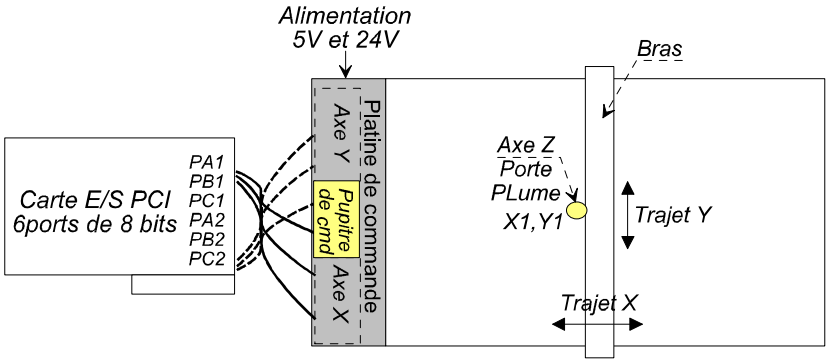
COMMANDE D'UNE TABLE XYZ : PRESENTATION

Objectifs :

Réaliser une commande en temps réel des axes d'une table XYZ suivant des trajectoires quelconques. Les trajectoires (X,Y) sont matérialisées par un tracé à l'aide d'une plume actionnée par l'axe Z.

1. PRESENTATION GENERALE

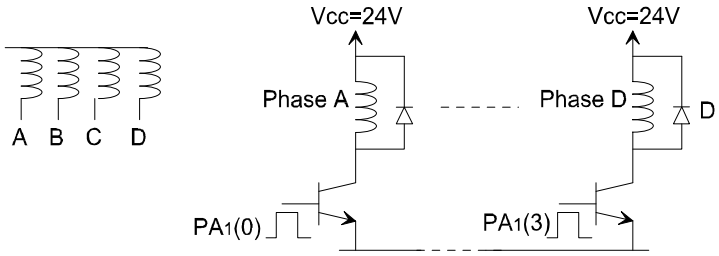
1.1 SCHEMA DE LA MAQUETTE



Le bras et le porte-plume sont mus par deux moteurs pas à pas (MPAP) 4 phases alimentés en 24V. La plume est commandée grâce à un électro-aimant.

1.2 PRINCIPES DE FONCTIONNEMENT D'UN MOTEUR PAS A PAS

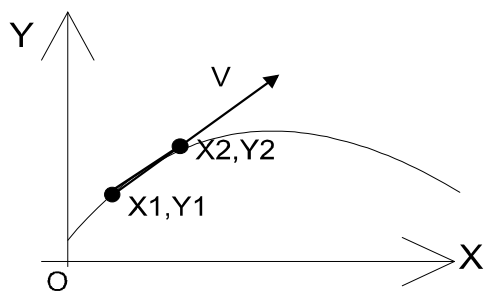
Les moteurs pas à pas utilisés sont référencés RTC ID 35014 (4 phases, 48 pas par tour).



En alimentant simultanément deux phases on augmente le couple moteur mais on perd sur la précision du positionnement.

- Commande en position : dans une séquence de commande AB, BC, CD, DA, AB etc.., le passage d'une commande à la suivante produit un déplacement d'un pas du MPAP (1/48^{ième} de tour).
- Commande en vitesse : une cadence T (en ms) de commande de phases (AB → délai T, BC → délai T, CD → délai T, etc..) correspond à une vitesse de rotation V=1000/T en pas par seconde.

1.3 PRINCIPE DU SUIVI DE TRAJECTOIRE



Pour respecter la géométrie des trajectoires, il faudrait respecter un facteur d'échelle sur chaque axe X et Y :

$$F_x = \frac{DistanceX\,(en\,mm)}{NbPasX} \qquad F_y = \frac{DistanceY\,(en\,mm)}{NbPasY}$$

Pour la table, Fx≠Fy. Ces facteurs d'échelle ne seront pas pris en compte dans le TP.

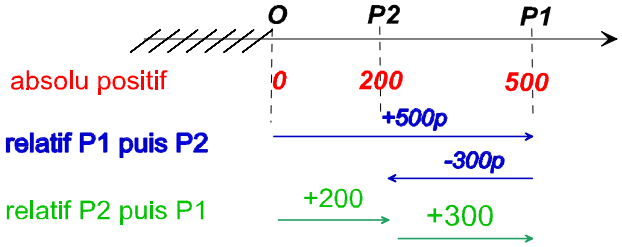
Pour obtenir une trajectoire donnée, il suffit de commander les axes afin d'obtenir chaque trajet rectiligne élémentaire composant la trajectoire.

