensuite, il faut ouvrir une invite de commande où l'on peut lancer le serveur à partir de la commande.



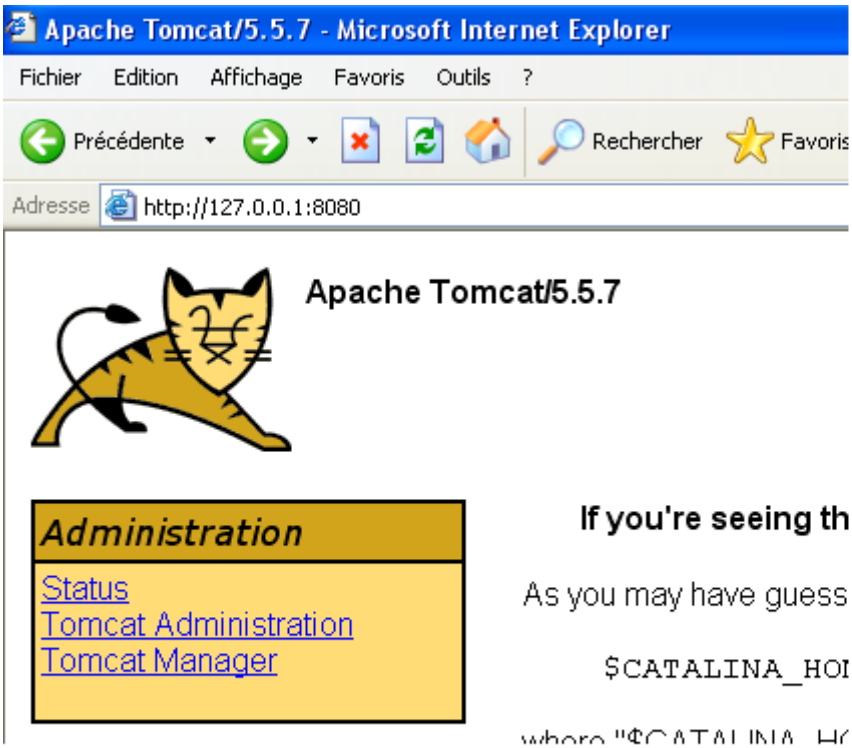
Dès le lancement effectué, une deuxième invite de commande s'ouvre: c'est le processus du serveur Tomcat qui s'effectue.



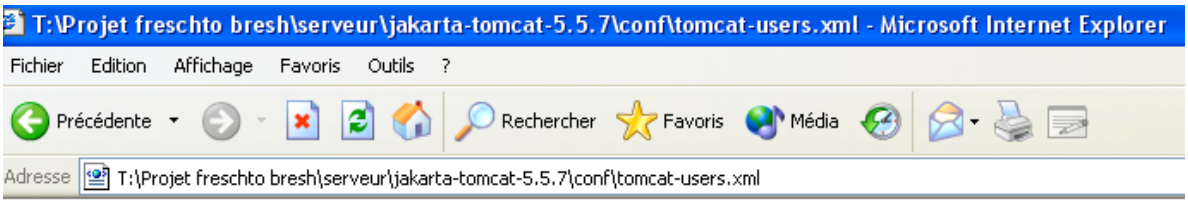
Une fois arrivée à cette étape, le serveur web Tomcat est employable.

3)*Utilisation du serveur web Tomcat*

Une fois le serveur installé, nous pouvons l'utiliser donc y incorporer les applications web.

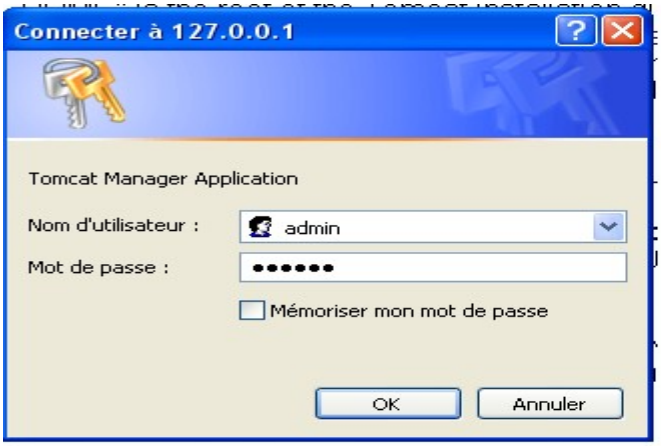


Pour attribuer un utilisateur au serveur, il faut éditer le fichier "tomcat-users.xml" qui se trouve dans le répertoire «conf» de Tomcat.

