Module Integration Robotique : Bras robotique industriel STAUBLI et station FESTO

BUT 3 AII

IUT GEII Toulouse

2023-2024

B. Vandeportaele

Sources

- https://bvdp.inetdoc.net/files/iut/staubliiut/robot_srs_et_controleur/Contr%C3%B4leur%20CS9%20elec.PDF
- https://bvdp.inetdoc.net/files/iut/staubli/documentations/Val3.PDF
- https://youtu.be/rGYzGPdJpOs
- https://bvdp.inetdoc.net/files/iut/staubliiut/doc/Modbus%20Configuration%20CS8C-CS9_TSS00001604A.pdf
- https://bvdp.inetdoc.net/files/iut/staubliiut/doc/D28093901K.PDF
- https://bvdp.inetdoc.net/files/iut/staubliiut/doc/Real%20Time%20Val3-Fieldbus%20Synchronization%20TSS00001404A.pdf
-
-

Sommaire

- Cellule robotique et communication en VAL3
- Composants utilisés dans une approche modulaire
- Modbus TCP et HAL
- Parallélisme de tâches

La cellule robotique

- Un ou plusieurs robots
  - Bras
  - Contrôleur
  - Boitier sélection de mode de marche (WMS)

    Boitier de commande manuel/pendant (MCP)
  - Outils effecteurs
    - Par exemple pince + mors adaptés à la tâche
    - Éventuellement changeur d'outils
- Dans un environnement adapté
  - Table/support/tapis pour le robots et les pièces à manipuler
  - Avec des éléments de sécurité :
    - Protections mécaniques
    - Capteurs
- En interaction avec
  - Des capteurs, des actionneurs, des automates
  - Interfaçage via E/S physiques, réseau….
- Monitoré(s) par un ordinateur pour le développement, la simulation (PHL) et la supervision

La cellule robotique

- Solution étudiée ici :
  - Utiliser le contrôleur du robot pour piloter le bras et les différents éléments de la cellule
  - Multitâche(parallélisme)
  - Interfaçage via Modbus TCP
    - Via connecteur dédié (J207)
    - Via connecteur standard avec les sockets (J205)
    - Contrôleur en client Modbus

6 axes
+ pince via Fast I/O

Client
Modbus

4 entrées
4 sorties
Via Modbus TCP

Serveur
Modbus

Communication en VAL3

- Communication avec l'extérieur gérée à l'aide de variables associées à des ressources de communication
- Permet de récupérer l'état de capteurs, de piloter des actionneurs, de dialoguer avec un automate/PC
- E/S digitales ↔ variable booléenne (type **dio** chez Staubli)
- Réseau
  - Sockets UDP/TCP et ports série ↔ variables chaînes de caractères particulières (type **sio** chez Staubli)
  - ...

Composants utilisés

Clients Modbus

Serveurs Modbus

Contrôleur CS9

J205

J207

tcp_server

tcp_server_simulator

Emulateur Contrôleur CS9 via SRS

J205

J207

Interface Modbus/24V arduino

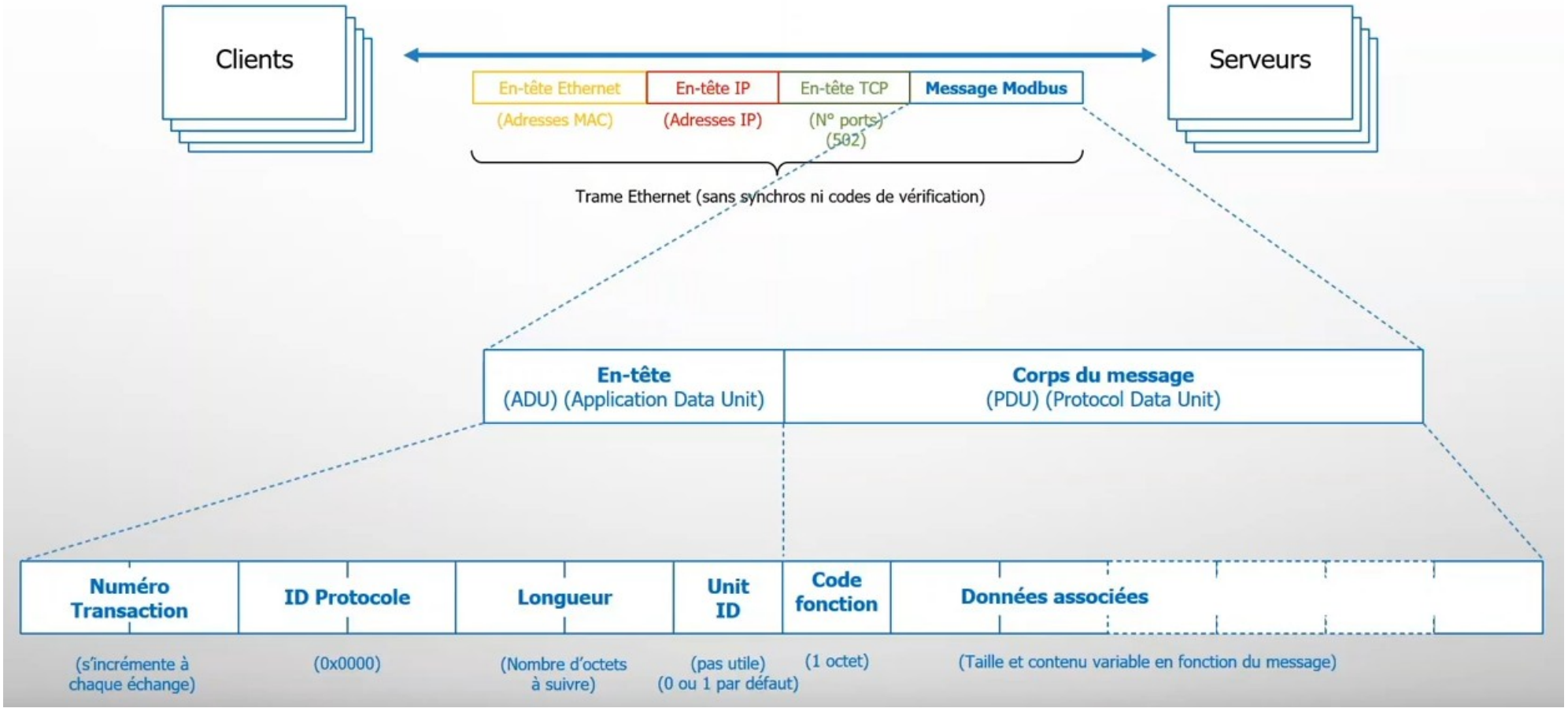
Station FESTO

EasyModbus TCP Client

EasyModbus TCP Server

Wireshark

Rappels Modbus TCP



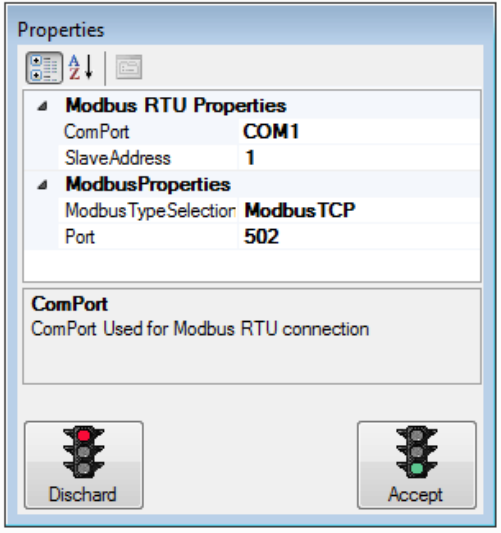
Source : https://youtu.be/rGYzGPdJpOs?t=142

