UD 05 - DISEÑO ORIENTADO A OBJETOS Y ELABORACIÓN DE DIAGRAMAS

1. Introducción a la orientación a objetos

En la construcción del software podría uno decantarse por un

- **enfoque estructurado**: El problema de partida se toma y se somete a un proceso de división en subproblemas más pequeños hasta llegar a problemas elementales resolubles utilizando una función. Luego estas funciones se entretejen para formar solución global (proceso centrado en procedimientos, codificando mediante funciones que actúan sobre estructuras de datos). Se intenta aproximar qué hay que hacer para resolver un problema.
- **enfoque orientado a objetos**: se simulan los elementos de la realidad asociada al problema de la forma más cercana posible mediante una abstracción llamada objeto (conjunto de atributos por los cuáles se caracteriza y de operaciones que definen su comportamiento. Las operaciones actúan sobre los atributos para modificar su estado. Cuando se indica a un objeto que ejecute determinada operación se dice que se le pasa un mensaje)

Las **aplicaciones orientadas a objetos** están formadas por un conjunto de objetos que interaccionan enviándose mensajes para producir resultados. Los objetos similares se abstraen en clases siendo un objeto una instancia de una clase. Al ejecutar el programa:

- Se crean los objetos a medida que se necesitan
- Los mensajes se mueven de un objeto a otro (o del usuario al objeto) a medida que el programa procesa información o responde a la entrada del usuario
- Cuando los objetos no se necesitan se borran y se libera la memoria

Como **ventajas** frente a otros paradigmas podría destacarse:

MODULARIDAD, ENCAPSULACIÓN, REUTILIZACIÓN, ESCALABILIDAD

- Puede desarrollarse software en mucho menos tiempo, menos coste y más calidad gracias a la reutilización de código (código modular, reusable en aplicaciones similares...)
- La calidad de los sistemas se aumenta, haciéndolos más extensibles (sencillo aumentar o modificar funcionalidad)
- Más fácil de modificar, más mantenible por sus criterios de modularidad y encapsulación
- Mayor adaptación al entorno, haciendo aplicaciones escalables. Sencillo modificar estructura y comportamiento sin cambiar la aplicación.

1.1. Principios de la orientación a objetos

- **Abstracción**: Se capturan las características y comportamientos similares de un conjunto de objetos para darles una descripción formal -> Se arma un conjunto de clases que permiten modelar la realidad.
- **Encapsulamiento / Encapsulación**: Reunir todos los elementos que puedan considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción (No confundir con ocultación)
- **Modularidad**: Subdividir una aplicación en partes más pequeñas (módulos), cada una de las cuales es tan independiente entre sí y del resto de la aplicación como sea posible. Es algo consubstancial en la POO: Los objetos son los módulos más básicos del sistema.
- **Principio de ocultación**: Se aíslan las propiedades del objeto para evitar su modificación por quién no tenga derecho a acceder a ellas (evita propagación de efectos colaterales con los cambios)
- **Polimorfismo**: Se reúnen bajo el mismo nombre comportamientos diferentes
- **Herencia**: Relación entre objetos en los que unos usan propiedades y comportamientos de otros formando una jerarquía. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.

- **Recolector de basura**: El entorno de objetos se encarga de destruirlos automáticamente y de desvincular su memoria asociada cuando ha quedado sin ninguna referencia a ellos.

1.2. Clases, atributos y métodos

Los objetos del sistema se abstraen en **clases**: Conjunto de procedimientos y datos que resumen características similares de un conjunto de objetos. /// Abstracción que define las características comunes de un conjunto de objetos relevantes para el sistema. // Abstracciones del dominio del sistema que representan elementos del mismo mediante una serie de características (atributos) y de comportamientos (métodos)

El objeto es único debido a los valores que tienen asignados sus atributos. La clase define sus características generales y comportamiento.

Sus propósitos son **definir abstracciones** y **favorecer la modularidad**.

