

Contenido

1. INTRODUCCIÓN	4
1.1 MODELADO CON UML	4
1.1.1 ¿Qué es UML?.....	4
1.1.2 Historia de UML.....	4
1.1.3 Conceptos básicos de Ingeniería del Software.....	4
1.1.4 Objetivos y ventajas de UML.....	7
1.1.5 Estructura del estándar	7
1.1.6 Clasificación de los diagramas	8
1.2 DIAGRAMAS DE UML.....	9
1.2.1 Diagramas de estructura.....	9
1.2.2 Diagramas de comportamiento	12
1.3 PRINCIPIOS BÁSICOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS	18
1.3.1 Introducción a la POO	18
1.3.2 Objetos y Clases	18
1.3.3 Relaciones entre Clases.....	19
1.3.4 Jerarquías entre Clases.....	19
1.4 ELEMENTOS DE NOTACIÓN COMUNES	20
1.5 CONCEPTOS BÁSICOS DE MODELADO.....	21
2. CASOS DE USO	23
2.1 INTRODUCCIÓN.....	23
2.2 CONCEPTOS BÁSICOS	23
2.3 DIAGRAMAS DE CASOS DE USO	25
2.4 ACTORES	26
2.4.1 Representación en UML.....	27
2.4.2 Relaciones entre actores.....	27
2.5 CASOS DE USO	28
2.6 RELACIONES ENTRE CASOS DE USO.....	29
2.6.1 Relación de Generalización.....	30
2.6.2 Relación de Extensión	30
2.6.3 Relación de Inclusión	31
2.6.4 Comentarios.....	32
2.7 ORGANIZACIÓN DE LOS CASOS DE USO Y DIAGRAMAS	33
2.8 COMPLETANDO EL MODELADO	34
2.8.1 Descripción de casos de uso.....	34
2.8.2 Validación de casos de uso.....	35
2.8.3 Realización de casos de uso	35
2.8.4 Testeo de casos de uso.....	36
3. DIAGRAMAS DE CLASES	37
3.1 INTRODUCCIÓN.....	37
3.2 CLASES Y RELACIONES ENTRE CLASES.....	38
3.2.1 Clases.....	38
3.2.2 Relaciones entre clases.....	39
3.2.3 Interfaces y paquetes	41
3.3 NOTACIÓN	42
3.3.1 Notación de las clases	42

3.3.2	Notación de las relaciones de clases	43
3.3.3	Notación de las interfaces y los paquetes.....	46
3.4	DIAGRAMAS DE OBJETOS	47
3.4.1	Diagrama de clases y diagrama de objetos.....	47
3.5	ASPECTOS AVANZADOS DEL DIAGRAMA DE CLASES.....	48
3.5.1	Clases de asociación	48
3.5.2	Restricción XOR	48
3.5.3	Asociación ternaria	49
3.5.4	Agregación compartida	49
3.5.5	Conjuntos de generalización	49
3.6	DIAGRAMA DE PAQUETES	51
3.6.1	Representación	52
3.6.2	Visibilidad.....	52
3.7	RELACIONES DE LOS DIAGRAMAS DE PAQUETES	53
3.7.1	Dependencia	53
3.7.2	Importación	53
3.7.3	Acceso.....	54
3.7.4	Dependencias cíclicas	55
4.	ESTRUCTURA COMPUESTA	56
4.1	ELEMENTOS CONSTITUTIVOS	56
4.2	MOSTRANDO LA ESTRUCTURA INTERNA	59
4.3	DISEÑANDO COLABORACIONES	60
4.4	PATRONES DE DISEÑO Y UML	60
5.	ARQUITECTURA FÍSICA.....	63
5.1	CONCEPTOS BÁSICOS	63
5.2	DIAGRAMAS DE COMPONENTES	63
5.2.1	Componentes	65
5.2.2	Artefactos.....	65
5.2.3	Interfaces.....	65
5.2.4	Relaciones de dependencia.....	66
5.3	DIAGRAMAS DE DESPLIEGUE	66
5.3.1	Nodos.....	67
5.3.2	Conexiones.....	68
5.3.3	Artefactos desplegados.....	68
6.	DIAGRAMAS DE INTERACCIÓN.....	70
6.1	LOS DIAGRAMAS DE INTERACCIÓN EN UML.....	70
6.2	DIAGRAMAS DE SECUENCIA.....	70
6.2.1	Participantes.....	71
6.2.2	Tiempo	72
6.2.3	Eventos, señales o mensajes	72
6.2.4	Barras de activación.....	72
6.3	EJEMPLOS DE DIAGRAMAS DE SECUENCIA	73
6.4	FRAGMENTOS COMBINADOS DE LOS DIAGRAMAS DE SECUENCIA.....	74
6.4.1	Fragmento combinado.....	74
6.4.2	Tipos de fragmentos.....	75
6.4.3	Ejemplos de tipos de fragmentos	76
6.5	DIAGRAMAS DE COMUNICACIÓN.....	77
6.5.1	Componentes del diagrama de comunicación.....	78

6.5.2	<i>Secuenciación de los mensajes</i>	78
6.5.3	<i>Cláusulas de iteración y condición</i>	79
6.6	COMPARACIÓN DEL DIAGRAMA DE SECUENCIA Y EL DIAGRAMA DE COMUNICACIÓN.....	79
6.7	DIAGRAMA GENERAL DE INTERACCIÓN	81
6.8	DIAGRAMAS DE TIEMPOS	82
7.	DIAGRAMAS DE ACTIVIDAD Y ESTADO	85
7.1	DIAGRAMA DE ACTIVIDADES. CONCEPTOS BÁSICOS	85
7.2	ACCIONES Y FLUJOS.....	85
7.2.1	<i>Decisiones</i>	86
7.2.2	<i>Acciones paralelas</i>	86
7.2.3	<i>Conectores</i>	87
7.3	PARTICIÓN DE ACTIVIDADES.....	87
7.4	OBJETOS, SEÑALES Y PINS.....	89
7.4.1	<i>Objetos</i>	89
7.4.2	<i>Señales</i>	89
7.4.3	<i>Pins</i>	90
7.5	MÁQUINA DE ESTADOS. INTRODUCCIÓN	91
7.6	TRANSICIONES.....	92
7.6.1	<i>Sintaxis de los eventos</i>	92
7.7	ENVÍO DE MENSAJES ENTRE MÁQUINAS Y ESTADOS COMPUESTOS.....	95
7.7.1	<i>Envío de mensajes entre máquinas</i>	95
7.7.2	<i>Estados compuestos</i>	95

1. INTRODUCCIÓN

1.1 MODELADO CON UML

1.1.1 ¿Qué es UML?

UML son las siglas de Unified Modeling Language (Lenguaje Unificado de Modelado). UML es el **lenguaje de modelado** de sistemas software más conocido y utilizado en la actualidad. Se trata de un lenguaje **gráfico** para visualizar, especificar, construir y documentar un **sistema de software**.

Hay que tener en cuenta que UML es una **técnica** y no un método o proceso. Puede usarse de varias formas para soportar una metodología de desarrollo de software (tal como el Proceso Unificado), pero en sí mismo no especifica qué metodología o proceso utilizar.

Aunque fue ideado inicialmente para modelar sistemas software, su flexibilidad le permite ser usado para modelar **cualquier sistema** p.ej. bases de datos o incluso sistemas no-software.

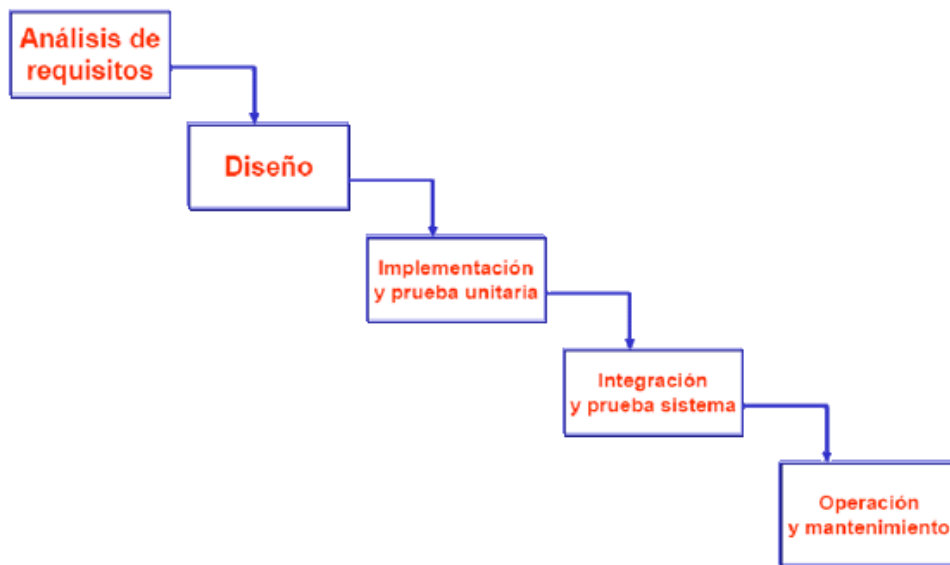
1.1.2 Historia de UML

- Guerra de las metodologías: (principios de los 90)
 - Booch - Grady Booch (Rational Software)
 - OMT (Object Modeling Technique) - James Rumbaugh (Rational Software)
 - OOSE (Object-Oriented Software Engineering) - Ivar Jacobson (Objectory)
- A finales de 1994 Booch y Rumbaugh de Rational Software empezaron a unificar las metodologías Booch y OMT. En 1995 se les unió Jacobson con su metodología OOSE.
- UML v1.1 aceptada por OMG (Object Management Group) (estandarización). 1997.
- IBM compra Rational. 2003.
- En 2004 aparece la versión 2.0 de UML.

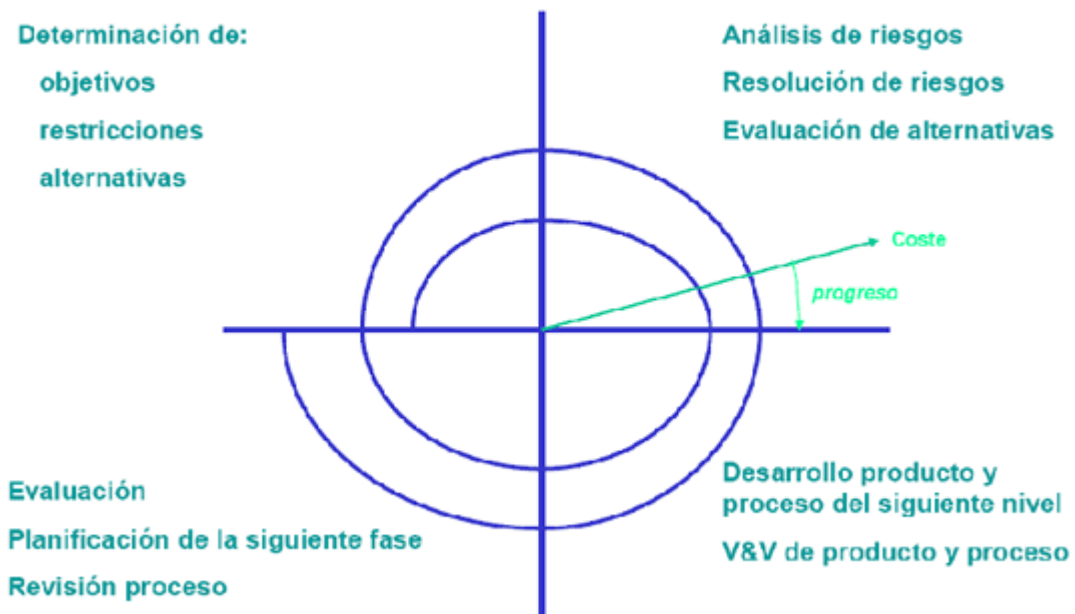
1.1.3 Conceptos básicos de Ingeniería del Software

Ciclo de vida

Idea "filosófica" sobre cómo construir un sistema de información o programa. Constituye el conjunto de fases y actividades empleadas en la obtención de un producto. Ejemplos: cascada, espiral, iterativo, incremental,...



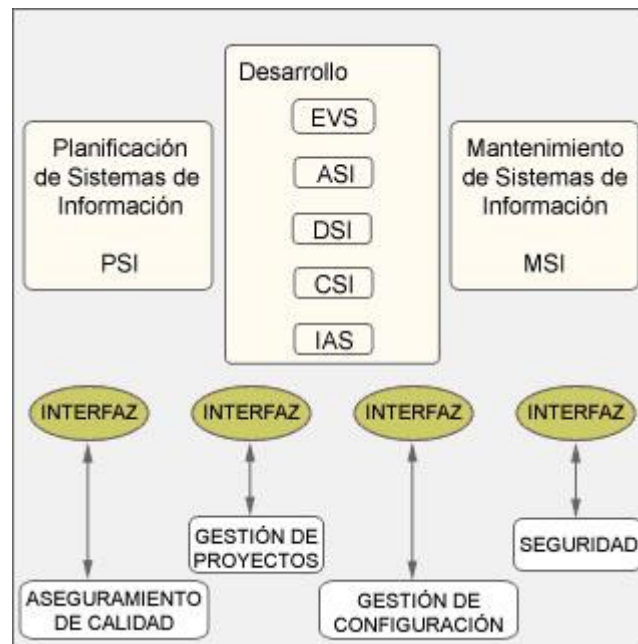
Ciclo de vida en cascada



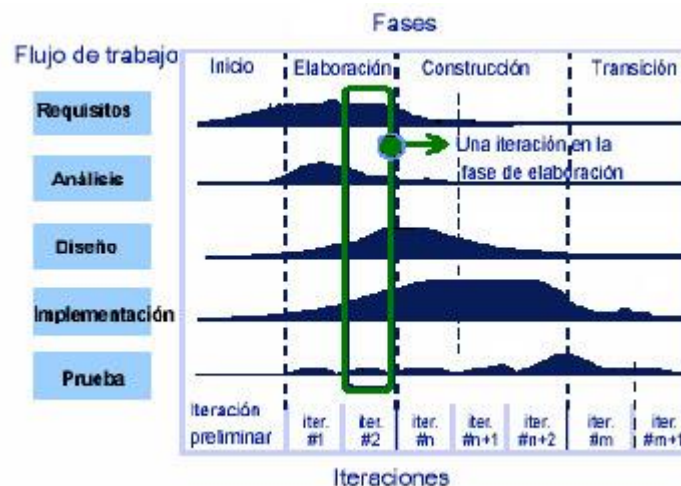
Ciclo de vida en espiral

Metodología

Implementa un ciclo de vida; es decir, define el conjunto de actividades que se llevan a cabo. Hay varias metodologías que emplean UML de distintas maneras. Ejemplos: Proceso Unificado, Métrica 3, Ágiles,...



Métrica 3



Proceso Unificado

Herramientas / Técnicas de apoyo

Son utilizadas por las metodologías para describir los conceptos y procesos. Ejemplos: UML, diagrama entidad-relación, diagrama de flujo de datos,...

Aplicaciones / Herramientas Software

Permiten diseñar y crear aplicaciones software utilizando las técnicas de apoyo. Ejemplos: IBM Rational Rose, IBM Software Architect, Sparx Systems Enterprise Architect, Oracle Designer, MS Visio,...

1.1.4 Objetivos y ventajas de UML

A continuación se enumeran los principales objetivos y ventajas de UML:

Objetivos de UML

- Su objetivo principal es definir una notación unificada, no un proceso.
- Establecer las bases para un método de desarrollo orientado a objetos común, estable y expresivo.
- Se ha conseguido hacer independientes la notación y el proceso.

Objetivos de UML 2.0 (específicos de la nueva versión del estándar)

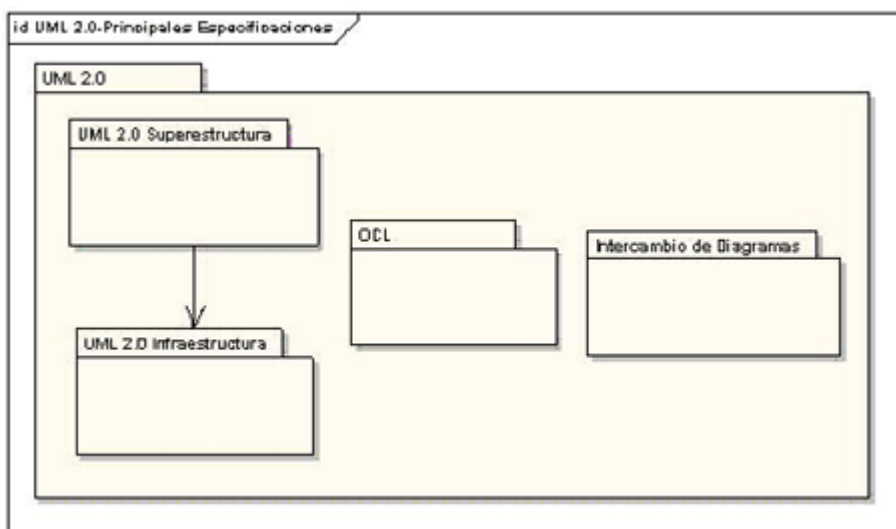
- Hacer el lenguaje de modelado mucho más extensible de lo que era.
- Permitir la validación y ejecución de modelos creados mediante UML.

Ventajas de UML

- Es un estándar abierto.
- Soporta el ciclo de vida de desarrollo de software completo.
- Soporta muchas áreas de aplicaciones.
- Se basa en experiencias y necesidades de la comunidad de usuarios.
- Soportado por muchas herramientas.

1.1.5 Estructura del estándar

La estructura de UML se describe con un diagrama de paquetes de UML. Por tanto, puede decirse que UML se describe a sí mismo:



Superestructura

Es la especificación que se usa más a menudo. Aquí se encuentran todos los diagramas que la mayoría de los desarrolladores conocen.

Infraestructura

Define los conceptos de bajo nivel de UML. Contiene las clases base que forman la superestructura. Permite ampliar el lenguaje definiendo nuevas superestructuras.

OCL (Object Constraint Language)

Lenguaje de restricción. Lenguaje sencillo para escribir restricciones y expresiones sobre los distintos elementos del diagrama. Junto con la infraestructura permite ampliar el lenguaje y reducirlo cuando sea necesario.

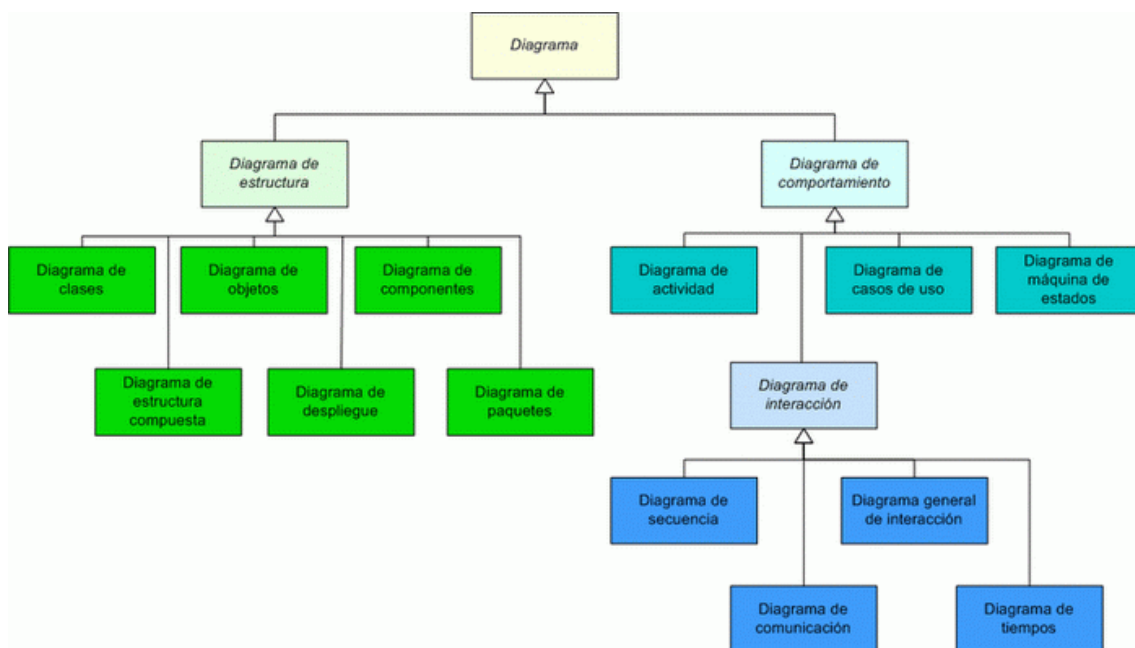
XMI / Intercambio de diagramas

Permite compartir diagramas entre diferentes herramientas de modelado UML.

1.1.6 Clasificación de los diagramas

Los diagramas UML se clasifican en dos grupos:

- **Diagramas de estructura:** reflejan la organización o estructura física del sistema (clases, métodos, atributos, interfaces, paquetes...).
- **Diagramas de comportamiento:** capturan el comportamiento y la interacción de los distintos elementos del sistema. Dentro de los diagramas de comportamiento se incluyen los diagramas de **interacción**.



1.2 DIAGRAMAS DE UML

Existen en total 13 tipos de diagramas que se dividen en 2 grandes grupos:

1.2.1 Diagramas de estructura

Diagrama de clases

Ilustra la estructura estática del sistema y las relaciones entre los distintos objetos. Usa clases e interfaces para mostrar información de las entidades que forman el sistema.

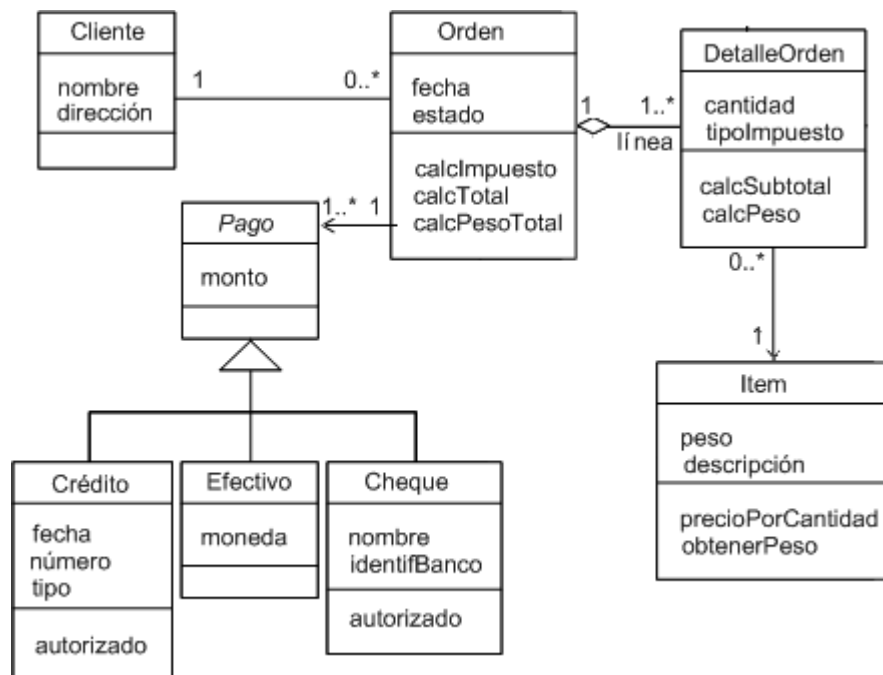


Diagrama de objetos

Usa la misma notación que los diagramas de clases y muestra cómo se relacionan las instancias de las clases (un objeto es una instancia de una clase) en un instante dado de tiempo.

Diagrama de componentes

Representa los componentes que componen el sistema, así como sus relaciones, interacciones e interfaces públicas. Suele mostrar componentes software finales: ejecutables (exe), librerías (dll), jar, ...

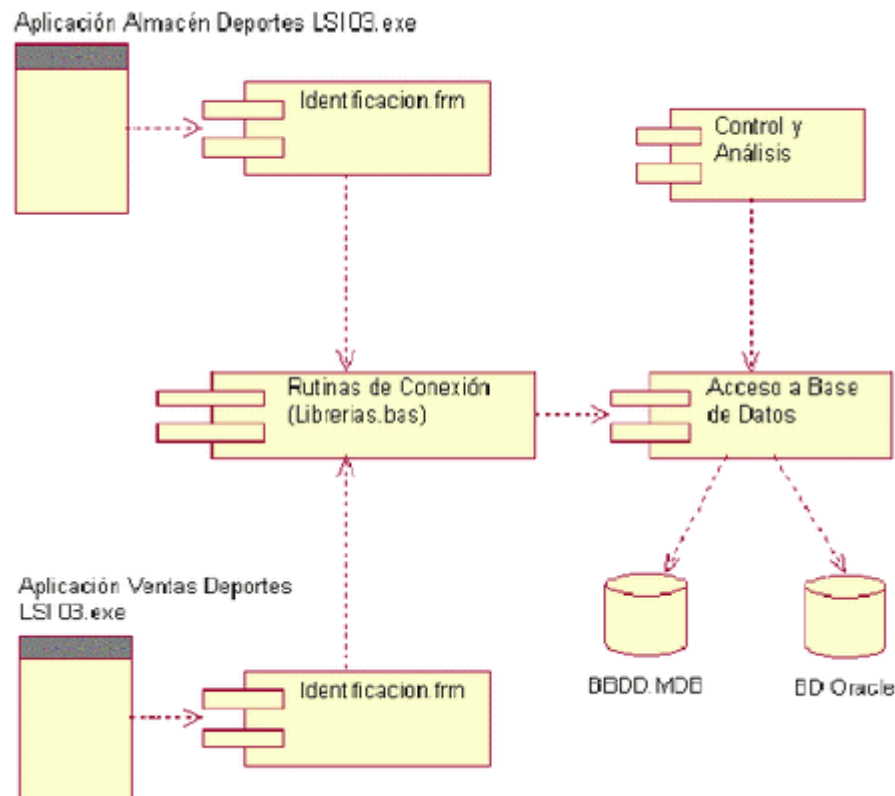


Diagrama de estructura compuesta

Este tipo de diagrama es nuevo en UML 2.0. Muestra la estructura interna de algún otro elemento, generalmente una clase, componente, caso de uso o comunicación, incluyendo sus relaciones con otras partes del sistema. Indica cómo se combinan los elementos del sistema para formar diseños más complejos.

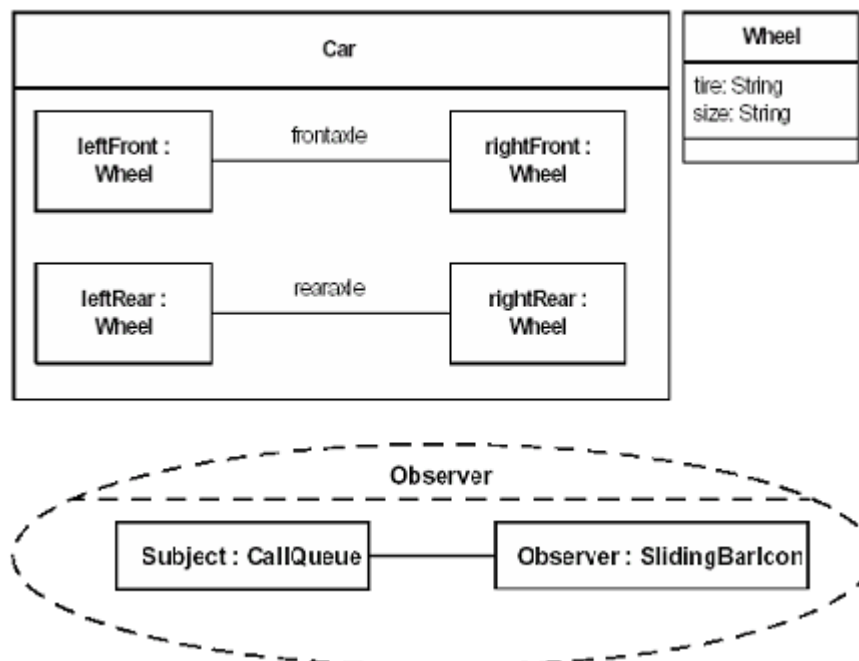


Diagrama de despliegue

Muestra como se despliega el sistema en un entorno específico y/o la estructura del entorno físico. Está formado por nodos (elementos físicos como equipos, impresoras, procesadores, elementos de red...), líneas de comunicación (que pueden ser cables, líneas inalámbricas, etc.) y componentes (que se instalan dentro de los nodos).