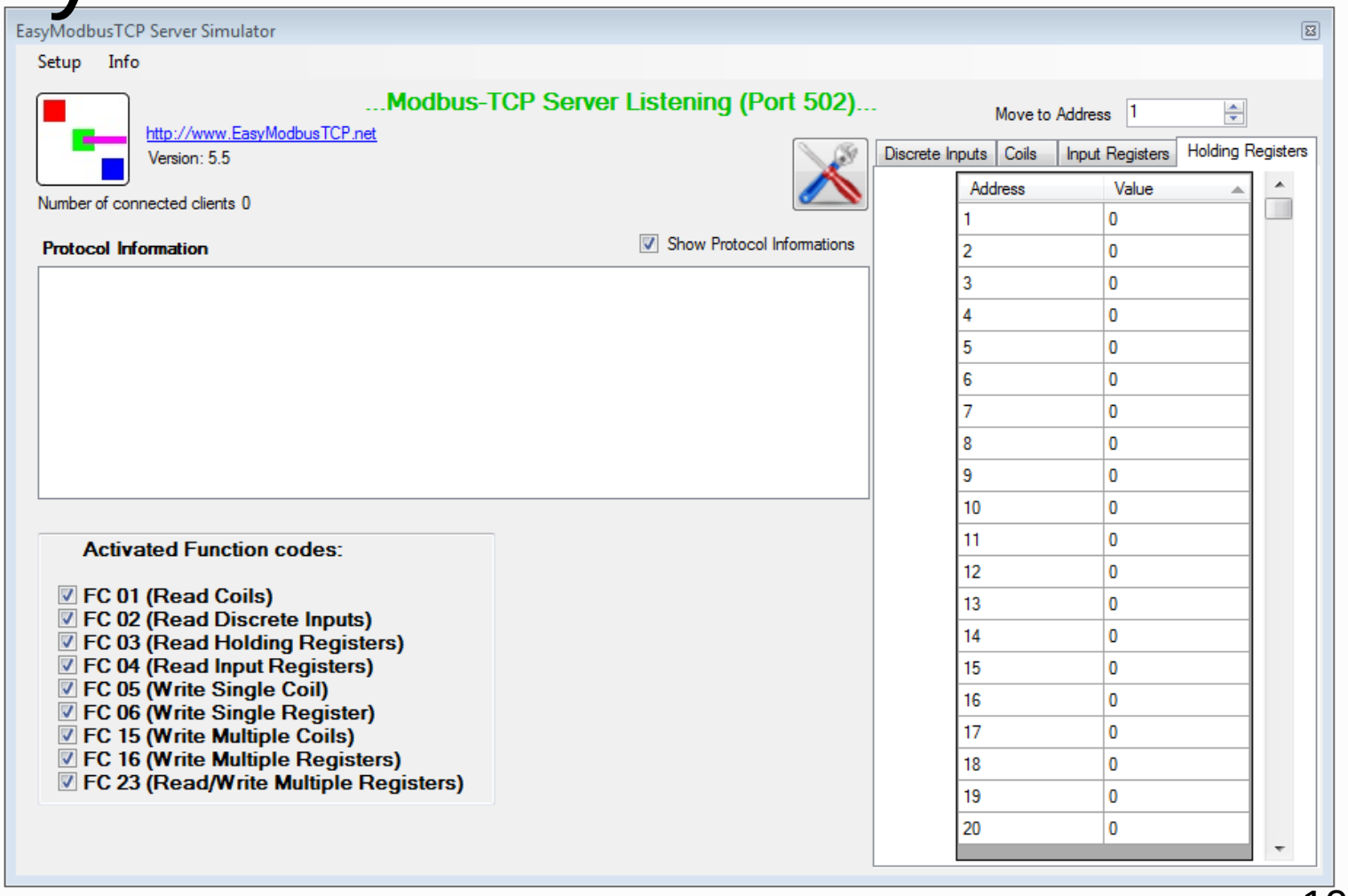
Rappels Modbus TCP

Data type	Mode	Function name	Function code (FC)
Bit	Digital input (read only)	Read Discrete Inputs	2
	Digital output (read / write)	Read Coils	1
		Write Single Coil	5
		Write Multiple Coils	15
Word (16 bits)	Analog input (read only)	Read Input Registers	4
	Analog output (read / write)	Read Holding Registers	3
		Write Single Register	6
		Write Multiple registers	16

Source : https://bvdp.inetdoc.net/files/iut/staubliiut/doc/Modbus%20Configuration%20CS8C-CS9_TSS00001604A.pdf#page=42

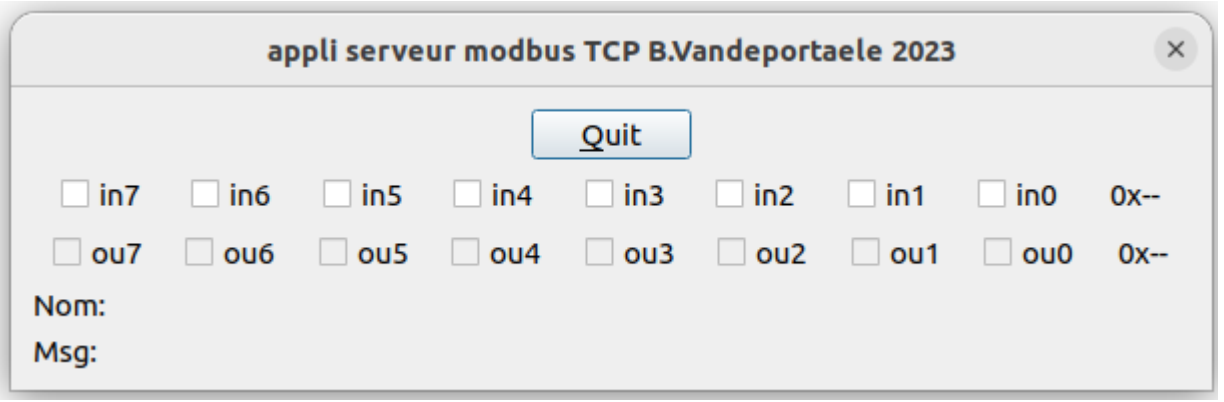
EasyModbus TCP Server





EasyModbus TCP Client

tcp_server

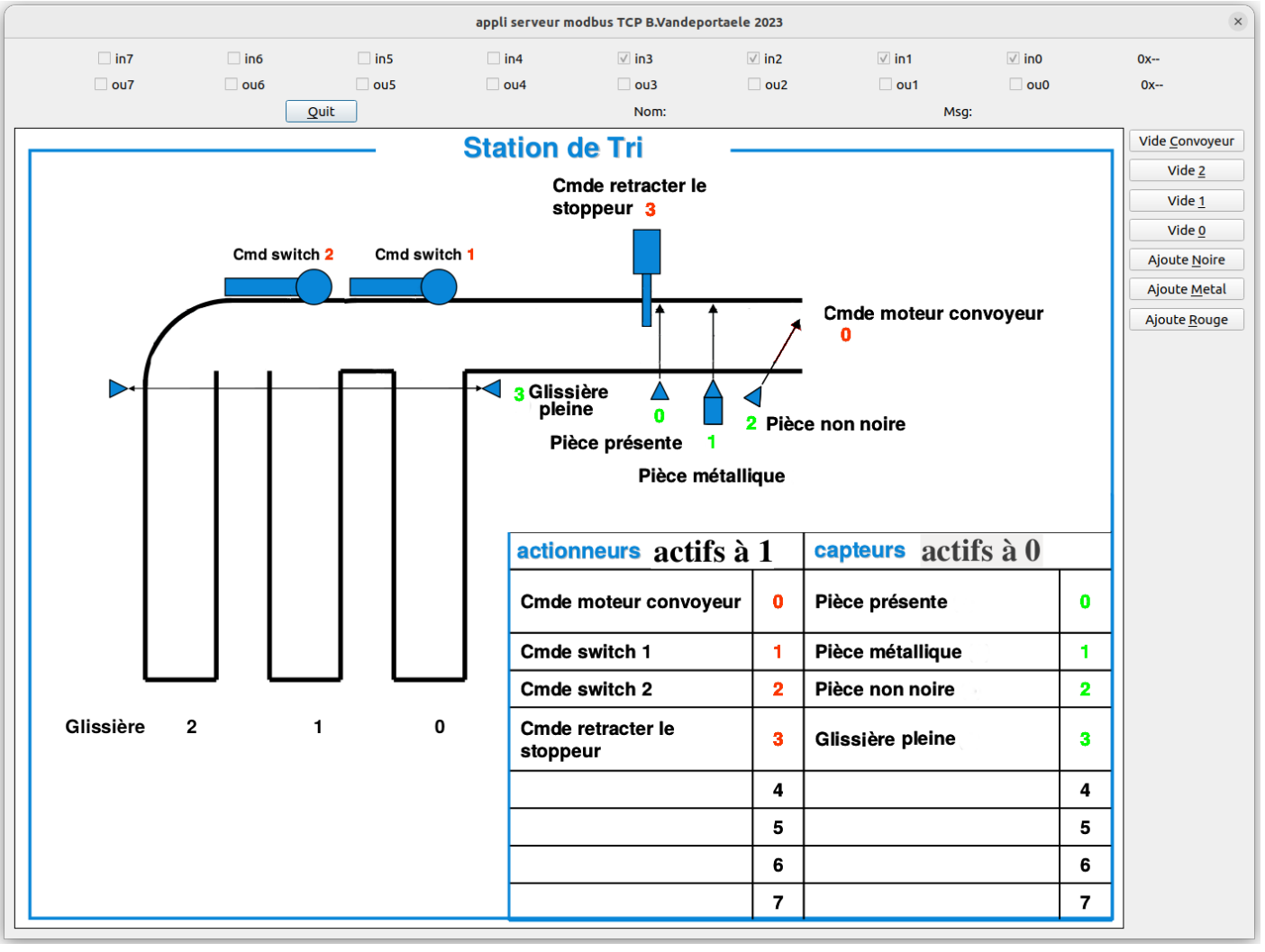


Simulation de l’interface E/S Modbus simplifiée

Implémente uniquement les Fonctions Codes 4 et 6
Unit ID non utilisé
Données situées dans la case mémoire n°0 (mais adresse non testée)
FC4 avec LEN=1 pour lire un mot de 16 bits codant les 8 entrées
FC6 avec LEN=1 pour écrire un mot de 16 bits codant les 8 sorties

L’utilisateur peut cliquer sur les entrées pour les cocher (mise à 1)

tcp_server_simulator



actionneurs actifs à 1		capteurs actifs à 0	
Cmde moteur convoyeur	0	Pièce présente	0
Cmde switch 1	1	Pièce métallique	1
Cmde switch 2	2	Pièce non noire	2
Cmde retracter le stoppeur	3	Glissière pleine	3
	4		4
	5		5
	6		6
	7		7

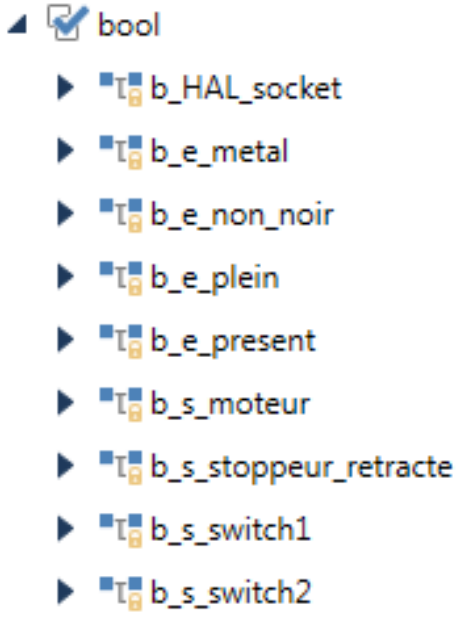
Simulation de l'interface E/S Modbus + la station FESTO