Los **miembros** de una clase son:

- **Nombre**: Identificativo. Se relaciona con lo que representa.
- **Atributos**: Características asociadas a la clase. Puede ser una relación binaria entre la clase y un dominio formado por todos los valores que puede tomar cada atributo. Al tomar valores válidos de este dominio definen el **estado** de un objeto. Los atributos se definen por su nombre y por su tipo (pudiendo ser simple o compuesto como otra clase).
- **Protocolo**: Operaciones (métodos, mensajes) que manipulan el estado:
  - Método: Procedimiento o función que se invoca para actuar sobre un objeto. Determina cómo actúan los objetos cuando reciben un mensaje (cuando se requiere que el objeto realice una acción descrita en un método se le envía un mensaje). El conjunto de mensajes a los que puede responder un objeto se conoce como *protocolo* del objeto.
  - Mensaje: Resultado de la acción efectuada por un objeto

Mencionar la **clase abstracta** que es aquella que no puede ser instanciada. Es usada para definir métodos de base para clases derivadas o para definir métodos genéricos relacionados con el sistema que no serán traducidos a objetos concretos (raramente)

1.3. Visibilidad

La visibilidad son los niveles de ocultación posibles (tipo de acceso a atributos y métodos)

- Público `public`: Desde cualquier clase
- Protegido `protected`: Solo operaciones de la clase o clases derivadas (herencia)
- Paquete `default`: Solo operaciones de clase del mismo paquete.
- Privado `private`: Solo operaciones de la clase

Debido al *principio de ocultación* (el estado se aísla de forma que solo pueda cambiarse mediante las operaciones definidas en una clase, evita la modificación por quién no tiene derecho a ello) las clases se dividen en dos partes:

- Interfaz: Visión externa de una clase (abstracción del comportamiento común a los ejemplos de esa clase)
- Implementación: Cómo se representa la abstracción; mecanismos que conducen al comportamiento deseado.

A la hora de definir la visibilidad debe tenerse en cuenta:

- El **estado debe ser privado**. Atributos deben modificarse mediante métodos creados a tal efecto.
- Las **operaciones que definen la funcionalidad de la clase deben ser públicas**
- Las **operaciones que ayudan a implementar parte de la funcionalidad** deben ser **privadas** o **protegidas*

1.4. Objetos. Instanciación

Instancia de clase: Construir un objeto en un programa informático a partir de una clase. Cada instancia es modelo de un objeto del contexto del problema relevante para su solución que puede hacer un trabajo, informar, cambiar su estado y "comunicarse" con otros objetos del sistema, sin revelar cómo se implementan estas características.

Objetos pueden ser... objetos físicos (aviones, casas, parques); elementos UI (menús, teclados, cuadros de diálogo); animales (vertebrados, invertebrados); alimentos (carnes, frutas ,verduras)...

El objeto se define por:

- Su estado (concreción de los atributos de la clase a un valor concreto)
- Su comportamiento (definido por los métodos públicos de su clase)
- Su tiempo de vida: Intervalo de tiempo a lo largo del programa en el que el objeto existe (desde su creación por el mecanismo de instanciación hasta su destrucción)

2. UML. Definición, elementos, clasificación

Unified Modeling Language (UML) o lenguaje unificado de modelado es un conjunto de herramientas que permite MODELAR, CONSTRUIR y DOCUMENTAR los elementos que forman parte de un sistema de software POO.

> Los desarrolladores pueden visualizar el producto de su trabajo en una colección de esquemas o diagramas estandarizados llamados **modelos** que representan el sistema desde diferentes perspectivas

Fue concebido por los autores de los tres métodos más usados de la POO (Booch (Método Booch), Jacobson (OOSE Object-Oriented Software Engineering) y Rumbaugh (OMT Object Modeling Technique))

Es útil porque:

- permite usar un **lenguaje común** para comunicarse en los equipos de desarrollo
- pueden **documentarse todos los artefactos** (toda información utilizada o producida mediante un proceso de desarrollo de software: requisitos, arquitectura, pruebas, versiones) disponiéndose de información que trasciende al proyecto
- **permite indicar información sobre estructuras que trascienden lo representable en un lenguaje de programación**, como la arquitectura del sistema, y con UML pueden indicarse incluso qué módulos de software van a desarrollarse y sus relaciones o en qué nodos hardware se ejecutarán cuando se trata de sistemas distribuidos
- puede conectarse a lenguajes de programación mediante **ingeniería directa** (transformación de modelo en código) e **ingeniería inversa** (transformación del código en un modelo)

