La Providence

146 Boulevard de Saint-Quentin

80090 AMIENS

**Compte-Rendu d’analyse**

Tri Pharmacie

BTS Systèmes Numériques

20/05/2022

BERNARD Léa (Etudiant 3)

BERTHIER Thomas (Etudiant 1)

LARIDANT Julien (Etudiant 2)

Table des matières

[Présentation du sujet 3](#_Toc103067817)

[Objectif du projet 3](#_Toc103067818)

[Principe du projet 3](#_Toc103067819)

[Diagramme de cas d’utilisation simplifié 5](#_Toc103067820)

[Diagramme d’exigence 6](#_Toc103067821)

[Diagramme de classe simplifié 11](#_Toc103067822)

[GANTT 12](#_Toc103067823)

[Répartition des tâches 13](#_Toc103067824)

[CRA 14](#_Toc103067825)

[Git Hub et Versionning 15](#_Toc103067826)

[Démarrage projet et classe de simulation 16](#_Toc103067827)

[Logiciel d’analyse et de développement 17](#_Toc103067828)

[Maquettage et Prototype 19](#_Toc103067829)

[Choix des composants 20](#_Toc103067830)

[Etude physique 21](#_Toc103067831)

[Protocole expérimentale 21](#_Toc103067832)

[Etude du signal 22](#_Toc103067833)

[Partie individuelle Thomas Berthier 24](#_Toc103067834)

[Fonctionnalités Activer le convoyeur et Désactiver le convoyeur 24](#_Toc103067835)

[Fonctionnalités Activer les vérins et désactiver les vérins 28](#_Toc103067836)

[Fonctionnalités Récupérer les données des capteurs et Afficher l’état actuel du système 33](#_Toc103067837)

[Fonctionnalité interpréter les messages modbus et Arduino 39](#_Toc103067838)

[Fonctionnalité Convoyer les médicaments 42](#_Toc103067839)

[Recette 47](#_Toc103067840)

[Tests d’intégration du prototype 48](#_Toc103067841)

[Avancement et Conclusion 49](#_Toc103067842)

# Présentation du sujet

Une pharmacie souhaite délocaliser son stock de médicament en sous-sol et automatiser l’acheminement des médicaments jusqu’au caisses afin de pouvoir agrandir sa surface de vente. Ce stockage en sous-sol va éviter d’agrandir la surface d’exposition et donc éviter des coûts supplémentaires. Ces changements vont également apporter eux avantages supplémentaires. Tout d’abord le pharmacien n’aura plus besoin d’aller chercher les médicaments ce qui va permettre d’améliorer le contact avec le client. De plus, le risque de vol s’en verra diminuer car le pharmacien sera présent en permanence.

# Objectif du projet

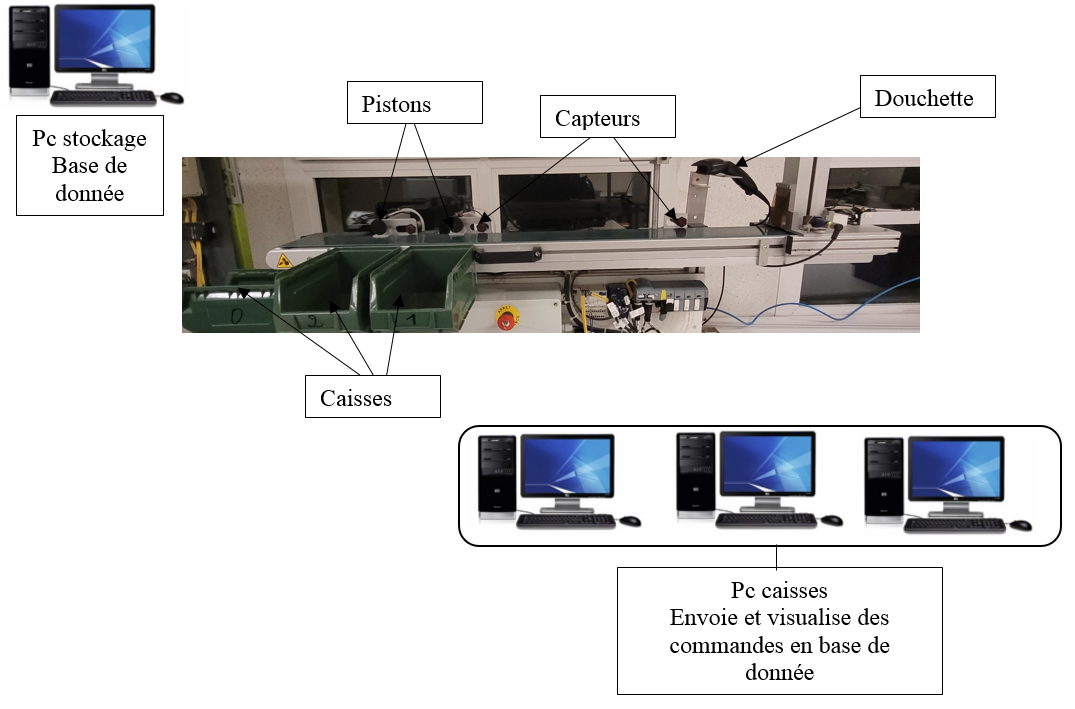
Le but du projet est donc de réaliser l’automatisation de l’acheminement des médicaments vers les caisses.

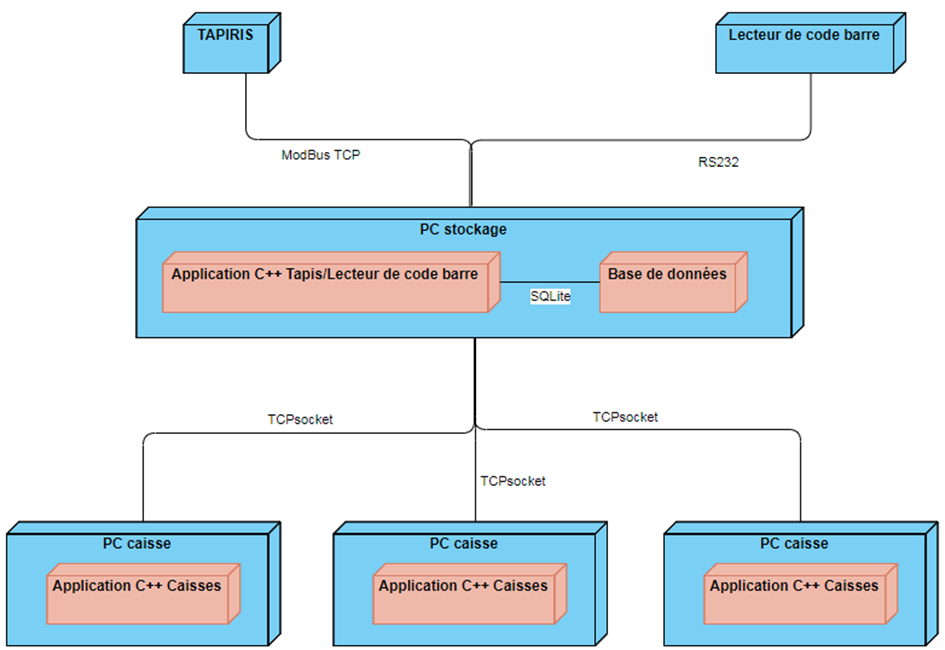
# Principe du projet

Afin de réaliser l’automatisation de l’acheminement des médicaments vers les caisses, trois parties vont être traitées.