Partant du point courant (X_1,Y_1), pour aller au point suivant (X_2,Y_2) en ΔT secondes il faut piloter :

1. un déplacement en X de X_1 à X_2 soit $N_x = X_2 - X_1$ pas en ΔT secondes. Cela représente une vitesse sur l'axe X de $V_x = \frac{N_x}{\Delta T}$ pas/seconde jusqu'à la position X_2. Le délai entre 2 commandes de phases du moteur X est donc $T_X = \frac{1000}{V_x}$ ms.
2. un déplacement en Y de Y_1 à Y_2 soit $N_y = Y_2 - Y_1$ pas en ΔT secondes. Cela représente une vitesse sur l'axe Y de $V_y = \frac{N_y}{\Delta T}$ pas/seconde jusqu'à la position Y_2. Le délai entre 2 commandes de phases du moteur Y est donc $T_Y = \frac{1000}{V_Y}$ ms.

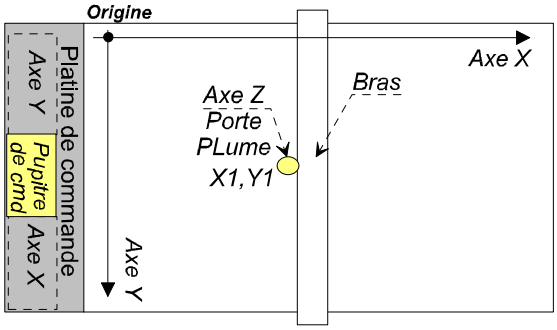
Le positionnement sur chaque axe est choisi de type ***absolu positif***.

Pour un axe : P ≥ 0



En relatif l'ordre du mouvement modifie les commandes pour atteindre un même point.

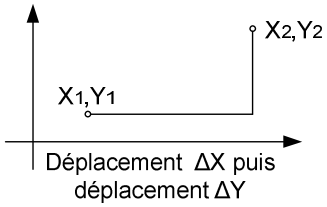
Pour la table : X ≥ 0 et Y ≥ 0

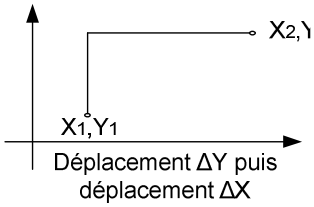


Chaque trajet rectiligne élémentaire consiste en deux couples (X, Vx) et (Y, Vy). X et Y représentent les positions absolues par rapport à une origine donnée. On a toujours la propriété : $\frac{X}{Vx} = \frac{Y}{Vy} = \Delta T$.

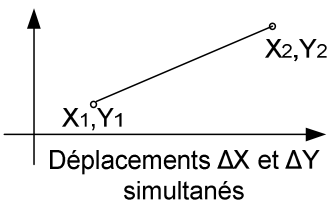
1.4 ARCHITECTURE DE LA COMMANDE

1.4.1 NECESSITE DU MULTITACHE



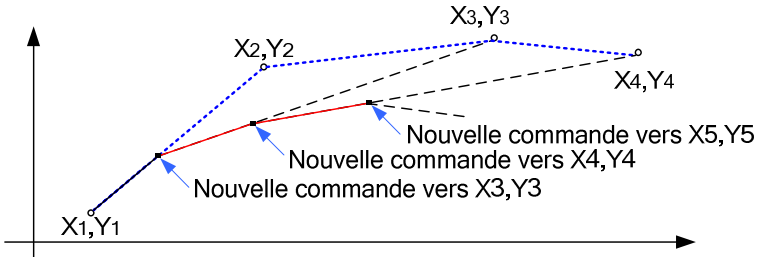


Pour chaque trajet rectiligne élémentaire, les deux déplacements doivent s'exécuter de manière simultanée pour respecter la géométrie du déplacement demandé.



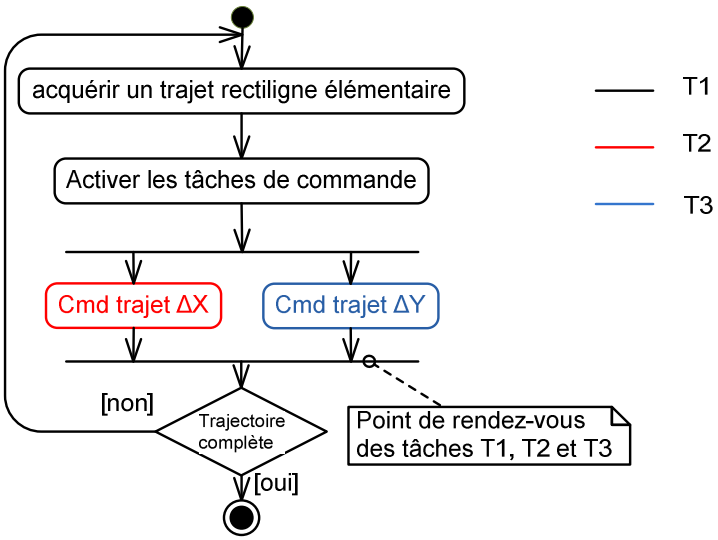
→ **La commande de la table est multitâche.**

1.4.2 NECESSITE DU RENDEZ-VOUS



Durant le suivi d'une trajectoire, la commande du trajet rectiligne élémentaire suivant ne doit pas être appliquée tant que le trajet rectiligne élémentaire précédent n'est pas fini.