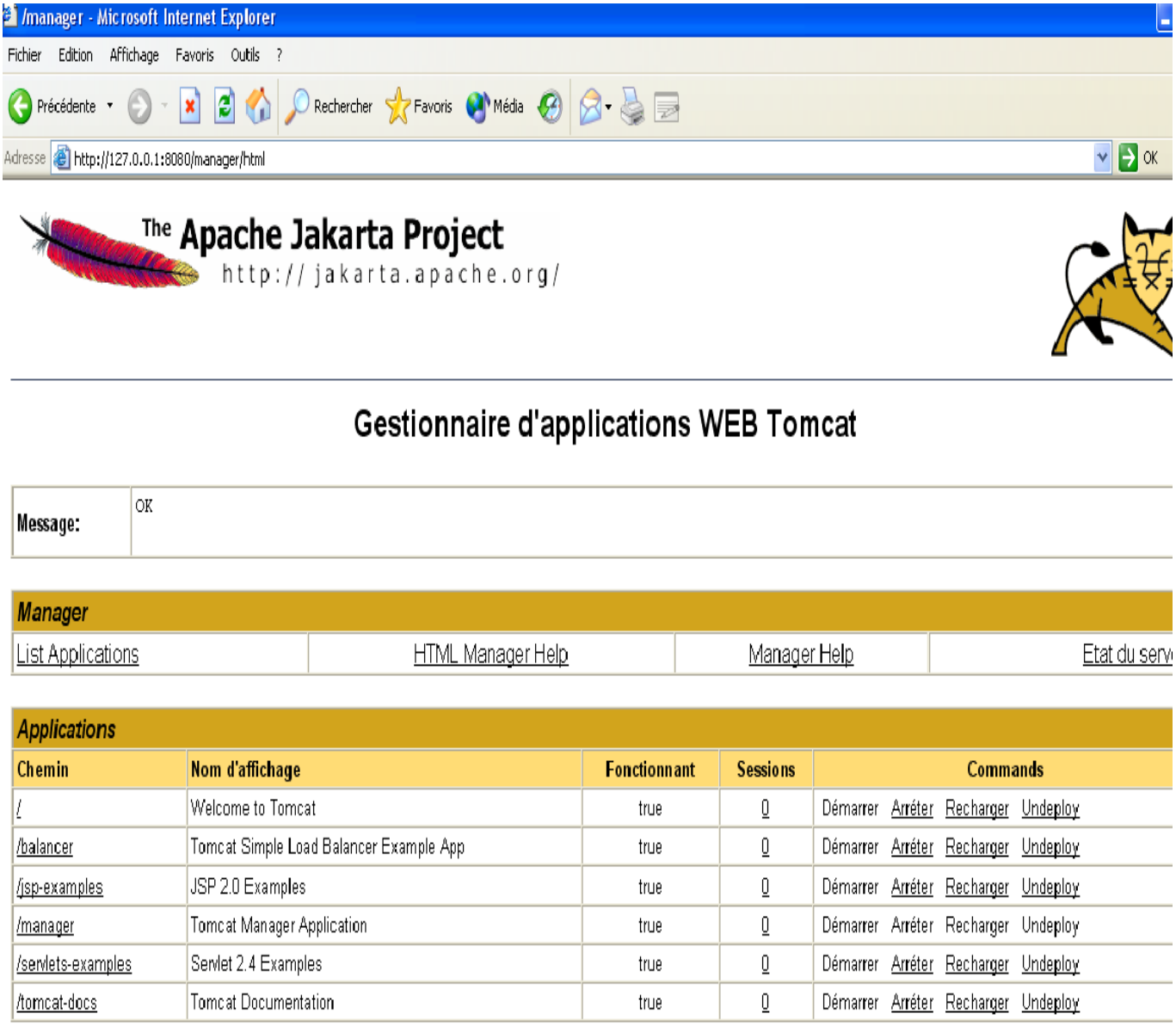



```
<?xml version="1.0" encoding="utf-8" ?>
- <tomcat-users>
    <role rolename="tomcat" />
    <role rolename="role1" />
    <role rolename="manager" />
    <role rolename="admin" />
    <user username="tomcat" password="tomcat" roles="tomcat" />
    <user username="role1" password="tomcat" roles="role1" />
    <user username="both" password="tomcat" roles="tomcat,role1" />
    <user username="admin" password="astral" roles="admin,manager" />
  </tomcat-users>
```

