Projet UML

Distributeur de café

Yavuz AKIN Charles CHREIM Mario NOHRA

Distributeur de café à coder en C++

Sommaire

Introduction

Dans le cadre du cours UML du semestre 7 du cursus ISMIN, nous avons réalisé un projet de groupe nous permettant de mettre en avant nos capacités à lire, interpréter, faire, et traduire en code des diagrammes UML.

Sujet H. Distributeur de café à coder en C++

A partir de votre analyse des phrases suivantes :

1. Un distributeur de café automatique délivre du café court, long, sucré ou pas.
2. Deux qualités de café différents sont disponibles : 'normal' ou 'luxe'.
3. Le distributeur accepte les pièces de 2€, 1€, 0,5€.
4. Le café normal coûte 1,5€, quelque soit le volume et la présence de sucre ou pas.
5. Le café luxe coûte 2€, quelque soit le volume et la présence de sucre ou pas.
6. Le monnayeur n'accepte plus de pièces si la somme introduite dépasse 2€
7. Le distributeur peut rendre la monnaie après sélection du produit demandé. Il peut libérer chacune des valeurs de pièces.
8. Le distributeur attend la présence d'un crédit suffisant avant de servir le café
9. Deux boutons permettent de choisir la qualité du café
10. Deux autres boutons permettent de choisir la présence de sucre et s'il on veut un 'long'. Ces boutons doivent être actionnés avant le choix de la qualité du café.
11. Le distributeur est composé de l'IHM évoqué avec le monnayeur, et d'actionneurs qui sont :
    1. une vanne d'eau qui ouvre et ferme pour produire le café : 3s pour un court, 5s pour un long
    2. un verseur de sucre,
    3. un distributeur de gobelet
    4. un bloc de production que l'on actionne par les deux commandes logiques « versecaféNormal » ou « versecaféluxe »

Le projet consiste à réaliser un distributeur de café en C++. Nous réaliserons ce distributeur en suivant les consignes indiqués ci-après :

Afin de pouvoir réaliser ce projet nous allons d'abord élaborer un diagramme des cas d'utilisation, puis un diagramme de classe, puis de façon optionnelle nous allons nous intéresser au diagramme de séquence du système. Finalement nous mettrons en place les diagrammes d'état/transition et présenterons le code résultant en C++.

Hypothèses pour la simulation

Nous formulons plusieurs hypothèses afin de simplifier notre simulation :

- Tout d'abord notre machine à café dispose d'une quantité infinie de café et d'eau. Cela pourrait revenir à dire que le remplissage des stocks est fait de façon assez récurrente pour que les stocks ne soient jamais terminés. (Ce n'est pas le cas des stocks de sucre et de gobelets qui seront gérés dans notre code).

  Nous aurions pu aussi prendre en compte les quantités d'eau et de café, mais une machine est généralement reliée à une source d'eau et il est rare qu'elle soit à court de café.

- Notre machine ne connaît pas d'usure ou ne tombe pas en panne. Ce qui signifie que nous n'aurons pas à gérer les cas de dysfonctionnement des éléments de la machine.

Phase 1 : Elaboration d'un diagramme des cas d'utilisation

Tout d'abord nous nous intéressons au diagramme UML des cas d'utilisation afin de décrire notre projet.

Le projet considère la présence de 2 acteurs :

- le Client qui utilise la machine à café pour acheter du café. Il s'agit d'un acteur primaire donc il se trouve à droite de notre diagramme.
- le Responsable Maintenance qui s'occupe de réapprovisionner la machine en gobelets et en sucre. Il s'agit d'un acteur secondaire donc il se trouve à gauche de notre diagramme.

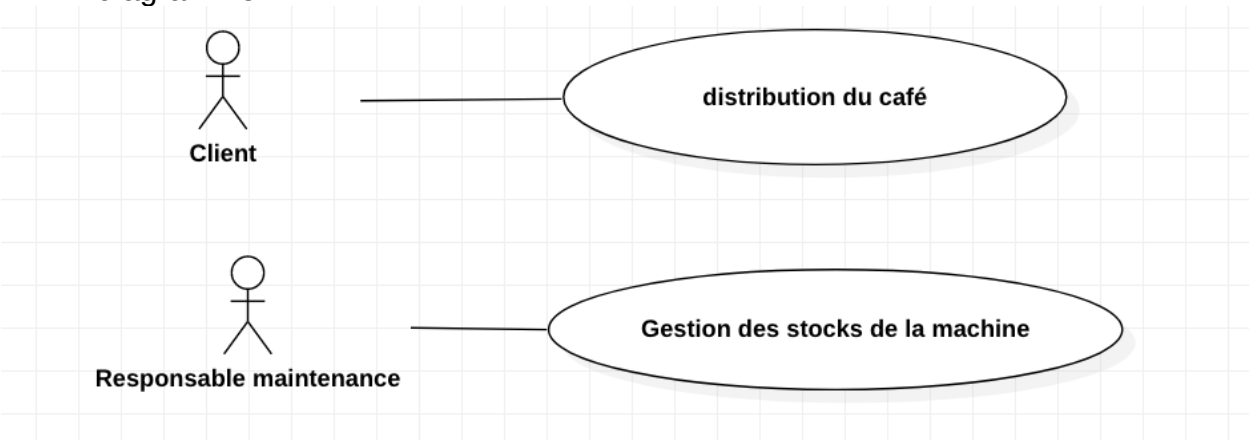

Figure 1: Phase 1 du diagramme de cas d'utilisation

De façon plus précise, le client a accès à plusieurs options lors du choix de son café. Il peut choisir le type de son café (court, long), la qualité de son café (normal et luxe) et s'il souhaite du sucre ou non.