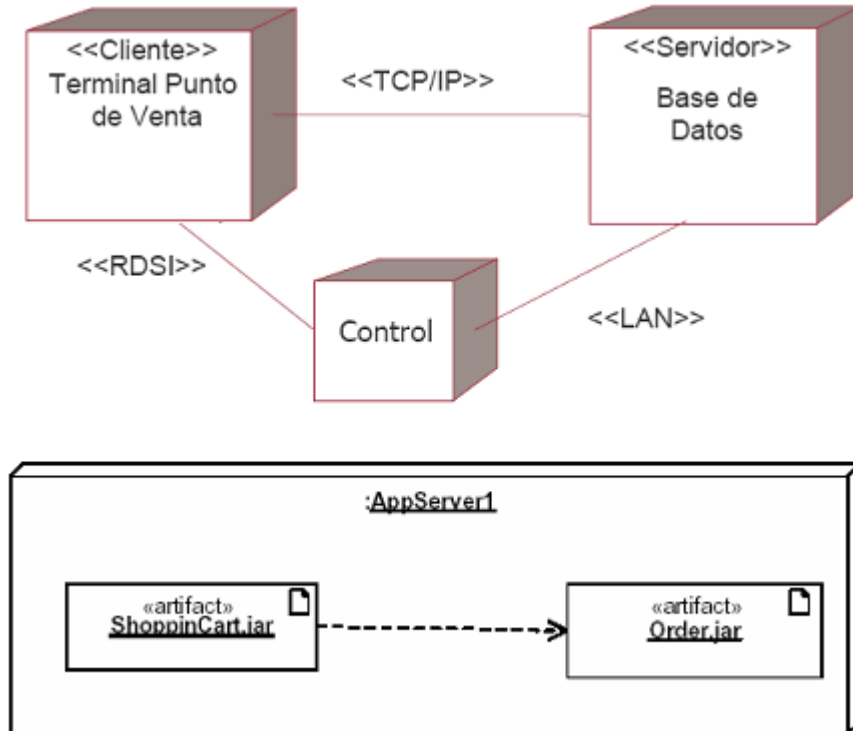
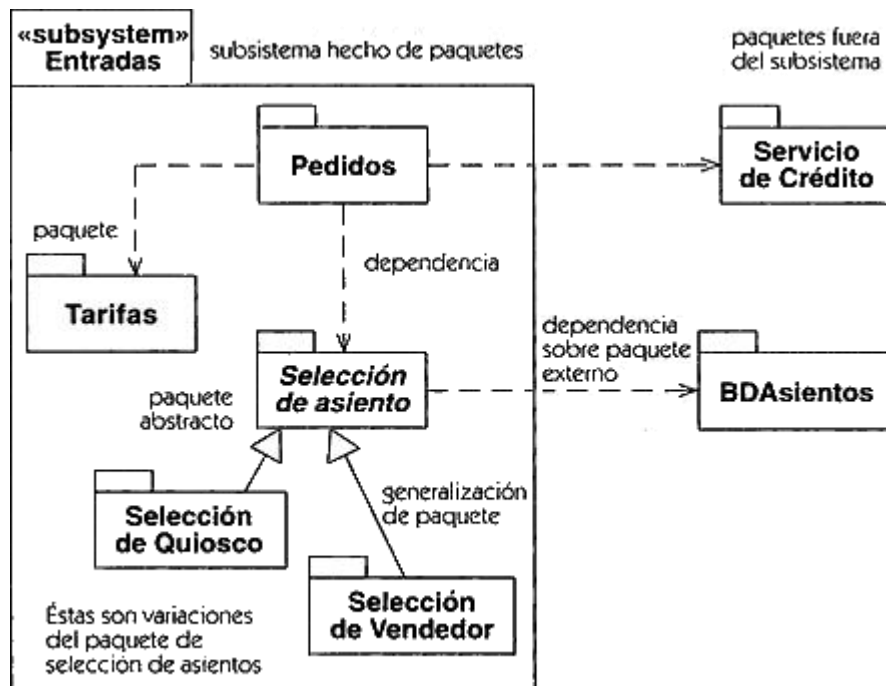


Diagrama de paquetes

Presenta cómo se organizan los elementos del modelo en paquetes y las dependencias entre ellos. Es en realidad un tipo especial de diagrama de clases pero se centra en cómo las clases se agrupan.



1.2.2 Diagramas de comportamiento

Diagrama de casos de uso

Captura los requisitos funcionales de un sistema. Proporciona una vista (independiente de la implementación) de las funcionalidades del sistema y permite centrarse en las necesidades del usuario en lugar de la implementación.

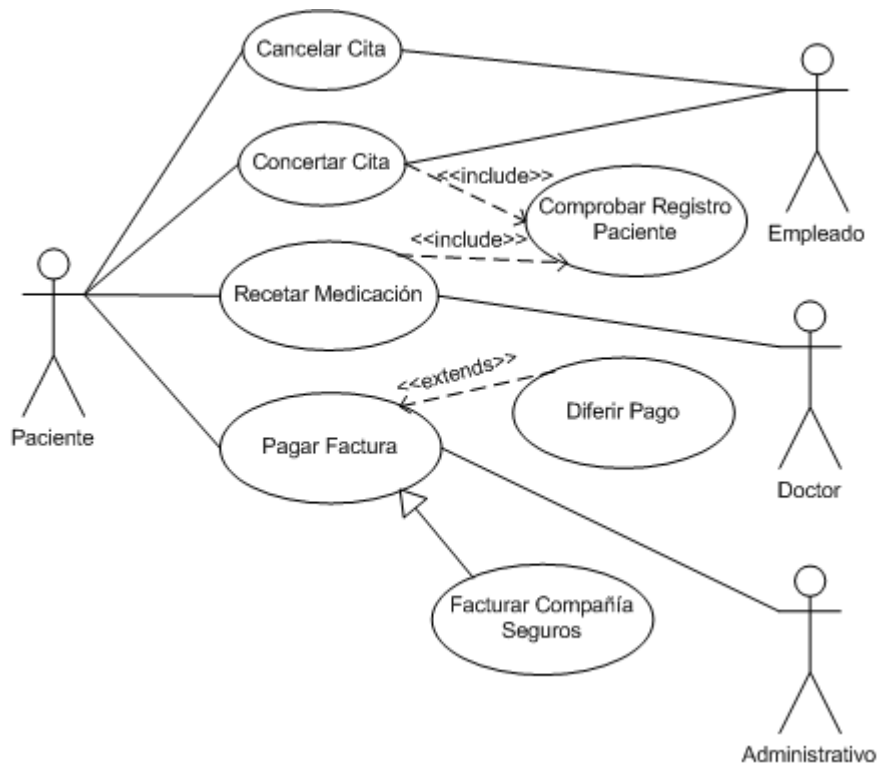


Diagrama de actividad

Muestra el flujo de trabajo que va de una actividad a otra. Es similar a los antiguos diagramas de flujo pero mucho más expresivo. Puede presentarse para modelar desde procesos de negocio de alto nivel (por ejemplo compras: desde el inicio de un pedido hasta que se cobra: presupuesto, pedido, albarán, pago, factura,...) hasta el funcionamiento interno de un método o procedimiento.

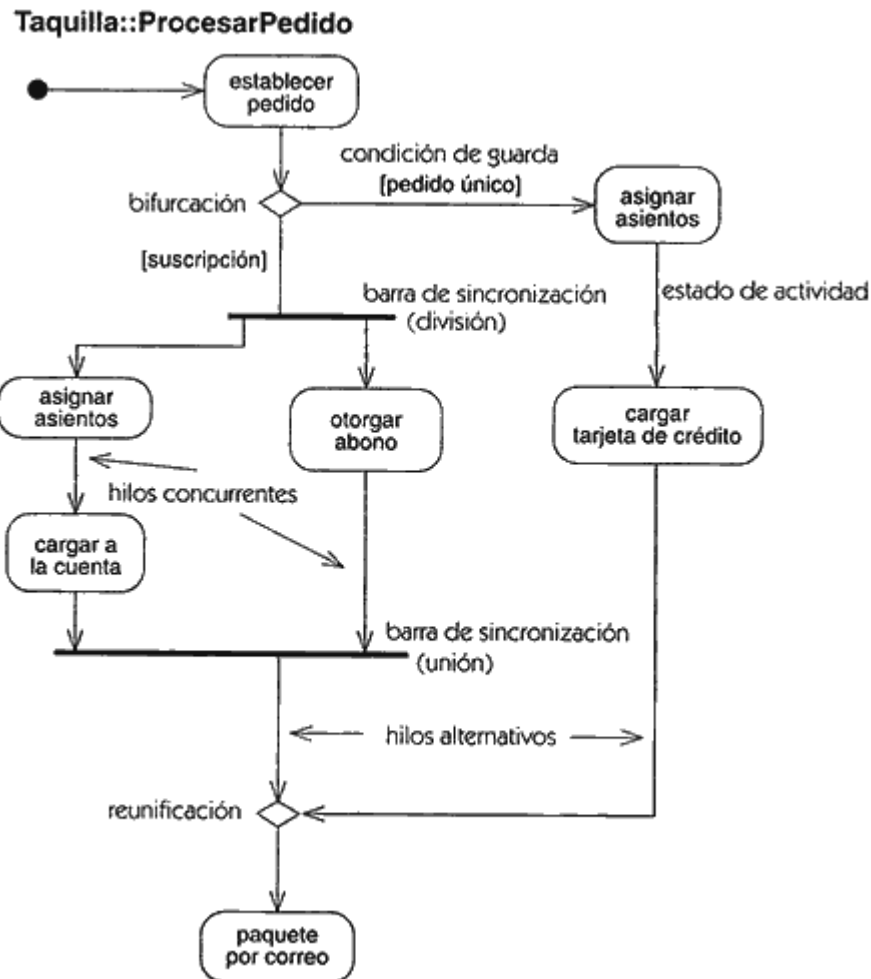


Diagrama de comunicación

Antes llamado diagrama de colaboración. Muestra el comportamiento dinámico de los objetos, al igual que los mensajes intercambiados entre ellos. Es equivalente al diagrama de secuencia. El de comunicación hace más énfasis en los objetos involucrados que en el orden y naturaleza de los mensajes.

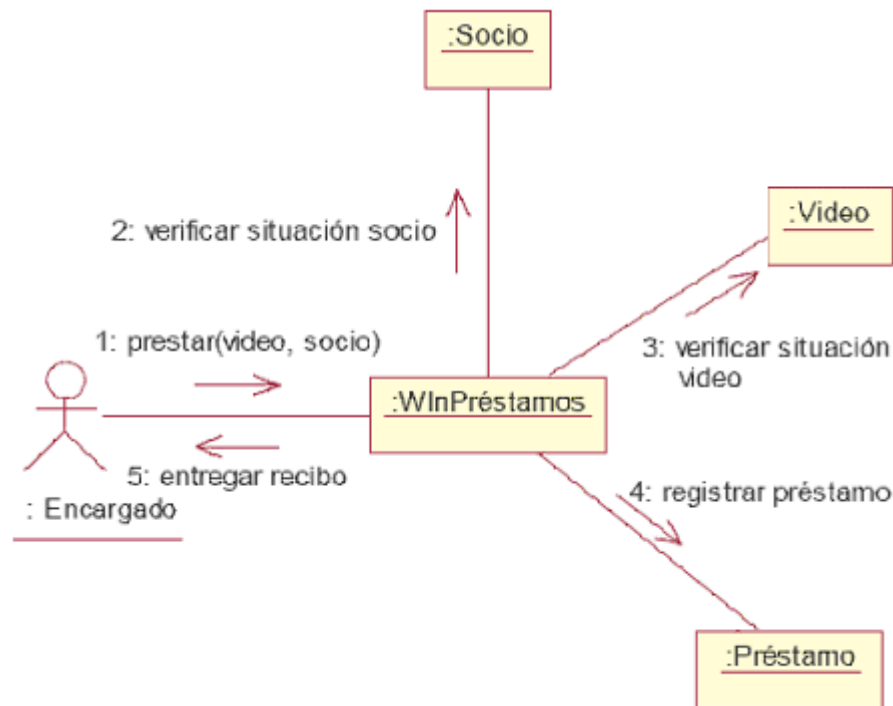


Diagrama de secuencia

Muestra el tipo y orden de los mensajes que son pasados entre elementos durante la ejecución. Es uno de los diagramas de interacción más común e intuitivo.

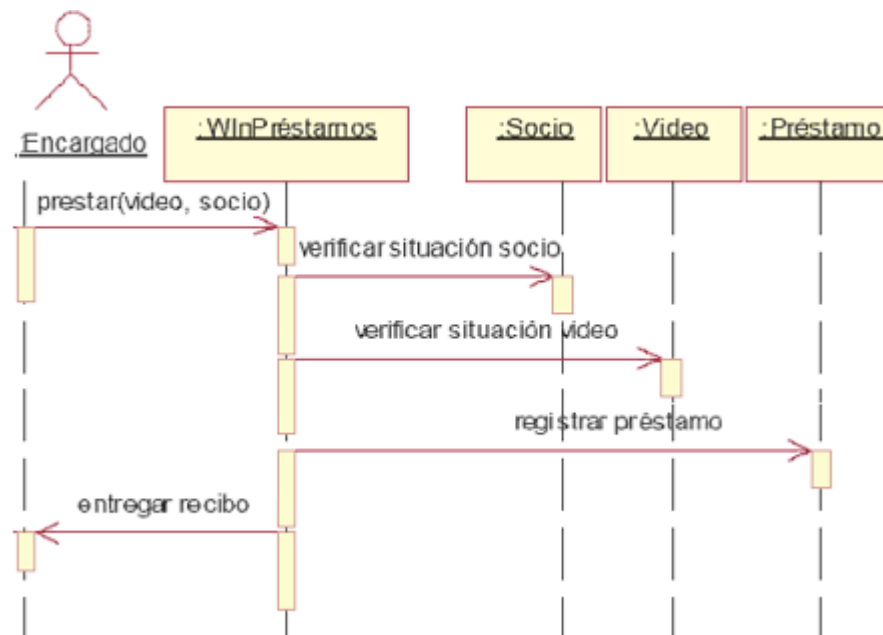


Diagrama general de interacción

Es una variante del diagrama de actividad. En lugar de hacer énfasis en cada actividad, el diagrama general hace énfasis en el elemento o elementos involucrados en realizar esa actividad. En el lugar de las actividades se puede mostrar cualquier tipo de diagrama de interacción que describa la actividad realizada y los elementos involucrados.

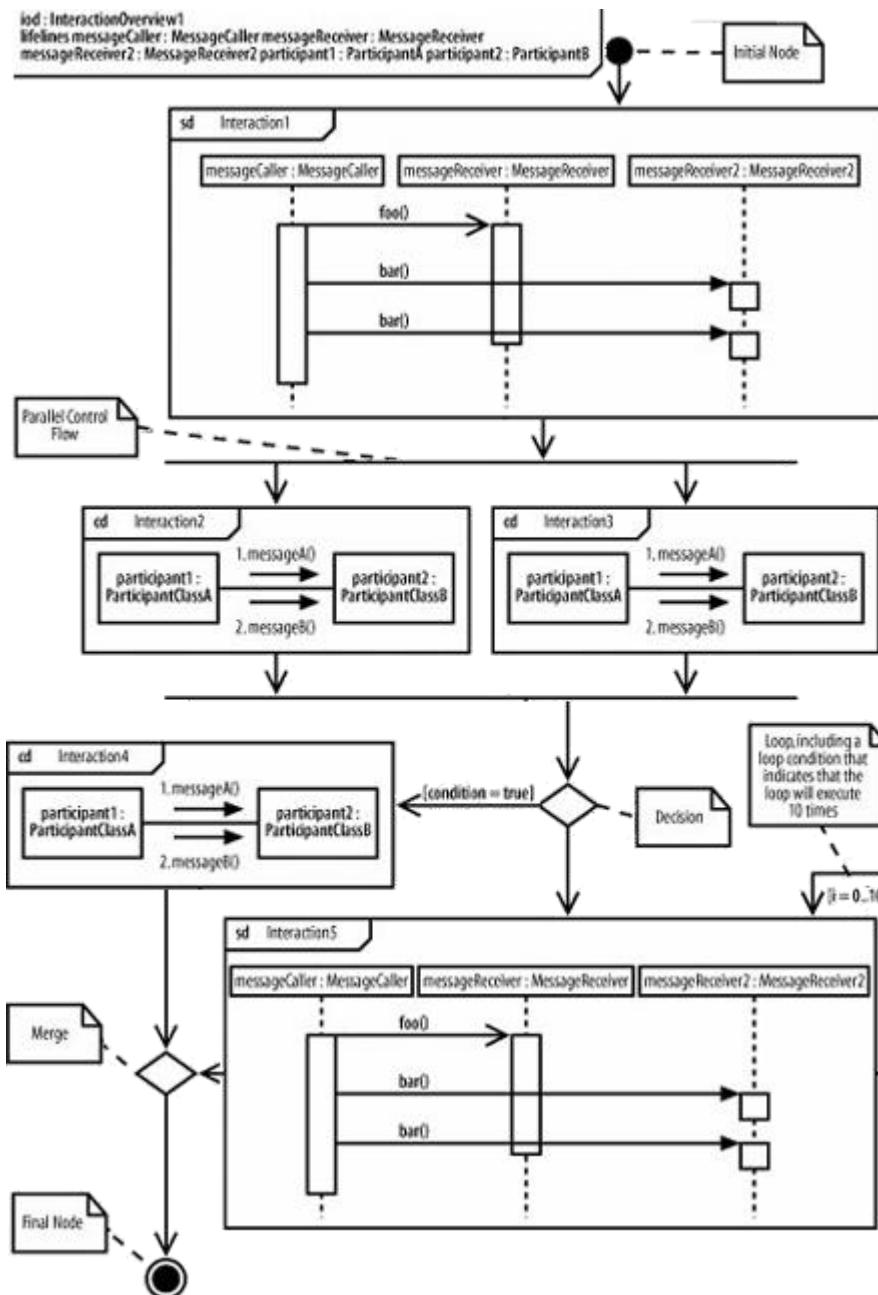


Diagrama general de interacción

Es una variante del diagrama de actividad. En lugar de hacer énfasis en cada actividad, el diagrama general hace énfasis en el elemento o elementos involucrados en realizar esa actividad. En el lugar de las actividades se puede mostrar cualquier tipo de diagrama de interacción que describa la actividad realizada y los elementos involucrados.

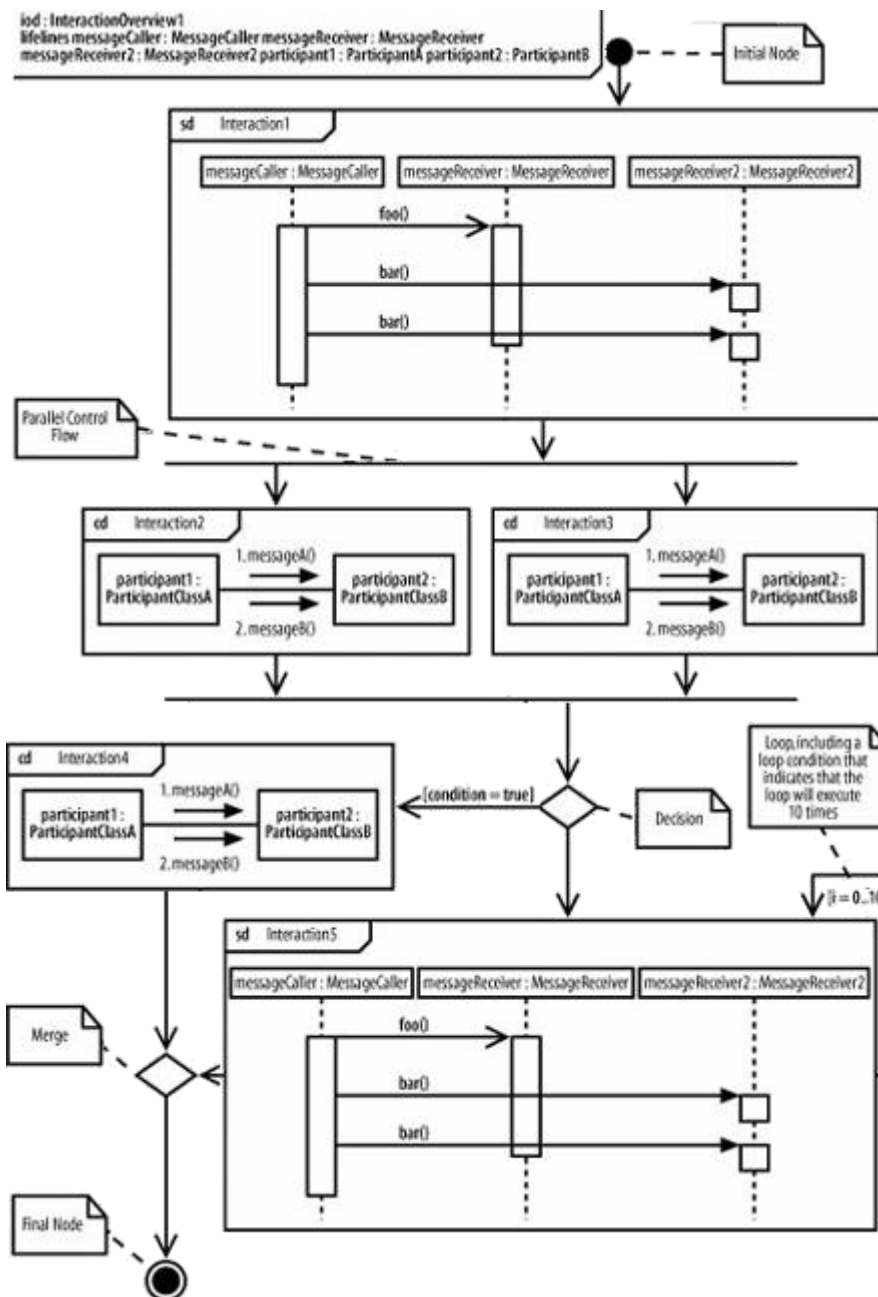


Diagrama de máquina de estados

Antes diagrama de estados. Muestra el comportamiento dinámico de un determinado elemento (puede ser desde una clase hasta el sistema completo), indicando los distintos estados por los que puede pasar y los eventos que permiten la transición entre estados. Es utilizado, entre otras cosas, en implementaciones de protocolos.

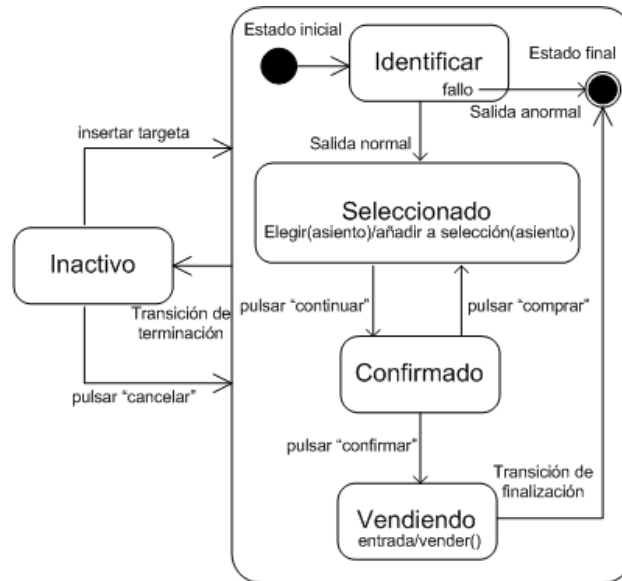
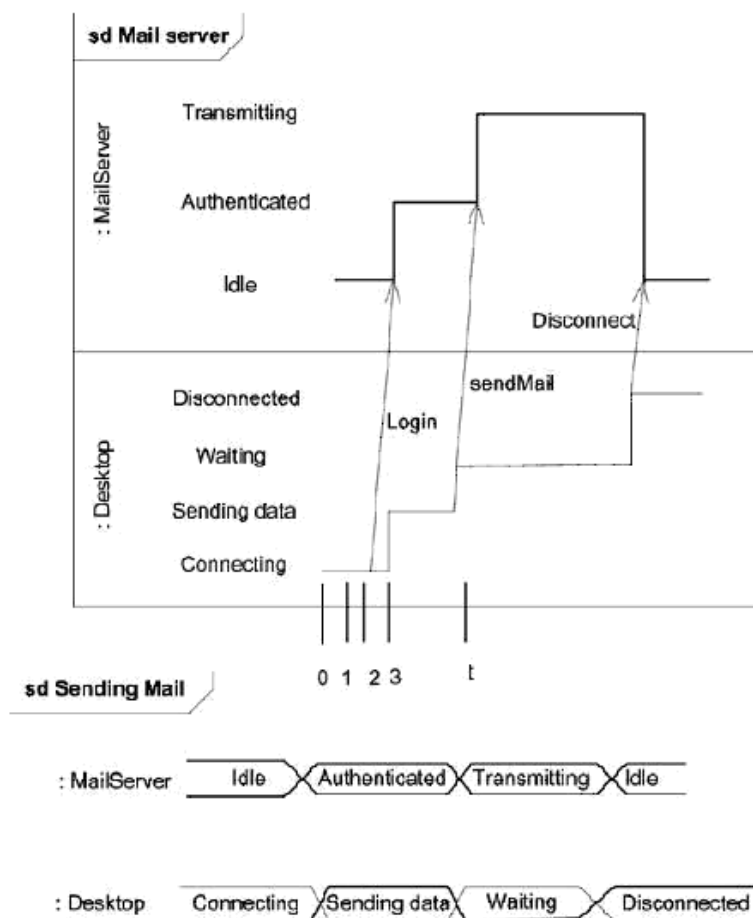


Diagrama de tiempos

Muestra especificaciones detalladas de tiempo producidas en el envío y recepción de mensajes. Es usado a menudo en sistemas de tiempo real, p. ej. comunicación con satélites. Tiene una notación específica que indica durante cuánto tiempo un sistema tiene que procesar o responder a mensajes y cómo responde a interrupciones externas.



1.3 PRINCIPIOS BÁSICOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

1.3.1 Introducción a la POO

UML es un lenguaje de modelado diseñado para construir sistemas orientados a objetos (aunque no es exclusivo para ellos), por ello sus elementos y diagramas están basados en el paradigma orientado a objetos.

Los principios sobre los que se fundamenta la **POO** (Programación Orientada a Objetos) pueden resumirse en:

- Se utilizan modelos basados en conceptos del mundo real para el desarrollo de aplicaciones.
- El software se organiza como una colección discreta de objetos a los que se asocian datos (atributos) y comportamiento (operaciones, procedimientos ó métodos).
- La comunicación de los objetos se realiza a través del paso de mensajes. Cuando un objeto es receptor del mensaje ejecuta una acción (operación, procedimiento ó método).

1.3.2 Objetos y Clases

Objetos

Un **objeto** representa una cosa que posee **identidad, estado y comportamiento**.

- **Identidad:** atributo o propiedad característica del objeto que le distingue de los demás.
- **Estado:** conjunto de valores concretos que caracterizan al objeto en un momento dado.
- **Comportamiento:** es la forma de actuación del objeto al recibir un mensaje o estímulo externo. También se puede definir como el conjunto de funciones que es capaz realizar el objeto

Clases

- Una clase representa un **conjunto de objetos** con propiedades similares (mismos estados y comportamientos), relaciones comunes con otros y una semántica común.
- Las clases son generalizaciones de objetos concretos que permiten agruparlos en categorías o familias.
- Cada objeto de una clase es un ejemplar o una **instancia** de la clase. Cuando creamos objetos concretos de una clase estamos realizando un proceso de instanciación.

1.3.3 Relaciones entre Clases

Asociación

Describe una relación genérica entre distintas clases. Una asociación describe un grupo de enlaces con estructura y semántica comunes.

Se entiende por enlace una conexión física o conceptual entre objetos.

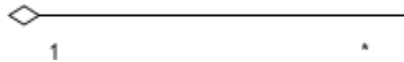
Por ejemplo, un enlace sería ‘Rosa trabaja en Telefónica’ y una asociación ‘Persona trabaja en Empresa’.



Agregación

Es una forma de asociación que **permite representar relaciones entre un todo y sus partes de forma que las partes pueden existir por sí mismas.**

Por ejemplo, ‘Departamento forma parte de Empresa’. En una agregación, una parte puede pertenecer a varios todos. Una relación de este tipo puede leerse como “...tiene un...”.



Composición

Representa una forma de agregación en la que **la existencia de las partes es totalmente dependiente del todo.**

Por ejemplo ‘Panel, Cabecera y Deslizador componen Ventana’. En una composición una parte sólo puede formar parte de un todo. Se puede leer como “...contiene a...”.



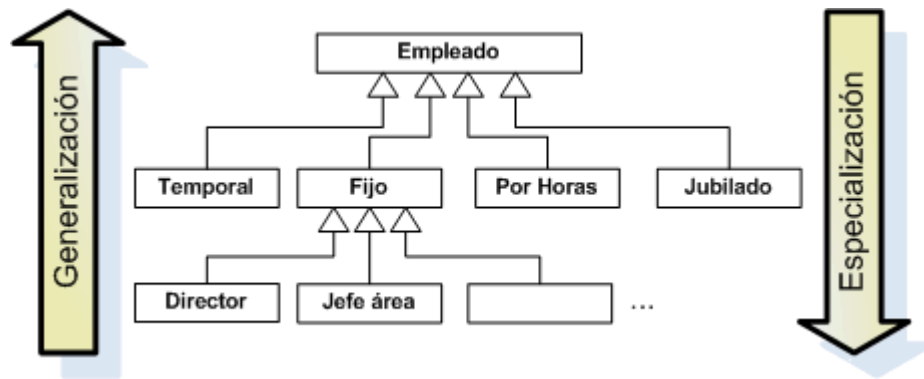
1.3.4 Jerarquías entre Clases

Generalización

Consiste en unir los elementos comunes de varias clases en una más general llamada **superclase**. Se leen normalmente como “...es un...”.

Especialización

Permite capturar las particularidades de un conjunto de objetos no discriminados por las clases ya identificadas en una subclase.