→ **La synchronisation est de type rendez-vous.**



1.5 CONFIGURATION MATERIELLE DE LA MAQUETTE

Moteur X	Moteur Y	Axe Z	Pupitre de Commande
Phases : P_{A1} (bits 0 à 3) en S *LED :* P_{B1} (bit 0) en S	*Phases :* P_{A2} (bits 0 à 3) en S *LED :* P_{B2} (bit 0) en S	OUT_1 : P_{B1} (bit 1) en S	*IN_1 :* P_{C1} (bit 0) en E

- L'entrée IN_1 est utilisée comme départ de cycle (DCY) pour commander une trajectoire.
- La sortie OUT_1 est utilisée pour commander la plume (axe Z). Sa mise à 1 descend la plume.

L'accès au matériel se fait par une carte Advantech de 48 entrées/sorties parallèles organisées en 6 ports de 8 bits configurables en entrée ou en sortie séparément (voir « Librairie Carte 6 ports Advantech.chm » dans la mail). Cette librairie fournit une classe *CCarte6P*. Dans un objet de cette classe, les accès aux ports se font à partir du numéro du port, de 1 pour P_{A1}, 2 pour P_{B1} etc.. à 5 pour P_{B2} et 6 pour P_{C2}.

La configuration en entrée ou sortie des ports est passée aux constructeurs de la classe CCarte6P sous la forme d'une chaîne de 6 caractères, E pour entrée S pour sortie. **Ainsi pour la maquette la configuration des ports est « SSESSE ».**

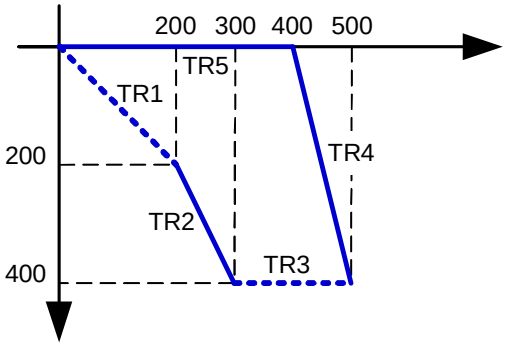
La classe CCarte6P est utilisable dans un contexte multitâche sans précautions particulières de la part de l'appelant car les méthodes de cette classe intègrent un mécanisme d'exclusion mutuelle lors des accès au matériel.

La classe Ccarte6P est disponible dans les librairies :

- *IODllCPPsc.dll* (*IODllCPPsc.lib* dans la mail) : les entrées sorties sont simulées par des enregistrements dans un fichier texte. Cette librairie « **S**ans **C**arte » est utilisable sur le poste de développement.
- *IODllCPP.dll* (*IODllCPP.lib* dans la mail) : cette librairie n'est utilisable que sur la cible disposant de la carte Advantech.

2. CAHIER DES CHARGES

Il s'agit de piloter les moteurs et l'électroaimant de la table XYZ de manière à obtenir le tracé d'une trajectoire définie par une succession d'au moins deux trajets rectilignes élémentaires.
La validation finale de l'application se fera sur la trajectoire suivante.



Les trajets en traits pleins se font plume baissée
Les trajets en traits pointillés se font plume levée.

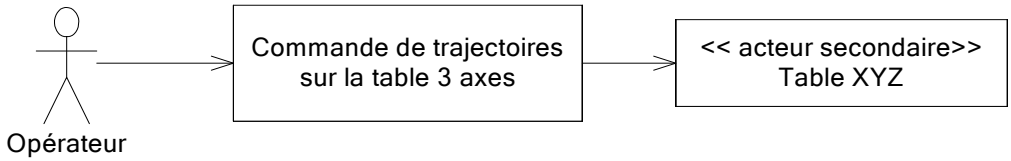
```
TR1 : X=200  Vx=20  Y=200  Vy=20
TR2 : X=300  Vx=10  Y=400  Vy=20
TR3 : X=500  Vx=40  Y=400  Vy=0
TR4 : X=400  Vx=10  Y=0    Vy=40
TR5 : X=0    Vx=20  Y=0    Vy=0
```

- Une trajectoire commence par un « départ cycle » (interrupteur DCY sur ON).
- Une trajectoire se termine avec le dernier trajet rectiligne élémentaire.