Chacun des choix qu'il fait engendre la vérification des stocks (de sucre, de gobelets). En cas de stock non suffisants le Client ne peut effectuer son achat.

De plus un Responsable Maintenance s'occupe de restaurer les stocks de la machine.
Le client a le choix de l'ordre dans lequel il appuie sur les boutons (tout comme dans les vraies machines).

La Machine gère le paiement notamment lorsque celui-ci est insuffisant ou lorsqu'il faut rendre la monnaie.
La distribution du café se fait dans le bloc de production et est servie à l'aide de gobelets du distributeur de gobelets (avec occasionnellement du sucre ajouté).

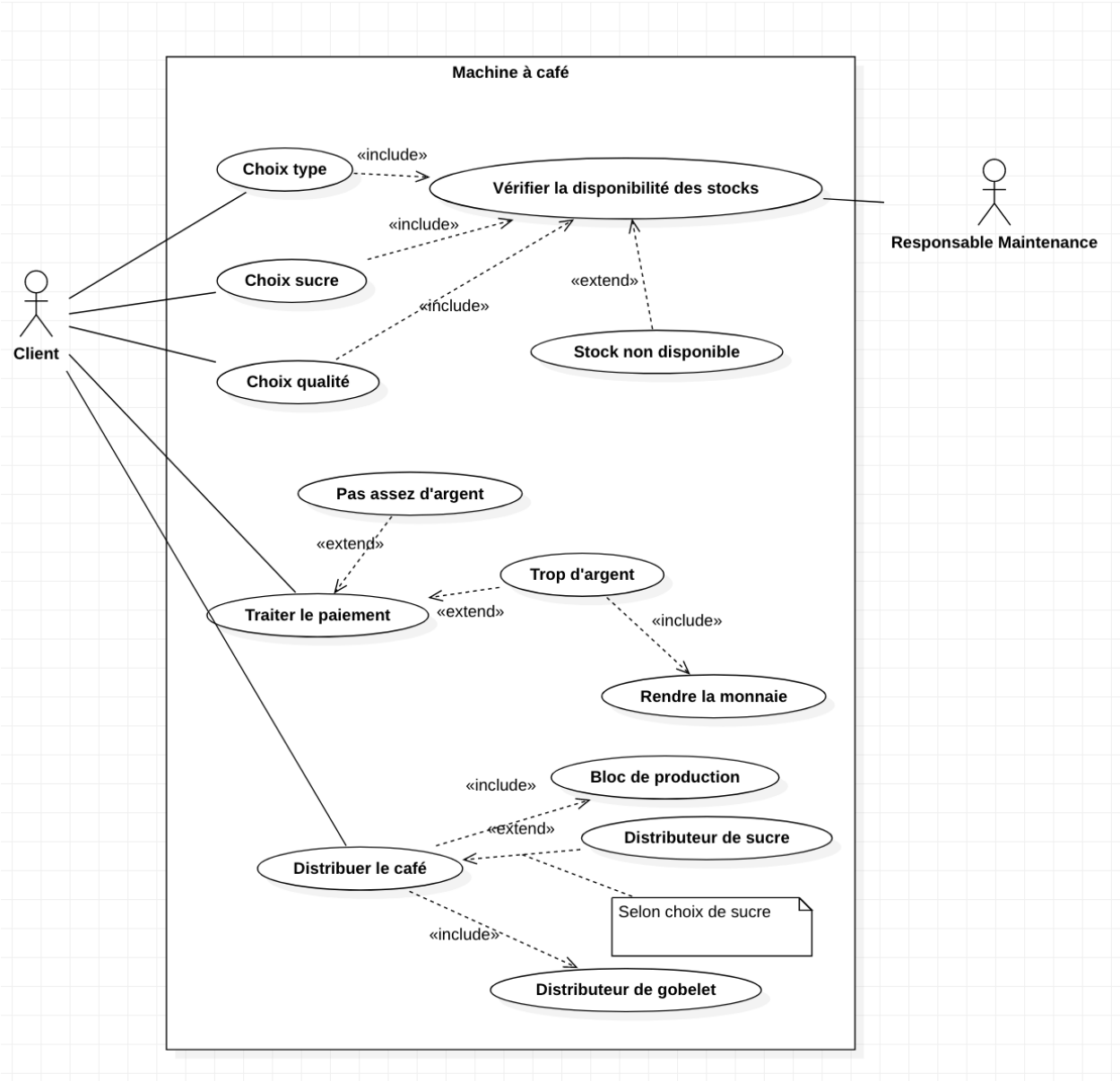

Figure 2: Phase final du diagramme de cas d'utilisation

Phase 2: Elaboration du diagramme de classe

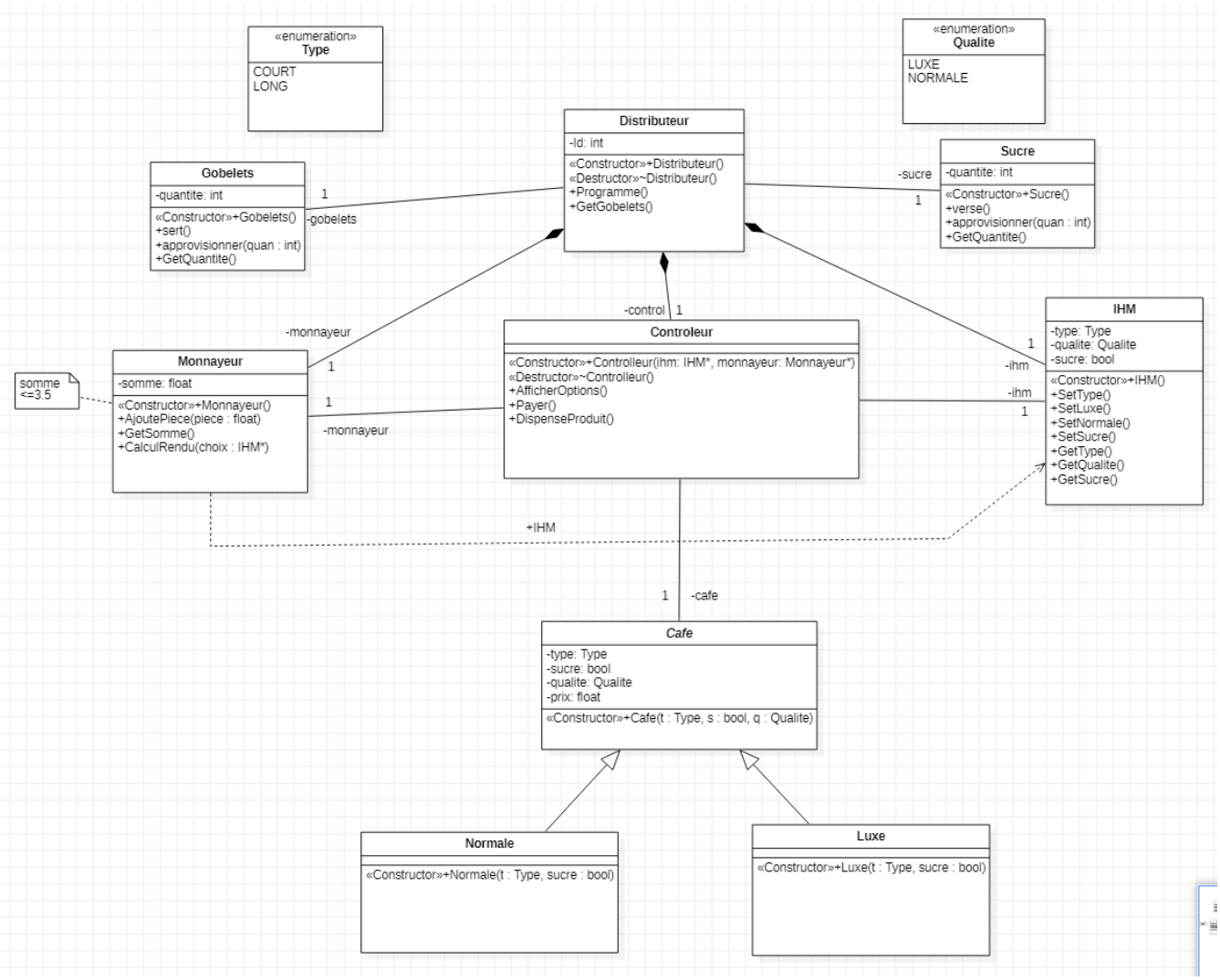

Figure 3: Le diagramme de classes

Pour réaliser une machine à café, il nous est demandé de réaliser un diagramme de classe qui décrit les classes utilisées pendant la simulation. Dans le paragraphe suivant, je vais vous décrire nos choix et les relations entre chaque classe.

Distributeur : c'est la classe centrale de notre système, elle est composée d'un contrôleur, une IHM, un monnayeur ainsi deux compartiments pour le sucre et les gobelets. Le distributeur contient comme attributs un numéro d'identification unique à chaque distributeur ainsi que 5 pointeurs pour les 5 classes associées à lui. En termes de méthodes, on a un constructeur, un destructeur pour détruire les composants ihm, contrôleur et monnayeur (nous ne détruisons pas les classes Gobelets et Sucre car nous avons décidé qu'il est réaliste de considérer ces classes indépendantes du Distributeur). Une fonction Getgobelets qui nous permet de vérifier la quantité

de gobelets restantes et la fonction programme qui simule le fonctionnement en utilisant toutes les classes disponibles.