2.1. Herramientas para la generación de diagramas UML

Desde lápiz y papel hasta herramientas CASE que suelen contar con ventanas wysiwyg (*What You See Is What You Get*). No solo generan los diagramas asociado al análisis y al diseño de la aplicación, también sirven para documentar los diagramas, integrarse con otros entornos, generar código automáticamente y procesos de ingeniería inversa generando diagramas a partir del código fuente o incluso binario. Mencionamos:

- **Rational Systems Developer de IBM**. **IBM Rational Rhapsody Developer**. Inicialmente creadores UML, después absorbida por IBM. Tiene versiones de prueba y software libre.
- **Visual Paradigm for UML**. Tiene versión para uso no comercial simplemente registrándose para conseguir un archivo de licencia. Incluye módulos para UML, diseño de bases de datos, Agile, ingeniería inversa y es multiplataforma y compatible con Eclipse, VisualStudio, IntelliJ IDEA, NetBeans...
- **ArgoUML**. Con licencia Eclipse. Genera código para Java y C++. Se ejecuta con Java. Tiene ingeniería directa e inversa.

2.2. Elementos de un diagrama UML

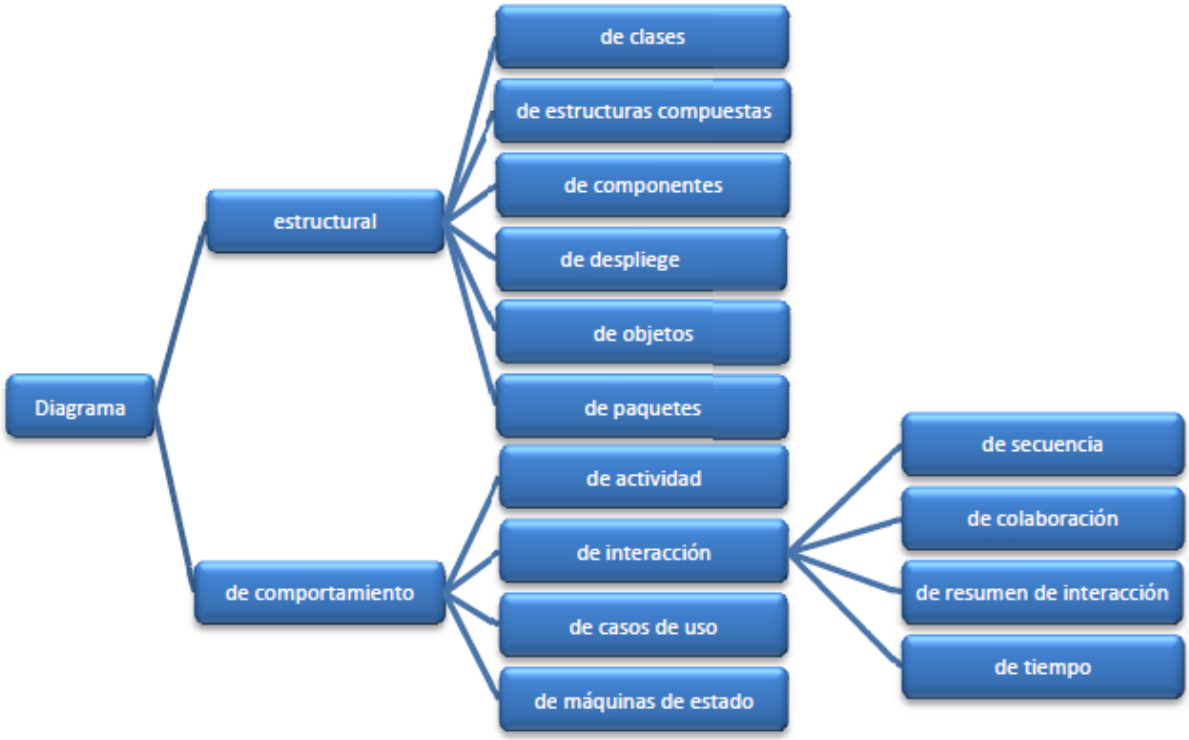
Los modelos suelen ser un grafo conexo de arcos (relaciones) y vértices (elementos del modelo).