Même comportement que tcp_server pour la communication Modbus

Les E/S sont visualisées seulement

L'utilisateur peut ajouter et retirer des pièces mobiles à différents endroits

Correspondances variables VAL3/ES

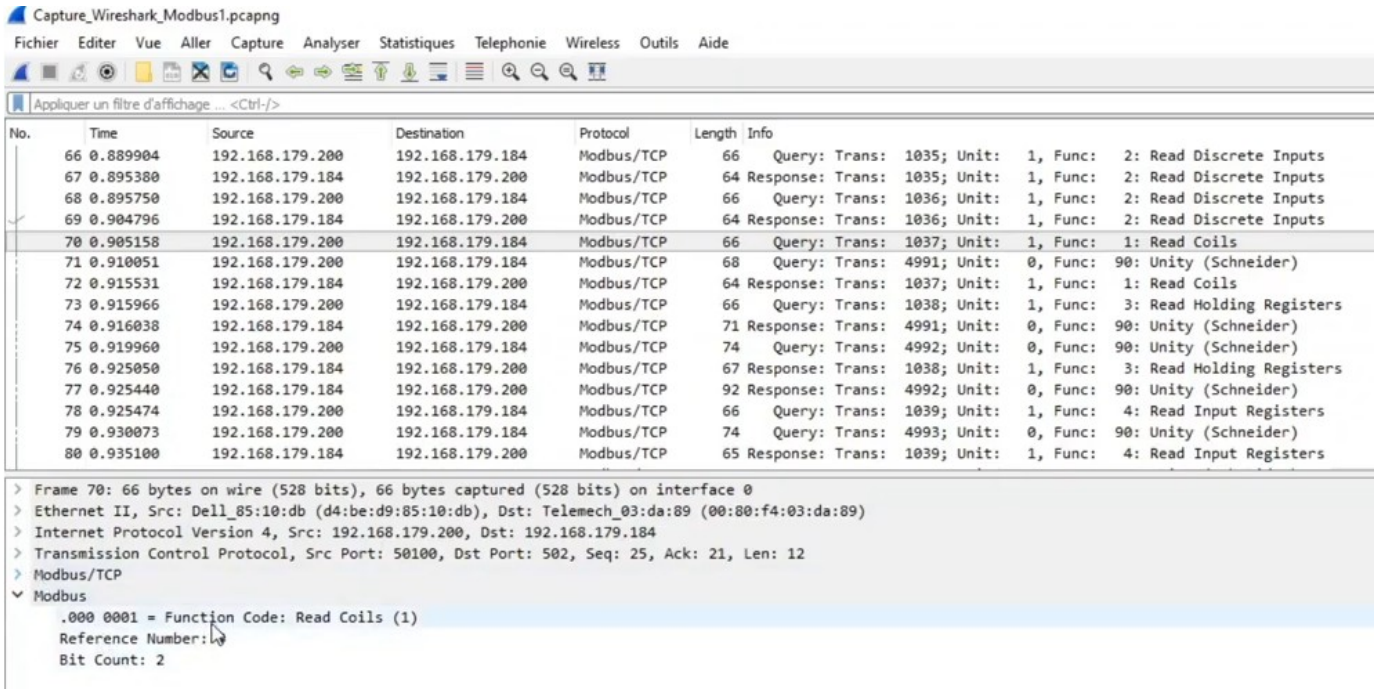


Variables globales partagées entre les applications

actionneurs actifs à 1		capteurs actifs à 0	
Cmde moteur convoyeur	0	Pièce présente	0
Cmde switch 1	1	Pièce métallique	1
Cmde switch 2	2	Pièce non noire	2
Cmde retracter le stoppeur	3	Glissière pleine	3
	4		4
	5		5
	6		6
	7		7

Les numéros des bits indiquent les emplacements dans les mots 16 bits échangés via Modbus

Wireshark



Utilisé pour analyser les échanges entre composants sur l'interface réseau (Sélectionner une ou plusieurs interfaces à espionner)

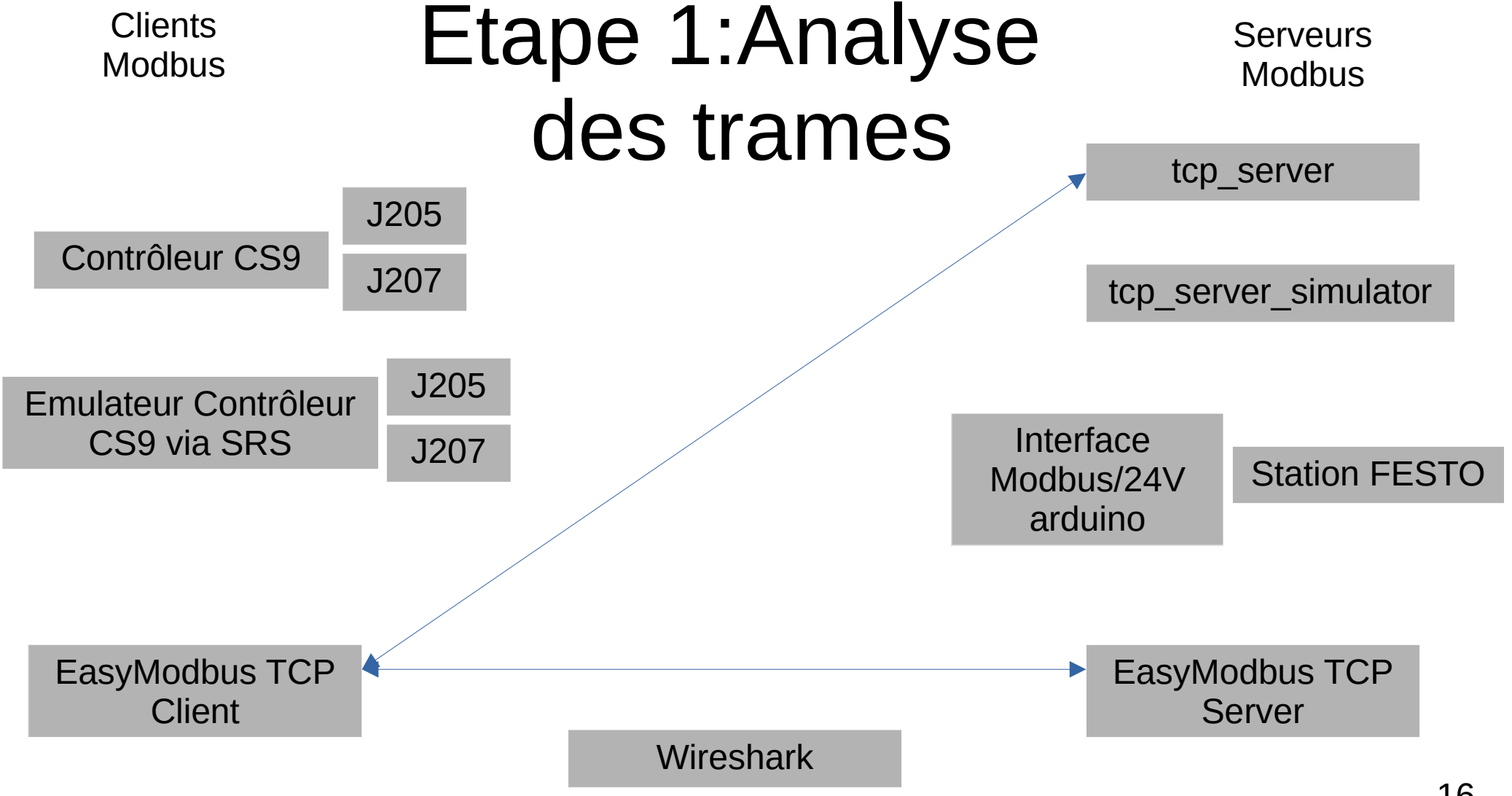
Permet de voir les étapes de synchronisation/acquittement etc…

Filtre pour une adresse IP : ip.src==172.16.8.132 || ip.dst==172.16.8.132

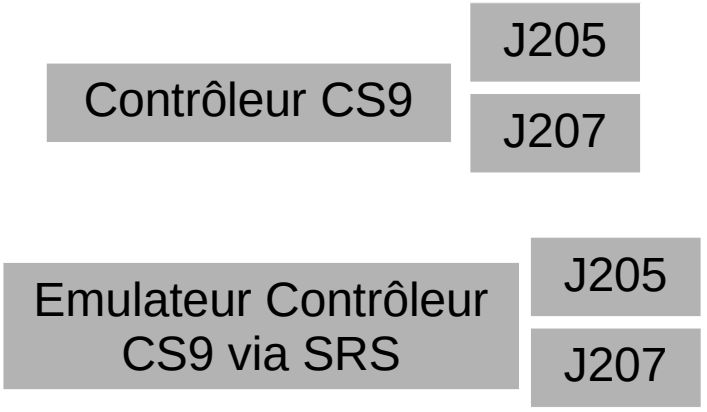
Filtre pour une adresse IP uniquement pour les échanges sur le port TCP 502: (ip.src==172.16.8.132 || ip.dst==172.16.8.132) && tcp.port==502