Le type de relation qui lie Distributeur à Contrôleur, IHM et Monnayeur est une relation de composition. La raison de ce choix est que ces trois classes existent uniquement lorsque le Distributeur existe. Ainsi la classe Distributeur assume la création, la copie et la destruction de chacune des classes.

NOTE : Dans notre code C++, les classes Contrôleur, IHM, et Monnayeur sont données en pointeur en tant qu'attribut de la classe Distributeur bien qu'elles y soient liées par une relation de composition. La raison pour laquelle nous pouvons nous permettre cela est le fait que chacune de ses classes est attribut d'autres classes d'où l'appel par adresse de ces classes (voir cours p.309)

Gobelets et Sucre : deux classes qui nous permettent de suivre la quantité respectivement des gobelets et du sucre, de les approvisionner et de mettre à jour la quantité après service. Les constructeurs permettent d'initier la quantité. Ils sont associés au Distributeur pour que celui-ci puisse prévenir le Client en cas de rupture.

Contrôleur : cette classe est le cœur de notre système, elle traite toutes les opérations élémentaires comme l'affichage des options et le traitement des choix de l'utilisateur, le paiement, et la création du produit. Elle contient 3 pointeurs : monnayeur, ihm et cafe. Les deux premiers servent à manipuler l'ihm et le monnayeur et le pointeur cafe contient le café cree. En outre, elle contient un constructeur, un destructeur.

IHM : Cette classe permet à l'utilisateur de choisir son café, elle contient en attributs le type de café choisi (LONG ou COURT), la qualite (LUXE ou NORMALE) et un bool qui permet au client d'ajouter du sucre. Le constructeur par défaut initie le type à COURT, la qualité à NORMALE et le bool sucre à 0 (pas de sucre). On a 4 fonctions "Set" qui représentent les 4 boutons de la machine : un pour un café long, un pour le sucre, et deux pour choisir la qualité. Ainsi que 3 fonctions "Get" qui extraient ces informations.

Monnayeur : Cette classe traite l'opération de paiement. Elle a un float en attribut qui cumule les pièces insérées pour un achat et le constructeur initie la somme à 0. Add pièce ajoute une pièce à la somme et GetSomme permet d'accéder à la somme cumulée. CalculRendu reçoit en paramètre le pointeur ihm pour accéder à la qualité et par la suite connaître le prix pour calculer la somme à rendre, ensuite elle remet somme à 0 et rend une somme si nécessaire.

Cafe, Normale et Luxe : Cafe est la classe abstraite (en italique) qui contient tous les renseignements nécessaires pour décrire un café, la création d'un objet de cette classe symbolise la création du café. Les deux classes filles Luxe et Normale permettent de faciliter la création du produit et de sécuriser notre code, ainsi on ne peut pas facilement changer le prix d'un café luxe ou normal puisque le constructeur de Luxe or Normale impose le prix du café.

En outre on utilise deux énumérations pour choisir le type et la qualité du café, on trouve que les énumérations sont plus parlantes que des bools.

Phase Bonus : Elaboration du diagramme de séquence

Le diagramme de séquence n'est pas demandé par l'énoncé mais elle peut nous permettre de mieux comprendre le fonctionnement de notre simulation de façon séquentielle et chacune des actions exécutées par les classes que nous avons créées dans le modèle.