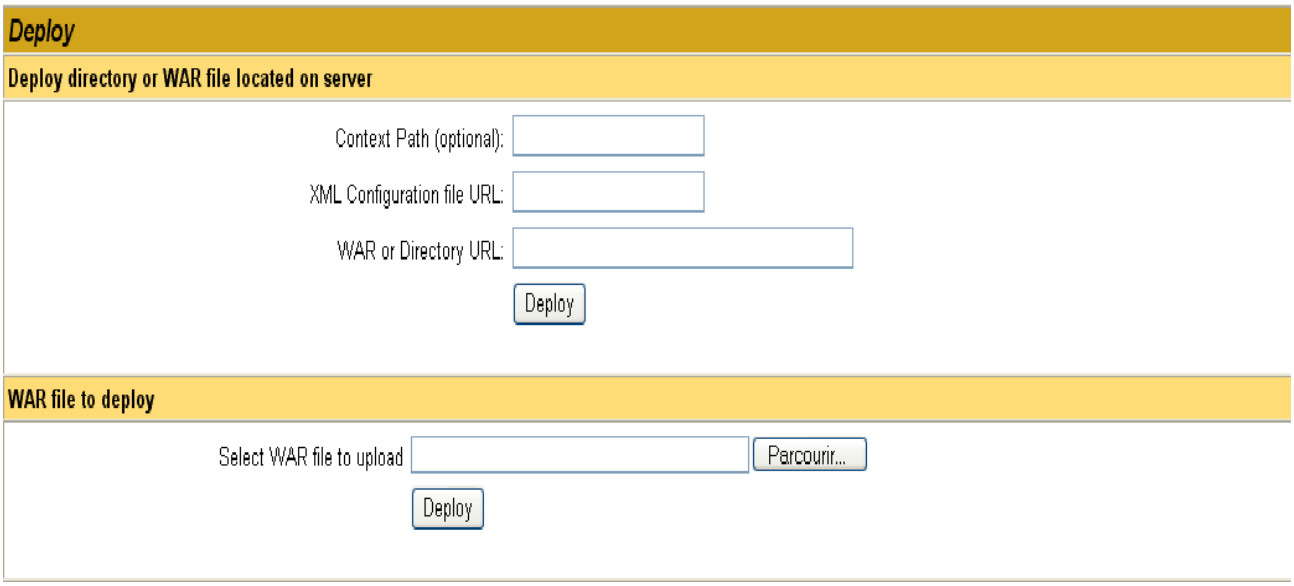
Pour donc attribuer un nouvel utilisateur et ajouter nos applications web, nous ajoutons l'utilisateur «admin» avec comme password «astral».



Désormais nous sommes à l'intérieur du serveur. Nous pouvons voir les applications web déjà présentes.



Pour déployer nos différentes applications, il suffit simplement de de modifier l'extension de l'application en .war.



8)INCREMENT 2

Création et gestion de la base de donnée

BUT

Le but de cet incrément et de permettre à la machine de supervision de donner différentes statistiques:

- le nombre de pellicules triées
- la cadence journalière
- le temps d'utilisation de la machine

Toutes ces statistiques seront visibles à partir de la page HTML du serveur web Tomcat venant de la base de données MySql.