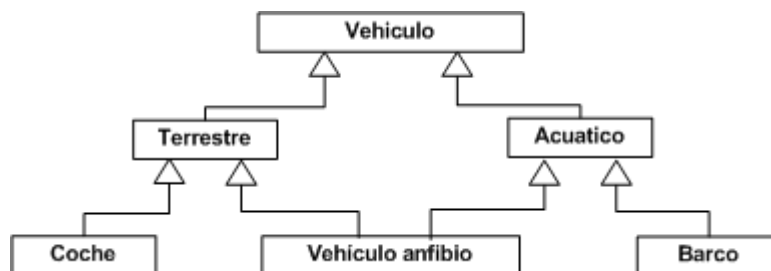


Herencia

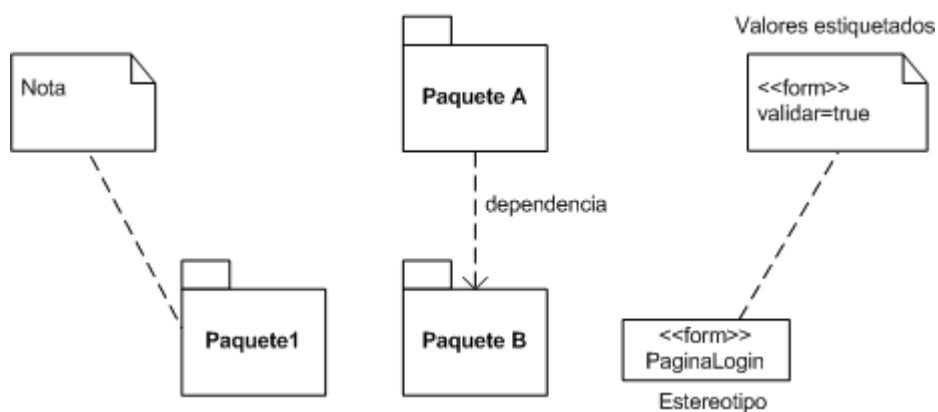
Es el mecanismo por el que las clases refinadas incorporan las características de una o más clases superiores (superclases) en la estructura jerárquica. Una subclase heredará atributos y operaciones de una superclase que está más arriba.

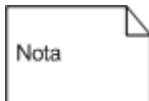
Hay dos tipos de herencia:

- Herencia simple: una clase sólo puede tener una superclase.
- Herencia múltiple: una clase puede tener más de una superclase y heredar características de todos sus antecesores



1.4 ELEMENTOS DE NOTACIÓN COMUNES





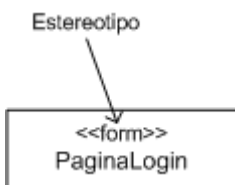
Nota: es un comentario textual que puede incluirse en un diagrama para ayudar a entender el significado de algún elemento. Se conecta al elemento mediante una línea discontinua.



Paquete: es un símbolo en forma de carpeta que representa una agrupación de elementos (p. ej. casos de uso, clases, componentes u otros paquetes).



Entre dos paquetes, o entre un elemento y un paquete, se puede establecer una **relación de dependencia** mediante una línea discontinua con flecha en un extremo, para indicar que algún elemento de un paquete, o un elemento de un diagrama, requiere de otro que pertenece a un paquete distinto.



Estereotipo: puede asociarse a cualquier elemento de los diagramas para diferenciar varias categorías del mismo tipo de elemento o, incluso, para definir nuevos elementos utilizando los símbolos estándar. Por ejemplo, en un diagrama de clases puede utilizarse un estereotipo asociado a las clases para diferenciar las que representan formularios (<<form>>) o módulos de clase (<<class module>>). Existen estereotipos definidos en el propio lenguaje y estereotipos definidos por el usuario.

1.5 CONCEPTOS BÁSICOS DE MODELADO

Un sistema modelado con UML consta de:

MODELOS

Cada uno **representa una forma de describir el sistema**. Ejemplos de modelos que describen un sistema son el modelo de casos de uso, el modelo de comportamiento o el modelo de despliegue.

DIAGRAMAS

Representación gráfica de una parte de un modelo, pero no necesariamente de todo el modelo.

Hay elementos de un modelo que no tienen por qué ser diagramas. Pueden ser descripciones textuales, pseudo-código, etc.

ELEMENTOS DE MODELADO

Son los **componentes de cada diagrama**. UML define qué elementos pueden usarse y cómo deben usarse.

1.5.1.1 Otros conceptos útiles en UML

- Un sistema se compone de uno o varios modelos.
- Un modelo se compone de uno o varios diagramas.
- Un diagrama se compone de uno o varios elementos de modelado
- UML sólo define cómo construir diagramas sintácticamente correctos. El resto del proceso (construir diagramas semánticamente correctos y construir modelos correctos) es cosa nuestra.

Clasificador (classifier)

Es el **elemento básico de modelado en UML**. Representa un grupo de elementos con comportamiento y propiedades comunes. Un clasificador se refiere al propio elemento UML, no a un sistema particular. Por ejemplo, una clase es un clasificador pero una clase concreta “Coche” sería una instancia de un clasificador.

Características

- Son clasificadores: casos de uso, clases, interfaces, componentes, subsistemas, estados, objetos, actividades, etc.
- Existen operaciones y elementos comunes que afectan a todos los clasificadores. P. ej. todos los clasificadores se pueden incluir dentro de paquetes.
- La notación estándar de un clasificador es un rectángulo con su nombre dentro (entre << >>) que puede dividirse en compartimentos para mostrar información específica (atributos, métodos...). Algunos tienen notaciones propias para poder distinguirlos visualmente.

Adornos (adornments)

Es un **tipo de información extra que se adjunta a un clasificador**. Pueden ser, por ejemplo, restricciones en los valores de una propiedad de un clasificador.

Características

- Dependen del tipo de elemento y diagrama.
- Suelen escribirse cerca del clasificador o en una nota adjunta.
- Por ejemplo. Plantillas o estereotipos son adornos aplicables a las clases.

2. CASOS DE USO

2.1 INTRODUCCIÓN

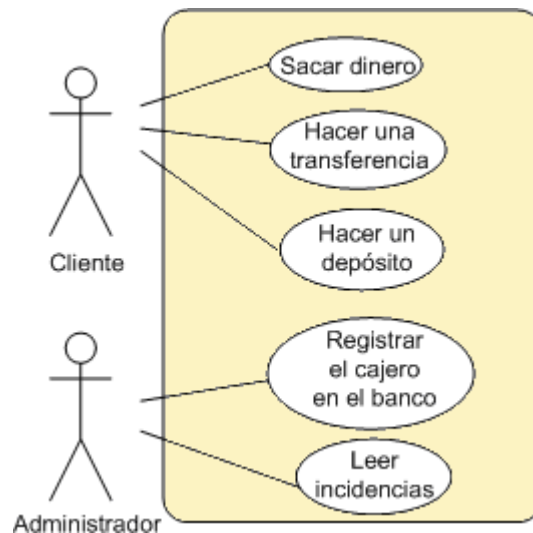
Un caso de uso es una **secuencia de transacciones** que son desarrolladas por un sistema en respuesta a un evento que inicia un actor sobre el propio sistema.

Los casos de uso son parte del análisis, de forma que nos ayudan a describir qué es lo que el sistema debe hacer. Describen un **uso del sistema** y **cómo este interactúa con el usuario**.

Un caso de uso muestra (con círculos y ‘monigotes’) de forma concisa y efectiva lo que hace un sistema de información.

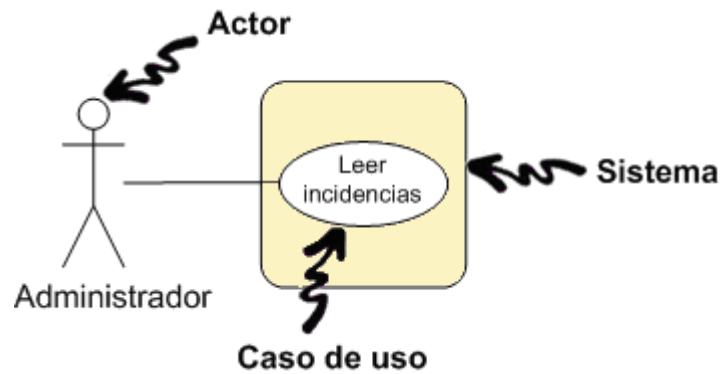
Un caso de uso se representa en el diagrama por una elipse, y denota un requerimiento solucionado por el sistema.

Cada caso de uso es una operación completa desarrollada por los actores y por el sistema en un diálogo, y va acompañado de un nombre significativo. El conjunto de casos de uso representa la totalidad de operaciones desarrolladas por el sistema.



2.2 CONCEPTOS BÁSICOS

Los componentes principales del diagrama de casos de uso son los **actores**, los **casos de uso** y el **sistema modelado**.



El **actor** es una entidad externa que tiene interés en interactuar con el sistema.

El **sistema** o sujeto se representa por un rectángulo y contiene los casos de uso.

Cada **caso de uso** especifica una unidad funcional completa (un proceso completo).

Los propósitos principales

Los **propósitos principales** de los casos de uso son:

- Decidir y describir los requisitos funcionales del sistema, como consecuencia de un acuerdo entre desarrolladores y usuarios.
- Dar una descripción clara y concisa de lo que el sistema debe hacer, que se usará durante todo el proceso de desarrollo.
- Facilitar la base para realizar las pruebas del sistema y validarlo.
- Proporcionar la capacidad para rastrear los requisitos funcionales hasta las clases y operaciones del sistema que los implementan. Esto simplifica los cambios y ampliaciones del sistema.

El proceso

El **proceso** para crear el **modelo de casos de uso** es:

1. Definir el sistema.
2. Identificar a los actores y a los casos de uso.
3. Identificar las relaciones entre actores y casos de uso.
4. Completar y validar el modelo.

El modelo de casos de uso se desarrolla mediante un proceso altamente iterativo que debe incluir al cliente y a representantes de los distintos actores (usuarios finales entre ellos).

El modelo de casos de uso se compone de una serie de Diagramas de casos de uso.

Los casos de uso de cada diagrama deben describirse (p. ej. con texto plano).

Las personas interesadas

Las personas que tienen interés en los modelos de casos de uso son:

- Cliente y usuarios finales.
- Desarrolladores.
- Jefes y gestores del proyecto.
- Testeadores (personal de pruebas).
- Cualquier otro involucrado en la funcionalidad del sistema: documentación, soporte, mantenimiento, calidad, ventas, marketing...

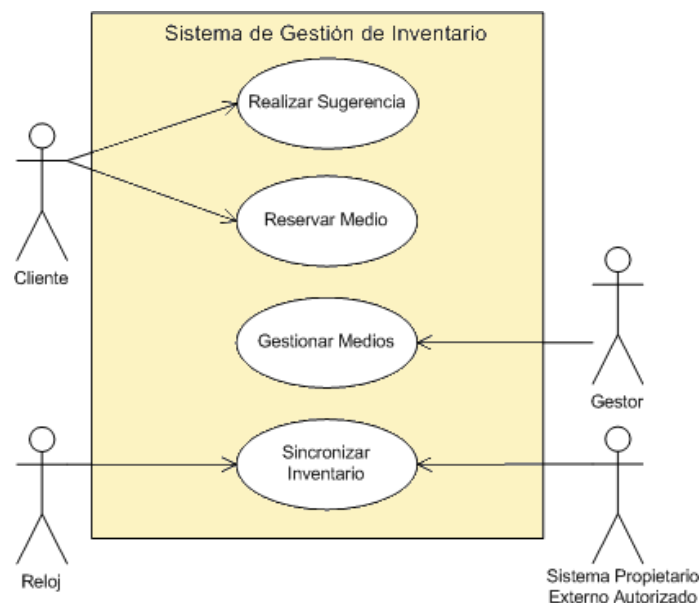
Desarrollo de nuevas versiones

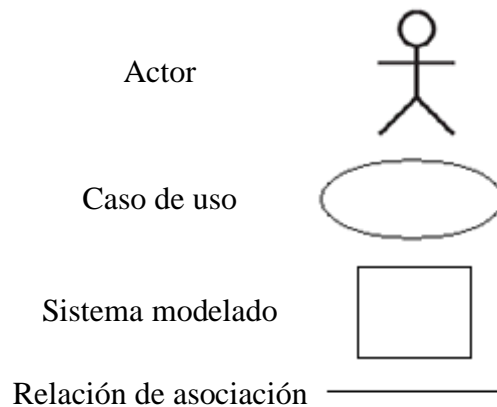
Para el desarrollo de nuevas versiones o generaciones de un sistema, también se emplea el modelado de casos de uso.

2.3 DIAGRAMAS DE CASOS DE USO

El Modelo de casos de uso se describe con una serie de **Diagramas de casos de uso**.

Un Diagrama de casos de uso representa la forma en como un Cliente (Actor) opera con el sistema en desarrollo, además de la forma, tipo y orden en como los elementos interactúan (operaciones o casos de uso) Un diagrama de casos de uso consta de los siguientes **elementos**.





El **Actor** es una entidad externa que tiene interés en interactuar con el sistema.

Cada **caso de uso** representa un conjunto de acciones llevadas a cabo por un sistema que dan lugar a un resultado observable.

El **Sistema** o **Sujeto** es una ‘caja negra’ que provee casos de uso y delimita la frontera del sistema o subsistema que se representa.

La **relación (de asociación)** representa una asociación entre un actor y un caso de uso con el que dicho actor interactúa.


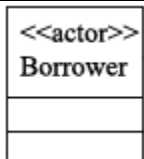

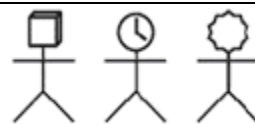
- Como parte del modelado de casos de uso deben definirse los **límites del sistema** (o sujeto) a desarrollar.
- La responsabilidad global del sistema no es fácil de definir.
- Otra consideración es la amplitud del sistema en la 1ª iteración.
- Es esencial capturar durante esta fase un catálogo de conceptos relacionados con el dominio del sistema, como la terminología (glosario) que posteriormente describe los casos de uso.
- En un diagrama el sistema se dibuja como una caja con el nombre del mismo en la parte superior de la misma.
- La caja contendrá en su interior los casos de uso.
- En ocasiones no se muestra el sistema en el diagrama

2.4 ACTORES

- Un **actor** representa un **rol** que juega un usuario u otro sistema (ordenador o dispositivo) que interactúa con el sistema que se representa.
- Debe de tener un nombre representativo del rol.
- El actor se comunica con el caso de uso enviando y recibiendo mensajes.
- Los casos de usos son siempre iniciados por actores mediante un estímulo.

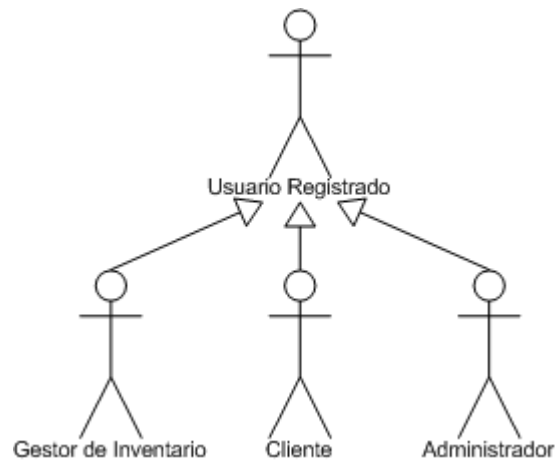
- Pueden ser activos (inician la interacción con el sistema) o pasivos (reciben los resultados de las operaciones llevadas a cabo en el sistema).
- Para **identificar a los actores** hay que:
 1. Establecer quien o qué está interesado en usar el sistema.
 2. Ponerse en el lugar de los actores e identificar los casos de uso.
 3. Realizar preguntas del tipo: ¿Quién usa la funcionalidad principal del sistema?, ¿Quién necesita que el sistema le ayude en sus tareas diarias?, ¿Quién mantiene o administra el sistema?...

2.4.1 Representación en UML

Icono estándar: Monigote (‘stickman’ en inglés) con el nombre debajo. Esta representación sirve tanto para actores que son personas como para otro tipo de actores	
Ajustándonos estrictamente al estándar un caso de uso es un “Classifier” con el estereotipo <<actor>> y su nombre asociado.	 
Representaciones alternativas (clarifican pero no añaden semántica)	 <p>Los actores representan, en este orden:</p> <ol style="list-style-type: none"> 1. Se usa para representar otros sistemas. 2. Se usa para representar relojes y elementos de sincronización externos. 3. Se usa para representar dispositivos (PDAs, sensores, etc.).
Relación con casos de uso: siempre debe estar asociado (relación de asociación unidireccional o bidireccional) con un caso de uso (directa o indirectamente)	

2.4.2 Relaciones entre actores

- Los actores pueden tener relaciones de generalización (**herencia**) entre ellos con la semántica habitual de este tipo de relación (el hijo hace lo mismo que el padre y algo más).
- Puede mejorar la entendibilidad del sistema. No debe usarse cuando el sistema se comprenda mejor usando roles individuales.



2.5 CASOS DE USO



Es una **operación/tarea** específica que se realiza tras una orden de algún agente externo, sea desde una petición de un actor o bien desde la invocación desde otro caso de uso.

Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica.



Las acciones involucran comunicación con actores (usuarios o sistema) y/o cálculos internos.

Las **características** de un caso de uso son:

- siempre es iniciado por un actor.
- proporciona valor a un actor.
- es completo.

Casos de Uso en UML

Se representan en el Diagrama de casos de uso mediante una elipse.	
El nombre se sitúa dentro o debajo de la elipse y debe reflejar la tarea específica que el actor desea llevar a cabo usando el sistema.	

Generalmente es colocado dentro de los límites de un sistema.	
Está conectado con uno o más actores mediante una asociación , (incluyendo el actor que inicia el caso de uso y otros).	
Técnicamente un caso de uso es un 'classifier'. Cada instancia del caso de uso se denomina 'escenario'.	

Para **identificar a los casos de uso** hay que:

- Comenzar con los actores previamente definidos.
- Realizar preguntas del tipo: ¿Qué funciones requiere cada actor del sistema? ¿Qué necesita?, ¿Tiene el actor que ser notificado sobre algo o es el actor quien notifica algo al sistema?, etc.

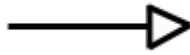
2.6 RELACIONES ENTRE CASOS DE USO

La razón para utilizar estos casos de uso no completos en algunos casos, es mejorar la comunicación en el equipo de desarrollo, el manejo de la documentación de casos de uso. En el caso de que queramos utilizar estos casos de uso más pequeños, las relaciones entre estos y los casos de uso ordinarios pueden ser de los siguientes tres tipos:

- **Generalización** (*generalization*): es una relación que amplía la funcionalidad de un caso de uso o refina su funcionalidad original mediante el agregado de nuevas operaciones y/o atributos y/o secuencia de acciones.
- **Extensión** (*extend*): es una relación que amplía la funcionalidad de un caso de uso mediante la extensión de sus secuencias de acciones. El comportamiento de un caso de uso es ampliado por otro.
- **Inclusión** (*include*): es una relación mediante la cual se re-usa un caso de uso encapsulado en distintos contextos a través de su invocación desde otros casos de uso. El comportamiento de un caso de uso es incorporado (tal cual esté) en otro caso de uso.

El objetivo de usar estas relaciones es **clarificar**. No deben usarse si complican el diagrama o si quien visualice los diagramas no las entiende

2.6.1 Relación de Generalización



Es una relación entre un actor (caso de uso) más general y otro más específico.

Se debe verificar la propiedad de sustituibilidad, es decir, se puede usar una instancia de la especialización siempre que se espere una instancia de la generalización.

Se representa por una línea continua entre los dos casos de uso, con el triángulo que simboliza generalización en UML (usado también para denotar la herencia entre clases) pegado al extremo del caso de uso más general.

El caso de uso hijo **hereda** las asociaciones y características del caso de uso padre. El caso de uso padre se trata de un caso de uso abstracto, que no está definido completamente.

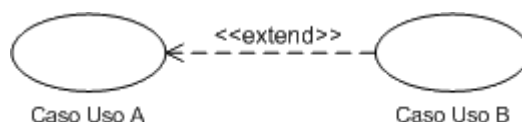
2.6.2 Relación de Extensión



Representa que un caso de uso puede extender el comportamiento de otro caso de uso.

Añade comportamiento a un caso de uso, completo en sí mismo, sin cambiar el caso de uso original.

Estereotipo <<extend>>, <<extends>>, <<extiende>>. El estándar de UML recomienda utilizar el primero.



El 'caso de uso A' se llama caso de uso **base**.

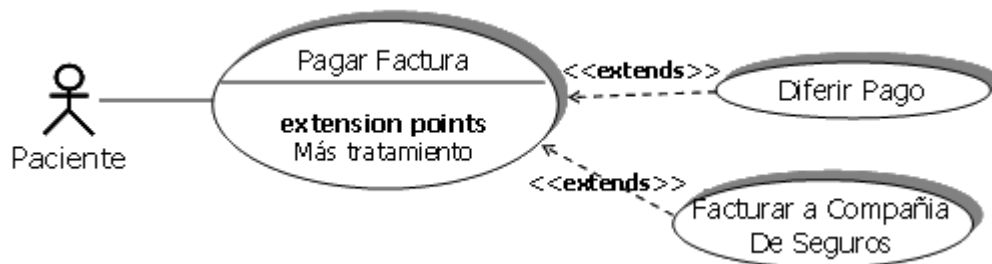
Se dice que 'caso de uso B', aumenta y/o **extiende la funcionalidad** de 'caso de uso A'. Hace lo mismo y algo más.

Es una relación de dependencia dirigida (representada por una línea de puntos discontinua).

Una notación alternativa es definir los puntos de extensión en el caso de uso Base

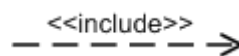
Ejemplo

El caso de uso que extiende describe un comportamiento opcional del sistema (a diferencia de la relación incluye que se da siempre que se realiza la interacción descrita).



Al activar el caso de uso **Pagar factura** opcionalmente se podrán realizar los otros dos casos de uso: **Diferir Pago** y **Facturar a Compañía de Seguros**.

2.6.3 Relación de Inclusión

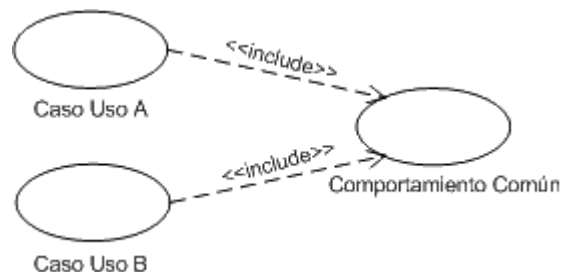


Representa que un caso de uso contiene el comportamiento definido en otro caso de uso.

Añade comportamiento a otro caso de uso, que no está completo en sí mismo, cambiándolo.

Esteriotipo <<include>>, <<includes>>, <<uses>>. El estándar UML 2.0 recomienda utilizar <<include>> siempre.

Cuando varios casos de uso tienen una parte de comportamiento común esta puede extraerse para ser incluida por el resto de casos de usos.



‘caso de uso A’ y ‘caso de uso B’ incorporan el comportamiento **completo** del caso de uso incluido durante su ejecución.

Los casos de uso incluidos (y el resto también) deben representar una funcionalidad completa.

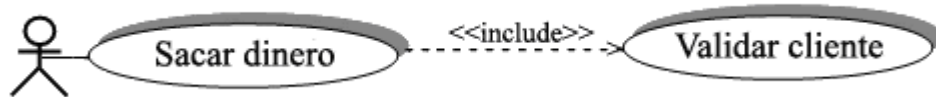
Si no es así, es que estoy descomponiendo el caso de uso base, por lo que hay que replantearse el caso de uso.

Si la funcionalidad extraída no se incluye en más de un caso de uso, hay que replantearse la descomposición.

A veces se entienden mejor funcionalidades completas.

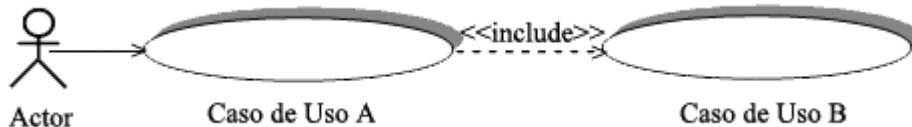
Fomenta la reutilización. Y el modelado orientado a la reutilización.

Ejemplo

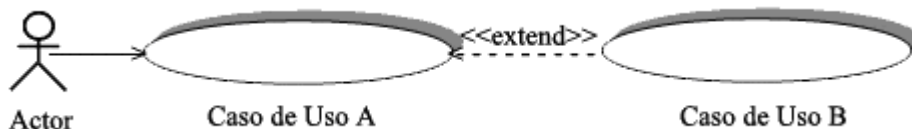


El caso de uso Sacar dinero **incluye** el comportamiento del caso de uso Validar cliente.

2.6.4 Comentarios



Si el actor activa el 'caso de uso A' siempre se ejecutará B



Si se activa el 'caso de uso A' opcionalmente se ejecutará B
Debe tener cuidado con la dirección de las flechas.

¿Cómo identificar las relaciones?

- Debe de hacerse tras identificar los actores y los casos de uso.
- Debemos realizarnos preguntas del tipo: ¿Cómo se comunican con el sistema los actores involucrados en un determinado caso de uso?, ¿Una serie de casos de uso contienen un flujo común que puede describirse mejor con una relación de inclusión?, ¿Quedan actores o casos de uso sin relación?, etc.

Los dos tipos de relación anteriores (inclusión y extensión) están orientados exclusivamente para casos de uso (y no para actores).

En una relación <<extend>>, un actor que lleve a cabo el caso de uso base puede realizar o no sus extensiones. Mientras, en una relación <<include>> el actor que realiza el caso de uso base también realiza el caso de uso incluido.

En general utilizaremos <<extend>> cuando se presenta una variación del comportamiento normal, y <<include>> cuando se repite un comportamiento en dos casos de uso y queremos evitar dicha repetición.

Ambos tipos de relación se representan como una **dependencia etiquetada** con el estereotipo correspondiente (<< >> ó <>). La flecha indica el sentido en el que debe leerse la etiqueta. Junto a la etiqueta <> se puede detallar el/los puntos de extensión del caso de uso base en los que se aplica la extensión.

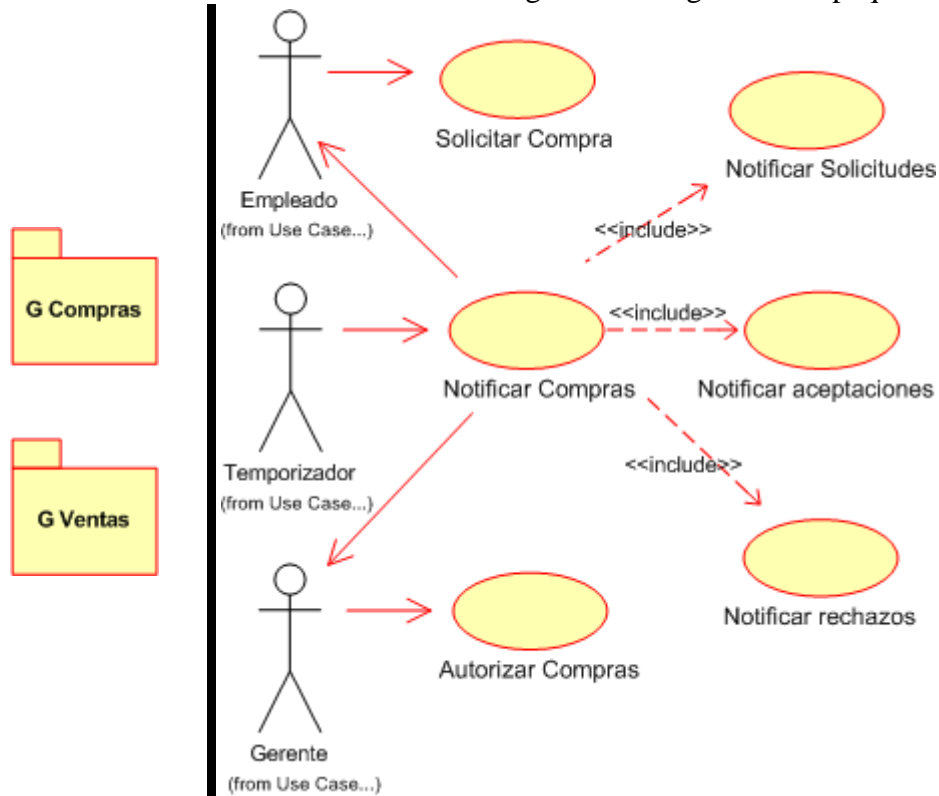
2.7 ORGANIZACIÓN DE LOS CASOS DE USO Y DIAGRAMAS

Cuando el sistema es grande un solo Diagrama de casos de uso no puede describirlo, por lo que hay que hacer varios diagramas.

