PROGRAMACIÓN CONCURRENTE Y DISTRIBUIDA

Seminario

Lenguaje de Modelado Unificado (UML)

J.M. Drake
Laura Barros

Notas:

UML es:

Un conjunto de elementos de modelado que **definen la estructura y funcionalidad del sistema** y que se agrupan en una base de datos única .

La presentación de esos conceptos a través de **múltiples vistas gráficas** con el fin de introducirlos, editarlos, y hacerlos comprensibles.

La idea general que podemos mantener:

Es una notación gráfica para dibujar diagramas de conceptos de software.

Representación del programa por el código

```
package codemodel;
public class Guitarist extends Person implements MusicPlayer {
      Guitar favoriteGuitar;
       public Guitarist (String name) {super(name);}
                // A couple of local methods for accessing the class's properties
      public void setInstrument(Instrument instrument) {
                if (instrument instanceof Guitar) {
                                this.favoriteGuitar = (Guitar) instrument;
                }else {
                                System.out.println("I'm not playing that thing!");
                 }
      }
      public Instrument getInstrument( ) {return this.favoriteGuitar;}
}
```

- Representa sólo la lógica e ignora el resto.
- El ser humano los interpreta muy lentamente.
- No facilita la reutilización del diseño.

Notas:

El código es un ejemplo extremo de lenguaje para describir un programa. En él no se ha realizado ninguna abstracción. Cada línea de código contiene los detalles de como el programa debe trabajar.

Problemas que tiene el uso del código como medio de descripción son:

•El código ha sido escrito para ser interpretado por el compilador, y en consecuencia contiene muchos detalles que sólo tienen interés para él y que hacen compleja su interpretación por un diseñador humano. La tendencia actual es a alejarse del código máquina.

•El código describe únicamente en el software en sí, e ignora el resto del sistema. Aún pensando que el código es una definición completa y no ambigua de lo que el software hace, no nos indica el como se utiliza y quien debe utilizarlo. Con el código se pierde la visión completa del software.

•El código no es un medio adecuado para intercambiar ideas entre programadores y diseñadores humanos, los cuales deben de inventar otros recursos para comunicarse.

•El código debe interpretándose leyendo un texto, y esto es muy lento para las personas.

•Si lo que se está haciendo es diseñar el sistema, su representación como código dificulta su posibilidad de reutilización en otros sistemas, posiblemente desarrollados en otros lenguajes.

Representación mediante un lenguaje natural

Guitarist is a class that contains six members: one static and five non-static. Guitarist uses, and so needs an instance of, Guitar; however, since this might be shared with other classes in its package, the Guitar instance variable, called favoriteGuitar, is declared as default.

Five of the members within Guitarist are methods. Four are not static. One of these methods is a constructor that takes one argument, and instances of String are called name, which removes the default constructor.

Three regular methods are then provided. The first is called setInstrument, and it takes one parameter, an instance of Instrument called instrument, and has no return type. The second is called getInstrument and it has no parameters, but its return type is Instrument. The final method is called play. The play method is actually enforced by the MusicPlayer interface that the Guitarist class implements. The play method takes no parameters, and its return type is void.

- Es ambigua y confusa
- Es lenta de interpretar
- Difícilmente puede ser procesada.

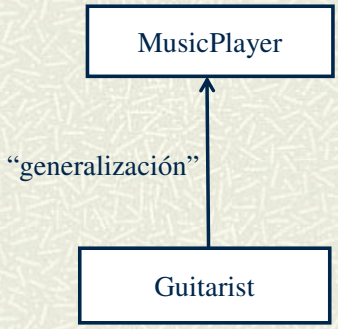
Notas:

En el extremo opuesto al código se encuentra la descripción del software mediante lenguaje natural. Este es informal y no sigue una notación bien definida y concensuada. Los posteriores lectores de esta descripción pueden entender muy diferentes interpretaciones de lo que representan.

También es textual, por lo que su interpretación por un humano es muy lenta y tediosa.

Representación gráfica

- Diagrama conceptual : “Un guitarrista es un músico”.
- No nos define el código fuente.

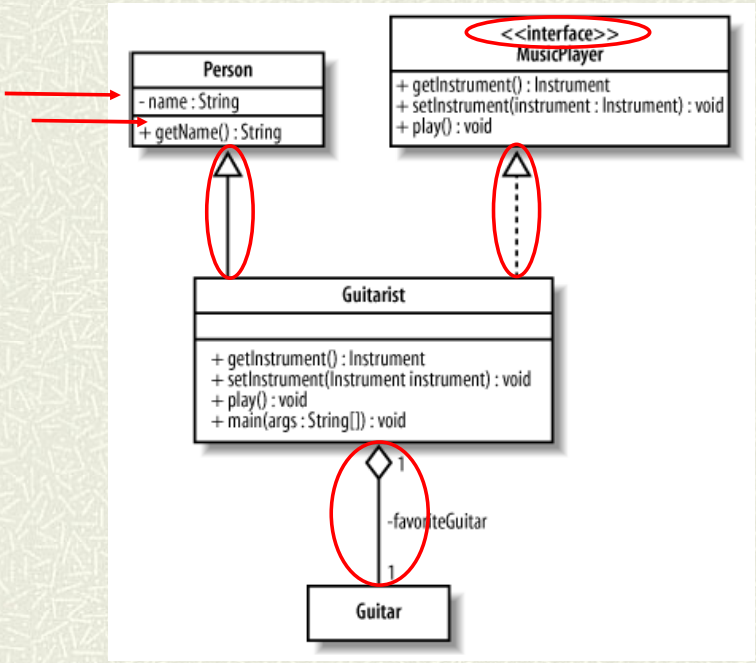



```
public interface MusicPlayer { }
public class Guitarist implements MusicPlayer { }
```


Notas:

El diagrama conceptual no nos define el código fuente, pero tampoco debe hacerlo, porque queremos abstraer, simplificar. Nos acercamos a lo que estamos buscando.

La representación mediante un modelo



Notas:

En la figura se muestra una vista de la descripción UML de una sección de código. Si conoce el lenguaje UML y la semántica de sus símbolos, una simple ojeada del grafico da una visión general de lo que describe, y además proporciona las vías por las que acceder a una información mas detallada de cualquier elemento (incluido sus códigos).

A resaltar de la representación:

1-Distintos tipos de relaciones. Asociaciones: generalización (herencia, especialización) o agregación (simple, composición).

2-Estereotipos. Añaden semántica.

3-Atributos y operaciones de las clases.

Lenguaje de Modelado Unificado.

- El **Lenguaje de Modelado Unificado (UML)** es un lenguaje estándar de tercera generación basado en la **metodología orientada a objetos**.
- El **objetivo** de UML es definir un modelo de cualquier tipo de sistema y mas específicamente de una aplicación software.
- Un modelo es un **conjunto integrado y coherente de abstracciones** que representa al sistema que se modela.
- El modelo consiste en:
  - Un conjunto de elementos de modelado que **definen la estructura y funcionalidad del sistema** y que se agrupan en una base de datos única .
  - La presentación de esos conceptos a través de **múltiples vistas gráficas** con el fin de introducirlos, editarlos, y hacerlos comprensibles.

Notas:

El Lenguaje de Modelado Unificado (UML) es un lenguaje basado en la metodología orientada a objetos creado por el Object Management Group (OMG) con la intención de convertirse en un estándar para el modelado de aplicaciones software, tanto a efectos de servir de base de la ingeniería software, servir de método de intercambio de ideas entre los equipos que desarrollan software y como base a las herramientas que utilizan.

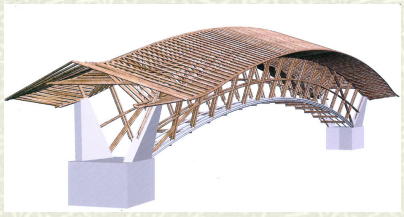
UML es un lenguaje de modelado universal con capacidad de describir cualquier tipo de sistema, sin embargo, su desarrollo semántico ha sido específicamente desarrollado para el modelado de aplicaciones software.

El objetivo de UML es permitir al diseñador formular un modelo del sistema y por modelo se entiende la definición de un conjunto de abstracciones semánticas que permiten describir simbólicamente los conceptos estructurales, de comportamiento y funcionales que se proponen para el sistema. Los elementos que constituyen el modelo se definen de forma integrada y coherente dentro de una única base de datos.

El modelo se presenta a través de múltiples vistas gráficas, que presentan parcialmente diferentes aspecto de la semántica del modelo. Las vistas son el recurso principal para introducir, editar y presentar el modelo. El conjunto de todas las vistas son parte o la descripción semántica completa del sistema.

Modelos en el mundo real

- Los ingenieros construyen modelos:
  - Son más baratos que los originales .Es más barato desechar un modelo.
  - Para ver si funciona.



Notas:

Ventajas del UML

- Está basado en **metamodelo** con una semántica bien definida.
- Se basa en una **notación gráfica** concisa y fácil de aprender y utilizar.
- Es un **estándar** soportado y mantenido por la OMG (Object Management Group).
- Es **escalable** y permite representar y gestionar tanto sistemas masivos como pequeños sistemas.
- Es un lenguaje concebido y evolucionado como **herramienta de desarrollo** de aplicaciones software.

Notas:

- UML está basado en un modelo bien definido semánticamente que se denomina "UML Metamodel". El modelo semántico es a la vez "amplio" (cubre la mayoría de los aspectos necesario para la especificación de sistemas software) y "profundo" (contiene la suficiente información para generar prototipo ejecutables destinados a su validación, o para generar esqueletos de código fuente).
- Es un lenguaje basado en una notación gráfica fácil de usar y comprender basado en un conjunto reducido de tipos de diagramas (diagramas de clases, diagramas de despliegue, diagramas de estados, diagramas de actividad, diagramas de secuencias y diagramas de casos de uso) estandarizados y que además se basan en un conjunto de principios comunes.
- Es un lenguaje gestionado en su creación y mantenimiento por el grupo internacional OMG (Object Management Group) que es independiente de cualquier fabricante con interese comercial, lo cual garantiza su permanencia y estabilidad, así como la existencia de múltiples fuentes y diferentes tipos de herramientas. La primera versión UML 1.1 fue propuesta a finales de 1997, y hace unos meses ha aparecido la versión UML 2.0. El curso se basará en la versiones 1.2 y 1.4. que son las soportadas por las herramientas.
- Es un lenguaje de tercera generación que ha heredado la experiencia de dos generaciones de herramientas previas y de cientos de especialistas que las han utilizado. Actualmente es un lenguaje refinado para que pueda ser eficientemente aplicado en el desarrollo de aplicaciones software, tanto si son proyectos pequeños llevados por una o dos personas o si son grandes proyectos desarrollados por cientos de personas.

Ventajas de UML II

- Es conveniente para comunicar conceptos de diseño entre los desarrolladores del equipo.
- Para explicar, por ejemplo, logaritmos mediante diagramas de secuencias o máquinas de estados mediante diagramas de estados.
- Para hacer guías detalladas (muchas clases dependiendo unas de otras).

Notas:

Aspectos semánticos del UML

- **La semántica de un sistema se define mediante:**
  - **Elementos estructurales**: Describen los elementos de que se compone el modelo, así como los parámetros que los cuanti-fican y de las relaciones entre ellos.
  - **Descripción de los comportamientos**: Describen los com-portamientos de los elementos de por sí y las interaciones entre grupos de elementos.
  - **Descripción funcional del sistema**: Describe la especifica-ción funcional del sistema con independencia de su imple-mentación.

Notas:

- Los elementos estructurales describen las "cosas" que constituyen el modelo, ya sean en fase de diseño como son las clases o tipos como en fase de ejecución como son los objetos o instancias. Así mismo, estos elementos se describen con diferentes niveles de abstracción, tanto a pequeño nivel próximo al código como son las clases, interfaces u objetos, como a gran nivel próximo a la organización arquitectural como son los subsistemas y los componentes. Así mismo, las relaciones entre las clases que describen sus interdependencias son también parte de la estructura del modelo.
- Los aspectos de comportamiento incluidos en el modelo definen como se comportan o interactúan entre si los diferentes elementos formulados en la estructura. Para los elementos estructurales individuales (clases, objetos, subsistemas, componentes, casos de uso) UML permite describir su comportamientos a través de diagramas de estados o diagramas de actividad que describen la secuencias de estado que siguen en función de las interacciones que reciben, mientras que para las agrupaciones de elementos estructurales, UML permite describir los posibles patrones de interacción a través de diagramas de colaboración o diagrama de secuencias.
- Los aspectos funcionales del sistema se refieren a la especificación funcional y no funcional (Q&S) del sistema que se desarrolla con independencia de la implementación que se proponga o realice. En UML la descripción funcional se realiza mediante diagrama de casos de uso.

Vistas del modelo UML.

- Un modelo UML de una aplicación puede tener muy **diferentes niveles de detalle**:
  - **Modelo arquitecturales**: con información cualitativa de alto nivel.
  - **Modelos detallados**: con información que permiten la validación o la generación de código.
- No se requiere una vista que contenga la **semántica completa** de la aplicación. La semántica reside en la base de datos.
- Las **múltiples vistas** permiten enfocar la atención en aspectos concretos con el adecuado nivel de detalle.
- La gestión de un modelo UML requiere una **herramienta específica** que mantenga la consistencia del modelo.

Notas:

Con UML el diseñador crea un modelo de aplicación que describe de forma completa, consistente y precisa el sistema que se desarrolla. La información contenida en el modelo puede permitir análisis y simulaciones del sistema a través del modelo, o así mismo ser la base para la generación automática de esqueletos o módulos completos de código fuente destinado a la compilación.

Cada vista del modelo describe algún aspecto particular de la semántica del sistema que se muestra a efecto de presentar un cierto nivel de abstracción.

La semántica reside en la información contenida en la base de datos en que se almacenan los elementos de modelado, y esta no reside en una vista singular en particular. En un modelo UML bien presentado, el conjunto de las vistas describe la semántica de la aplicación.

La multiplicidad de las vistas es de gran utilidad ya que permiten enfocar la atención del usuario a aspectos concretos que requieren una atención especial, con una cantidad de detalles adecuados al estudio que se realiza.

Para gestionar un modelo UML se requiere una herramienta específica, ya que no solo debe permitir dibujar las vistas, sino mantener la coherencia entre los componente que se incluyen en las diferentes vistas.

Diagramas UML

- Estáticas: Recurso de organización
  - **Diagramas de clases**
  - **Diagramas de componentes**
  - **Casos de usos**
- Dinámicas: Recurso de mostrar el comportamiento dinámico:

  Describen escenarios (casos particulares)
  - **Diagramas de secuencias**
  - **Diagramas de colaboración.**

  Describen la dinámica completa:
  - **Diagramas de estados (modelo de eventos)**
  - **Diagramas de actividad (modelo operativo)**

Notas:

UML ofrece muchos recursos para expresar la funcionalidad de los elementos estructurales. Estos recursos pueden utilizarse bien para especificar la funcionalidad, o bien para describir detalladamente esa funcionalidad. En el primer caso se puede considerar como una herramienta de ayuda al análisis o diseño, en el segundo caso, puede llegar incluso a constituir un lenguaje de programación, del que se puede generar automáticamente el código.