PRINCIPE

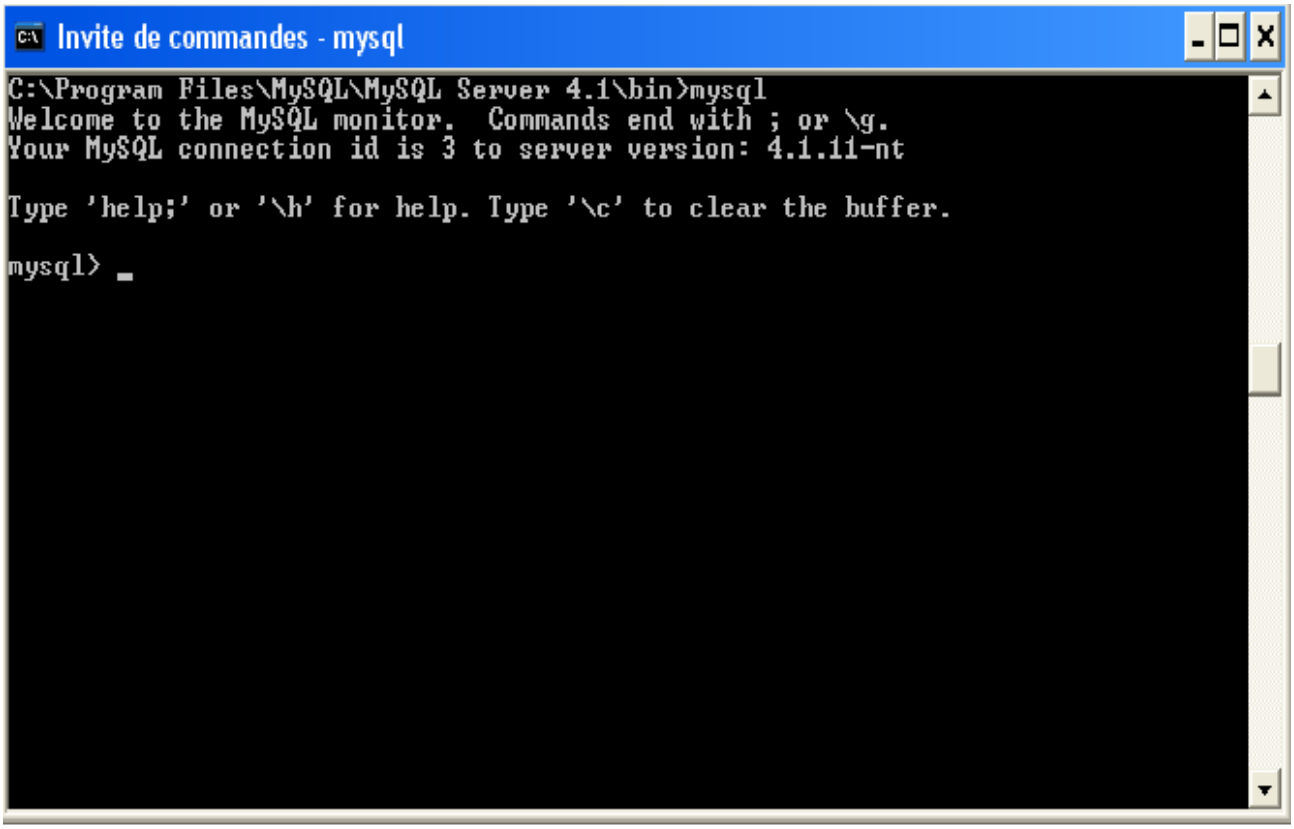
1)*Installation et configuration du serveur MySql*

L'installation de MySql s'effectue en plusieurs étapes:

- *téléchargement de l'installation « mysql-essential-4.1.11-win32.msi »sur le site http://www.mysql.com/*
- *installation de MySql sous le c:*

Le serveur MySql doit désormais être configuré de tel façon à être exploitable.

On peut désormais lancer l'exécutable et travailler grâce à l'interpréteur de commande.



Il faut maintenant voir les différentes statistiques que nous voulons réaliser.

Il va falloir compter par type de pellicules(100 ASA,200 ASA,400 ASA), le nombre de pellicules triées.

De plus, il est nécessaire d'indiquer la cadence de la machine c'est à dire le nombre de pellicules triées par jour ainsi que l'heure où elles sont triées.

2)Création et utilisation de la base de donnée

Il faut donc créer une base sous MySql. Nous allons donner un nom à la base de donnée:*festo.*

Ensuite, la base de donnée contiendra une table appellé *donnee.*

Enfin, notre base de donnée utilisera différentes champs: *type, nombre, heure.*

- Création de la base *festo*

==>create database festo;

Nous avons créer la base de donnée *festo*.