3. MODELISATION FOURNIE

3.1 CAPTURE DU BESOIN

3.1.1 CONTEXTE DE L'APPLICATION



Un acteur principal : l'opérateur

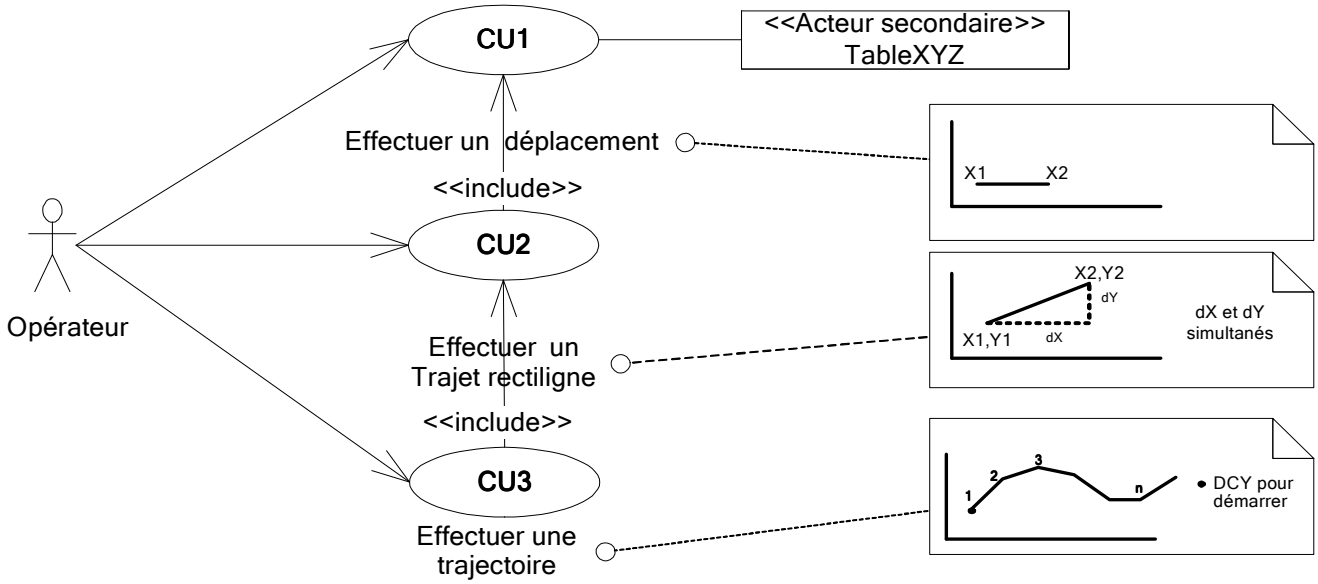
Il effectue les saisies informatiques et les actions sur la PO en vue d'obtenir les fonctionnalités décrites par les cas d'utilisation.

Un acteur secondaire : La table 3 axes XYZ

Elle reçoit des commandes et fournit des états (voyants) via les ports de sortie et d'entrée de la carte 6 ports.

3.1.2 DIAGRAMME DES CAS D'UTILISATION DE " COMMANDE DE TRAJECTOIRES SUR LA TABLE XYZ "

Le travail à réaliser est découpé en cas d'utilisation techniques permettant d'aborder progressivement les difficultés de conception et de codage.



3.2 ANALYSE GENERALE

L'analyse du problème a été faite dans les chapitres précédents.

TRAVAIL DEMANDE

Le travail demandé porte sur la modélisation UML, le codage et les tests unitaires :

- Toutes les modélisations demandées seront fournies en UML 2.0.
- Tous les projets sont des projets console C++ win32 utilisant le noyau multitâche objet MTR860BJ dans une solution *TableXYZ.sln*.
- La conception et le codage se feront cas d'utilisation par cas d'utilisation. Chaque cas d'utilisation pourra donner lieu à plusieurs incréments. Chaque incrément pourra être constitué de plusieurs itérations. Aucune itération ne devra être perdue. Les projets seront identifiés par :

CUn_INCi_ITj soit Cas d'utilisation n, incrément i, itération j

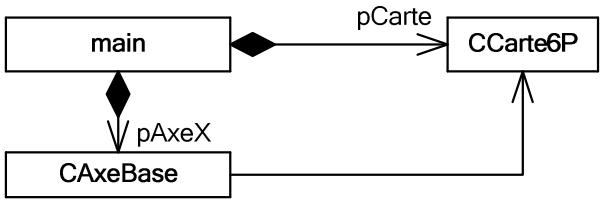
1 CAS D'UTILISATION CU1 « EFFECTUER UN DEPLACEMENT »

