3 Especificación funcional.UML.

3.1 Introducción a UML

El producto principal de un equipo de desarrollo es un buen software que satisfaga las necesidades de los usuarios y la empresa. Sin embargo, el modelado es también muy importante.

Un modelo es una simplificación de la realidad. Un modelo proporciona los planos de un sistema. Puede incluir planos detallados así como planos más generales del sistema. Un buen modelo incluye los elementos que tienen mayor influencia y omite los elementos menores que no son relevantes para el nivel de abstracción dado.

Un modelo puede ser estructural, destacando la organización del sistema, o dinámico, en cuyo caso destaca su dinámica.

Gracias al modelado se consigue visualizar más claramente cómo es el sistema o cómo se desea que sea. También permiten especificar la estructura o el comportamiento de dicho sistema. Además proporcionan plantillas que orientan en la construcción de un sistema y permiten documentar las diferentes decisiones que se han tomado.

Muchas veces un único modelo no es suficiente. Los sistemas no triviales se describen mejor mediante un pequeño conjunto de modelos casi independientes.

Actualmente el desarrollo de software tiende cada vez más a un entorno orientado a objetos. Así, el principal bloque de construcción de cualquier sistema software es el objeto o clase.

El Lenguaje Unificado de Modelado (UML, Uniform Modeling Language)

UML es un lenguaje estándar para visualizar planos de software. Puede utilizarse para visualizar, especificar, construir y documentar sistemas que involucren gran cantidad de software.

UML es sólo un lenguaje y por lo tanto una parte de un método de desarrollo de software. UML es independiente del proceso, aunque para utilizarlo óptimamente se debe usar en procesos dirigidos por los casos de uso, centrados en la arquitectura, iterativos e incrementales.

Los lenguajes de modelado orientados surgen entre la mitad de los sesenta y finales de los ochenta. Muchos de los usuarios de estos métodos tenían problemas al intentar encontrar un lenguaje que satisfaga completamente sus necesidades, con lo que se generó una “guerra de métodos”.

Más tarde aparecieron nuevas generaciones de métodos. Entre éstos destacaron principalmente el método de Booch, el método OOSE (Object Oriented Software Engineering, Ingeniería de software orientada a objetos) de Jacobson y el método OMT (Object Modeling Technique, Técnica de modelado de objetos) de Rumbaugh.

Cada uno de estos métodos era completo, pero cada uno tenía sus puntos fuertes y sus debilidades. Por ejemplo, el método de Booch era muy expresivo durante las fases de diseño y construcción de los proyectos; OOSE proporcionaba un excelente soporte para los casos de uso para dirigir la captura de requisitos y al análisis y diseño de alto nivel; y OMT en su segunda versión era muy útil para el análisis de sistemas de información con gran cantidad de datos.

En la primera mitad de los noventa, Grady Booch, Ivar Jacobson y James Rumbaugh empezaron a adoptar ideas de cada uno de los tres métodos, que estaban considerados los tres principales a nivel mundial.

Por tanto, los tres métodos evolucionaban hacia los otros dos. Si a este hecho se le une la gran estabilidad que proporcionaría al mercado la existencia de un lenguaje de modelado unificado y la esperanza de que la colaboración tuviera como resultado un lenguaje mejor, es comprensible tomaran la decisión de unificar los tres métodos.

También pidieron realimentación a la comunidad internacional de desarrolladores de software. De esta forma, a la definición de la versión 1.0 de UML contribuyeron organizaciones como DEC, HP, I_Logix, Intellicorp, IBM, ICON Computing, MCISystemhouse, Microsoft, Oracle, Rational, Texas Instruments y Unisys. UML 1.0 se ofreció para su estandarización a la OMG (Object Management Group) en enero de 1997, en respuesta a su solicitud de propuestas para un lenguaje unificado de modelado.

En los meses siguientes se amplió el grupo de colaboradores para incluir a Andersen Consulting, Ericsson, ObjectTime Limited, Platinum Technology, Ptech, Reich Technologies, Softeam, Sterling Software y Taskon.

La especificación UML 1.1 fue adoptada por el OMG el 14 de noviembre de 1997.

Modelo conceptual de UML

UML incluye tres clases de bloques de construcción: elementos, relaciones y diagramas.

Un elemento es una abstracción que es un ciudadano de primera clase de un modelo. Son los componentes básicos del modelo.

Una relación liga a varios elementos entre sí.

Los diagramas agrupan colecciones interesantes de elementos. Normalmente, según el tipo de diagrama intervienen unos elementos u otros.

A su vez hay cuatro tipos de elementos: estructurales, de comportamiento, de agrupación y de anotación.

Los elementos estructurales son los nombres de los modelos UML. Suelen ser estáticos y representan cosas conceptuales o materiales. Hay siete tipos de elementos estructurales: clase, interfaz, colaboración, caso de uso, clase activa, componente y nodo.

Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Gráficamente se representa como un rectángulo que incluye el nombre, los atributos y las operaciones.

Una interfaz es una colección de operaciones que especifican un servicio de una clase o componente. Describe el comportamiento visible exteriormente de ese elemento. Es decir, ofrece un servicio pero no se detalla como se lleva a cabo ese servicio, ya que incluye la especificación de las operaciones, pero no la implementación. Normalmente está conectada a la clase o componente que la realiza. Se representa gráficamente por un círculo.

Una colaboración define una interacción. Es una sociedad en la que varios elementos colaboran para proporcionar un comportamiento cooperativo mayor que la suma de los comportamientos de sus elementos. Una misma clase puede participar en varias colaboraciones. Se representa gráficamente por una elipse de borde discontinuo.

Un caso de uso es una descripción de un conjunto de secuencias de acciones que un sistema ejecuta y que produce un resultado observable de interés para un actor particular. Se utiliza para estructurar los aspectos de comportamiento en un modelo y suele ser realizado por una colaboración. Se representa gráficamente por una elipse

Una clase activa es una clase cuyos objetos tienen varios procesos o hilos de ejecución, por lo que pueden originar actividades de control. Es decir, los objetos de esta clase son elementos cuyo comportamiento es concurrente con otros elementos. Se representa gráficamente como una clase con las líneas más gruesas.

Un componente es una parte física y reemplazable de un sistema. Está formado por un conjunto de interfaces pero proporciona la implementación de dicho conjunto. Normalmente es un agrupación físico de diferentes elementos como clases, interfaces y colaboraciones. Se representa gráficamente como un rectángulo con pestañas.

Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional. Es decir, normalmente dispone de memoria e incluso capacidad de procesamiento. Un grupo de componentes puede residir en un nodo. Se representa gráficamente como un cubo.

Una vez descritos los elementos estructurales, les toca el turno a los elementos de comportamiento.

Los elementos de comportamiento son las partes dinámicas de los modelos UML. Hay dos tipos principales: interacción y máquina de estados.

Una interacción es un comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos, dentro de un cierto contexto y para alcanzar cierto fin. Se representa gráficamente como una línea dirigida.

Una máquina de estados es un comportamiento que especifica las secuencias de estados por las que pasa un objeto o una interacción durante su vida en respuesta a eventos, junto a sus reacciones frente a estos eventos. Una máquina de estados involucra además

a otros elementos, como estados, transiciones, eventos y actividades. Cada estado se representa gráficamente con un rectángulo de esquinas redondeadas.

Los elementos de agrupación son las partes organizativas de un modelo UML. Hay un elemento de agrupación principal, el paquete.

El paquete es un mecanismo de propósito general para organizar elementos en grupos. En un paquete se pueden incluir elementos estructurales, elementos de comportamiento o incluso otros paquetes. Sólo existe en tiempo de desarrollo, no en tiempo de ejecución, es conceptual. Se representa gráficamente por una carpeta.

Sólo quedan los elementos de anotación para terminar con los tipos de elementos UML.

Los elementos de anotación son partes explicativas de los modelos UML. Son comentarios para clarificar o describir cualquier elemento de un modelo. Hay un tipo principal de anotación, llamado nota. Una nota es un símbolo para mostrar restricciones y comentarios junto a un elemento o colección de ellos. Se representa gráficamente por un recuadro con una esquina doblada.

Una vez finalizada la descripción de los distintos tipos de elementos es el turno de las relaciones.

Hay cuatro tipos de relaciones en UML: dependencia, asociación, generalización y realización.

Una dependencia es una relación semántica entre dos elementos en la cual el cambio en un elemento puede afectar a la semántica del otro. Por lo tanto habrá un elemento dependiente y otro independiente. Se representa gráficamente por una flecha discontinua, que puede incluir una etiqueta.

Una asociación es una relación estructural que describe un conjunto de conexiones entre objetos. Un caso especial de asociación es la agregación, que representa una relación estructural entre un todo y sus partes. Se representa gráficamente como una línea continua, que puede ser dirigida, y que puede incluir una etiqueta, las multiplicidades y los nombres de rol.

Una generalización es una relación de especialización/generalización en la cual los objetos del elemento especializado (el hijo) pueden sustituir a los objetos del elemento general (el padre). Se representa gráficamente por una línea continua con punta de flecha vacía que apunta al padre.

Finalmente, la realización es una relación semántica entre clasificadores, donde un clasificador especifica un contrato que el otro garantiza que cumplirá. Suelen darse entre los interfaces y las clases y componentes que las realizan, y entre los casos de uso y las colaboraciones que los realizan. Su representación gráfica es una línea discontinua con punta de flecha vacía.

El último bloque de construcción UML que queda por ver es el diagrama.

Un diagrama es la representación gráfica de un conjunto de elementos, visualizado la mayoría de las veces como un grafo conexo de nodos (elementos) y arcos (relaciones). Lo más normal es que cada tipo de elemento aparezca sólo en algunos diagramas.

UML incluye nueve tipos de diagramas: de clases, de objetos, de casos de uso, de secuencia, de colaboración, de estados, de actividades, de componentes y de despliegue.

El diagrama de clases muestra las clases, interfaces y colaboraciones y las relaciones entre ellos. Son los más comunes. Cubren la vista de diseño estática.

El diagrama de objetos muestra un conjunto de objetos y las relaciones entre ellos. Representan instantáneas de instancias de los elementos que aparecen en los diagramas de clases. Cubren la vista de diseño estática desde la perspectiva de casos reales.

El diagrama de casos de uso muestra un conjunto de casos de uso y actores, que son un tipo especial de clases. Cubren la vista de casos de uso estática de un sistema. Son muy importantes en el modelado y organización del comportamiento de un sistema.

En el anterior listado de tipos de diagramas no aparecen los diagramas de interacción. Los diagramas de interacción muestran interacciones, es decir, un conjunto de objetos y sus relaciones, incluyendo los mensajes que pueden ser enviados entre ellos. Cubren la vista dinámica del sistema.

Tanto los diagramas de secuencia como los de colaboración son casos particulares de diagramas de interacción. Los diagramas de secuencia resaltan la ordenación temporal de los mensajes, mientras que el diagrama de colaboración resalta la organización estructural de los objetos que envían y reciben mensajes.

