SysML Diagramme d'état

Exemples de sur un portail coulissant

LYCEE EUGENE LIVET NANTES

Dans ce document nous allons présenter la programmation d'un portail coulissant sur une carte Arduino Méga en utilisant les diagrammes d'états. Plusieurs cahiers des charges seront abordés.

1 CABLAGE ET MATERIEL

Le câblage de la partie puissance (moteur à deux sens de rotation) : **OU**verture (KM1), **FE**rmeture (KM2) est présenté figure 1.

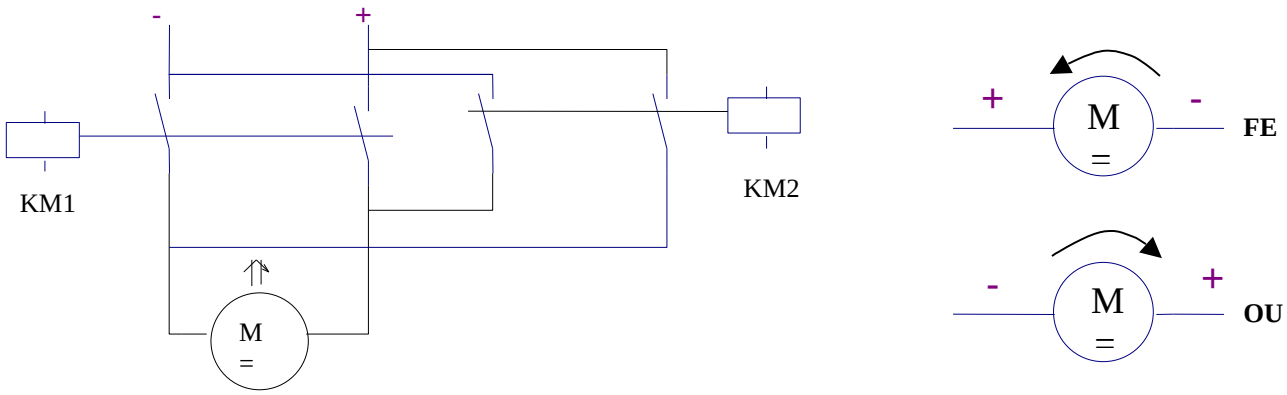

Figure 1 : Câblage de la partie puissance

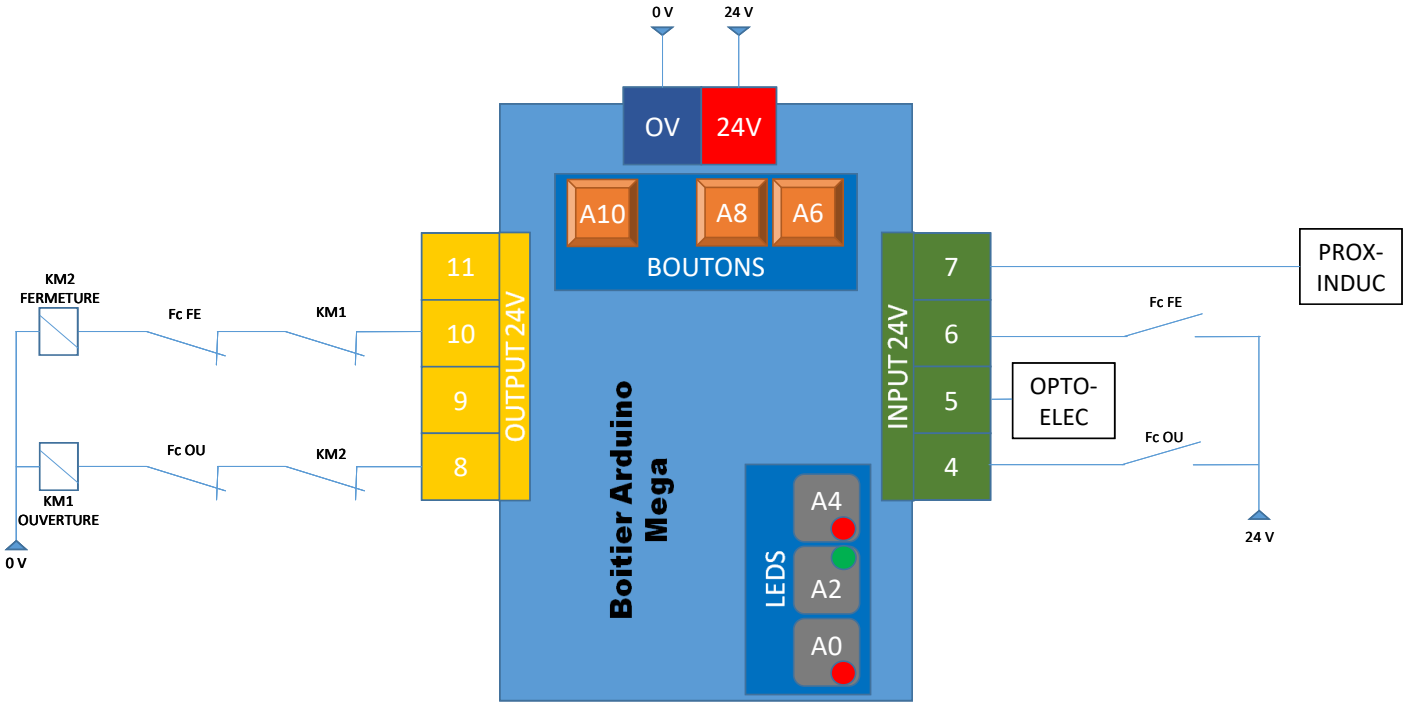

Figure 2 : Câblage de l'arduino

La figure 2 présente le câblage de l'arduino. Les contacteurs du moteur étant en 24V, nous avons utilisé à l'intérieur du « Boitier Arduino Méga » des « Level shifter » pour faire l'adaptation 5V vers 24V. Pour les entrées, l'adaptation est réalisée par des optocoupleurs. Les boutons poussoirs sont directement câblés sur l'arduino en utilisant la résistance de tirage intégrée à l'arduino (pullup).

Ref	Nom	Commentaires
KM1	Relais KM1	commande 24V
KM2	Relais KM2	commande 24V
Fc FE	Fin de course fermeture	normalement ouvert ET normalement fermée
Fc OU	Fin de course ouverture	normalement ouvert ET normalement fermée
OPTO-ELEC	Barrière optique de sécurité	24V
PROX-INDUC	Top tour axe moteur	24V

Figure 3 : Entrées / Sorties

2 EXEMPLE DE PRISE EN MAIN

2.1 Graphes d'états