Elementos para la descripción de elementos estructurales y su funcionalidad:

•**Diagrama de estado**: sirve para describir un elemento estructural a través de una máquina de estados finitos.

•**Diagrama de actividad**: es un diagrama de estado con una semántica específica formulado como forma de expresar algoritmos complejos a través de diagramas de flujo.

Elementos para la descripción de mecanismos de operación de interacción entre grupos de objetos del sistema:

•**Diagrama de secuencias**: Permite representar a través de diagramas de transferencia de mensajes entre objetos organizados en el tiempo cualquier escenario de intercolaboración entre ellos.

•**Diagrama de colaboración**:Permite representar a través de diagramas gráficos los escenarios de colaboración que se establecen entre grupos de objetos.

Elementos de modelado estructural de bajo nivel

- Describen los elementos básicos que constituyen el sistema:
  - **Objeto**: Es un concepto de ejecución y es una instancia de datos junto con la descripción de las operaciones que actúan sobre ellos.
    - Los datos del objeto residen en "**atributos**".
    - Las operaciones que actúan sobre los datos son los "**métodos**"
  - **Clase**: Es un concepto de diseño y describe las características de un tipo de objetos, de los que se pueden crear múltiples instancias.
  - **Interfaz**: Representa un conjunto de operaciones que en conjunto representa un servicio. Es un elemento de diseño principalmente orientado hacia la reusabilidad.

Notas:

Los objetos, las clases y la interfaces son elementos de modelado estructural básico, destinados a representar e incorporar la información que corresponden con los conceptos de bajo nivel que constituyen la aplicación.

Un objeto modela una estructura de datos que se instancia durante la fase de ejecución de la aplicación. El objeto modela tanto la información que lleva asociada y que se describe mediante atributos y las operaciones que pueden operar sobre ella y que se describe mediante métodos. Estos métodos son invocados por otros objetos clientes o por otros métodos del mismo objeto.

Una clase describe durante la fase de diseño a un conjunto de objetos que instancian las mismas estructuras de datos y admiten los mismos tipos de operaciones. Durante la fase de ejecución pueden instanciarse muchos objetos que correspondan a la misma clase, mientras que un objeto es siempre la instancia de una única clase.

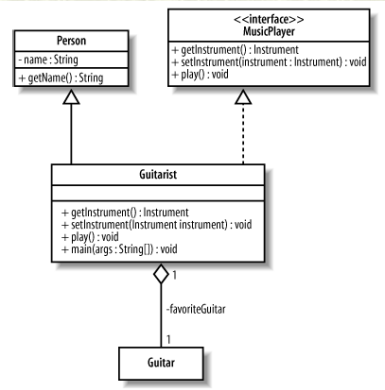
Las clases se pueden definir en función de otras clases definidas previamente, bien por especialización, extensión, agregación, asociación, etc. lo que reduce considerablemente la complejidad de la definición del gran número de instancias que requiere una aplicación.

Las interfaces permiten nombrar conjuntos de operaciones que representan un servicio completo y que pueden ser ofertados por diferentes clases que los ofertan. Su función es independizar el servicio de la implementación. Dos clases que realicen una misma interface, ofrecen el mismo servicio, lo que significa que si una clase requiere un servicio, lo puede obtener de cualquier clase que realice la misma interfaz.

Una interfaz no es directamente instanciables. Las interfaces se realizan a través de clases. Para ello la clase debe implementar un método concreto, por cada operación de la interfaz.

La clase

- Se utiliza para modelar de forma común a grupos de objetos que tienen:
  - Las mismas propiedades (Atributos)
  - El mismo comportamiento (Operaciones)
  - Las mismas relaciones con otros objetos (Asociaciones)
  - Semántica común (Estereotipo)



Notas:

Representa un grupo de objetos que tienen similares propiedades (atributos) , comportamiento (operaciones) y formas semejantes de relacionarse con objetos de otras clases (asociaciones) y una semántica común (estereotipo). Se utilizan como mecanismo de describir de forma unificada todos aquellos objetos que aunque tienen diferente entidad corresponden a una misma descripción.

Todos los objetos tienen una clase, a veces por ser los únicos elementos de la clase la descripción de ambos se hace conjuntamente y sin marcar la diferencias.

En el análisis buscamos el modelado del problema por identificar los objetos que forman parte de él y agruparlos en clases que describen sus características y comportamiento. La clases y objetos están en el enunciado del problema y el análisis trata de descubrirlos, pero también hay una fase de invención ya que tienen que ser abstraídos y dotados de los mecanismos que instrumentan su comportamiento.

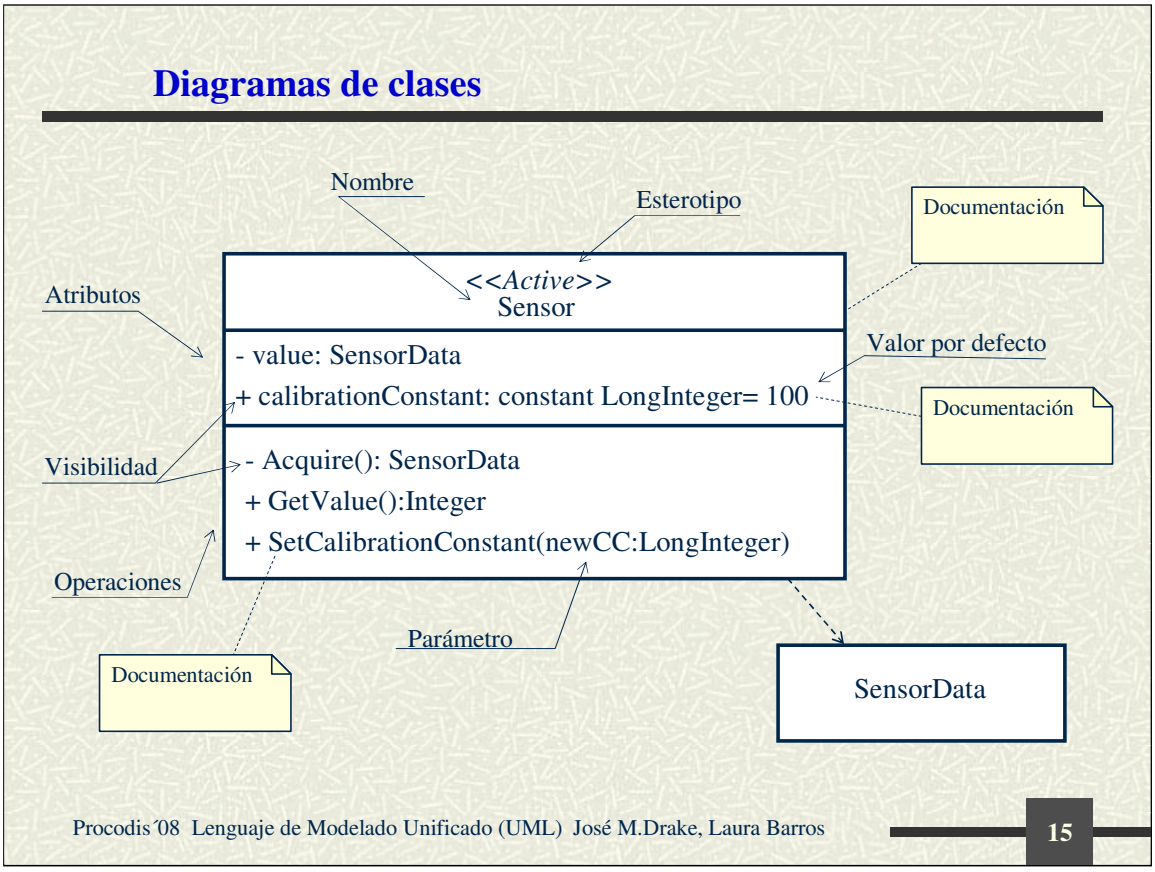
Las clases y objetos suelen aparecer del análisis de elementos como:

•Cosas tangibles: Coches, Facturas, etc.

•Personas: Que realizan o llevan a cabo alguna reponsabilidad o role (madre, profesor, etc.)

•Eventos: Aterrizaje, interrupción,requerimiento,etc.

•Interacciones: Carga, encuentro, intersección, etc.



Notas:

En UML una clase se representa mediante un rectángulo con tres compartimentos, el superior contiene el identificador y en su caso el estereotipo, el medio contiene los atributos, y el inferior, las operaciones o métodos que tienen asociados.

El estereotipo que puede ser asociado no solo a las clases, sino a cualquier elemento UML, le asocia una semántica especial.

Los atributos se definen mediante un identificador, y en su caso, su tipo un valor por defecto y su visibilidad. El tipo puede corresponder a algún tipo primitivo definido en el dominio o cualquier otra clase definida en el sistema. La visibilidad indica quien puede hacer referencia a ellos (privados, públicos, reservados, etc.) y es fuertemente dependiente del lenguaje que se utilice.

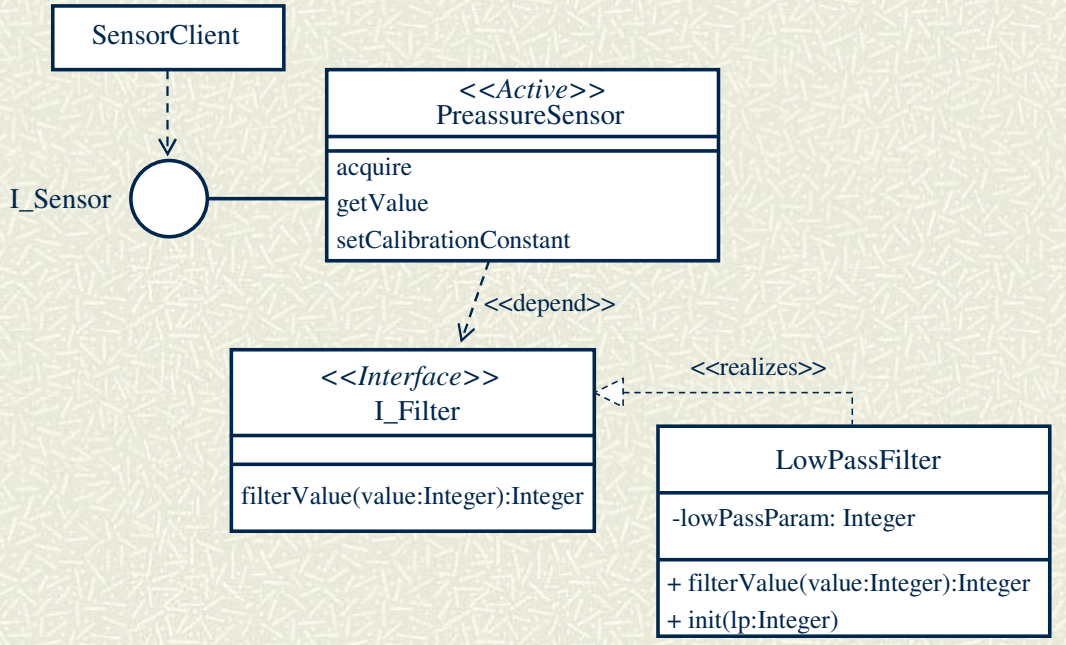
Las operaciones se definen mediante un identificador, un conjunto de parámetros con sus identificadores, tipos y valores por defecto, el tipo del valor de retorno, y la visibilidad.

Las clases pueden estar declaradas como abstracta, en cuyo caso no son directamente instanciables, y solo tienen la función de servir como base para la declaración de nuevas clases derivadas de ellas.

Las clases pueden ser de instanciadas, lo que representan que los atributos y los procedimientos están definidos en función de la clase y no de sus instancias. Los atributos y las operaciones también pueden declararse como de clase, y ser compartidos por todas las instancias.

Cualquier elemento de la clase tiene asociada un texto de documentación y a través de ellos se le pueden asociar nuevos parámetros introducidos por el diseñador.

Declaración de Interfaces



Notas:

Las interfaces son conjuntos de operaciones agrupadas bajo un identificador por corresponder a un servicio que pueden ser ofrecidos por múltiples clases. En UML se representan como una clase con el estereotipos <<interface>>.

En un diagrama de clases una interfaz se representa o bien con el símbolo de ordinario de clase o bien mediante un icono consistente en un círculo.

Una clase que realiza una interfaz puede representarse bien asociándole el icono de la interfaz, o bien representando la clase como derivada de la clase que representa la interfaz y asignando a la asociación el estereotipo <<realizes>>.

Cuando una clase requiere el servicio representado por la interfaz para implemetar su métodos, se relacionan mediante una asociación de dependencia con el estereotipo <<depend>>. En fase de ejecución la interfaz deberá ser sustituida por una implementación de una clase que realice esta interfaz.

Elementos de modelado estructural a alto nivel

- Describen los elementos de alto nivel que constituyen el sistema desde el punto de vista arquitectural:
  - **Paquete:** Elemento contenedor que permite organizar los elementos de modelado en bloques a efecto de su gestión.
  - **Subsistema:** Elemento estructural de alto nivel que sirve para agrupar conjuntos de elementos de modelado en función de que van a contribuir a un aspecto de comportamiento del sistema.
  - **Componente:** Elemento estructural de alto nivel que agrupa conjuntos de elementos de modelado en función de que que constituyen un bloque reemplazable.
  - **Nudo**: Modela un elemento o entorno físico del sistema en el que se puede ejecutar algún elemento software.