- **Estructuras**: Nodos del grafo. Describen el tipo de diagrama
- **Relaciones**: Arcos del grafo que se establecen entre los elementos estructurales

- **Notas**: Cuadro donde se pueden escribir comentarios para entender algún concepto que se quiere representar
- **Agrupaciones**: Al modelar sistemas grandes se usan para separar su desarrollo por bloques

2.3. Clasificación de diagramas UML

Como ejemplo, podrían describirse 13 diagramas para modelar diferentes aspectos de un sistema, aunque no es necesario usarlos todos; dependerá del tipo de aplicación a generar y del sistema, solo debe generarse un diagrama si es necesario.



Existen dos grandes grupos:

- **Diagramas estructurales**: Visión estática del sistema (modelo estático abstracto). Especifican clases y objetos y cómo se distribuyen físicamente en el sistema.
  De prioridad **ALTA**

  - ***Diagrama de clases**: Elementos del modelo estático abstracto. Formado por el conjunto de clases y sus relaciones.
  - **Diagrama de objetos**: Elementos del modelo estático en un momento concreto. Formado por el conjunto de objetos y de sus relaciones. Habitualmente usado para casos especiales de un diagrama de clases o de comunicaciones.

  De prioridad **MEDIA**

  - **Diagrama de componentes**: Organización lógica de la implementación de una aplicación, sistema o empresa, indicando componentes, interrelaciones, interacciones, interfaces públicas y dependencia entre ellos.
  - **Diagrama de despliegue**: Configuración del sistema en tiempo de ejecución. Con nodos de procesamiento y sus componentes. Ejecución de la arquitectura del sistema: Nodos, ambientes operativos (hardware o software) e interfaces que las conectan. Muestra cómo los componentes de un sistema se distribuyen en los ordenadores que los ejecutan. Se usa en sistemas distribuidos.

  De prioridad **BAJA**
  - **Diagrama de paquetes**: Organización de los elementos del modelo en paquetes y dependencias entre estos paquetes. Útiles para sistemas de mediano o gran tamaño.

    - **Diagrama de estructuras compuestas / diagrama integrado de estructuras (UML 2.0)**: Estructura interna de una clasificación (clase, componente, caso típico) con los puntos de interacción de esta clasificación con otras partes del sistema.
- **Diagramas de comportamiento**: Conducta en el tiempo de ejecución del sistema, tanto desde el punto de vista del sistema completo como de las instancias u objetos que lo integran.

  De prioridad **ALTA**

    - ***Diagramas de actividad**: Orden en el que se van realizando las tareas dentro de un sistema. Procesos de alto nivel de la organización. Incluye flujo de datos o modelo de la lógica compleja dentro del sistema.
    - **Diagramas de interacción- De secuencia**: Ordenación temporal en el paso de mensajes. Modela la secuencia lógica a través del tiempo de los mensajes entre instancias.

  De prioridad **MEDIA**

    - ***Diagramas de casos de uso**: Acciones a realizar en el sistema DESDE EL PUNTO DE VISTA DE LOS USUARIOS SIN FORMACIÓN EN DESARROLLO. Acciones, usuarios y relaciones entre ellos. Especifica la funcionalidad y el comportamiento mediante su interacción con usuarios y/o sistemas.
    - **Diagramas de máquinas de estado**: (Diagrama de estado, diagramas de estados y transiciones, diagrama de cambio de estados) Comportamiento de un sistema dirigido por eventos. Estados que puede tener un objeto o interacción y transiciones entre estados.