Filtre uniquement pour les échanges modbus : modbus

Etape 1:Analyse des trames



Différentes interfaces réseau

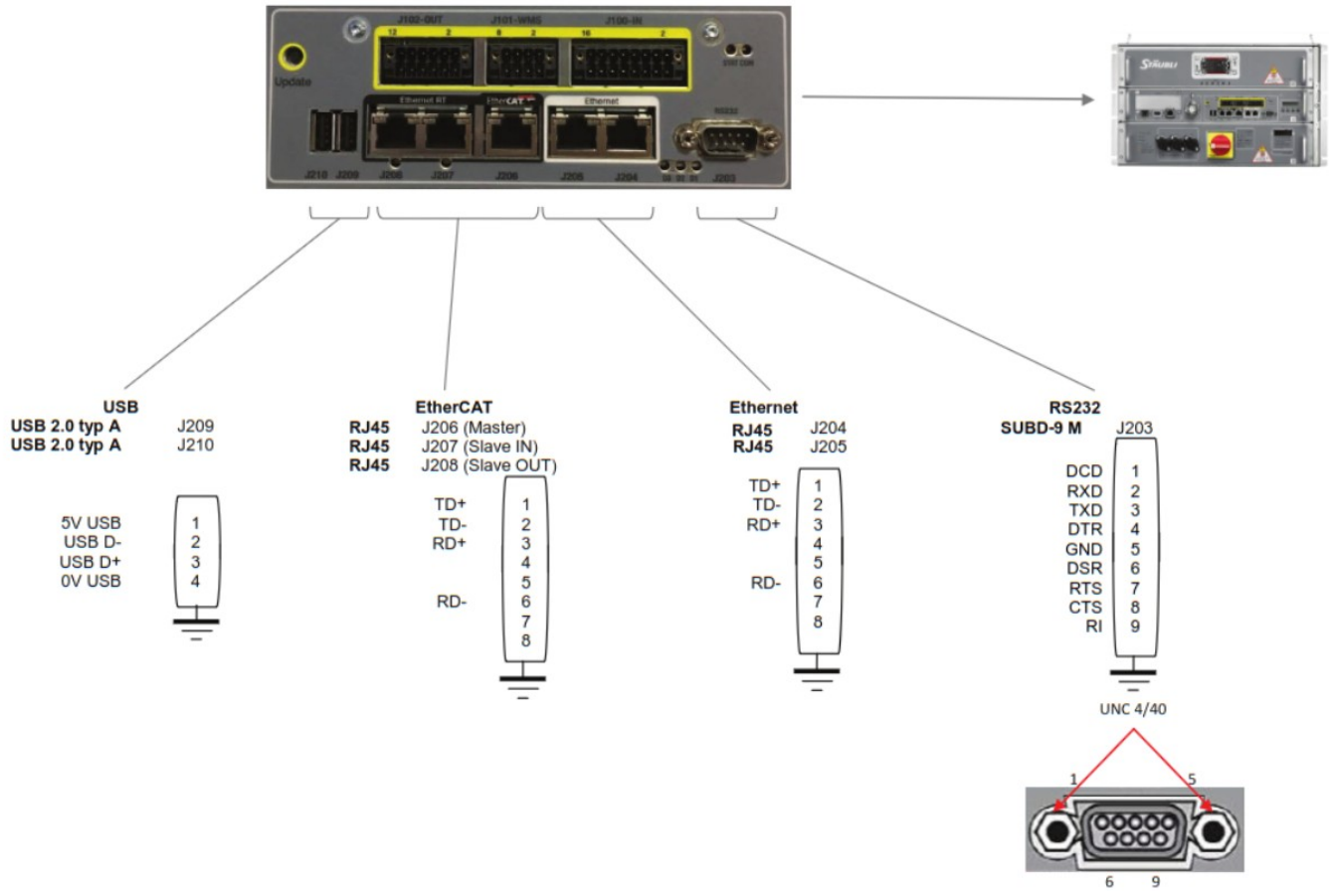


J205 : interface ethernet « Standard »
communication via des sockets
flexible : implémentation « à la main » des protocoles
simulable
IP sur le vrai contrôleur : 172.16.8.250

J207 : interface bus de terrain « dédiée »
configurable via l'outil SYCON
libère le développeur de la tâche de communication
association variable/ressources de communication
non simulable (hormis via boite à bouton)
IP sur le vrai contrôleur : 172.16.8.132

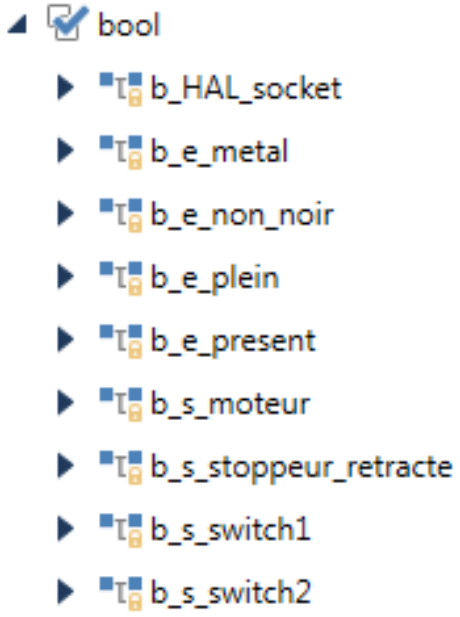
Rappel : Les sockets chez Staubli n'ont qu'un seul numéro de port, l'émulateur de contrôleur ne peut donc pas communiquer avec une autre application sur le même hôte, nous utiliserons donc 2 PC, un pour simuler le contrôleur, l'autre pour le serveur Modbus

Différentes interfaces réseau



Source : https://bvdp.inetdoc.net/files/iut/staubliiut/doc/D28093901K.PDF#page=44

Couche d’abstraction du matériel (HAL)



Variables globales partagées entre les applications

HALinit(string IP)
HALlireEntrees()
HALecrireSorties()

En fonction de b_HAL_socket, utilise J205 ou J207

Mêmes fonctions que l’on utilise J205 ou J207

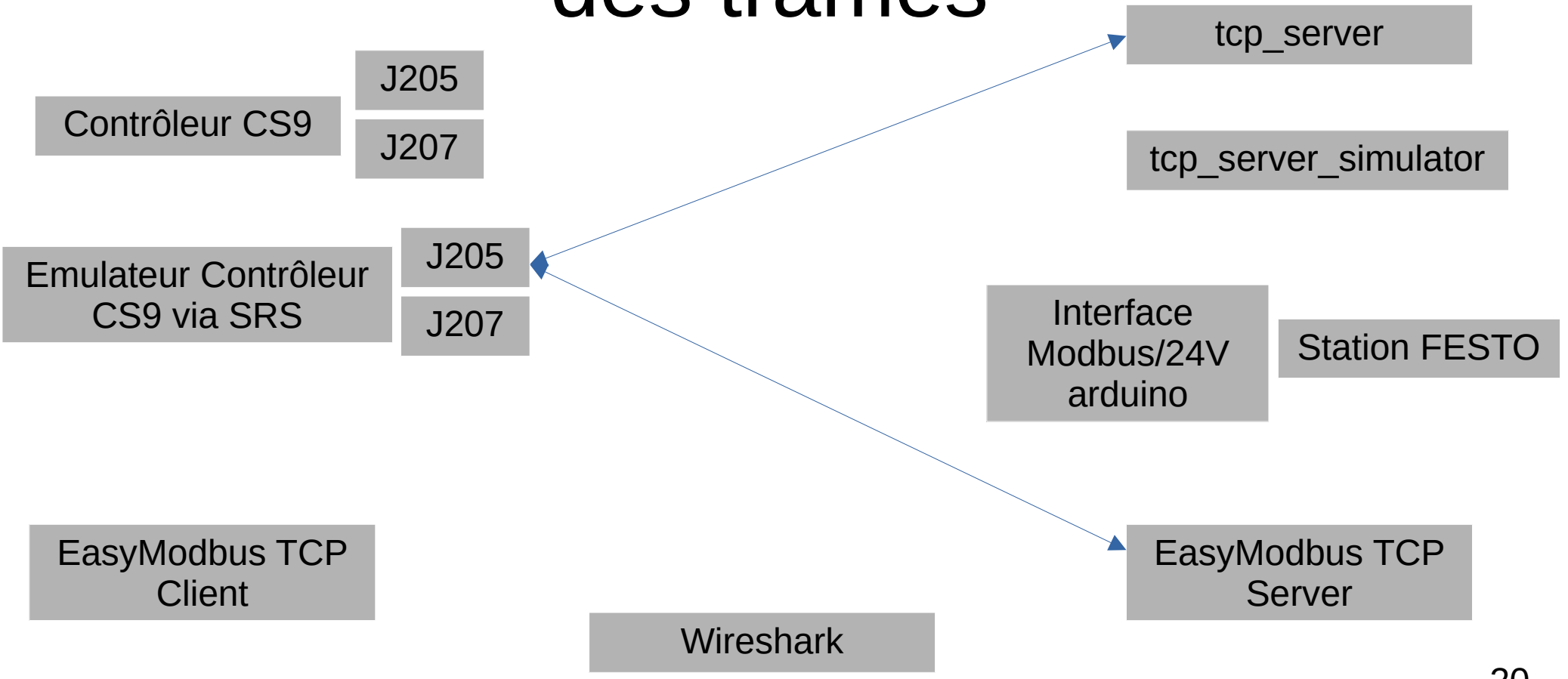
actionneurs actifs à 1		capteurs actifs à 0	
Cmde moteur convoyeur	0	Pièce présente	0
Cmde switch 1	1	Pièce métallique	1
Cmde switch 2	2	Pièce non noire	2
Cmde retracter le stoppeur	3	Glissière pleine	3
	4		4
	5		5
	6		6
	7		7

Les numéros des bits indiquent les emplacement dans les mots 16 bits échangés via Modbus

