Análisis y desarrollo de la implementación del desafío 2

Informática II

Jose Andrés Henao Alzate
Departamento de Ingeniería Electrónica
Universidad de Antioquia
Medellín, Colombia
andres.henao9@udea.edu.co

***Resumen*—Este proyecto consiste en el desarrollo de un sistema de gestión de combustible para la red nacional de estaciones de servicio TerMax en Colombia. Utilizando programación orientada a objetos (POO) en C++, el sistema gestiona eficientemente las ventas de combustible y realiza el seguimiento de transacciones en múltiples estaciones. El sistema simula operaciones como el procesamiento de ventas, la gestión de inventarios y la detección de fugas. Al modelar estos procesos de manera precisa, el sistema proporciona una simulación de las operaciones de una estación de servicio, optimizando la gestión y distribución de recursos.**

***Index Terms*—Lenguaje C++, Eficiencia, Análisis, Clases, Abstracción, Encapsulamiento, Atributos, Métodos.**

I. Introducción

Las estaciones de servicio son infraestructuras fundamentales para el suministro de combustible. En Colombia, la gestión eficiente de estas estaciones es clave para garantizar la continuidad del transporte y el abastecimiento de bienes. El proyecto TerMax tiene como objetivo mejorar la administración de estaciones de servicio mediante un sistema automatizado, que optimiza operaciones como la venta de combustible, gestión de inventarios y detección de fugas. Este informe detalla el diseño e implementación del sistema utilizando programación orientada a objetos en C++. Además, se describen los problemas encontrados y los resultados obtenidos durante la implementación.

II. Análisis del problema y procedimiento de solución

El primer paso en la solución fue comprender el problema y llevarlo a un enfoque orientado a objetos. Se identificaron tres clases clave o principales: *Estación de servicio*, *Surtidor*, y *Tanque*, ilustradas en el diagrame UML de la figura 1. La clase principal, *Estación de servicio*, aunque es la que engloba y es base del procedimiento, tiene solo unas cuantas isntancias esto debido a que las demás clases abarcan métodos que cubren casi que el 100 % de las necesidades del problema, sin embargo, es importante el resaltar los atributos de dicha clase, los cuale son: **nombre, código, gerente, región** y **ubicación geográfica**. Estos atributos son extraídos del archivo principal, y no son de alta complejidad en el entendimiento de estos por lo que no se profundiza en ellos aquí.

El métdodo principal de esta clase (dejando de lado los contructores y desctructores), son el método mostrar codigo(), una breve explicación de dicho código se presenta a continuación:

- **void mostrar codigo():** Muestra el codigo de cada estación de servicio, es decir, la identificación única de cada estación. Este método es el enargado de retornar dicho atributo, el cual es usado como base de comparación en varios procedimientos.

Para la clase *Surtidor*, sus atributos describen características estándar del surtidor, estos son: *Codigo*, *modelo*, *codigo etacion*, y *estado*, mientras que sus métodos principales son:

- **void venta():** Genera y registra los datos de una venta en particular, esto respetando un formato previamente establecido.
- **void consultar_venta():** Busca en el archivo de ventas las transacciones realizadas por ese surtidor y calcula su producción.
- **string Mostrar codigo():** Retorna el código único del surtidor instanciado, es decir del objeto en particular, esto es de utilidad para la detección de dicho objeto en el programa.
- **void activar():** Sirve para activar o desactivar el objeto(surtidor), en particular, esto es controlado desde la estación misma.

Para la clase *Tanque*, sus atributos, describen carcterísitca ¿básicas y útiles de la clase, para el caso particular los atributos abstraidos son: *cpacidad regular*, *capacidad premium*,*Capacidad ecoextra*, y *Codigo*, donde este último atributo da información de a que estación gasolinera pertenece dicha instancia de dicha clase. Los Métodos de la clase son:

- **void venta():** Genera y registra los datos de una venta en particular, esto respetando un formato previamente establecido.
- **void consultar_venta():** Busca en el archivo de ventas las transacciones realizadas por ese surtidor y calcula su

producción.

- **string Mostrar codigo():** Retorna el código único del surtidor instanciado, es decir del objeto en particular, esto es de utilidad para la detección de dicho objeto en el programa.
- **void activar():** Sirve para activar o desactivar el objeto(surtidor), en particular, esto es controlado desde la estación misma.

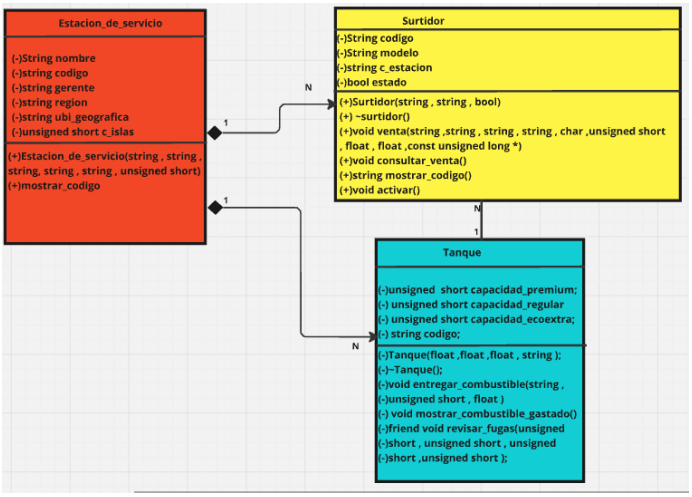

Figura 1. Diagrama UML simplificado.

III. Problemas encontrados y evolución de la implementación

Aunque la implementación del sistema no presentaba una alta complejidad técnica, el tiempo disponible fue un factor determinante. Debido a problemas externos, la carga de trabajo recayó en una sola persona, lo que afectó el avance del proyecto. Aunque se logró implementar todas las clases, y una gran parte de menú, las implementaciones hechas presentan pocas o casi nulas validaciones de datos, lo cual le resta calidad al trabajo entregado. Por el lado de la eficiencia algunos algoritmos podrían tener una mejor calidad de eficiencia, sin embargo, en un margen general se cumple con las funcionalidades.

Algo importante a resaltar es que las clases fueron probadas de forma individual, y las pruebas arrojaron resultados satisfactorios. Las operaciones de escritura en los archivos también funcionaron como se esperaba, lo cual es un indicador de que los métodos y atributos fueron correctamente implementados y que con un tiempo adicional, se da una garantía de una entrega mas óptima y de mayor calidad.

IV. Discusión de resultados

A nivel técnico, las clases y sus funcionalidades arrojaron resultados positivos en las pruebas individuales. Sin embargo, debido a limitaciones de tiempo, no se completó la implementación del menú y la integración total del sistema, lo que afectó el resultado general del proyecto. Aun así, los módulos probados confirmaron el funcionamiento correcto de las operaciones críticas del sistema y las implementaciones del menú mostraron buenos resultados funcionales.

V. Conclusiones

- **Implementación parcial:** Aunque las clases fueron correctamente implementadas y probadas de forma individual, la falta de tiempo impidió la integración completa de todas las funcionalidades, como el menú principal del sistema. No obstante, se muestra que los resultado vistos son satisfactorios para lo buscado, lo cual indica que se iba por un buen camino.
- **Resultados técnicos:** Las pruebas realizadas sobre las clases y las operaciones de escritura en archivos fueron satisfactorias y cumplieron con lo esperado.
- **Impacto del tiempo:** La principal limitación fue la falta de tiempo, debido a la concentración de la carga de trabajo en una sola persona. Este proyecto resalta la importancia de una gestión de tiempo eficiente y una distribución equitativa de tareas para completar proyectos de mayor envergadura.