  De prioridad **BAJA**

  Diagramas de interacción
    - ***De comunicación/colaboración (UML 2.0)**: Organización estructural de los objetos que se reciben y envían mensajes. Instancias de clases, interrelaciones, flujo de mensajes entre ellas.
    - ***De interacción**: Conjunto de objetos y relaciones junto con los mensajes que se envían entre ellos. Es variante del diagrama de actividad que permite mostrar el flujo de control dentro de un sistema o proceso organizativo. Cada nodo de actividad puede representar otro diagrama de interacción.
    - ***De tiempo**: Cambio de un estado o una condición de una instancia o un rol a través del tiempo. Se usa para exhibir el cambio en el estado de un objeto en el tiempo, en respuesta a eventos externos.

3. Diagramas UML estructurales -> Diagramas de clases

- El más importante por representar el modelo estático del sistema, atributos, comportamientos y relaciones entre ellos. A partir de él se generarán las clases que formarán el programa informático que solucionará el problema.
- Contiene las clases de **dominio del problema** (dominio: área de conocimiento caracterizada por un conjunto de conceptos y terminología comprendida por los practicantes del dominio; dominio del problema: dominio sobre el que se define el problema a resolver por el sistema que se va a generar)
- Se componen de:
    - Clases:
    - Relaciones: Que pueden ser de asociación, agregación y herencia. Ej.: Clases "Persona" y "Coche" se establece relación CONDUCE.
    - Notas: Cuadro con comentarios para entender conceptos mejor
    - Elementos de agrupación: Se modela sistema grande agrupando clases y relaciones en paquetes que se relacionan entre sí

Clase

Un rectángulo dividido en tres filas: Primero el nombre de la clase; Segundo sus atributos con visibilidad; Tercero sus métodos con visibilidad. (-) privados; (+) públicos. Solo sería obligatorio definir el nombre, no los métodos, ni los atributos.

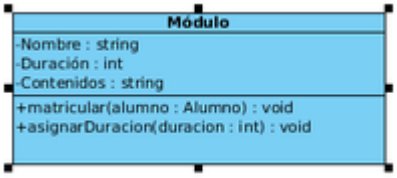
Atributos

Los atributos tienen que tener **nombre**, **tipo**, **visibilidad**. Puede indicarse un valor inicial.

Métodos

Necesario **nombre**, **parámetros**, **tipo de retorno**, **visibilidad**. Puede indicarse una descripción del método que aparecerá en la documentación.

El **método constructor** no devuelve nada y tiene el mismo nombre de la clase. Se usa para ejecutar acciones necesarias al instanciar la clase. Para destruir el objeto (según el lenguaje) también podrá existir un **método destructor** (En Java no).



Relaciones entre clases

Relación: Conexión entre dos clases que se incluye en el diagrama cuando hay algún tipo de relación entre ellas en el dominio del problema.

Estas relaciones no siempre es necesario que aparezcan especificadas en la lista de requisitos del problema. Es posible que no aparezcan específicamente pero se infieran (como las relaciones de herencia)

Las relaciones se representan como una línea continua. Los mensajes se envían entre objetos de clases relacionadas (en varias direcciones o quizás solo en una).

Las relaciones se caracterizan por la **cardinalidad** o **multiplicidad de clases** que representa cuántos objetos de una clase pueden verse involucrados en la relación.

Cardinalidad	Significado
1	Uno y solo uno
0..1	Cero o uno
N..M	Desde N hasta M
*	Varios
0..*	Cero o varios
1..*	Uno o varios (al menos uno)

Aquí por ejemplo:

- un alumno puede MATRICULARSE de 1 o más módulos (miro a la derecha de la relación)
- un módulo puede TENER MATRICULADOS 0 o más alumnos (miro a la izquierda de la relación)