El diagrama de estados muestra una máquina de estados, que consta de estados, transiciones, eventos y actividades. Cubren la vista dinámica del sistema. Son especialmente importantes en el modelado del comportamiento de interfaces, clases y colaboraciones y resaltan el comportamiento dirigido por eventos de un objeto.

El diagrama de actividades es un tipo especial de diagrama de estados en el que se muestra el flujo de actividades dentro de un sistema. Cubren la vista dinámica del sistema.

El diagrama de componentes muestra la organización y las dependencias entre un conjunto de componentes. Cubren la vista de implementación estática de un sistema. Se relacionan con los diagramas de clases porque un componente se suele corresponder con una o más clases, interfaces o colaboraciones.

El diagrama de despliegue muestra la configuración de nodos de procesamiento en tiempo de ejecución y los componentes que residen en ellos. Cubren la vista de despliegue estática de una arquitectura. Se relacionan con los diagramas de componentes ya que cada nodo suele incluir uno o más componentes.

Después de la definición de todos estos bloques de construcción es conveniente agruparlos en un esquema:

Bloques de Construcción UML	Elementos	Estructurales	Clase
			Interfaz
			Colaboración
			Caso de uso
			Clase activa
			Componente
			Nodo
		Comportamiento	Interacción
			Máquina de estado
		Agrupación	Paquete
		Anotación	Nota
	Relaciones	Dependencia	
		Asociación	
		Generalización	
		Realización	
	Diagramas	Clases	
		Objetos	
		Casos de uso	
		Secuencia	
		Colaboración	
		Estados	
		Actividades	
		Componentes	
		Despliegue	

Cuando se ha definido cada uno de los diferentes bloques de construcción de UML se ha hecho referencia a términos como vista de diseño, vista de implementación, etcétera.

Estos conceptos hacen referencia a la arquitectura del sistema. La arquitectura de un sistema que tenga una gran cantidad de software puede describirse a través de cinco vistas relacionadas entre sí. Cada una de estas vistas es una proyección de la organización y estructura del sistema, centrada en un aspecto particular de ese sistema, como se representa en la figura 3.1.1.

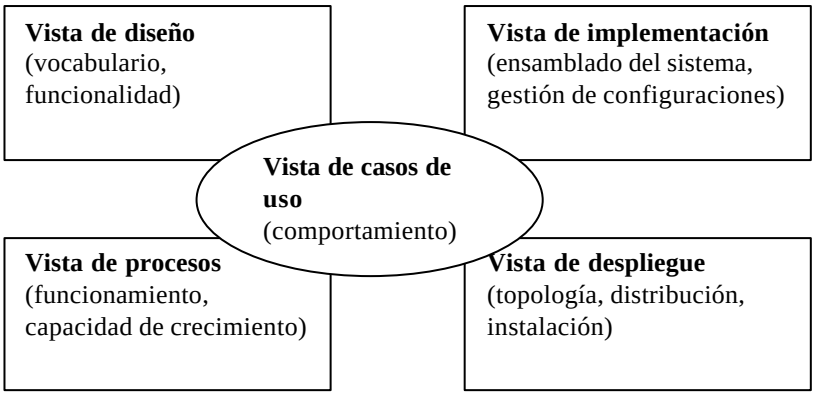

Figura 3.1.1: Modelado de la arquitectura de un sistema

La vista de casos de uso comprende los casos de uso que describen el comportamiento del sistema tal y como es percibido por los usuarios finales, analistas y encargados de las pruebas. En UML los aspectos estáticos de esta vista se muestran en los diagramas de los casos de uso y los dinámicos en los diagramas de interacción, de estados y de actividades.

La vista de diseño comprende las clases, interfaces y colaboraciones. Soporta los requisitos funcionales del sistema, entendiendo por ello los servicios que el sistema debería proporcionar a sus usuarios finales. En UML los aspectos estáticos se muestran en los diagramas de clases y de objetos, y los dinámicos en los diagramas de interacción, de estados y de actividades.

La vista de procesos comprende los hilos y los procesos que forman los mecanismos de sincronización y concurrencia del sistema. En UML tanto los aspectos estáticos como dinámicos se muestran en los mismos diagramas que la vista de diseño, pero con centrándose en las clases activas.

La vista de implementación comprende los componentes y archivos que se utilizan para ensamblar y hacer disponible el sistema físico. En UML los aspectos estáticos se muestran en el diagrama de componentes, y los aspectos dinámicos en los diagramas de interacción, estados y actividades.

La vista de despliegue contiene los nodos que forman la topología hardware sobre la que corre el sistema. En UML los aspectos estáticos se muestran en el diagrama de despliegue, los aspectos dinámicos en los diagramas de interacción, estados y actividades.

Cada una de estas cinco vistas puede existir por sí misma, pero también pueden interactuar entre sí. UML permite expresar cada una de estas vistas y sus interacciones.

Cuando se modela un sistema desde diferentes vistas, realmente se está construyendo el sistema desde múltiples dimensiones. En primer lugar hay que decidir las vistas que se

necesitan para expresar mejor la arquitectura del sistema. Una vez elegidas, hay que decidir qué diagramas UML se necesitan para mostrar los aspectos esenciales.

Para una aplicación monolítica que se ejecuta en una máquina es suficiente con los siguientes diagramas: diagrama de casos de uso para la vista de casos de uso, diagramas de clases y de interacción para la vista de diseño. No es necesario implementar las otras tres vistas.

Para sistemas complejos y distribuidos es necesario emplear todos los diagramas UML para expresar la arquitectura del sistema:

- vista de casos de uso: diagramas de casos de uso y de actividades
- vista de diseño: diagramas de clases, de interacción y de estados.
- vista de procesos: diagramas de clases y de interacción
- vista de implementación: diagramas de componentes.
- vista de despliegue: diagramas de despliegue.

Tras esta introducción general, a continuación se entrará en más profundidad en cada uno de bloques de construcción más importantes del lenguaje UML.

En primer lugar se tratarán los elementos, a continuación las relaciones y finalmente los diagramas. Siempre que sea posible, se usarán como ejemplos elementos, relaciones y diagramas que aparezcan en el modelo de la aplicación, con el fin de facilitar la comprensión de los mismos.

Elementos

Antes de entrar con los Elementos, es conveniente definir el concepto de clasificador en UML.

Algunos elementos de UML no tienen instancias, por ejemplo, los paquetes. Todos los elementos de modelado que pueden tener instancias se llaman clasificadores

Clases

Son los bloques de construcción más importantes de cualquier sistema orientado a objetos. Una clase es una descripción de una serie de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Una clase es el clasificador más importante en UML.

Cada clase debe tener un nombre. Es simplemente una cadena de texto identificativa. Éste es el nombre simple. El nombre de camino incluye el nombre simple precedido del nombre del paquete que la contiene.

Un atributo es una propiedad de una clase. Representa una propiedad del elemento que se está modelando que es compartida por todos los objetos de esa clase. Cada objeto de la clase tendrá valores específicos para cada un o de sus atributos.

Una operación es la implementación de un servicio que puede ser requerido a cualquier objeto de la clase. En UML las operaciones no equivalen a los métodos. Un método es cada una de las implementaciones posibles de una operación. Debido al polimorfismo, una misma operación puede tener varios métodos.

Al dibujar una clase no hay que mostrar todos los atributos ni operaciones, ya que normalmente hay demasiados. Además es conveniente mostrar sólo los atributos y operaciones realmente relevantes en cada vista.

También se pueden añadir a las clases Responsabilidades. Son cadenas de texto que expresan un contrato o obligación de la clase. Normalmente, expresan el cometido de la clase.

Uno de los detalles más importantes que se pueden especificar para los atributos y operaciones de un clasificador es la visibilidad. En UML hay tres niveles de visibilidad:
- public: cualquier clasificador externo con visibilidad hacia el clasificador puede acceder a la característica. Se puede especificar anteponiendo el símbolo +.
- protected: cualquier descendiente del clasificador puede usar la característica. Se especifica anteponiendo el símbolo #.
- private: sólo el propio clasificador puede utilizar la característica. Se especifica precediéndola del símbolo -.

Otro detalle muy importante es el Alcance. Especifica si la propiedad aparece en cada instancia del clasificador (el alcance tendría el valor instancia) si sólo hay una instancia de la característica para todas las instancias del clasificador (el alcance tendría el valor clasificador). Cuando el alcance es de clasificador se denota subrayando la propiedad.

También se pueden definir clases abstractas, es decir, que no puede tener instancias directas. Se denotan poniendo su nombre en cursiva. Estas clases suelen usarse para derivar otras clases de ellas.

Además las operaciones de las clases hijas pueden redefinir el comportamiento de la clase padre. Una operación abstracta implica que debe usarse la implementación que proporcione la clase hija. Esto se corresponde con las funciones virtuales puras en C++. Estas operaciones se denotan escribiéndolas en cursiva.

Por otro lado, también existen las operaciones hoja, es decir, que no pueden ser redefinidas. Se denotan por la palabra leaf entre llaves (*{leaf}*).

Se puede restringir el número de instancias de una clase mediante la multiplicidad. Se denota escribiendo dicha multiplicidad junto al nombre de la clase. También se puede aplicar la multiplicidad a los atributos. Se denota encerrando la multiplicidad entre corchetes ([]) tras el nombre del atributo.

Se pueden especificar muchas propiedades de los atributos. En concreto, las visibilidad, el alcance, la multiplicidad, el tipo, el valor inicial y los cambios posibles de cada un o de ellos. Hay tres propiedades predefinidas para los cambios posibles:
- changeable: No hay restricciones para modificar el valor del atributo. Es el comportamiento por defecto.

- addOnly: Para atributos con multiplicidad mayor que uno, se pueden añadir valores adicionales pero un atributo ya creado no puede ser modificado.
- frozen: El atributo no se puede modificar una vez creado (equivale a const en C++).

El formato completo para definir un atributo es:
[visibilidad] nombre [multiplicidad] [:tipo] [=valor inicial] [{propiedades}]
Todo lo que está entre corchetes es opcional.

Para las operaciones se puede especificar la visibilidad, el alcance, los parámetros, el tipo de retorno, la semántica de concurrencia y otras propiedades de cada operación.

La forma completa de la sintaxis una operación es:

[visibilidad] nombre [(lista de parámetros)] [:tipo retorno] [{propiedades}]

De nuevo, los encerrados entre corchetes son opcionales.
Cada uno de los parámetros sigue la siguiente sintaxis:
[direccion] nombre : tipo [=valor por defecto]

dirección puede tomar uno de los siguientes valores:
-in: el parámetro no se puede modificar, es de entrada.
-out: puede modificarse para comunicar información al comunicador
-inout: el parámetro es de entrada pero puede modificarse, por lo que también puede devolver información al invocador.

Además de la propiedad leaf ya vista, se pueden usar las siguientes con las operaciones:
- isQuery: La ejecución no cambia el estado del sistema.
- sequential: Los invocadores deben coordinarse para que en el objeto haya sólo un único flujo de control al mismo tiempo.
- guarded: La integridad del objeto se garantiza en presencia de múltiples flujos de control, gracias a la secuenciación de todas las llamadas a todas las operaciones con guardas del objeto.
- concurrent: La integridad del objeto se garantizan en presencia de múltiples flujos de control, tratando la operación como atómica.