Notas:

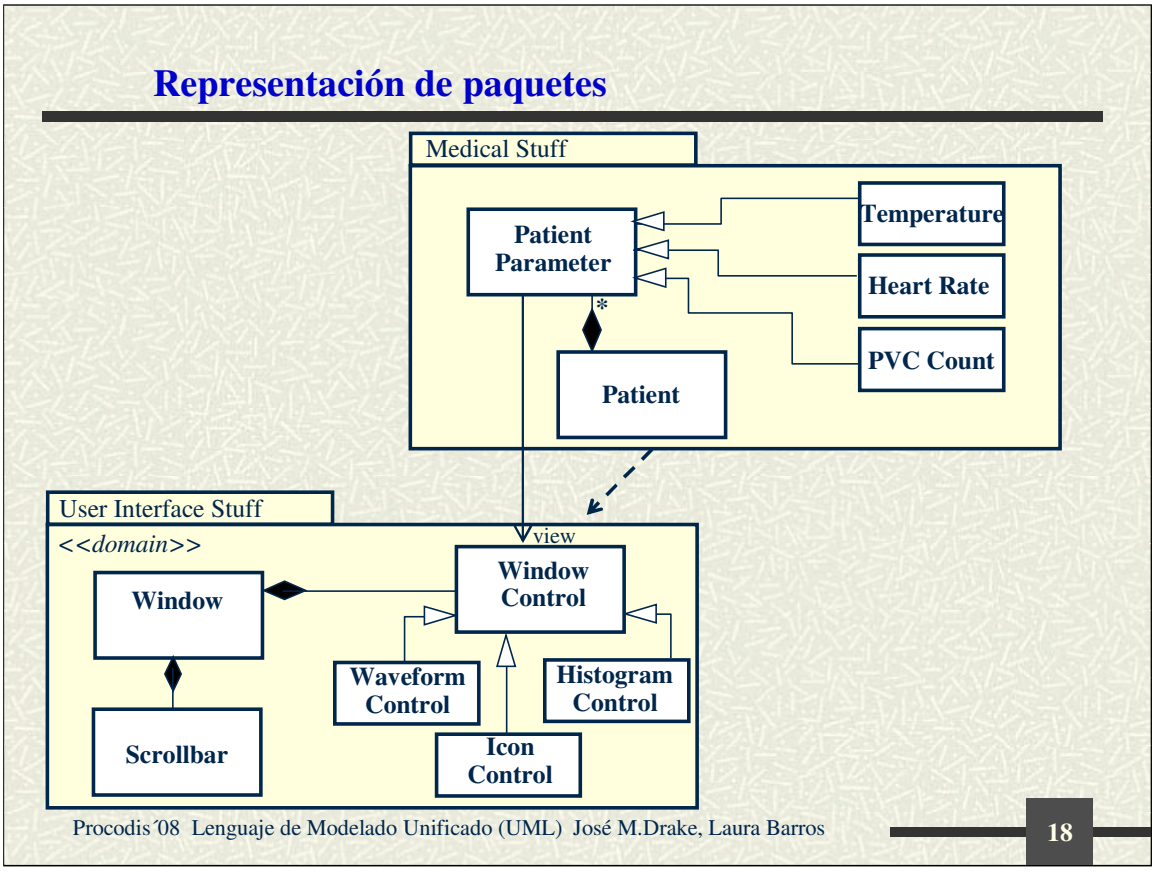
Dada la complejidad de los sistemas que actualmente se desarrolla, es habitual que el sistema deba dividirse en grandes módulos que facilite su gestión. UML ha definido la semántica de ciertos elementos de modelado que facilitan la gestión de estas grandes unidades.

Los paquetes son elementos de modelado que tienen como función servir de contenedores de otros elementos de modelado a efectos de su organización.Los paquetes se utilizan para dividir los modelos en fragmentos a fin de que puedan ser gestionados mas eficientemente o procesados por personas o equipos diferentes. Los paquetes solo contienen elementos que existen enla fase de diseño y nunca corresponden a algo que sea instanciables. Sirven para organizar modelos o para proporcionar un dominio de denominación.

Los subsistemas son utilizados para organizar módulos de elementos que van a existir en la fase de ejecución del sistema y que se agrupan en función de que colaboran para proporcionar una determinada funcionalidad. Los sistemas son los elementos de mayor nivel de abstracción destinados a describir la arquitectura física de un sistema. Los subsistemas contienen información sobre las operaciones, servicios y especificaciones de QoS e implementación.

Los componentes son elementos de organización de módulos de un sistema, semejantes a los subsistemas, pero que por corresponder a servicios completos y bien definidos se caracterizan por constituir unidades reemplazables.

Los nudos son elementos de modelado de unidades hardware (procesadores, redes de comunicación, equipos, etc.) que constituyen la plataforma en la que se va a ejecutar el sistema que se desarrolla.



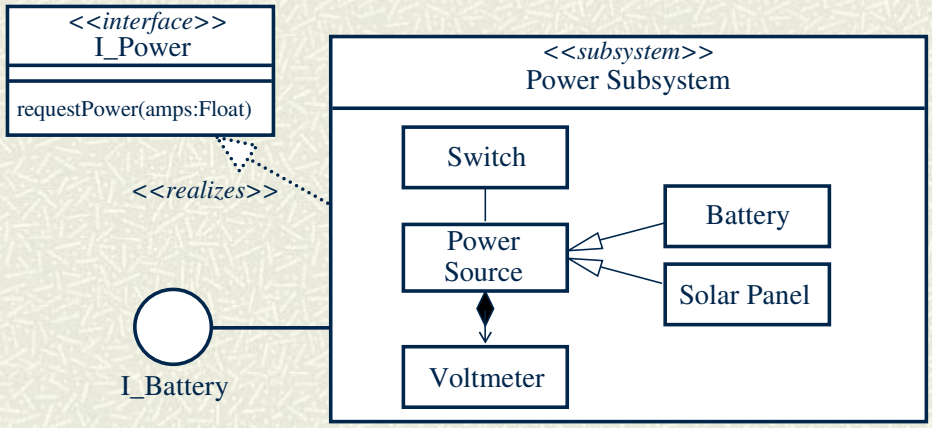
Notas:

En UML los paquetes se representan por un icono en forma de carpeta. En ella se incorporan los elementos de modelado que contienen.

Las carpetas como todo elemento UML pueden tener asignado un estereotipo. En el ejemplo la carpeta User Interface Stuff se le asignado el estereotipo <<domain>>, lo que significa que utilizamos el paquete como forma de agrupar elementos que corresponden a un mismo dominio del problema.

Las carpetas UML tienen la misma semántica que las carpetas de los sistemas de ficheros, y son elementos contenedores que pueden agrupar cualquier tipo de elemento de modelado o de diagrama de visualización. Aunque los paquetes pueden representarse en los diagramas de clases a efectos de organizar los elementos de modelado estructural, su contenido real debe ser observado a través de los navegadores del modelo, que se organizan en función de los paquetes que se definen.

Subsistemas

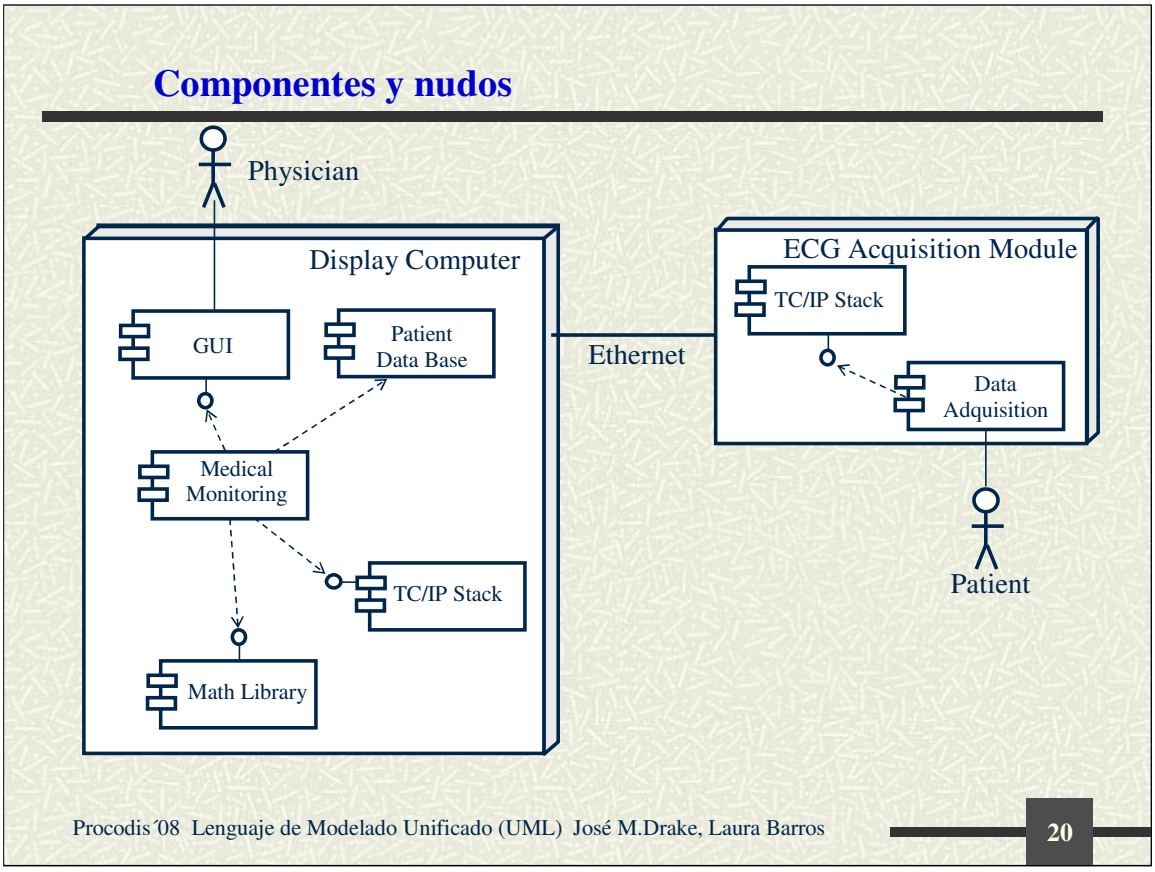


Notas:

Los subsistemas se representan mediante iconos de clases con el estereotipo <<subsystem>>.

Los elementos que incluye un subsistema pueden incorporarse mediante superposición (como en la figura) o a través del campo de atributos y haciendo referencia a clases definidas en el modelo, mediante asociaciones (habitualmente de tipo composición) con otras clases definidas en la vista.

La funcionalidad que ofrece el subsistema puede modelarse a través del campo de operaciones que ofrece la clase, o también expresando que el subsistema realiza interfaces que son descritas externamente al subsistema.



Notas:

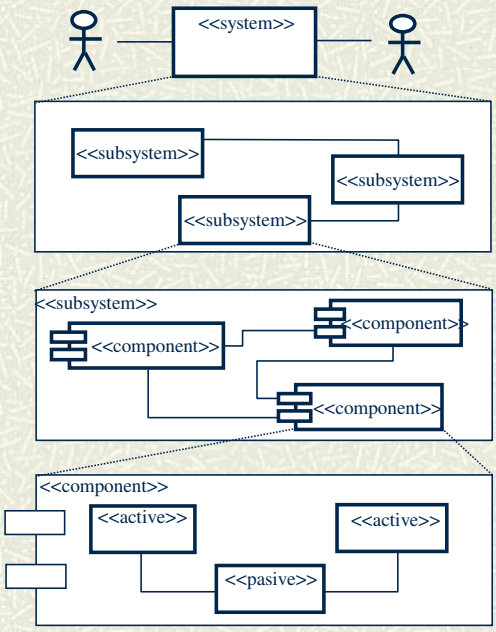
Los componentes se representan en UML a través de clases con el estereotipo <<component>> o a través de un icono específico (ver figura).

El nudo se representa mediante un icono de tres dimensiones.

En la figura se muestran también actores, que son elementos físicos que interaccionan con el sistema que se desarrolla pero que están fuera del ámbito de diseño.

A menudo los componentes, subsistemas y nudos se representan conjuntamente en diagramas de despliegue

Jerarquía de elementos estructurales



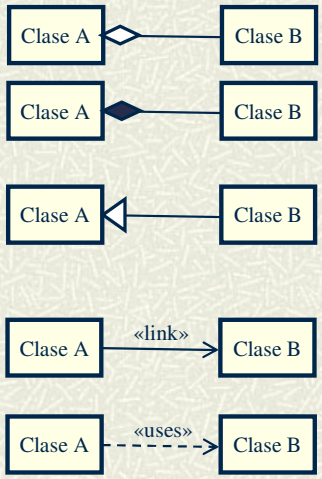
Notas:

Aunque los conceptos de sistema, subsistema, componentes tienen una semántica bien definida y suele estar claro su uso, en UML no está definida su jerarquía relativa, ni su granularidad.

En este curso seguiremos el criterio (no requerido en UML) de que los sistemas se componen de subsistemas (con los niveles de especificación que se necesiten), los subsistemas se componen de componentes (si el concepto de componente se considera necesario) y por último, unos y otros se describen en función de clases, interfaces y objetos.

Asociaciones

- Asociación: Está relacionado con...
  - **Agregación**: Es parte de ..
    - **Agregación simple**: Está contenido en ...
    - **Composición**: Se compone de ...
  - **Generalización**: Es un tipo de ...
    - **Herencia**: Extiende a ...
    - **Especialización**: Especializa a...
  - **Dependencia**
    - **Link:** Tiene acceso a …
    - **Abstraction**: (<<refine>>, <<realice>>,...)
    - **Bind**: (Entre clases genericas y concretas)
    - **Usage**: (Hace uso de ..., tiene visibilidad sobre...)
    - **Permission**: (<<friend>>, <<son>>,...)



Notas:

Las asociaciones son los elementos que ofrece UML para representan las relaciones que existen entre los elementos estructurales que se utilizan en los modelos. La asociación es un concepto de la fase de diseño y se establece entre clases, mientras que el enlace (“link”) es su instanciación para los modelos de la fase de ejecución.

En UML se definen muchas clases de asociaciones:

•Asociación : Representa que durante alguna fase de su existencia una clase requiere tener acceso o referencia a otra clase para poder satisfacer su funcionalidad.

•Agregación: Representa una clase que es parte de otra.

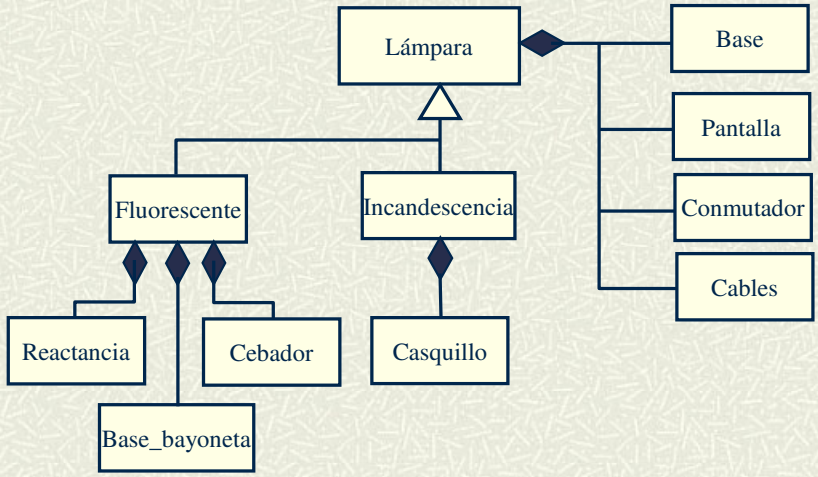
•Composición: Es un tipo de agregación fuerte que significa que una clase se compone de otras clases definidas. La diferencia entre agregación simple y composición esta en la exclusividad (una clase puede estar agregadas en varias otras clases) y de responsabilidad de creación y destrucción (si una clase está relacionada por composición con otra, ésta es responsable de que aquella se cree y se destruya con su creación y destrucción.

•Generalización: una clase es una generalización de otra si es un tipo de ella, esto es, donde pueda estar la clase mas abstracta puede estar la clase mas concreta. Existen dos formas básicas de generalización:

•Herencia: Una clase se crea a partir de otra, incorporando de la antecesora todo lo establecido en la definición de ella mas nuevos aspectos que se añaden en su declaración específica.