La création de la base de donnée ne la sélectionne pas pour l'utilisation.Il faut le faire explicitement en rendant festo dans la base courante, on utilise cette commande:
==>use festo;

- Création de la table *donnee*

==>create table donnee(type int,nombre int,heure int);

- Remplir le champ *type*

==>insert into type values(100);

insert into type values(200);

insert into type values(400);

Ainsi le champ *type* possede 3 valeurs représentant les 3 types de pellicules.

```
mysql> select type from donnee;
+------+
| type |
+------+
|  100 |
|  200 |
|  400 |
+------+
3 rows in set (0.20 sec)
```

- Visualisation de l'état actuel de la table

==>select *from donnee;

```
mysql> select *from donnee;
+------+--------+-------+
| type | nombre | heure |
+------+--------+-------+
|  100 |   NULL |  NULL |
|  200 |   NULL |  NULL |
|  400 |   NULL |  NULL |
+------+--------+-------+
3 rows in set (0.00 sec)

mysql>
```

Les 2 autres champs, *nombre* et *heure*, seront modifiés suivant une actualisation.

Les valeurs qui seront contenues dans la table *nombre* seront incrémentées. Il va falloir faire correspondre l'affichage d'un capteur avec l'incrémentation des valeurs dans la table *nombre.*

La table *heure* va contenir l'heure à laquelle la pellicule sera contenu dans la goulotte puis être actualiser à chaque nouvelle pellicule.

9)INCREMENT 3

Intégration avec étudiant 2:

récupération et affichage des données

BUT

L'objectif de cet incrément est d'afficher, sur l'IHM de supervision dynamique, les statistiques (nombre de pellicules triées, cadence journalière...).

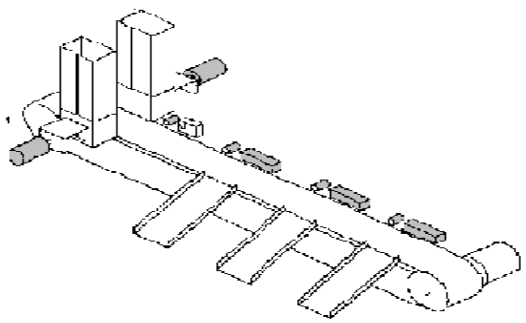
PRINCIPE

1)*Affichage de l'applet*

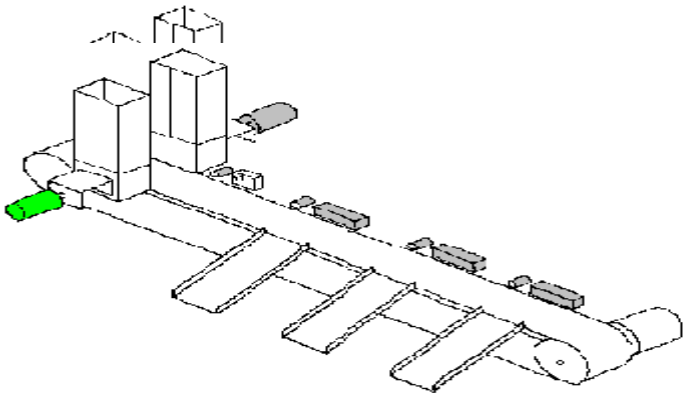
Pour l'affichage de l'avancement du système, nous allons utiliser le principe des applets.

En effet la création de l'applet Java, va permettre l'affichage des images correspondant au système. La page sera raffraichit permettant de montrer le déroulement du système.