1.1 LA CLASSE CAXEBASE : PROJET CU1_INC1_IT1

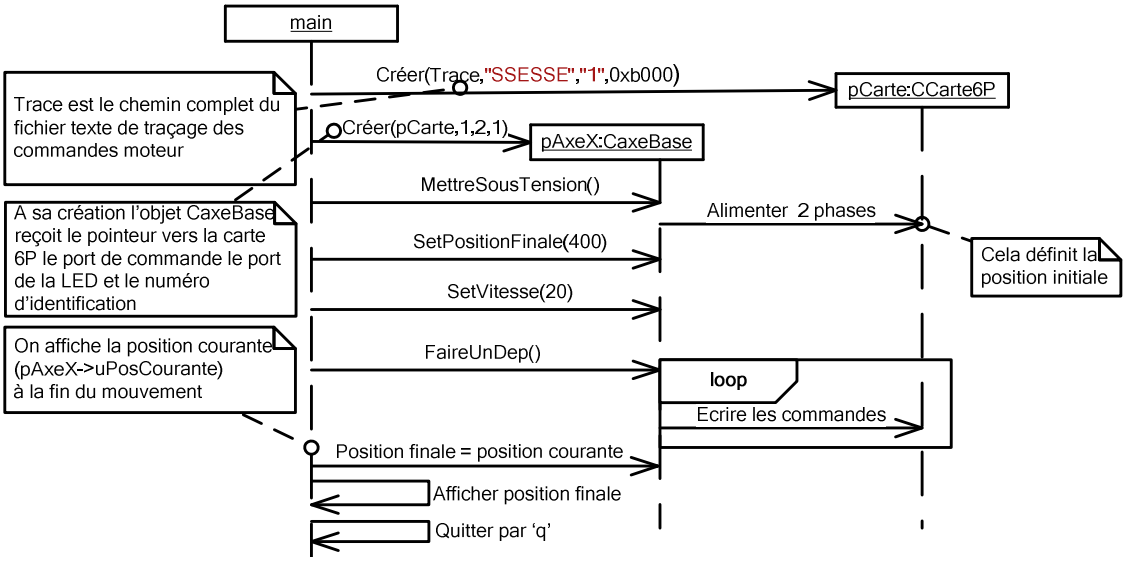
- Pour le poste de développement, on fournit dans une DLL *Caxebasesc.dll* (*CaxeBasesc.lib* et une documentation *CAxeBase.chm* dans la mail) une classe *CAxeBase* qui encapsule la commande d'un axe.
- Pour le test sur la cible on recompilera avec *CaxeBase.lib* afin d'utiliser *CAxeBase.dll*.

La commande d'un déplacement en X de 400 pas à la vitesse de 20pas/s à l'aide de la classe *CaxeBase* se modélise comme suit.

Diagramme des classes participantes (DCLA) de « Effectuer un déplacement » (DSEQ)



Séquence nominale de « Effectuer un déplacement » (DSEQ)



- « Trace » est une chaîne nommant le fichier qui recevra la trace de toutes les opérations d'entrée sortie sur la carte 6 ports.

- "SSESSE" est la chaîne de configuration des 6 ports de la carte.
- "1" et 0xE400 sont les données d'installation de la carte Advantech utilisée.

On donne dans la mail une solution *tableXYZ.sln* contenant un projet *CU1_INC1_IT1*.

1. **Compléter la fiche de test unitaire fournie *TU1-CU1_INC1_IT1.doc***
2. **Compiler pour le poste puis pour la cible et tester le programme fourni. Renseigner la fiche de test unitaire.**

Remarques : simulation du fonctionnement matériel → traceur graphique `CTraceur`

```
// créer un traceur graphique pour la simulation
CTraceur* lpTraceur = new CTraceur(-1);
// pour le traçage graphique : relier la grandeur à afficher
CTraceur::InitVoies(&(pAxeX->uPosCourante),H);  // axe 1 =  axe horizontal
```

Note : le traceur graphique, outil de test, n'apparaîtra pas dans les diagrammes UML.