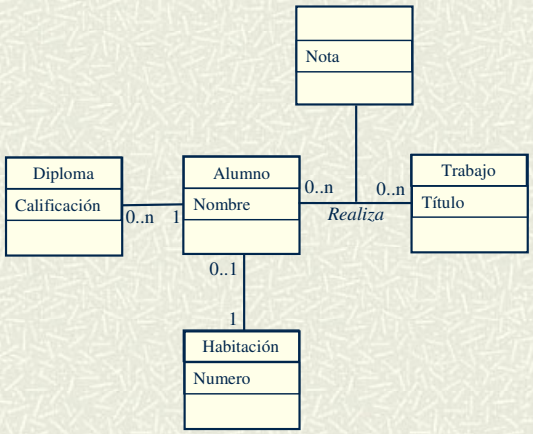
•Especialización: Una clase se crea a partir de otra, especializando su funcionalidad de acuerdo con nuevas necesidades.

Ejemplo de asociaciones



Notas:

Multiplicidad de las asociaciones



1	Una y solo una
0..1	Cero o una
0..n	Cero o mas
1..n	1 o mas
-	Inespecificado

Notas:

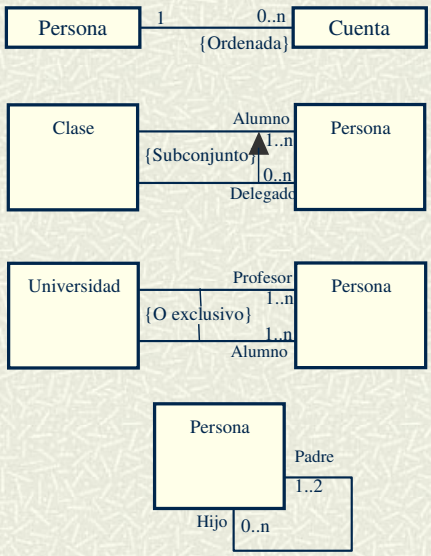
Cada función de una asociaciones lleva una indicación de multiplicidad que muestra cuantos objetos de la clase considerada pueden ser relacionados con un objeto de otra clase. La multiplicidad es una información consecuencia de la función, bajo una expresión entera acotada.

Un valor de multiplicidad mayor que 1 implica un conjunto de objetos y debe ser implementado con algún tipo de contenedor (array, lista, cola, etc.)

Los valores de multiplicidad implica restricciones relacionadas con el ámbito de la aplicación, válidas durante toda la existencia de los objetos. Las multiplicidades no deben considerarse durante los regímenes transitorios, como en la creación y destrucción de objetos.

La materialización de las asociaciones toma todo su interés para las asociaciones n a n. En las asociaciones 1 a 1, los atributos pueden desplazarse a cualquiera de los objetos. En las relaciones 1 a n pueden asociarse a la clase del lado n, sin embargo en la clase n a n el desplazamiento no es posible, y se requiere asociar una clase (contenedora del atributo) a la propia asociación.

Restricciones sobre las asociaciones



Notas:

Pueden definirse diferentes tipos de restricciones entre las asociaciones de una clase. Las restriciones se representan en los diagramas por expresiones encerradas entre llaves.

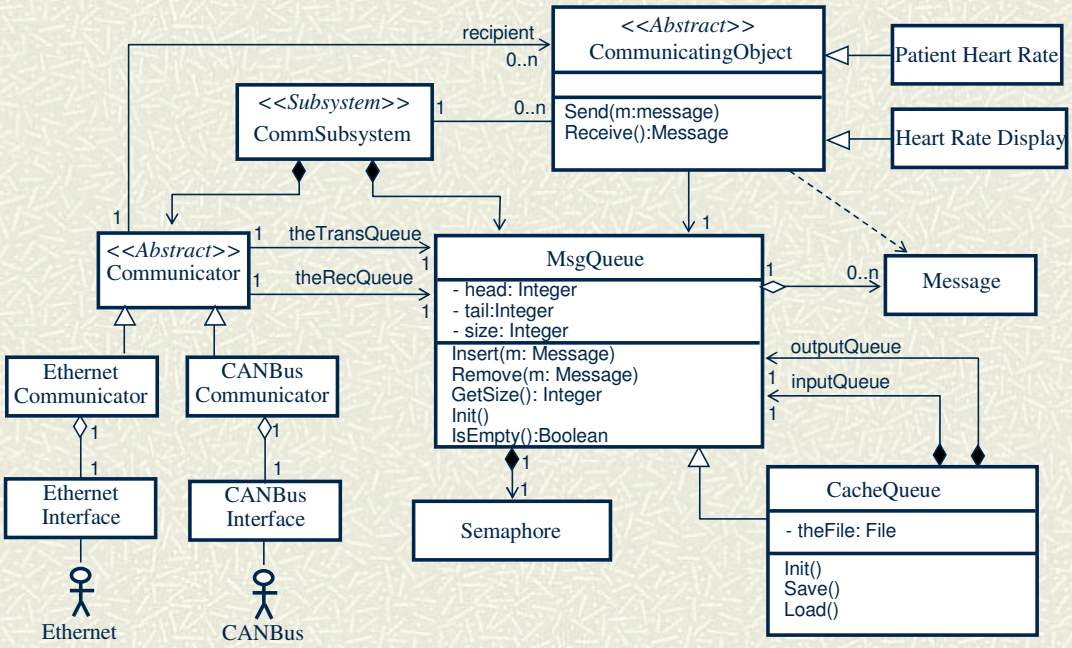
La restricción {ordenada} puede colocarse sobre una asociación para especificar una relación de orden entre los objetos enlazados por la asociación.

La restricción {subconjunto} indica que el conjunto de una asociación está incluido en el conjunto establecida por la otra asociación.

La restricción {O exclusiva}indica que para un objeto dado, es válida una sola asociación entre un grupo de asociaciones.

Las asociaciones pueden también enlazar una clase consigo misma. Este tipo se llama asociaciones reflexivas. El role de las asociaciones toma toda su importancia para distinguir las instancias que participan en la asociación.

Representación de asociaciones en diagramas de clases.



Notas:

Dependencia: Es un tipo de asociación muy débil en la que se indica que para definir o explicar una clase se debe tener en cuenta la definición de la otra. Ejemplos de dependencias son: <<abstraction>> una clase resulta como una concreción de otra mas abstracta; <<bind>> una clase resulta de concretar una clase genérica; <<usage>> una clase hace uso internamente de otra, y en consecuencia, requiere visibilidad sobre ella. <<permission>> una clase tiene derechos especiales de acceso a los recursos de otra.

Una asociación se representa en los diagramas de clases UML como una línea que enlaza las clases que se realciona.

La agregación se representa mediante un pequeño rombo en el extremo próximo a la clase contenedora. Si la agregación es simple, el rombo está vacío, y si es una composición el rombo esta relleno.

La herencia se representa mediante un pequeño triángulo vacío en el extremo de la clase mas general.

La dependencia se representa mediante una línea punteada.

Las asociaciones pueden estar orientadas o no. Una asociación orientada se expresa con una flecha y es solo utilizada por la clase origen para hacer uso de la clase destino. Si no no es orientada se representa sin flecha, y en ese caso es utilizable por ambas clases.

En los extremos de una asociación se puede incluir la cardinalidad, esto es el número de objetos de esa clase con los que se puede estar enlazado en fase de ejecución.

En los extremos de una asociación se pueden incluir un identificador o “role”.

Modelado del comportamientos

- Son recursos ofrecidos por UML para describir el comportamiento dinámico de los elementos estructurales (objetos, clases, sistemas, subsistemas y componentes) cuando el sistema se ejecuta.
  - **Diagramas de estados:** Se utiliza para describir la evolución interna o el efecto dinámico de una clase o de un método.
  - **Diagrama de actividad**: Es diagrama de flujo concebido para representar el efecto de un método.
  - **Diagrama de secuencias**: Describe la dinámica de interacción entre diferentes objetos, formulados como secuencias de eventos organizados en el tiempo.
  - **Diagrama de colaboración**: Descrie la dinámica de interacción entre diferentes objetos, formulados sobre una instanciación concreta de objetos del sistema.

Notas:

UML ofrece muchos recursos para expresar la funcionalidad de los elementos estructurales. Estos recursos pueden utilizarse bien para especificar la funcionalidad, o bien para describir detalladamente esa funcionalidad. En el primer caso se puede considerar como una herramienta de ayuda al análisis o diseño, en el segundo caso, puede llegar incluso a constituir un lenguaje de programación, del que se puede generar automáticamente el código.

Elementos para la descripción de elementos estructurales y su funcionalidad:

•Diagrama de estado: sirve para describir un elemento estructural a través de una máquina de estados finitos.

•Diagrama de actividad: es un diagrama de estado con una semántica específica formulado como forma de expresar algoritmos complejos a través de diagramas de flujo.

Elementos para la descripción de mecanismos de operación de interacción entre grupos de objetos del sistema:

•Diagrama de secuencias: Permite representar a través de diagramas de transferencia de mensajes entre objetos organizados en el tiempo cualquier escenario de intercolaboración entre ellos.

•Diagrama de colaboración:Permite representar a través de diagramas gráficos los escenarios de colaboración que se establecen entre grupos de objetos.

Acciones y actividades. .

- **Acción (Action)**: Abstracción de una primitiva ejecutable que da lugar a un cambio en el estado del sistema (generación de un evento, cambio del valor de un atributo o enlace).
  - Modela una estamento computacional simple que tiene la semántica "run to completion" (Se inicia para ser terminado)
  - Ejemplos de acciones:
    - Acción de creación (**CreateAction**) o eliminación de un objeto (**Destroy Action**).
    - Invocación síncrona de una operación o método (**CallAction**).
    - Retorno de control desde una operación (**ReturnAction**) o desde una secuencia de acciones (**TerminateAction**)
    - Transferencia asíncrona de un evento (**SendAction**)
    - Acción compuesta de Secuencia de acciones (**ActionSequence**)
- **Actividad (Activity)**: Es una secuencia de acciones que se ejecuta en un estado de un objeto y que finaliza cuando el objeto cambia de estado (por un evento externo, por temporización o por finalizarse la secuencia)
  - No tiene la semántica "run to completion"

Notas:

Una acción es una especificación de una primitiva ejecutable que tiene como consecuencia un cambio de estado en el modelo, y puede consistir en la transferencia de un mensaje, modificar un enlace o cambiar de valor un atributo.

Las acciones son primitivas computacionales simples que tienen la semántica de iniciarse para ser terminada ("run-to-completion"). Esto no significa que no pueda ser expulsada de su ejecución por otra de mayor prioridad, sino que aun en este caso posterior se retorna a ella para completarla. Un objeto que este ejecutando una acción, no acepta nuevos mensajes hasta que la concluya.

Clases de acciones que especifica UML 1.4 son: "CreateAction", "CallAction", "ReturnAction", "SendAction", "TerminateAction", "DestroyAction", "ActionSequence" y "UninterpretedAction"

Una actividad es una primitiva ejecutable que se ejecuta mientras que el elemento se encuentra en un determinado estado y que termina cuando finaliza o cuando el elemento cambia de estado. Una actividad no tiene la semántica "run-to-completion" y si un objeto está ejecutando una actividad y recibe un mensaje, puede concluirla y cambiar de estado.

Diagramas de estado

- Describe el comportamiento de un elemento estructural o de un método como una máquina de estados finitos, basada en la representación de los estados del elemento y de especificar las causas de transiciones entre ellos.
- El comportamiento se describe asociando acciones a:
  - Las transiciones entre estados
  - La entrada a los estados
  - La salida de los estados
- A la estancia en un estado se puede asociar una actividad.
- Los diagramas de estados de UML son muy potentes y escalables:
  - Permite agregar un diagrama de estado a un estado.
  - Admite concurrencia a través de estados AND.
  - Permite introducir semántica dinámica a través de pseudoestados.
  - Admite transiciones condicionales y sincronizadas.

Notas:

Un diagrama de estados describe una máquina de estados finita basada en un conjunto de estados en que se puede encontrar el sistema, el conjunto de transiciones entre estados disparados por eventos, y los conjuntos de acciones o actividades que se ejecutan en función de las transiciones que se realizan o de los estados en que se encuentra el sistema.

Los diagramas de estado UML admiten:

•Estados formulados como diagramas de estados.

•Máquinas de estados concurrentes sobre un mismo diagrama, y por tanto la definición de estados compuestos AND del elemento.

•Diferentes tipos de psudoestados para organizar la inicialización, acceso, salida o persistencia en las máquinas de estado.

•Admite transiciones condicionales en las que en función de condiciones de guarda las posibles transiciones se habilitan o deshabilitan.

•Admite pseudoestados destinados a la sincronización de eventos concurrente o a la generación simultánea de multiples eventos.

Permite utilizar eventos asíncronos que son implementados a través colas de eventos en el destino, esto es, eventos que el que los genera los olvida, pero que se mantienen encolados en el destinatario está que está en un estado en el que puede gestionarlo.

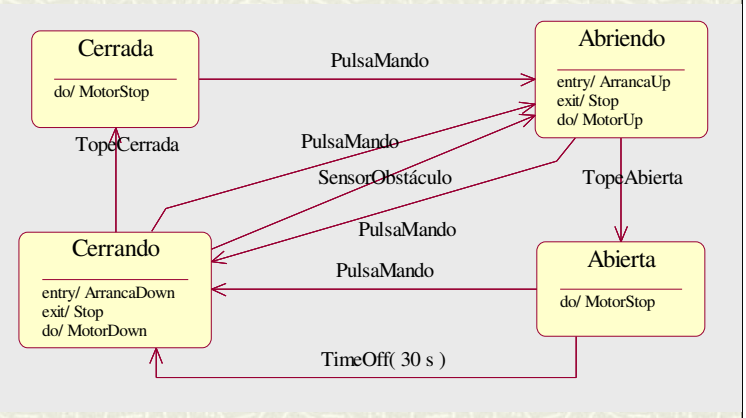
También admite eventos síncronos en el que se sigue la política de que el que los envía permanece bloqueado hasta que el evento es gestionado.

Ejemplo: Puerta de garaje

Eventos externos:
- PulsaMando
- TopeAbierto
- TopeCerrado
- SensorObstáculo

Acciones:
- ArrancaUP
- ArrancaDown
- Stop

Actividades:
- MotorUP
- MotorDown
- MotorStop



•Acción: no se interrumpe.

