Práctica 3. Identificación de las clases en la especificación de requerimientos

1. Introducción

Ahora empezaremos a diseñar el sistema de monitorización que presentamos en la Práctica 2. En esta práctica identificaremos las clases necesarias para crear el sistema planteado, analizando los sustantivos y las frases nominales que aparecen en la especificación de requerimientos. Presentaremos los diagramas de clases de UML para modelar las relaciones entre estas clases. Este primer paso es importante para definir la estructura de nuestro sistema.

2. Identificación de las clases en un sistema

Para comenzar nuestro proceso de DOO, vamos a identificar las clases requeridas para crear el sistema multisensorial. Más adelante describiremos estas clases mediante el uso de los diagramas de clases de UML y las implementaremos en C++. Primero debemos revisar la especificación de requerimientos que ya hicimos en la Práctica 2 e identificar los sustantivos y frases nominales clave que nos ayuden a identificar las clases que conformarán el sistema. Tal vez decidamos que algunos de estos sustantivos y frases nominales sean atributos de otras clases en el sistema. Tal vez también concluyamos que algunos de los sustantivos no corresponden a ciertas partes del sistema y, por ende, no deben modelarse. A medida que avancemos por el proceso de diseño podemos ir descubriendo clases adicionales.

Actividad 1: Comienza por confeccionar la lista de los sustantivos y frases nominales que se encuentran en la especificación de requerimientos (Práctica 2).

Vamos a crear clases sólo para los sustantivos y frases nominales que tengan importancia en el sistema. Por ejemplo, no modelamos la *empresa* como una clase, ya que la empresa no es una parte del sistema; la empresa sólo quiere que nosotros construyamos el sistema multisensorial. *Cliente* y *usuario* también representan entidades fuera del sistema; son importantes debido a que interactúan con nuestro sistema, pero no necesitamos modelarlos como clases en el software del sistema. Recuerda que modelamos un usuario del sistema (es decir, un trabajador de la empresa) como el actor en el diagrama de casos de uso de la figura que representamos en la Práctica 2.

En nuestro sistema, que incluye la recepción de datos de varios sensores, modelaremos las distintas acciones que pueden ser realizadas: *Ver temperatura*, *Ver humedad*, *Ver calidad del aire*, etc. como clases individuales. Estas clases poseen los atributos específicos necesarios para ejecutar las acciones que representan. Por ejemplo, para *Ver temperatura*, se necesita conocer el dato *temperatura* que el sensor de temperatura proporciona. Sin embargo, una solicitud de *Ver temperatura* no requiere datos adicionales. Lo que es más, las distintas clases correspondientes a las distintas acciones que se pueden realizar en el sistema muestran comportamientos únicos. Para *Ver temperatura* se requiere mostrar el valor *temperatura* al usuario, mientras que para *Ver humedad* se requiere mostrar el valor *humedad* al usuario.

Nota: cuando veamos *polimorfismo*, factorizaremos las características comunes de todas las operaciones que se pueden hacer en nuestro sistema en una clase de *Ver dato* general, mediante el uso de los conceptos orientados a objetos de las clases abstractas y la herencia.

Actividad 2: Determina las clases necesarias para implementar el sistema propuesto con base en los sustantivos y frases nominales que has confeccionado para la Actividad 1.

Con base en la lista que hemos creado en la Actividad 2, ya podemos modelar las clases en nuestro sistema. En el proceso de diseño escribimos los nombres de las clases con la primera letra en mayúscula (una convención de UML), como lo haremos cuando escribamos el código de C++ para implementar nuestro diseño. Si el nombre de una clase contiene más de una palabra, juntaremos todas las palabras y escribiremos la primera letra de cada una de ellas en mayúscula, lo que se conoce como *notación camello* (por ejemplo, `NombreConVariasPalabras`). Utilizando esta convención, vamos a crear las clases. Construiremos nuestro sistema mediante el uso de todas estas clases como bloques de construcción. Sin embargo, antes de empezar a construir el sistema, debemos comprender mejor la forma en que las clases se relacionan entre sí.

3. Modelado de las clases

UML nos permite modelar, a través de los diagramas de clases, las clases en el sistema ATM y sus interrelaciones. La Figura 1 representa a la clase `SistemaMonitorizacion`. En UML, cada clase se modela como un rectángulo con tres compartimentos. El compartimento superior contiene el nombre de la clase, centrado horizontalmente y en negrita. El compartimento intermedio contiene los atributos (datos) de la clase. El compartimento inferior contiene los métodos (funciones) de la clase. En la Figura 1, los compartimentos intermedio e inferior están vacíos, ya que no hemos determinado los atributos y métodos de esta clase todavía.