1.2 DEPLACEMENT Y → PROJET CU1_INC1_IT2

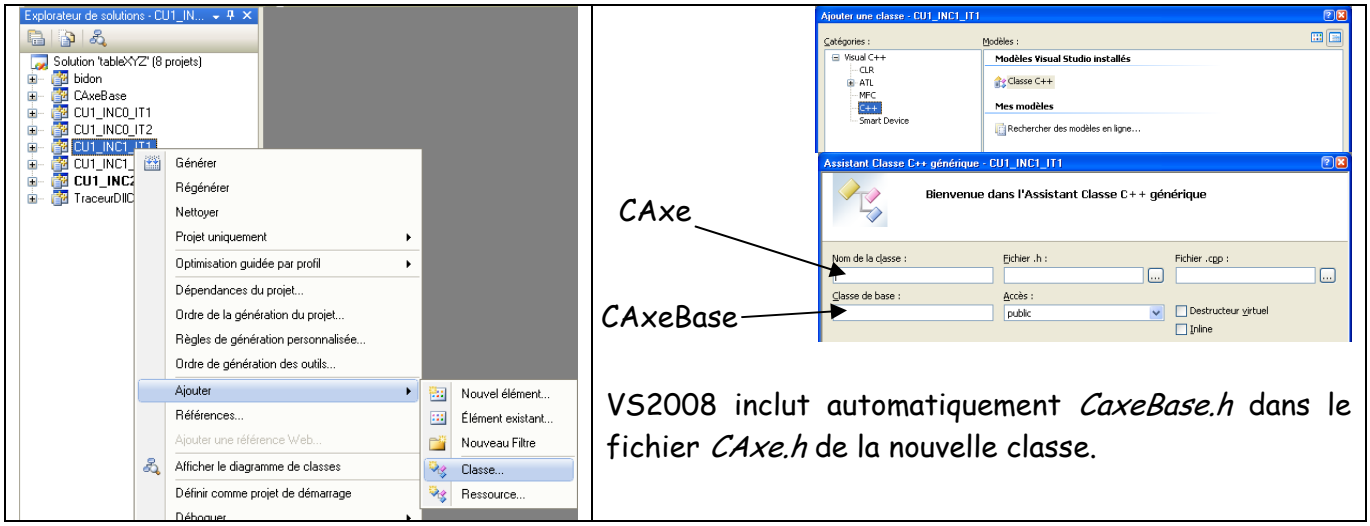
1. **Donner le DCLA et le DSEQ dans le cas d'un déplacement en Y de la position actuelle (supposée l'origine des déplacements) à la position 200 pas à la vitesse de 40 pas/s.**
2. **En modifiant le programme fourni précédemment, coder puis tester ce déplacement en Y avec affichage de la position finale.**

2 CAS D'UTILISATION CU2 « EFFECTUER UN TRAJET RECTILIGNE »

2.1 CLASSE ACTIVE CAXE → PROJET CU2_INC1_IT1

On a vu que la commande des axes doit être multitâche pour permettre les déplacements simultanés en X et Y. La commande d'un axe doit être encapsulée dans une classe active.

1. **Donner dans un diagramme de classes la conception d'une classe *CAxe* active dérivant à la fois de CAxeBase pour les fonctionnalités de commande d'axe et de la classe *Tache* de MTROBJ pour les fonctionnalités du multitâche.**
2. **Coder la classe active CAxe à l'aide des assistants de Visual Studio 2008.**



VS2008 inclut automatiquement *CaxeBase.h* dans le fichier *CAxe.h* de la nouvelle classe.

Compléter *CAxe.h* et *CAxe.cpp* afin que la classe *CAxe* dérive également de *Tache*.

```
Class CAxe : public CAxeBase, public Tache {
  // les déclarations supplémentaires
  // main : code de la tâche -> FaireUnDep
  virtual public void main (void*);
};
```

3. **Tester la classe active *CAxe* pour un déplacement de la position actuelle X=0 à la position X1=400 à 20pas/s. Compléter avec l'affichage de la position finale.**

Pour cela, modifier le code du *main* de *CU1_INC1_IT1* afin de l'adapter à la classe active CAxe.
L'opérateur attend la fin du trajet avant de fermer l'application en tapant la touche 'q'.

2.2 DEPLACEMENT EN Y AVEC CAXE → PROJET CU2_INC1_IT2

1. **Tester maintenant un déplacement en Y de la position actuelle supposée l'origine Y=0 à la position Y1=200 à 40pas/s. Afficher la position finale atteinte.**

2.3 COMMANDE D'UN TRAJET RECTILIGNE ELEMENTAIRE → PROJET CU2_INC2_IT1

A l'aide de la classe *CAxe* programmer un trajet rectiligne de la position actuelle supposée l'origine O(0,0) à la position M(400,200) en 10 secondes. On affichera la position atteinte en X et en Y en fin de déplacement.
L'opérateur attend la fin du trajet rectiligne avant de fermer l'application en tapant la touche 'q'.