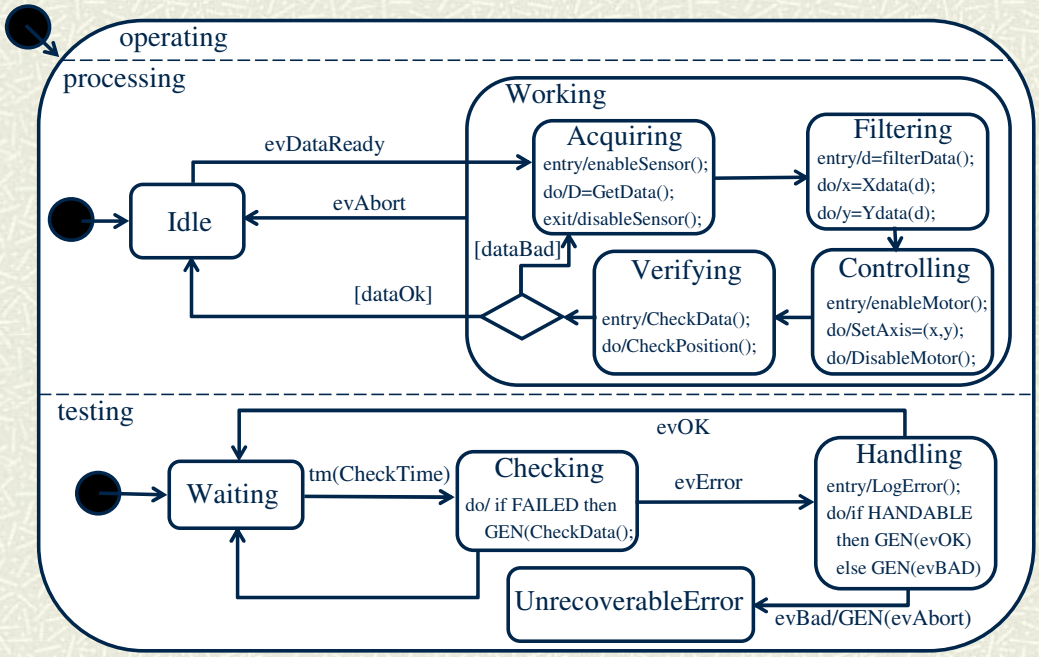
•Actividad: interrumpible.

Notas:

Acción: no se interrumpe.

Actividad: interrumpible.

Representación de los diagramas de estado.



Notas:

Un diagrama de estado es representado mediante óvalos que representan los estados y aristas que representan las transiciones.

A un estado se puede asociar las acciones que se ejecutan cuando se accede a él (entry/) o se sale de él (exit/) y la secuencia de actividades que se ejecutan mientras se está en él (do/)

A una transición se le pueden asociar condiciones de guarda ([ ]) que habilitan/deshabilitan su tránsito. Así mismo, a una transición se pueden asociar las acciones que se ejecutan cuando se transita por ella.

Dentro de un estado se pueden incluir una sección de diagrama estados, que indican subestados discernibles dentro de él.

Mediante líneas discontinuas se separan diferentes máquinas de estados finitos que concurren dentro de un mismo diagrama.

Existen definidos muchos pseudoestados para organizar el diagrama:

•Estado inicial: (circulo relleno) Punto de inicio de la actividad en el diagrama.

•Estado salida: (ojo de buey) Punto por el que se abandona la actividad.

•Condicional: (rombo) Se elige una transición de acuerdo con una condición.

•Historico:(Circulo con H) Se inicia la actividad por el estado en que se abandonó.

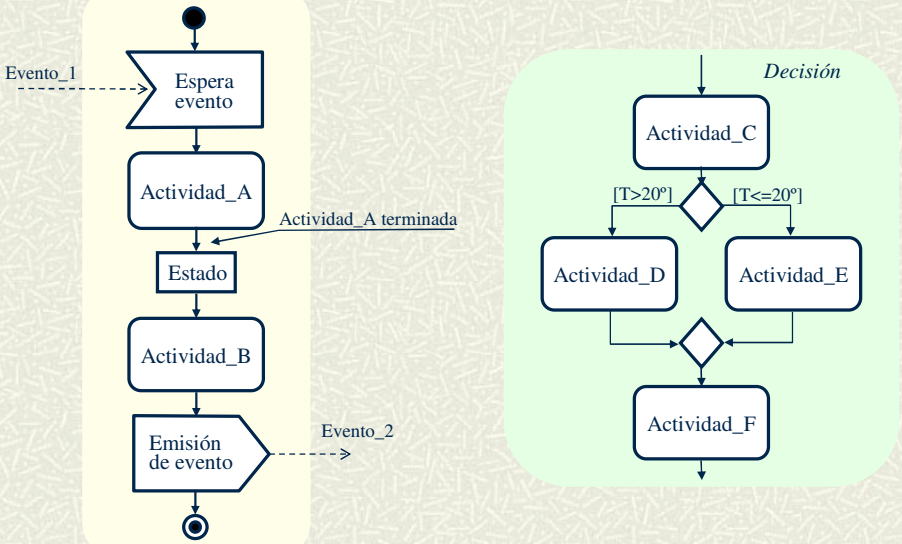
•

Diagrama de actividad

- Es un tipo especializado de diagrama de estados cuya función es implementar un diagrama de flujo para modelar el control de flujo que conlleva la ejecución de una operación.
- Modela flujos de control secuenciales y concurrentes.
- La transición básica entre estados es producida por conclusión de la actividad asociada al estado.
- El uso mas habitual de los diagramas de actividad es describir algoritmos complejos.

Notas:

Componentes de los diagramas de actividad.



Notas:

Actividad: Tarea que es realizada dentro de procedimiento o caso de uso que se describe.

Estado: Etiqueta que hace referencia a un estado de ejecución del proceso descrito por el diagrama.

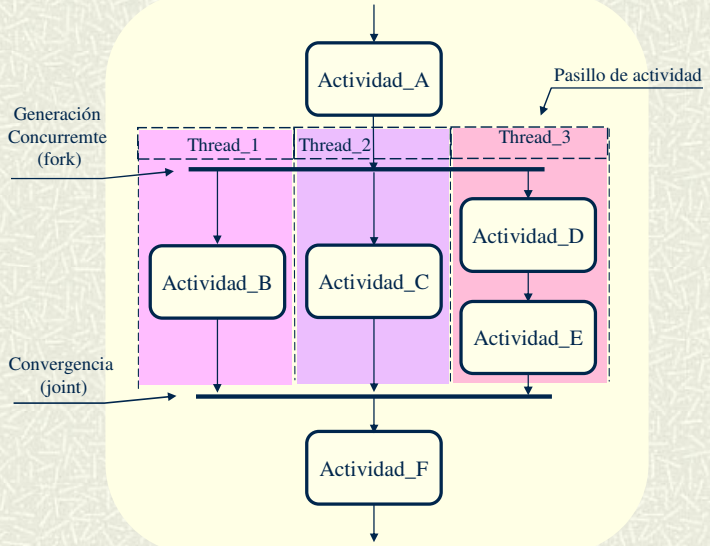
Espera de evento: El flujo de control se suspende hasta que se genera un evento o mensaje.

Emisión de un evento: Se genera un evento o mensaje que puede ser un resultado o una forma de interacción con otras secciones de la tarea.

Bifurcación condicionada (Branching): El flujo de control pasa a una u otra rama del diagrama, en función de que satisfaga o no una cierta condición.

Convergencia (Merge): Varia líneas alternativas se encauzan hacia una secuencia de actividades.

Representación de la concurrencia.



Notas:

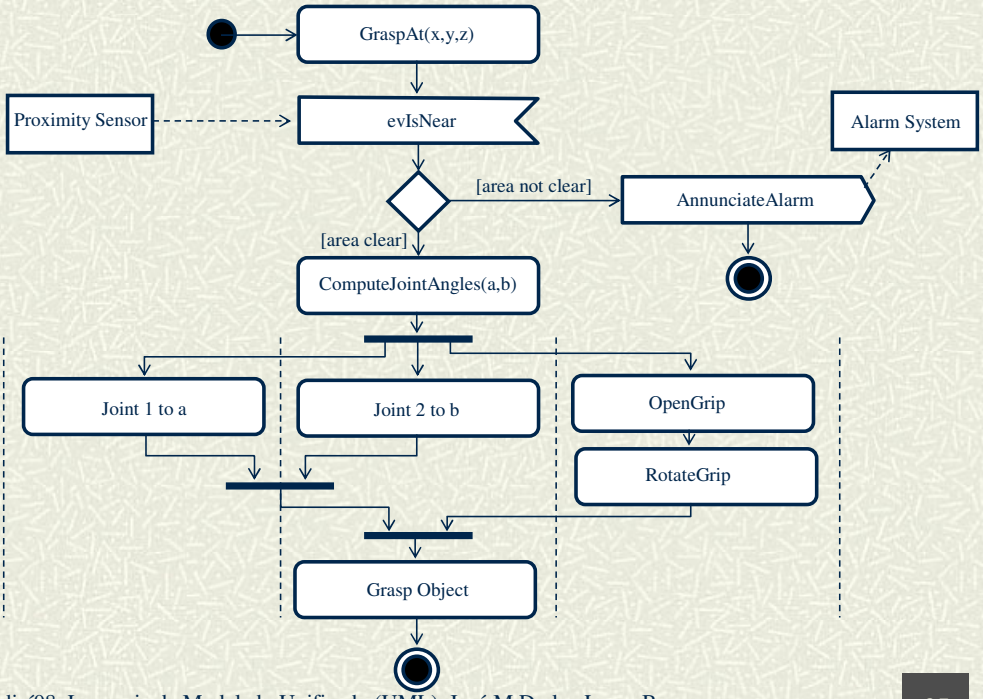
Los diagramas de actividad permiten formular ejecuciones concurrentes de actividades.

Barras de generación de concurrencia (Fork): El flujo de control se replica en varias líneas que se ejecutan concurrentemente.

Barras de sincronización (Joint): Las diferentes líneas de control de entrada se suspenden en ella hasta que todas la ha alcanzado. Luego se unifican en una única linea de control.

Pasillos de actividad (Swimlane): Representan a los objetos o procesos que soportan las diferentes líneas de flujo de control concurrentes.

Representación gráfica de diagramas de actividad.



Notas:

Diagrama de Interacción

- Describen los modelos de **comportamiento colaborativo** de grupos de elementos de modelado estructural.
- Se formulan como **escenarios parciales** de intercambios entre grupos de objetos que interactuan de mensajes (eventos, invocación de operaciones, acciones de creación y destrucción)
- Hay dos tipos equivalentes:
  - **Diagramas de colaboración:** Son diagramas de objetos instanciados en los que se explicitan las secuencias ordenadas de intercambio de mensajes.
  - **Diagramas de secuencia**: Son diagramas de interacción entre objetos en las que los mensajes se ordenan temporalmente.

Notas:

Diagramas de secuencias

- Un diagrama de secuencia muestra las interacciones entre objetos ordenados desde un punto de vista temporal.
  - Permite representar gráficamente las interacciones que se producen entre un conjunto de objetos que colaboran para llevar a cabo una función.
- Se corresponde con el mecanismo básico con el que una persona no experta concibe una interacción.
- Un conjunto de diagramas de secuencias puede llegar a describir de forma completa una funcionalidad, pero es tedioso y se suele describir con otros recursos.
  - Solo permite representar un escenario, esto es una posible ejecución de una función o caso de uso.

Notas:

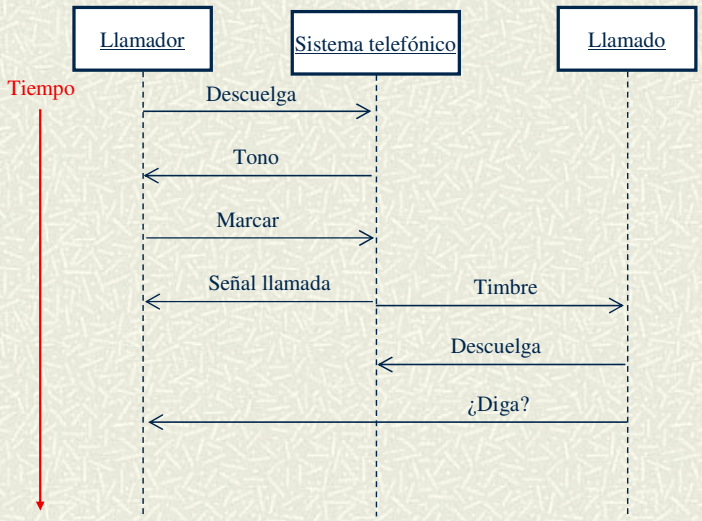
Los diagramas de secuencias son representaciones gráficas del flujo de control y son útiles en particular para visualizar las ejecución de los casos de uso. Además de utilizarlo como una herramienta en el análisis de requisitos también es muy útil como herramienta de análisis, como se verá en el próximo tema.

Los diagramas de secuencias requiere que se piense en términos de objetos. En un diagrama de secuencia, la vida de cada objeto se muestra como una línea continua vertical con el nombre del objeto y su clase en la parte superior.

Cada interacción entre objetos se muestra como una flecha horizontal desde el objeto que la inicia hasta el objeto que la recibe.

El orden de envío de los mensajes viene dado por la posición sobre el eje vertical.

Ejemplo:Establecimiento de una llamada telefónica



¿ Y si comunica?

Notas:

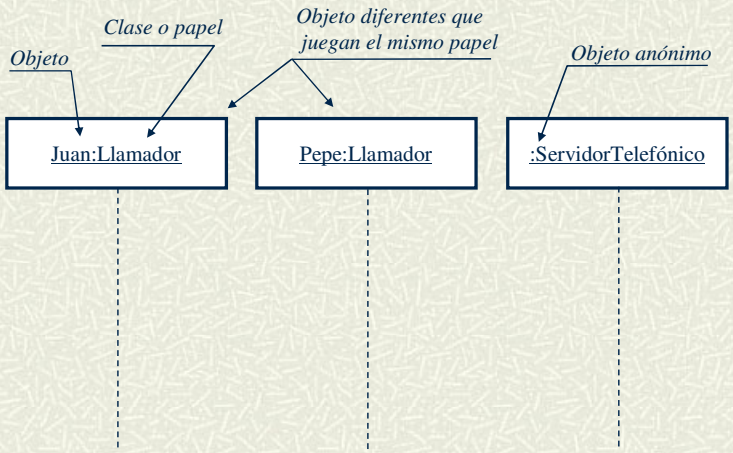
En la figura se muestra un ejemplo de diagrama de secuencias que describe el proceso de comunicación de dos personas (Llamador y llamado) a través de una línea telefónica.

La interacción entre objetos se conciben como mensajes que se intercambian entre objetos.

El diagrama de secuencias no describe la totalidad de las posibilidades que se pueden plantear, sino que únicamente describe una secuencia posible. Por ejemplo, en este caso no se contempla la posibilidad de que el objeto Llamado esté comunicando, y en consecuencia falle la conexión. Esta situación diferente debería formularse con otro diagrama de secuencia.

Generalmente no consideramos el tiempo que supone enviar un mensaje de un objeto a otro. En la mayoría de los lenguajes orientados a objetos ese tiempo es virtualmente instantáneo.