Relación de herencia

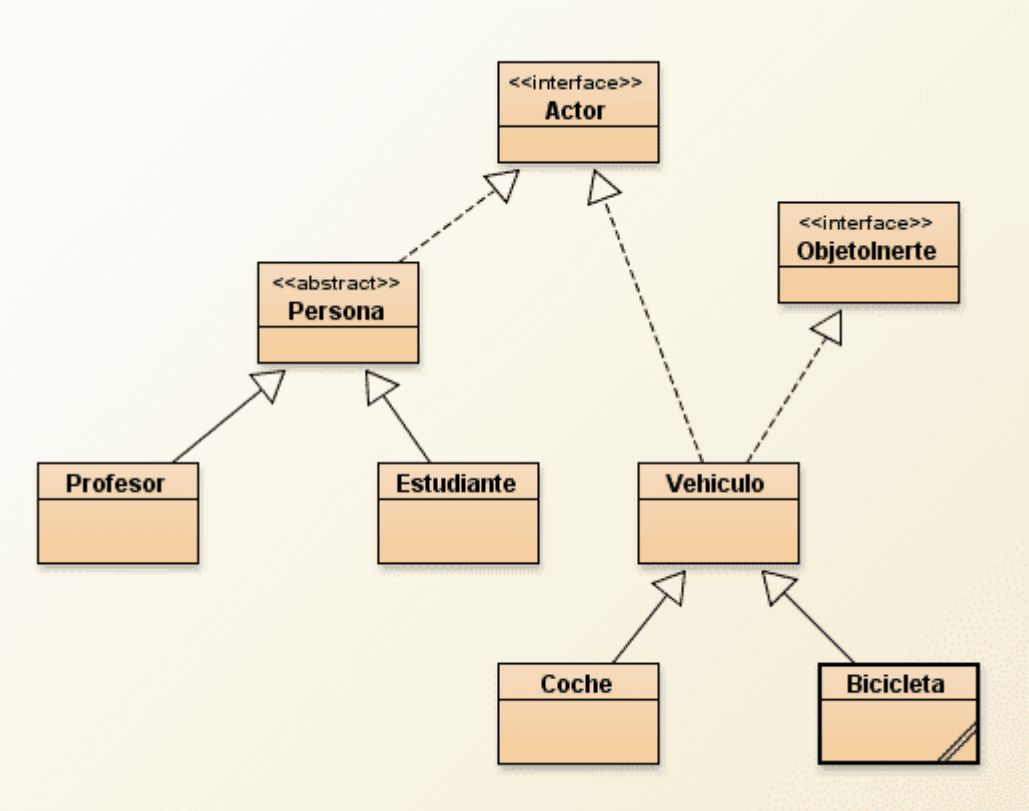
Herencia: Capacidad de un objeto para usar estructuras de datos y métodos presentes en sus antepasados. Permitiendo la reutilización de código. Pueden añadir su propio código especial y datos e incluso cambiar los que necesiten que sean diferentes.
Puede haber:

- Herencia simple: Un ascendente

- Herencia múltiple: Varios ascendentes (No permitido en Java)

Los atributos métodos y relaciones se muestran en el nivel más alto de la jerarquía en el que son aplicables.
La relación de herencia se representa en el diagrama como un triangulo en el extremo de la clase base (clase de la que hereda)

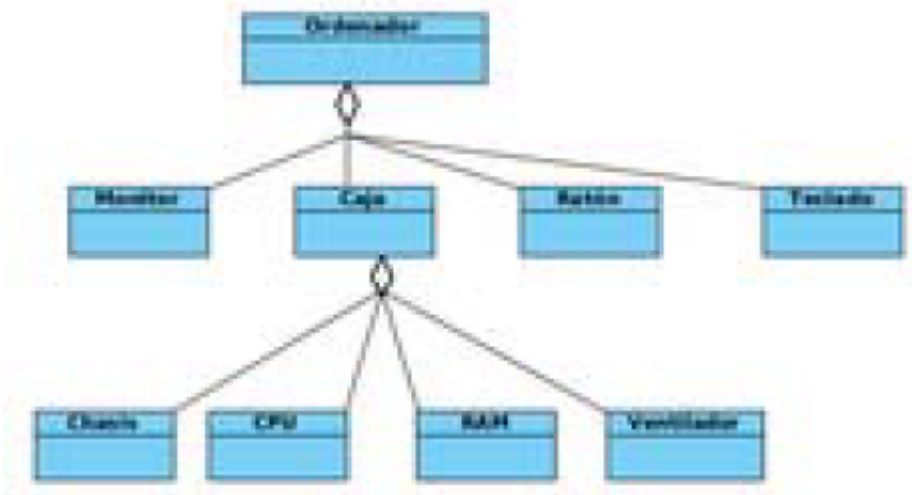
Cuando se **implementan** interfaces (caso de Java) se representa también con un triángulo en el extremo de la interfaz y la línea de la relación se pone punteada.