Se pueden organizar:

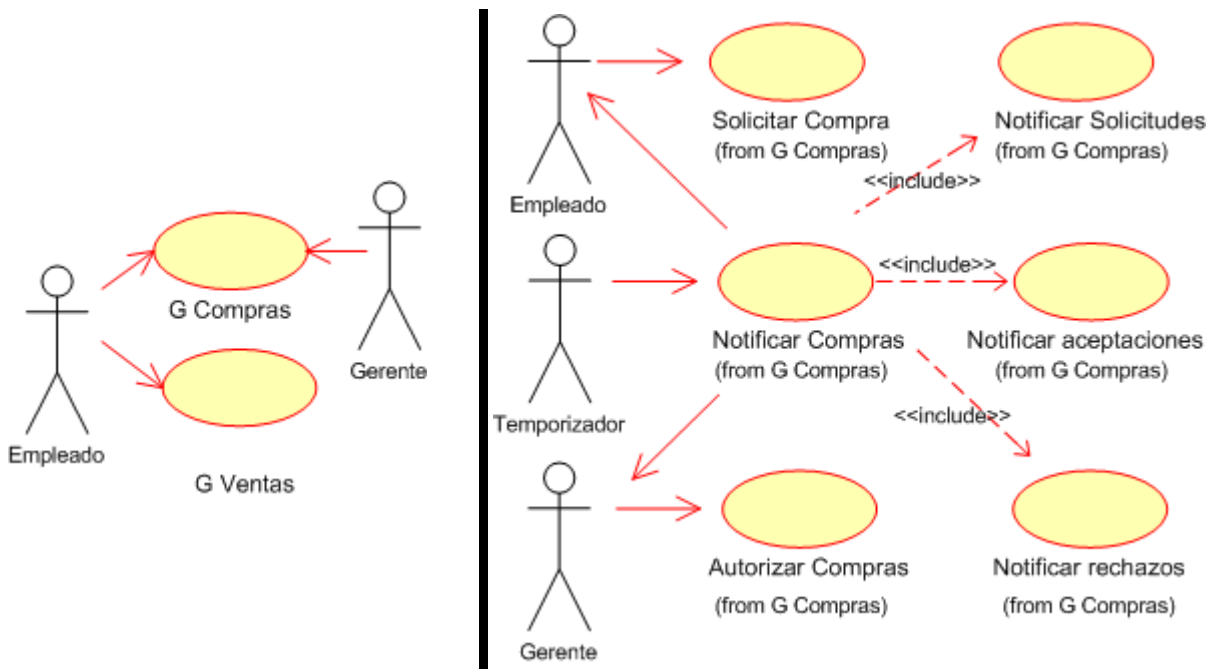
- **Mediante paquetes**

Tanto los casos de uso como los Diagramas se organizan en paquetes.



- **Mediante sub-diagramas de casos de uso**

Cada caso de uso complejo se amplía mediante un diagrama que lo describa. Los sub-diagramas son “contenidos” por los casos de uso a los cuales describen.



- Las dos cosas

Los casos de uso se pueden dividir:

- Subsistemas o partes del sistema principal
- Desde el punto de vista de la funcionalidad del usuario

2.8 COMPLETANDO EL MODELADO

Los casos de uso no son la ‘solución’ que recogen todos los requisitos de los usuarios.

Existen **otras técnicas de captura de requisitos**, otros requisitos y otras fuentes de requisitos que no podrán ser capturados con casos de uso:

- Rendimiento, seguridad.
- Restricciones impuestas por el cliente o algún sistema.
- Interfaz de usuario (walkthrough, click-through).
- Storyboards.

2.8.1 Descripción de casos de uso

Generalmente se hace mediante **texto**.

Recordar que hay que concentrarse en el **comportamiento**.

Debe incluir:

- Objetivo del caso de uso.

- Condiciones de inicio del caso de uso ¿Cómo y quién lo arranca?.
- Flujo de mensajes entre los actores y el caso de uso.
- Flujos alternativos del caso de uso.
- Requisitos especiales o suplementarios aplicables a este caso de uso.
- Finalización del caso de uso y valor aportado al actor/es.

La descripción debe ser **clara** y **consistente** para que el cliente pueda entenderla y validarla.

Un caso de uso puede describirse con un Diagrama de Estructura Compuesta o un Diagrama de Actividades.

- El objetivo sigue siendo la **claridad** para que lo entienda el cliente, por lo tanto estos diagramas es mejor que complementen y no que sustituyan al modelo.

La descripción se puede complementar con un par de “escenarios”. Ejemplos o supuestos particulares de realización del caso de uso.

2.8.2 Validación de casos de uso

Hay que asegurarse de que el sistema bajo desarrollo cubre las necesidades del cliente.

El modelo de caso de uso sirve para validar el sistema en etapas tempranas del desarrollo.

Los usuarios deben validarlo en un proceso iterativo.

Si el modelo de caso de uso se hace mal y no se valida adecuadamente, hay un problema: estamos desarrollando un sistema que los usuarios no necesitan.

2.8.3 Realización de casos de uso

Para realizar un caso de uso se debe:

- **Conectar** los casos de uso con los objetos del sistema (clases) que finalmente implementan la lógica.
- **Definir** que elementos del modelo lógico (clases) van a ser los responsables de implementar cada caso de uso y cómo lo van a hacer. (por mensajes).
- **Explicar** que una colaboración requiere una descripción (se pueden emplear diagramas y/o texto).

Los **diagramas** que se pueden emplear son.

- Interacción (comunicación, secuencia y revisión interacción).
- Máquina de estados.
- Actividad.

(El caso de uso puede ‘poseer’ estos diagramas)

Realizar el caso de uso requiere que el diseñador transforme los diferentes pasos que lleve a cabo el caso de uso en operaciones (llamadas a **métodos**) entre las clases que colaboran en la realización del caso de uso.

Esto se consigue con sentido común, experiencia y trabajo.

No hay **ningún método establecido**.

2.8.4 Testeo de casos de uso

Hay que confirmar que el sistema está implementado correctamente de acuerdo a los requerimientos especificados.

Sólo se puede hacer cuando haya artefactos ejecutables intermedios (prototipos) o al final del desarrollo.

Los casos de uso también ayudan a definir los casos de prueba.

3. DIAGRAMAS DE CLASES

3.1 INTRODUCCIÓN

El diagrama de clases es el **diagrama** principal de **diseño** y **análisis** para un sistema.

En él se crea el diseño conceptual de la información que se manejará en el sistema, los componentes que se encargarán del funcionamiento y la relación entre uno y otro.

El objetivo principal de este modelo es la representación de **los aspectos estáticos** del sistema, utilizando diversos mecanismos de abstracción (clasificación, generalización, agregación).

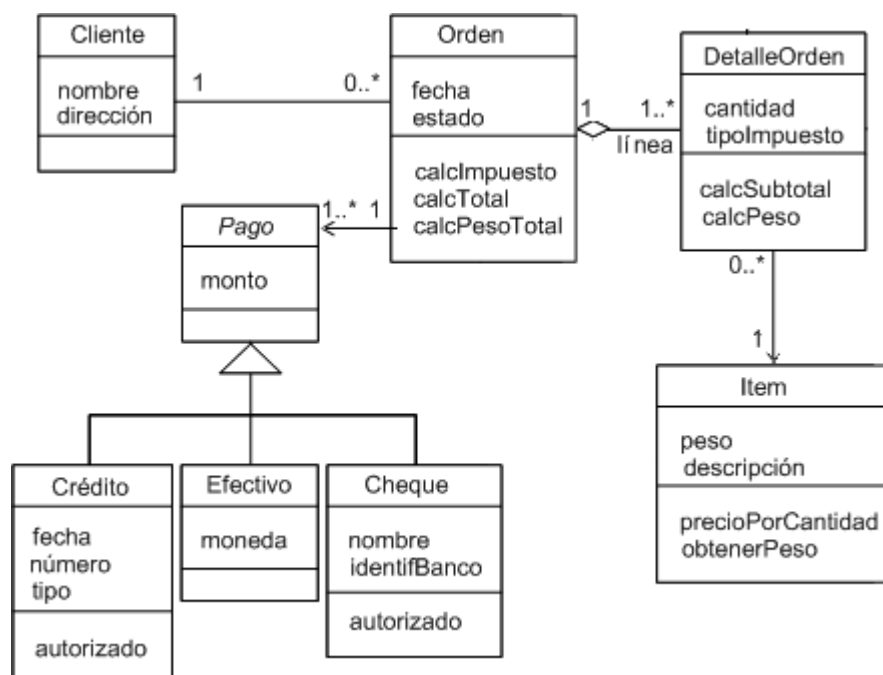
Es decir, es donde daremos rienda suelta a nuestros conocimientos de diseño orientado a objetos, definiendo las clases e implementando las relaciones de herencia y agregación.

En el diagrama de clases será donde definiremos las características de cada una de las clases, interfaces, colaboraciones y relaciones de dependencia y generalización.

Se representa la estructura de cada uno de los objetos del sistema y sus relaciones con los demás objetos, pero no muestra información temporal.

Para facilitar la comprensión del diagrama, se pueden **incluir paquetes** como elementos del mismo, donde cada uno de ellos agrupa un conjunto de clases.

Ejemplo



3.2 CLASES Y RELACIONES ENTRE CLASES

3.2.1 Clases

Una **clase** describe un conjunto de objetos con propiedades (atributos) similares y un comportamiento común.

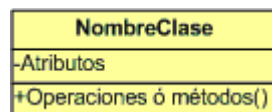
Los objetos son instancias de las clases.

Las clases suelen corresponderse con sustantivos que hacen referencia al ámbito del sistema de información y que se encuentran en los documentos de las especificaciones de requisitos y los casos de uso.

Representación de una clase

Una clase está representada por un rectángulo que dispone de tres apartados:

El primero para indicar el **nombre**, el segundo para los **atributos** y el tercero para los **métodos**



Cada clase debe tener un **nombre** único, que la diferencie de las otras.

Los **atributos** representan los datos asociados a los objetos instanciados por esa clase. Identifican las características propias de cada clase.

Los atributos pueden representarse solo mostrando su nombre, mostrando su nombre y su tipo, e incluso su valor por defecto.

Las **operaciones** o **métodos** representan las funciones o procesos propios de los objetos de una clase, caracterizando a dichos objetos.

Los atributos pueden representarse solo mostrando su nombre, mostrando su nombre y su tipo, e incluso su valor por defecto.

Clases abstractas

El diagrama de clases permite representar **clases abstractas**.

Una clase abstracta es una clase que no puede existir en la realidad, pero que es útil conceptualmente para el diseño del modelo orientado a objetos.

Las clases abstractas no son instanciables directamente, sino a través de sus descendientes.

Una clase abstracta suele ser situada en la jerarquía de clases en una posición que le permita ser un depósito de métodos y atributos para ser compartidos o heredados por las subclases de nivel inferior.

Estereotipo

Las clases y en general todos los elementos de los diagramas, pueden estar clasificados de acuerdo a varios criterios, como por ejemplo su objetivo dentro de un programa.

Esta clasificación adicional se expresa mediante un *estereotipo*.

3.2.2 Relaciones entre clases

Asociación

Las relaciones de asociación representan un conjunto de enlaces entre objetos o instancias de clases.

Es el tipo de relación más general, y denota básicamente una **dependencia semántica**.

Cada asociación puede presentar elementos adicionales que doten de mayor detalle al tipo de relación:

- **Rol y/o nombre** de la asociación: describe la semántica de la relación en el sentido indicado.
Por ejemplo, la asociación entre Persona y Empresa recibe el nombre de ‘trabaja para’.
- **Multiplicidad**: describe la **cardinalidad** de la relación, es decir, especifica cuántas instancias de una clase están asociadas a una instancia de la otra clase.

Cardinalidad

Indica el grado y nivel de dependencia en una relación.

Se anota en cada extremo de la relación.

Pueden ser:

- Cero a muchos: 0..* (0..n).
- Uno a muchos: 1..* (1..n).
- Uno a uno: 1..1.
- Muchos a muchos: n..m.
- Número fijo: m (m denota el número).



Un cliente puede tener asociadas muchas Órdenes de Compra (0..*), en cambio una orden de compra solo puede tener asociado un cliente.

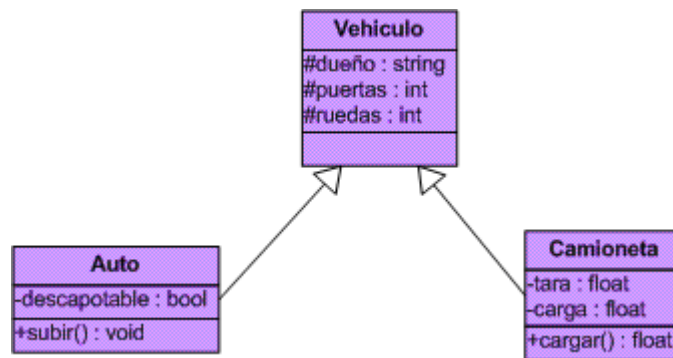
Herencia

Las jerarquías de **generalización/especialización** se conocen como herencia.

Permite a una clase de objetos incorporar atributos y métodos de otra clase, añadiéndolos a los que ya posee.

La clase de la cual se hereda se denomina **superclase**, y la que hereda **subclase**.

Con la herencia se refleja una relación “**es_un**” entre clases:



Auto y Camioneta heredan de Vehículo, es decir, Auto posee las características de Vehículo y además posee algo particular: atributo Descapotable. En cambio Camioneta también hereda las características de Vehículo pero posee como particularidad propia Tara y Carga.

Agregación

La agregación es un tipo de relación jerárquica entre un objeto que representa la totalidad de ese objeto y las partes que lo componen.

Permite el agrupamiento físico de estructuras relacionadas lógicamente.

Los objetos “**son-parte-de**” otro objeto completo.

Por ejemplo, motor, ruedas, carrocería son parte de automóvil.

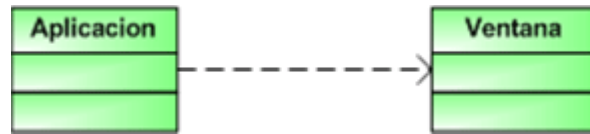
Composición

Es una forma de agregación donde la relación de propiedad es más fuerte, e incluso coinciden los tiempos de vida del objeto completo y las partes que lo componen.

Por ejemplo, una ventana o formulario de una aplicación contiene una serie de controles (etiquetas, campos de texto, etc...), siendo esta relación una composición.

Dependencia

Una relación de dependencia se utiliza entre dos clases o entre una clase y una interfaz, e indica que una clase requiere de otra para proporcionar alguno de sus servicios (una clase usa a otra).



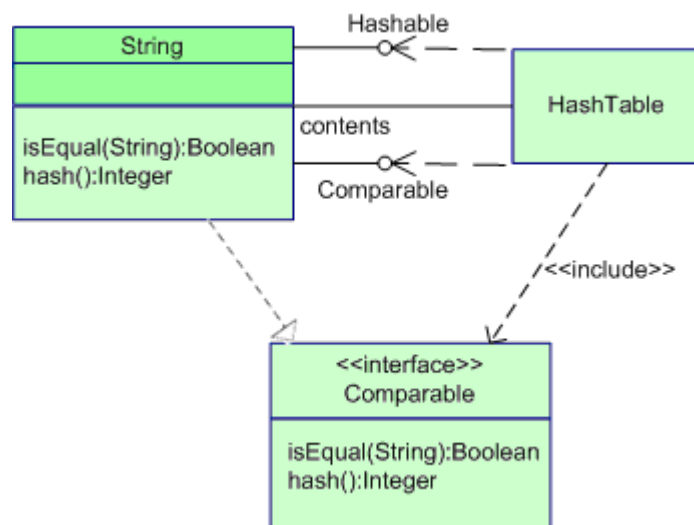
Por ejemplo una aplicación grafica que instancia una ventana (la creación del Objeto Ventana está condicionado a la instanciación proveniente desde el objeto Aplicacion).

Cabe destacar que el objeto creado (en este caso la Ventana gráfica) no se almacena dentro del objeto que lo crea (en este caso la Aplicación).

3.2.3 Interfaces y paquetes

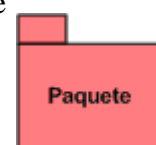
Interfaz

- Una **interfaz** es una especificación de la semántica de un conjunto de operaciones de una clase o paquete que son visibles desde otras clases o paquetes.
- Cada interfaz especifica a menudo sólo una parte limitada del comportamiento de una clase real.
- Una interfaz no tiene implementación.
- Normalmente, se corresponde con una parte del comportamiento del elemento que la proporciona.



Paquetes

- Los **paquetes** se usan para dividir el modelo de clases del sistema de información, agrupando clases u otros paquetes según los criterios que sean oportunos.
- Las dependencias entre ellos se definen a partir de las relaciones establecidas entre los distintos elementos que se agrupan en estos



paquetes.

3.3 NOTACIÓN

3.3.1 Notación de las clases

Una **clase** se representa como una caja, separada en **tres zonas** por líneas horizontales:

Nombre de la clase

- En la **zona superior** se muestra el **nombre de la clase** y propiedades generales como el **estereotipo**.
- El nombre de la clase aparece centrado y si la clase es abstracta se representa en cursiva.
- El estereotipo, si se muestra, se sitúa sobre el nombre y entre el símbolo: << >>.

Atributos

- La **zona central** contiene una lista de **atributos**, uno en cada línea.
- La notación utilizada para representarlos incluye, dependiendo del detalle, el **nombre** del atributo, su **tipo** y su **valor** por defecto, con el formato:

visibilidad / nombre : tipo[multiplicidad] = valor-inicial { propiedades }.

Operaciones

- En la **zona inferior** se incluye una lista con las **operaciones** que proporciona la clase.
- Cada operación aparece en una línea con el formato:
visibilidad nombre (lista-de-parámetros): tipo-devuelto { propiedades }.
- La lista de parámetros es una lista con los parámetros recibidos en la operación separados por comas.
El formato de un parámetro es:
 dirección nombre : tipo[multiplicidad] = valor-por-defecto { propiedades }
 Dirección: in, out, inout

Los atributos y métodos de clase se escriben subrayados.

Las clases y métodos abstractos se indican con la propiedad {abstract} y/o *en cursiva*.

3.3.2 Notación de las relaciones de clases

Una relación puede tener un **nombre** y un **estereotipo**, que se colocan junto a la línea.

- El **nombre** suele corresponderse con expresiones verbales presentes en las especificaciones, y define la semántica de la asociación.
- Los **estereotipos** permiten clasificar las relaciones en familias y se escribirán entre el símbolo: << ... >>.

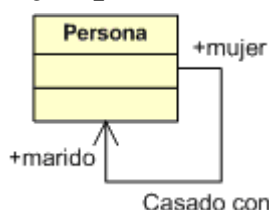
Las diferentes propiedades de la relación se pueden representar con la siguiente notación.

- **Nombre de la asociación:** es el nombre para identificarla. Generalmente es un verbo, aunque también se permite un nombre.
- **Multiplicidad:** puede ser un número concreto, un rango o una colección de números. La letra 'n' y el símbolo '*' representan cualquier cantidad. (1, 0..1, 0..*, *, 1..*). Desde UML 2.0 se admite cualquier número (p. ej. 1..4).
- **Orden:** Se puede especificar si las instancias guardan un orden con la palabra clave '{ordered}'. Si el modelo es suficientemente detallado, se puede incluir una restricción que indique el criterio de ordenación.
- **Navegabilidad:** La navegación desde una clase a la otra se representa poniendo una flecha sin relleno en el extremo de la línea, indicando el sentido de la navegación.
- **Rol:** se coloca junto al extremo de la línea que esta unida a una clase, para expresar cómo esa clase hace uso de la otra clase con la que mantiene la asociación. Puede haber dos roles en una relación.

Ejemplo 1



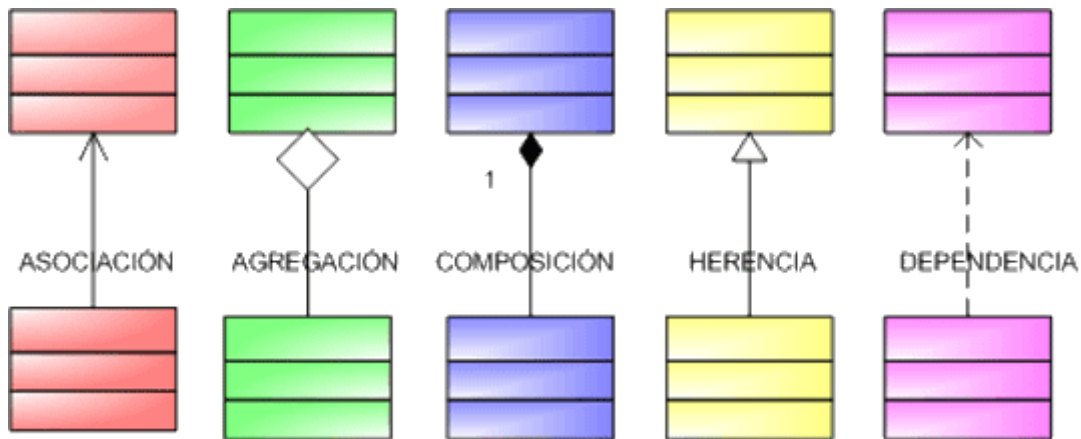
Ejemplo 2



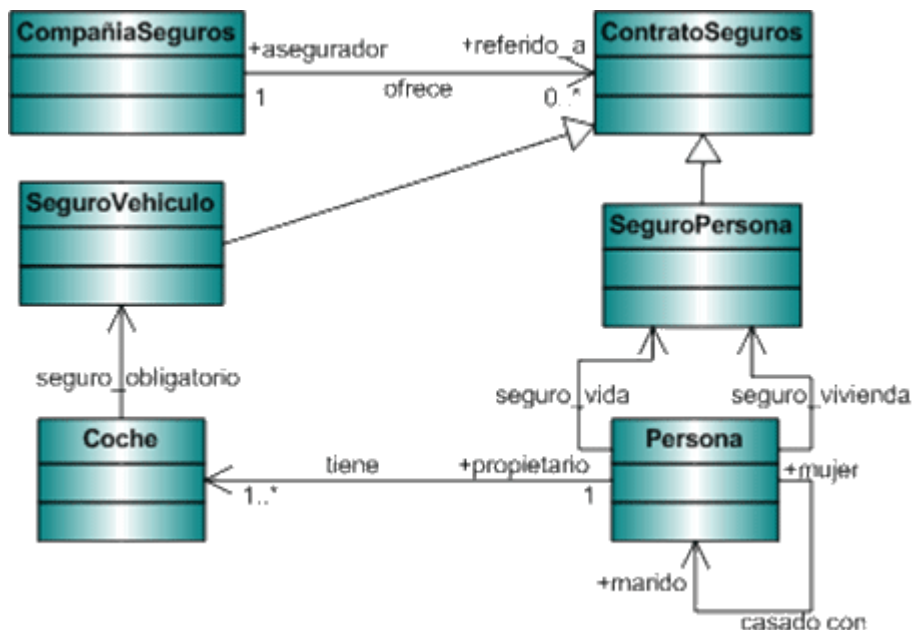
Existen **notaciones específicas** para los otros tipos de relación.

- **Asociación:** Se representa como una línea continua entre las clases asociadas. En una relación de asociación, ambos extremos de la línea pueden conectar con la misma clase, indicando que una instancia de una clase, está asociada a otras instancias de la misma clase, lo que se conoce como asociación reflexiva.
- **Agregación:** Se representa con un rombo hueco en la clase cuya instancia es una agregación de las instancias de la otra.
- **Composición:** Se representa con un rombo lleno en la clase cuya instancia contiene las instancias de la otra clase.
- **Herencia:** Esta relación se representa como una línea continua con una flecha hueca en el extremo que apunta a la superclase.
- **Dependencia:** Se representa con una línea discontinua con una flecha apuntando a la clase cliente. La relación puede tener un estereotipo que se coloca junto a la línea, y entre el símbolo: << ... >>.

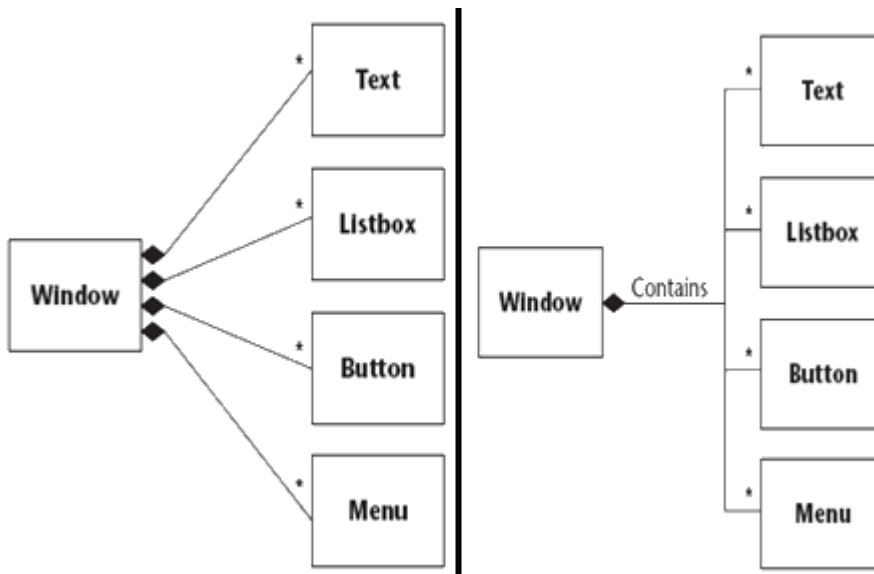
Representaciones



Ejemplo



Composiciones



Se pueden implementar

- Igual que las asociaciones.
- Colocando las partes como atributos del todo.

Estereotipos

Estereotipos estándar para las relaciones de dependencia

- **<<uses>>**. Una clase requiere a otra clase. (En Java debe haber un import).
- **<<permit>>**. Una clase requiere el permiso o la habilitación de otra. (En C++ se puede usar para los accesos 'friend').
- **<<refine>>**. Para mostrar una relación entre dos elementos a diferentes niveles. Una clase (o conjunto de clases) refina o realiza otra mejorando su funcionalidad, comportamiento, etc ...
- **<<trace>>**. Dos elementos representan el mismo concepto en diferentes modelos.

Realización

- Es un tipo especial de dependencia.
- Es usada para indicar que un elemento completa, implementa o realiza a otro más general.
- Generalmente es usada con interfaces (implements en Java).



3.3.3 Notación de las interfaces y los paquetes

Notación de las interfaces

Una interfaz se representa como una **caja con compartimentos**, igual que las clases.

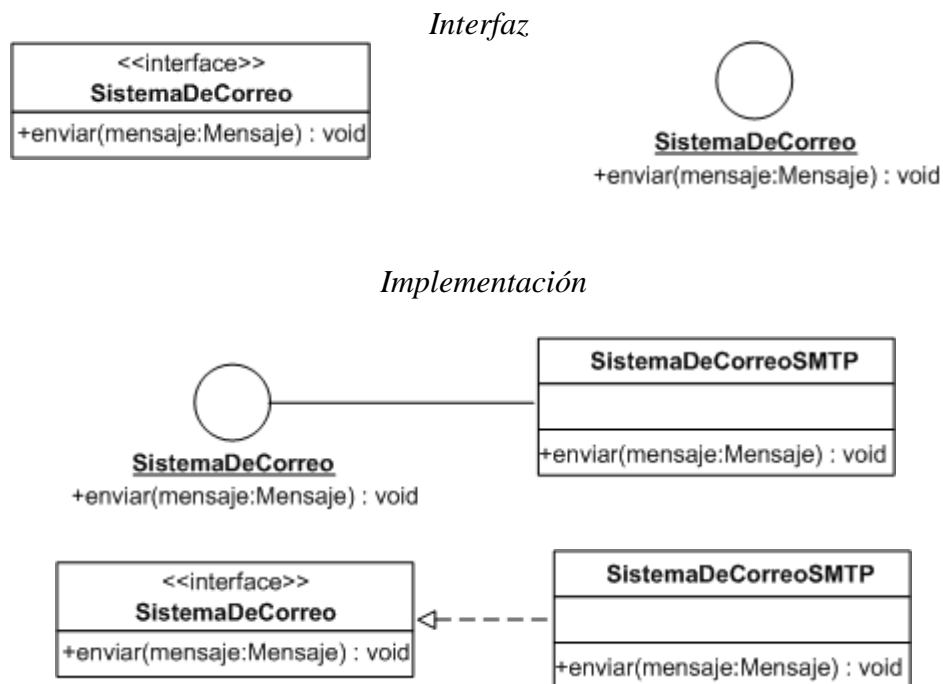
- En la zona superior se incluye el nombre y el estereotipo <<interface>>.
- En la zona inferior se coloca la lista de operaciones, como en las representaciones de clases.
- La zona en la que se listan los atributos estará vacía o puede omitirse.

Otra representación más simple para la interfaz es **un círculo**.

Entre una clase que implementa las operaciones que una interfaz ofrece y esa interfaz, se establece una relación de **realización**.

Dependiendo de la notación elegida, se representará con una línea continua entre ellas cuando la interfaz se representa como un círculo y con una flecha hueca discontinua apuntando a la interfaz cuando se represente como una clase.

Ejemplos



Notación de los paquetes

Los paquetes se representan mediante un icono con forma de carpeta y las dependencias con flechas discontinuas entre los paquetes dependientes.



3.4 DIAGRAMAS DE OBJETOS

Los **diagramas de objetos** se pueden considerar un caso especial de un diagrama de clases en el que se muestran instancias específicas de clases (objetos) en un instante particular durante la ejecución del sistema.

Emplean la misma notación y relaciones que el diagrama de clases.