* Automatiser le tapis TAPIRIS pour permettre l’acheminement des médicaments vers les caisses
* Gérer la douchette code-barres pour permettre l’aiguillage des médicaments sur le tapis
* Créer et gérer une base de données pour permettre de lier les médicaments à des commandes ainsi que différencier les différentes commandes et leurs destinations sur le tapis

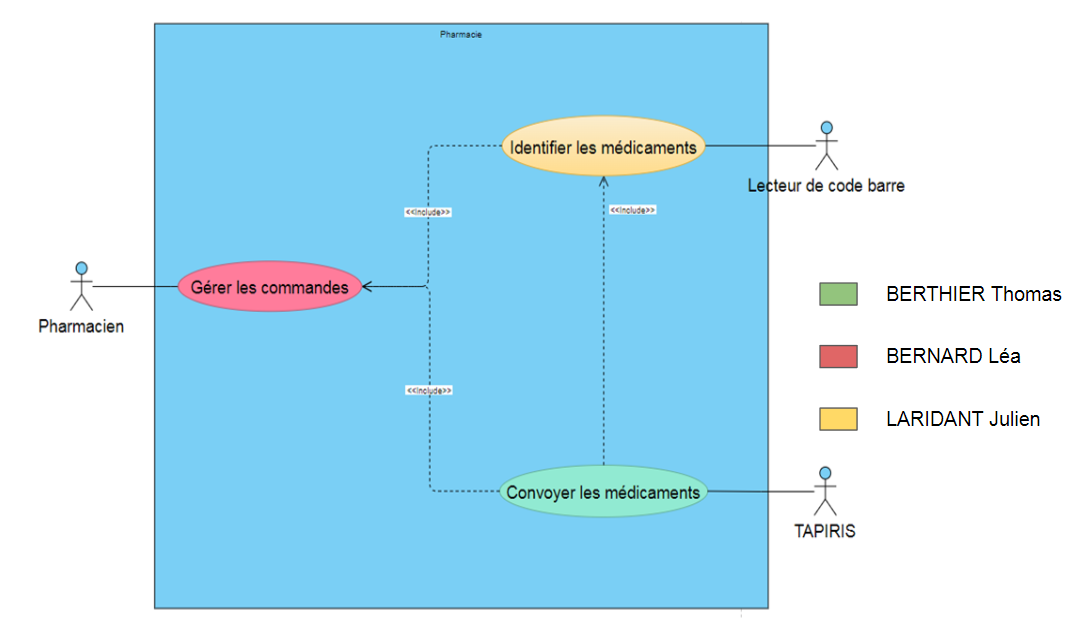
Afin de permettre cette réalisation, le système sera installé de la manière suivante :





Chaque caisse possèdera une application pour enregistrer et visualiser des commandes. Ces commandes seront envoyées au PC Stockage pour être enregistrer en base. Une autre application sera présente sur le PC Stockage qui permettra de gérer le tapis TAPIRIS ainsi que le lecteur de code-barres.

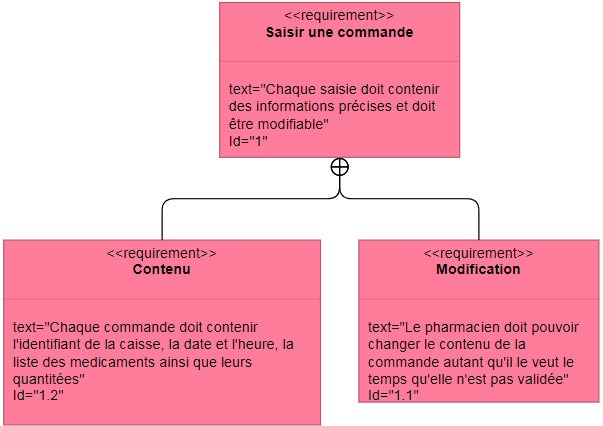
# Diagramme de cas d’utilisation simplifié

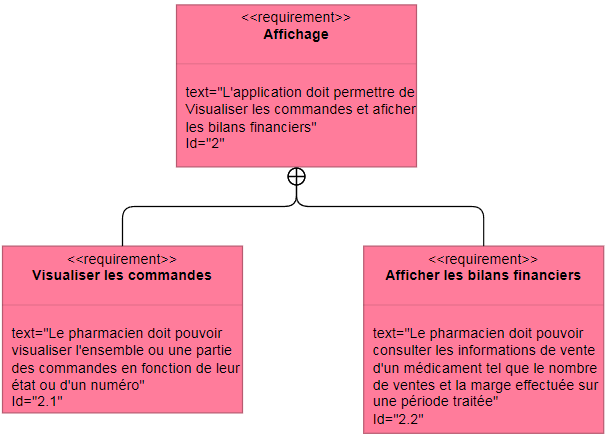


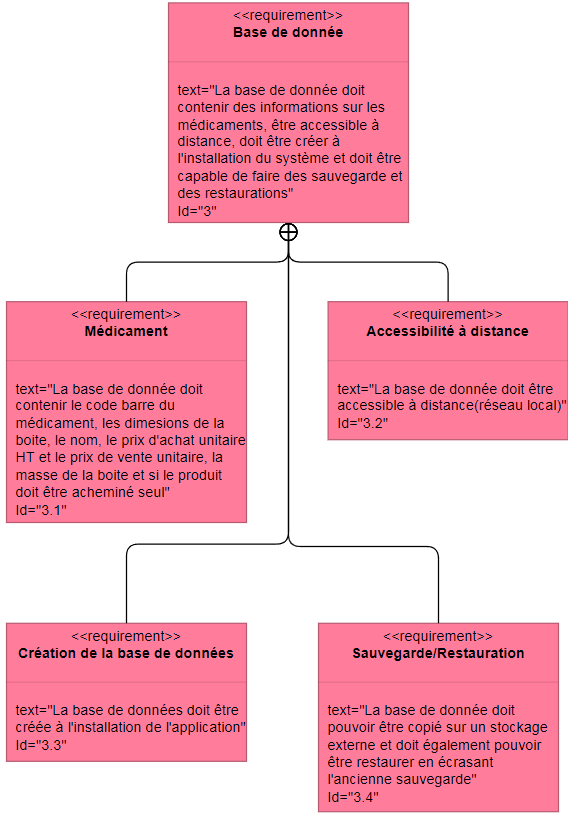
Un pharmacien aura ainsi possibilité de gérer les commandes. Pour ce faire, il sera nécessaire que le lecteur de code-barres identifie les médicaments et que le tapis TAPIRIS les convoie.