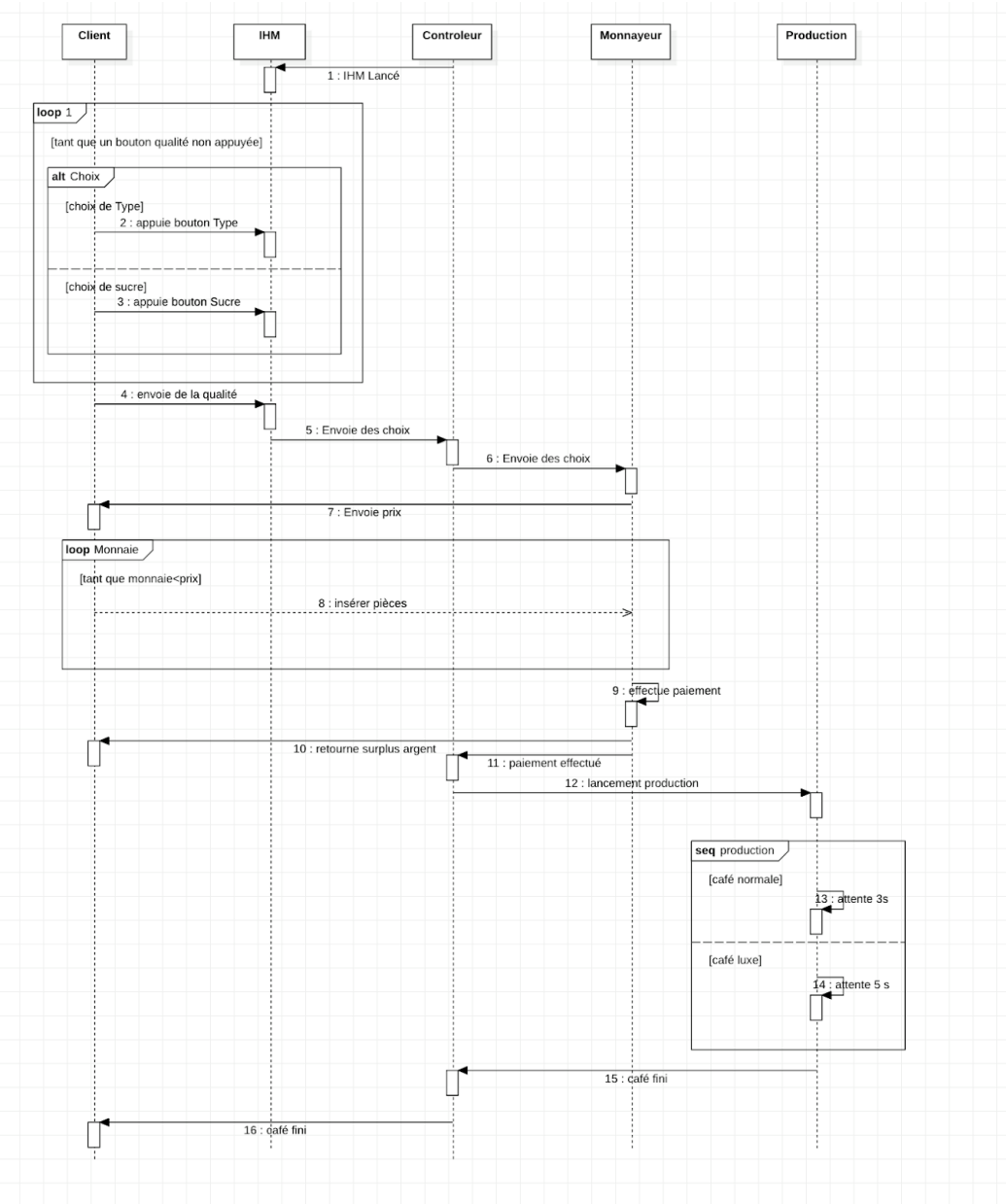

Figure 4 : Diagramme de séquence

Phase 3 : Elaboration des diagramme d'état/transition

Diagramme d'état du Monnayeur

Le diagramme dispose de plusieurs qui correspondent aux diverses sommes que le monnayeur est autorisé à contenir. En effet d'après les instructions données dans l'énoncé, seules les pièces de 0.5, 1, et 2 euros sont acceptées (d'ou les transitions qui n'acceptent que ces pièces), de plus on n'accepte plus de pièces une fois que le montant dans le monnayeur a dépassé 2 euros (d'où les boucles de retour dans certains états qui retournent l'argent entré dans le monnayeur).

Dans chaque état on affiche la quantité d'argent présent dans le monnayeur.

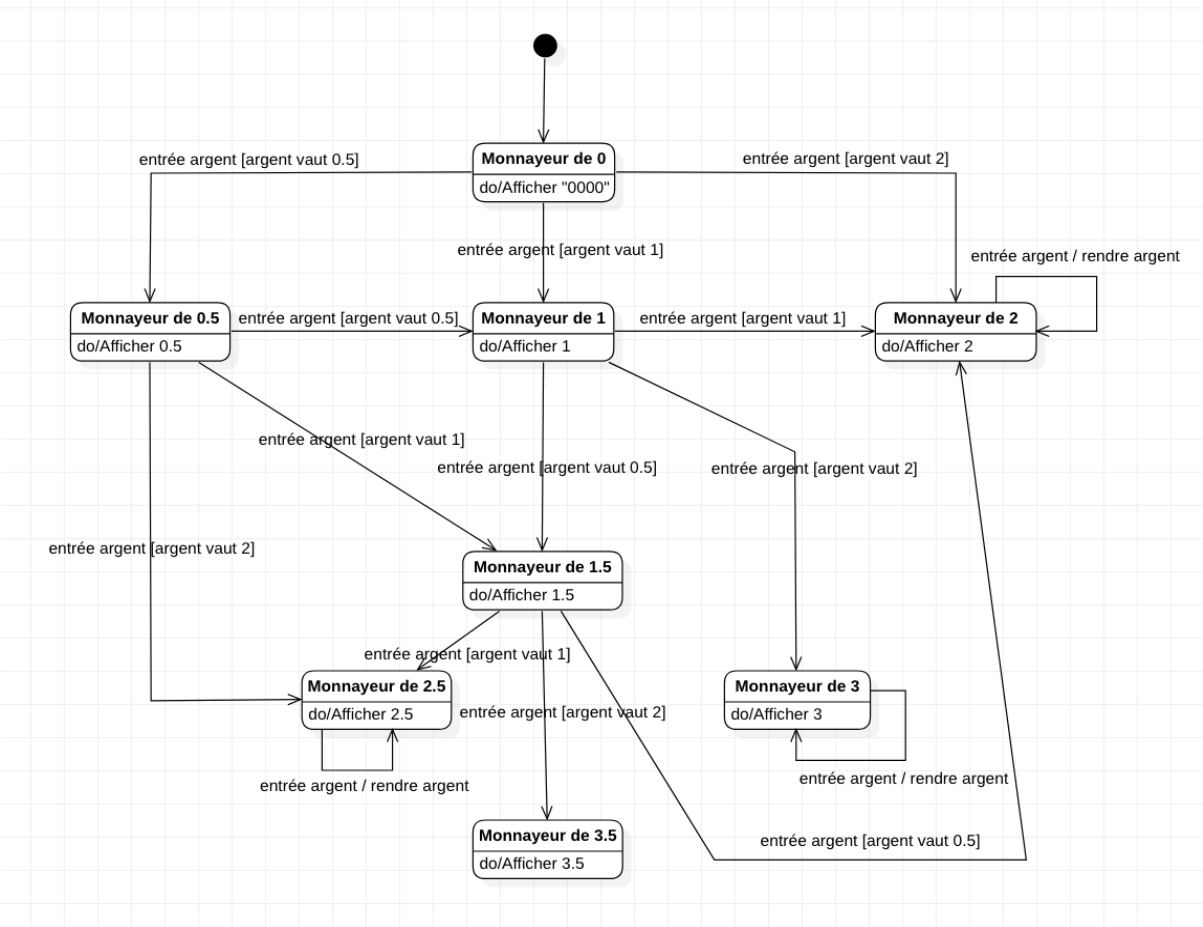

Figure 5 : Diagramme d'états du monnayeur

Diagramme d'état de la Sélection de café

La sélection de café est la première opération que le Client effectue.

Par défaut, la sélection du café est mise à court et sans sucre. L'utilisateur peut choisir d'avoir un café long ou sucré en appuyant sur leurs boutons respectifs. S'il veut de nouveau modifier les choix qu'il a fait, il n'a qu'à appuyer de nouveau sur le même bouton pour annuler le choix.

Les boutons Normal et Luxe permettent de choisir la qualité du café et lancent automatiquement la production du café si la quantité d'argent insérée est suffisante. Dans le cas contraire, la machine attend d'avoir assez d'argent pour la production.

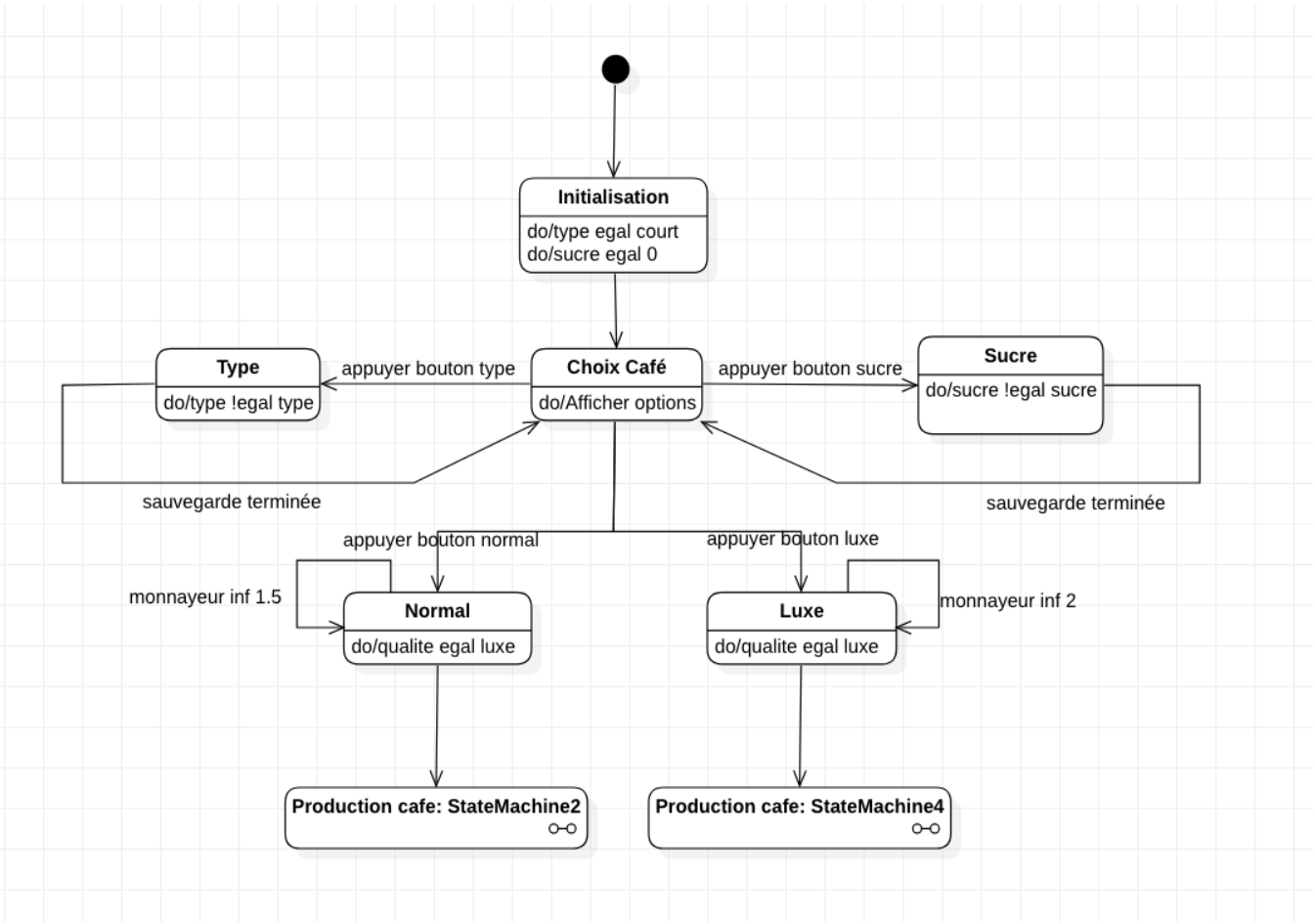

Figure 6 : Diagramme d'états de la partie sélection du café

Diagramme d'état de la production

Dans le bloc production, on produit le café après paiement. Le gobelet est d'abord ajouté. En fonction du type de café, la vanne d'eau est activée 3 ou 5 secondes puis est refermée. Ensuite est ajouté un sucre si souhaité et du café est versé en fonction de sa qualité.

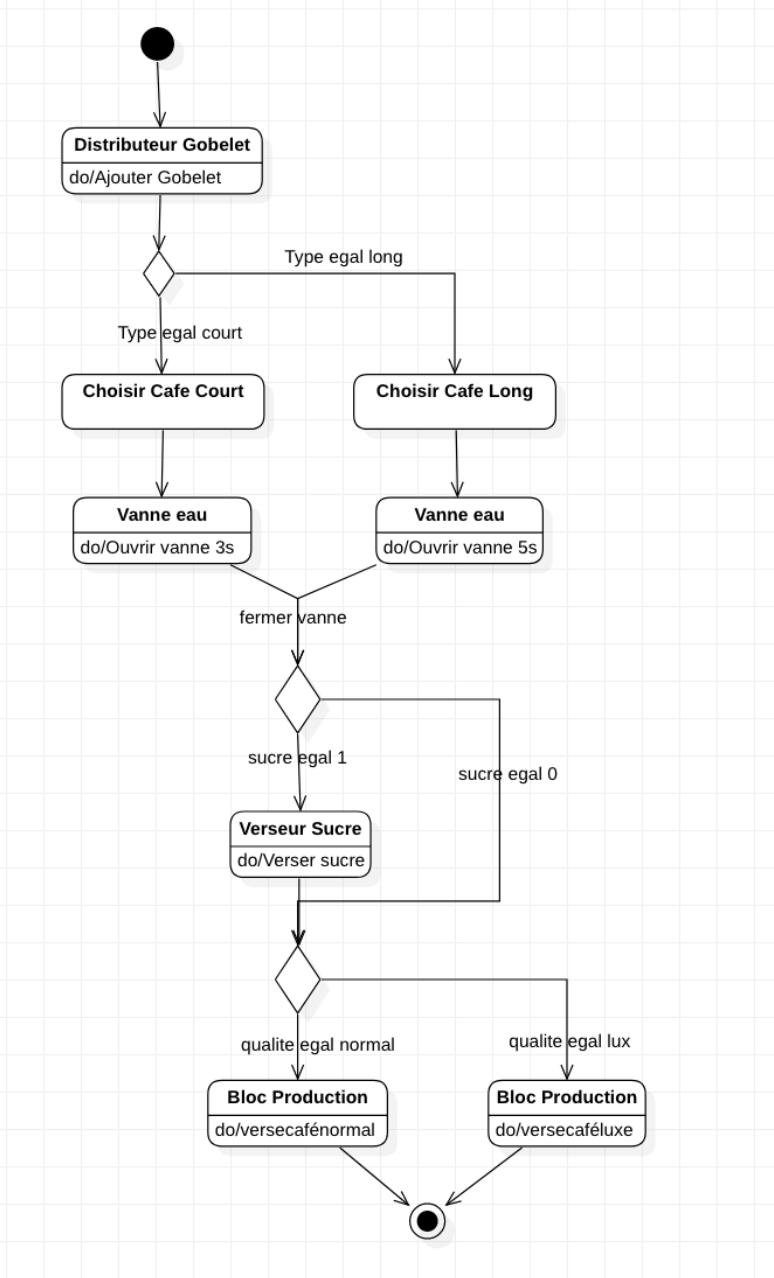

Figure 7 : Diagramme d'états de la partie production du café

Phase 4 : Codage en C++

Suite à la création du diagramme de classe sur StarUML, nous avons généré le code C++ avec l'extension dédiée à cette tâche. Ceci nous a permis d'avoir le squelette du code que nous avons complété. Avant de commencer le code, nous avons décidé de modéliser l'appuis sur les boutons par des entrées que l'utilisateur effectue sur le terminal (donc par des *cin* sur C++). Conformément aux attentes de l'énoncé, nous nous sommes limités à quatre boutons (l'un choisissant la présence de sucre ou non dans le café, l'un déterminant le type de café, et deux autres pour choisir entre normal ou luxe).

Cependant, nous avons décidé d'ajouter des fonctionnalités à ces boutons qui dépendent du moment où ils sont pressés. Nous avons, par exemple, décidé d'ajouter deux modes d'utilisation pour la machine. Le premier, le mode admin, permet au responsable maintenance de réapprovisionner les stocks de la machine (en gobelets et en sucre). Pour choisir ce mode, il suffit d'appuyer sur le bouton 0. Le deuxième mode est le mode d'utilisation "classique" qui permet à l'utilisateur de commander un café. Il faut cette fois appuyer sur le bouton 1 pour choisir ce mode. La Figure 7 permet de visualiser tous les cas d'utilisation de chaque bouton.

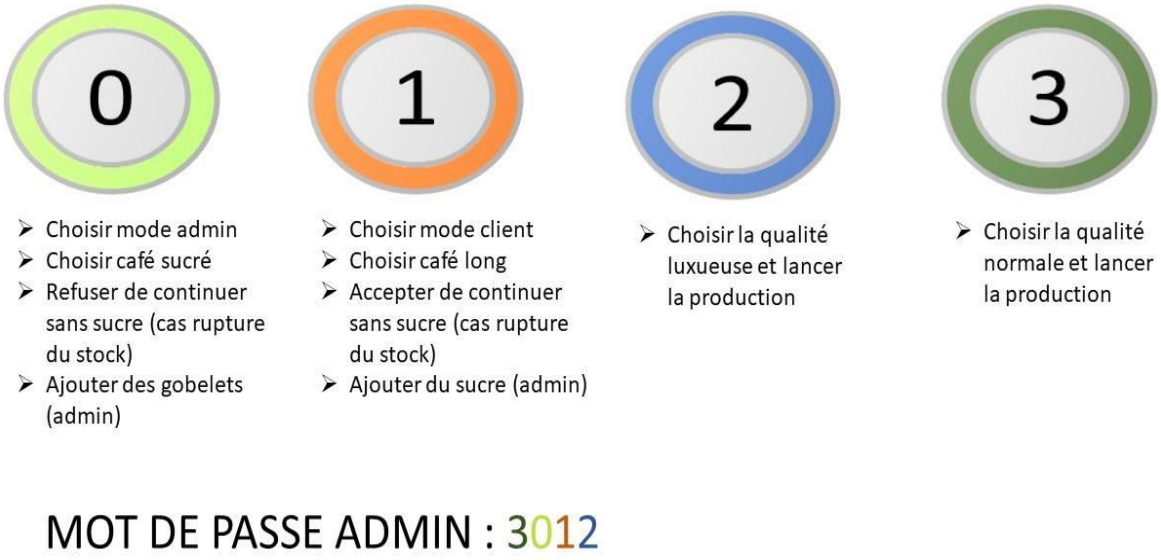

Figure 8 : Différents cas d'utilisation pour chaque bouton

Le code permet de simuler la commande d'un café personnalisé selon les trois critères donnés en énoncé (le type, la présence ou non de sucre, et la qualité).

Il permet aussi de gérer les stocks du distributeur de café :

En cas de rupture de gobelet, l'utilisateur est immédiatement informé (avant même de pouvoir personnaliser son café) que la commande de café est impossible. Il faut alors attendre qu'un admin vienne approvisionner la machine.

En cas de rupture des stocks de sucre, si l'utilisateur a commandé un café avec du sucre, deux options lui sont proposées. Il peut soit interrompre la production du café, dans ce cas la machine lui rembourse le prix de son café, soit continuer la production de son café sans sucre. Si l'utilisateur avait choisi un café sans sucre, il n'est pas informé de l'absence de sucre et la production de café se fait normalement.

Afin de simuler le temps durant lequel la vanne d'eau verse l'eau dans les gobelets (3s pour un café court et 5s pour un café long), nous avons inclus deux librairies de C++: chrono et thread. La première permet de définir une seconde et la deuxième permet d'utiliser la fonction sleep_for. Le programme affiche ainsi le compte à rebours dans le terminal en attendant que le café soit prêt.

A la fin du programme, une entité de la classe café est créé conformément aux options choisies par l'utilisateur. Pour simuler la distribution du café, nous utilisons *cout* pour afficher les spécificités du café sur le terminal.

Conclusion

En conclusion, nous avons réalisé notre distributeur de café en prenant en considération le diagramme de cas d'utilisation, le diagramme de séquence, le diagramme de classes et les diagrammes d'état/transition. Notre code fonctionne correctement et se rapproche du fonctionnement classique d'une machine à café.

Il aurait été intéressant de coupler ce cours avec celui de Programmation des Systèmes que nous avons reçu l'année dernière afin de pouvoir utiliser des threads au sein de la simulation qui nous permettraient d'effectuer plusieurs tâches de façon simultanée ce qui nous aurait permis d'être plus réaliste au niveau de la simulation mais aussi d'approches les diagrammes UML d'un autre angle.