Como se habrá podido observar, las tres últimas propiedades sólo tienen sentido con clases activas, procesos e hilos.

Un ejemplo sencillo de clase sería la clase Contexto:

Figura 3.1.2: Clase

Pueden apreciarse las distintas propiedades y operaciones, aunque en la representación gráfica no se muestra gran cantidad de información definida para la clase. Por ejemplo, no se muestra el valor de retorno de las operaciones o los parámetros que se le pasan a cada una de ellas.

Clases plantilla (template en C++)

Una plantilla es un elemento parametrizado. Una clase plantilla define una familia de clases, y una función plantilla define una familia de funciones. La plantilla incluye huecos para clases, objetos y valores. Estos huecos son parámetros de la plantilla.

En la aplicación se usa una clase plantilla, *Lista_inf*.

Puede apreciarse la construcción de la clase plantilla a partir de la clase *TList*. Obsérvese como recibe un parámetro, en este caso Y. Y es la clase de los elementos a almacenar en la lista.

Puede apreciarse también cómo se modela en UML una lista de objetos de la clase *Punto* usando la plantilla *Lista_inf*.

En realidad UML no es necesario mostrar la relación de dependencia entre la clase plantilla (*Lista_inf*<*Punto*>) y la clase que sirve como parámetro (*Punto*) si en dicha clase plantilla se define explícitamente el parámetro que recibe (se define como *Lista_inf*<*Punto*> en lugar de *Lista_inf* a secas). Sin embargo parece conveniente incluir esta relación de dependencia para clarificar aún más la definición de este tipo de clases.

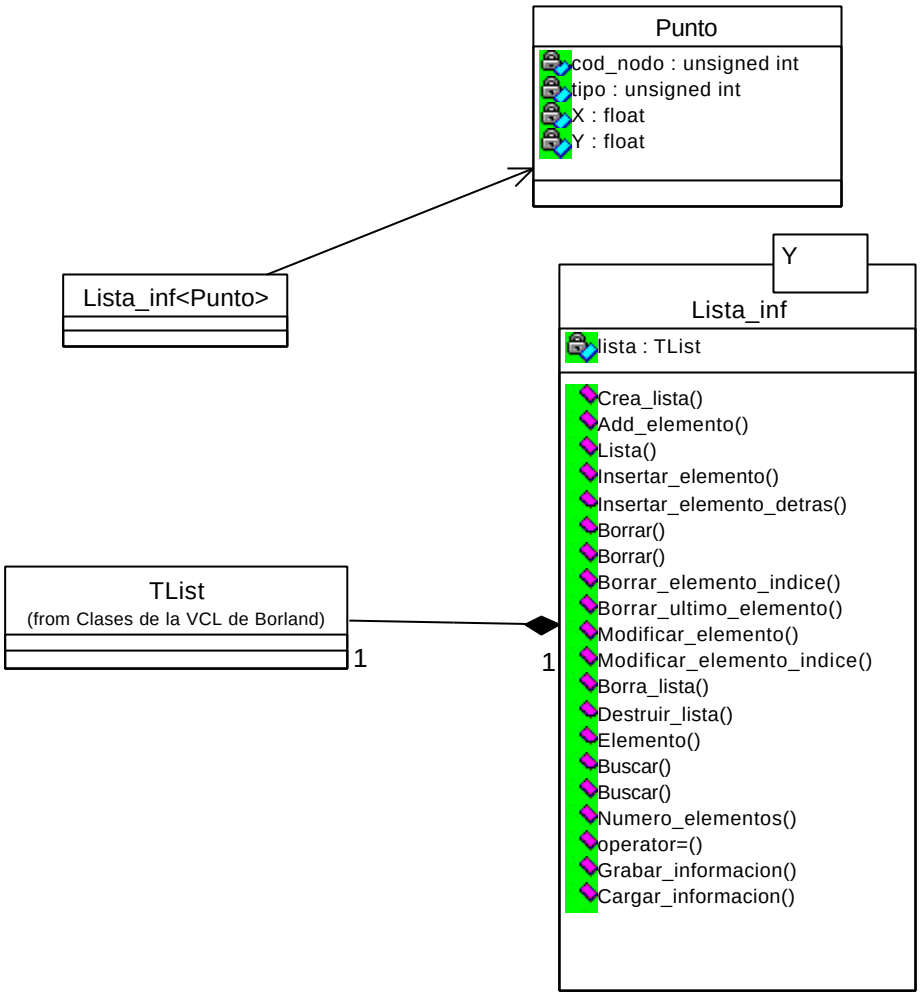

Figura 3.1.3: Clase plantilla

Interfaces

En primer lugar se recuerda la definición de interfaz. Interfaz era una colección de operaciones que se usa para especificar un servicio de una clase o componente.

Cada interfaz debe tener un nombre distintivo, una simple cadena de texto. Éste es el nombre simple. El nombre de camino consta del nombre de la interfaz precedido por el nombre del paquete en que se encuentra.

Normalmente, para no confundir nombres de clases con nombres de interfaz, a los nombres de éstos se les suele anteponer una I mayúscula, por ejemplo, Iinterfaz1.

Al contrario que las clases, las interfaces no especifican ninguna estructura, así que no pueden incluir atributos ni métodos.

Puede ser muy útil para entender el modelo dibujar una interfaz como una clase estereotipada, listando sus operaciones en el contexto apropiado.

Un estereotipo es un metatipo, equivale a incluir una nueva clase el metamodelo de UML. Cuando se aplica un estereotipo sobre un elemento como, por ejemplo, una clase, se está extendiendo UML, creando un nuevo bloque de construcción. Un estereotipo se representa como un nombre entre comillas tipográficas (<< >>) y se coloca sobre el nombre de otro elemento.

La relación entre un elemento y un interfaz se puede representar de dos formas:
- el elemento interacciona directamente con el interfaz, se recuerda que entonces el símbolo del interfaz es un círculo.
- el elemento interacciona con una clase estereotipada como interfaz, en la que se pueden detallar las operaciones del interfaz. A su vez la clase que proporciona el interfaz debe tener una relación de realización hacia la clase estereotipada como interfaz. Se recuerda que la realización es una mezcla entre la generalización y la dependencia.

Paquetes

Los paquetes tienen como propósito organizar elementos de modelado en grupos.

Cada paquete debe tener un nombre distintivo, una simple cadena de texto. Éste es el nombre simple. El nombre de camino consta del nombre de la interfaz precedido por el nombre del paquete en que se encuentra.

Cada elemento incluido en un paquete pertenece exclusivamente a ese paquete. Elementos de distintos tipos (clase, componente, etc...) pueden tener el mismo nombre dentro de un paquete. Se pueden tener dos clases llamadas iguales en paquetes diferentes.

Los paquetes pueden limitar la visibilidad de los elementos que contienen.

Mediante la importación es posible que una clase de un paquete interaccione con otra clase de otro paquete. En UML se modela como una relación de dependencia con estereotipo import.

También existe generalización entre paquetes, similar a la de las clases.

UML define cinco estereotipos estándar para los paquetes:
-facade: Especifica un paquete que sólo es una vista de otro paquete.
-framework: Especifica un paquete que consta principalmente de patrones.
-stub: Especifica un paquete que sirve de proxy para ver el contenido de otro.
-subsystem: Especifica un paquete que representa una parte independiente del sistema completo que se modela.
-system: Especfica un paquete que modela el sistema completo que se está modelando.

En el modelo hay varios paquetes que agrupan elementos con ciertas características comunes. Esto no implica que no haya relaciones entre elementos de distintos paquetes, que de hecho las hay.

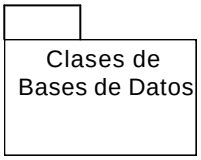

Figura 3.1.4: Paquete

En la aplicación también se usan relaciones de dependencia entre paquetes. De hecho todos los paquetes tienen dicha relación con los paquetes *Clases de Bases de Datos* y *Clases de la VCL de Borland*.

Instancias y objetos

Una instancia es una manifestación concreta de una abstracción a la que se puede aplicar un conjunto de operaciones y que posee un estado que almacena el efecto de las operaciones. Instancia y objeto son en gran parte sinónimos, ya que los objetos son abstracciones de clases y la mayoría de las instancias son abstracciones de clases. Se representa gráficamente subrayando su nombre.

Cada instancia debe tener un nombre distintivo, una simple cadena de texto. Éste es el nombre elemental. El nombre de camino consta del nombre de la interfaz precedido por el nombre del paquete en que se encuentra.

Cuando se da nombre a un objeto de forma explícita se le está dando un nombre utilizable por una persona. Sin embargo, en otros casos, el nombre real del objeto sólo es conocido por el computador en el que existe.

Cuando es necesario representar grandes colecciones de objetos se pueden modelar multiobjetos, que representan una colección de objetos anónimos.

Con UML también se puede mostrar el valor de los atributos de un objeto o el estado del mismo si la clase está asociada a una máquina de estados.

Todos los mecanismos de extensibilidad de UML pueden aplicarse a instancias. Sin embargo, normalmente el estereotipo no se aplica a la instancia, sino a la abstracción asociada (por ejemplo, la clase).

UML define dos estereotipos estándar que se aplican a las relaciones de dependencia entre objetos y clases:
- instanceof: El objeto origen es una instancia del clasificador de destino de la dependencia.
- Instantiate: La clase origen crea instancias de la clase destino de la dependencia

Y hay dos estereotipos que se aplican a los mensajes y transiciones:
- become: el destino es el mismo objeto que el origen pero en un instante posterior y con valores o estados diferentes.
- copy: el objeto destino es una copia exacta del de origen, pero completamente del origen.

Como ejemplo, se adjunta el objeto contexto de la clase Contexto, que interviene en algunos diagramas de secuencia.

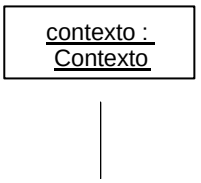

Figura 3.1.5: Objeto

La línea discontinua que aparece debajo de la caja es la línea de vida, como se verá en los diagramas de secuencia. El diagrama de secuencia es el único tipo de diagrama usado para modelar la aplicación que incluye objetos.

Casos de uso

Un caso de uso especifica el comportamiento de un sistema o parte del mismo, es una descripción de un conjunto de secuencias de acciones que ejecuta un sistema para producir un resultado observable de valor para un actor.

Se emplean para mostrar el comportamiento deseado del sistema en desarrollo, sin tener que especificar la implementación de ese comportamiento. Es decir, especifican un comportamiento deseado, no imponen como se llevará a cabo ese comportamiento.

Cada caso de uso debe tener un nombre distintivo, una simple cadena de texto. Éste es el nombre simple. El nombre de camino consta del nombre de la interfaz precedido por el nombre del paquete en que se encuentra. Normalmente se usa el nombre simple.

Un actor representa un conjunto de roles que los usuarios de los casos de uso juegan al interactuar con éstos. Un actor suele representar un rol que es jugado por una persona, un dispositivo hardware u otro sistema. Aunque se usan en los modelos, los actores no forman parte del sistema, son externos a él.