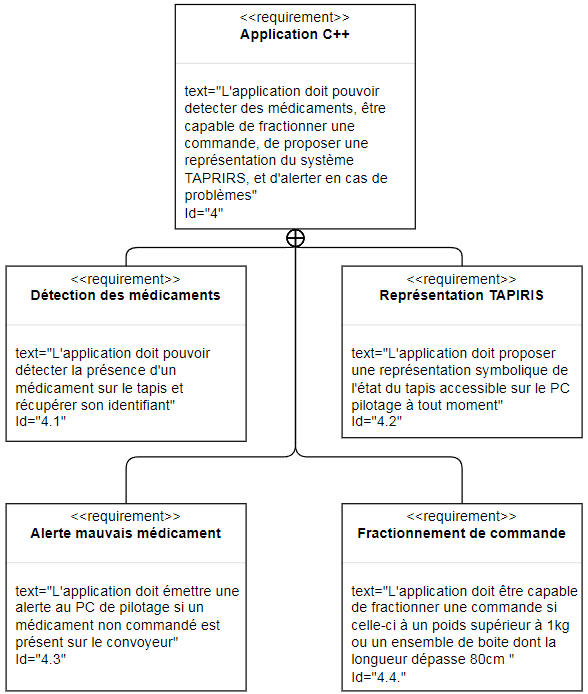
# Diagramme d’exigence

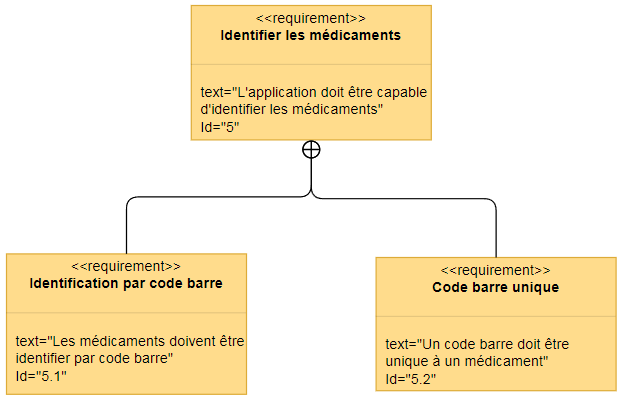


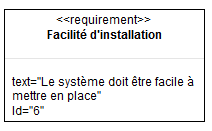




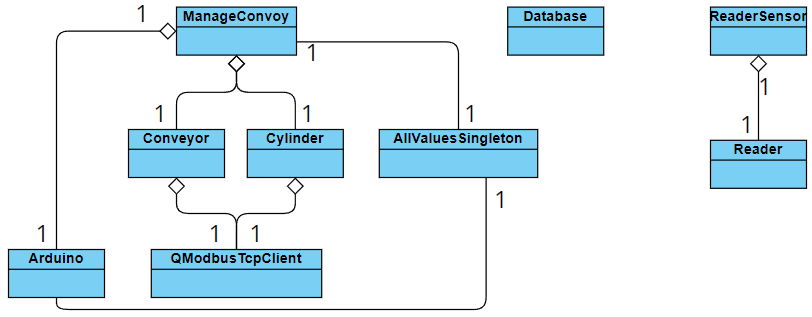








# Diagramme de classe simplifié



La classe QModBusTcpClient permet la communication avec l’automate ETZ512.

La classe Arduino permet la communication avec la carte arduino et ses capteurs.

La classe Conveyor permet de gérer le convoyeur et agrège la classe QModBusTcpClient.

La classe Cylinder permet de gérer les vérins et agrège la classe QModBudTcpClient.

La classe ManageConvoy permet de gérer l’ensemble des systèmes nécessaires pour faire correctement fonctionner le tapis TAPIRIS. Elle agrège les classes Arduino, Conveyor et Cylinder

La classe AllValuesSingleton permet d’enregistrer toutes les dernières données des capteurs. Elle est associée au classes ManageConvoy et Arduino.

La classe Database permet

La classe ReaderSensor permet

La classe Reader permet

https://lh5.googleusercontent.com/wy4U1NaQvEiOni3Px-plj_S3QLvhT7HnZVvAQK3yHFkmpS4L2on-WXTJ_CCB1EKLBF-JWcPVBoOgZnatnIHiBCwNEksJ9B5cH1r0Qejyq_P6c1skRwnCAWkdB6rhEM23qmVPAKoFjwgo

# GANTT

# Répartition des tâches

La répartition des tâches, imposées par le sujet de BTS, est donnée comme tel :

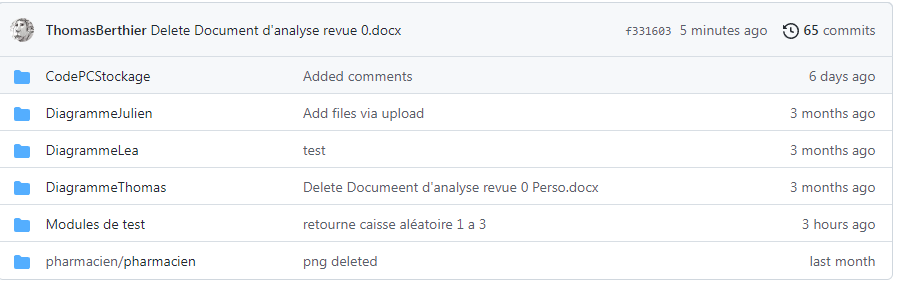
Etudiant 1 (BERTHIER Thomas) :

* Effectuer la connexion via le réseau sur l’automate
* Permettre d’effectuer les différents paramétrages
* Permettre de faire l’association entre le médicament et sa destination (caisse)
* Interface permettant d’associer une caisse à une localisation sur tapis
* A la demande, afficher au moment de la demande l’état de TAPIRIS, des capteurs et actionneurs
* Créer l’IHM du PC Stock/Pilotage qui permettra de visualiser et mettre à jour les commandes

# CRA

# Git Hub et Versionning

Pour travailler facilement en groupe, nous avons utilisé le logiciel de versionning Git ainsi que sa plateforme d’hébergement Github.com. Ce logiciel nous a permis de partager ainsi que récupérer facilement les dernières versions du code de chacun. Ce logiciel à également servi de back-up en cas de problèmes avec la dernière version.

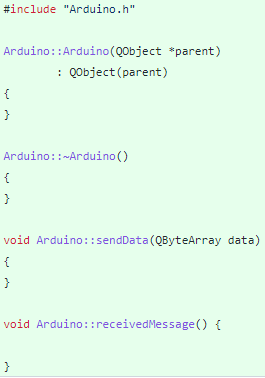
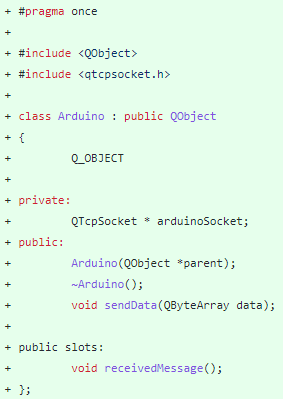


# Démarrage projet et classe de simulation

Nous avons tout d’abord commencé le projet en créant chaque classe ainsi que les prototypes de leurs méthodes. Les méthodes ont ensuite été simulé si besoin puis ont été implémentées au fur et à mesure de l’avancement du projet.

Exemple de classes avec des méthodes non implémentées :

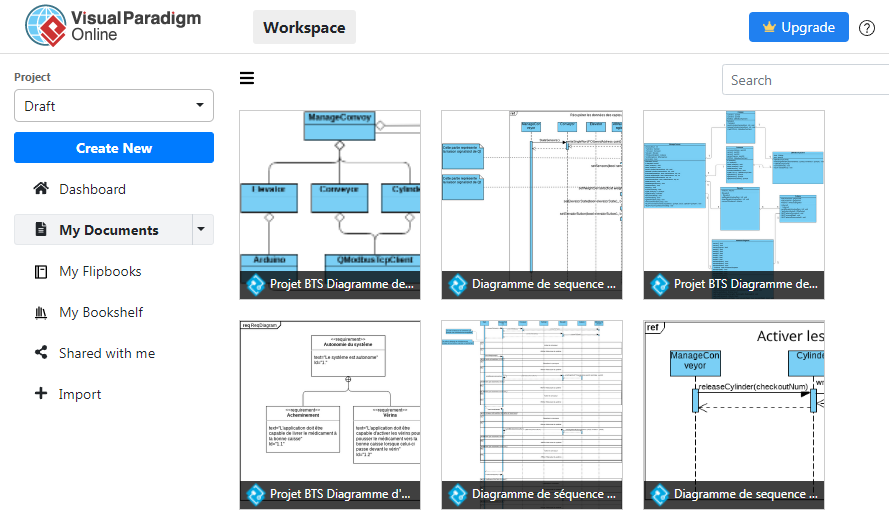
Arduino.cpp Arduino.h



# Logiciel d’analyse et de développement

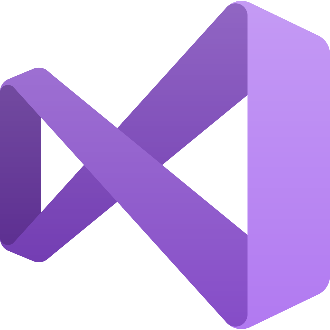
Au cours du projet, nous avons utilisé plusieurs logiciels afin d’effectuer l’analyse, le développement ou pour partager des fichiers communs.