La carte Arduino se programme en C/C++. En tant que tel, ce langage n'est pas directement adapté à la programmation de systèmes séquentiels. Pour décrire le comportement d'un système séquentiel, nous pouvons utiliser les graphes d'état issus de la norme UML/SYSML.

Pour mémoire, Les graphes d'état d'UML/SysML décrivent le comportement interne d'un objet à l'aide d'un automate à états finis. Ils présentent les séquences possibles d'états et d'actions qu'une instance de classe peut traiter au cours de son cycle de vie en réaction à des événements discrets (de type signaux, invocations de méthode). Le graphe d'état est le seul graphe, de la norme UML/SysL, à offrir une vision complète et non ambiguë de l'ensemble des comportements de l'élément auquel il est attaché.

Pour programmer en graphe d'état dans l'environnement arduino, il est nécessaire d'utiliser la librairie **obj_stat** qui comporte les différentes classes utiles. Pour l'utiliser, les fichiers *obj_stat.h* et *obj_stat.cpp* doivent se trouver soit dans le **répertoire de travail** ou dans le répertoire *obj_stat* du répertoire librairies arduino (Par défaut : C:\Program Files (x86)\Arduino\libraries\obj_stat).

L'inclusion la librairie dans votre fichier de travail se fait de manière classique en utilisant la directive :

```
#include "obj_stat.h"
```

si les fichiers *obj_stat.h* et *obj_stat.cpp* sont dans votre répertoire de travail et :

```
#include <obj_stat.h>
```

si les fichiers *obj_stat.h* et *obj_stat.cpp* sont dans le répertoire *obj_stat* du répertoire librairies arduino (Par défaut : C:\Program Files (x86)\Arduino\libraries\obj_stat).