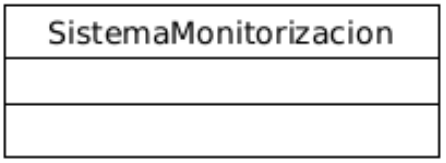

Figura 1: Representación de una clase en UML mediante un diagrama de clases

Los diagramas de clases también muestran las relaciones entre las clases del sistema. En la Figura 2 se muestra cómo nuestras clases `SistemaMonitorizacion` y `VisualizacionTemperatura` se relacionan una con la otra. Observa que los rectángulos que representan a las clases en este diagrama no están subdivididos en compartimentos. UML permite suprimir los atributos de las clases y sus operaciones de esta forma, cuando sea apropiado, para crear diagramas más legibles. Un diagrama de este tipo se denomina diagrama con elementos omitidos (*elided diagram*): su información, tal como el contenido de los compartimentos segundo y tercero, no se modela. Más adelante colocaremos información en estos compartimentos.

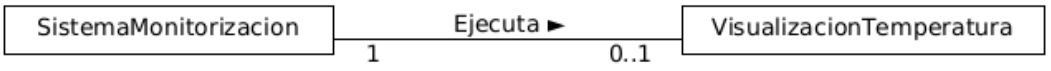

Figura 2: Diagrama de clases que muestra una asociación entre clases

Actividad 3: Modela el conjunto de las clases del sistema que habías determinado. Para ello, usaremos la herramienta *umlet* (Figura 3), ya instalada en los equipos del laboratorio y disponible para instalar desde los repositorios oficiales de cualquier distribución de Ubuntu (también en otros S.S.O.O.). Es una herramienta muy simple, básica, pero a la vez muy potente. Todos los elementos disponibles están a mano derecha; para usar cualquiera de ellos basta con hacer doble click sobre él o arrastrarlo. En la parte inferior derecha encontramos las propiedades del elemento actual, para modificarlo según las necesidades.