Relación de agregación y composición

Agregación: **Elementos pueden existir sin el elemento contenedor**. Asociación binaria todo-parte (pertenece a, tiene un, es parte de). El tiempo de vida de los objetos no tiene por qué coincidir. Ej.: Ordenador compuesto de piezas sueltas con entidad por sí mismas.

Se representa con un rombo en el extremo de la entidad contenedora. En agregación es blanco.

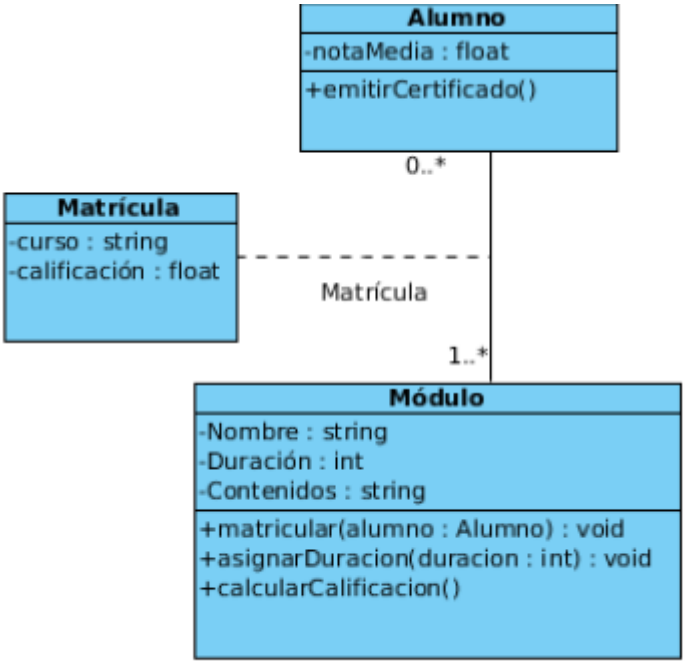


Composición: **Unos no pueden existir sin otros. Cuando el objeto "todo" es eliminado, también se eliminan los objetos "parte"**. Agregación fuerte en la que una "parte" está relacionada como máximo con un "todo" en un momento dado. Ej.: Rectángulo y vértices. Centro comercial y tiendas.

Se representa con un rombo en el extremo de la entidad contenedora. En composición es negro.

Atributos de enlace

Para completar información de alguna forma pueden añadirse atributos a la relación. Ej.: Alumno ---- Curso y una relación Matrícula con atributos



Paso de requisitos del sistema al diagrama de clases

OBTENCIÓN DE CLASES

Partiendo de los requisitos del sistema, situar en una tabla:

- Nombres (sinónimos) que se refieran a clases
- Entidades externas: Que producen o consumen información (sistemas, dispositivos, personas) que usará la aplicación
- Ocurrencias o sucesos que ocurren en una operación del sistema
- Papeles o roles de personas que interactúan con el sistema
- Lugares que establecen el contexto del problema y la función del sistema
- Estructuras que definen una clase de objetos o clases relacionadas de objetos

No incluir en la lista cosas que no sean objetos (como operaciones de otro objeto que darían lugar a un método).

Después, estudiar cada objeto de la tabla para ver si pasará al diagrama de clases, cosa que se hará si cumple todos o muchos de estos criterios:
1.-La información del objeto es necesaria para que el sistema funcione
2.-El objeto tiene un conjunto de atributos que pueden encontrarse en cualquier ocurrencia del objeto. Si solo es un atributo normalmente será añadido como atributo de otro objeto.
3.-El objeto tiene un conjunto de operaciones identificables para cambiar el valor de sus atributos y son comunes a cualquier ocurrencia del objeto
4.-Es una entidad externa que consume o produce información para la producción de cualquier solución en el problema

Además de estos, aquellos adicionales que completen el modelo. También ver que diferentes descripciones del problema pueden implicar diferentes decisiones en la creación de objetos y atributos.