Las interacciones son entre objetos

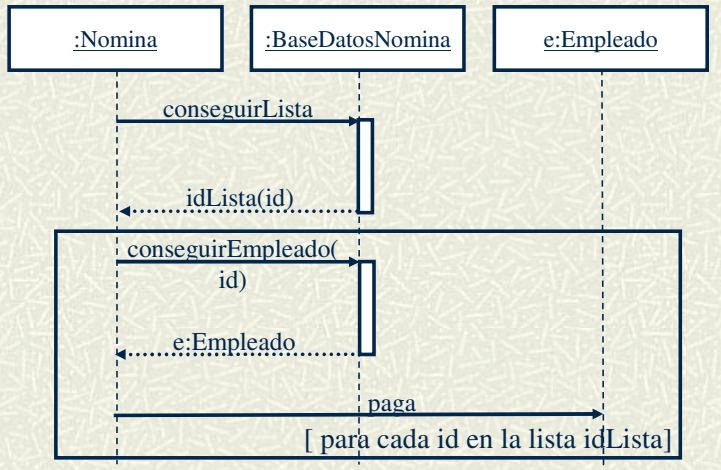


Notas:

Los participantes en un diagrama de secuencias son objetos concretos. Su denominación consta del identificador del objeto y de la clase o papel al que pertenecen.

Puede designarse un objeto como representativo de una clase haciendo únicamente referencia a la clases.

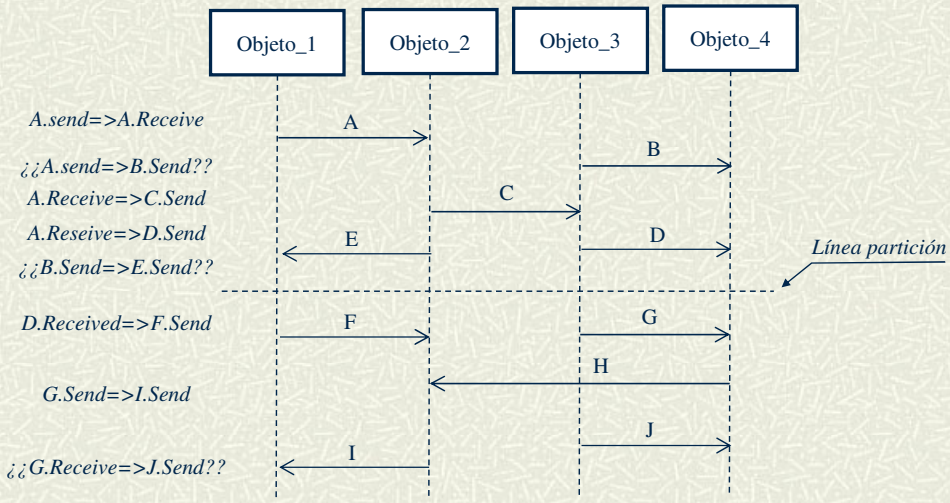
Ejemplo: nóminas de empleados




```
public class Empleado{
        public void paga(){...}
}
```


Notas:

Referencias temporales.



Notas:

Aunque la dirección vertical representa la evolución del tiempo, solo son comparables los instantes de las acciones si están referenciadas a la actividad de un mismo objeto.

• La acción de enviar un evento es siempre anterior a su recepción. Así, A.Send =>A.Receive (la acción A.Send precede a la acción A.Receive)

• Para dos acciones que no tengan referencia con un objeto común no son ordenables:

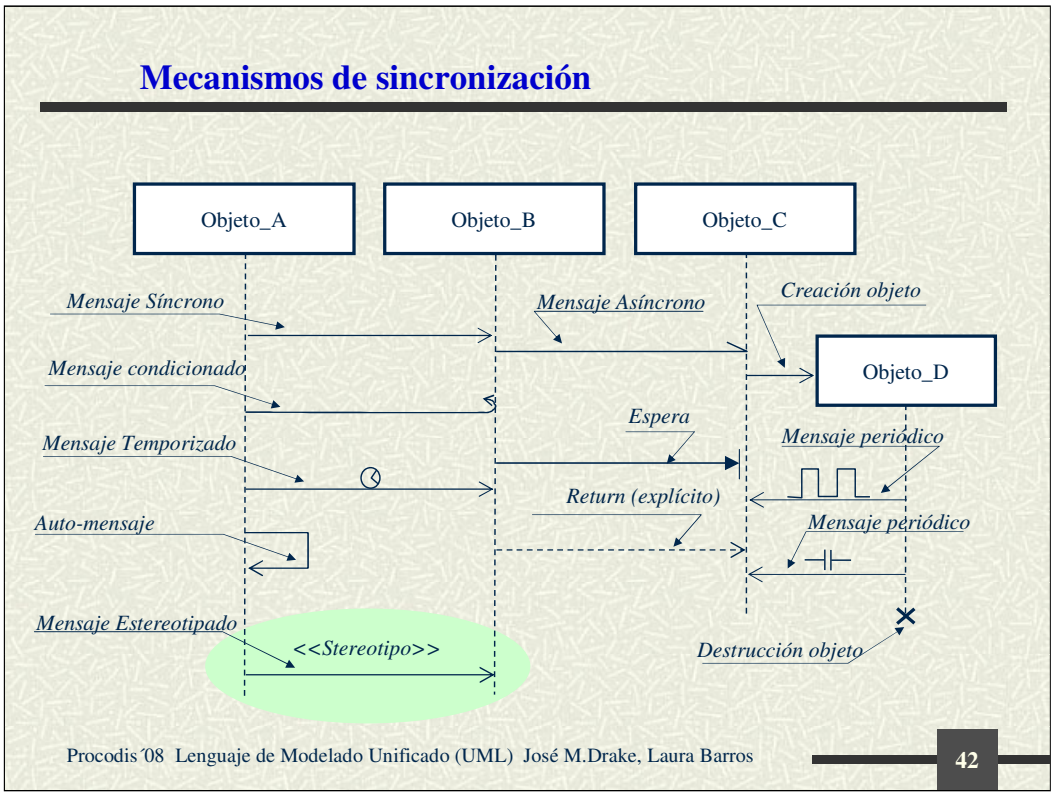
A.Send puede preceder o seguir a B.Send.

• Cuando existe una secuencia de referencias comunes, puede establecerse la precedencia.

G.Send=>G.Receive=>H.Send=>H.Receive=>I.Send=>I.Receive

• A través de una línea de partición horizontal, se pueden establecer precedencias explícitas:

E.Send=>E.Receive=>G.Send=>G.Receive



Notas:

Mensaje Síncrono: El flujo en el objeto que envía el mensaje se suspende hasta que es recibido por el objeto receptor.

Mensaje Asíncrono: El emisor transfiere el mensaje y continua sin esperar a que el mensaje sea recibido.

Mensaje Condicionado(Balking): El mensaje es transferido si el receptor está disponible para aceptarlo de forma inmediata, en caso contrario, no se transfiere.

Mensaje Temporizado: El mensaje es transferido con un tiempo de guarda. Si transcurre el tiempo de timeout y el receptor lo acepta, el mensaje no se emite.

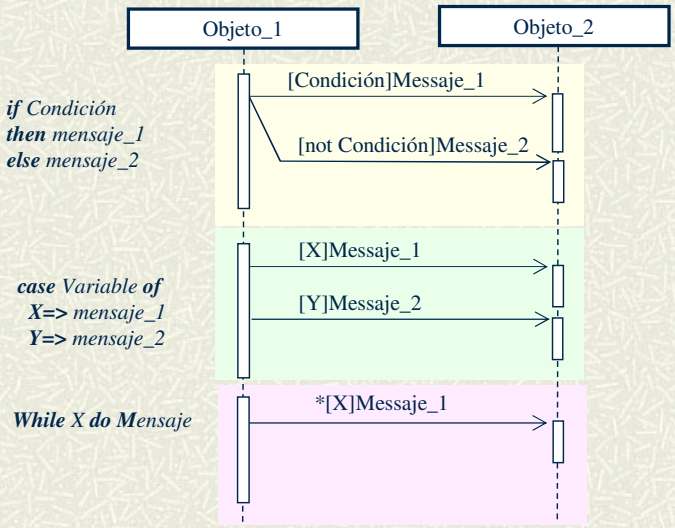
Espera: Un objeto se suspende hasta que otro objeto alcanza un cierto estado.

Return: Habitualmente el retorno de una invocación es implícita. Algunas veces interesa manifestar su ocurrencia.

Los patrones de generación de los mensajes pueden explicitarse mediante iconos.

Todos estos iconos están definidos en el estándar UML, pero no suelen estar soportados por las herramientas CASE. Por ello, se suele utilizar el método equivalente de la inclusión del estereotipo explícito.

Sintaxis en los diagramas de secuencias

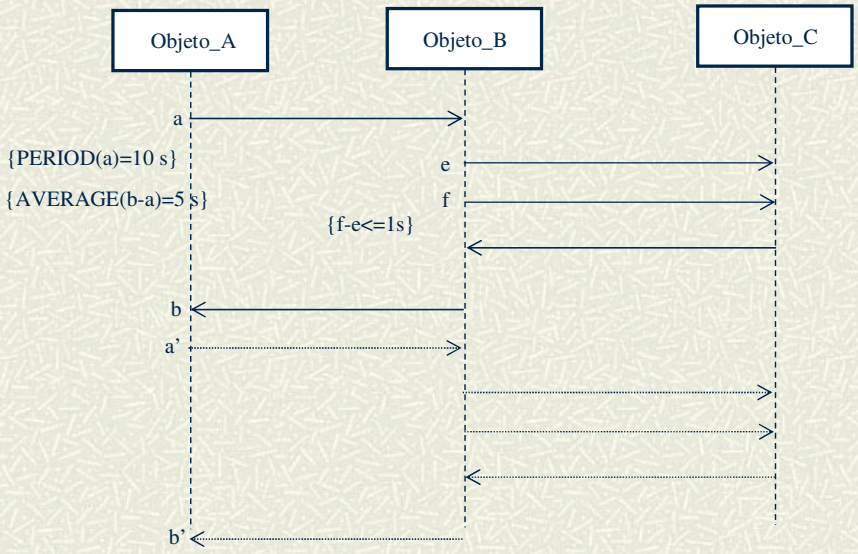


Notas:

En los diagramas de secuencias pueden establecerse precondiciones para la realización de una interacción. La precondición se formula entre paréntesis cuadrados. El mensaje se transfiere solo si la condición es cierta.

Haciendo uso de interacciones condicionadas, pueden describirse bifurcaciones condicionales, bifurcaciones enumeradas y bucles enumerados o condicionales

Restricciones temporales en los diagramas de secuencias. Etiquetas



Notas:

Los instantes en que se producen las acciones se pueden designar mediante etiquetas. Y haciendo uso de ellas se pueden formular múltiples restricciones sobre las respuestas temporales de los sistemas.

Diagrama de colaboración

- Modela las interacciones entre grupos de objetos que colaboran para implementar una funcionalidad o caso de uso.
- Hacen principal referencia a las vías de interconexión entre los objetos. Es especialmente idóneo para identificar interfaces y protocolos.
- Sólo describe escenarios concretos y no describe comportamientos generales.
- Tiene la misma capacidad expresiva que los diagramas de secuencias. La diferencia es que una hace referencia a los canales de comunicación y el otro a la secuencialidad temporal.

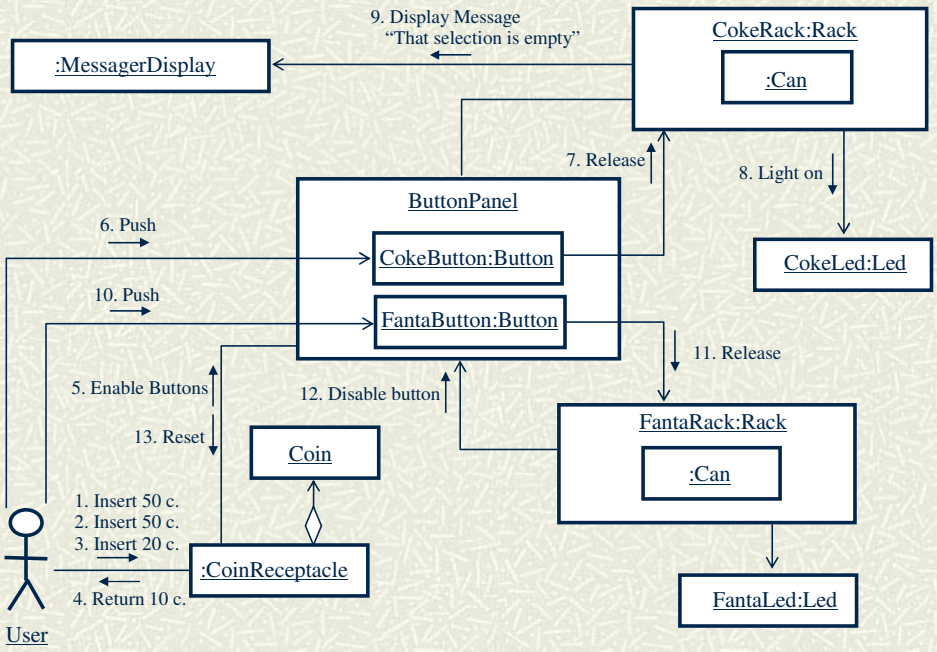
Notas:

Los diagramas de colaboración muestran interacciones entre objetos, resaltando la estructura espacial estática que permite la colaboración del grupo de objetos. Los diagramas de colaboración expresan a la vez el contexto de un grupo de objetos (a través de de la representación de los objetos y de los enlaces que están establecidos entre ellos) y las interacciones entre los objetos (por la representación de envío de mensajes).

El contexto de una interacción puede comprender los argumentos, las variables locales creadas durante la ejecución, así como los enlaces entre los objetos que participan en la interacción.