La librairie *obj_stat* contient les classes suivantes :

- **Event** : permet de gérer les entrées numériques (front montant, front descendant, changement d'état).
- **DigitalOut** : permet de gérer les sorties numériques (on, off, on/off pendant une durée donnée, clignotement on/off suivant un durée donnée).
- **Stat** : permet de gérer le fonctionnement d'un graphe d'état.
- **Bistable** : permet de gérer 2 sorties numériques antagonistes comme des contacteurs inverseurs de moteur ou des distributeurs. Par construction, les deux sorties ne peuvent être actives en même temps.
- **Button** : permet de gérer les entrées numériques provenant d'un bouton. Cette classe reprend les événements de Event (front montant, front descendant, changement d'état) et en ajoute d'autres (clic, double clic, appui long).
- **Timer** : permet de gérer les temporisations.
- **Pulse** : permet de générer un top tous les x ms.

Lorsque l'on programme en graphe d'état, il est important de maitriser le temps de cycle de la fonction *loop*. En conséquence il ne faut jamais utiliser la fonction **delay** qui bloque l'exécution du microcontrôleur. Les temporisations doivent absolument être gérées par des timers, soit en créant des objets de la classe *Timer*, soit en utilisant la méthode *after* de la classe *Stat*.

Pour plus d'informations, on se reportera à la documentation complète de la librairie et au document d'accompagnement. Pour tout ce qui concerne la syntaxe en C/C++, on se reportera au site internet de référence arduino : https://www.arduino.cc/en/Reference/HomePage

2.2 Exemple de prise en main

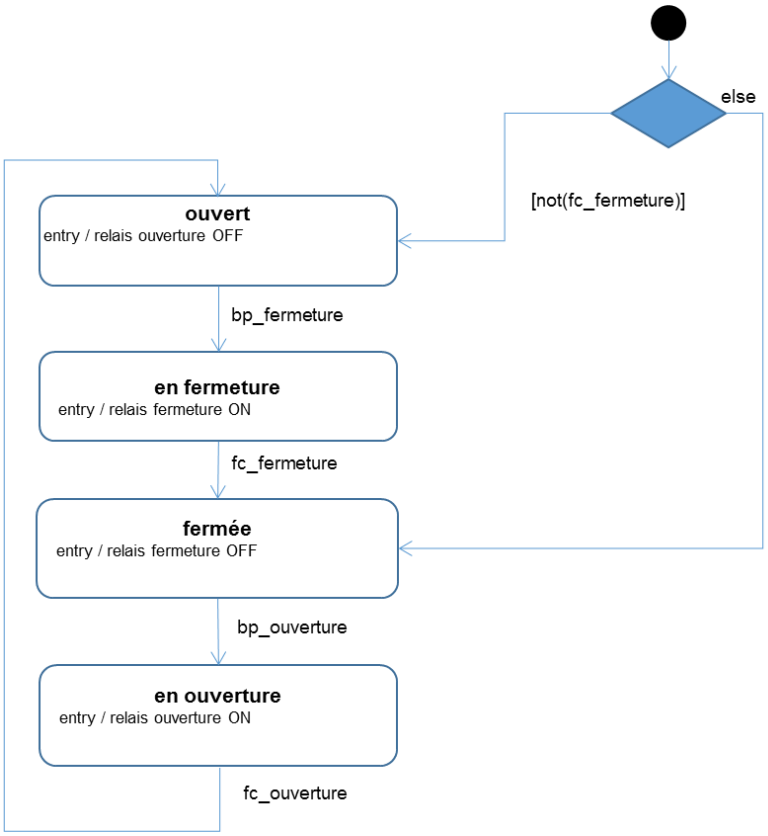

Figure 4 : graphe d'état de prise en main

Tableau 1: graphe de prise en main Entrées/Sorties

type	E/S	Borne arduino	alias	classe	instance	commentaires
Bouton poussoir	E	A6	BP_OUVERTURE	Button	bp_ouverture	Bouton grove
Bouton poussoir	E	A8	BP_FERMETURE	Button	bp_fermeture	Bouton grove
Entrée câble	E	4	FC_OUVERTURE	Event	fc_ouverture	24V via opto-coupleur
Entrée câble	E	6	FC_FERMETURE	Event	fc_fermeture	24V via opto-coupleur
Sortie câble	S	8	OUVERTURE	Bistable	moteur	24V via relais
Sortie câble	S	10	FERMETURE	Bistable	moteur	24V via relais