1. **Calculer les consignes en X (X1,Vx1) et en Y (Y1,Vy1)**
2. **Donner le diagramme des classes participantes en précisant les classes actives.**
3. **Donner le diagramme d'activité du programme demandé**
4. **Programmer et tester le trajet rectiligne élémentaire**

```
// pour le traçage graphique : relier les grandeurs à afficher
CTraceur::InitVoies(&(pAxeX->uPosCourante),H);  // axe 1 =  axe horizontal
CTraceur::InitVoies(&(pAxeY->uPosCourante),V);  // axe 2 =  axe vertical
```

3 CAS D'UTILISATION CU3 « EFFECTUER UNE TRAJECTOIRE »

3.1 ENCHAINEMENT DE 2 DEPLACEMENTS → PROJET CU3_INC1_IT1

On souhaite enchaîner deux trajets rectilignes de O(0,0) à M_1(200,400) en 10s puis de M_1 à M_2(400,0) en 10s. Le second déplacement ne peut être commandé tant que le premier n'est pas terminé. Il y a un rendez-vous (voir diagramme d'activité du § 1.4.2) entre :

- Les commandes en X et Y qui se font dans le *main* du *CU2_INC2_IT1*
- La tâche de commande de l'axe X
- La tâche de commande de l'axe Y

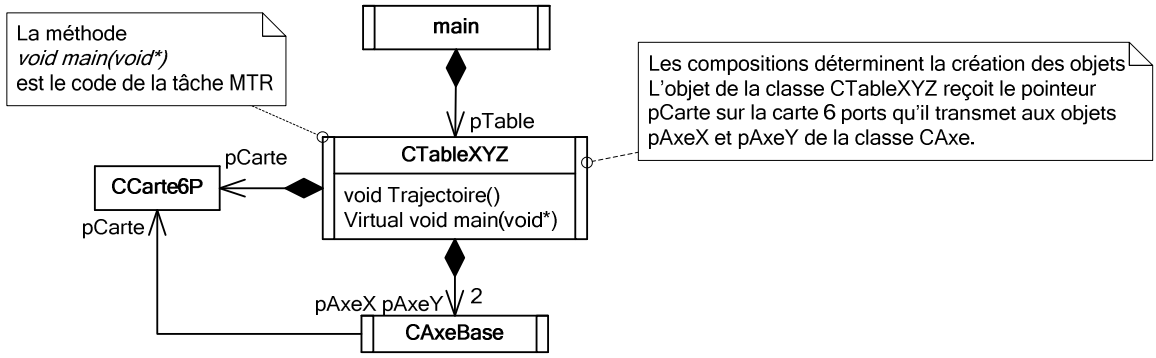
Les objets *RendezVous* de MTR86OBJ ne sont accessibles qu'aux tâches MTR. Il faut donc que la commande des deux axes (ancien *main*) se fasse dans une tâche MTR86OBJ.

Pour cela, on donne une classe *CTableXYZ* qui hérite de la classe *Tache*. C'est cette classe qui sera responsable d'appliquer les consignes de déplacements et de lancer les commandes d'axes dans une méthode appelée *Trajectoire()*. Le programme se termine lorsque l'opérateur entre le caractère 'q'.

Le pseudo code de la méthode *trajectoire()* est :

```
ALGO Trajectoire DEBUT
  Renseigner les déplacements X et Y pour le premier trajet rectiligne
  Activer les tâches de commande des axes X et Y
  Attendre la fin des déplacements en se mettant en rendez-vous
  Afficher la position finale en X
  Afficher la position finale en Y
  Renseigner les déplacements X et Y pour le second trajet rectiligne
  activer les tâches de commande des axes X et Y
  Attendre la fin des déplacements en se mettant en rendez-vous
  Afficher la position finale en X
  Afficher la position finale en Y
  Terminer le programme en fermant MTR (mtr86exit et retour du wait_for_terminate).
FIN
```

Le DCLA suivant donne les classes participantes de l'incrément.



1. **Compléter le DCLA précédent en incluant le rendez-vous (classe *RendezVous*).**
2. **Donner le diagramme de la séquence nominale correspondant au pseudo-code de *trajectoire()***
3. **Donner les modifications de code dans la classe *CAxe*.**
4. **Donner le code des méthodes *CTableXYZ::Trajectoire()* et *CTableXYZ::main()***
5. **Coder le programme principal *main* et tester l'enchaînement demandé :**
   a. En commentant le code associé à `Attendre la fin des déplacements en se mettant en rendez-vous`. Que se passe t il ? Expliquez.
   b. En rétablissant ce code. Expliquez.

3.2 TRAJECTOIRE SIMPLIFIEE → PROJET CU3_INC2_IT1

Une trajectoire simplifiée :

- se fait toujours plume levée
- est une succession de trajets rectilignes élémentaires
- se termine avec le dernier trajet rectiligne.