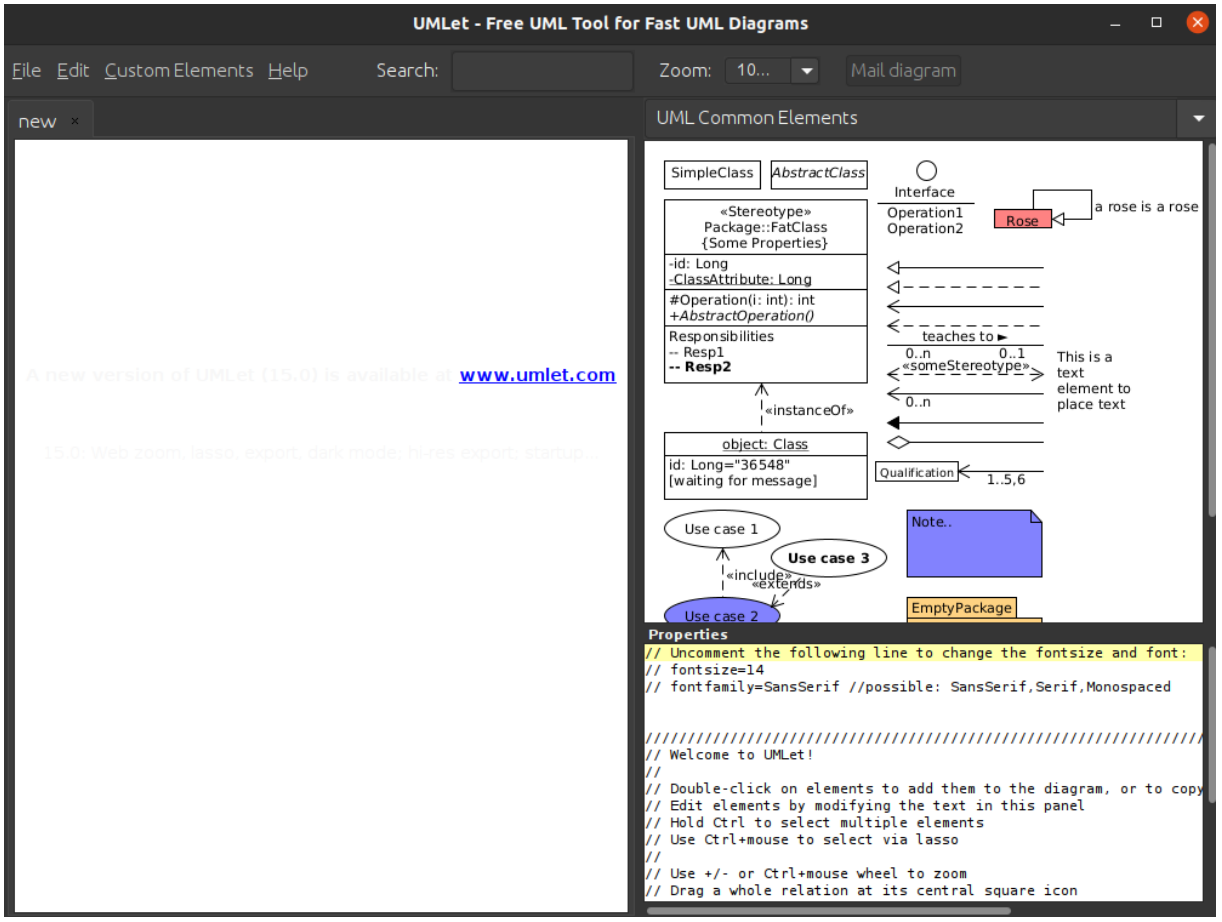

Figura 3: Interfaz de la herramienta umlet, de modelado UML

En la Figura 2, la línea sólida que conecta a las dos clases representa una **asociación**: una relación entre clases. Los números cerca de cada extremo de la línea son valores de **multiplicidad**; éstos indican cuántos objetos de cada clase participan en la asociación. En este caso, al seguir la línea de un extremo al otro se revela que, en un momento dado, un objeto `SistemaMonitorizacion` participa en una asociación con cero o con un objeto `VisualizacionTemperatura`; cero si el usuario actual no está realizando una operación de visualización de temperatura o si ha solicitado un tipo distinto de acción/visualización, y

uno si el usuario ha solicitado una operación de visualización de temperatura. UML puede modelar muchos tipos de multiplicidad. En la Tabla 1 se listan y explican los tipos de multiplicidad.

Símbolo	Significado
0	Ninguno
1	Uno
m	Un valor entero
0..1	Cero o uno
m, n	m o n
$m..n$	Cuando menos m, pero no más que n
$*$	Cualquier entero no negativo (cero o más)
$0..*$	Cero o más (idéntico a *)
$1..*$	Uno o más

Cuadro 1: Tipos de multiplicidad

Una asociación puede tener nombre. Por ejemplo, la palabra `Ejecuta` por encima de la línea que conecta a las clases `SistemaMonitorizacion` y `VisualizacionTemperatura` en la Figura 2 indica el nombre de esa asociación. Esta parte del diagrama se lee así: *un objeto de la clase SistemaMonitorizacion ejecuta cero o un objeto de la clase VisualizacionTemperatura*. Los nombres de las asociaciones son direccionales, como lo indica la punta de flecha rellena; por lo tanto, sería inapropiado, por ejemplo, leer la anterior asociación de derecha a izquierda como *cero o un objeto de la clase VisualizacionTemperatura ejecuta un objeto de la clase SistemaMonitorizacion*.

También se puede añadir una palabra en el extremo, en este caso, de `VisualizacionTemperatura` de la línea de asociación en la Figura 2. Esta palabra sería un nombre de rol, el cual identificaría el rol que desempeña el objeto `VisualizacionTemperatura` en su relación con el `SistemaMonitorizacion`. Un nombre de rol agrega significado a una asociación entre clases, ya que identifica el rol que desempeña una clase dentro del contexto de una asociación. Una clase puede desempeñar varios roles en el mismo sistema. Por ejemplo, en un sistema de personal de una universidad, una persona puede desempeñar el rol de *profesor* respecto a los estudiantes. La misma persona puede desempeñar el rol de *colega* cuando participa en una asociación con otro profesor, y de *entrenador* cuando entrena a los atletas estudiantes. Sin embargo, a menudo, los nombres de los roles se omiten en los diagramas de clases, ya que el significado de una asociación suele estar claro sin ellos.

Además de indicar relaciones simples, las asociaciones pueden especificar relaciones más

complejas, como cuando los objetos de una clase están compuestos de objetos de otras clases. Considera, por ejemplo, un sistema de monitorización real. ¿Qué piezas reúne un fabricante para construirlo? Nuestra especificación de requerimientos nos indica que el sistema está compuesto de una pantalla, un teclado, un micrófono y un interruptor.