Le graphe d'état ci-dessus décrit le comportement attendu. Le programme correspondant se trouve dans le répertoire GTI1. Tester le programme **GTI1.ino** en visualisant les messages du Moniteur Série (Menu Outils,

vitesse 9600 bauds). Ce programme permet d'ouvrir le portail lorsqu'on appuie sur le bouton poussoir ouverture (A6) et de fermer celui-ci lorsqu'on appuie sur le bouton fermeture (A8).

3 EXEMPLE 2 : AJOUT DE REPERES VISUELS

La figure 5 présente une évolution du graphe d'état de prise en main. On souhaite dans cette partie ajouter des repères visuels au moyen de 3 leds. On allume la led rouge A4 (***led_fc_fermeture***) lorsque le portail est fermé, on allume la led verte A2 (***led_fc_ouverture***), lorsque le portail est ouvert et on fait clignoter la led A0 (***led_mvt***) lorsque le portail est en mouvement.

Le tableau suivant récapitule les entrées sorties du projet.

Tableau 2: graphe 2

type	E/S	Borne arduino	alias	classe	instance	commentaires
Led rouge	***S***	***A0***	***LED_MVT***	***DigitalOut***	***led_mvt***	***Led grove Rouge***
Led verte	***S***	***A2***	***LED_OUVERT***	***DigitalOut***	***led_fc_ouverture***	***Led grove Verte***
Led rouge	***S***	***A4***	***LED_FERME***	***DigitalOut***	***led_fc_fermeture***	***Led grove Rouge***
Bouton poussoir	E	A6	BP_OUVERTURE	Button	bp_ouverture	Bouton grove
Bouton poussoir	E	A8	BP_FERMETURE	Button	bp_fermeture	Bouton grove
Entrée câble	E	4	FC_OUVERTURE	Event	fc_ouverture	24V via opto-coupleur
Entrée câble	E	6	FC_FERMETURE	Event	fc_fermeture	24V via opto-coupleur
Sortie câble	S	8	OUVERTURE	Bistable	moteur	24V via relais
Sortie câble	S	10	FERMETURE	Bistable	moteur	24V via relais

Le programme **GTI2.ino**, correspond au programme **GTI1.ino** avec en supplément l'intégration des nouvelles entrées/sorties ***A0***, ***A2*** et ***A4***.

Le graphe d'état figure 5 décrit le comportement attendu.