Pour réaliser nos analyses, nous avons utilisé l’outil en ligne Visual Paradigm qui permet de créer des diagrammes et de les retrouver sauvegardés et donc disponible de n’importe quels ordinateurs.



Nous avons également utilisé un Template Excel pour réaliser le diagramme de GANTT (cf. p10)

Pour le développement, nous avons utilisé logiciel Visual Studio Community ainsi que le FrameWork Qt qui offre de nombreuses possibilités de manières grandement simplifiées.

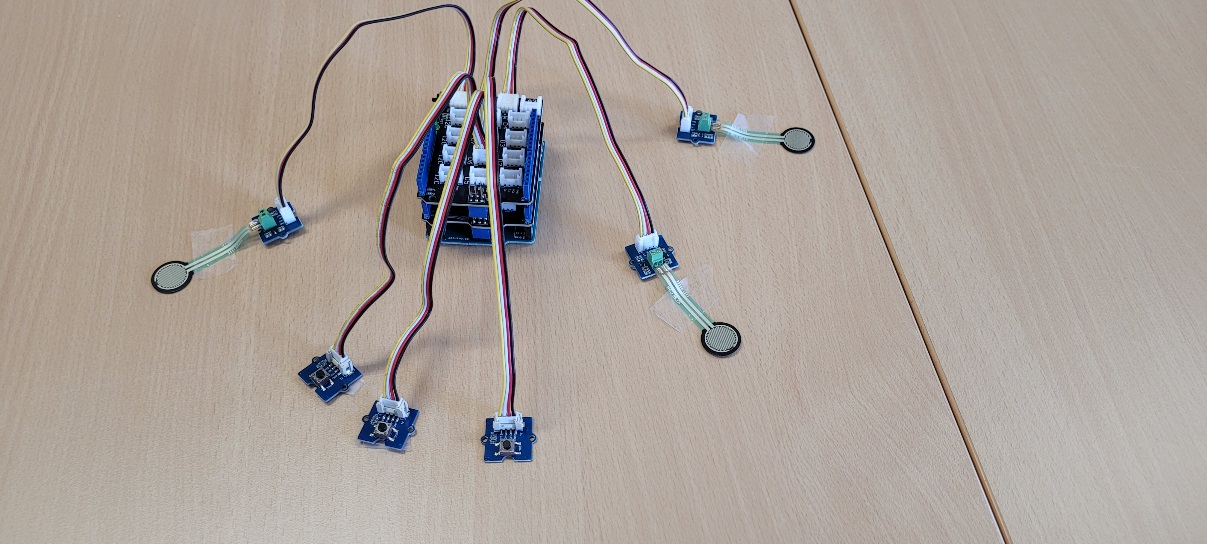


Pour la partie partage et stockage, nous avons utilisé l’outil en ligne Google Drive pour permettre un stockage de fichiers en ligne et accessible par tous les membres du projet.



# Maquettage et Prototype

Pour le prototype, nous avons utilisé une carte Arduino Uno. Nous y avons ensuite attaché un shield Ethernet afin de pouvoir communiquer par le réseau. Finalement nous avons ajouté un shield Grove afin de brancher des capteurs de poids, des boutons Grove et des LED. Les capteurs de poids permettront de vérifier que le poids ne dépasse pas les exigences, les LED vont indiquer l’état des ascenseurs (si la LED est allumée l’ascenseur est parti, si la LED est éteinte l’ascenseur est revenu) et les boutons vont servir à indiquer au système que les ascenseurs sont redescendus. Nous avons choisi des câbles Grove pour éviter une breadboard ce qui facilitera l’intégration du système au TAPIRIS et donnera un résultat plus propre et présentable.



# Choix des composants

|  |  |  |  |
| --- | --- | --- | --- |
| Composant | Caractéristique | Nom et quantité | Prix |
| XUB1APANM12 | Détecteur photo-électrique polarisé.  Sortie numérique | 3 Capteur photoélectrique XUB1APANM12 | 45,95€ unité  Ce capteur n’a pas eu besoin d’être acheté car il était déjà présent dans le système. |
| Capteur de force Grove 101020553 | 0-2kg  Adaptabilité Grove | 3 Capteur de force Grove 101020553 | 15,65€ unité |
| 101020003- Carte de développement. Seeed Studio, Bouton | RS Components | Adaptabilité Grove | 3 Boutons poussoir 101020003 | 2,14€ unité |
| Coupleur web TSX ETZ 410 | Mode client/serveur pour Modbus.  Module Ethernet qui permet la liaison entre le réseau (Ethernet) et l’automate (Liaison en série par modem) | Modicon TSX Micro – TSXETZ410 | 940€  Ce module Ethernet n’a pas eu besoin d’être acheté car il était déjà présent dans le système. |
| TSX3721001 Telemecanique TSX Micro 37 21/22 PLC configurations - Santa  Clara Systems | Automate qui reçoit les trame du TSXETZ410, les interprète et envoie les instructions au modBus ASI | Modicon TSX Micro – Base TSX micro 3721 | 720€  Cet automate n’a pas eu besoin d’être acheté car il était déjà présent dans le système. |
| TSXSAZ10 | Compteur Schneider Electric, pour Modicon TSX Micro PLC | RS  Components | Coupleur maître qui reçoit les trames de l’automate et les renvoi au esclaves du bus | Modicon TSX Micro – TSXSAZ10 | 685€  Ce Coupleur maître n’a pas eu besoin d’être acheté car il était déjà présent dans le système. |
| P1A S010DS 0100 Mini ISO6432 Cylindre Double Agissant Calibre 10mm coup 100mm | -20-80 °C  0-10 bar | 2 Cylindre P1A S010DS 0100 pour pousser les médicaments dans les caisses | 120,95€ unité  Ce cylindre n’a pas eu besoin d’être acheté car il était déjà présent dans le système. |

# Etude physique

Le but de notre étude et d’enregistrer un signal du scanner afin de pouvoir le déchiffrer. Nous avons donc réalisé un protocole expérimental afin de mettre en œuvre notre étude.

## Protocole expérimentale

Pour enregistrer un signal nous aurons donc besoin d’un picoscope afin de récupérer le signal, d’un scanner pour créer et envoyer le signal, d’un câble RS232 mâle afin de transmettre le signal ainsi qu’un ordinateur possédant l’application PicoScope 6 afin de pouvoir modéliser le signal.

Pour mener à bien cette expérimentation, il faut tout d’abord brancher le câble RS232 au scanner et coupé l’autre côté afin de rendre visible les différents fils du câble. Il faut ensuite dénuder les fils et mettre sous tension la douchette. La prochaine étape et de relier le picoscope au scanner et de paramétrer certains critères sur PicoScope 6. Pour relier le scanner il suffit d’attraper le fil de tension (fil rouge) du câble RS232 avec la pince branchée sur la sortie A du picoscope. Il ne reste plus qu’à paramétrer la sensibilité (tension), le balayage(seconde) et le point de déclenchement ainsi que le déclenchement seul.

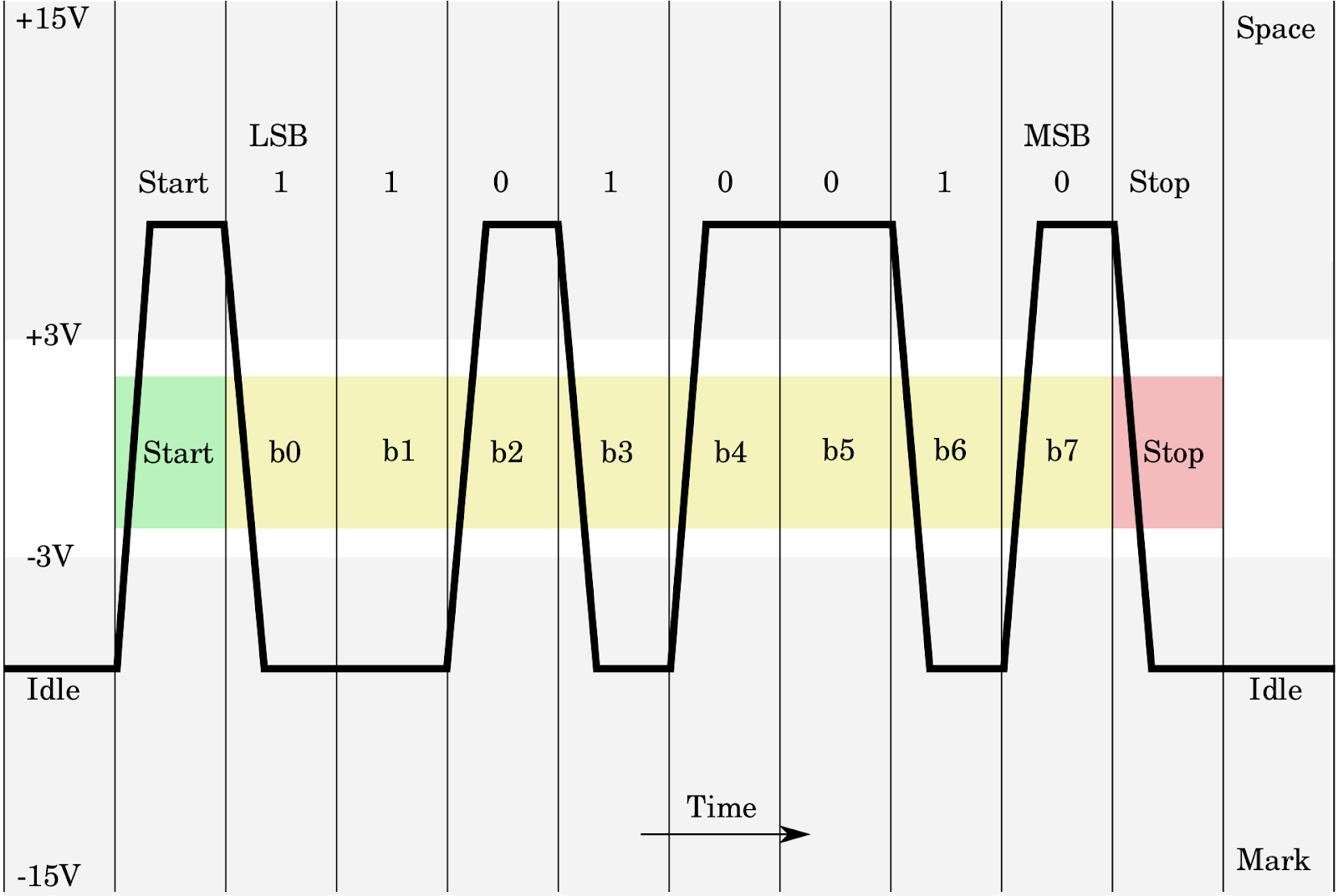
Une fois tous ces critères accomplis, il ne reste plus qu’à prendre la mesure du signal et de stopper le défilement de la courbe afin de pouvoir observer le signal.

Voici le résultat obtenu lors de notre protocole :



## Etude du signal

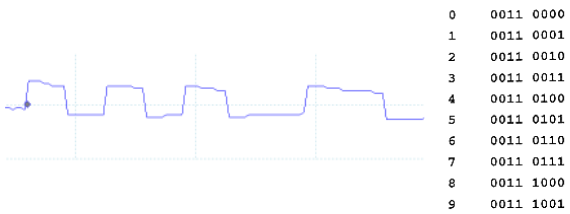
Afin de déchiffrer ce signal, plusieurs étapes sont nécessaires. La première est de déchiffrer le signal RS232. Voici comment fonctionne ces signaux :



Nous pouvons remarquer plusieurs indication grâce à cette image. Tout d’abord, le signal est composé d’un bit de start de 8 bits de données et d’un bit de stop. Les huit bits de données se lisent du LSB au MSB c’est-à-dire du plus petit bit (bit de poids faible) au plus grand bit (bit de poids fort) ce qui veut dire que les données sont envoyées en partant de la fin de la donnée. Finalement on remarque que les bits à 1 sont à l’état bas et ceux à 0 à l’état haut ce qui veut dire que le bit de start est un 0 et que le bit de stop est un 1.

En regardant la documentation du signal on apprend également que les données sont envoyées en ASCII.

Il nous est désormais possible de déchiffrer le signal entier. Essayons avec le premier caractère :



Le signal commence par un bit de stop puis est suivi par un état bas, un haut, un autre bas et un autre haut. Le signal affiche ensuite un état bas plus long ce qui signifie qu’il y a deux bit à 0 suivi de deux bits à 1 le signal fini par un état bas et donc le bit de stop.

En reconstituant les 8 bits de données nous obtenons « 10101100 » il nous suffit maintenant de lire ce chiffre depuis la fin afin de remettre le MSB au début et le LSB à la fin.

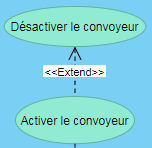
Nous obtenons finalement « 00110101 » qui correspond au code ASCII du chiffre 5 ce qui veut dire que dans ce signal est envoyée le chiffre 5 ce qui correspond au code-barres enregistré par le scanner (voir code-barres ci-dessous).

Les caractères de la partie CRLF représente une fin de transmission qui contient un retour à la ligne.



# Partie individuelle Thomas Berthier

# Fonctionnalités Activer le convoyeur et Désactiver le convoyeur



Scenarii

Activer le convoyeur :

1)Le système prépare une trame composé d’un code d'écriture et la variable du convoyeur à changer.

2)La trame modbus est envoyé à l’automate via TCP.

3)L’automate reçoit la trame et démarre le tapis.

4)L’automate envoi un accusé d'écriture.

Variante 1 : En 3, le convoyeur ne démarre pas car la trame est erronée.

Préconditions :

Qu’il y ait au moins une commande en cours.

Le système vérifie que l’automate est en écoute.

Post-condition :

Le convoyeur est en marche.

Désactiver le convoyeur :

1)Le système prépare une trame composé d’un code d'écriture et la variable du convoyeur à changer.

2)La trame modbus est envoyé à l’automate via TCP.

3)L’automate reçoit la trame et arrête le tapis.

4)L’automate envoi un accusé d'écriture.

Variante 1 : En 3, le convoyeur ne s’arrête pas car la trame est erronée.

Préconditions :

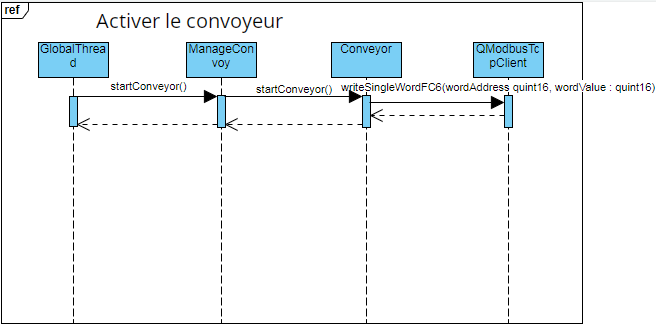
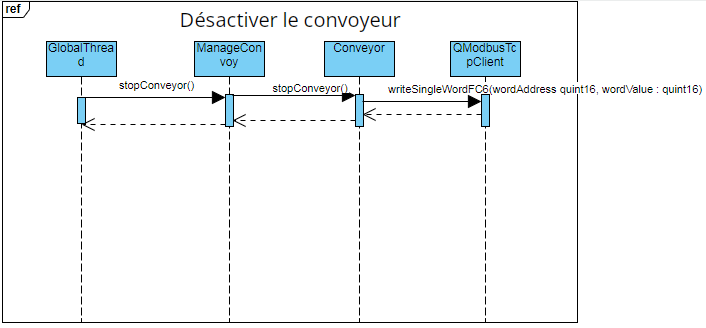
Qu’il n’y ait plus aucune commande en attente et en cours.

Le système vérifie que l’automate est en écoute.

Post-condition :

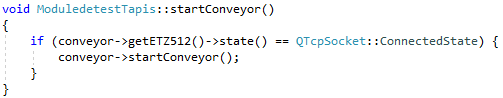
Le convoyeur est à l'arrêt.

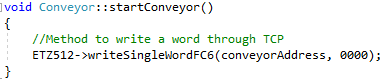
Diagramme de séquence

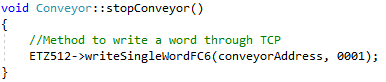
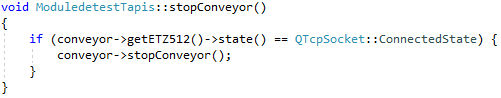
 

Mise en application du diagramme de séquence

Pour effectuer le test lié au diagramme précédents, la classe ManageConvoy a été remplacée par la classe ModuledetestTapis afin de mettre en œuvre des signaux slots et une interface pour démarrer le tapis qui n’est pas présent dans la classe ManageConvoy. De plus, la classe GlobalThread n’a pas été utilisée.

Activer le convoyeur : 

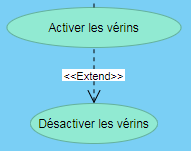


Désactiver le convoyeur : 

Résultat

La fonctionnalité est opérationnelle, le tapis se démarre et s’arrête lorsque l’on appuie sur les boutons correspondant.

# Fonctionnalités Activer les vérins et désactiver les vérins



Scenarii

Activer les vérins :

1)Le système prépare une trame composé d’un code d'écriture et la variable du vérin correspondant à la caisse à changer.

2)La trame modbus est envoyé à l’automate via TCP.

3)L’automate reçoit la trame et active le vérin.

4)L’automate envoi un accusé d'écriture.

Variante 1 : En 3, le vérin ne s’active pas car la trame est erronée.

Préconditions :

Qu’il y ait une commande en cours, que le numéro de caisse soit connu et que le médicament passe devant le capteur lié au vérin.

Le système vérifie que l’automate est en écoute.

Post-condition :

Le vérin est activé.

Désactiver les vérins :

1)Le système prépare une trame composé d’un code d'écriture et la variable du vérin correspondant à la caisse à changer.

2)La trame modbus est envoyé à l’automate via TCP.

3)L’automate reçoit la trame et désactive le vérin.

4)L’automate envoi un accusé d'écriture.

Variante 3 : En 4, le vérin ne se désactive pas car la trame est erronée.

Préconditions :

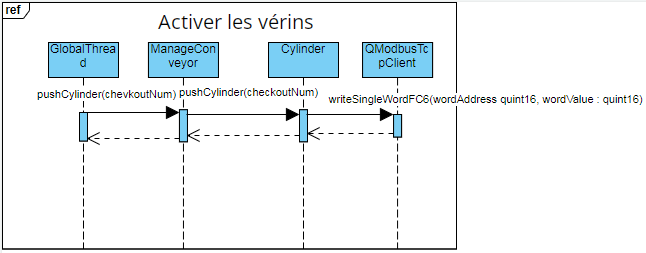
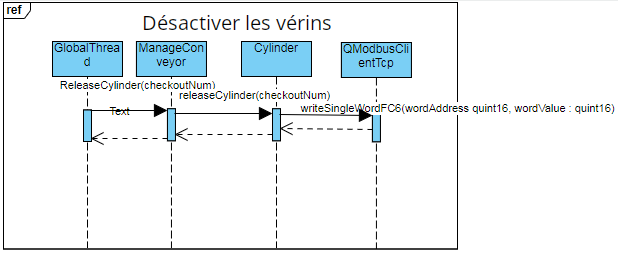
Qu’il y ait une commande en cours, que le numéro de caisse soit connu, que le médicament passe devant le capteur lié au vérin et que le vérin soit activé.

Le système vérifie que l’automate est en écoute.

Post-condition :

Le vérin est désactivé.

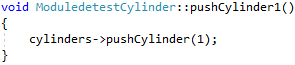
Diagramme de séquence

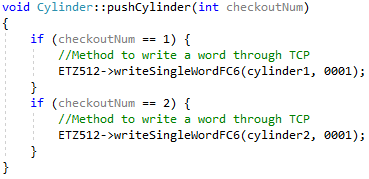
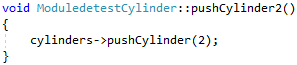
 

Mise en application du diagramme de séquence

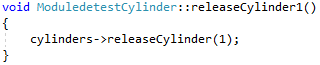
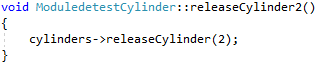
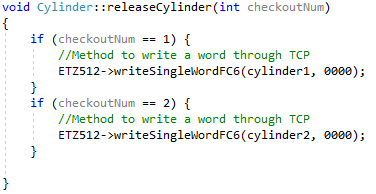
Pour effectuer le test lié au diagramme précédents, la classe ManageConvoy a été remplacée par la classe ModuledetestCylinder afin de mettre en œuvre des signaux slots et une interface pour démarrer gérer les vérins qui n’est pas présent dans la classe ManageConvoy. De plus, la classe GlobalThread n’a pas été utilisée.

Activer les vérins :





Désactiver les vérins :

Résultat

La fonctionnalité est opérationnelle, les vérins s’activent et se désactivent lorsque l’on appuie sur les boutons correspondant.

# Fonctionnalités Récupérer les données des capteurs et Afficher l’état actuel du système





Scenarii

Récupérer les données des capteurs :

1)Le système prépare une trame composée d’un code d'écriture et la variable des capteurs à l’automate. Pour la carte Arduino, les valeurs sont envoyées périodiquement.

2)La trame modbus est envoyé à l’automate via TCP.

3)L’automate reçoit la trame et traite les données.

4)L’automate renvoi les données qui seront récupérées grâce à une méthode Qt.

Variante 1 : En 1, les valeurs de l’arduino ne sont pas reçues car l’arduino n’est pas connectée.

Préconditions :

Qu’il y ait une commande en cours.

Le système vérifie que l’automate est en écoute.

Le système vérifie que l’automate est en écoute.

Post-condition :

Les données sont récupérées et stockées.

Afficher les données actuelles du système :

1)Le système récupère les dernières valeurs des capteurs depuis la classe AllValuesSingleton.