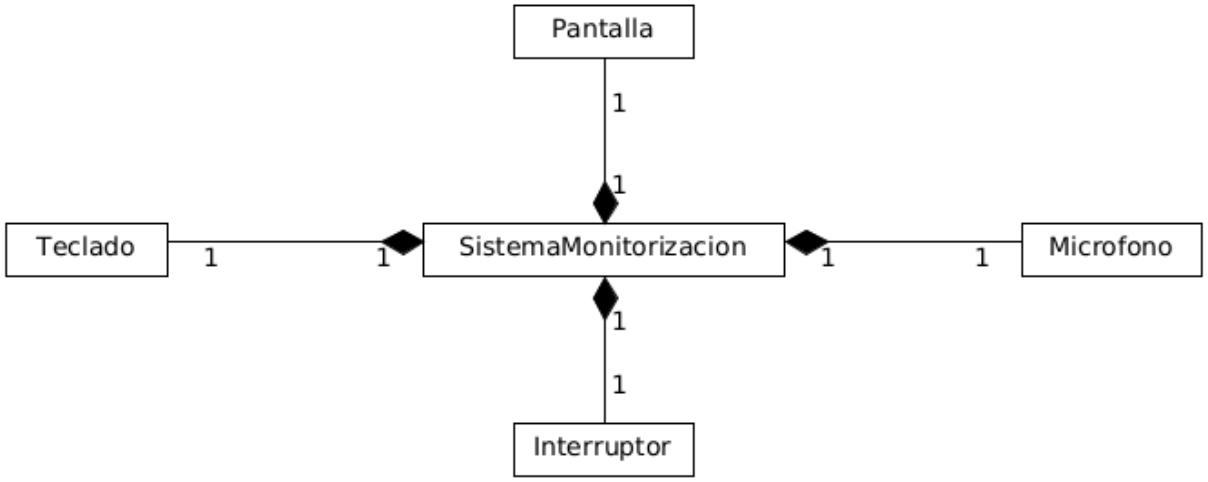

Figura 4: Diagrama de clases que muestra las relaciones de composición

En la Figura 4, los rombos sólidos que se adjuntan a las líneas de asociación de la clase `SistemaMonitorizacion` indican que esta clase tiene una **relación de composición** con las clases `Pantalla`, `Teclado`, `Microfono` e `Interruptor`. La composición implica una relación en todo/en parte. La clase que tiene el símbolo de composición (el rombo sólido) en su extremo de la línea de asociación es el todo (en este caso, `SistemaMonitorizacion`), y las clases en el otro extremo de las líneas de asociación son las partes; en este caso, las clases `Pantalla`, `Teclado`, `Microfono` e `Interruptor`. Las composiciones en la Figura 4 indican que un objeto de la clase `SistemaMonitorizacion` está formado por un objeto de la clase `Pantalla`, un objeto de la clase `Teclado`, un objeto de la clase `Microfono` y un objeto de la clase `Interruptor`. El sistema *tiene una* pantalla, un teclado, un micrófono y un interruptor. La relación *tiene un* define la composición (cuando veamos el tema de herencia y polimorfismo, veremos que la relación *es un* define la herencia).

De acuerdo con la especifi cación de UML, las relaciones de composición tienen las siguientes propiedades:

1. Solo una clase en la relación puede representar el todo; es decir, el rombo puede colocarse solo en un extremo de la línea de asociación. Por ejemplo, la pantalla es parte

del sistema o el sistema es parte de la pantalla, pero la pantalla y el sistema no pueden representar ambos el *todo* dentro de la relación.

2. Las partes en la relación de composición existen solo mientras exista el todo, y el todo es responsable de la creación y destrucción de sus partes. Por ejemplo, el acto de construir un sistema incluye la manufactura de sus partes. Lo que es mas, si el sistema se destruye, también se destruyen su pantalla, teclado, micrófono e interruptor.

3. Una parte puede pertenecer sólo a un todo a la vez, aunque esa parte puede quitarse y unirse a otro todo, el cual entonces asumirá la responsabilidad de esa parte.

Los rombos sólidos en nuestros diagramas de clases indican las relaciones de composición que cumplen con estas tres propiedades. Si una relación *tiene un* no satisface uno o más de estos criterios, UML especifica que se deben adjuntar rombos sin relleno a los extremos de las líneas de asociación para indicar una **agregación**: una forma más débil de la composición. Por ejemplo, una PC y un monitor de ordenador participan en una relación de agregación: la computadora *tiene un* monitor, pero las dos partes pueden existir en forma independiente, y el mismo monitor puede conectarse a varios PC a la vez, con lo cual se violan las propiedades segunda y tercera de la composición.

Ejercicio

Llegados a este punto, ya hemos identificado las clases en nuestro sistema de monitorización, aunque tal vez descubramos otras, a medida que avancemos con el diseño y la implementación. Como ejercicio final, completa el diagrama de clases para el sistema planteado. Este diagrama debe modelar las clases que ya habías modelado en la Actividad 3 así como las asociaciones entre ellas que podemos inferir de la especificación de requerimientos.

Exporta el resultado final a formato pdf. En el repositorio de esta práctica deberían resultar dos ficheros: `practica2.uxf` (que contiene el fuente de *umlet*), y `practica2.pdf` (que contiene el resultado final impreso en pdf).

Más adelante determinaremos los atributos para cada una de estas clases y, utilizando estos, analizaremos la forma en que cambia el sistema con el tiempo. Y cuando ya tengamos los atributos establecidos, determinaremos las operaciones de las clases en nuestro sistema.