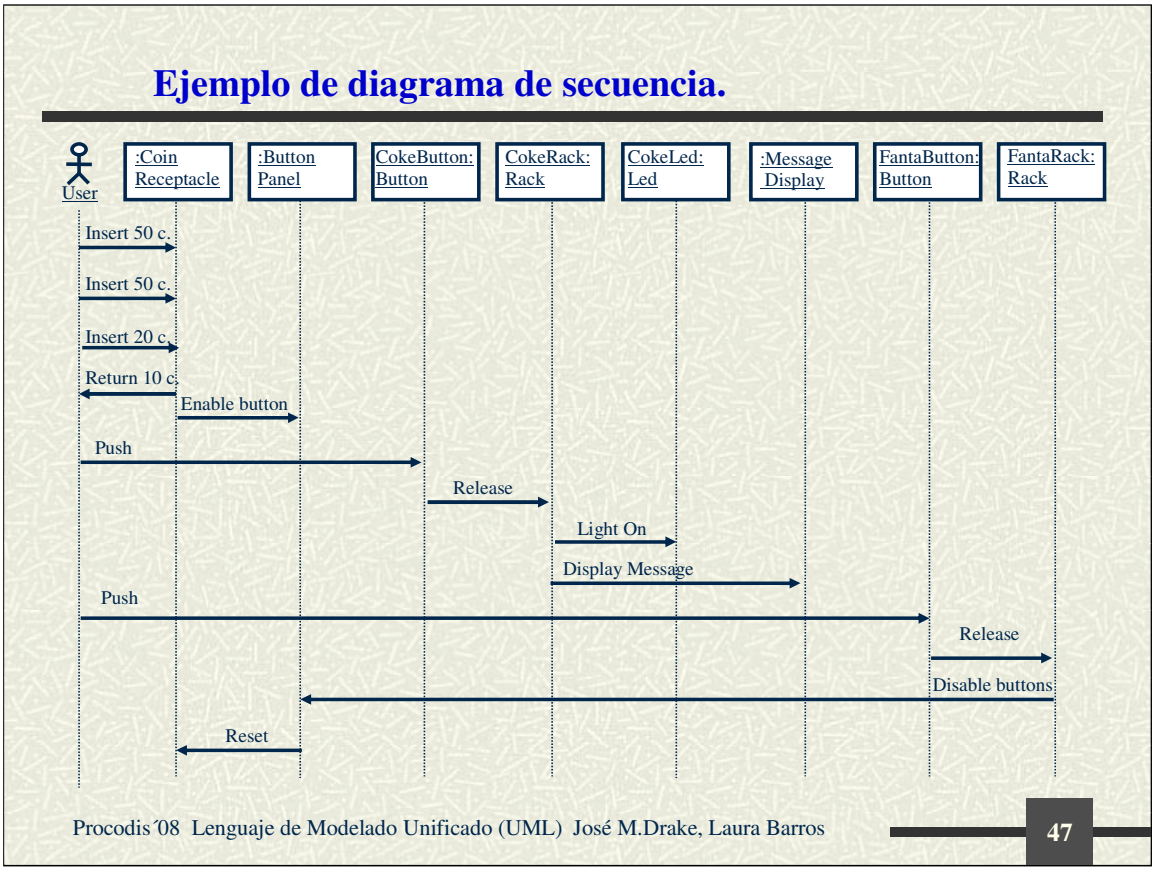
Una interacción se realiza mediante un grupo de objetos que colaboran intercambiando mensajes. Los mensajes se representan a los largo de los enlaces que enlazan los objetos por medio de flechas orientadas hacia el destinario del mensaje.

Ejemplo de diagrama de colaboración.



Notas:

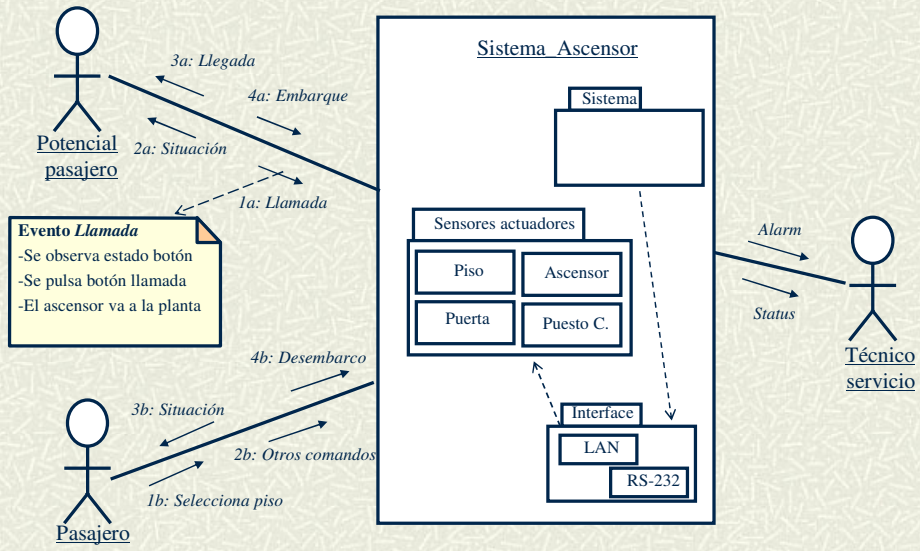
Escenario ejemplo: Drink Machine: El usuario introduce 60 céntimos como dos monedas de 50 céntimos y una moneda de 20 céntimos, y la máquina retorna una moneda de 10 céntimos. Selecciona Coke, pero la máquina no tiene este tipo de bebida y lo manifiesta a través de un LED y un mensaje. Luego elige Fanta que si está disponible, por lo que la máquina la suministra.



Notas:

Escenario ejemplo: Drink Machine: El usuario introduce 60 céntimos como dos monedas de 50 céntimos y una moneda de 20 céntimos, y la máquina retorna una moneda de 10 céntimos. Selecciona Coke, pero la máquina no tiene este tipo de bebida y lo manifiesta a través de un LED y un mensaje. Luego elige Fanta que si está disponible, por lo que la máquina la suministra.

El orden se expresa como índices de secuencia.



Notas:

En los diagramas de colaboración el tiempo (orden en el tiempo) se puede expresar introduciendo un índice en los mensajes.

En sistemas concurrentes en los que existen diferentes sesiones de interacción se utiliza un segundo índice de sesión.

Casos de Uso.

- Un caso de uso es una descripción del comportamiento de un sistema.
  - No hacen referencia a la implementación del sistema.
- Se escribe desde el punto de vista de un usuario (actor) que sabe que el sistema hace algo en particular.
- Captura una secuencia de eventos visible por los que pasa un sistema como respuesta a un estímulo de un único usuario.
  - No describen los mecanismos ocultos del sistema.
- Los casos de uso constituyen una representación de la funcionalidad en la fase de análisis y se utiliza como guía de las restantes fases (diseño, codificación, prueba y despliegue).
- Los casos de uso son descripciones textuales de los requisitos de comportamiento.

Notas:

Los diagramas de casos de uso tienen son un método alternativo y complementario a los diagramas de contexto como medio de especificar los requisitos de una aplicación software.

Jacobson creó la idea de los casos de uso al observar que, a pesar del gran número de ejecuciones potenciales, muchas aplicaciones están concebidas en términos de un número relativamente pequeño de interacciones típicas.

Muestran los tipos básicos de interacción entre el sistema y los elementos del entorno que operan con él. Los diagramas de casos de uso proporcionan un recurso alternativo bien para verificar el diagrama de contexto o de ayuda para construirlo.

Los casos de uso capturan una vista general de la funcionalidad del sistema con un método muy adecuado para ser interpretado por personas no técnicas como son los usuarios y los expertos de dominio. Suelen interpretarse como una guía de los escenarios de uso del sistema sobre los que se especifican los requerimiento del sistema.

Los casos de uso son también un método de descomposición de la funcionalidad del sistema en elemento de funcionalidad más básica, ya que UML proporcionan una semántica para expresar los casos de usos mas generales en función de casos de usos mas simples.

Los casos de uso son contenedores de la información que describen los requerimientos del sistema. Estos se pueden formular con diferentes niveles de detalles, mas cualitativos para el uso de los expertos de dominio y mas detallado y formales para el uso de los analistas de la aplicación.

La mayoría de los estándares de documentación establecidos (incluida IEEE 830-1993)son anteriores a la definición del caso de uso y en consecuencia no los tienen en cuenta. Es habitual tener que extender su formulación para que tengan cabida.

Actores

- Representa un tipo de objeto externo al sistema pero que interacciona con él.
- Un caso de uso siempre empieza en un actor.
- Actores pueden ser:
  - Actores principales: Usuarios que utilizan las funciones principales del sistema.
  - Actores secundarios: Personas que efectúan tareas administrativas o de mantenimiento del sistema.
  - Elementos externos: Equipos y dispositivos que forman el ámbito de la aplicación pero que no se desarrollan con ella.
  - Otros sistemas: Sistemas externos al que se desarrolla que interactuan con él.
- Un objeto puede implementar varios actores, y cada actor puede tener múltiples implementaciones.

Notas:

Los actores son cualquier cosa que interacciona el sistema que se desarrolla, por ejemplo, personas, otros software, hardware, dispositivos, redes, almacenes de datos, etc.. Cada actor define un particular "role". Cada entidad externa al sistema puede ser representado por uno o mas actores. Así, una persona física puede ser representada por varios actores , debido a que la persona juega diferentes papeles con relación al sistema. O varios objetos físicos pueden estar representados por un mismo actor, porque ellos mantienen una misma interacción en relación con el sistema.

Existen cuatro grandes tipos de actores:

•Los actores principales: Agrupan a alas personas que utilizan las funciones principales del sistema. En el caso del software de un cajero son los clientes que hacen uso de él.

•Los actores secundarios: Son las personas que hacen tareas administrativas o de mantenimiento. En el ejemplo de un cajero es p.e. la persona que recarga el cajero.

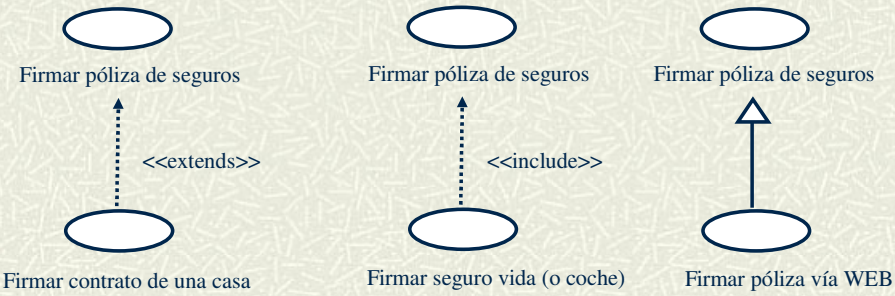
•Los Elementos externos: Agrupa a los dispositivos que forman parte del ámbito de la aplicación y que deben ser utilizado. En el ejemplo del cajero pueden ser p.e. la impresora.

•Otros sistemas: Agrupa a los sistemas externos con lo que el sistema debe interactuar. En el ejemplo del cajero, un actor de este tipo puede ser el sistema del sistema bancario.

Los actores son siempre externos al sistema. Para tratar de localizar los actores, debemos buscar categorías de cosas que aparezcan como consecuencia de las siguientes preguntas: ¿Quien usa el sistema? ¿Quién instala, arranca y apaga el sistema? ¿Quien mantiene al sistema? ¿Que otros sistemas usa el sistema? ¿Quién obtiene información del sistema? ¿Quién provee información al sistema? ¿Ocurre algo de forma automática en tiempos prefijados?.

Diagramas de casos de uso

- Los diagramas de caso de uso constituyen un método alternativo y complementario a los diagramas de contexto para formular los requisitos del sistema.
- Debe ser concebida como un contenedor para incluir diagramas, textos, etc. con los que se detallan las funcionalidades y las QoS del sistema.
- Entre los casos de uso pueden establecerse relaciones de:
  - Generalización.
  - Inclusión(include).
  - Extensión(extend).



Notas:

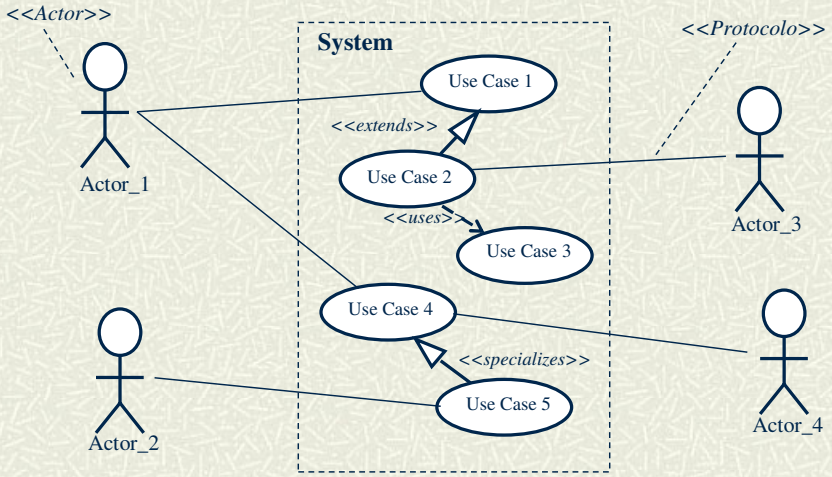
UML define tres tipos de relaciones entre casos de uso:

•**Inclusión**: La relación inclusión utiliza un estereotipo de dependencia.Establece que un segmento de comportamiento que esta representado por el caso de uso base es utilizado por otro caso de uso que llamamos caso de uso cliente. Esta relación es especialmente útil si existe un segmento de funcionalidad que se repite en diferentes contextos y no se desea repetir su especificación.

•**Extensión**: La relación de extensión es representada por una relación de dependencia entre los dos casos de uso, con la flecha apuntando hacia la clase base. El caso de uso a ser extendido es denominado caso de uso base, y el caso de uso extendido es llamado caso de uso cliente. La idea es que el caso de uso base identifica un número de puntos de extensión específicos en descripción de comportamiento. Estos puntos son localizaciones a partir de los que los clientes pueden insertar un **comportamiento adicional.**

•**Generalización**: Esta relación declara que un caso de uso (llamado general) es una forma mas general que otro (llamado derivado). Al igual que en la generalización de las clases, la clase mas especializada hereda de la base todas sus características, de las cuales algunas pueden ser sobrescritas y especializadas por la derivada.

Elementos de los diagramas de casos de uso UML.



Notas:

El modelo de casos de uso de una aplicación se compone de actores, el sistema (como una caja negra) y los propios casos de uso. La funcionalidad de un sistema se formula examinando las necesidades funcionales que requiere del sistema de cada actor, expresadas en forma de familias de interacciones que se incluyen en un caso de uso.

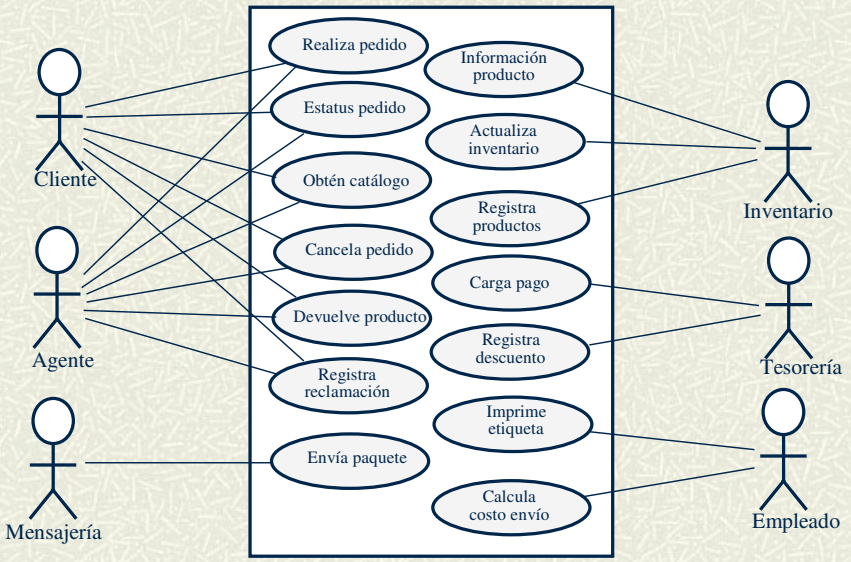
Un actor representa un papel interpretado por una persona u otro sistema que interacciona con el sistema que se describe. La misma persona física puede representar diferentes papeles y asi mismos, muchas personas pueden ser representados por un mismo actor. Los actores deben ser descritos de una forma clara mediante un texto de sólo algunas líneas.