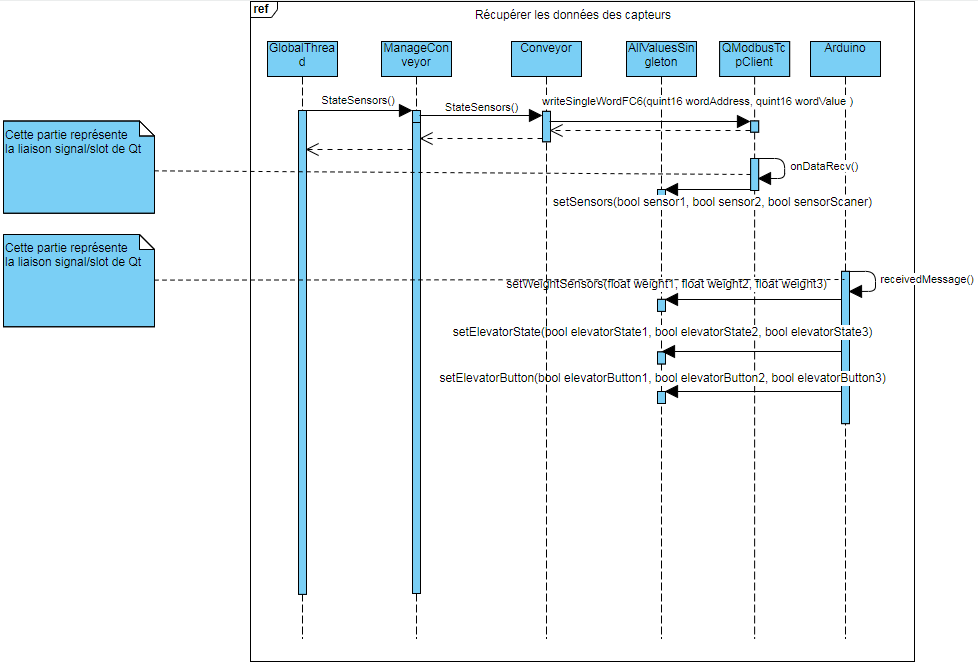
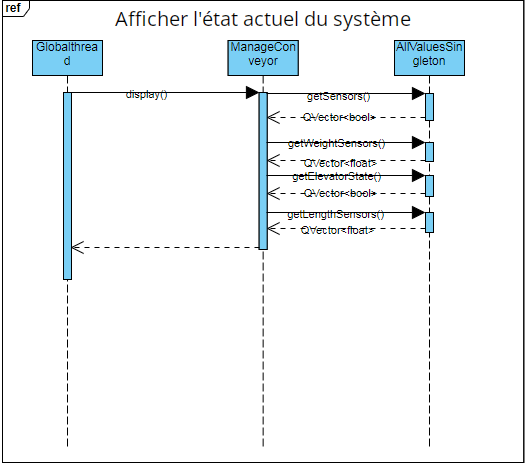
2)Le système affiche les valeurs dans la console.

Variante 1 : En 1, les valeurs des capteurs ne sont pas récupérées donc en 2 les valeurs affichées sont les anciennes valeurs.

Post-condition :

Les données sont affichées.

Diagramme de séquence

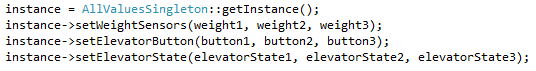
 

Mise en application du diagramme de séquence

Pour effectuer le test lié au diagramme précédents, la classe ManageConvoy a été remplacée par la classe Moduledetestcapteurs afin de mettre en œuvre des signaux slots et une interface pour démarrer gérer les vérins qui n’est pas présent dans la classe ManageConvoy. De plus, la classe GlobalThread n’a pas été utilisée.

Récupérer les données des capteurs :

Partie Arduino :



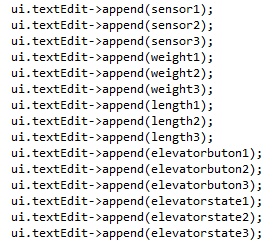
Après avoir été interprété (cf. fonctionnalité interpréter les messages arduino), les valeurs sont ensuite stockées dans la classe AllValuesSingleton grâce au trois mutateurs(setters).

Partie Automate :



On appelle la méhode stateSensors() pour demander l’état des capteurs de l’automate. Les données sont stockées dans la classe AllValuesSingleton grâce à un mutateur.

Afficher l’état actuel du système : 



On récupère toutes les valeurs des capteurs grace aux accesseurs puis on transforme les valeurs en texte afin de pouvoir les afficher. Finalement on ajoute le texte dans la zone de texte de l’UI.

Résultat

La fonctionnalité est opérationnelle, les données des capteurs sont bien récupérées ainsi que stockées et l’état actuel du système est affiché.

# Fonctionnalité interpréter les messages modbus et Arduino





Scenarii :

Interpréter les messages arduino :

1)Le système reçoit les réponses de la carte arduino.

2)Le système interprète le message.

Variante 1 : En 1, les réponses ne sont pas interprétées car la connexion arduino n’est pas effectuée.

Post-condition :

Les réponses sont interprétées.

Interpréter les messages modbus :

1)Le système interprète les réponses de la carte modbus.

2)Le système interprète le message.

Variante 1 : En 1, les réponses ne sont pas interprétées car la connexion modbus n’est pas effectuée.

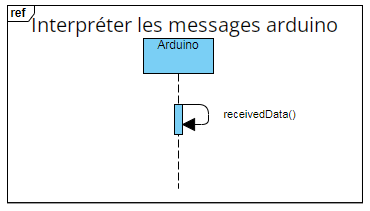
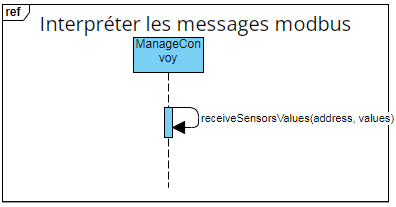
Préconditions :

Qu’un message ait été envoyé à la carte modbus.

Post-condition :

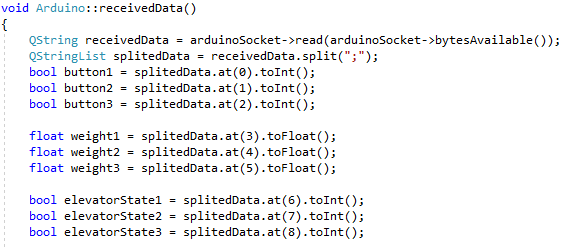
Les réponses sont interprétées.

Diagramme de séquence



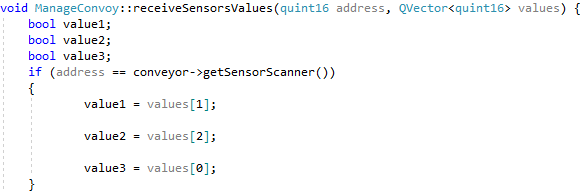
Mise en application du diagramme de séquence

Interpretation arduino :



Lorsque la trame est récupérée, elle est découpé à chaque point virgules. On récupère ensuite chaque partie et l’associe à des variables correspondantes.

Interpretation modbus :



L’automate retourne un tableau de vecteur comprenant les valeurs des capteurs demandés dans l’ordre croissant des adresses des composants. On peut donc directement associer chaque valeur à une variable.

Résultat

La fonctionnalité est opérationnelle, les messages sont interprétés correctement.

# Fonctionnalité Convoyer les médicaments



Cette fonctionnalité represente l’ensemble des fontionnalités précédentes et donc la globalité de ma partie.

Scenarii

Convoyer les médicaments :

1)Le système convoie les médicaments vers les caisses selon les scenariis.

Variante 1: En 1, les médicaments ne sont pas convoyés car un des scenariis précédent a rencontré une variante.

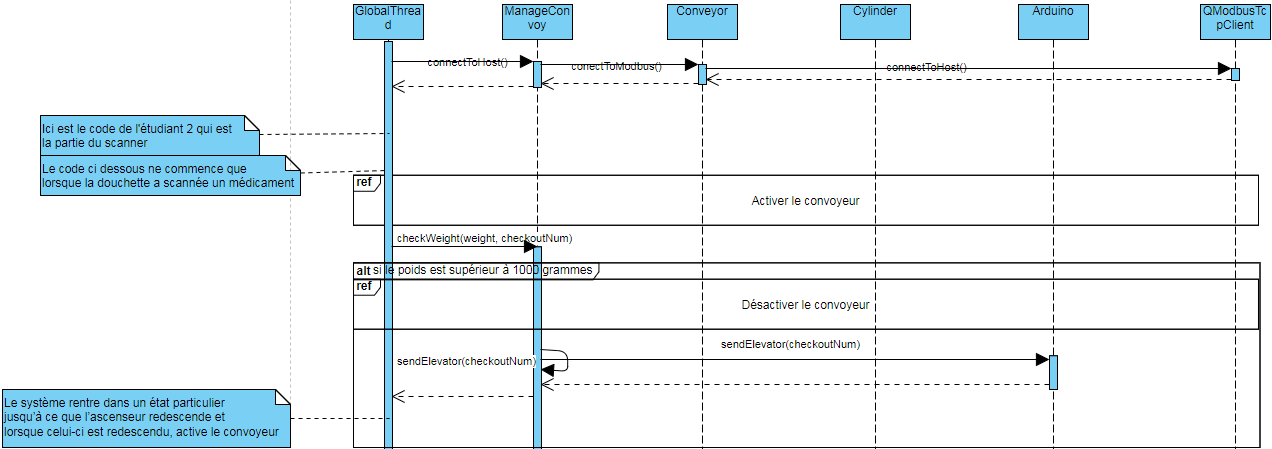
Pré-conditions :