Los actores sólo pueden conectarse a los casos de uso mediante asociaciones. Una asociación entre un actor y un caso de uso indica que el actor y el caso de uso se comunican entre sí, y pueden enviarse y recibir mensajes.

El comportamiento de un caso de uso puede especificarse describiendo un flujo de eventos de forma textual, de forma suficientemente clara para que alguien ajeno al sistema lo entienda fácilmente. Normalmente se usa un diagrama de secuencia para especificar el flujo principal de una caso de uso, y se usan variaciones de ese diagrama para especificar lso flujos excepcionales del caso de uso.

Un escenario es una secuencia específica de acciones que ilustra un comportamiento. Por ello puede decirse que un escenario es a un caso de uso lo que un objeto a una clase: una instancia, un caso real.

La realización de un caso de uso puede especificarse explícitamente mediante una colaboración. Sin embargo la mayoría de las veces el caso de uso se especifica

exactamente por una colaboración, por lo que no será necesario mostrar explícitamente esa relación.

Los casos de uso pueden organizarse en paquetes, al igual que las clases. También pueden organizarse usando relaciones de generalización, inclusión y extensión entre ellos.

La generalización es como en las clases: el caso de uso hijo hereda el comportamiento del padre y puede añadir algo nuevo o redefinir el comportamiento del padre.

La relación de inclusión entre casos de uso significa que un caso de uso base incorpora explícitamente el comportamiento de otro caso de uso en el lugar especificado en el caso base. Se pueden usar relaciones de inclusión para evitar describir el mismo flujo de eventos repetidas veces, poniendo el comportamiento común en un caso de uso aparte, que será incluido por un caso de uso base. La relación de inclusión se especifica como una dependencia, estereotipada con include.

Por el contrario, la relación de extensión entre casos de uso significa que un caso de uso incorpora implícitamente el comportamiento de otro caso de uso en el lugar especificado indirectamente por el caso de uso que extiende la base. Esta relación puede usarse para modelar la parte de un caso de uso que el usuario puede ver como un comportamiento opcional del sistema. También para modelar un subflujo separado que se ejecuta sólo bajo ciertas condiciones o para modelar varios flujos que se pueden insertar en un punto dado, controlados por la interacción explícita con un actor. La relación de extensión se representa como una dependencia, estereotipada con extend.

Los casos de uso son clasifiadores y, por lo tanto, se pueden asociar a máquinas de estado. Éstas se pueden usar como otra forma de describir el comportamiento representado por un caso de uso.

En cualquier caso, para cada caso de uso debe incluirse una especificación del comportamiento, ya sea a través de texto, interacciones o una máquina de estados.

Un ejemplo de caso de uso que aparece en la aplicación puede ser el siguiente.

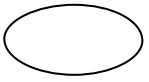

Figura 3.1.6: Caso de uso

Este caso de uso modela la secuencia de acciones que tienen que tener lugar para hacer posible las operaciones de zoom y windowing .

Componentes

Un componente es una parte física y reemplazable de un sistema que conforma con un conjunto de interfaces y proporciona la realización de esos interfaces. Los componentes se usan para modelar los elementos físicos que pueden encontrarse en un nodo, como ejecutables, bibliotecas, tablas, archivos y documentos.

Cada componente debe tener un nombre distintivo, una simple cadena de texto. Éste es el nombre simple. El nombre de camino consta del nombre de la interfaz precedido por el nombre del paquete en que se encuentra. Normalmente se usa el nombre simple.

Los componentes son muy parecidos a las clases: pueden realizar interfaces, participar en relaciones de dependencia, generalización y asociación, pueden anidarse, pueden tener instancias, pueden participar en interacciones... Pero también hay diferencias. Las clases representan abstracciones lógicas, pero los componentes representan elementos físicos. Además las clases pueden tener atributos y operaciones directamente accesibles. En general, los componentes sólo tienen operaciones que sólo pueden alcanzarse a través del interfaz.

Un interfaz que es realizado por un componente se denomina interfaz de exportación, es decir, el interfaz ofrece servicio a otros componentes. La interfaz utilizada por un componente se llama interfaz de importación, lo que significa que es una interfaz por el que el componente conforma y sobre el que se puede basar para construir.

Hay tres tipos de componentes:
- componentes de despliegue: componentes necesarios para formar un sistema ejecutable, por ejemplo, bibliotecas dinámicas (DLL), y ejecutables (EXE).
- componentes producto del trabajo: son productos que quedan al final del proceso de desarrollo. Se han usado durante el proceso de desarrollo pero no forman parte del sistema ejecutable, como por ejemplo, los archivos de código fuente o de datos.
- componentes de ejecución: Se crean como consecuencia de un sistema en ejecución, como un objeto COM + a partir de una DLL.

Los componentes se pueden agrupar en paquetes al igual que las clases, y también especificando relaciones de dependencia, generalización, asociación y realización.

UML define cinco estereotipos estándar para componenetes:
- executable: es un componente que se puede ejecutar en un nodo.
- library: especifica una biblioteca de objetos estática o dinámica.
- table: especifica un componente que representa una tabla de una base de datos.
- file: especifica un componente que representa un documento que contiene código fuente o datos.
- document: especifica un componente que representa un documento.

Nodos y componentes

Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional que suele tener memoria y capacidad de procesamiento.

Cada nodo debe tener un nombre distintivo, una simple cadena de texto. Éste es el nombre simple. El nombre de camino consta del nombre de la interfaz precedido por el nombre del paquete en que se encuentra. Normalmente se usa el nombre simple.

Los nodos son muy parecidos a los componentes, pero existen diferencias entre ellos. En primer lugar, los componentes son los elementos que participan en la ejecución de un sistema: los nodos son los elementos donde se ejecutan los componentes.

En segundo lugar, los componentes representan el empaquetamiento físico de los elementos lógicos, mientras que los nodos representan el despliegue físico de los componentes.

La relación entre un nodo y el componente que despliega puede mostrarse explícitamente como una relación de dependencia.

Los nodos se pueden agrupar en paquetes, al igual que mediante relaciones de dependencia, generalización y asociación entre ellos.

El tipo más común de relación entre nodos es la asociación. En este caso representa una conexión física entre ambos: cualquier medio físico que los conecte: Ethernet, línea serial, etc...

Además en cada nodo se pueden incluir atributos, como memoria RAM y velocidad y tipo de CPU, por ejemplo, en el caso de un servidor. Otra opción es dividir el equipo entre distintos dispositivos y modelar cada uno de ellos como un nodo.

Colaboración

Una colaboración designa una sociedad de clases, interfaces y otros elementos que colaboran para proporcionar algún comportamiento cooperativo mayor que la suma de los comportamientos de sus elementos. También es la especificación de cómo un elemento tal como un clasificador (interfaz, componente, nodo, ...) o una operación es realizada mediante un conjunto de clasificadores y asociaciones.

Cada colaboración debe tener un nombre distintivo, una simple cadena de texto. Éste es el nombre simple. El nombre de camino consta del nombre de la interfaz precedido por el nombre del paquete en que se encuentra. Normalmente se usa el nombre simple.

Las colaboraciones tienen dos aspectos, una parte estructural que especifica las clases, interfaces y otros elementos que colaboran para llevar a cabo la colaboración, y una parte dinámica que especifica la dinámica de cómo interactúan esos elementos.

La parte estructural de una colaboración puede incluir cualquier combinación de clasificadores. Éstos pueden organizarse usando las relaciones normales de UML. Pero, al contrario que los paquetes, las colaboraciones no contienen a sus elementos estructurales, pero hace referencia a ellos y los utiliza. Es por esto que una colaboración puede atravesar muchos niveles de un sistema. Un mismo elemento puede formar parte de varias colaboraciones o no pertenecer a ninguna. Es posible mirar “dentro” de una colaboración para ver sus componentes. Estos componentes pueden aparecer formando un conjunto de clases.

La parte de comportamiento, sin embargo, se representa normalmente por un (o varios) diagrama de interacción. Por otra parte, si se desea destacar la parte temporal se puede hacer un diagrama de secuencia.

Una colaboración puede refinar a otras colaboraciones.

Las colaboraciones suelen usarse principalmente en el modelado de la realización de casos de uso. Normalmente, un caso de uso estará modelado por una o más colaboraciones, usando relaciones de realización.

Máquinas de estados

Antes de entrar con las máquinas de estado será práctico ver algo sobre señales y eventos.

Un evento es la especificación de un acontecimiento significativo que ocupa un lugar en el tiempo y en el espacio. Una señal es un tipo de evento que representa la especificación de un estímulo asíncrono que es transmitido entre instancias.

Hay dos tipos de eventos, internos y externos. Los externos fluyen entre el sistema y los actores (click de un ratón, por ejemplo) y los internos fluyen entre los objetos del sistema.

En UML se pueden modelar cuatro tipos de eventos:
- señales: representa un objeto con nombre que es lanzado asíncronamente por un objeto y recibido por otro. El lenguaje de programación C++ soporta señales.
- llamadas: representa la invocación de una operación. Se modela igual que las señales, pero en este caso el evento suele ser síncrono.
- de tiempo: representa el paso del tiempo. Se modela usando la palabra after seguida de una expresión para evaluar el tiempo.
- de cambio: representa un cambio en el estado o el cumplimiento de alguna condición. Se modela con la palabra when seguida de una expresión booleana, es decir, que devuelva verdadero o falso.

En UML los eventos de llamada que puede recibir un objeto se modelan como operaciones sobre la clase del objeto. Las señales con nombre que puede recibir un objeto se modelan designándolas en un compartimento extra de la clase.

En UML las excepciones son también un tipo de señales, que se modelan como clases estereotipadas con el estereotipo exception.

Volviendo a las máquinas de estado, éstas pueden usarse para modelar el comportamiento dinámico de un objeto individual. Las máquinas de estado modelan la vida de un único objeto.

Las máquinas de estado pueden visualizarse de dos formas. La primera es mediante un diagrama de actividades, centrándose en las actividades que tienen lugar dentro del objeto. La segunda es mediante la utilización de diagramas de estados, centrándose en el comportamiento dirigido por eventos.

Una máquina de estados es un comportamiento que especifica las secuencias de estados por las que pasa un objeto a lo largo de su vida en respuesta a eventos, junto con sus respuestas a estos eventos. Es la mejor forma de modelar objetos que dependen de estímulos asíncronos o cuyo estado depende del pasado.

Un estado es una condición o situación en la vida de un objeto durante la cual cumple una serie de condiciones o espera algún evento. Un estado tiene varias partes:
- nombre: para distinguir un os estados de otros.
- acciones de entrada/salida: acciones ejecutadas al entrar y salir del estado, sin importar qué transición fue la que condujo al objeto el estado. La acción de entrada se etiqueta con el evento entry y la de salida con el evento exit.
- transiciones internas: transiciones que se manejan sin provocar cambio de estado.
- subestados: un estado puede englobar subestados disjuntos (activos secuencialmente) o concurrentes (activos concurrentemente).
- eventos diferidos: lista de eventos que no se manejan en este estado y se añaden a una cola para ser manejados en otro momento.