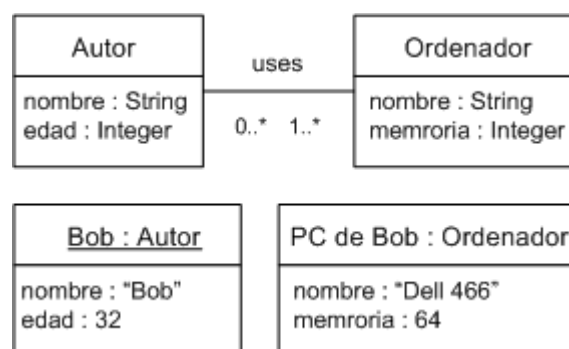
Muestra instancias específicas de clases y los enlaces entre ellas en un determinado momento de tiempo.

Es usado para

- Ilustrar cómo un diagrama de clases complejo puede instanciarse en objetos.
- Mostrar como los objetos de un diagrama de clases se combinan entre ellos en un determinado momento de tiempo.

Notación de los diagramas de objetos

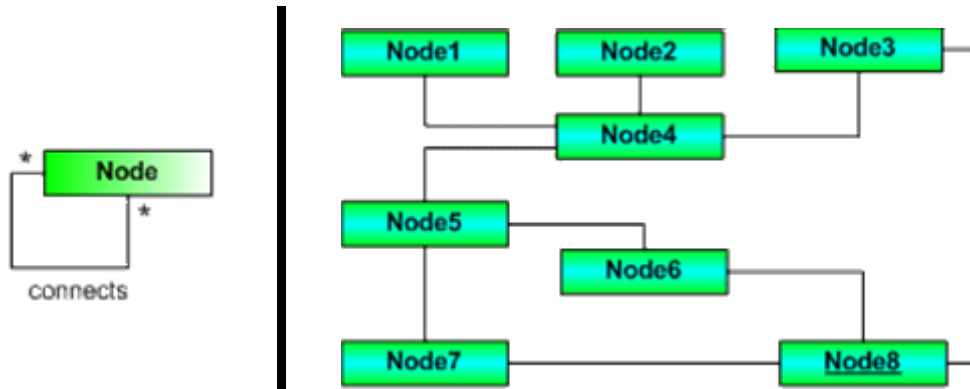
- Los objetos se representan con un rectángulo con nombre.
- El nombre: nombreObjeto:clase (subrayado). Debe especificarse obligatoriamente al menos uno de los dos.
- Las relaciones normalmente no contienen nombre ni multiplicidad.
- Se puede especificar datos de los objetos (atributos, estados, ...).



3.4.1 Diagrama de clases y diagrama de objetos

Diagrama de Clases |

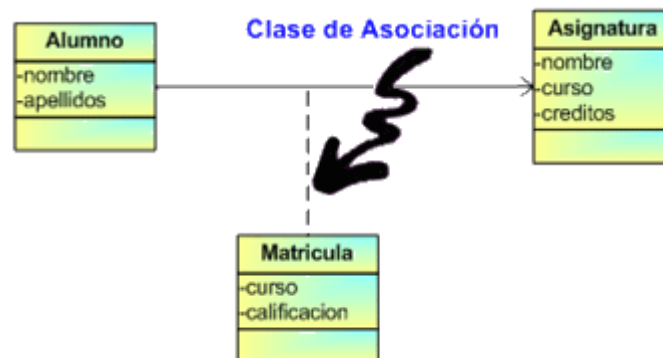
Diagrama de Objetos



3.5 ASPECTOS AVANZADOS DEL DIAGRAMA DE CLASES

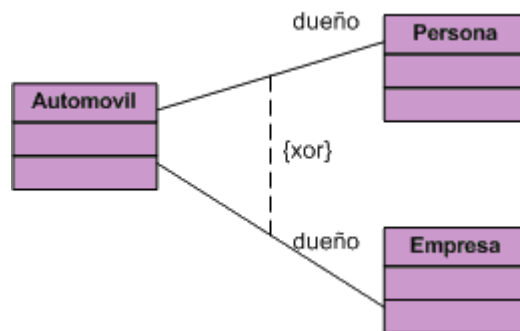
3.5.1 Clases de asociación

- Una clase de asociación es una clase conectada a una relación.
- Se usa cuando la asociación contiene atributos y operaciones propias.
- Cada asociación solo puede tener una clase de asociación.
- Por cada instancia de la relación debe existir una instancia de la clase.
- Se asocia con la relación mediante una **línea discontinua**.



3.5.2 Restricción XOR

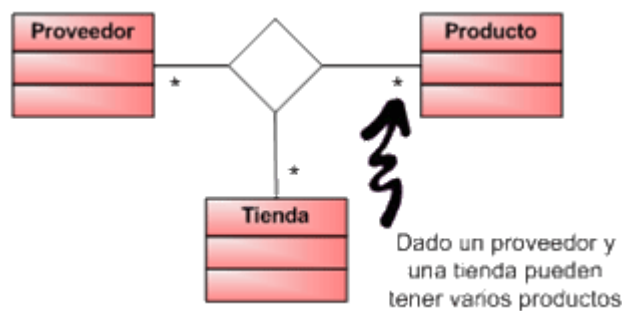
- La **restricción XOR** indica que sólo una asociación es válida.
- Este tipo de relación se representa con una línea punteada que une dos asociaciones junto con la aclaración de la restricción que rige sobre ellas.



Un automóvil puede tener como dueño una persona o una empresa, pero no ambos.

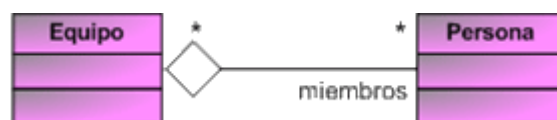
3.5.3 Asociación ternaria

- Una **asociación ternaria** es una asociación entre tres clases.
- Se representa como un rombo del cual salen líneas de asociación a las clases.
- Pueden definirse asociaciones n-arias.



3.5.4 Agregación compartida

- La **agregación compartida** es cuando la parte puede pertenecer a más de una todo de la misma clase.
- Se indica con una multiplicidad * (1..*, 0..* ó *) en la parte del todo.

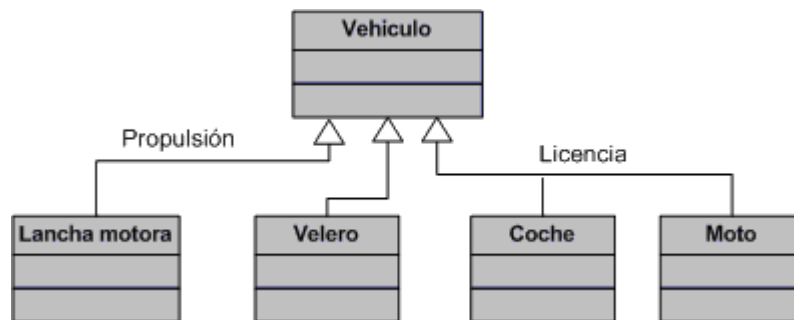


Una persona puede pertenecer a más de un equipo

3.5.5 Conjuntos de generalización

- Los **conjuntos de generalización** (Generalization Sets) sirven para organizar diferentes conjuntos de subclases.

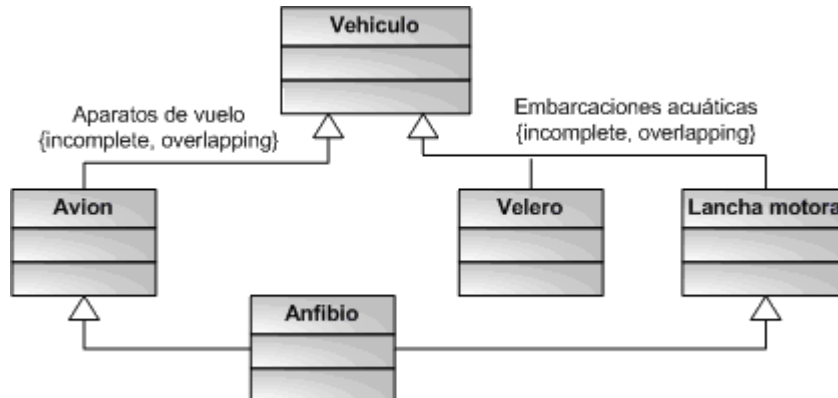
- Especifican particiones en las cuales pueden organizarse las relaciones de herencia que no son mutuamente excluyentes.



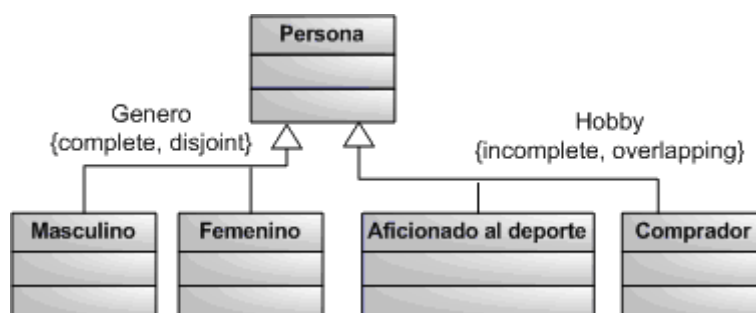
Los conjuntos pueden poseer los **atributos**:

- **Complete** o **incomplete**: Indica si se pueden añadir más clases derivadas al conjunto.
Es decir, si todas las instancias de la superclase están en una subclase o no.
- **Disjoint** u **overlapping**: Indica si subsiguientes clases pueden heredarse de más de una fuente de subconjuntos distintos.
Es decir, Si cada instancia de la superclase está como mucho en una subclase o no

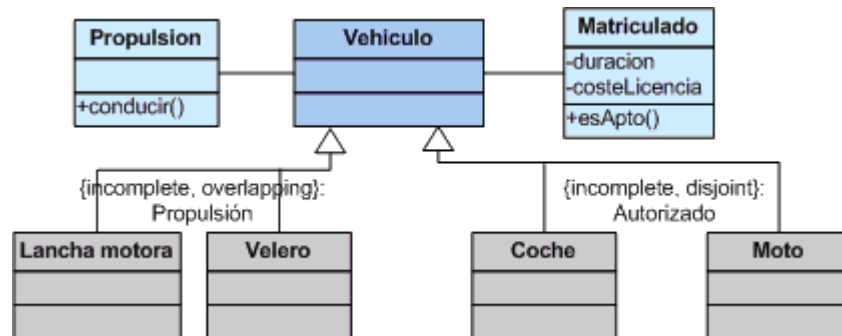
Ejemplo 1



Ejemplo 2



- Un **supertipo** (Powertype) sirve para modelar las clases hechas de las subclases.
- Permite especificar claramente la clase en función de la cual se realiza la partición.
- Dependiendo del supertipo la clase heredará unos atributos u otros.



3.6 DIAGRAMA DE PAQUETES

Los **diagramas de paquetes** se emplean para visualizar la organización en paquetes de los diferentes elementos que conforman el sistema, de forma que se puede especificar de manera visual el nombre de los espacios de nombres.

Normalmente este modelo está **asociado a diagramas de casos de uso o de clases**, aunque no está limitado su uso a estos dos modelos solamente.

Los elementos que componen un paquete poseen el mismo espacio de nombres, por lo que sus nombres deberán ser únicos.

Estos paquetes podrán ser utilizados tanto para representar relaciones lógicas como físicas.

Las relaciones permitidas entre paquetes son: **generalización** y **dependencia**.

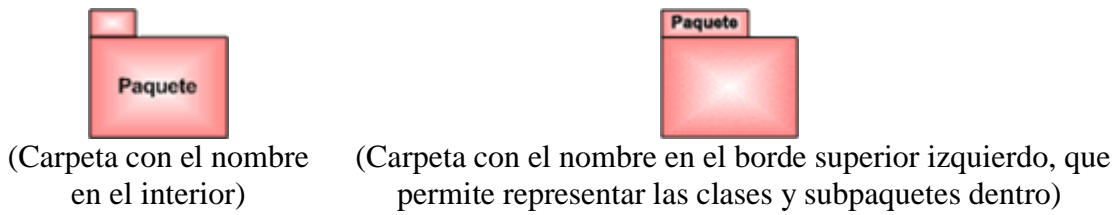
Cuando se decida incluir clases dentro de los paquetes

- Las clases de una misma jerarquía deberían estar dentro del mismo paquete.
- Las clases que colaboren entre sí, deberían formar parte del mismo paquete. Por ejemplo las clases de la interfaz de usuario, de entrada/salida, base de datos, etc.

Paquetes

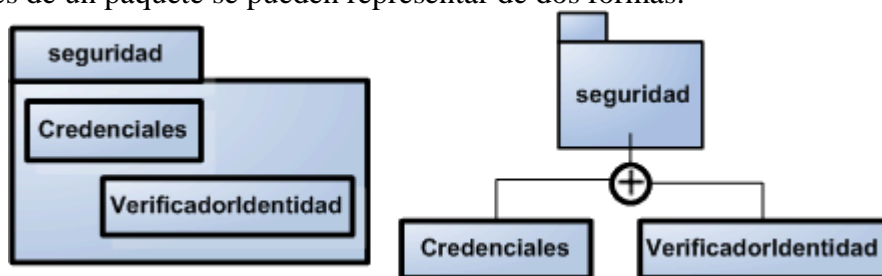
- Son un mecanismo para la organización de los modelos/subsistemas agrupando elementos de modelado.
- No tienen instancias.

- Un paquete puede contener otros paquetes, sin límite de anidamiento, pero cada elemento pertenece a sólo un paquete.
- Hay dos posibles representaciones:

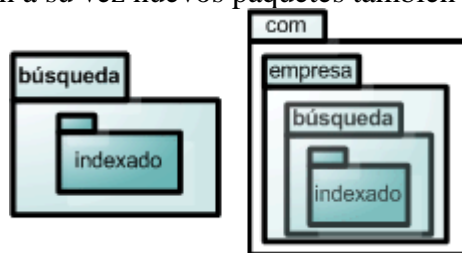


3.6.1 Representación

Las clases de un paquete se pueden representar de dos formas:



Los paquetes que incluyan a su vez nuevos paquetes también es posible representarlos:

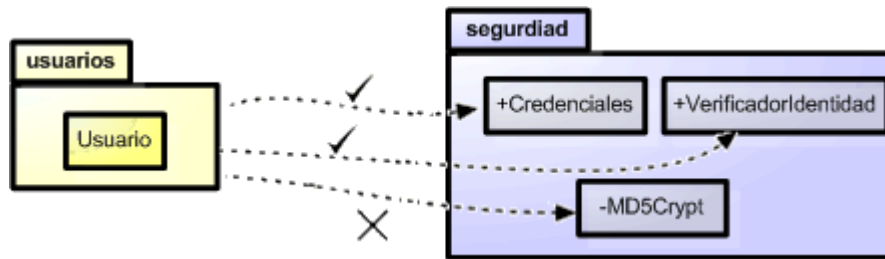


Si el número de paquetes es excesivo para su representación, se puede optar por esta representación:



3.6.2 Visibilidad

Los elementos de un paquete tendrán una visibilidad determinada, que puede hacer que en algunos casos sean accesibles desde el propio paquete, pero no desde el exterior.



Tipos de visibilidad considerados en UML 2.0 para los paquetes

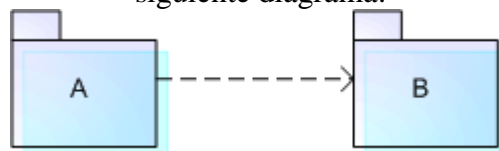
- **Pública:** Todos los miembros son visibles.
- **Privada:** Los elementos son accesibles para el propio paquete y para los que lo importan (dependencia).
- **Protegida:** Los elementos son accesibles para el propio paquete, para los que lo importan, y para sus paquetes derivados (generalización).
- **Paquete:** Es la más restrictiva. Los elementos solo están disponibles para el paquete que los posee (no se pueden importar).

3.7 RELACIONES DE LOS DIAGRAMAS DE PAQUETES

3.7.1 Dependencia

La **dependencia entre paquetes** aparece cuando una clase de un paquete necesita de otra clase que pertenece a un paquete distinto.

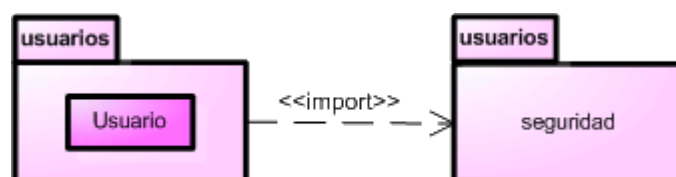
Por ejemplo, si un elemento del paquete A, necesita de otro del paquete B, tendríamos el siguiente diagrama:

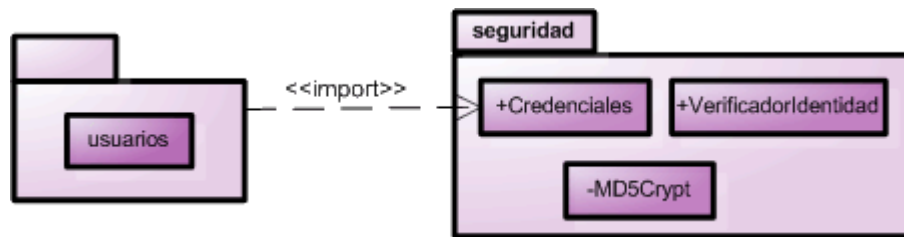


3.7.2 Importación

Cuando un paquete importa a otro paquete, los elementos del paquete que realiza la importación, podrán usar los elementos del paquete importado sin necesidad de utilizar su referencia completa (**paquete.Elemento**).

El estereotipo es `<<import>>` y es una **importación pública**.



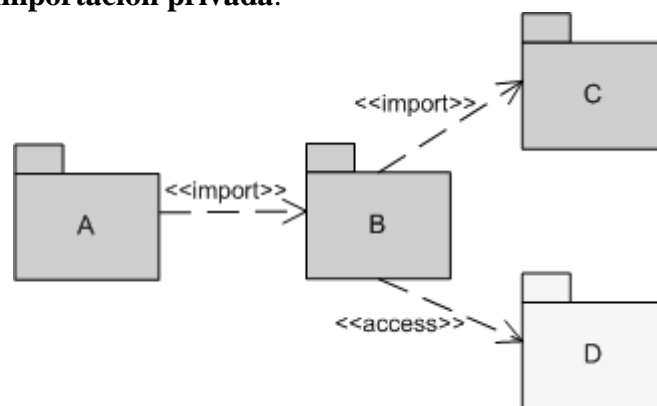


La importación lleva todas las clases públicas y privadas al paquete objeto

3.7.3 Acceso

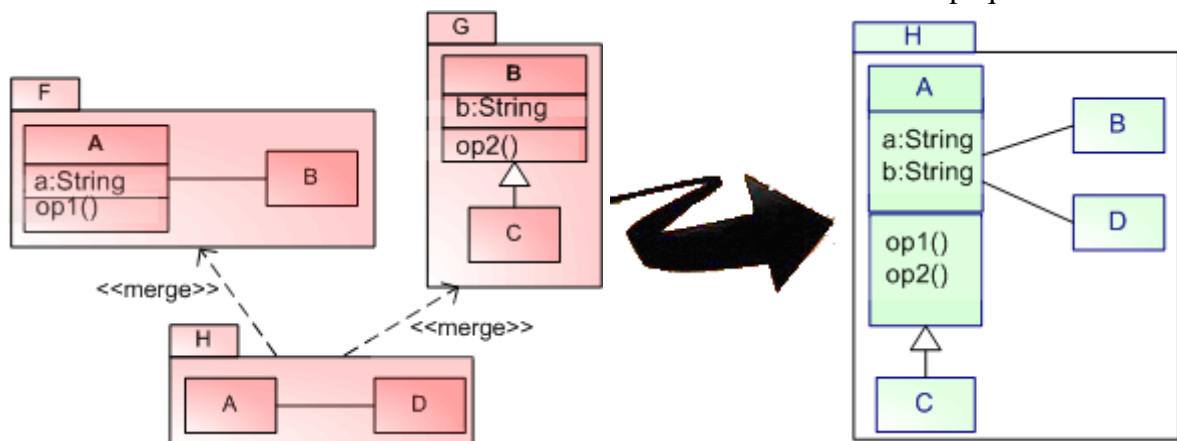
El estereotipo es **<<access>>** y se utiliza para acceder a un paquete sin traer su contenido.

El acceso es una **importación privada**.

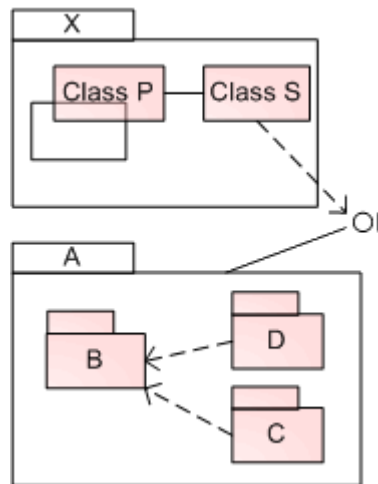


El estereotipo **<<merge>>** sirve para combinar dos paquetes.

Debe usarse cuando existan elementos con el mismo nombre en distintos paquetes.

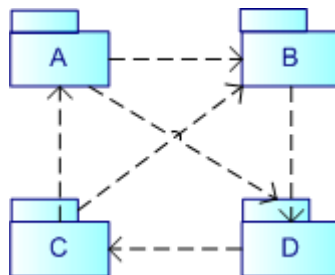


Los paquetes pueden hacer públicas las interfaces que implementan.

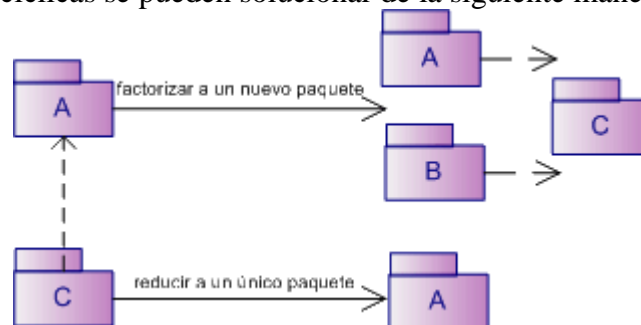


3.7.4 Dependencias cíclicas

En ocasiones las dependencias entre paquetes son complicadas. Hacen que un cambio en la visibilidad de un elemento, pueda afectar a todo el conjunto del sistema.



Las dependencias cíclicas se pueden solucionar de la siguiente manera.



4. ESTRUCTURA COMPUESTA

4.1 ELEMENTOS CONSTITUTIVOS

El **diagrama de estructura compuesta** permite visualizar de manera gráfica las partes que definen la estructura interna de un clasificador (p.ej. una clase), incluyendo sus puntos de interacción con otras partes del sistema.

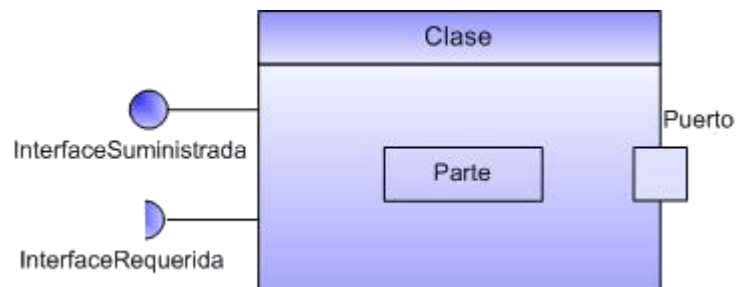
En el diagrama de clases se describen con detalle todas las clases (con sus atributos y métodos) y las relaciones que se establecen entre ellas. El **diagrama de estructura compuesta** permite representar las **clases** como elementos compuestos, mostrando sus interfaces, puertos, y demás partes que la componen.

El diagrama de estructura compuesta también permite mostrar y detallar **colaboraciones** entre varios elementos para realizar una funcionalidad.

Dentro del diagrama de estructura compuesta destacan los siguientes elementos:

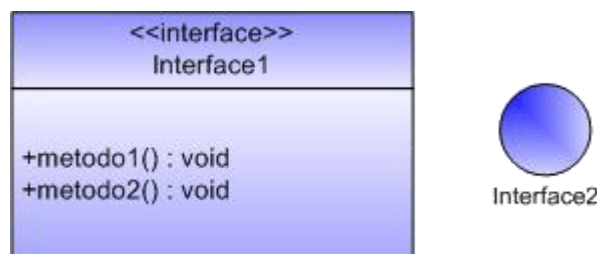
Clase

Es la representación de un objeto (componente), que refleja su estructura y comportamiento dentro del sistema.



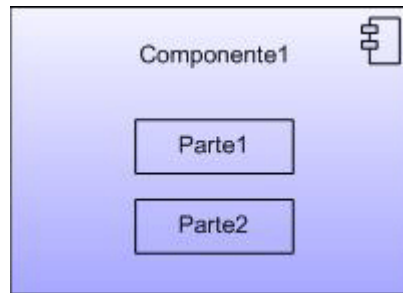
Interfaz

Se trata de clases que poseen solamente las definiciones de los métodos (públicos), y como atributos solamente pueden tener constantes.



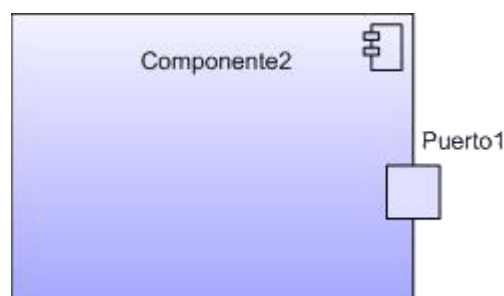
Parte

Representa un conjunto de una o más instancias de una clase o interfaz. Sintaxis de una parte: {<nombre>} / <rol> : <tipo>.



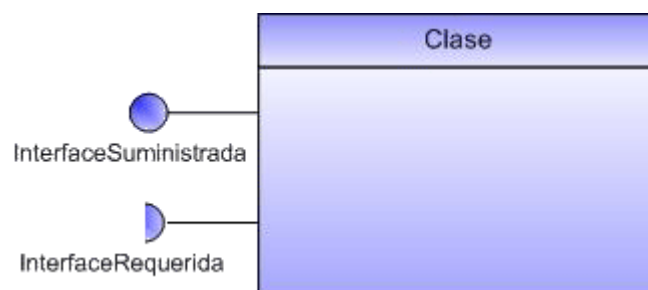
Puerto

Define la interacción de un componente con su ambiente. Las interfaces expuestas controlan esta interacción.



Interfaz Expuesta

Es la manera gráfica de mostrar la forma de comunicación del componente (clase, interfaz o parte).



Colaboración

Define un conjunto de roles cooperantes y sus conexiones, de forma que entre todos muestren el conjunto de una funcionalidad.



Conector

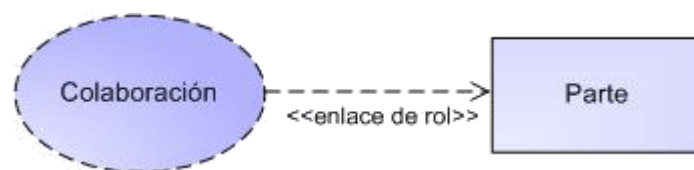
Ilustra un enlace de comunicación entre partes para llegar al propósito de la estructura.

Ensamblado (assembler)

Enlace que se establece entre una interfaz expuesta “requerida” de un componente, y una interfaz expuesta “provista” por otro componente.

Enlace de Rol

Es el mapeo entre los roles internos de una colaboración y las respectivas partes necesarias para implementar una situación específica.



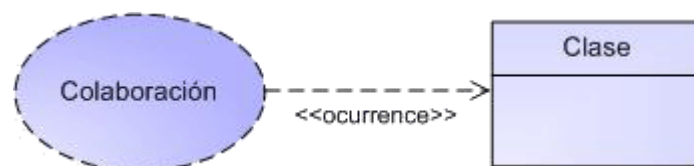
Representación

Indica que una colaboración representa a un componente.



Ocurrencia

Indica que una colaboración se usa en un componente.