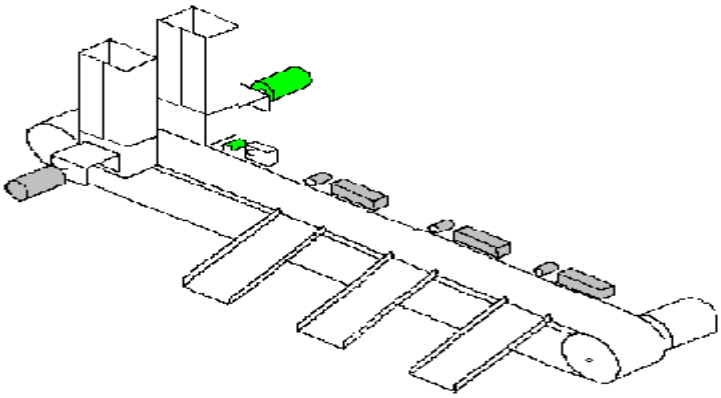
Présentation des images utilisées:



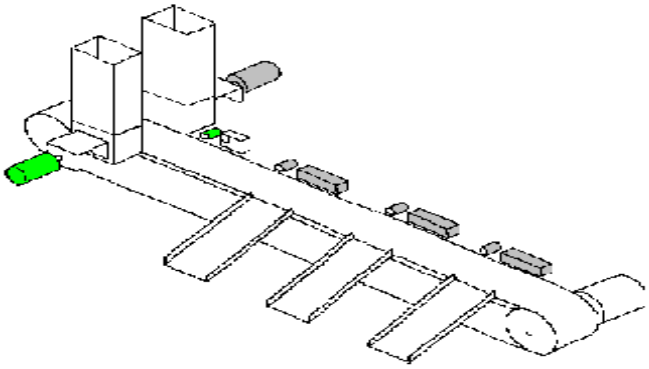
système à l'etat initial



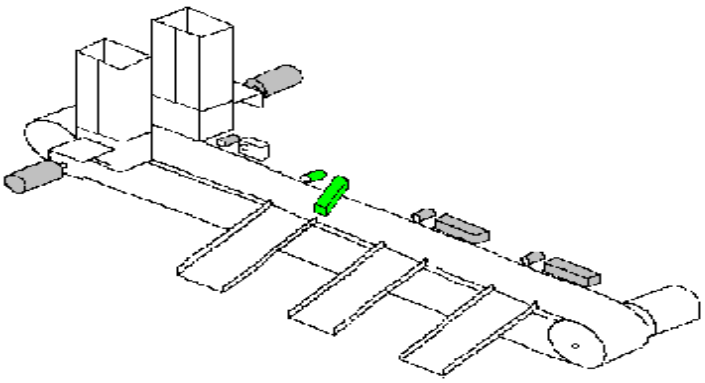
pousseur A



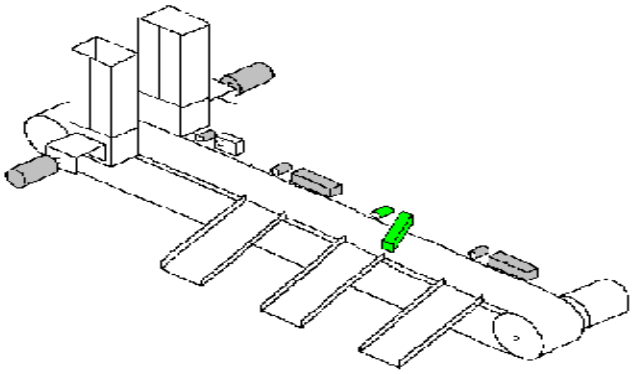
pousseur B +
code barre



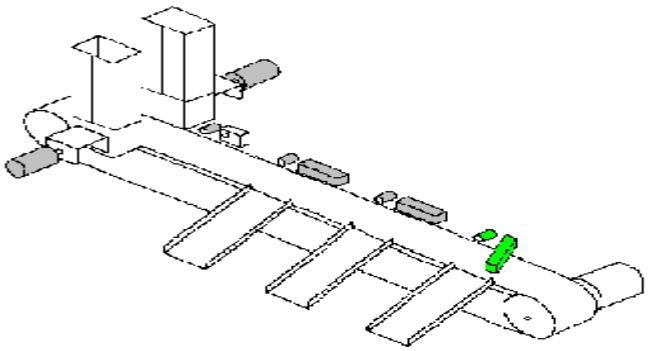
pousseur A
+ code
barre



aiguilleur
100 ASA



aiguilleur
200 ASA



aiguilleur
400 ASA

Création de l'applet:

L'applet contiendra les images actualisées suivant le déroulement des évènements.

