UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Documentación Tarea 3: Semáforos como Objeto de Software C++

ELO329 Diseño y Programación Orientada a Objetos

Profesor: Cristobal Nettle

Nombres alumnos: Paula Amigo (201504013-3)
Luis Bahamondes (201421077-9)
Jairo González (201304502-2)

Fecha de entrega: 20 de agosto de 2019

Índice

1. Descripción del Desafío

En esta tarea se pide modelar semáforos en una intersección de calles. En particular analizaremos el funcionamiento de los semáforos ubicados en la intersección de Avenida Sporting Club con 1 Norte en Viña del Mar.

2. Objetivos

* Ejercitar la configuración de un ambiente de trabajo para desarrollar aplicaciones en lenguaje C++ (se sugiere trabajar con Jgrasp o Eclipse).

* Reconocer clases y relaciones entre ellas en lenguaje C++.

* Ejercitar la entrada y salida de datos en C++.

* Ejercitar la preparación y entrega de resultados de software (creación de makefiles, readme, documentación, manejo de repositorio -GIT).

* Familiarización con una metodología de desarrollo “iterativa” e “incrementa”.

3. Diagrama de Alto Nivel - Stage 4

A continuación se presenta un diagrama general de alto nivel que representa las interacciones entre las diferentes clases pertenecientes a la etapa 4 (Stage4).

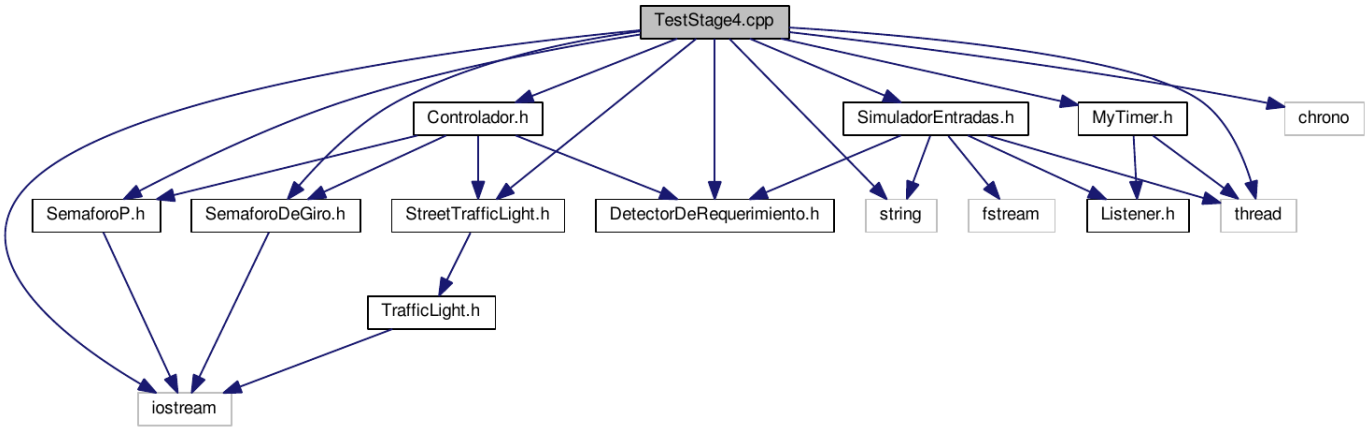

Figura 1: Diagrama de Alto Nivel Stage 4

4. Explicación de las clases - Stage 4

A continuación se explican las clases que componen el stage final del proyecto.

4.1. Controlador

La clase Controlador representa el control de todos los modelos de los distintos semáforos existentes. En esta clase se define el estado inicial de la intersección vial de las calles Sporting Club con 1 Norte y se define el método mannageTraffic(). Este funciona de manera similar que en las tareas previas, pero con diferencias en nombres de variables. Cabe destacar que se planificó el semáforo pensando en dar oportunidad a todos para cruzar. En el caso en que se presione el botón de cruce antes de darle el paso a los autos de 1 Norte, se les dará un segundo de verde y luego amarilla para que alcancen a pasar los primeros, para luego darle el cruce al peatón.

4.2. DetectorDeRequerimiento

Basado en las tareas previas, la clase DetectorRequerimiento conserva su estructura. Esta clase permite configurar como encendido o apagado algún requerimiento: solicitud de cruce peatonal o solicitud de giro vehicular. Representa el modelo para los sensores y botones.

4.3. MyTimer

Clase dedicada a realizar acciones cada cierto tiempo en [ms], esta clase realiza el llamado al Simulador de Entradas para la lectura de los requerimientos desde un archivo.

4.4. SemaforoDeGiro

Clase dedicada al modelo del semáforo de giro, se almacena su estado y los métodos para su funcionamiento.

4.5. SemaforoP

Clase dedicada al modelo del semáforo peatonal, se almacena su estado y los métodos para su funcionamiento.

4.6. SimuladorEntradas

Clase enfocada a la lectura de un archivo y al cambio de los requerimientos.

4.7. StreetTrafficLight

Esta clase hereda de *TrafficLight*. Corresponde a los semáforos de tres tiempos. Se utiliza el constructor de la clase padre.

4.8. TrafficLight

Clase base de los semáforos, el modelo. Esta clase incluye los métodos para cambiar de estado entre FOLLOW, TRANSITION y STOP. Aquí se definen los parámetros que contienen los tiempos en luz verde y amarillo. Es posible leer los tiempos de funcionamiento y el estado actual mediante los métodos definidos.

4.9. TestStage4

Clase principal, la cual incluye el método main(). Aquí se crean todos los objetos que participan en el funcionamiento de la intersección vial (semáforos, botones, sensores).

5. Dificultades y Soluciones

Dentro de las dificultades existentes, estuvo plantear la lógica detrás del controlador, creando un funcionamiento lo mas cercano a la realidad y que no presentara "bugs' o fallas en su procedimiento. Por otro lado, la migración estructural de códigos desde java a C++ presento uso extra de tiempo.

Otra dificultad fue la división de los archivos en un .cpp y un .h y la generación de un Makefile, manteniendo el orden del código y logrando compilarlo de manera correcta. Esto se logró gracias a la investigación y discusión sobre esta implementación.

Finalmente la utilización de la clase Timer y el paso del objeto de la clase SimuladorEntradas también fue de especial atención al momento de su uso dado que genero problemas en el llamado de la misma hasta poder darnos cuenta que la modificación de los parámetros de entrada era necesaria.