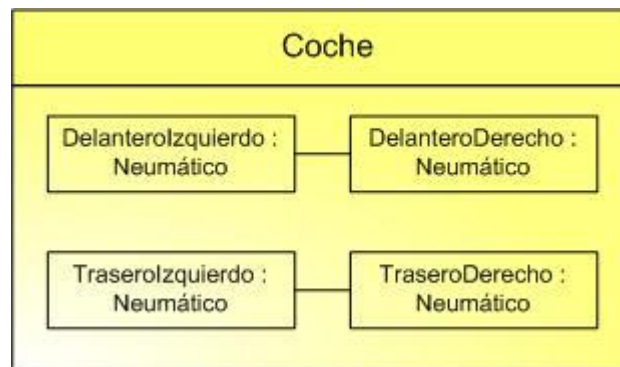


4.2 MOSTRANDO LA ESTRUCTURA INTERNA

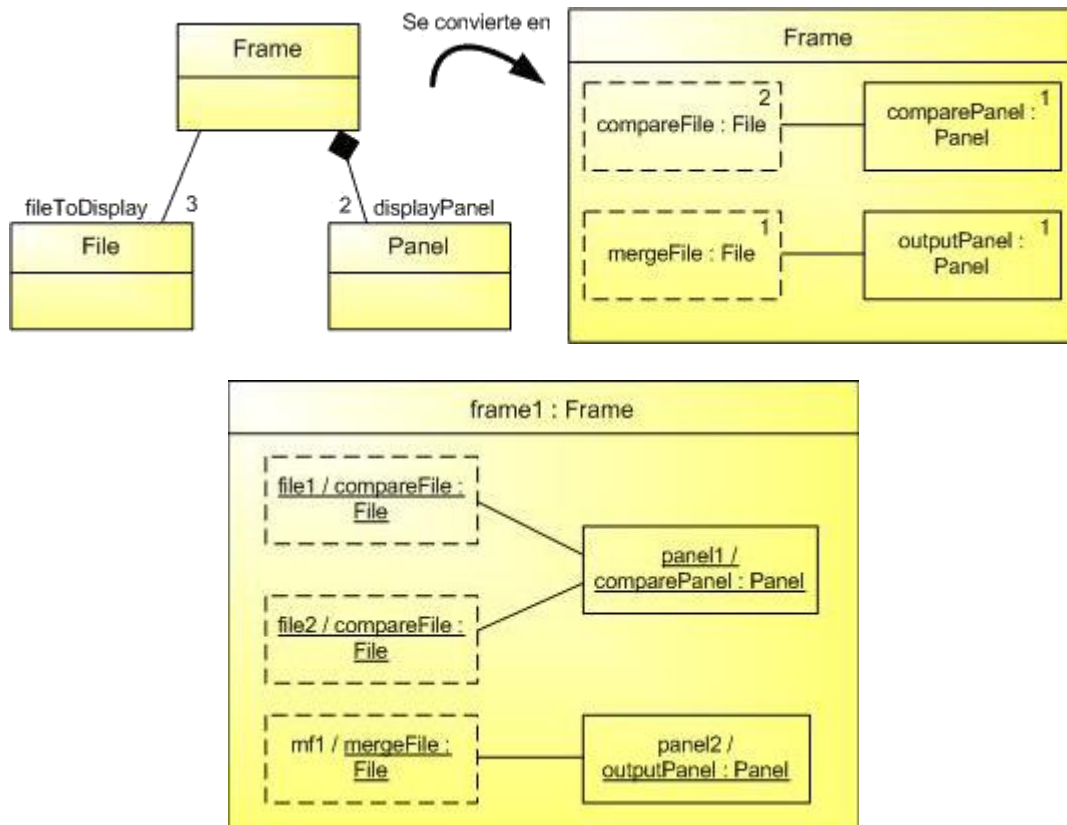
Los diagramas de estructura compuesta se pueden emplear para mostrar la estructura interna de un clasificador mediante partes y conectores.

Las partes no representan otros clasificadores ni instancias, representan el rol que un clasificador tomará en un determinado momento (conjunto de instancias que pueden representar ese papel en un momento concreta).

Los conectores indican relaciones entre partes, pueden usar multiplicidad igual que las asociaciones.



Ejemplo

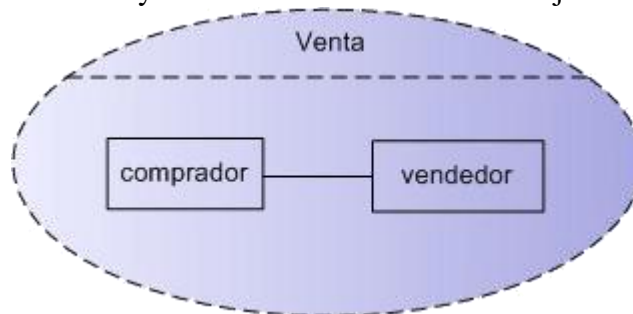


4.3 DISEÑANDO COLABORACIONES

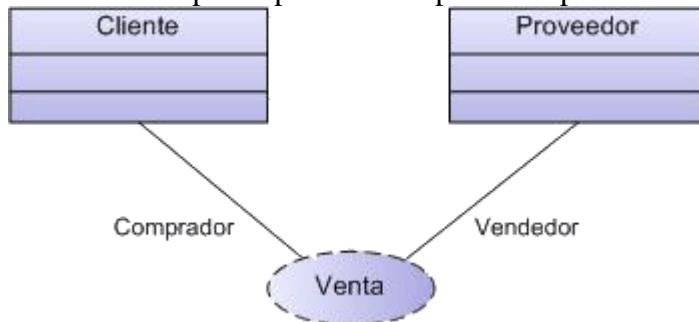
Una **colaboración** es una colección de instancias relacionadas entre sí, usando conectores que muestran el flujo de comunicación. Es decir, permiten describir los aspectos más relevantes de una cooperación entre un conjunto de instancias especificando los roles de cada una

Características

- No son **instanciables**.
- Se representan con una **elipse** con una **línea discontinua**.
 - El nombre se coloca en parte superior.
 - Los roles y los conectores se colocan debajo del nombre.



- Las **colaboraciones** deben ser **descritas**. Por ejemplo, usando diagramas de interacción, diagramas de objetos o texto.
- Mediante **asociaciones** (enlaces de rol) se puede (y debe) expresar qué clasificadores participan en una aplicación particular de la colaboración.



Nota: Las colaboraciones definen comportamientos comunes capturados durante el diseño. Pueden **reutilizarse** en el mismo u otros diseños con otros participantes.

4.4 PATRONES DE DISEÑO Y UML

Los **patrones de diseño** definen mecanismos genéricos de diseño. Son soluciones inteligentes, genéricas, bien probadas, simples y reusables a problemas comunes que aparecen en sistemas orientados a objetos.

Los patrones se componen de una **descripción de un problema** y de una **solución**.

Esta solución describe mediante clases y objetos, su estructura y comportamiento dinámico (diagramas de secuencia y diagramas de clases).

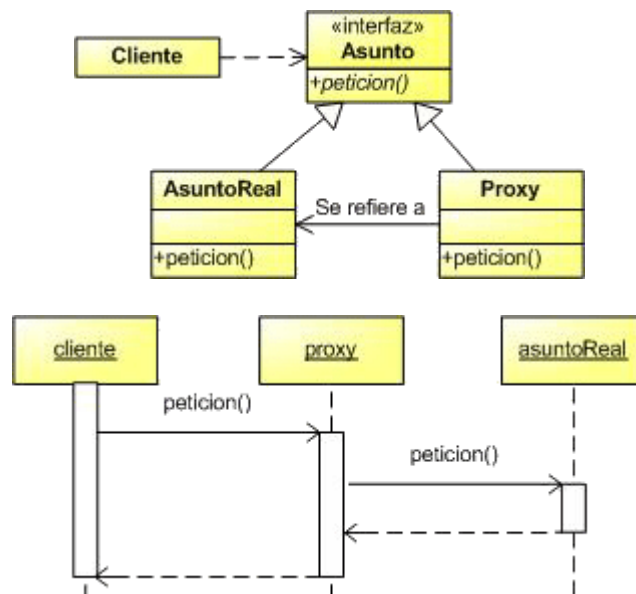
Ejemplos de patrones:

- Singleton, Proxy, Adapter, Chain of responsibility (COR), Iterator, Memento, Observer, Model-View-Controller (MVC).

Ejemplo Patrón Proxy (Diagrama de clases)

Problema: Un objeto (objeto real) no puede ser instanciado directamente por razones de rendimiento, localización, o restricciones de acceso y seguridad.

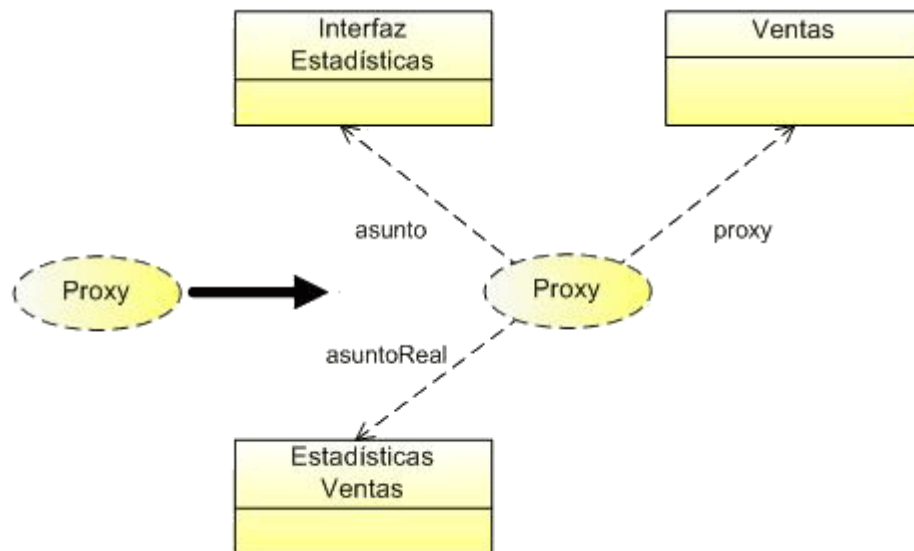
Implementación: Subrogar todas las peticiones a un objeto intermedio (Proxy).



Descripción: Tenemos un objeto padre Asunto del que heredan otros dos: AsuntoReal y Proxy, todos ellos tienen un método petición(). El cliente llamaría al método petición() de Asunto, el cual pasaría la petición a Proxy, que a su vez instanciaría AsuntoReal y llamaría a su petición().

Los patrones se expresan mejor mediante **colaboraciones**.

Ejemplo Patrón Proxy (Diagrama de interacción)



5. ARQUITECTURA FÍSICA

5.1 CONCEPTOS BÁSICOS

¿Qué es la Arquitectura Física?

Muestra cómo el sistema diseñado se empaqueta en componentes ejecutables y se distribuye en los elementos físicos con capacidad para ejecutarlos.

Elementos Hardware

- **Dispositivo** (device). Elemento con capacidad de procesamiento.
- **Línea de Comunicación** (communication path). Conexiones entre procesadores.
- **Entorno de ejecución** (execution environment). Proveen el contexto necesario para que los componentes software puedan ejecutarse. Generalmente son subnodos dentro de los dispositivos

Elementos Software

- **Componentes**. Conjunto de elementos de arquitectura lógica agrupados.
- **Artefacto** (artifact). Implementación de un componente en el entorno de desarrollo.
 - Artefacto fuente. <<source>>.
 - Artefacto ejecutable. <<executable>>.
- **Especificador de despliegue** (deployment specifier). Conjunto de propiedades que determinan los parámetros de ejecución de un artefacto o componente.

5.2 DIAGRAMAS DE COMPONENTES

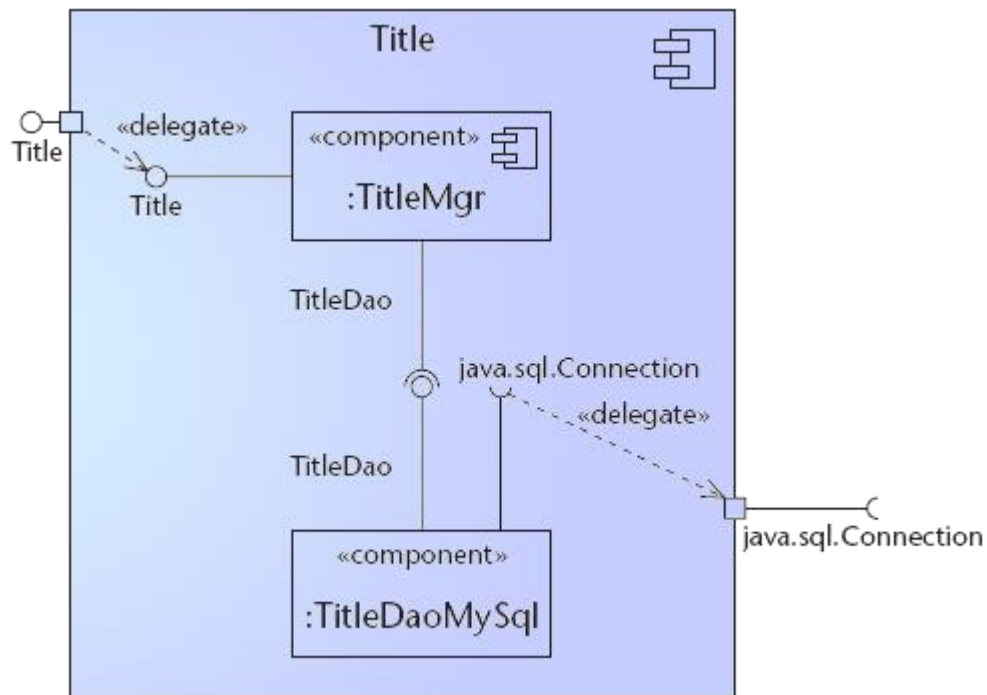
El **diagrama de componentes** proporciona una visión física de la construcción del sistema de información. Muestra la organización de los componentes software, sus interfaces y las dependencias entre ellos.

Un **componente** es un módulo de software que puede ser código fuente, código binario, un ejecutable, o una librería con una interfaz definida. Es un elemento o grupo de elementos identificables.

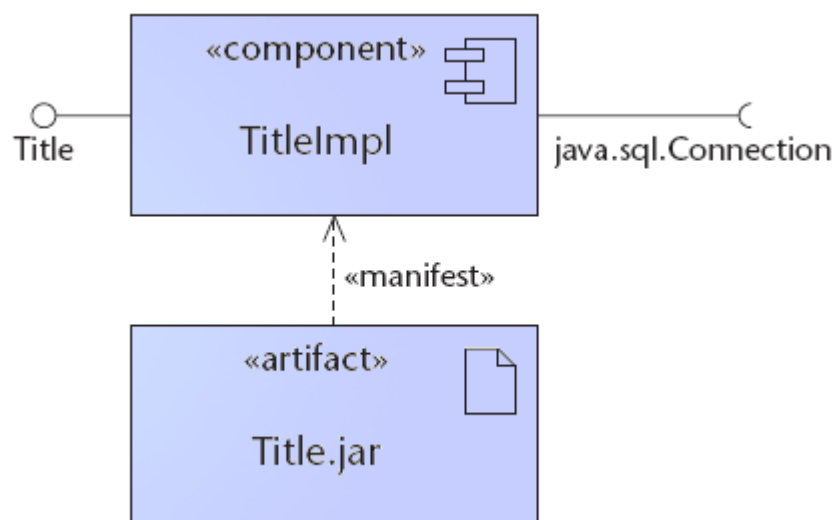
Los componentes representan elementos **reusables** (en el mismo u otro sistema) de la arquitectura.

Una **interfaz** establece las operaciones externas de un componente, las cuales determinan una parte del comportamiento del mismo. Además se representan las dependencias entre componentes o entre un componente y la interfaz de otro, es decir uno de ellos usa los servicios o facilidades del otro.


Ejemplo 1



Ejemplo 2



5.2.1 Componentes


Rectángulo con el estereotipo `<<component>>`, el nombre del componente y opcionalmente el icono  en la parte superior derecha.



Un componente puede poseer subcomponentes.

5.2.2 Artefactos

Para distinguir distintos tipos de artefactos (y distinguirlos de los componentes) se les puede asignar un estereotipo: `<<artifact>>`, `<<source>>` o `<<executable>>`. También se pueden especificar estereotipos para artefactos de entornos específicos: `<<EJB>>`, `<<DLL>>`, etc.


Se puede mostrar el icono  en la parte superior derecha del artefacto.



5.2.3 Interfaces

La **interfaz implementada** por un componente se representa como un pequeño círculo situado junto al componente que lo implementa y unido a él por una línea continua. La interfaz puede tener un nombre que se escribe junto al círculo. Un componente puede proporcionar más de una interfaz.



Las **interfaces requeridas** se muestran con el conector . Se pueden usar **puertos** para mostrar la interfaz requerida e implementada por cada componente formalizando sus puntos de interacción.

Los ensambles (assembler) se pueden usar para mostrar claramente las relaciones entre interfaces requeridos e implementados.

5.2.4 Relaciones de dependencia

Una **relación de dependencia** se representa mediante una línea discontinua con una flecha que apunta al componente o interfaz que provee del servicio o facilidad al otro. Si la dependencia ocurre entre:

- artefactos de código. Al recompilar uno debe recompilarse el otro.
- artefactos ejecutables. Un componente requiere de otro en tiempo de ejecución.

También pueden darse dependencias entre componentes / artefactos / interfaces / especificaciones.

La relación puede tener un estereotipo

- <<**delegate**>>: Para indicar que un puerto o interfaz delega la implementación del interfaz en otro componente. Se nombra responsable a otro componente de implementar el interfaz.
- <<**manifest**>>: Para indicar que un artefacto representa la implementación de un componente.

5.3 DIAGRAMAS DE DESPLIEGUE

El objetivo del **diagrama de despliegue** es mostrar la disposición de las particiones físicas del sistema de información y la asignación de los componentes software a estas particiones. Es decir, las relaciones físicas entre los componentes software y hardware del sistema.

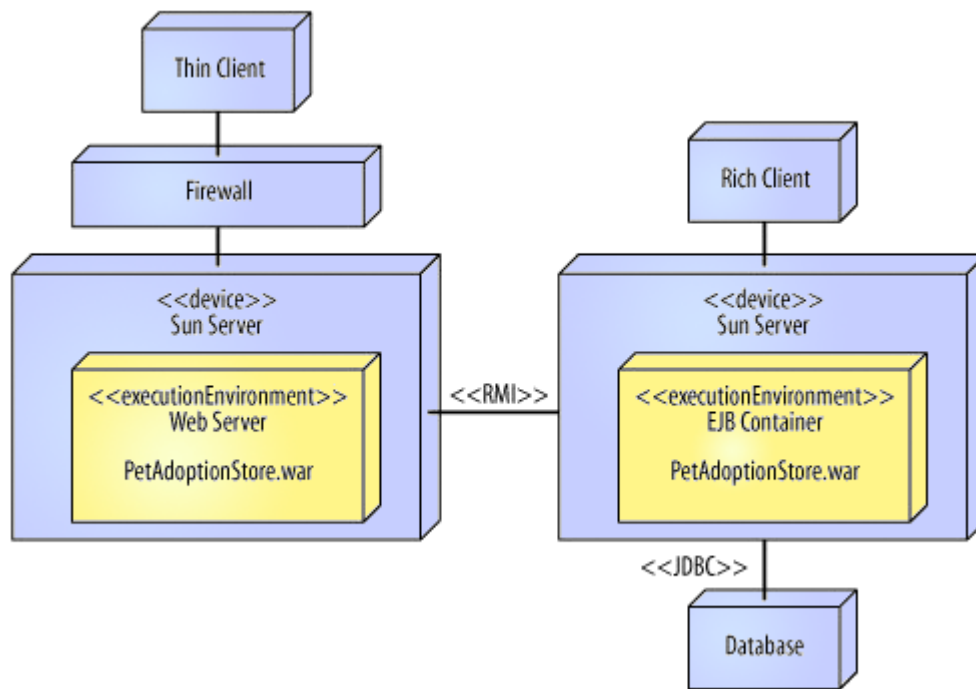
Describe la arquitectura en tiempo de ejecución de los dispositivos, los entornos de ejecución y los artefactos.

En este diagrama se representan dos tipos de elementos, **nodos** y **conexiones**, así como la distribución de componentes del sistema de información con respecto a la partición física del sistema.

Se puede emplear para:

- mostrar la **topología** de los diferentes **elementos hardware** de un sistema.
- mostrar los **artefactos software** distribuidos en cada nodo.

Ejemplo

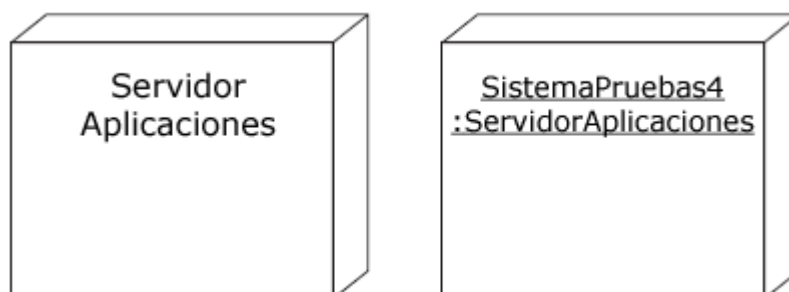


5.3.1 Nodos

Los **nodos** son **recursos computacionales** en los que se pueden desplegar los artefactos.

Características:

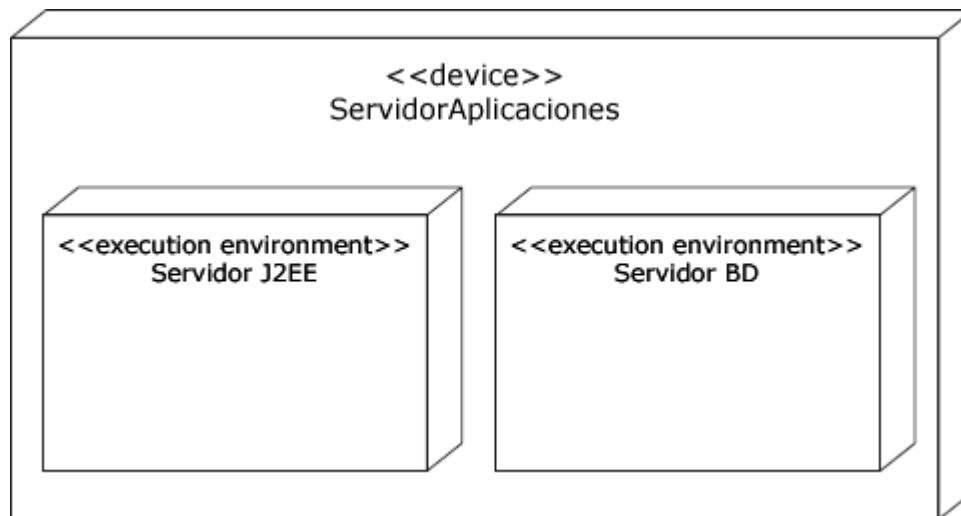
- Son **dispositivos físicos**: ordenadores, dispositivos móviles, dispositivos de comunicaciones, lectores de tarjetas, sensores, detectores, etc.
- Pueden incluir **subnodos** para reflejar distintos **entornos de ejecución**.
- Se representa como un cubo con el nombre en su interior.
- Un nodo es un clasificador (classifier).
- El nombre puede ser de clase o de instancia.



Los **nodos** pueden incluir **subnodos** para reflejar distintos **entornos de ejecución**, por ejemplo, *contenedores J2EE*, *bases de datos*, *monitores de teleproceso*, etc.

Para distinguir los nodos que representan dispositivos de los que representan entornos de ejecución se emplean estereotipos:

- <<device>> → Dispositivos.
- <<execution environment>> → Entornos de ejecución.

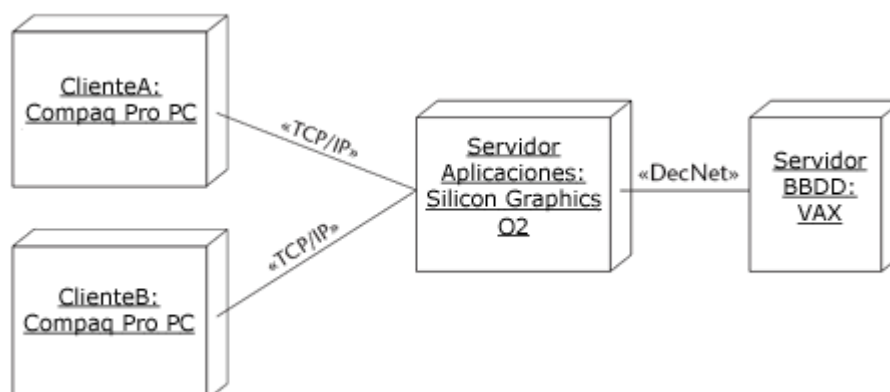


5.3.2 Conexiones

Las **conexiones** representan las **formas de comunicación** entre nodos.

Características:

- Indican que hay algún **tipo de comunicación entre los nodos**, y que los nodos intercambian objetos y/o mensajes.
- Se representan como **asociaciones bidireccionales**.
- Se puede emplear un estereotipo para identificar el protocolo o el tipo de red utilizador.
- También pueden ser llamadas **enlaces** o **rutas de comunicación**.

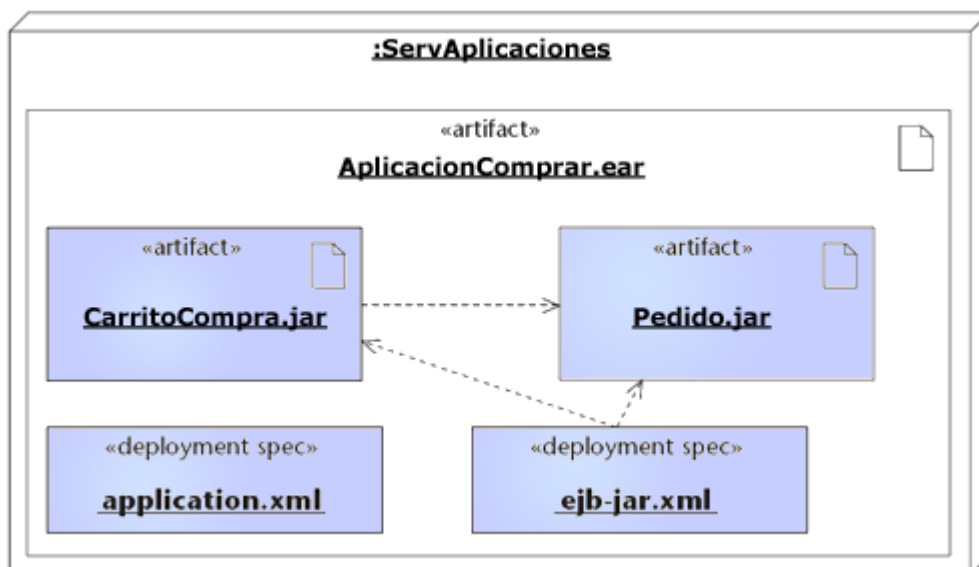


5.3.3 Artefactos desplegados

Los **artefectos** se pueden **desplegar en los nodos**.

Características:

- Se muestran con el **nombre subrayado**.
- Un artefacto desplegado en un nodo puede presentarse con una serie de propiedades describiendo los parámetros de ejecución de ese artefacto para ese nodo en particular.
Esto puede modelarse:
 - Directamente dentro del componente como propiedades del mismo.
 - Como una especificación de despliegue. Clasificador con el estereotipo **<<deployment spec>>**.
- Se pueden mostrar relaciones de dependencia entre **artefectos** y **entre un artefacto y una especificación de despliegue**.
- Un artefacto puede contener otros artefactos y especificaciones de despliegue.



6. DIAGRAMAS DE INTERACCIÓN

6.1 LOS DIAGRAMAS DE INTERACCIÓN EN UML

El **diagrama de interacción**, representa la forma en como un Cliente (Actor) y Objetos (Clases) se comunican entre sí en respuesta a un evento.

Los **diagramas de interacción** modelan:

- la manera en que el sistema hace su trabajo.
- las interacciones más importantes que tienen lugar durante la ejecución y que conforman el sistema

Los diagramas de interacción forman parte del modelo lógico.

Los tipos de diagramas de Interacción son:

- **Diagrama de Secuencia:** Muestra objetos interactuando a lo largo de su línea de vida, representando el orden general.
Los diagramas de secuencia son los más populares.
- **Diagrama de Colaboración:** Muestra los mensajes intercambiados entre los objetos, centrándose en los enlaces entre los objetos.
- **Diagrama General de Interacción:** Tratan al resto de diagramas como una unidad, mostrando como colaboran para dar respuesta a una interacción mayor (sin mostrar los detalles de ellas).
- **Diagrama de Tiempos:** Muestra las interacciones usando un eje de tiempo preciso

6.2 DIAGRAMAS DE SECUENCIA

Un **diagrama de secuencia** muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo.

Se centran en la secuencia de mensajes, es decir, en cómo los mensajes se envían y reciben entre un determinado número de objetos.

El diagrama tiene dos ejes:

- Vertical: muestra el tiempo
- Horizontal: muestra el conjunto de objetos

Un diagrama de secuencia revela una interacción para un escenario específico (no para todo el caso de uso).

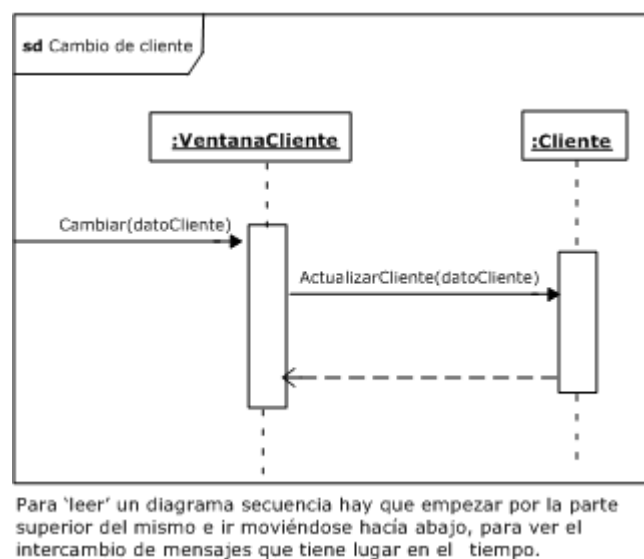
Deben colocarse dentro de un marco rectangular con su nombre en la esquina superior derecha precedido de 'sd' y encuadrado en una etiqueta pentagonal.

Notacion en UML

En la parte superior se muestran horizontalmente los **objetos** (llamados *participantes* en UML 2) involucrados en la interacción. Cada objeto se representa como un **rectángulo** con el *nombre del objeto y/o clase* en su interior.

Bajo el rectángulo se muestra una **línea vertical punteada** que representa la *vida del objeto* durante la ejecución de la secuencia. En algún momento pasa a ser un rectángulo: Tiempo en el que el objeto está activo (su código se ejecuta). La comunicación entre objetos se representa mediante **mensajes** (líneas horizontales entre los objetos).