1. **Proposer une classe *CTrajet* pour encapsuler les propriétés d'un trajet rectiligne.**

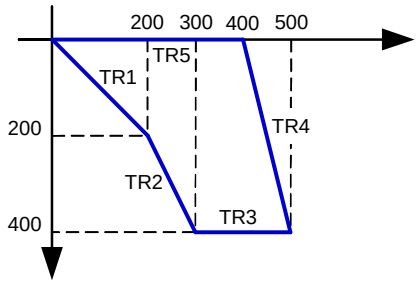
   Une trajectoire peut alors être implémentée comme un tableau de trajets rectilignes dont la taille serait le nombre de trajets rectilignes dans la trajectoire.

   - **L'indice 0 correspond au premier trajet.**
   - **L'indice N-1 correspond au dernier trajet donc à la fin de la trajectoire.**

```
CTrajets Trajectoire[N] ; // La trajectoire comporte N trajets rectilignes
```

Un tableau d'objets *CTrajet* sera créé dans le programme principal et passé à l'objet *CTableXYZ* lors de sa construction.

2. **Compléter le DCLA du §3.1 1.**
3. **Donner la séquence nominale de l'incrément *CU3_INC2_IT1*.**
4. **Modifier le code de la classe *CTableXYZ* afin que le programme effectue l'ensemble d'une trajectoire de N trajets rectilignes reçue par son constructeur.**
5. **Tester avec la trajectoire suivante.**




```
TR1 : X=200  Vx=20  Y=200  Vy=20
TR2 : X=300  Vx=10  Y=400  Vy=20
TR3 : X=500  Vx=40  Y=400  Vy=0
TR4 : X=400  Vx=10  Y=0    Vy=40
TR5 : X=0    Vx=20  Y=0    Vy=0
```

3.3 TRAJECTOIRE AVEC DEPART DE CYCLE → PROJET CU3_INC3_IT1

Il s'agit maintenant de gérer le départ de cycle donné par l'état du bit 0 du port 3 de la carte.

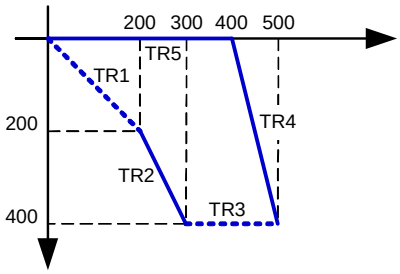
1. **Proposer une classe *CPupitre* encapsulant une méthode *bool dcy()* :**

```
SI BitDCY == 1 ALORS  // BitDCY est lu sur le bit 0 port 3 de la carte 6 ports
  dcy() retourne VRAI
SINON
  dcy() retourne FAUX
FINSI
```

2. **Tester la classe *CPupitre***

3.4 TRAJECTOIRE COMPLETE → PROJET CU3_INC3_IT2

La trajectoire est maintenant celle du §2 Cahier des charges soit



Les trajets en traits pleins se font plume baissée
Les trajets en traits pointillés se font plume levée.

```
TR1 : X=200  Vx=20  Y=200  Vy=20
TR2 : X=300  Vx=10  Y=400  Vy=20
TR3 : X=500  Vx=40  Y=400  Vy=0
TR4 : X=400  Vx=10  Y=0    Vy=40
TR5 : X=0    Vx=20  Y=0    Vy=0
```

1. **Comment faut-il modifier la classe *CTrajet* pour tenir compte de l'état de la plume lors des trajets rectilignes ? Modifier le code de la classe CTrajet.**

2. **Ajouter une classe *CPlume* permettant de commander la plume de la table avec la méthode *void CmdPlume (bool Abaisser):***

```
SI Abaisser == VRAI ALORS
  Alimenter l'électroaimant → envoyer 1 sur le bit associé (Bit 1 du port 2)
SINON
  Couper l'électroaimant → envoyer 0 sur le bit associé
FINSI
```

3. **Modifier le code de *CTableXYZ::Trajectoire()* afin de prendre en compte la gestion de la plume.**

4. **Tester la trajectoire complète du cahier des charges.**

Pour simuler le fonctionnement, on utilise maintenant un objet *CTraceurPlume*. La simulation représente la plume levée par un tracé en rouge et la plume baissée par un tracé en noir. Le code suivant effectue les initialisations nécessaires.

```
#include « M:\TP\iris2\TableXYZ\Ctraceur.h »    // Pour le traceur avec plume

// initialiser le traceur avec plume en X Y
CTraceurPlume* pTraceur = new CTraceurPlume(-1);
CTraceurPlume::InitVoies(&pAxeX->uPosCourante,H); // axe 1 =  axe horizontal
CTraceurPlume::InitVoies(&pAxeY->uPosCourante,V); // axe 2 =  axe vertical
CTraceurPlume::InitPlume((UINT*)&(UnDep.pDep()->PlumeBas)); // Etat de la plume
```