Etape 2: (Dé)Codage des trames

Clients Modbus

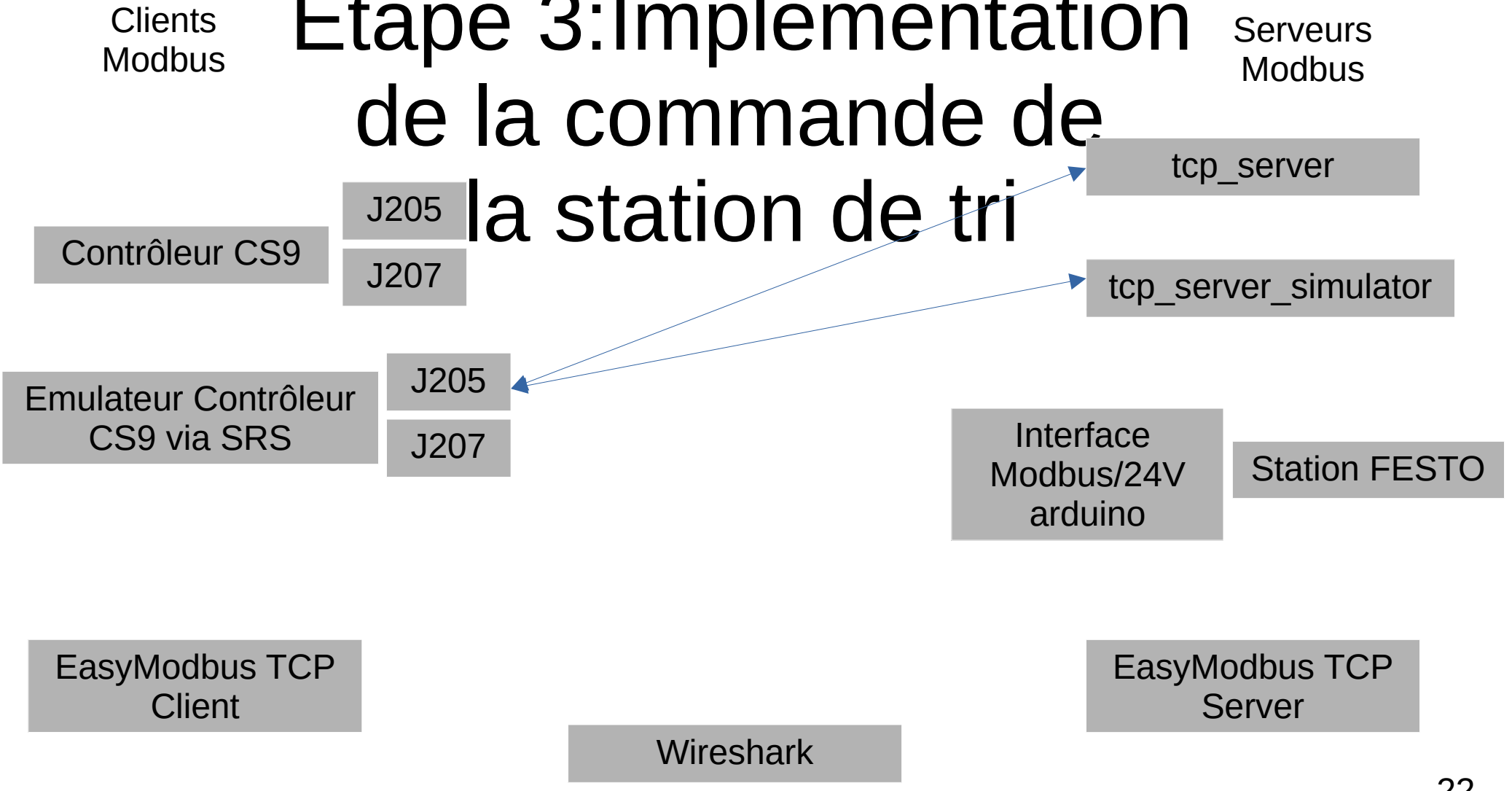
Serveurs Modbus



(Dé)Codage des trames

- Implémentation des fonctions HALinit(string IP), HALlireEntrees() et HALecrireSorties() pour b_HAL_socket=true
- Contrôleur en client Modbus qui fait des requêtes FC4 et FC6
- Utilisation des fonctions de conversion binaire nbits/numérique et de masquage binaire
- Gestion des données à envoyer/recevoir par variable tableau de num

Etape 3:Implémentation de la commande de la station de tri

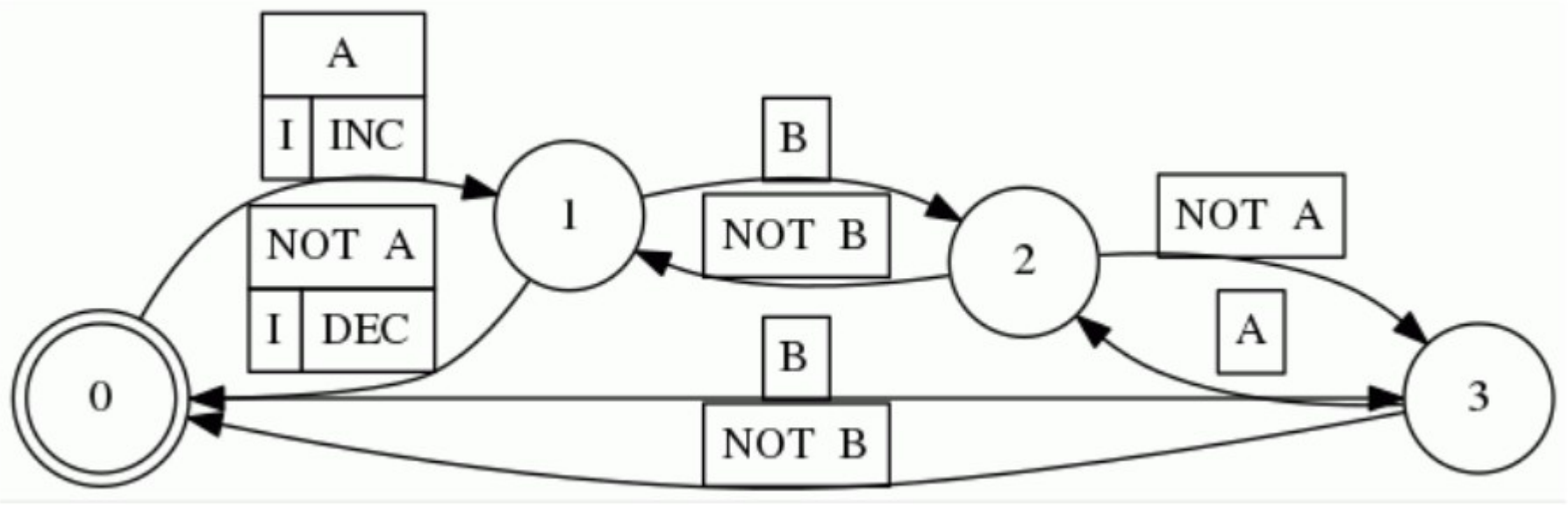


Implémentation de la commande de la station de tri

- Système de commande modélisé par une Machine A Etats (MAE)
- Représentation graphique
- Le système (ou une partie) est dans un état à un instant donné
- Transition vers un autre état soumis à une condition (flèche orientée depuis l'état d'origine vers l'état final), condition indiquée au dessus de la flèche
- Il peut y avoir plusieurs transitions depuis un même état
- Les conditions de changement d'état depuis un même état doivent être exclusives (priorité)
- Des actions peuvent être associées à des états ou à des transitions
- Evolution synchrone de la MAE: une horloge la cadence

Implémentation de la commande de la station de tri

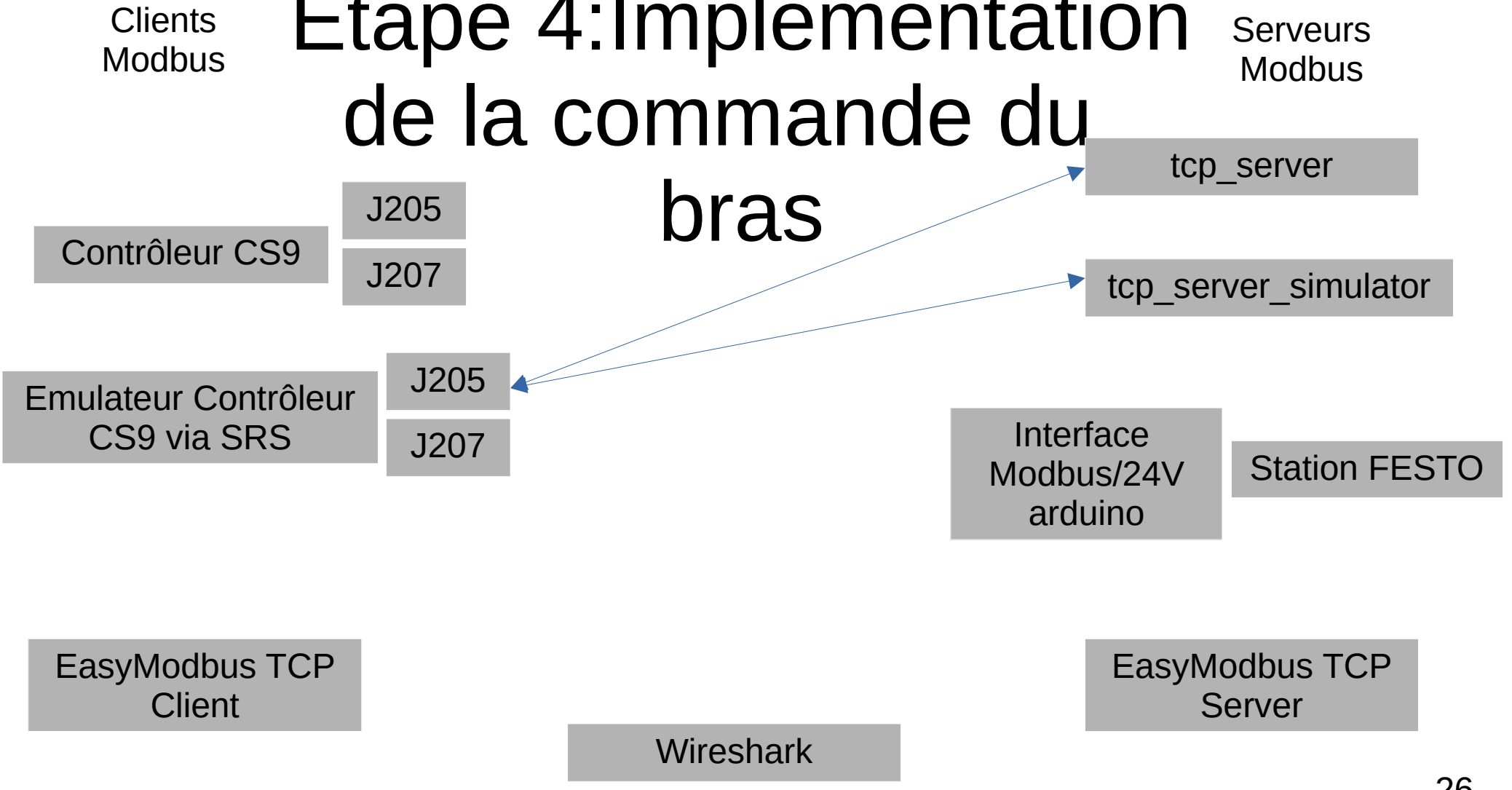
- Exemple de MAE :