Ejemplo



6.2.1 Participantes

Los participantes son los **elementos más importantes** del diagrama. Son **rectángulos** que se colocan horizontalmente en la parte superior, con el nombre del participante en el interior. **Participante**

Tienen su correspondiente línea de tiempo.

El nombre de los participantes (que está en el interior) se escribe: nombre [selector] : clase ref descomposición.

Por ejemplo:

```
admin:Administrator,  
eventHandlers[2]:EventHandler,  
:ContentManagementSystem.
```

En sistemas OO(Orientado a Objetos) suelen ser objetos (y se muestran subrayados) y actores, aunque UML 2 no lo limita.

6.2.2 Tiempo

El tiempo en un diagrama de interacción indica **orden**, no duración. Empieza en la parte superior y ‘transcurre’ hacia abajo. El orden en el que están colocadas las interacciones indica el orden en el que ocurren en el tiempo.



6.2.3 Eventos, señales o mensajes

Ilustran las interacciones.

Son **flechas dirigidas** desde el emisor hasta el receptor (es decir, desde el objeto que emite el mensaje hacia el objeto que lo ejecuta).

Las flechas llevan una descripción textual: atributo = nombre_msg (argumentos) : tipo

Ejemplo:

```
mensaje,  
mensaje(v1, v2),  
var=mensaje():clase
```

Los eventos, señales o mensajes pueden representarse de las siguientes maneras:

Mensaje síncrono:

Mensaje asíncrono:

Mensaje de creación:

Mensaje de destrucción:

Mensaje interno:

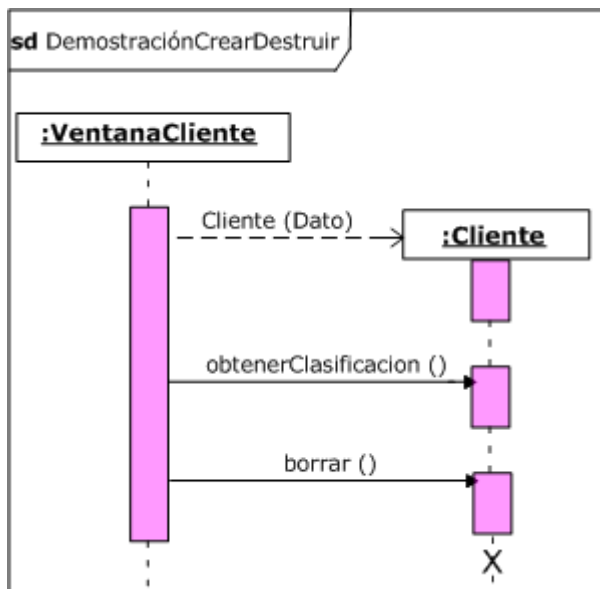
6.2.4 Barras de activación

- Muestran el período de tiempo en el cual el objeto se encuentra desarrollando alguna operación.

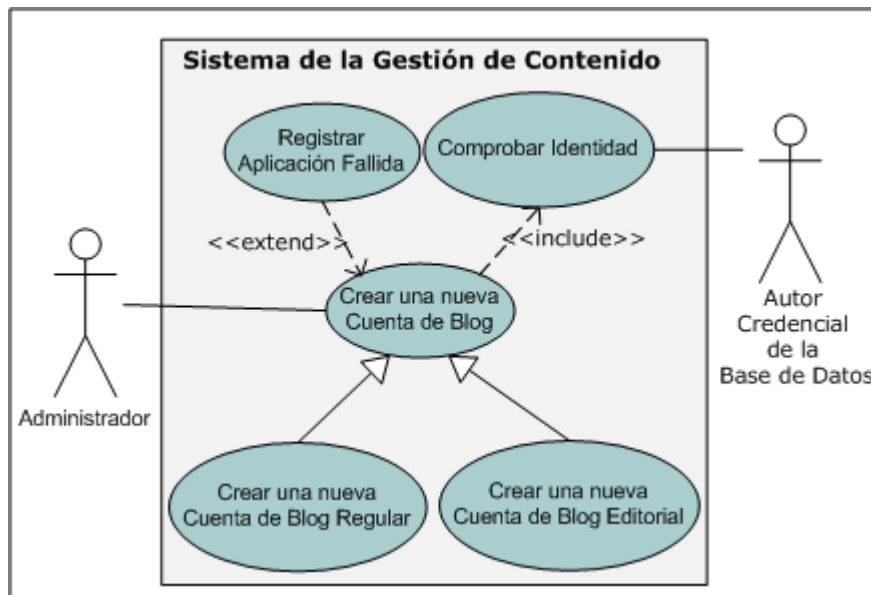
- Cuando se pasa un mensaje a un participante, este ejecuta o invoca alguna acción, por lo que se dice que está activo.
(También denominado: 'Execution Occurrence' o 'Foco de Control').
- Un objeto activo está ejecutando su propio código o esperando una respuesta de algún otro objeto.
- Las barras de activación son opcionales.
- Se representan como un rectángulo delgado que sustituye a la línea de vida del objeto.

6.3 EJEMPLOS DE DIAGRAMAS DE SECUENCIA

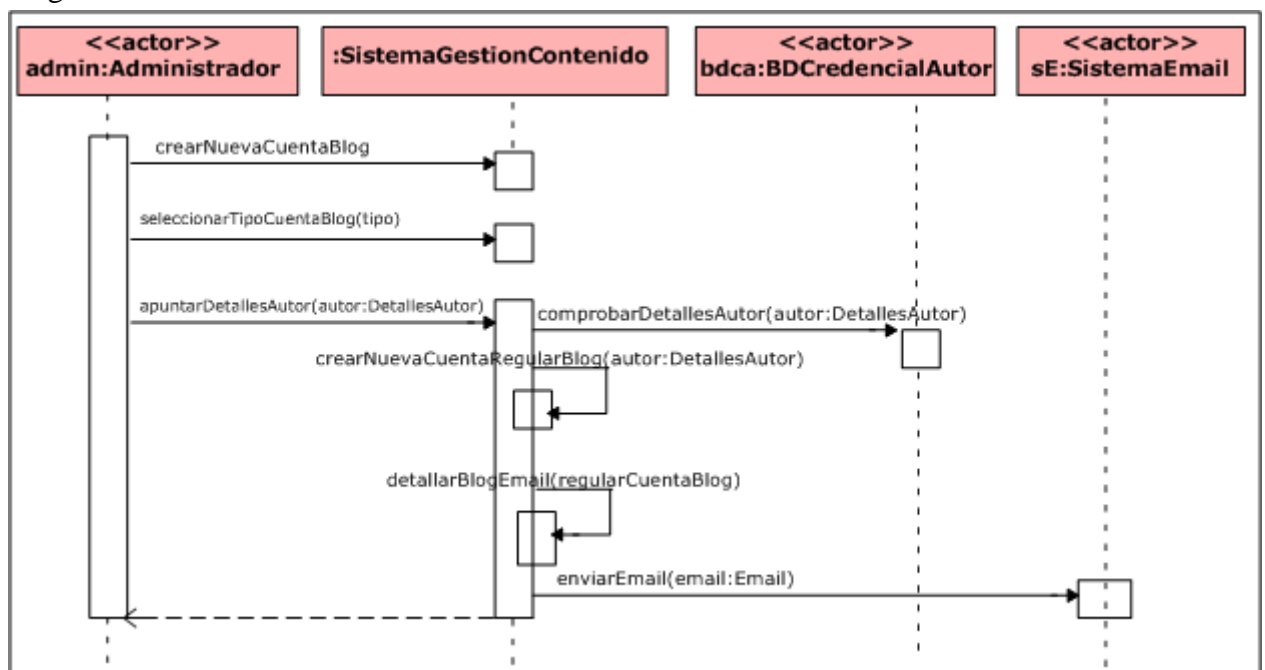
Ejemplo: Creación y destrucción de objetos



Ejemplo: Modelando la interacción de un caso de uso



Este diagrama describe el caso de uso ‘**Crear una Nueva Cuenta Regular de Blog**’ del diagrama de casos de uso anterior.



6.4 FRAGMENTOS COMBINADOS DE LOS DIAGRAMAS DE SECUENCIA

6.4.1 Fragmento combinado

Las modificaciones añadidas a los diagramas de secuencia en UML 2 tienden básicamente a permitir la reutilización de los diagramas, agregando los elementos de tipos **Fragmento Combinado**.

Un **Fragmento de secuencia (o Fragmento Combinado)** es una caja que incluye una porción de las iteraciones dentro de un diagrama de secuencia. No se permite que un mensaje ‘cruce’ la caja.

- Se muestran como un rectángulo con las mismas características que tenía el marco de un diagrama de secuencia.
- Se componen de un operando de la interacción más uno o más fragmentos de interacción.
- Puede contener cualquier número de interacciones y los fragmentos pueden anidarse.
- El operador del fragmento (en la esquina superior izquierda) indica el tipo de operación de que se trata.

6.4.2 Tipos de fragmentos

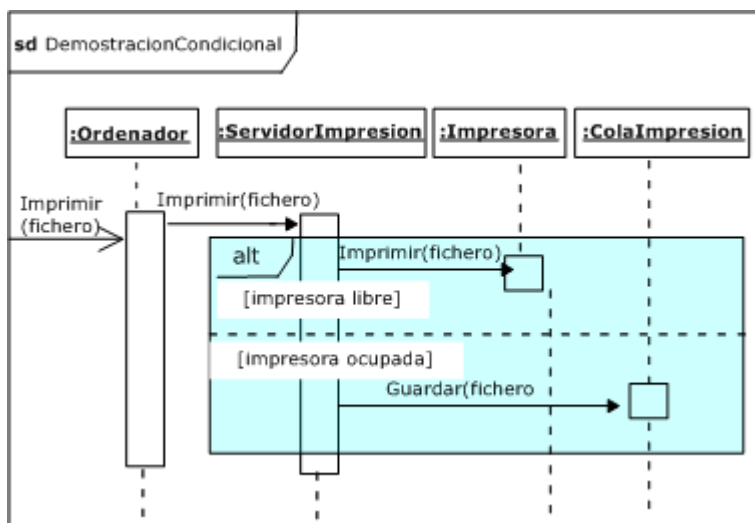
Algunos fragmentos requieren parámetros adicionales como parte de su definición y otros requieren condiciones de guardia en sus fragmentos de interacción.

Tipo	Parámetros	Uso
ref	Ninguno	Representa una interacción definida en algún otro lugar del modelo.
assert	Ninguno	Especifica que las interacciones contenidas en el fragmento deben ocurrir exactamente como están indicadas; de lo contrario el fragmento es declarado inválido y debe elevarse una excepción.
loop	min veces, max veces, [C. guardia]	Ejecuta un bucle en las interacciones contenidas en el fragmento un número especificado de veces o mientras que la condición de guardia sea falsa.
break	Ninguno	Si las interacciones contenidas dentro de este fragmento se completan, entonces la interacción donde está contenido el fragmento ‘break’, generalmente un bucle, debe terminarse.
alt	[C. Guardia1], [C. Guardia2], [else]	Se evalúan las condiciones de guardia asociadas a cada fragmento de interacción comenzando por la primera. Cuando una devuelva true se ejecuta su fragmento asociado y se finaliza la interacción.
opt	[C. Guardia]	Las interacciones contenidas en el fragmento se ejecutarán únicamente si la condición es verdadera.
neg	Ninguno	Declara que las interacciones del fragmento nunca se ejecutarán.
par	Ninguno	Las interacciones pueden ejecutarse en paralelo.
region o	Ninguno	Las interacciones forman parte de una región crítica.

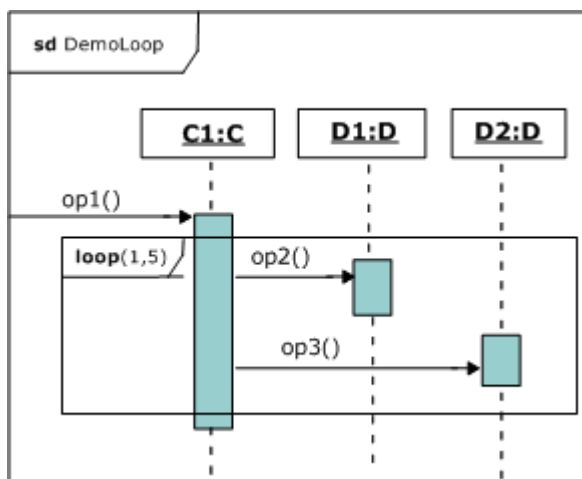
critical		
seq	Ninguno	Indica una ejecución paralela de interacciones con secuenciación débil.
strict	Ninguno	Indica una ejecución paralela de interacciones con secuenciación estricta.

6.4.3 Ejemplos de tipos de fragmentos

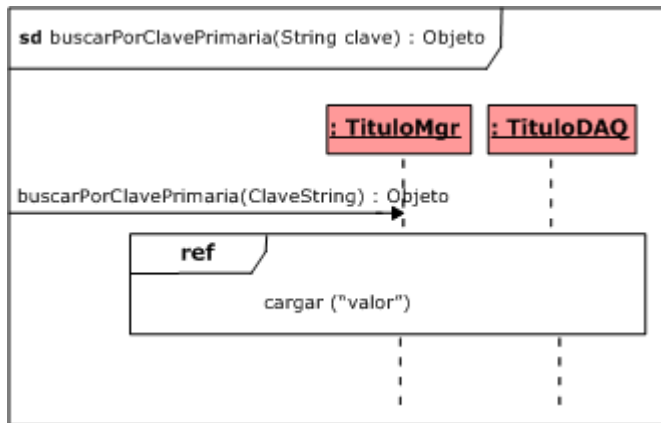
Fragmento alt



Fragmento loop



Fragmento ref



6.5 DIAGRAMAS DE COMUNICACIÓN

Los **diagramas de comunicación**:

- Muestran la colaboración dinámica entre los elementos.
- Añaden otra perspectiva a la interacción centrándose en los enlaces entre los objetos.
- Son útiles para mostrar los enlaces necesarios entre los objetos para pasar los mensajes de interacción.

Los enlaces implican paso de mensajes.

El orden de los eventos suele considerarse secundario.

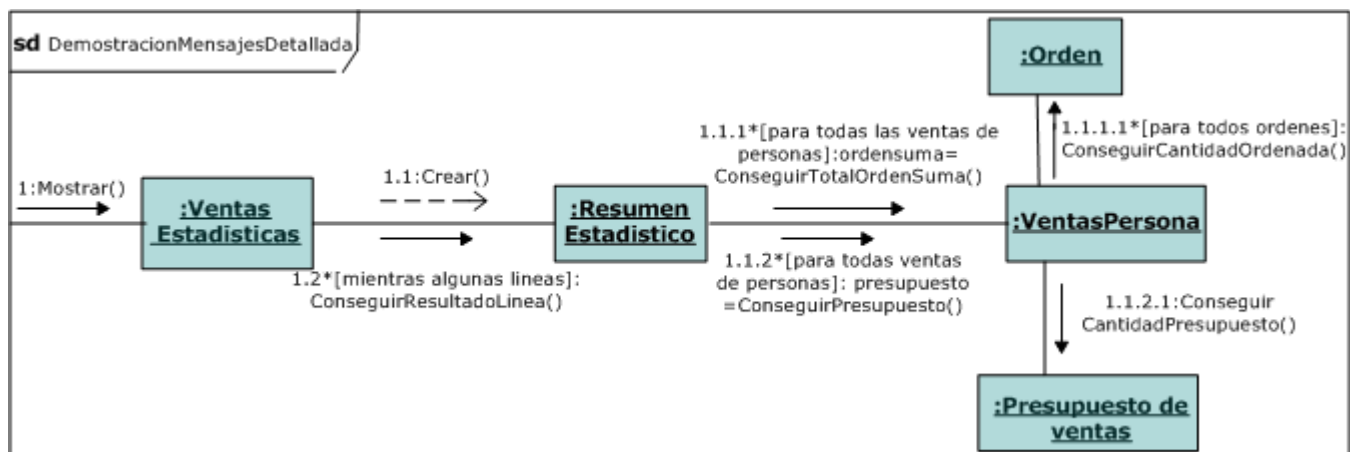
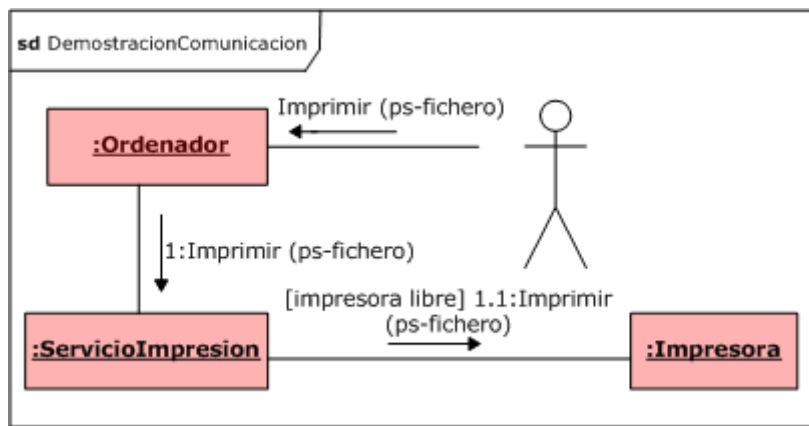
Son muy similares a los diagramas de secuencia:

- Los diagramas de secuencia muestran la interacción de los objetos en el tiempo.
- Los diagramas de comunicación describen como los objetos interactúan en el espacio, más allá de la interacción dinámica.
- También se muestra como enlazan los objetos entre ellos.

Ambos diagramas describen información similar y pueden ser transformados unos en otros sin dificultad.

La selección entre uno u otro suele ser personal.

Ejemplos



6.5.1 Componentes del diagrama de comunicación

6.5.1.1 Participantes, enlaces y mensajes

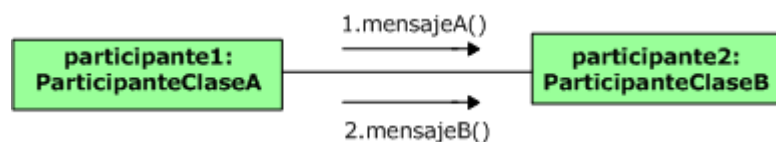
Los **participantes** son rectángulos con el nombre en su interior (igual que en el diagrama de secuencia).

Los **enlaces** son líneas que conectan dos participantes.

Los **mensajes** son flechas pequeñas sobre un enlace entre el emisor y el receptor.

Contienen:

- Nombre y lista de parámetros
- N° de secuencia del mensaje



6.5.2 Secuenciación de los mensajes

El esquema de numeración es:

- Para **mensajes anidados**: Notación con puntos y números.
Ejemplo: 1.1, 1.2, 1.2.1, 1.2.2, 1.3



- Para **mensajes paralelos**: Notación con puntos empleando números y letras.
Ejemplo: 1, 2a, 2b, 2c



6.5.3 Cláusulas de iteración y condición

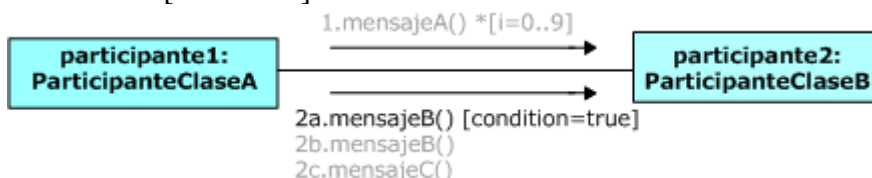
La **cláusula de iteración** permite indicar que un mensaje se invoca varias veces.

La sintaxis es: *[condición o nº de iteraciones]



La cláusula de condición indica que un mensaje sólo se invoca si la condición se evalúa a true.

La sintaxis: [condición]



6.6 COMPARACIÓN DEL DIAGRAMA DE SECUENCIA Y EL DIAGRAMA DE COMUNICACIÓN

La siguiente tabla muestra distintas **características** a comparar indicando cual de los diagramas es mejor para cada una:

Característica	Diagrama
Mostrar a los participantes eficazmente	D. Comunicación
Mostrar los enlaces entre participantes	D. Comunicación. En el de secuencia son implícitos
Mostrar la información de los	Empate

mensajes	
Soporte para mensajes paralelos	D. Secuencia muestra mejor el paralelismo y cuenta con estructuras para moldearlo mejor
Facilidad para leer el orden de los mensajes	Este es su punto fuerte del D. Secuencia
Facilidad para crear y mantener el diagrama	Empate. Los dos pueden ser difíciles de mantener

Diagrama de Secuencia

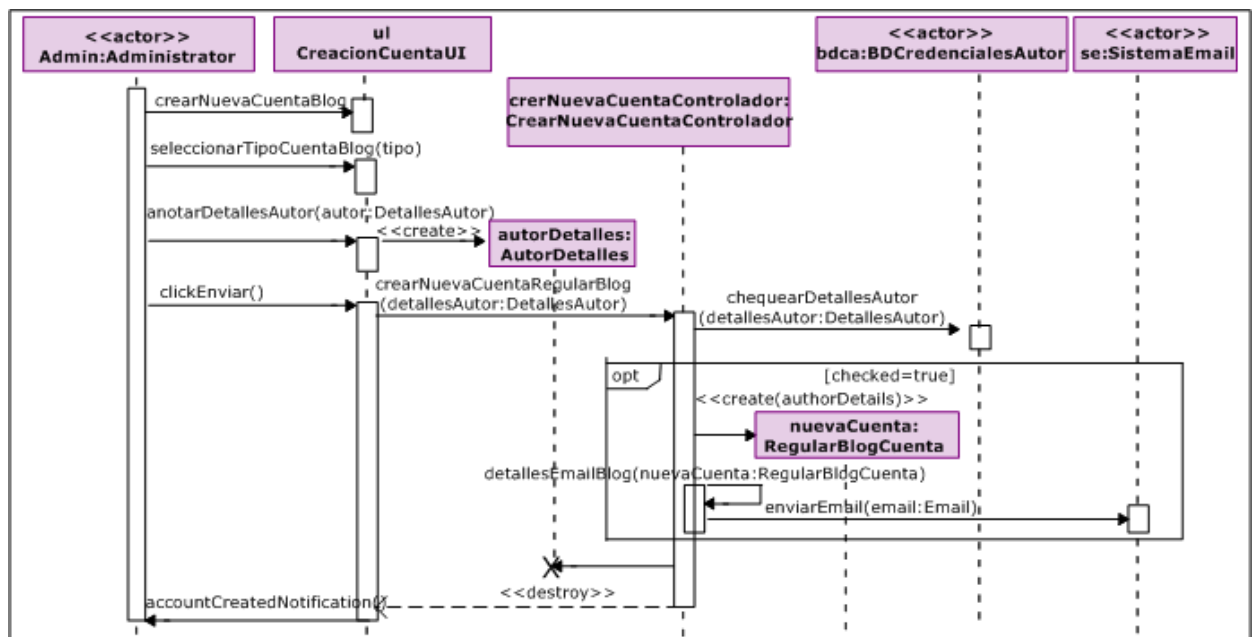
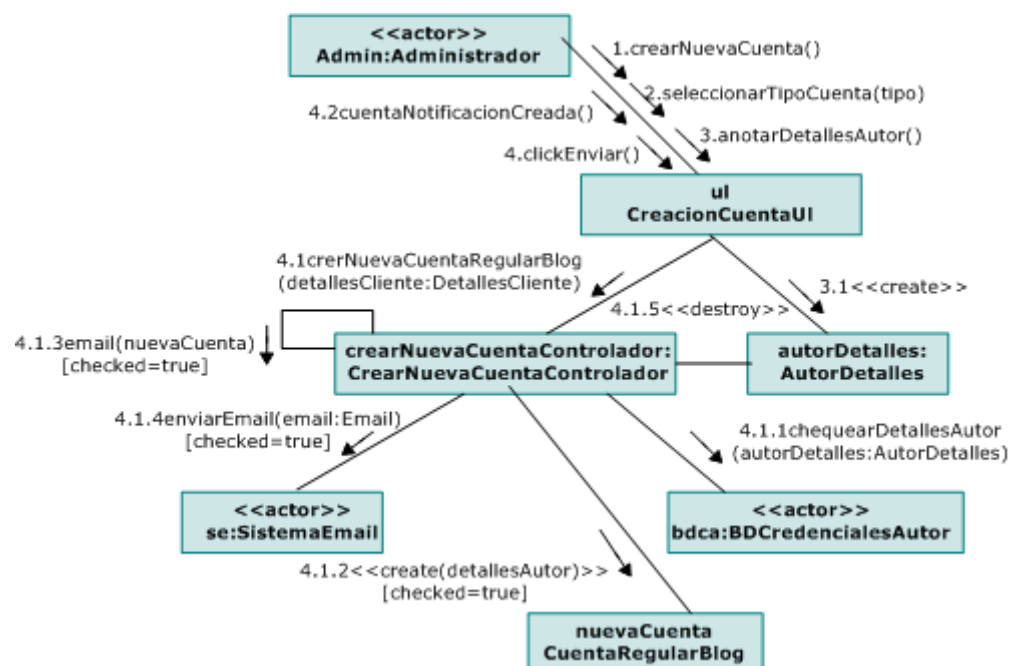


Diagrama de Comunicación



6.7 DIAGRAMA GENERAL DE INTERACCIÓN

Los **diagramas generales de interacción** proveen una vista de alto nivel mostrando como varias interacciones 'trabajan' juntas para implementar un aspecto del sistema (por ejemplo un caso de uso).

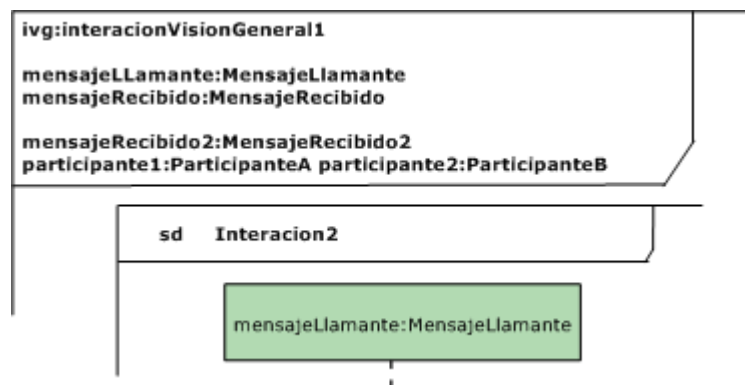
También se llama diagrama de [revisión, vista global o visión] de la interacción.

Se parece a un diagrama de actividades, sustituyendo a las acciones por referencias a las interacciones (a otros diagramas).

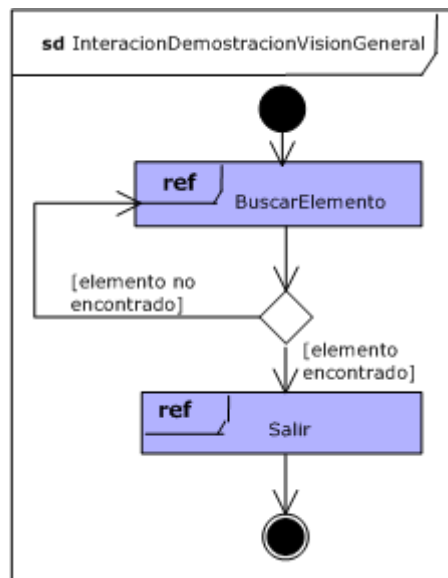
Los diagramas de secuencia, colaboración y tiempo modelan los mensajes específicos que conforman una interacción, y un diagrama general de interacción 'junta' varios de ellos para constituir una imagen completa de las interacciones.

Componentes del diagrama general de interacción:

- Los mismos de un diagrama de actividad (punto de inicio y fin, flujos, nodos de condición, etc) excepto las acciones.
- Las acciones son sustituidas por interacciones o referencias a otros diagramas.
- En el título del diagrama deben referenciarse las líneas de vida que contiene (todos los objetos o instancias de todos los participantes).



Ejemplo



6.8 DIAGRAMAS DE TIEMPOS

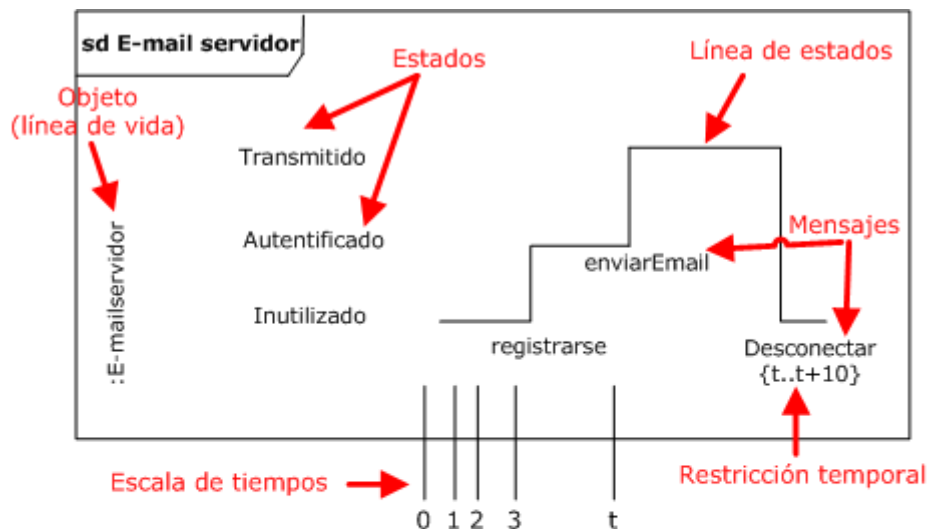
Los **diagramas de tiempos** son un caso especial de diagramas de interacción. Muestran el transcurrir de las interacciones a lo largo de una línea temporal indicando de forma precisa los mensajes intercambiados entre objetos, los cambios de estado que produce este intercambio de mensajes y las restricciones temporales impuestas al sistema.