Existen dos estado especiales, el estado inicial y el estado final. El estado inicial indica el punto de comienzo por defecto para la máquina de estados o el subestado. Se representa por un círculo negro. El estado final indica el fin de la ejecución de la máquina de estados. Se representa como un círculo negro dentro de un círculo blanco.

Una transición es una relación entre estados que indica que un objeto que esté en un estado realizará ciertas acciones y entrará en un segundo estado cuando ocurra un evento especificado y se satisfagan unas condiciones especificadas. Cuando se produce un cambio de estado la transición se ha disparado.

Una transición tiene 5 partes:
- estado origen: el estado afectado por la transición
- evento de disparo: el evento cuya recepción por el estado origen provoca el disparo de la transición si se satisface la condición de guarda.
- condición de guarda: expresión booleana que se evalúa cuando la transición se activa por la recepción del evento de disparo. Si la expresión es cierta, la transición se dispara.
- Acción: computación atómica ejecutable que puede actuar sobre el objeto asociado a la máquina de estado o sobre otros objetos visibles al objeto.
- estado destino: estado activo tras producirse la transición.

Algunas veces se produce una transición desde un estado hasta ese mismo estado. Puede que en esas ocasiones no sea interesante ejecutar de nuevo las acciones de entrada y salida. Para que estas acciones no se ejecuten se debe realizar una transición interna.

Se puede usar la transición especial do para indicar la acción a realizar mientras no se produzca una transición.

Se puede usar la transición especial defer para añadir un evento a la lista de eventos diferidos. Estos eventos se procesarán cuando se llegue a un estado en el que no aparezcan marcados como defer.

Un subestado es un estado anidado dentro de otro. Los estados que tienen subestados se llaman estados compuestos. En UML se representa igual que uno simple, pero con un compartimento gráfico opcional donde se muestra una máquina de estados anidada.

Un estado de historia permite que un estado compuesto que contiene subestados secuenciales recuerde el último subestado activo antes de la transición que provocó la salida del estado compuesto. En UML se representa como un círculo con un H en su interior. La historia no se guarda si la máquina de subestados alcanza la posición final.

A veces puede ser necesario usar subestados concurrentes. Estos subestados permiten especificar dos o más máquinas de estados que se ejecutan en paralelo en el contexto del objeto que las contiene.

Clases activas

Un objeto activo es aquel que tiene un proceso o hilo y puede iniciar actividad de control. Una clase activa es aquella cuyas instancias son objetos activos.

En un sistema concurrente habrá varios flujos de control, al contrario que en un sistema secuencial en el que hay uno solo. En UML las clases activas representan un proceso o hilo que constituyen la raíz de un flujo de control independiente, que será concurrente con otros flujos de control.

UML define dos estereotipos estándar que se pueden usar con las clases activas:
- process: especifica flujo pesado que se puede ejecutar concurrentemente con otros procesos.
- thread: especifica un flujo ligero que se puede ejecutar concurrentemente con otros hilos dentro del mismo proceso.

Los objetos pueden intercambiar mensajes entre sí, independientemente de si los emisores o los receptores son clases activas o no.

La concurrencia tiene el problema de que si en un momento dado existen varios flujos de control dentro de un mismo objeto, el estado de éste se puede corromper. Es decir, hay que tratar a los objetos como regiones críticas.

En UML hay tres enfoques para solucionar el problema de la exclusión mutua sobre la región crítica:
- secuencial: los emisores deben coordinarse fuera del objeto para que un instante dado sólo exista un flujo de control sobre el objeto.
- con guardas: la integridad del objeto se garantiza en presencia de múltiples flujos de control, tratando secuencialmente las llamadas a las operaciones con guardas del objeto. Esto en realidad es “secuenciar” la concurrencia.
- Concurrente: la integridad del objeto se garantiza en presencia de múltiples flujos de control tratando las operaciones como atómicas. Esto quiere decir que o se completa totalmente la operación o no se hace ninguna modificación. Es un problema que en bases de datos se resuelve mediante transacciones.

Interacciones

Una interacción es un comportamiento que incluye un conjunto de mensajes intercambiados entre objetos dentro de un contexto para lograr un propósito.

Sirven para modelar los aspectos dinámicos de las colaboraciones. Los aspectos dinámicos se especifican, construyen y documentan como flujos de control que pueden incluir hilos secuenciales a través de la misma o flujos complejos que impliquen bifurcaciones. Las interacciones pueden modelarse de dos formas: de acuerdo a la ordenación temporal de los mensajes (se representa en un diagrama de secuencia) o según la secuencia de mensajes en el contexto de una organización estructural de objetos (se representa en un diagrama de interacción). Los objetos que participan en una interacción son elementos concretos (un objeto que representa algo del mundo real) o elementos prototípicos, es decir, un objeto que represente a cualquier instancia de una clase.

Un mensaje es la especificación de una comunicación entre objetos que transmite información, con la expectativa de que de desencadenará alguna actividad. La recepción de una instancia de un mensaje puede considerarse una instancia de un evento.

Un enlace es una conexión semántica entre objetos, es una instancia de una asociación. Siempre que una clase tenga una asociación con otra clase, podría existir un enlace entre las instancias de las dos clases. Siempre que haya un enlace entre dos objetos uno puede enviarle mensajes al otro.

Existen los siguientes estereotipos estándar para los extremos del enlace:
- association: indica que el objeto correspondiente es visible por existir una asociación.
- self: indica que el objeto correspondiente es visible porque es el invocador de la operación.
- global: indica que el objeto correspondiente es visible porque su alcance contiene al actual.
- local: indica que el objeto correspondiente es visible porque su alcance es local.
- parameter: indica que el objeto correspondiente es visible porque es un parámetro.

La multiplicidad no se aplica a los enlaces, al ser instancias de asociaciones.

Cuando se pasa un mensaje, la acción resultante es una instrucción ejecutable que constituye una abstracción de un procedimiento computacional. Además una acción puede provocar un cambio de estado.

En UML se pueden modelar varios tipos de acciones:
- llamada: invoca una operación sobre un objeto. Un objeto puede enviarse un mensaje a sí mismo, esto se conoce como invocación local de la operación.
- retorno: devuelve un valor al invocador.
- envío: envía una señal a un objeto.
- creación: crea un objeto.
- destrucción: destruye un objeto. Puede destruirse a sí mismo.

Un flujo de mensajes forma una secuencia. El inicio de cada secuencia debe tener su origen en algún proceso o hilo. Cada proceso e hilo dentro de un sistema define un flujo de control distinto, y dentro de cada flujo los mensajes tendrán lugar secuencialmente. Para visualizar mejor esta secuencia, se puede expresar explícitamente la posición de cada mensaje dentro de la secuencia, mediante un número de secuencia. El número de secuencia se indica antes del nombre del mensaje y separado de éste por dos puntos.

Pueden existir flujos anidados. En este caso la punta de flecha del mensaje se representa como sólida. Al haber mensajes anidados éstos se numeran como mensajes anidados dentro de una secuencia: se indica el número de la secuencia, y separado por un punto, se indica a continuación la posición del mensaje dentro de la secuencia. Finalmente se añaden los dos puntos y a continuación el nombre del mensaje.

Cuando se modelan interacciones que incluyen varios flujos de control hay que especificar el proceso o flujo antes del número de secuencia del mensaje.

En UML se pueden especificar no sólo los parámetros enviados, que se representan entre paréntesis tras el nombre, sino también los valores devueltos. El valor devuelto se representa tras el número de secuencia, separado de éste por dos puntos. A continuación se añaden otros dos puntos y el nombre el mensaje.

En algunas interacciones alguno de los objetos o enlaces puede crearse o destruirse. En UML se le pueden asociar las siguientes restricciones al elemento:

- new: la instancia o enlace se crea durante la ejecución de la interacción que la contiene.
- destroyed: la instancia o enlace se destruye antes de completarse la ejecución de la interacción que la contiene.
- transient: la instancia o enlace se crea durante la ejecución de la interacción que la contiene, pero se destruye antes de completarse la operación.

A lo largo de una interacción un objeto puede cambiar los valores de sus atributos o su estado. Esta modificación se puede modelar incluyendo copias del objeto, con los valores de sus atributos diferentes. En un diagrama de secuencia se colocaría cada variante del objeto en la misma línea de vida del objeto. En un diagrama de interacción, se conectaría cada variante con un mensaje become.

Un ejemplo sencillo de interacción puede ser la llamada un objeto para ejecutar una de sus operaciones. Otra interacción sería la respuesta de dicho objeto: bien la devolución de un valor o la confirmación de que la operación se ha realizado satisfactoriamente.

A continuación se muestra el diagrama de secuencia que describe los pasos necesarios para que cualquier interfaz pueda representar las dos imágenes que representan la red vial.

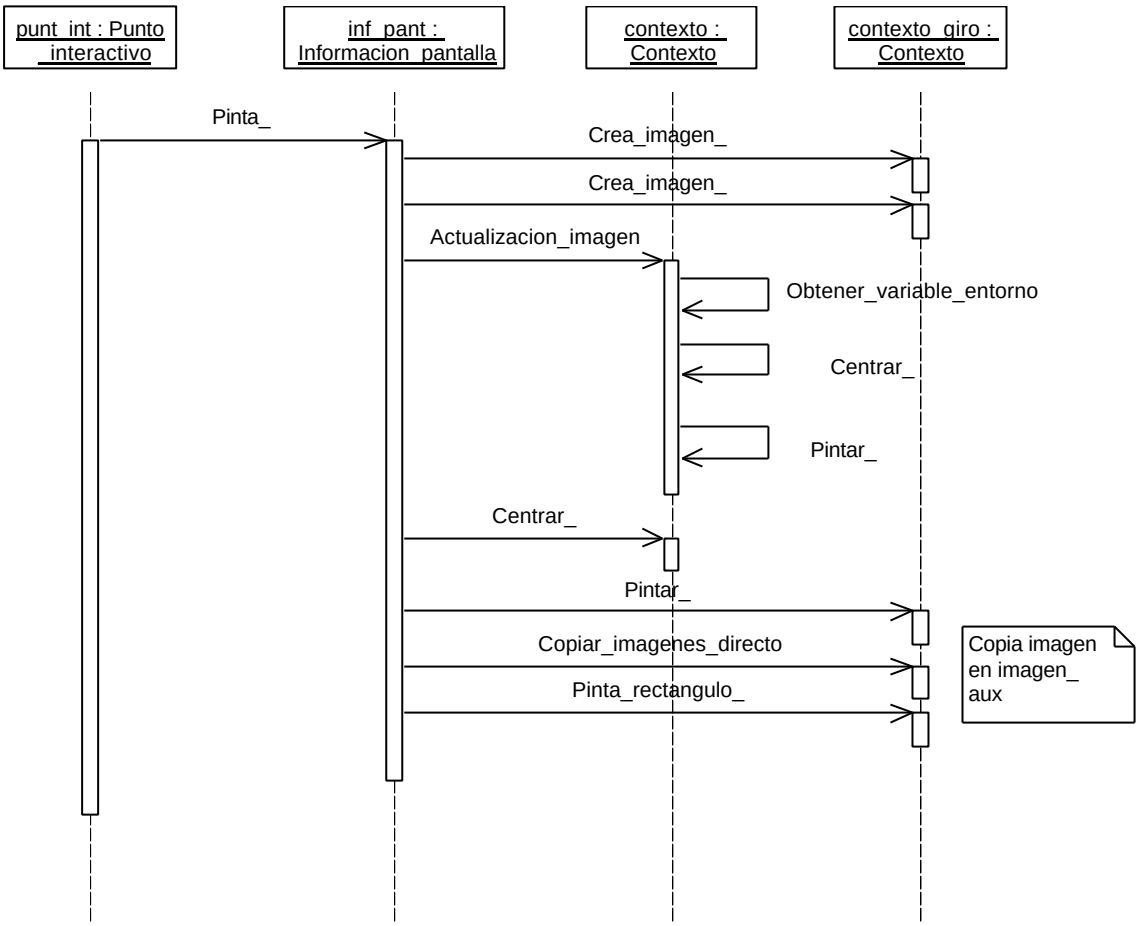

Figura 3.1.7: Mensajes

En este diagrama de secuencia se aprecian las llamadas desde un objeto a operaciones que pertenecen a otros. Estas llamadas se modelan como interacciones, que son las flechas que van desde un objeto a otro.
Puede apreciarse como todas las operaciones que se han usado no devuelven ningún valor (*void*).

Relaciones

Una vez vistos los principales elementos en UML, es hora de ver las relaciones posibles entre dichos elementos.

Aunque hay tres tipos de relaciones, dependencias, generalizaciones, asociaciones y realizaciones, las tres primeras son las más importantes y suelen usarse en los diagramas de clases.

Las dependencias son relaciones de uso. Las generalizaciones conectan clases más generales con clases más especializadas. Se conocen como relaciones padre/hijo y superclase/subclase. Las asociaciones son relaciones estructurales entre elementos.

No es bueno modelar relaciones en exceso, pues se hace el modelo incomprensible, pero tampoco demasiado poco, pues se pierde información del sistema.

Dependencia

Es una relación de uso que declara que el cambio en la especificación de un elemento puede afectar a otro elemento que la utiliza, pero no necesariamente a la inversa. Normalmente se usa con las clases.

Las dependencias pueden tener nombres pero es raro que se necesiten.

Aunque no suele ser necesario, las dependencias pueden incluir muchos matices. Estos se consiguen aplicando estereotipos.

Hay 8 estereotipos aplicables a las dependencias entre clases y objetos en un diagrama de clases:
- bind: el origen de la dependencia instancia a la plantilla destino con los parámetros reales dados.
- derive: el origen puede calcularse a partir del destino.
- friend: el origen tiene visibilidad especial en el destino. Se corresponde con las clases friend de C++.
- instanceOf: el origen es una instancia del clasificador destino.
- instantiate: el origen crea instancias del destino.
- powertype: el destino es un supratipo del origen. Un supratipo es un clasificador cuyos objetos son todos los hijos de un padre dado.
- refine: el origen está en un grado de abstracción más detallada que el destino.
- use: la semántica del elemento origen depende de la del destino.

Hay dos estereotipos que se aplican a la dependencia entre paquetes:
- access: el paquete origen tiene permiso para referenciar elementos del paquete destino
- import: un tipo de acceso que especifica que los contenidos públicos del paquete destino entran en el espacio de nombres de origen, como si hubiesen sido declarados en el origen.

Hay dos estereotipos que se aplican a la dependencia entre casos de uso:
- extend: el caso de uso destino extiende el comportamiento del origen.
- include: el caso de uso origen incorpora explícitamente el comportamiento del destino en la posición especificada por el origen.

Hay tres estereotipos que se aplican a las interacciones entre objetos:
- become: el destino es el mismo objeto que el origen pero han cambiado algunos atributos, o el estado, etc... Representa un instante posterior.
- call: la operación de origen invoca a la de destino.
- copy: el objeto destino es una copia exacta pero independiente del origen.

Para las máquinas de estado hay un estereotipo:
- send: la operación origen envía el evento destino.

Hay un estereotipo que se usa en el contexto de la organización de los elementos de un sistema en subsistemas y modelos.
- trace: el destino es un antecesor histórico del origen.

Por ejemplo, en el modelo se da una dependencia en caso de, por ejemplo, la clase *Tramo*. Para definir un tramo hay que apoyarse en los dos nodos que tiene como extremos. Por ello aparece una relación de dependencia entre ellos. En este caso el estereotipos sería use. Este tipo de dependencia es la que más se ha dado en el modelado de la aplicación.

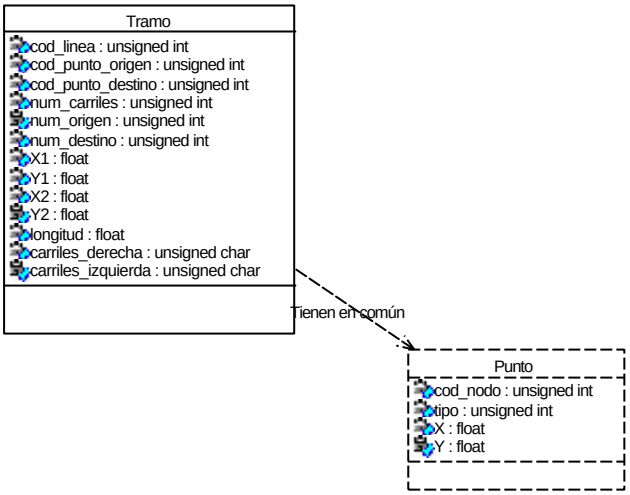

Figura 3.1.8: Dependencia

A la dependencia se la ha denominado *Tienen en común*. Este nombre hace referencia a que cada objeto de la clase *Tramo* implícitamente contiene dos nodos (aunque en la definición de la clase lo que aparecen son sus códigos), que son los extremos, el origen y el destino.

Generalización

Es una relación entre un elemento general (superclase o clase padre) y un caso más específico (subclase o clase hijo). Significa que la subclase puede sustituir a la superclase.

Las clases sin padre se llaman clases raíz o clases base. Una clase sin hijos se denomina clase hoja. Una clase con único padre usa herencia simple, si tiene varios padres, usa herencia múltiple. Son muy frecuentes entre clases e interfaces.

El estereotipo implementation puede usarse para indicar que el hijo hereda la implementación del padre pero no hace públicos los interfaces y por tanto no los soporta, violando así la semántica de la sustitución. Se usa al modelar herencias privadas en C++.

Y se pueden aplicar 4 restricciones estándar:
- complete: todos los hijos en la generalización se han especificado en el modelo y no se admiten hijos adicionales.
- incomplete: no se han especificado todos los hijos en la generalización y se permiten más.

- disjoint: los objetos padre no pueden tener más de uno de los hijos como tipo.
- overlapping: los objetos del padre pueden tener más de uno de los hijos como tipo.

Asociación

Es una relación estructural que especifica que los objetos de un elemento están conectados con los objetos de otro.

Hay cuatro adornos que se pueden añadir a las asociaciones:
- nombre: se usa para describir la naturaleza de la relación. Se le puede asignar una dirección al nombre para que no exista ambigüedad.
- rol: cada clase que participa en una asociación tiene un rol específico en ella. Un rol es simplemente la parte de la clase que uno de los extremos presenta al otro. Se puede nombrar el rol de cada uno de los extremos. Una misma clase puede jugar el mismo o diferentes roles en distintas asociaciones.
- multiplicidad: sirve para indicar cuántos objetos pueden conectarse a través de una instancia de una asociación.
- agregación: cuando se desea modelar una relación en la cual una clase está formada por otras más pequeñas. Esta relación se llama agregación y se especifica añadiendo un rombo vacío en el extremo de la asociación donde está la clase contenedora de la otra u otras. También existen otras agregaciones donde el rombo aparece relleno, como se verá.

Y también hay una serie de propiedades:
- navegación: por defecto la navegación entre los objetos de un extremo y el otro es bidireccional. Se puede limitar a una dirección añadiendo una punta de flecha a la asociación.
- visibilidad: por defecto, los objetos de una clase pueden ver y navegar hasta los objetos de la otra, a menos que se restrinja mediante la navegación. Mediante la visibilidad se puede indicar la visibilidad de los objetos de un extremo desde los objetos externos a la asociación. Puede ser pública, privada y protegida.
- calificación: un calificador es un atributo de una asociación cuyos valores particionan el conjunto de objetos relacionados con un objeto a través de una asociación. Se representa como un rectángulo junto al extremo de la asociación con los atributos dentro.
- especificador de interfaz: se usan para definir el rol que presenta un objeto a la asociación. Así, pueden definirse interfaces distintos según el rol que juegue el objeto.
- composición: es una variación de la agregación simple, pero con una fuerte relación de pertenencia y vidas coincidentes de la parte o partes con el todo. Esto quiere decir que las partes una vez creadas viven y mueren con el todo. En una agregación compuesta, un objeto puede formar parte de sólo una parte compuesta (un todo a la vez). En la agregación simple, una parte podía pertenecer a varios todos. La composición se representa rellenando el rombo de la agregación.
- clases asociación: la asociación puede tener sus propias propiedades. Se representa con un símbolo de clase unido por línea discontinua a la asociación.
- restricciones: UML define 5 restricciones que pueden aplicarse a asociaciones:
    - implicit: la relación no es manifiesta, sino conceptual.
    - ordered: el conjunto de objetos de un extremo de una asociación sigue un orden explícito.

- changeable: se pueden añadir, eliminar y modificar libremente los enlaces entre objetos.
- addOnly: se pueden añadir nuevos enlaces desde un objeto del otro extremo de la asociación.
- frozen: un enlace, una vez añadido desde un objeto del otro extremo de la asociación, no se puede modificar ni eliminar.

La composición se entiende más fácilmente si se mira en un lenguaje de programación concreto como C++. Cuando una clase, por ejemplo la clase A, tiene un atributo que es un objeto de la clase B llamado objetoB, en UML objetoB se modela como un atributo de la clase A cuyo tipo es la clase B.

Si la clase A no tuviese como atributo el objetoB, sino un puntero a un objeto de la clase B, por ejemplo, punteroB, en UML puntero B también se modelaría como un atributo de la clase A cuyo tipo es la clase B.

¿ Cuál es la diferencia entre los dos casos ? En el primer caso, el objetoB forma parte de la clase A. Cuando se cree un objeto de la clase A, se creará el objeto objetoB para esa clase. Cuando se destruya un objeto de la clase A, se destruirá también el objeto objetoB correspondiente. Cada objeto de la clase A tiene su objeto objetoB independiente de los demás objetos objetoB de cada uno de los objetos de la clase A.

En el segundo caso, el puntero se usará para contener la dirección de memoria de un objeto de la clase B. Sin embargo, varios objetos punteroB de distintos objetos de la clase A pueden contener la misma dirección de memoria, de manera que esos objetos estarían compartiendo el atributo del tipo clase B en UML. Además, puede destruirse un objeto de la clase A sin que se destruya el objeto de la clase que tiene atributo, que puede seguir siendo usado por los demás objetos de la clase A. Este atributo no se destruirá hasta que alguno de los objetos de la clase A libere la memoria ocupada por él.

El primer caso es una composición, el segundo es una agregación.

La asociación simple es quizá la relación más usada, ya que si una clase llama a un método de otra ya hay entre ellas una asociación.

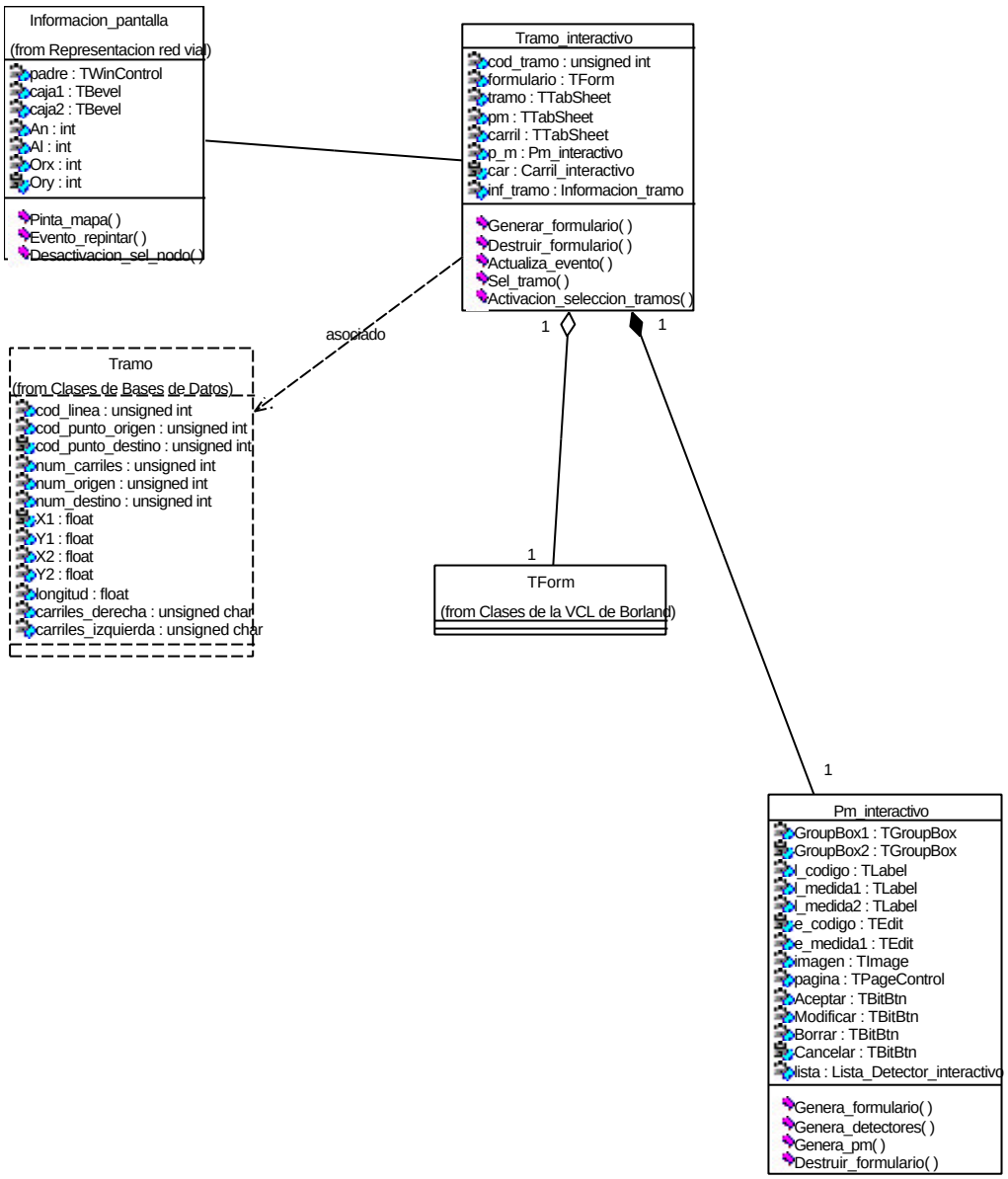

Figura 3.1.9: Asociación, agregación y composición

En la figura se representa una porción del diagrama de clases perteneciente al paquete Interfaz de tramos donde se muestran varios tipos de relaciones.

La clase *Tramo_interactivo* tiene una relación de asociación con Informacion_pantalla porque usa el método *Pinta_mapa* de dicha clase. También tiene una relación de dependencia con la clase *Tramo*, porque toda la información a la que se va a acceder hace referencia a un tramo, en definitiva, un objeto de la clase *Tramo*.

También existe una relación de agregación con la clase *TForm*, ya que la clase *Tramo_interactivo* contiene una propiedad que es un puntero a objeto de la clase *TForm*. Y finalmente existe una relación de composición con la clase *Pm_interactivo*, porque una de las propiedades de la clase *Tramo_interactivo* es un objeto de la clase *Pm_interactivo.*

Realización

Es una relación semántica entre clasificadores, en la cual un clasificador especifica un contrato que otro clasificador garantiza que cumplirá. Se representa como una línea discontinua con una gran punta de flecha vacía que apunta al clasificador que especifica el contrato.

Semánticamente es una mezcla entre dependencia y asociación. Se utilizará en dos circunstancias: en el contexto de interfaces y en el contexto de colaboraciones.

En el contexto de interfaces se usa para representar un interfaz en la forma canónica, con la clase estereotipada como interfaz y la clase que lo implementa.

En el contexto de colaboraciones se usa para especificar la relación entre un caso de uso y la colaboración que implementa ese caso de uso.

Diagramas

Por último, hay que estudiar los distintos tipos de diagramas existentes en UML sobre los que mostrar los distintos elementos y relaciones.

Diagrama de clases

Son los más utilizados en el modelado de sistemas orientados a objetos. Muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. También pueden contener paquetes o subsistemas que se usan para agrupar a los elementos del modelo.

Se usan para modelar la vista de diseño estática del sistema. Esta vista debe mostrar los requisitos funcionales del sistema, los servicios que éste debe proveer a los usuarios finales.

Con los diagramas de clases se puede hacer ingeniería directa e ingeniería inversa. La ingeniería directa consiste en obtener el código a partir del modelo. Normalmente se produce una pérdida de información, ya que el modelo UML es más rico en matices que cualquier lenguaje de programación orientado a objetos. La ingeniería inversa es el proceso de transformar el código en un modelo. La ingeniería inversa también es incompleta. El modelo creado tendrá, en general, un bajo nivel de detalle.

Por supuesto que para los dos tipos de ingeniería se precias especificar el lenguaje que se va a usar para leer código o crearlo. Debe ser un lenguaje orientado a objetos como C++, Java; Smalltalk, etcétera.

La figura incluida en las asociaciones es una parte de un diagrama de clases.

Diagrama de objetos

Modelan las instancias de los elementos contenidos en los diagramas de clases. Muestra un conjunto de objetos y sus relaciones en un momento concreto, ya que se usa para modelar tanto la vista de diseño como la de procesos pero en ambos casos estática.

Los diagramas de objetos incluyen objetos y enlaces, aunque también pueden contener objetos y subsistemas.

Diagrama de casos de uso

Se usan para modelar la vista de casos de uso estática del sistema. Su misión principal es documentar y especificar el comportamiento de un elemento. Así, los usuarios deben saber cómo utilizarlo y los desarrolladores pueden implementarlo.

Los diagramas de casos de uso contienen casos de uso, actores y relaciones de dependencia, generalización y asociación. También pueden contener paquetes.

Se pueden usar para modelar el contexto del sistema, asegurando los actores que interactúan con él y sus roles. Pero también pueden usarse para modelar los requisitos del sistema, especificando qué debería hacer el sistema, independientemente de cómo lo haga.

Un ejemplo de diagramas de casos de uso es la relación existente entre el usuario del sistema y uno de los interfaces, por ejemplo el interfaz de nodos.

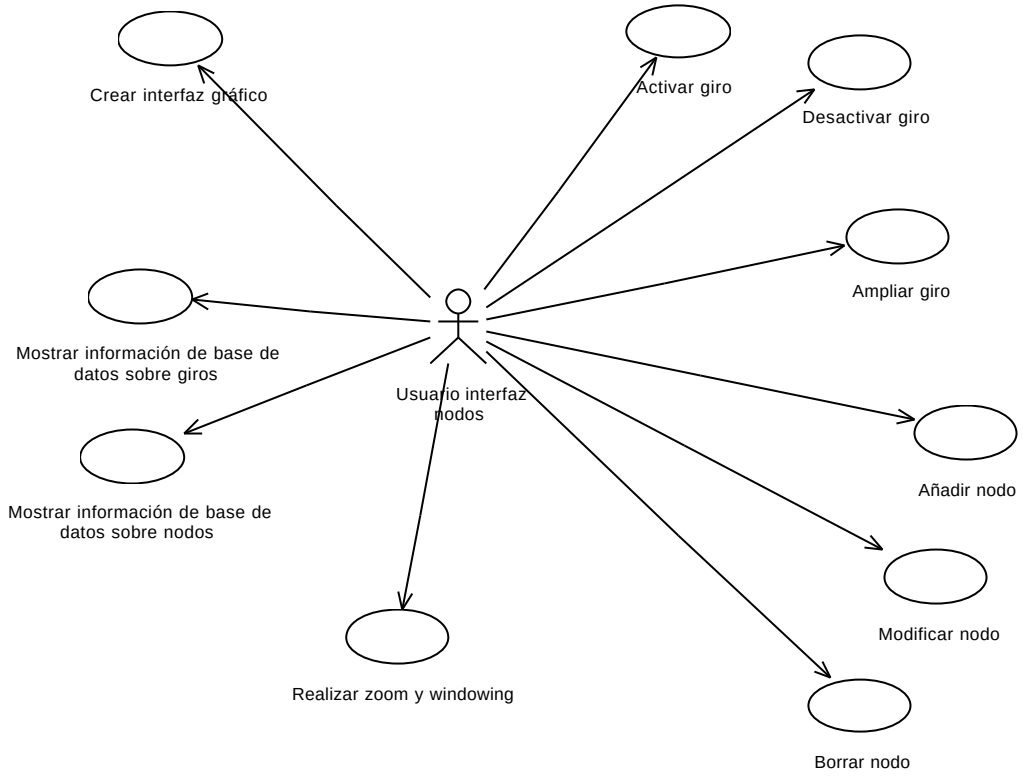

Figura 3.1.10: Diagrama de casos de uso

En este diagrama se muestran todas las posibles operaciones que puede realizar el usuario cuando se encuentra en el interfaz de nodos. Cada uno de estos casos de uso se implementará de una cierta manera que en este tipo de diagrama no tiene importancia, pero que se describe en otro tipo de diagrama. En el caos de esta aplicación, la implementación de los diagramas de casos de uso se detalla en los diagramas de secuencia.

Diagramas de interacción

Bajo esta denominación se agrupan los diagramas de secuencia y los diagramas de colaboración. El diagrama secuencial destaca la ordenación temporal de los mensajes, mientras que el diagrama de colaboración destaca la organización estructural de los objetos que reciben y envían mensajes.

Los diagramas de interacción se usan para modelar los aspectos dinámicos del sistema.

El diagrama de secuencia consta de una serie de objetos (instancias de clases, normalmente) dispuestos a lo largo del eje X y los mensajes se representan ordenadamente en el tiempo en el eje Y. El diagrama de colaboración es una colección de nodos y arcos.

Normalmente los diagramas de interacción contienen objetos, enlaces y mensajes.

En los diagramas de secuencia los objetos tienen lo que se conoce por línea de vida. Es una línea discontinua vertical que representa la existencia de un objeto a lo largo de un período de tiempo. Pueden crearse y destruirse objetos con mensajes estereotipados como create y destroy, respectivamente.

En los diagramas de secuencia el foco de control es un rectángulo delgado y estrecho que representa el período de tiempo durante el cual un objeto ejecuta una acción, directamente o a través de procedimientos subordinados, por ejemplo, llamadas a métodos o funciones. Pueden existir anidamientos de focos de control, por ejemplo, una función recursiva. Esto se representaría añadiendo un nuevo foco de control a la derecha del anterior.

En los diagramas de colaboración se representan en primer lugar los objetos que participan en la colaboración. A continuación se representan los enlaces que conectan esos objetos.

Hay dos características principales que diferencian los diagramas de colaboración de los de secuencia. La primera de ellas es la existencia de camino. Para indicar cómo se enlaza un objeto a otro se asocia un estereotipo de camino al extremo más lejano de un enlace. Sólo es necesario representar explícitamente los caminos local, parameter, global y self. La segunda de ellas es el número de secuencia. Se usa para especificar la ordenación temporal de los mensajes. Para representar la anidación se usa la notación decimal de Dewey. Esto quiere decir que si se tiene un mensaje 1, los mensajes anidados serán 1.1 para el primer mensaje anidado dentro del mensaje 1, 1.2 será el segundo mensaje anidado dentro del mensaje 1, y así sucesivamente.

Para modelar un flujo más complicado, como una iteración, se puede expresar el número de secuencia del mensaje como una expresión de iteración. También se puede expresar el número de secuencia como una condición, de manera que se puedan modelar bifurcaciones del flujo de control.

Ambos diagramas son equivalentes semánticamente.

Este es el diagrama de secuencia que sirve para definir la implementación del caso de uso *Cargar datos de la red vial para representar gráficamente.*

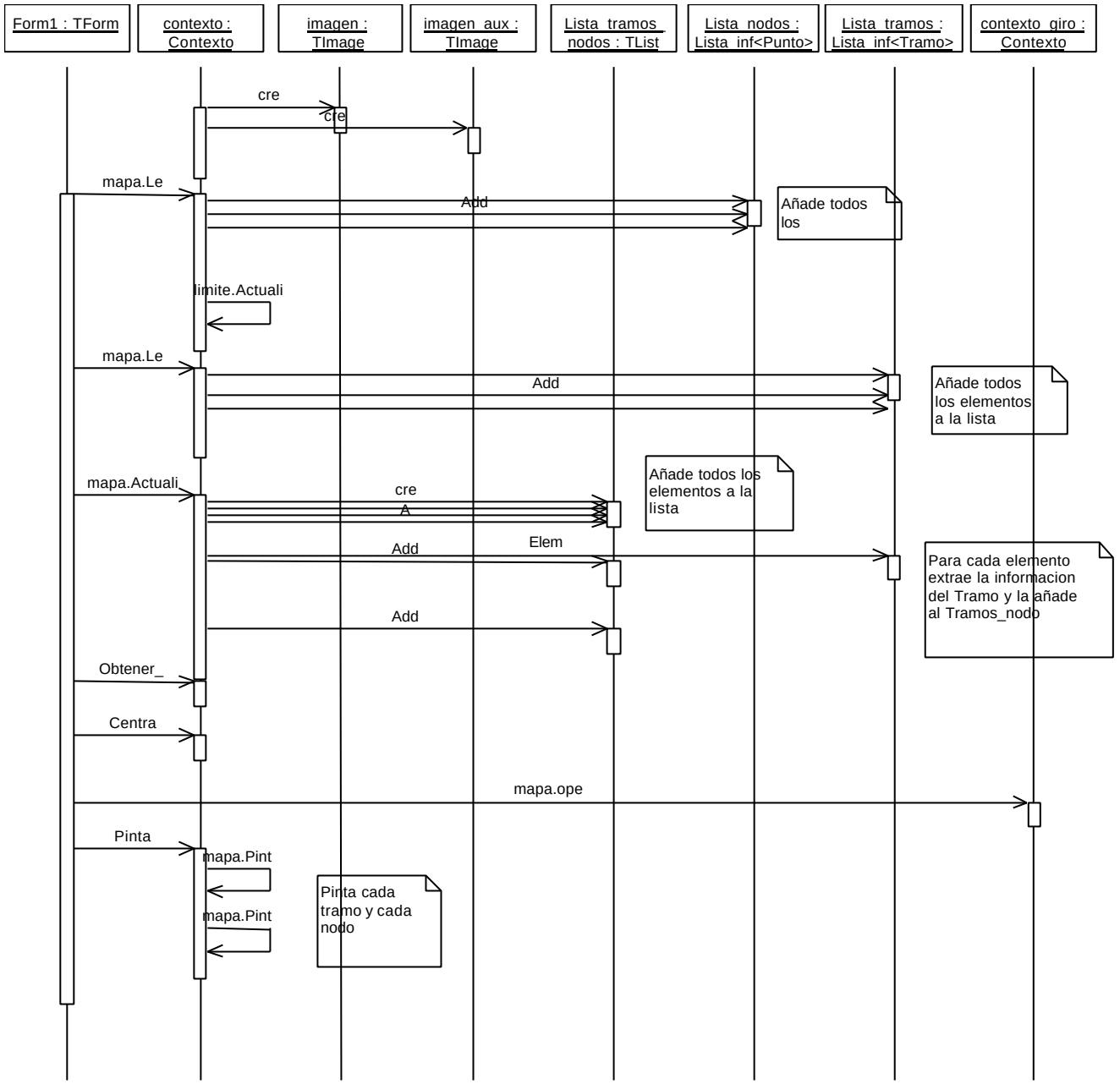

Figura 3.1.11: Diagrama de secuencia

En este diagrama se aprecian, dando especial importancia al aspecto cronológico, las interacciones entre objetos para lograr la implementación del caso de uso correspondiente.

Diagrama de actividades

Se usan para modelar los aspectos dinámicos del sistema. Los diagramas de interacción destacaban el flujo de control entre objetos, mientras que los diagramas de actividades van a destacar los flujos de control entre actividades.

Es similar a los conocidos diagramas de Pert: el diagrama de actividades muestra el flujo de actividades. Una actividad es una ejecución no atómica en curso dentro de una máquina de estados.

Los diagramas de actividades suelen contener estados de actividad y de acción, transiciones y objetos.

En el flujo de control modelado por este tipo de diagramas suceden cosas: evaluación de expresiones, invocación de operaciones sobre objetos, o creación y destrucción de los mismos. Estas acciones se llaman estados de acción y son atómicas. Las acciones de estado se representan como rectángulos con los bordes redondeados. No se pueden descomponer puesto que son atómicos y en teoría ocurren en tiempo cero, es decir, una cantidad de tiempo insignificante.

Los estados de actividad pueden descomponerse algo más. Estos estados de actividad no son atómicos, y pueden tardar algún tiempo en llevarse a cabo. Un estado de actividad puede descomponerse en otros estados de actividad y estados de acción.

Ambos estados, de actividad y de acción, son tipos especiales de estados de una máquina de estados.

Al completarse la acción o actividad de un estado, el flujo de control pasa a la siguiente actividad o acción mediante una transición. En UML se representa como una línea dirigida.

Pueden usarse bifurcaciones. Una bifurcación tendrá una transición de entrada y dos o más de salida. Cada transición de salida tendrá asignada una expresión booleana (verdadero/falso) de manera que el flujo de control continúa por la transión verdadera. Es por esto que las transiciones no pueden solaparse, es decir, no puede haber más de una que sea cierta, y siempre debe haber al menos una cierta. La bifurcación se representa por un rombo. Mediante el uso de bifurcaciones es fácil implementar iteraciones.

Una división representa la separación de un flujo de control simple en dos o más concurrentes. Una unión representa la sincronización de dos flujos de control concurrentes. El primer flujo que llega debe esperar a todos los demás. La división tendrá una transición de entrada y varias de salida, mientras que la unión tendrá varias transiciones de entrada y una sola de salida. Tanto división como unión se representan como una línea vertical perpendicular a las transiciones entrantes y salientes.

En este tipo de diagramas también existen calles. Las calles sirven para agrupar varios estados de actividad del diagrama. El motivo puede ser, por ejemplo, que cada calle sea implementada por un conjunto concreto de clases. Las calles dividen el diagrama de actividades en franjas verticales, y cada actividad pertenece a una única calle, aunque las transiciones pueden cruzar de una calle a otra.

Además en un flujo de control pueden verse involucrados objetos, que serán creados o modificados por las actividades. Estos objetos se añaden al diagrama y se unen a las actividades mediante relaciones de dependencia. Se pueden representar los valores de los atributos del objeto, así como su estado (entre corchetes []) en la misma caja del objeto.

Diagrama de estado

Se usan para el modelado dinámico del sistema. Un diagrama de estados muestra una máquina de estados, destacando el flujo de control entre estados. Los diagramas de estado suelen contener estados simples y compuestos y transiciones (eventos y acciones). Los diagramas de estado pueden contener bifurcaciones, divisiones, uniones, estados de acción, estados de actividad, objetos, estados iniciales y finales, etcétera.

Diagrama de componentes

Se usan para implementar la vista de implementación estática del sistema. Los diagramas de componentes son principalmente diagramas de clases que se centran en los componentes de un sistema.

Normalmente los diagramas de componentes suelen contener componentes, interfaces y relaciones de dependencia, generalización, asociación y realización. También pueden contener paquetes y subsistemas.

Se pueden usar de cuatro formas: para modelar código fuente, para modelar versiones ejecutables, para modelar bases de datos físicas y para modelar sistemas adaptables.

Diagramas de despliegue

Muestra la configuración de los nodos que participan en la ejecución y de los componentes que residen en ellos. Modelan la vista de despliegue estática del sistema, y son fundamentalmente diagramas de clases que se ocupan de modelar los nodos de un sistema.

Los diagramas de despliegue suelen contener nodos y relaciones de dependencia y asociación. También pueden contener paquetes o subsistemas y componentes.

En software que reside en una máquina y sólo interacciona con los dispositivos estándar de esa máquina (gestionados por el sistema operativo) se puede ignorar el diagrama de despliegue.

Los diagramas de despliegue se suelen usar de una de las tres maneras siguientes: modelado de sistemas empotrados, sistemas cliente/servidor y sistemas completamente distribuidos.