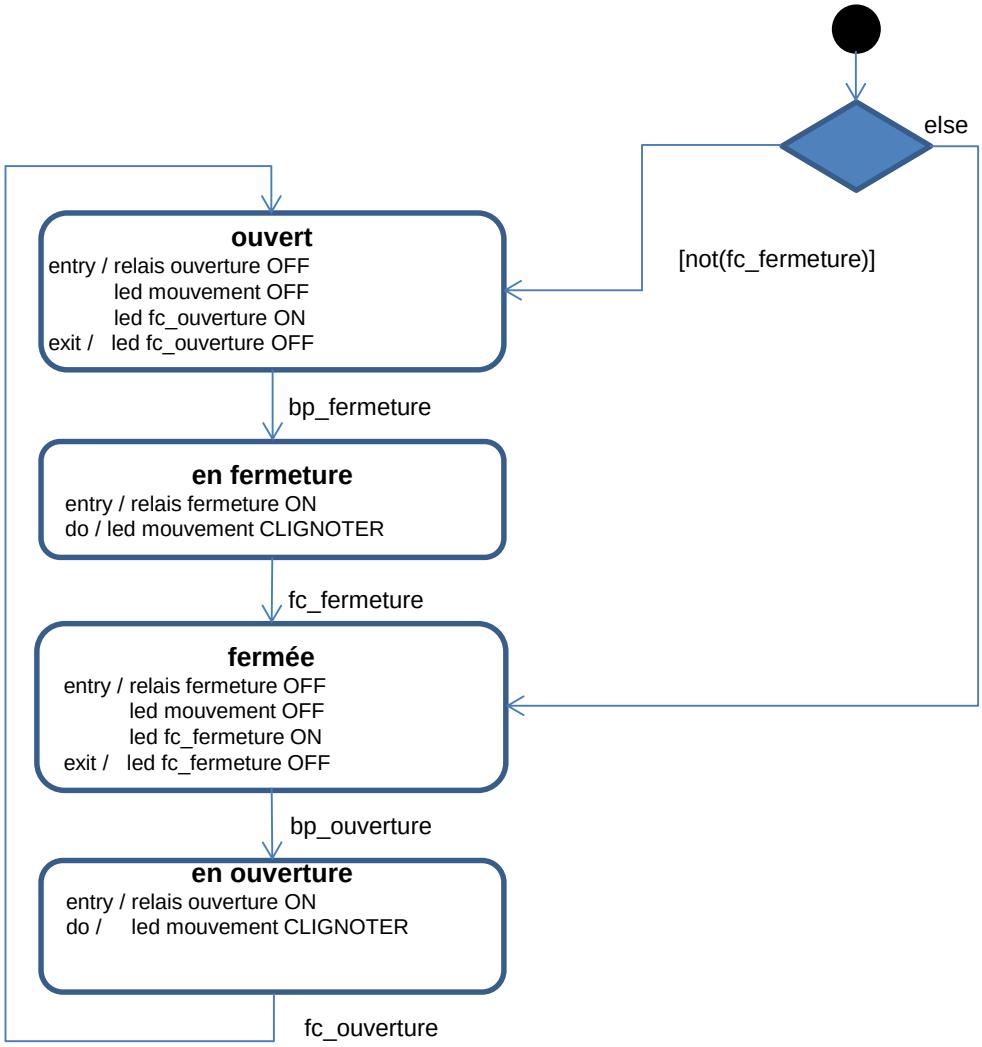

Figure 5 : graphe d'état 2

4 EXEMPLE 3 : SECURITE A LA FERMETURE

On souhaite intégrer une sécurité à la fermeture. Lorsque la barrière optique est coupée pendant la phase de fermeture, le portail doit se ré-ouvrir après une pause de 2 secondes. Pendant cette pause, la led caractérisant le mouvement A0 doit rester allumée.

Le tableau suivant donne la nouvelle entrée à intégrer au projet.

type	E/S	Borne arduino	alias	classe	instance	commentaires
Entrée câble	*E*	*5*	*OPTO*	*Event*	*opto*	*24V via opto-coupleur*

Le programme **GTI3.ino**, correspond au nouveau cahier des charges.

5 EXEMPLE 4 : OUVERTURE PIETONNE

Dans cette partie nous allons intégrer une ouverture partielle pour les piétons. Pour cela, nous allons compter le nombre de tours du pignon du système d'entrainement pignon/crémaillère.

Le système est doté d'un capteur inductif au niveau du pignon. Ce capteur inductif est activé par la présence de la tête de vis du pignon. Le graphe présenté figure 6, intègre un compteur qui s'incrémente à l'ouverture et qui se décrémente à la fermeture. Le compteur est mis à 0 lorsque le portail est fermé.