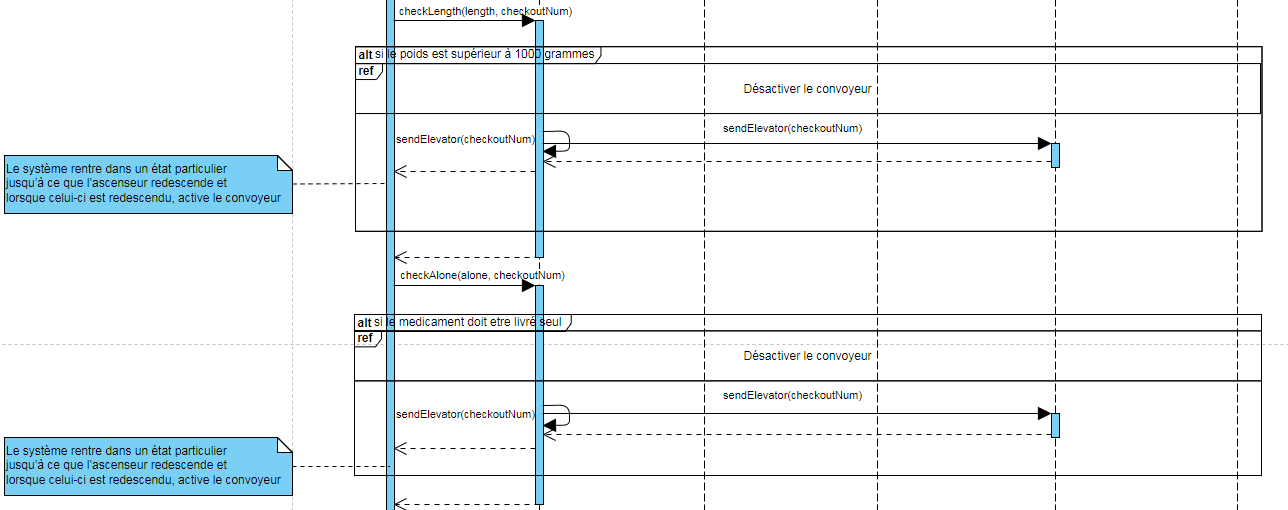
Qu’il y ait une commande en cours.

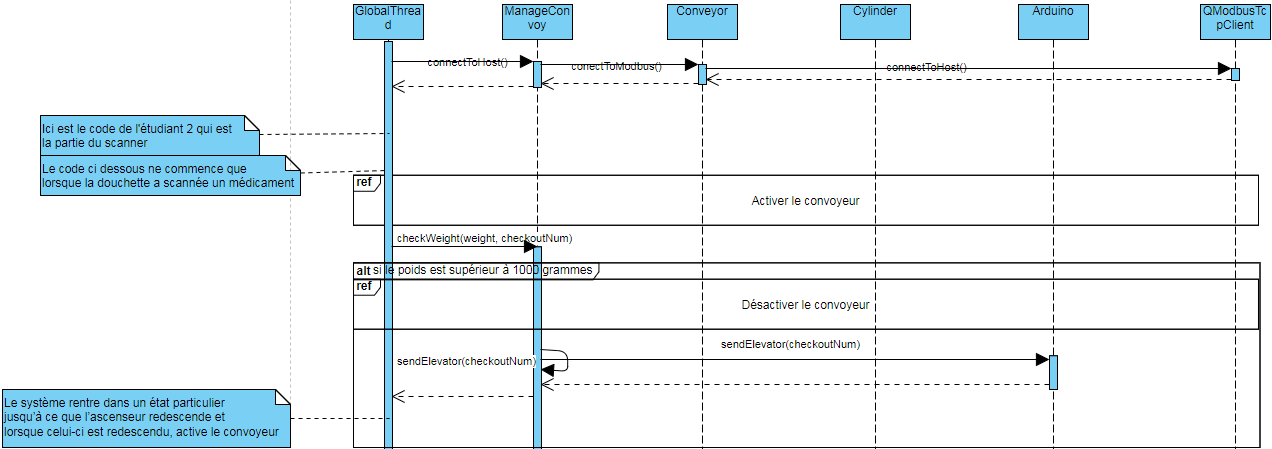
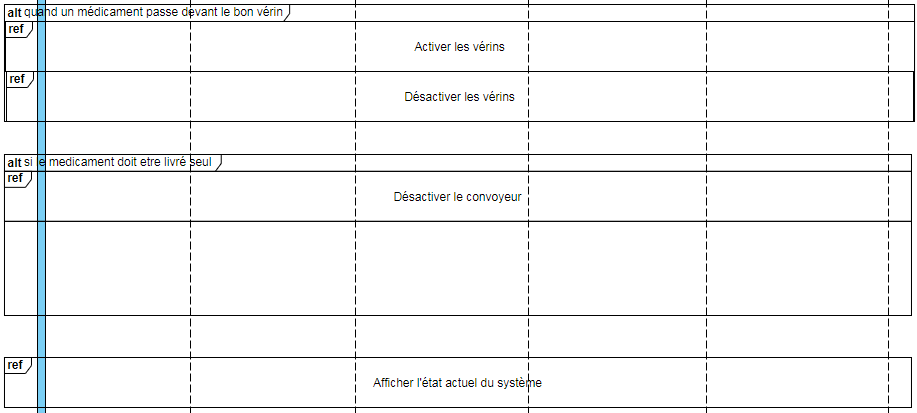
Post-condition :

Les médicaments sont convoyés.

Diagramme de séquence

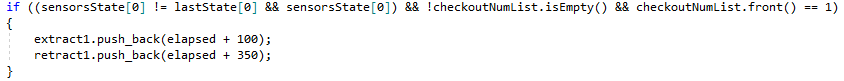




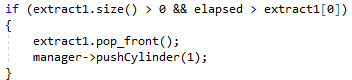
 

Mise en application du diagramme de séquence

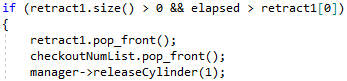
Cette partie étant trop volumineuse je ne me concentrerai que sur la partie des vérins. Un seul vérin sera expliqué car les deux vérins fonctionnent de la même manière.



Tout d’abord, on fixe des conditions pour l’activation du vérin. Le vérin sera activé seulement si un médicament est détecté par le capteur lié au vérin, si l’état du capteur est différent du dernier état (pour éviter que le vérin ne s’active pas plusieurs fois sur un médicament long) et que le numéro de caisse corresponde à l’ascenseur de ce vérin. Si toutes ces conditions sont réspectées, deux valeur sont stockées dans deux files. La première est un temps a attendre pour faire sortir le vérin et la deuxième est un temps a attendre pour faire rentrer le vérin. La variable elapsed est le temps écoulé depuis le lancement du programme.



Une fois que les valeurs sont stockées dans ces files, on vérifie si la file est vide et si la valeur ajoutée est inférieure au temps passé. Si c’est le cas, on enlève la valeur de la file et on active le vérin.



La même action est répétée pour rentrer le vérin. Le numéro de caisse est également enlevé de la file car le médicament est tombé dans l’ascenseur et a donc fini son chemin.

Résultat

La fonctionnalité est opérationnelle, le vérin s’active lorsque les conditions sont vraies et se rentre de la même manière.

# Recette

# Tests d’intégration du prototype

# Avancement et Conclusion