Implémentation de la commande de la station de tri

- Règles de codage systématique en VAL3 sera vue en TD

Etape 4:Implémentation de la commande du bras

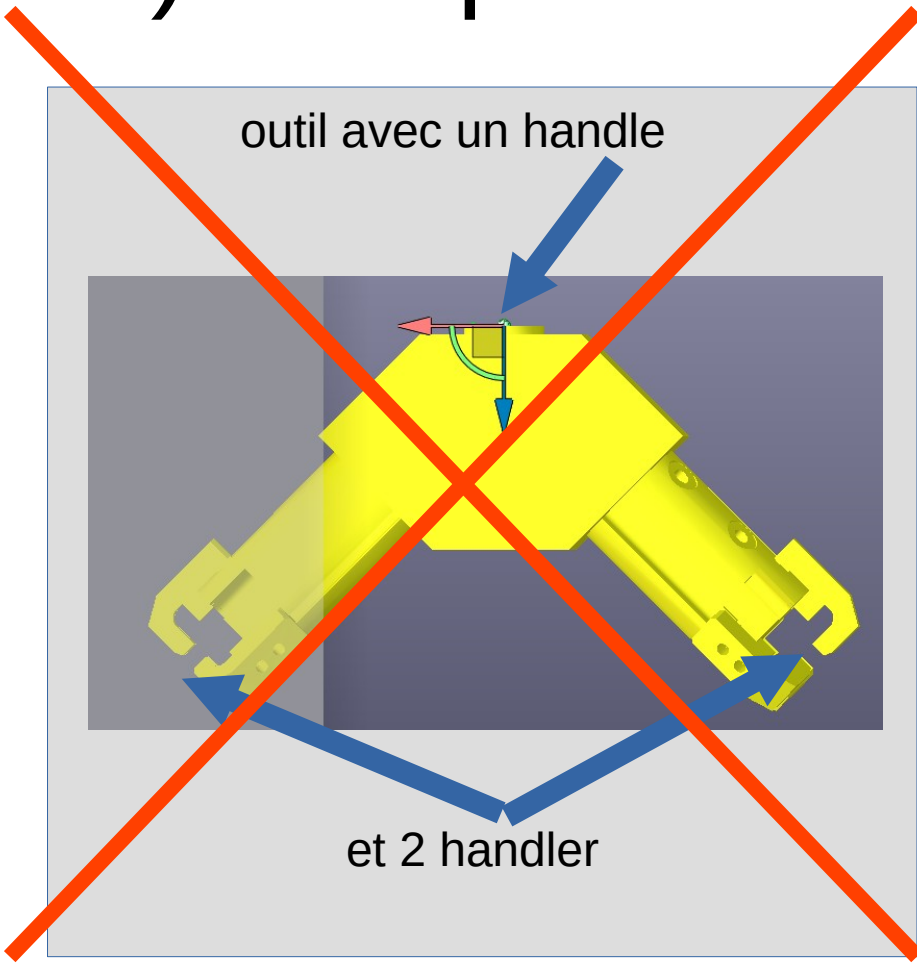


Implémentation de la commande du bras

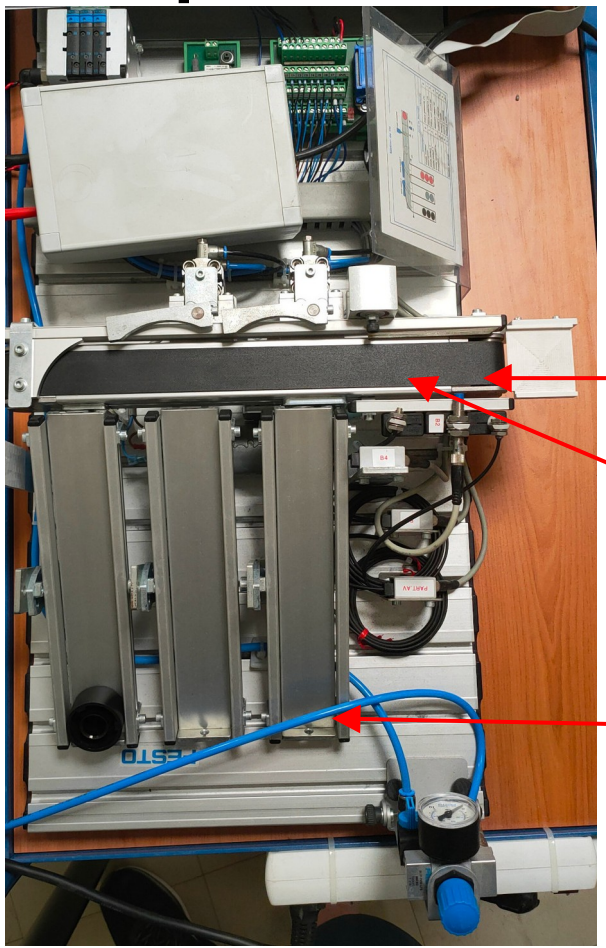
- Plusieurs zones de travail :
  - Le robot passe de l'une à l'autre par des mouvements dans l'espace articulaire (movej) après s'être éloigné des éléments de la zone précédente avec un mouvement de dégagement dans l'espace cartésien (movel)
    - 1 variable joint par zone de travail
  - À l'intérieur de chaque zone de travail, le robot ne fait que des mouvements dans l'espace cartésien (movel et movec)

Outils (organe terminal) et repères

- Dans ce projet pour tout le monde :
  - Le TCP est la bride
  - Les points sont définis dans le repère world
  - Ceci permet d'utiliser des points fournis par l'enseignant afin d'éviter les collisions avec les éléments de la cellule
  - Pas de pièces mobiles pour les cylindres, pas de handle non plus



Implémentation de la commande du bras



Cycle en U avec des points d'approche et de dégagement obligatoire pour tous les points!
Orientation de la pince importante !

pDebutTapis

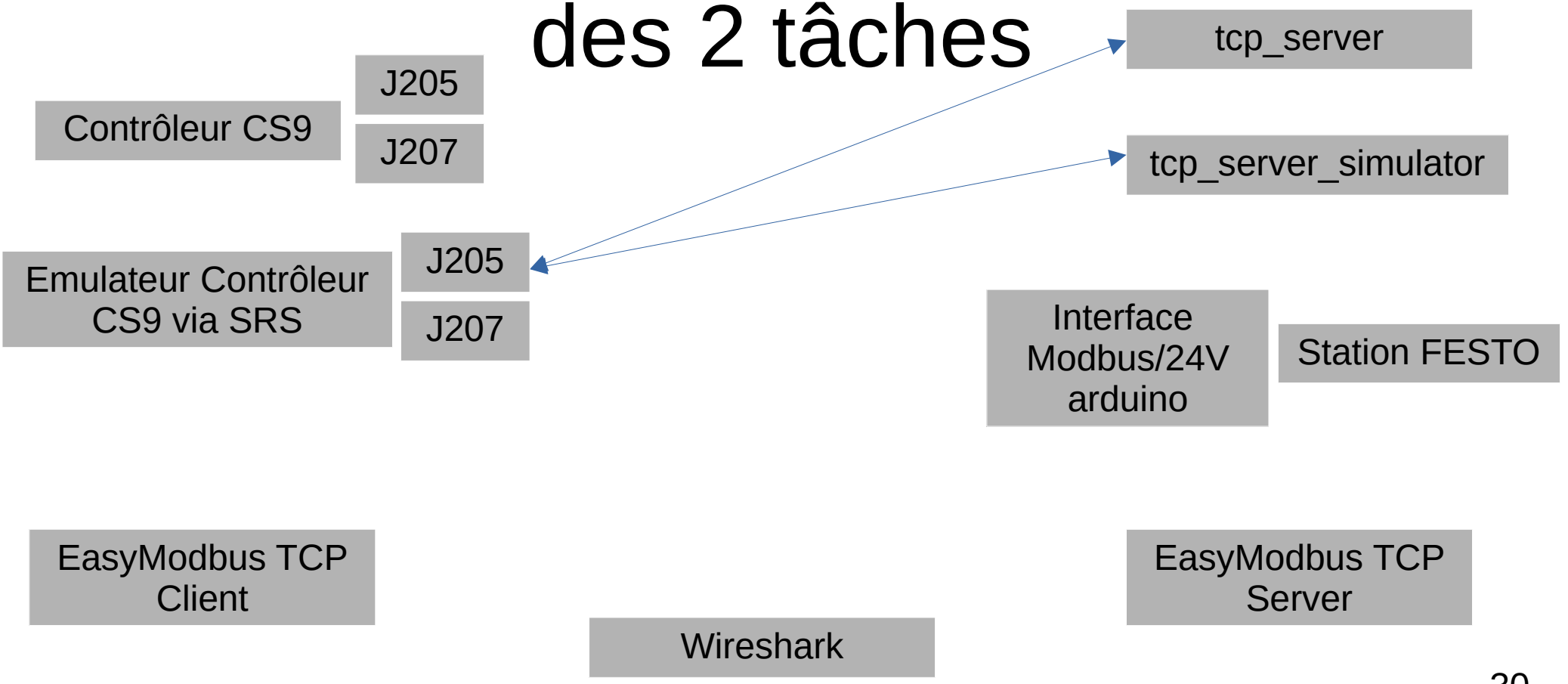
pStoppeur

Tableau de 3 points pour la saisie dans les glissières : pGlissi

Clients Modbus

Etape 5:Synchronisation des 2 tâches

Serveurs Modbus



Parallélisme et synchronisation des 2 tâches

- Plusieurs programmes sont exécutés par le contrôleur :
  - Certains programmes sont lancés automatiquement (commande du bras, gestions des IOs, communication)
  - d'autres sont lancés au démarrage ou à l'arrêt du programme (start/stop)
  - d'autres sont lancés par les programmes
    - call pour appeler un sous programme (qui retourne quand il est terminé)
    - taskCreate pour créer une tâche asynchrone parallèle
    - taskCreateSync pour créer une tâche synchrone parallèle
- Ils peuvent communiquer en échangeant des valeurs via des variables partagées (en prenant des précautions : MUTEX)

3.1 What are the basic system tasks on CS9?

- Synchro system task (System)

This task handles the robot trajectory and refresh of system IOs boards, which are: CpuIO, DsiIO, FastIO, PowerSupplyIO, Rsi9IO, StarcIO.

- IO Refresh task (System)

This task handles the refresh of user IOs boards, which are, J206 (EtherCAT master), J207/J208 (Real Time Ethernet), Fieldbus (Hilscher CIFX50E-xx PCIe boards).