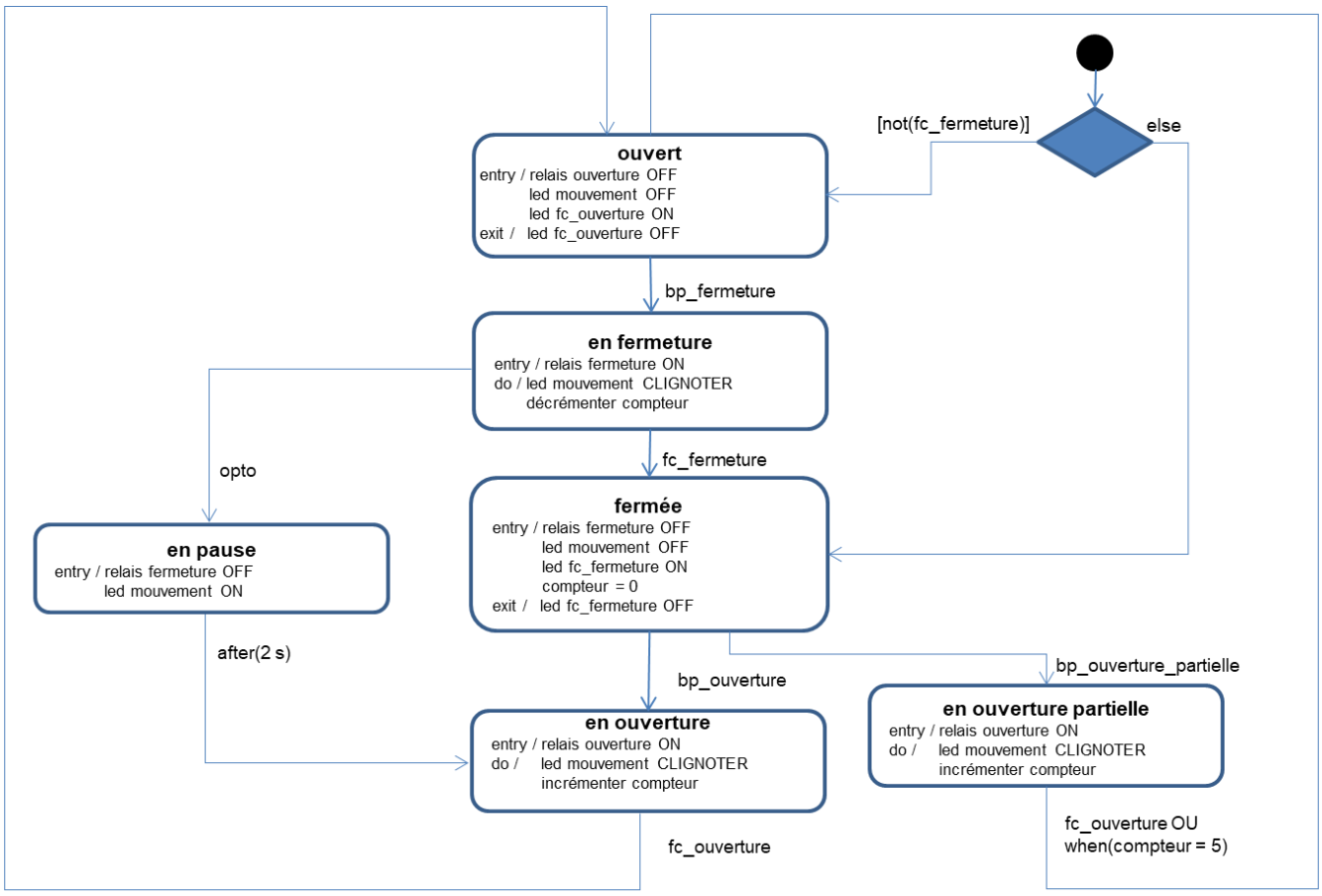

Figure 6 : graphe d'état 4 (complet)

Le tableau suivant donne la nouvelle entrée à intégrer au projet.

type	E/S	Borne arduino	alias	classe	instance	commentaires
Entrée câble	*E*	*7*	*INDUCTIF*	*Event*	*inductif*	*24V via opto-coupleur*
Bouton poussoir	*E*	*A10*	*BP_OUVERTURE_PIETON*	*Button*	*bp_ouverture_pieton*	*Bouton grove*

En pratique, on constate que parfois, lorsque la tête de vis passe devant le capteur inductif, il y a 2 fronts et parfois un seul ce qui nuit au comptage. Ce problème provient du fait que la tête de la vis est creuse. Pour régler le problème, une solution consiste à utiliser un *timer* qui mesure le temps écoulé depuis le dernier front montant du capteur inductif. Si un nouveau front apparait alors qu'il s'est écoulé moins de 1 seconde, on n'actualise pas le compteur (objet de classe *Timer*).

Pour déclencher l'ouverture piéton, un utilise le bouton poussoir A10. L'ouverture piéton correspond à 5 tours de pignon.

Le programme **GTI4.ino**, correspond au nouveau cahier des charges.