Con otros diagramas no hay forma de detallar información sobre el tiempo.

Éste diagrama está generalmente asociado a sistemas en **tiempo real**, pero no está limitado a ellos.

En un diagrama de tiempos cada evento tiene información temporal asociada a él que describe:

- cuando es invocado
- cuanto tiempo pasa hasta que el otro participante recibe el evento
- cuanto tiempo el receptor estará en un determinado estado



El tiempo transcurre de **izquierda a derecha** y se puede mostrar la escala y unidad empleada.

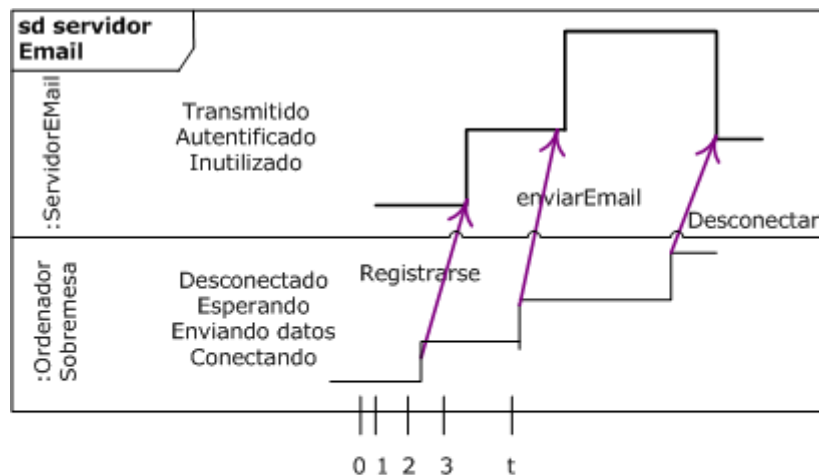
Se pueden usar tiempos absolutos, relativos (0, t, 2t, 3t, etc...) o combinados. La línea se denomina **línea de estado** porque muestra los estados por los que pasa el objeto con el transcurrir del tiempo.

Los **cambios de estado** en un objeto se producen como consecuencia de eventos (externos) o mensajes.

Las **restricciones de tiempo** describen de manera precisa cuanto puede durar una determinada porción de la interacción.

Ejemplos: $\{t..t+5s\}$, $\{<5s\}$, $\{>5s, <10s\}$, $\{t..t*5\}$

También añaden información completamente nueva, difícil de expresar con otros diagramas de interacción UML.



Puede haber **más de un objeto** en el mismo diagrama mostrándose de forma precisa:

1. los mensajes intercambiados entre ellos
2. los cambios de estado que estos provocan
3. el tiempo exacto que tardan en enviarse y recibirse los mensajes
4. todas las restricciones temporales asociadas

Notación alternativa:



Los estados se muestran mediante **hexágonos enlazados** (con el nombre dentro).

Se muestran como una lista.

Los eventos se muestran sobre las X, que son los cambios de estado.

Es útil cuando hay muchos estados que mostrar.

Permite ajustar la escala y reducir el tamaño.

7. DIAGRAMAS DE ACTIVIDAD Y ESTADO

7.1 DIAGRAMA DE ACTIVIDADES. CONCEPTOS BÁSICOS

Los casos de uso muestran lo que un sistema debería hacer. Los **diagramas de actividades** te permiten especificar **cómo** este sistema debe llevar a cabo sus objetivos.

Los diagramas de actividades **capturan acciones, su flujo y sus resultados**.

La **transición entre acciones** se produce **directamente**, sin esperar necesariamente a un evento.

Las acciones pueden agruparse en diferentes particiones para mostrar **quién es el responsable** de cada actividad.



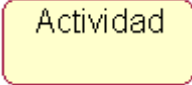

Los diagramas de actividades se usan para **describir el flujo de un sistema** expresando:

- **cómo y qué** hacen las acciones. (Cambios de estado de los objetos).
- **cuándo** tienen lugar las acciones. (Secuencia).
- **dónde** se realizan las acciones. (Particiones).

Los propósitos de los diagramas de actividades son los siguientes:

- **Capturar el trabajo** (acciones) que se realiza cuando se **ejecuta una operación** (instancia de la implementación de una operación).
- **Capturar el trabajo** interno de un **objeto**.
- Mostrar **cómo un conjunto de acciones relacionadas se realizan** y cómo afectan a los objetos involucrados.
- Mostrar **cómo una instancia de un caso de uso se lleva a cabo** en términos de acciones.
- Mostrar **procesos de negocio** en términos de trabajadores, flujos de trabajo, y objetos.

7.2 ACCIONES Y FLUJOS

Punto de inicio.	
Punto de finalización.	
Acción o actividad.	
Flujo o transición. Flecha dirigida.	

Flujo o Transición

En las transiciones se pueden detallar las siguientes expresiones:

- **Evento** (solo en el caso de las flechas que vayan desde el estado inicial a alguna acción). Indica el evento que inicia las acciones.
- **Condición de guardia**. Se evalúa una expresión booleana y si el valor de ésta es *true* se puede pasar a la siguiente actividad.
- **Acción**. Operaciones que se producen en la transición.
- **Cláusula de envío**. Es un tipo especial de acción utilizado para enviar un mensaje a otro objeto.

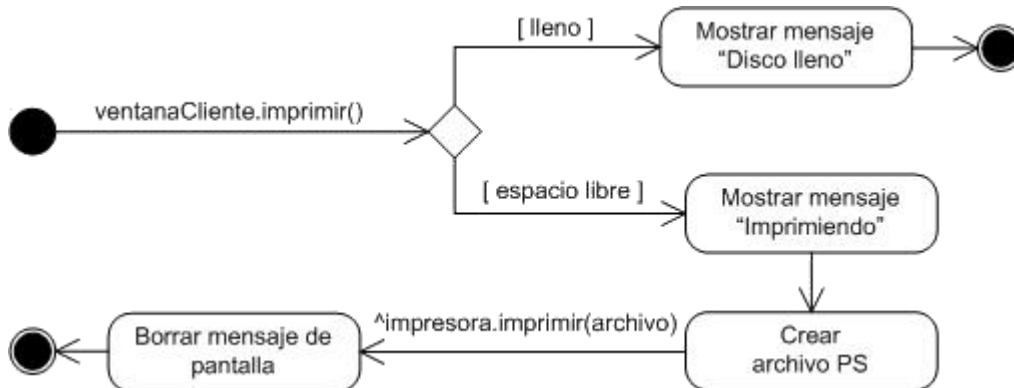
Notación de la transición:

evento [CGuardia] / Acción ^obj.mensaje(arg)

7.2.1 Decisiones

Este elemento se usa cuando el **flujo puede tomar dos o más alternativas**.

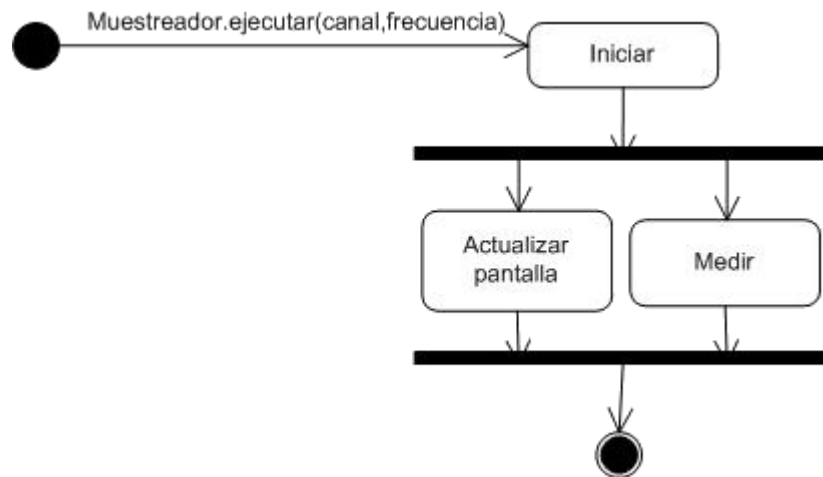
Se representa mediante un **rombo** y **condiciones de guardia** en cada una de las salidas.



7.2.2 Acciones paralelas

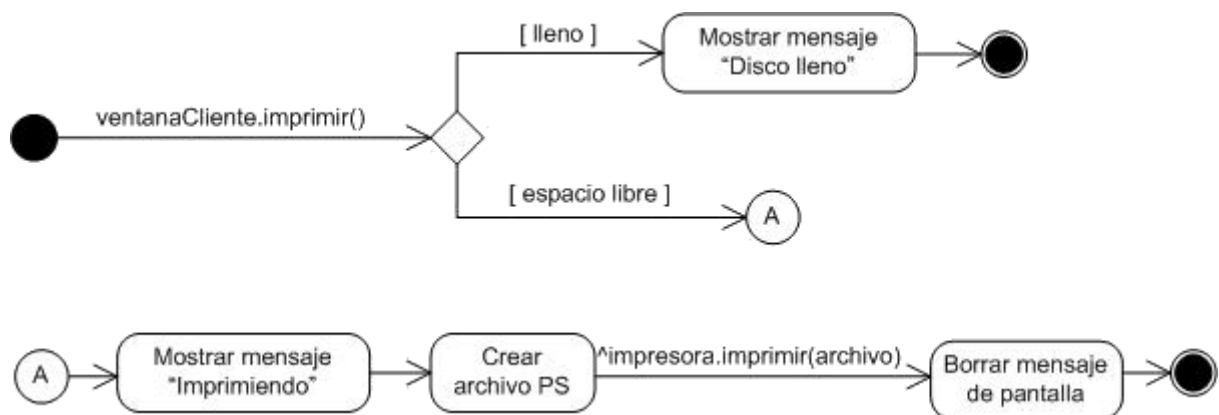
Podemos expresar la **ejecución concurrente de dos o más acciones**.

Una **línea en negrita** representa la división del flujo y otra la unificación.



7.2.3 Conectores

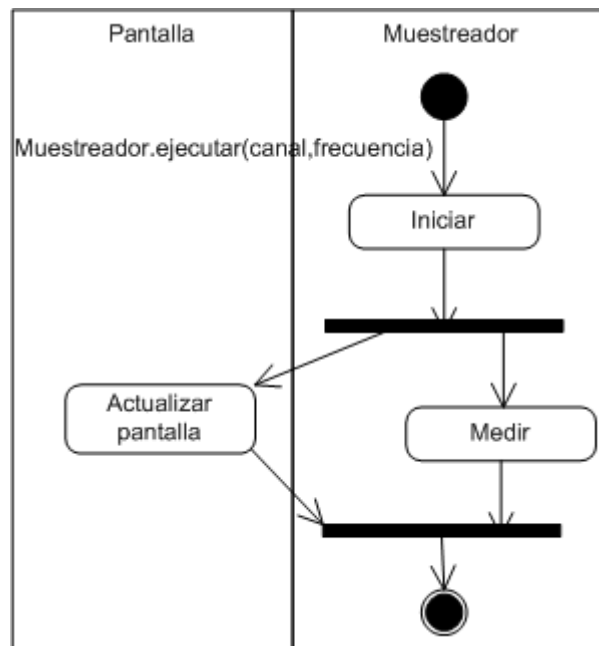
Cuando las flechas complican la legibilidad del diagrama, se usan conectores para simplificarlo. No añaden semántica.



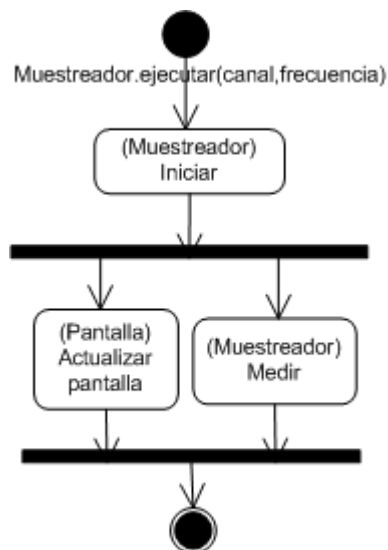
7.3 PARTICIÓN DE ACTIVIDADES

Las **acciones** pueden agruparse en diferentes particiones para mostrar quién es el responsable de cada actividad.

- Las **particiones** se muestran como "calles" representadas con rectángulos verticales.
- Las **acciones** se sitúan en sus correspondientes calles.

**NOTA:**

Se indica el nombre de la calle entre paréntesis dentro de la actividad y no se representan las calles.

Ejemplo**Notación alternativa:**

Las particiones también se pueden organizar de otras formas: de izquierda a derecha, descomponerse en "sub-calles", en tablas, etc.

7.4 OBJETOS, SEÑALES Y PINS

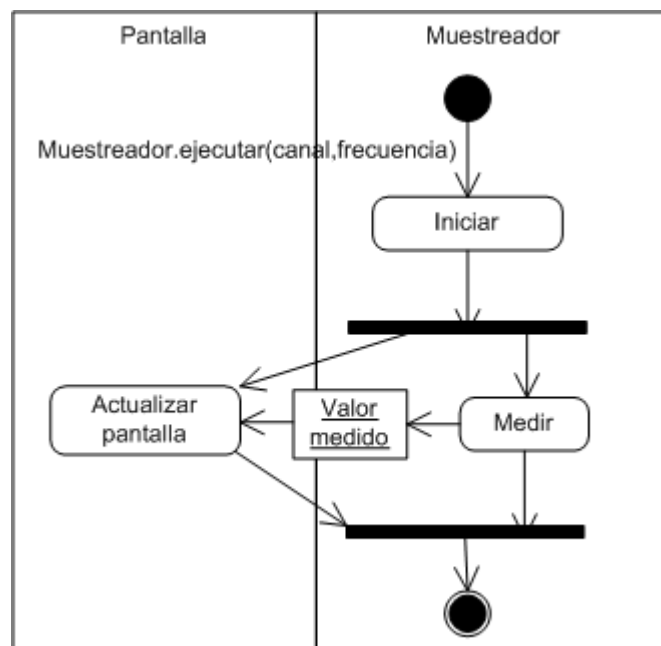
7.4.1 Objetos

Pueden aparecer **objetos** en un diagrama de actividades.

Estos objetos se pueden usar:

- como **entrada y/o salida de las acciones**. (Se representa con una flecha entre el objeto y la acción).
- para mostrar que un **objeto se ve afectado por una acción**.

Se representa mediante un **rectángulo** indicando el nombre del objeto, y opcionalmente la clase y el estado entre [].



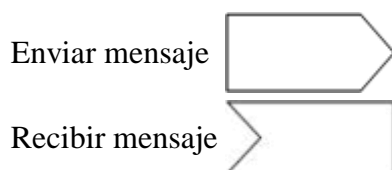
7.4.2 Señales

Las **señales** sirven para representar el **envío y recepción de mensajes**.

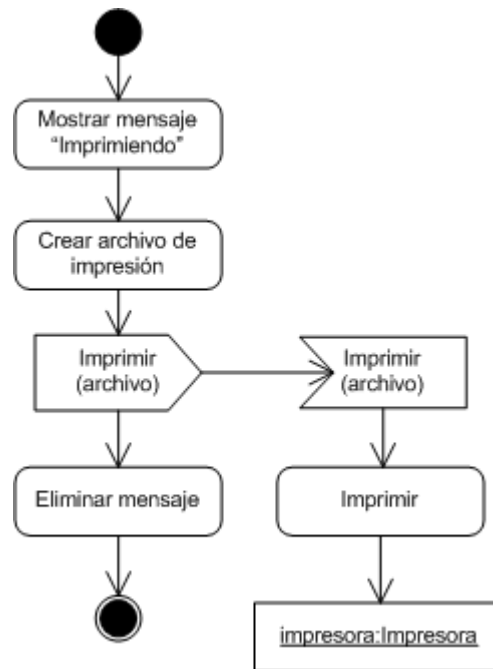
Los mensajes pueden enviarse y recibirse entre:

- **actividad – actividad.**
- **actividad – objeto.**

Se representan con pentágonos y flechas para denotar el flujo.



Ejemplo

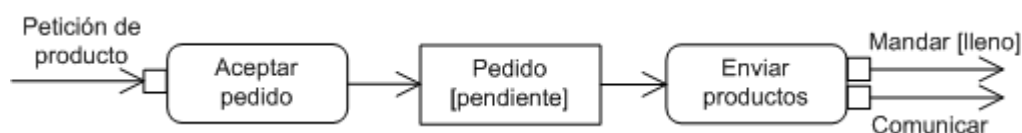


7.4.3 Pins

Los **pins** permiten formalizar la **semántica de inicio y final** de una actividad.

Es una forma de añadir más información al diagrama indicando la información que necesita y/o genera cada acción.

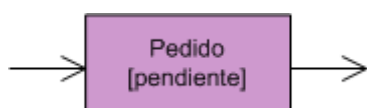
Un **pin** se representa con un **cuadrado pequeño** junto a la acción.



Hay pins:

- **de entrada:** Indica los valores que la actividad acepta. (Presentan prerequisites para que la actividad se ejecute).
- **de salida:** Indica los valores que la actividad produce.

Si un pin de salida, es a la vez pin de entrada de la siguiente actividad se puede representar de la siguiente manera:



7.5 MÁQUINA DE ESTADOS. INTRODUCCIÓN

Un **estado** es el resultado de las acciones previas realizadas por el objeto, y **queda determinado por el valor de algunos de sus atributos y sus enlaces con otros objetos**. (Una clase puede tener un atributo específico para determinar el estado). **Todos los objetos tienen estados**.

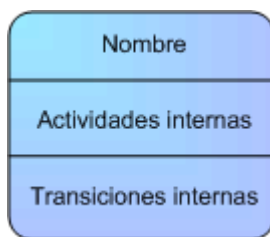
Las **transiciones** cambian el estado del objeto cuando se produce un **evento**.

El **diagrama de máquina de estados** muestra el conjunto de estados por los cuales pasa un objeto durante su vida en una aplicación, junto con los cambios que permiten pasar de un estado a otro.

Notación básica



Los estados pueden mostrar 3 compartimentos destinados a:



Compartimento actividades internas

Las **actividades internas** muestran el comportamiento en respuesta a ciertos eventos.

Se pueden definir eventos propios o usar los eventos estándar de UML.

Su sintaxis es: nombre_evento lista_argumentos / acciones.

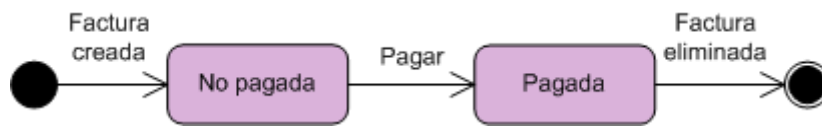
Compartimento transiciones internas

Un estado puede tener **sub-estados**.

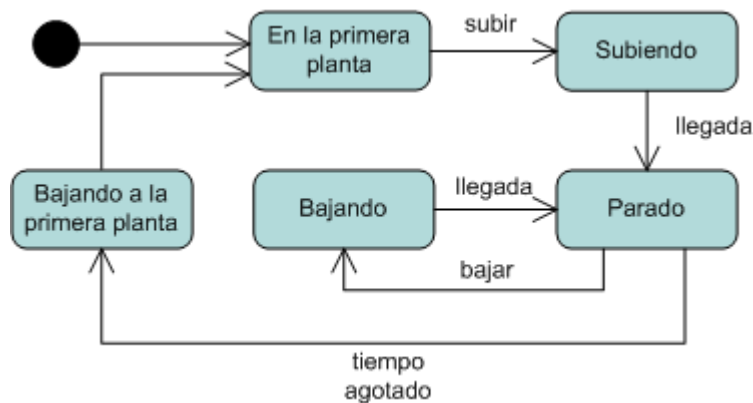
Las **transiciones internas** determinan el funcionamiento interno de estos sub-estados.

Su sintaxis es: `signatura_evento [condición_guarda] / acción ^ cláusula_envío`

Ejemplo 1



Ejemplo 2



7.6 TRANSICIONES

La transición entre estados puede realizarse debido a un evento o también debido a acciones internas del estado.

- **Con evento:** Si la transición está asociada a un evento, ésta se realiza cuando se produce dicho evento.
- **Sin evento:** Si la transición no está asociada a ningún evento, el cambio de estado ocurre cuando las acciones del interior del estado origen finalicen. En caso de que no haya acciones a realizar dentro del estado, la transición se produce instantáneamente.



7.6.1 Sintaxis de los eventos

signatura-evento [condición-guarda] / acción ^ cláusula-envío

Signatura de los eventos

Es el nombre del evento.

Especifica el evento que provoca la transición entre los estados.

Además puede contener un listado de parámetros con sus tipos.

Ejemplos:

- dibujar (figura, color)
- clic ()
- pagarRecibo (recibo)

Condición de guarda

La condición de guarda es una expresión booleana situada en una transición de estado.

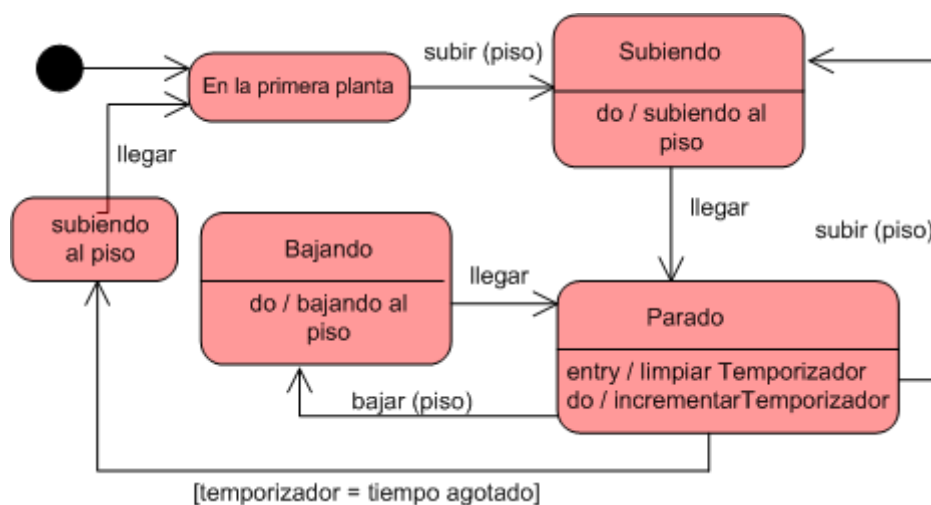
Tienen dos usos posibles:

1. **Combinada con un evento:** Cuando se produce el evento, se evalúa la condición, y sólo se cambia de estado en el caso de que la condición sea verdadera.
2. **Se encuentra sola en la transición:** El cambio de estado se produce en el momento que la condición se vuelve verdadera.

Se representa **entre corchetes** [].

Ejemplos:

- [t=15]
- [número de recibos>n]
- reintegro(cantidad) [cantidad<totalDisponible]



Acción o Expresión de acción

Una **acción** es una expresión procedural que se ejecuta cuando la transición se dispara.

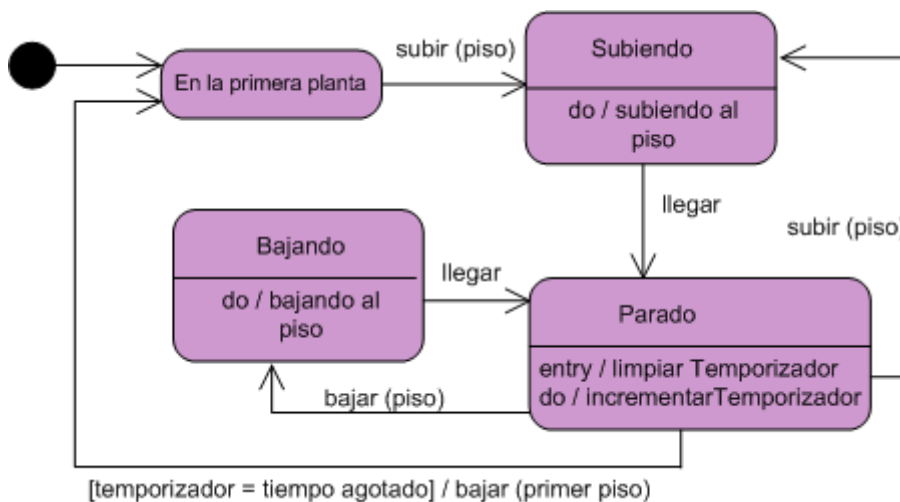
Estas acciones se escriben usando:

- atributos del propio objeto.
- parámetros indicados en la signatura del evento.

Las acciones se indican con una barra '/' delante de cada acción. Si hay varias acciones, se ejecutan en el mismo orden que aparecen.

Ejemplos:

- incrementar() / n:= n+1 / m:=m+1
- sumar(n) / sum:=sum+1
- /flash



Cláusula de envío

La cláusula de envío es un caso especial de acción. Sirve para enviar un mensaje a otro objeto cuando se produce una transición entre dos estados.

Sintaxis

Es la misma que la llamada a cualquier método y se indica con el carácter '^' delante de la cláusula.

Ejemplos:

- sinPapel() ^indicador.luz()
- click(sitio) /color:=capturarColor(sitio) ^lapiz.set(color)
- [timer=time out] ^self.goDown(firstFloor)

7.7 ENVÍO DE MENSAJES ENTRE MÁQUINAS Y ESTADOS COMPUESTOS

7.7.1 Envío de mensajes entre máquinas

Las **máquinas de estados** pueden enviarse **mensajes** entre ellas.

Estos mensajes pueden expresarse mediante:

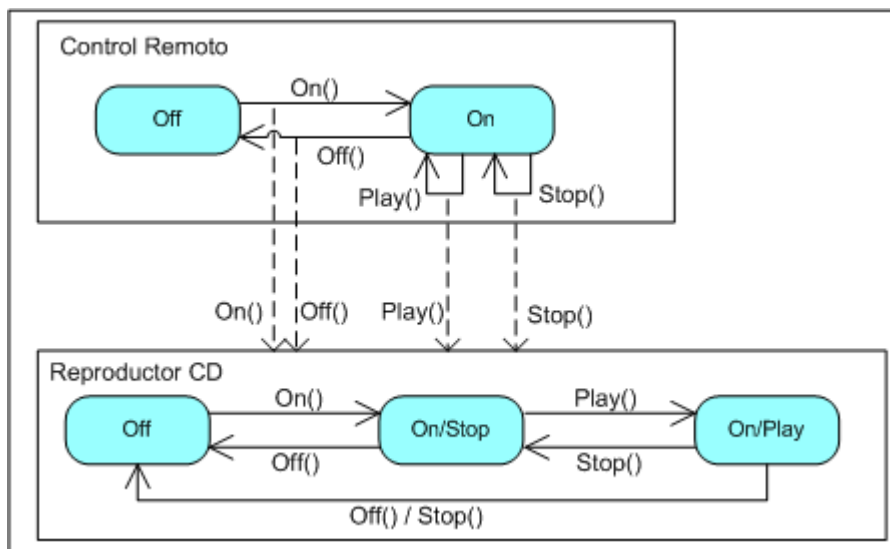
- **cláusulas de envío** expresadas en las transiciones.
- **líneas discontinuas** entre las máquinas de estados.

Existen dos técnicas, dibujar una flecha:

- Desde la transición de la máquina origen hasta el borde la máquina destino.
- Desde el borde del objeto origen hasta el borde del objeto destino.

(En ambos casos debe haber una transición en la máquina destino que corresponda y capture este evento).

Ejemplo



7.7.2 Estados compuestos

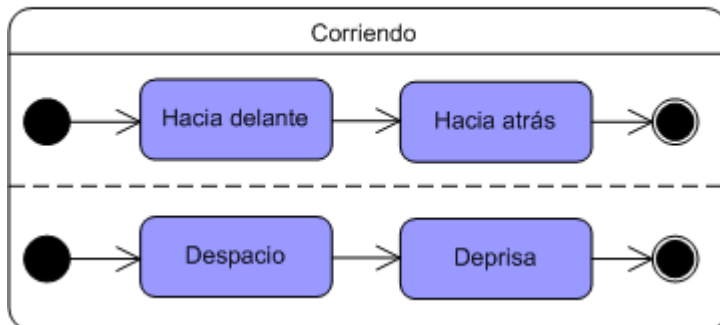
Un estado puede componerse de **sub-estados** que pueden expresarse como una máquina de estados. Para ello se sitúan los sub-estados dentro del estado compuesto en un segundo compartimento debajo del nombre.

UML define dos tipos de sub-estados:

- **Ortogonal o subestado-o** (or-subestate). El estado tiene varios subestados, pero el sistema u objeto solo puede estar en uno de ellos en un determinado momento.
- **No-ortogonal o subestado-y** (and-subestate). El estado puede tener **múltiples estados concurrentes**. Se puede usar una línea discontinua para marcar los 'caminos' de concurrencia.

Ejemplos

Ejemplo subestado-o



Ejemplo de estado compuesto