Los casos de uso se determinan observando y precisando, actor por actor , las secuencias de interacción (que llamaremos escenarios) que llevan a cabo con el sistema. Los casos de uso agrupan a familias de escenarios que desempeñan un mismo papel funcional visto desde el usuario. Los casos de uso son abstracciones del diálogo entre los actores y el sistema: describen interacciones potenciales, sin entrar en los detalles de cada usuario.

Los escenarios son instancias de los caos de uso. Los escenarios representan secuencias de mensajes entre objetos que colaboran para producir algún tipo de comportamiento del sistema. Los escenarios constituyen un método muy intuitivo de representar los detalles de los requerimiento, en particular los que implican restricciones temporales.

Los enlaces entre actores y casos de uso representan los protocolos, que constituyen el hilo de conductor de la secuencia de eventos que intercambian ambos.

Diagrama de casos de uso del ejemplo WEB-Cántabra



Notas:

Casos de uso

Realiza pedido: Un cliente crea un pedido, selecciona los productos y ordena el pago

Estatus pedido: Un cliente requiere información del estado de su pedido.

Obtén catálogo: Un cliente requiere el catálogo de productos.

Cancela pedido:Un cliente da de baja un pedido ya registrado.

Devuelve producto: Un cliente devuelve un producto por fallo.

Registra reclamación: Un cliente envía un mensaje de reclamación a la empresa.

Envía paquete: Se ordena al sistema de mensajería que se envíe un paquete.

Información del producto: Informa el estado de un producto en el inventario.

Actualiza inventario: Se analiza el inventario y se realiza los pedidos a los suministradores.

Registra producto: Da de alta en el inventario un producto recibido de los suministradores.

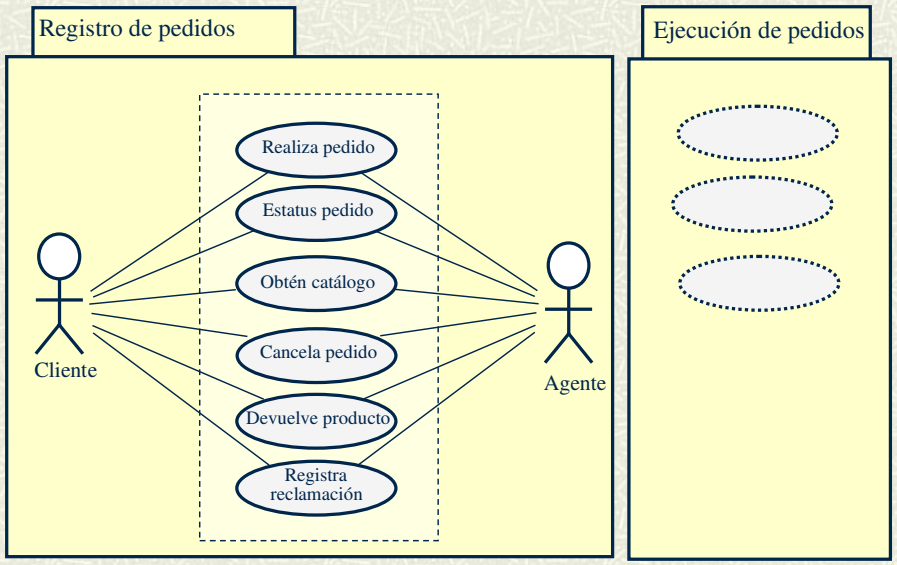
Carga pago: Anota el pago relativo a un pedido.

Registra descuento: Anota en la cuenta de un cliente un descuento recibido.

Imprime etiqueta: Imprime la etiqueta de envío de un pedido.

Calcula costo envío: Calcula el gasto de envío de un pedido.

Uso de paquetes para organizar los casos de uso.



Notas:

Si el diagrama de casos de uso que resulta es demasiado grande y enmarañado, se pueden crear diferentes diagramas de casos de uso y cada uno de ellos representaría una vista del diagrama general. Cada diagrama puede representar un área de principal de funcionalidad del sistema.

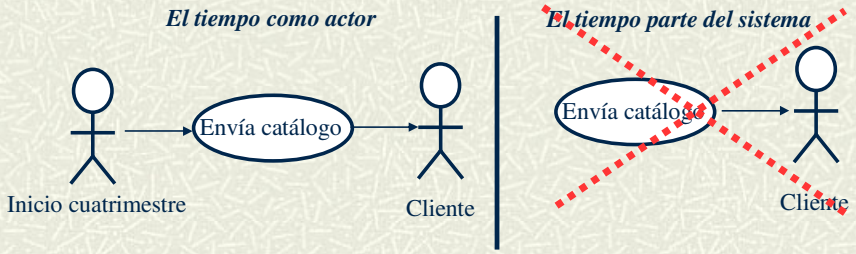
O la funcionalidad relativa a un actor, etc. Incluso puede ser conveniente que un mismo caso de uso a parezca en diferentes diagramas. En estos casos la utilización de una herramienta CASE es imprescindible para mantener la coherencia del modelo.

En sistema grandes puede ser útil organizar los diagramas en paquetes, cada uno de ellos representa la funcionalidad de un área o de un subsistema.

Gestión de interacciones temporizadas.

- Ciertas actividades tienen lugar en instantes determinados de tiempo y son iniciadas desde el reloj interno.
- Hay dos formas de abordar esta situación:
  - El tiempo puede ser tratado como un actor que inicia el caso de uso.
  - El tiempo se considera parte del sistema y el actor que se relaciona con el caso de uso realiza la interacción motivado por ella.

Es preferible la primera situación ya que mantiene el principio de que los casos de uso se inician siempre por un actor.



Notas:

Documentación de un caso de uso: "Realiza pedido"

Identificador: Realiza_pedido **Actores que lo inician:** Cliente y Agente.

Preconciciones: Un cliente registrado en el sistema ha accedido correctamente al sistema.

Secuencia de eventos de flujo:

1. El cliente introduce su nombre y dirección.
2. Si el cliente introduce el ZIP, el sistema introduce la ciudad y región.
3. El cliente introduce los códigos de los productos que desea incluir en el pedido.
4. El sistema aporta la descripción y el precio del producto.
5. El sistema almacena temporalmente la lista de productos incluidos en el pedido.
6. El cliente introduce la información de la tarjeta de pago.
7. El cliente pulsa el control Ejecuta.
8. El sistema verifica la información, almacena el pedido temporalmente y requiere confirmación del banco. Si la información es incorrecta, el sistema requiere su corrección al cliente.
9. Cuando el pago es confirmado, se acepta el pedido, se le asigna un ID que se retorna al cliente.

Postcondiciones: Si el pedido no ha sido cancelado, es registrado en el sistema y confirmado al cliente.

Notas:

La información asociada a cada caso de uso es:

Identificador

Agente o agentes que lo inician.

Precondiciones (estado de partida previsto) y post-condiciones (situación que se produce cuando el error concluya).

Funcionalidad básica.

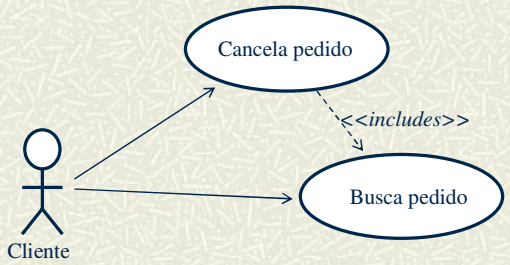
Alternativas de error o excepción.

Asociaciones de casos de uso

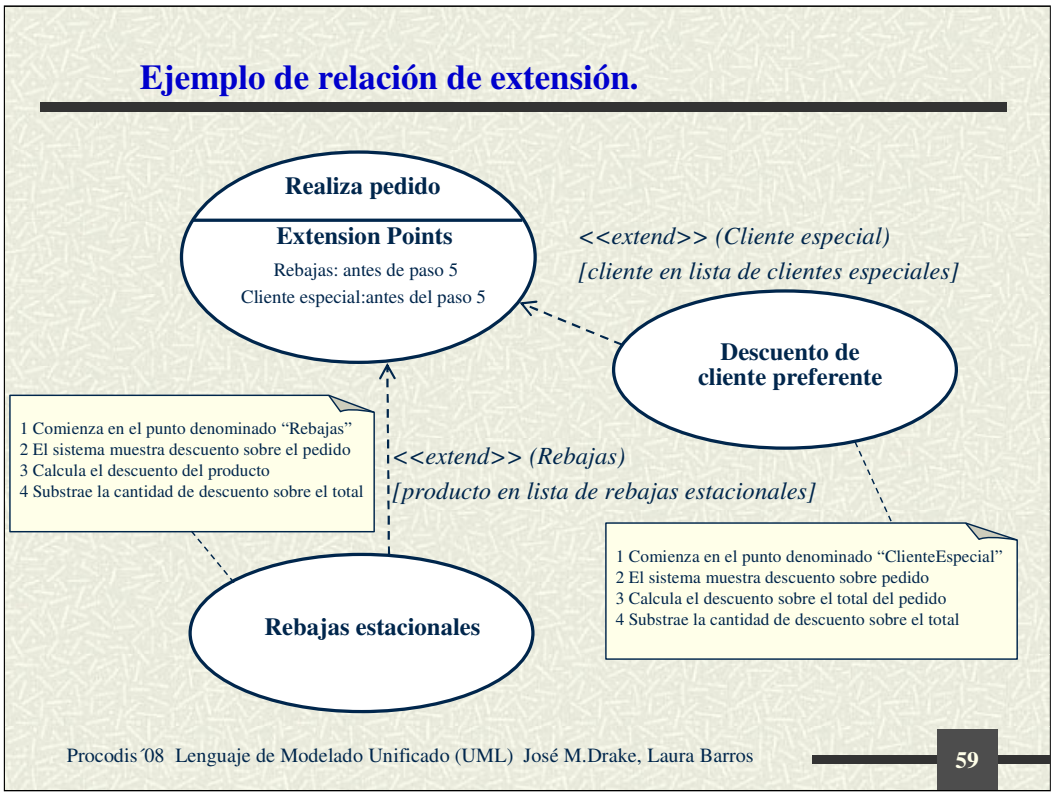
- Cuando se realiza el análisis de requerimientos se simplifica si unos casos de uso se formulan en función de otros previamente definidos.
- Existen tres tipos de relaciones básicas:
  - **Inclusión** <<includes>> o <<uses>>: Es una relación de dependencia entre casos de uso. Un caso de uso cliente incorpora en su secuencia el contenido de otro caso de uso ya definido. Esta relación evita repetir segmentos de funcionalidad que se replican en múltiples contextos.
  - **Extensión** <<extends>>: El caso de uso que se extiende es tomado como base para definir otros casos de uso derivados de él por extensión. El caso base ofrece un conjunto de puntos de extensión, y a partir de ellos los casos derivados definen la nueva funcionalidad.
  - **Herencia**: Un caso de uso es mas general que otro, el segundo hereda sus características mas ciertos elementos de especialización

Notas:

Ejemplo relación de inclusión



Notas:



Notas:

Extensión de caso de uso: "Rebajas estacionales"

Secuencia básica:

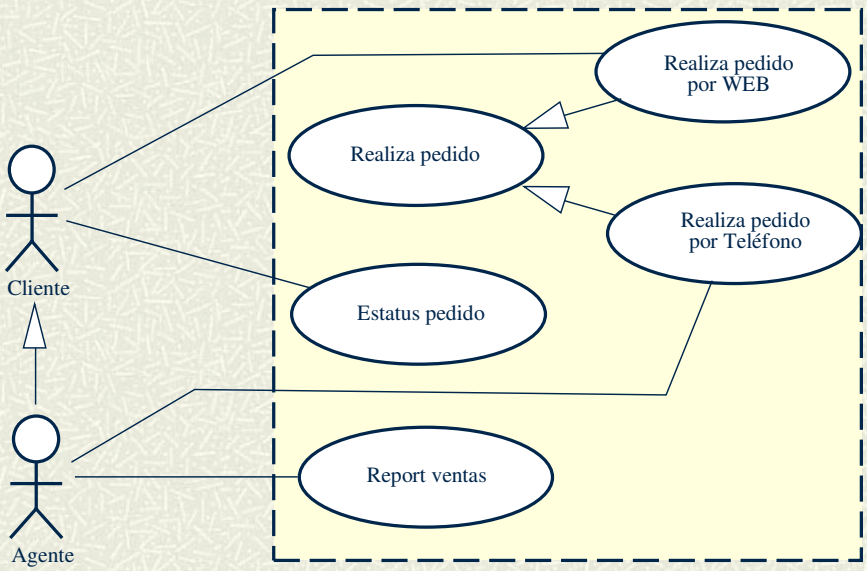
1. El caso de uso comienza en el paso de nominado "Rebajas".
2. El sistema muestra el descuento sobre el pedido.
3. El sistema calcula el descuento sobre el producto.
4. El sistema substrae la cantidad de descuento sobre la suma total del pedido.

Extensión de caso de uso: "Descuento de cliente preferente"

Secuencia básica:

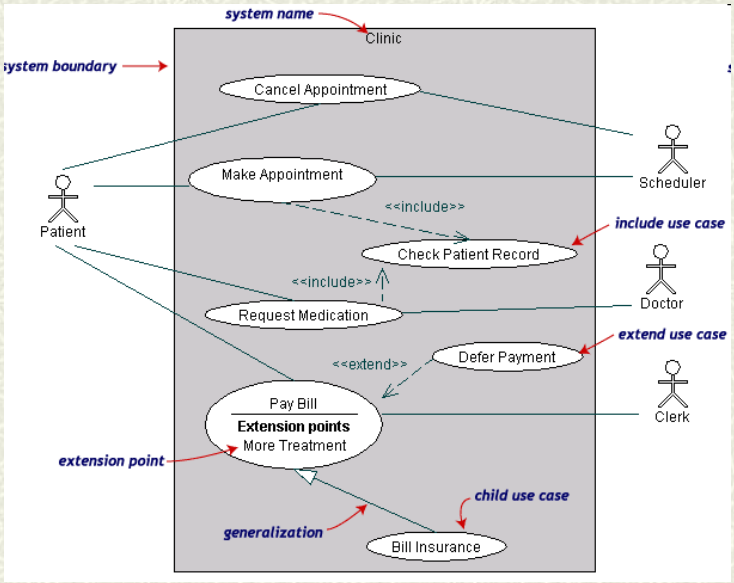
1. El caso de uso comienza en el paso de nominado "Cliente especial".
2. El sistema muestra el descuento sobre el pedido.
3. El sistema calcula el descuento total sobre el pedido del cliente.
4. El sistema substrae la cantidad de descuento sobre la suma total del pedido.

Ejemplo de “Herencia” entre Casos de Uso



Notas:

Diagrama de casos de uso del ejemplo visita al médico



- Caso uso Paciente visita al médico:
  - Pide cita médica en la seguridad social.
  - El doctor hace chequeo médico.
  - El cliente paga la cuenta que le entrega el administrativo.

Notas: