TP_AP4A

Alexandre BRUNOUD - Jean-Charles CREPUT

Automne 2024

1 Présentation du projet

1.1 Introduction

Le TP de AP4A va prendre la forme d'un mini-projet, que vous allez devoir réaliser en solitaire, qui va vous permettre de mettre en pratique les connaissances acquises lors du cours ; il sera également un aperçu du projet Java, ainsi que ces attentes, que vous devrez rendre en fin de semestre.

Vous allez disposer de trois séances de trois heures chacune pour réaliser votre projet et étant donné la nature de l'UE, la difficulté de ce dernier sera croissante.

1.2 Mise en contexte

Votre projet a pour but la réalisation d'un simulateur, permettant de modéliser un écosystème IOT spécialisé dans le monitoring de qualité de l'air d'espace de travail.

Cet écosystème sera composé d'un serveur communiquant avec divers types de capteurs répartis au sein de l'espace de travail, tels que des capteurs de température, des capteurs d'humidité, des capteurs de lumière, et des capteurs sonores. De plus, les informations provenant des capteurs pourront être stockées sous forme de fichiers de logs ou encore partagées dans une console afin de les visualiser.

2 Les outils de développement

Vous utiliserez le standard C++17, qui est largement supporté et facile à compiler à la fois sur Linux et sur Windows. Concernant la compilation, vous utiliserez le compilateur GNU pour le langage C++, "g++". Sur Windows, il s'appelle MinGW.

Après avoir compilé un premier programme de type "hello world" par ligne de commande à la console, vous créerez un programme plus élaboré, avec un main.cpp, et une classe Server (voir ci-dessous) implémentée dans Server.cpp et Server.h, pour la mise en place et le test de vos outils de développement, compilation et exécution. L'utilisation des stations de travail de la salle de TP est à privilégier et vous pourrez installer vos IDE favoris sur vos machines personnelles.

Parmi les IDE que vous pourrez utiliser, on trouve par exemple qtcreator (par défaut sur Linux), devC++, clion, visual studio code, visual studio, etc. Cependant, pour des raisons de rendu du travail et donc de compilation et exécution multi-plateforme de vos programmes, à la fois sur Linux, Windows ou MacOS, il sera demandé d'utiliser un utilitaire de gestion de projet multi-plateforme, soit l'outil cmake associé à la plupart des IDE, dont le fichier de description de projet est CMakeLists.txt, soit l'outil qmake associé à l'IDE qtCreator, dont le fichier de description de projet est <projet>.pro. Le principe consiste en la génération automatique de Makefile pour la plateforme cible à partir d'une description du projet. Des exemples d'utilisation et de configuration seront présentés en TP. À vous de choisir l'IDE le plus simple et commode à utiliser permettant un transfert aisé des sources et une reconfiguration rapide en cas de changement de système. Dans tous les cas de figure, **un fichier CMakeLists.txt devra être intégré au rendu final, ou sinon un fichier <projet>.pro**.

3 Première séance

Lors de cette première séance, vous allez commencer à mettre en pratique les fondamentaux de la programmation objet de par son architecture, son écriture ainsi que sa compilation.

3.1 Votre premier programme

Pour votre premier programme, vous allez rédiger un grand classique de la programmation, le bien connu "Hello World !". Une fois ce dernier écrit, vous devrez le compiler, puis l'exécuter.

Concernant la compilation, vous utiliserez le compilateur GNU pour le langage C++, "g++". Vous devrez savoir nommer votre fichier de sortie, séparer la phase de compilation de la phase d'édition de liens, avec les options de compilation qui conviennent, puis vous devrez afficher les warnings. Pour savoir comment rédiger la commande g++, vous utiliserez la documentation Linux présente dans la console, le "man".

Une fois le travail effectué, vous le ferez constater par un professeur. Pour le reste de cette UE, vous pouvez faire appel aux IDE précédemment évoqués pour la compilation.

3.2 Le simulateur

Nous allons maintenant nous pencher sur le développement de notre simulateur. Comme expliqué précédemment, notre simulateur est composé d'un serveur (classe Server) ainsi

que d'un ensemble de capteurs (classe Sensor). Il comporte aussi une classe pour l'ordonnancement des exécutions (classe Scheduler).

Votre première tâche consistera à créer la classe "Server". Cette classe doit nous permettre de recevoir des données envoyées par l'ensemble des capteurs, de les analyser, de les formater, puis de stocker les informations utiles dans des fichiers de logs ou encore de les visualiser dans la console.

Pour cela, vous allez commencer par implémenter la forme canonique de Coplien de la classe et diverses fonctions, vues comme des services, pour la gestion centralisée des informations reçues des capteurs. On prendra des conventions standards pour les noms des fichiers, classes et variables, et la mise en forme du code. Vous écrirez deux fonctions :

- "*consoleWrite*" afin de visualiser les données reçues des capteurs dans la console.
- "*fileWrite*" pour stocker les données des capteurs dans des fichiers de logs (chaque type de capteur devra disposer de son fichier de logs). On fera attention au format des données écrites permettant au besoin leur chargement dans un tableur de type Excel.

Travail en autonomie.

À la fin de cette première séance, vous aurez normalement fini d'implémenter votre classe "Server" à travers son Coplien ainsi que certaines de ses fonctions.

Cependant, pour que vous ne restiez pas sans pratiquer jusqu'à la prochaine séance, il vous est conseillé d'organiser votre projet de manière structuré afin d'assurer une compréhension claire de son architecture. Cela inclut la gestion des fichiers et la mise en place de la structure de base des autres classes, comme Scheduler et Sensor.
Pour les différentes classes créées vous ajouterez des opérateurs friend d'entrée/sortie << en commençant par la classe Server. Deux modes d'utilisation devront être testés vous permettant de rediriger le flux de sortie, soit vers la console, soit vers un fichier de log.

4 Deuxième séance

Lors de cette deuxième séance, vous allez mettre en place l'architecture de votre projet. Vous pourrez commencer par implémenter le système global sans référence aux mécanismes d'héritage dans un premier temps, notamment si ceux-ci n'ont pas été encore abordés en cours et en TD. Vous considerez dans ce cas les classes Server, Scheduler, Sensor, en leur attribuant un fonctionnement par défaut. Les mécanismes d'héritage et de polymorphisme seront introduits dans un deuxième temps, donnant toute sa généricité au système complet.

Vous commencerez votre séance par une introduction à l'architecture logicielle et une mise en pratique, afin de commencer à prévoir le fonctionnement global et le programme final.

Il vous sera ensuite proposé un schéma UML que vous aurez à compléter, de manière individuelle. Cela vous permettra de mieux visualiser l'architecture du projet demandé.

Vous allez maintenant créer les deux nouvelles classes indiquées sur votre diagramme de classe : la classe "Sensor" et la classe "Scheduler".

4.1 Sensor

La classe "Sensor" représente la classe mère sur laquelle se basent les classes de vos capteurs. Les capteurs sont des processus autonomes dont on va simuler l'exécution via l'utilisation de la classe Scheduler (ordonnanceur). Le scheduler gère l'activation des processus sur une base de temps donnée de façon à simuler leur activation autonome, et chaque processus lui-même gère son propre comportement selon son propre temps interne spécifique au capteur. Dans notre approche, les processus (capteurs) sont animés par le Scheduler, cependant nous pouvons imaginer des extensions avec l'utilisation directe de threads systèmes. C'est pourquoi nous assimilons les capteurs à des processus.

Pour représenter les données spécifiques à chaque capteur, la classe Sensor devra inclure, en tant qu'attributs, un identifiant unique du capteur, un type par défaut (via une "string"), une donnée pour la valeur du capteur, ainsi qu'un pointeur vers le Server dédié. La classe comprendra une fonction *update* spécifiant le comportement dynamique du capteur (le corps du processus) et exécutée par l'ordonnanceur (Scheduler) suivant une base de temps prédéfinie, et une fonction génératrice de données *execute* qui met à jour la valeur mesurée par le capteur selon une certaine base de temps inhérente au capteur cette fois, puis l'envoie au Server. Cette procédure simule la lecture d'une valeur de capteur avec de l'aléatoire.

Dans un premier temps, tous vos capteurs vous retourneront le même type de données, car issues d'une même classe par défaut, la classe Sensor, notamment si vous n'avez pas commencé à mettre en œuvre les mécanismes d'héritage. Pour votre rendu final, vous devrez pouvoir générer des données de plusieurs types différents, en fonction des différents types de capteurs. Pour un maximum de généricité dans la classe Sensor, on pourra définir une classe Data générique et la particulariser suivant le type de capteur, ou bien utiliser des fonctions génériques renvoyant des chaines de caractères préformatées. Cela dépendra du niveau de sophistication recherché.

Par exemple, on pourra avoir des données hétérogènes diverses :

- Temperature, Humidity : FLOAT

- Sound : INTEGER
- Light : BOOLEAN

Le nombre d'instances de capteurs de différents types est variable et les informations de chaque capteur peuvent être récupérées et transmises à des intervalles de temps différents.

4.2 Scheduler

L'ordonnanceur, implémenté dans la classe Scheduler, gère l'animation des processus, leur donne une activation suivant une base de temps prédéfinie. Dans cette classe, vous allez définir une fonction *simulation* qui organise le séquencement de l'exécution des processus (capteurs) via leur méthode *update*. Seuls les capteurs sont des processus, néanmoins on peut se poser la question pour le serveur selon les choix de réalisation. La fonction *simulation* doit permettre aux capteurs de récupérer les informations de mesure et de les retourner au serveur dédié.

4.3 Server

Le code de cette classe doit être complété en conséquence de la définition des classes Scheduler et Sensor. Notez que le serveur est considéré jusqu'à présent suivant un mode de fonctionnement passif : ses services sont invoqués de l'extérieur par les capteurs. On peut éventuellement lui conférer un mode de fonctionnement plus dynamique si besoin.

5 Troisième séance

On implémente tous les mécanismes d'héritage et de polymorphisme pas encore implémentés ou définis. Les classes de base deviennent des classes abstraites, et/ou des interfaces. Les mécanismes de généricité sont poussés à leur maximum et les dernières fonctionnalités manquantes du projet sont réalisées. Les concepteurs qui sont le plus avancés peuvent imaginer des fonctions d'analyse des mesures des capteurs, des fonctions de décision ou autres, exécutées en autonomie par le serveur qui au besoin, devient aussi un processus actif. Une liberté de choix est laissée au programmeur pour faire la différence avec le minimum requis.

6 Synthèse

Voici la liste des logiciels que vous avez pu mettre en place pour votre projet :

- g++
- man
- make pour faire des makefile (compilation automatisée)
- gdb (débeuger)
- valgrind (détection fuite mémoire)
- éditeur de code basique : VIM, EMACS, Gedit, atom, nano, imax, sublimeText
- IDE : qtcreator, visual studio code, clion, devC++.
- éditeur d'architecture logicielle (UML): le seul et unique "ASTAH"

Parmi les notions orientées objet du C++ que vous avez mis en oeuvre on aura, sans être exhaustif :

- classes avec constructeurs, destructeur, liste d'initialisation, forme de Coplien
- surcharge de méthode et d'opérateur
- utilisation de l'héritage et du polymorphisme, redéfinition de méthode
- concept d'interface, méthodes virtuelles
- conversions ascendantes, voir dans certains cas conversion descendante
- variable statique pour la gestion des identifiants uniques
- utilisation de la STL (conteneurs, itérateurs, algorithmes)
- utilisation des bibliothèques standards pour la gestion des dates, temporisations, nombres aléatoires

La notation lors de l'évaluation du projet tiendra compte des citères suivants :

- réalisation des objectifs à chaque TP et rendu final dans les temps
- lisibilité et structuration standard du code, compilation multi-plateforme
- implémentation correcte des principaux mécanismes du C++ (voir ci-dessus)
- compilation et exécution sans erreur
- gestion correcte de la mémoire
- niveau de sophistication de l'implémentation

7 Rendu de projet

Le rapport de TP et le code sont à rendre individuellement pour le **10 novembre 2024** avant minuit. Vous déposez le rapport et le code dans la section "devoir" dédiée à ce TP sur **Moodle**. Seuls les codes en C++ sont acceptés et les rapports en format PDF. Ne pas oublier le fichier CMakeLists.txt ou le fichier <projet>.pro pour la compilation multi-plateforme. Donc, vous avez deux fichiers à télécharger :

1. **Rapport_AP4A_Groupe_Nom_Prenom.pdf** : Rapport + le diagramme UML.

2. **Code_AP4A_Groupe_Nom_Prenom.zip** : Code en c++.

Concernant le nom de groupe, vous prenez le nom de votre groupe TP auquel vous ajoutez la lettre A ou B selon la semaine où vous êtes inscrit. Il varie selon votre créneau de TP AP4A:

1. TP1A: Lundi semaine A
2. TP1B: Lundi semaine B
3. TP3A: Jeudi semaine A
4. TP3B: Jeudi semaine B
5. TP2A: Vendredi semaine A
6. TP2B: Vendredi semaine B