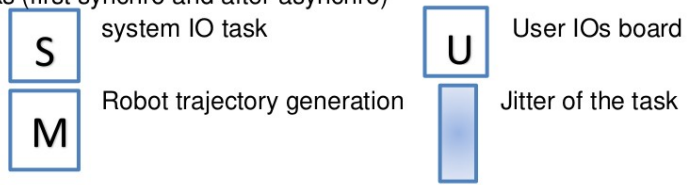
- Val3 task (User)

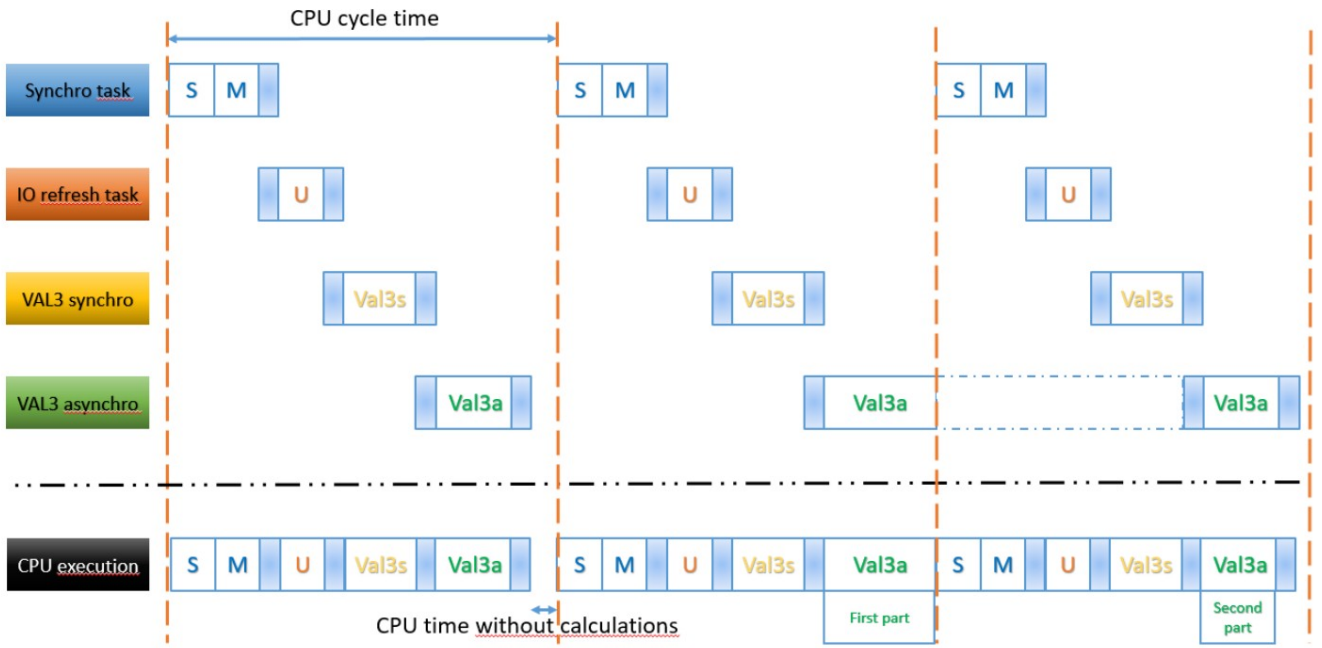
This section indeed contains many tasks, all of them can be written by user. Each of these tasks can be Asynchronous or Synchronous. In theory there is no limit to the number of different tasks, in reality the limit is the computation power of the CPU. Pay attention that a synchronous Val3 task consumes a lot of CPU resources. See chapter 4 of this document for more detailed information.

Source : https://bvdp.inetdoc.net/files/iut/staubliiut/doc/Real%20Time%20Val3-Fieldbus%20Synchronization%20TSS00001404A.pdf

In each scan the CPU executes:

1. Firstly the synchro system task
2. Secondly the IO refresh task
3. Thirdly Val3 tasks (first synchro and after asynchro)
4. Graph legend:





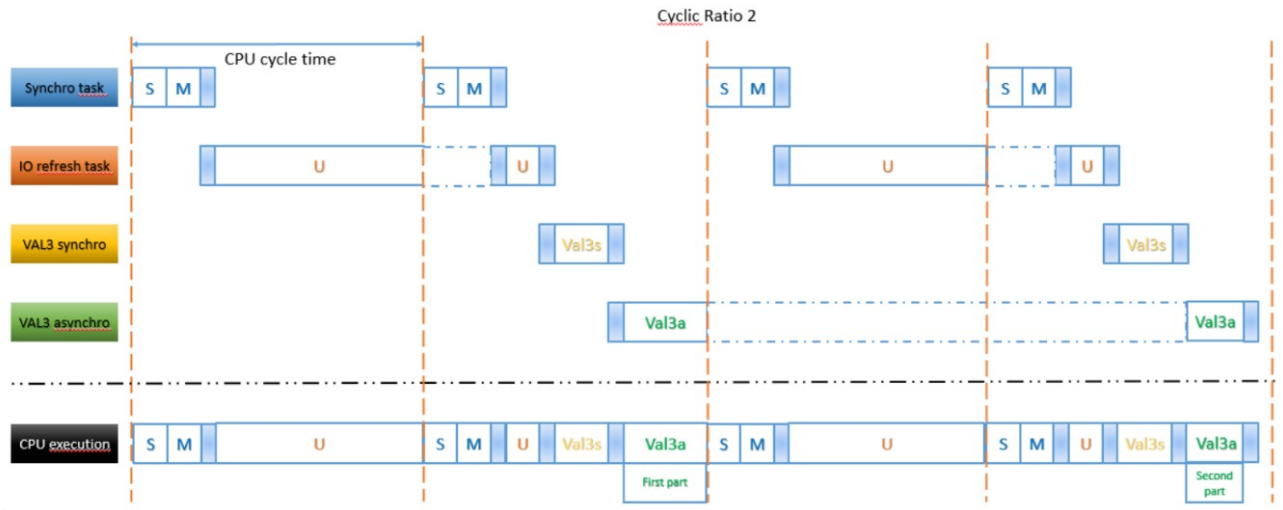
Source : https://bvdp.inetdoc.net/files/iut/staubliiut/doc/Real%20Time%20Val3-Fieldbus%20Synchronization%20TSS00001404A.pdf

3.3 Cyclic ratio management

When the number of IOs is very high, the CPU could not be able to refresh all IOs in one scan. If it is, the IO refresh task stops with an error. For this reason, it is possible to configure the IO refresh cycle ratio: the variable means how many scan of the CPU the IO refresh task can take to update all IOs. The standard value is one, if for example cyclicRatio is set to 2, it means that the IO refresh task can take up to 2 scan of the CPU to refresh all IOs.

To change the cyclicRatio, in usr/configs/cell.cfx

```
<UserIORefresh>
      <Float name = "cyclicRatio" value = "2" />
</UserIORefresh>
```

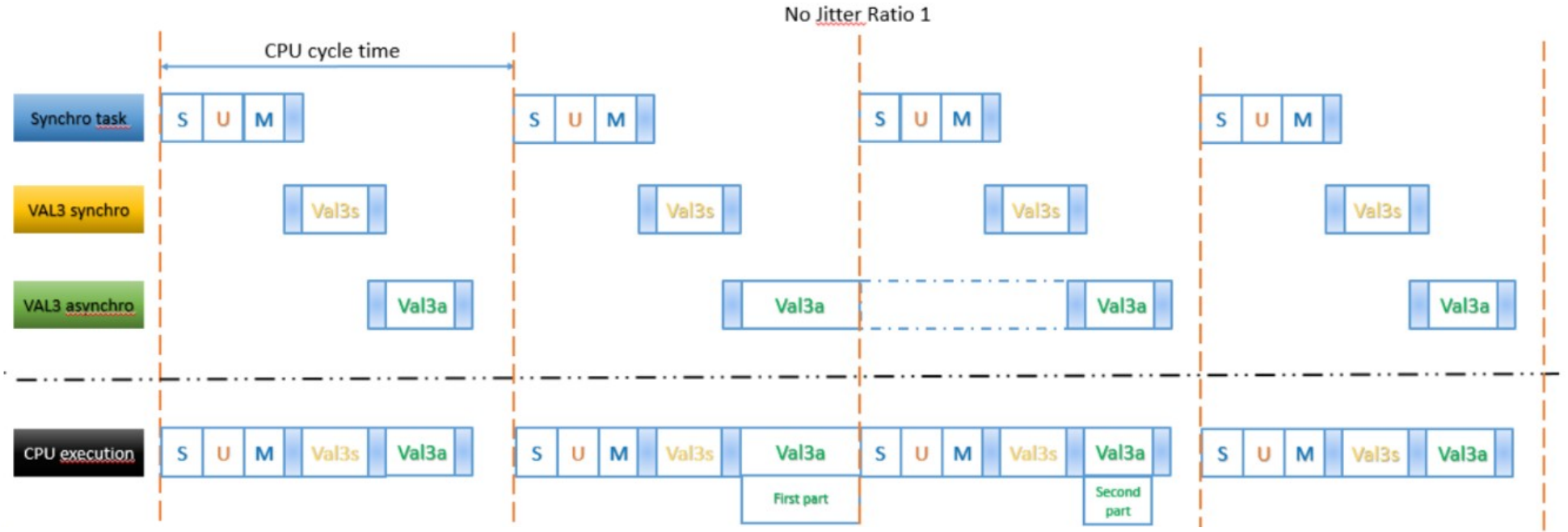


Obviously it is recommended to set this value as little as possible. In other words to increase this value only if it is needed. A bigger value of cyclicratio will lead to a bigger discrepancy between the Val3 program and the IO refresh.