Les applets se présentent comme des images sur une page Web. Les images sot, toutfois, interractives.

Dans le cas des applets, le serveur Tomcat va permettre le déploiement d'application Web. Contrairement aux Servlets et aux JSP, l'interprétation du code de l'applet n'est pas analysé par le serveur, mais par la machine virtuelle Java intégrée par le Plug-in.

Codage de l'applet:

Il faut créer une classe principale Test contenant le « main » qui va créer la fenêtre de l'applet.

```
==>public class Test extends JApplet{
     public static void main(){
     new Test().show();
        }
    public Test(){
    getContentPane().add(new fenetre());          //creation de la fenêtre de l'applet
        }
        }
class fenetre extends Jpanel{                     //creation de la methode permettant de
                                                  //dimensionner la fenêtre

public void paintComponent(Graphics surface){
        surface.draw0val(150,50,400,150);              //dimension de la fenêtre de
                                                       //l'applet
}
}
```

Dans la page HTML, il va falloir désormais appeler la classe Test qui créer la fenêtre de l'applet.

```
<APPLET CODEBASE="../prog" CODE="Test.class" ALIGN=BOTTOM WIDTH=500
HEIGHT=500>
</APPLET>
```

Une fois crée ces balises, nous avons la visualisation de la fenêtre de l'applet.

2)*Récupération des données*

Une fois avoir afficher les images correspondant aux états des actionneurs, il va falloir récupérer les informations sur l'IHM pour les statistiques. Les différents champs établit dans la table vont donc être remplit pour la supervision.

Le JSP, introduit dans la page HTML, va rendre le contenu dynamique.

Codage de la récupération:

Pour pouvoir récupérer les données, il faut se connecter sur la base de donnée MySql car c'est elle qui contient les valeurs que l'on va afficher et aussi écrire à l'intérieur de celle-çi.

```
<%@page language="java" import="java.sql.*" %>          //balise JSP permettant
                                                        //l'appel de la librairie MySql

<jsp:usebean id="festo1" scope="page" class="festo"/>

<%ResultSet RS=festo1.executeQuery("select *from donnee;");
                                                        //instruction MySql permettant
                                                        //l'affichage des données contenu
                                                        //dans la base de donnée

while (RS.next())
{                                                       //affichage sur la page HTML des
                                                        //contenues des champs de la
                                                        //table « donnee »

out.print("<LI>"+RS.getString("type")+"</LI>");
out.print("<LI>"+RS.getString("nombre")+"</LI>");
out.print("<LI>"+RS.getString("heure")+"</LI>");
}
RS.close();
%>
```

3)*Raffraîssent de la page*

Après avoir réussi toutes les implentations des images et de la base de donnée, il va falloir raffrâchir continuellement la page.

Pour cela, il suffit d'ajouter dans la balise HTML <head> la fonction « refresh » et de configurer le délai.

Codage du raffaichissement:

```
<head><META HTTP-EQUIV="REFRESH" CONTENT="2";    //durée du raffrichissement 2s
URL="T:\127.0.0.1:8080/essai.html">               //lien avec l'@ ou se situe la
                                                  //page
</head>
```

10)Conclusion

• *Tâches accomplies*

Le but de ma partie était d'effectuer une Interface Homme-Machine(IHM) permettant la visualisation du système sur un poste distant grâce à un serveur Web.

Pour cela nous avions un délai de 5 mois(de Janvier à Mai).

Il a fallut que j'utilise mes connaissances concernant le réseau sur le fonctionnement Client/Serveur et mes quelques acquis sur les bases de donnée. Par contre, j'ai dû apprendre de nouveax outils matériels et logiciels comme l'utilisation du serveur Tomcat, le fonctionnement des technologies Java notamment le JSP(Java Server Pages).

J'ai donc réussi à créer ma page HTML contenant mon applet Java qui va afficher les images correspondant à l'état du système. Une simulation a permi de valider cet incrément. Ensuite, la création de la base de donnée a été réalisé sous soucis particulier. J'ai également pû entrer les valeurs dans les champs de la table.

- *Ce qu'il resta à faire*

En dépit de bonnes volontés, l'intégration avec l'étudiant 2 n'a pas été réalisé au moment de la rédaction du rapport.