Source : https://bvdp.inetdoc.net/files/iut/staubliiut/doc/Real%20Time%20Val3-Fieldbus%20Synchronization%20TSS00001404A.pdf

In no jitter mode, the IO refresh task does not exist and the user boards are refreshed during the Synchronous system task. Basically in each scan the CPU executes:

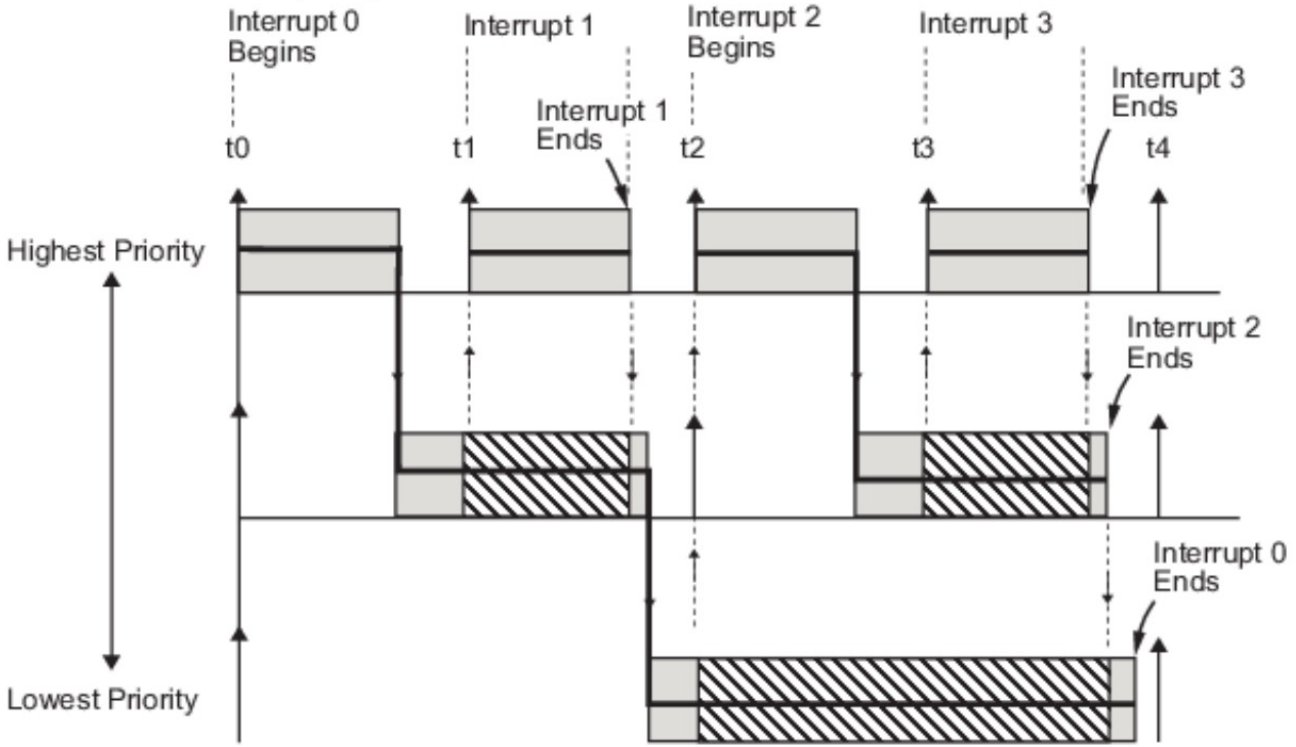
1. Firstly the synchro system task, which contains IOs
2. Secondly Val3 tasks (first synchro and after asynchro)



If the user board refresh time becomes too long an overrun will occur on the synchro system task: the robot is stopped and a reboot is needed. In this way we can force the system to be sure that the IOs are correctly updated each cycle. If not, the robot stops.

4.1 How asynchronous Val3 task management works

When several tasks of an application are running, they appear to run concurrently and independently. This is true if the whole application is observed over a sufficiently long period of time (about a second), but not true if its specific behavior is examined over a short period of time.
In fact, as the system has only one processor, it can only execute one task at a time. Simultaneous execution is simulated by very fast sequencing of the tasks that execute a few instructions in turn before the system moves on to the next task. In the following picture is explained in which way is possible to execute many Val3 tasks at the same time with different priority:



Basically if the cycle time of the CPU is set to 4ms, every cycle of the CPU

Basically if the cycle time of the CPU is set to 4ms, every cycle of the CPU

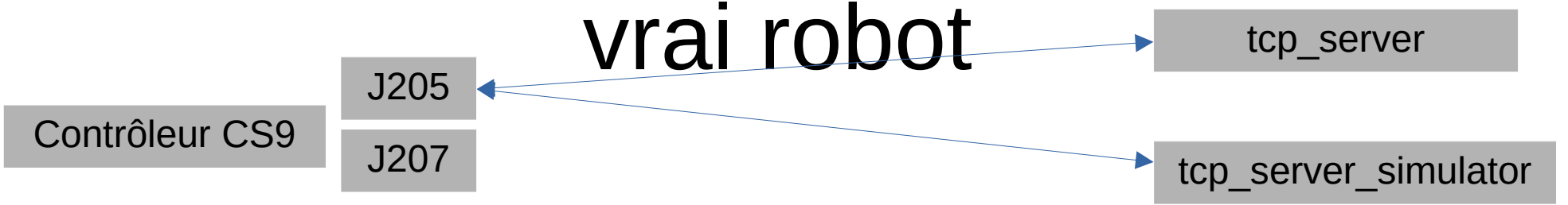
VAL 3 asynchronous task sequencing obeys the following rules:
1. The tasks are sequenced in the order in which they were created
2. During each sequence, the system attempts to execute a number of VAL 3 instruction lines corresponding to the priority of the task.
3. When an instruction line cannot be terminated (runtime error, waiting for a signal, task stopped, etc.) the system moves on to the next VAL 3 task.
4. When all VAL 3 tasks have been completed, the system keeps some free time for lower priority system tasks (such as network communication, user screen refresh, file access), before a new cycle is started.
The maximum delay between two sequential cycles is equal to the duration of the last sequencing cycle; but, most of the time, this delay is null because the system does not need it.
The VAL 3 instructions that can cause a task to be sequenced immediately are as follows:

- **watch()** (condition wait timeout)
- **delay()** (timeout)
- **wait()** (condition waiting time)
- **waitEndMove()** (arm stop waiting time)
- **open()** and **close()** (arm stop waiting time followed by timeout)
- **taskResume()** (waits until the task is ready for restart)
- **taskKill()** (waits for the task to be actually killed)
- **disablePower()** (waits for power to be actually cut off)
- The instructions accessing the contents of the disk (**libLoad, libSave, libDelete, libList, setProfile**)
- The sio reading/writing instructions (operator =, **sioGet()**, **sioSet()**)
- **setMutex()** (waits for the Boolean mutex to be false)

Clients Modbus

Etape 6:Tests sur le vrai robot

Serveurs Modbus



Emulateur Contrôleur CS9 via SRS

J205

J207

Interface Modbus/24V arduino

Station FESTO

EasyModbus TCP Client

Wireshark

EasyModbus TCP Server

Etape 7:Tests sur le vrai robot avec la FESTO

Clients Modbus

Serveurs Modbus

Clients Modbus

Etape 8:Tests sur le vrai robot avec la FESTO via J207

Serveurs Modbus

tcp_server

tcp_server_simulator

J205

J207

Contrôleur CS9

Emulateur Contrôleur CS9 via SRS

J205

J207

Interface Modbus/24V arduino

Station FESTO

EasyModbus TCP Client

EasyModbus TCP Server

Wireshark

Configuration client Modbus TCP sur J207

Utiliser SYCON pour configurer l'interface
Les paramètres seront données en TD

Utiliser l'onglet IO pour associer une variable (de type num) à chaque registre Modbus de la FESTO

Les fonctions de la HAL doivent convertir les variables 16 bits en booléens et inversement

Le téléchargement des réglages vers le contrôleur se fait via la section IO, il ne faudra pas que vous chargiez vos réglages, l'enseignant s'en chargera (car les réglages restent sur le contrôleur et nécessite un redémarrage).

Tâche robotique

- Définir ce que le robot doit faire (ou pas)
- Comment il interagit avec :
  - Les pièces sur lesquelles il agit
  - L'environnement (les éléments fixes et mobiles/variables de la cellule)
  - Les éventuels opérateurs humains
- Choix des capteurs et actionneurs
- Le robot est généralement un élément d'une chaîne :
  - Gestion des flux de pièces en entrée et sortie

Étapes de la programmation du robot

1) Définition de la tâche robotique (définir le maximum de contraintes en amont)
2) Identification des sous tâches (décomposition)
3) Positionnement des éléments de la cellule (en simulation idéalement et dimensionnement (choix) du robot)
4) Paramétrisation des trajectoires
   - Quels sont les points « réglables » et les repères dans lesquels les définir
5) Programmation (Approche Coarse to Fine : trajectoires grossières et/ou saccadées dans un premier temps)
6) Simulation
7) Apprentissage des points et repères sur le vrai robot
8) Adaptation de la simulation au points mesurés sur le vrai robot
9) test sur le vrai robot à vitesse réduite
10) Mesure des performances en simulation et/ou en réel (par exemple temps de cycle)

A chaque étape, il est possible de rétroagir sur les étapes précédentes

Approche modulaire (Idéalement): Simulation des éléments matériels et logiciels en interaction avec le robot

Simulation de la tâche idéalement en amont de la construction de la cellule (par exemple pour choisir le bon robot après en avoir essayé plusieurs, et positionner au mieux les éléments de la cellule)

Exemple de Programme VAL3

```
<?xml version="1.0" encoding="utf-8" ?>
<programList xmlns="ProgramNameSpace”>
<program name="demo" public="false”>
<description/>
<paramSection/>
<localSection/>
<source>
<code>

</code>
</source>
</program>
</programList>
```

```
Begin
//mouvement de connexion pour reprise
resetMotion(jBrasLeve) ;
//sortie de configuration singuliere par
mouvement en espace articulaire
movej(p[3],flange,mNomSpeed)
WaitEndMove()
While true
//répétition d’un cycle
  movel(p[0],flange,mNomSpeed)
  movel(p[1],flange,mNomSpeed)
  movej(p[3],flange,mNomSpeed)
  movel(p[0],flange,mNomSpeed)
  movec(p[2],p[1],flange,mNomSpeed)
  movej(p[3],flange,mNomSpeed)
  waitEndMove()
endWhile
end
```