OBTENCIÓN DE ATRIBUTOS; OPERACIONES; RELACIONES

Los atributos deben contestar a la pregunta ¿Qué elementos definen completamente al objeto en el contexto del problema?
Las operaciones describen el comportamiento y modifican características manipulando datos, realizando algún cálculo, monitorizando el objeto frente a la ocurrencia de un suceso de control.
Para las relaciones... buscar mensajes que se pasen entre objetos y relaciones de composición y agregación. Las relaciones de herencia se suelen encontrar al ver objetos con atributos y métodos comunes.

REVISIÓN FINAL

Finalmente revisar el diagrama obtenido y ver si cumple las especificaciones. Refinar el diagrama completando aspectos que no aparezcan en la descripción recurriendo a entrevistas con el cliente o a conocimiento en la materia.

Generación de código y documentación a partir del diagrama de clases

En algunos casos las herramientas CASE, como Visual Paradigm, podrán generar (o actualizar) las clases en código fuente a partir del diagrama de clases UML.

- Con el SDE integrado de VP-UML en Netbeans. Diferenciar entre "Sincronizar con el código" (Código fuente eliminado no se recupera, solo se actualiza el existente). "Forzar sincronizado a código" (Se actualiza todo el código que pueda partir del modelo, incluido el del código eliminado de Netbeans).
- Desde VP-UML puede darse a Herramientas >> Generación instantánea >> Java configurando idioma, clases a generar y otras características.

Es "Specification" o "Comentarios" se podrían añadir comentarios y luego con estos generar informes (HTML, PDF, doc) con la documentación de los diagramas seleccionados.
Indicar qué diagramas se quiere que intervengan, qué elementos se añadirán, propiedades de página, marca de agua...

4. Diagramas UML de comportamiento

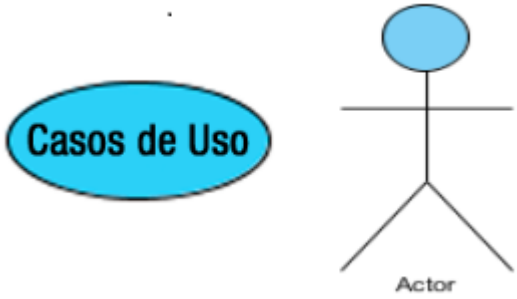
COMPORTAMIENTO DINÁMICO DE LOS OBJETOS : como la creación-destrucción de objetos, paso de mensajes entre objetos, orden en que deben hacerse, funcionalidad que el usuario espera realizar, cómo influyen elementos externos en el sistema...

4.1. Diagramas de casos de uso

- **Resuelve la falta de comunicación entre el EQUIPO DE DESARROLLO y el EQUIPO QUE DEMANDA UNA SOLUCIÓN SOFTWARE**. Son **fáciles de interpretar**, facilitan comunicación con el cliente.
- Determina **QUÉ puede hacer** cada tipo de usuario con el sistema, de una forma que **las personas ajenas al desarrollo puedan entenderlo**.
- Los casos de uso son visualización gráfica de los **REQUISITOS FUNCIONALES DEL SISTEMA**, representan las funciones que un sistema puede ejecutar.
- Su **función** es dirigir el proceso de creación de software definiendo qué se espera de él
- De los diagramas de casos de uso se desprenden otros que describirán la estructura y el comportamiento del sistema, influyen directamente en la implementación del sistema y en su arquitectura final. También se usa en la fase de pruebas para verificar que el sistema cumple con los requisitos funcionales, creándose muchos de los casos de prueba de caja negra a partir de los casos de uso.
- Normalmente los diagramas de casos de uso se hacen incluso antes del diagrama de clases

Los casos de uso se representan como elipses
Los actores que interactúan con ellos se representan como monigotes

Son grafos no conexos en los que los nodos son actores y casos de uso y las aristas son las relaciones que existen entre ellos.



Actores

Los actores representan un tipo de usuario del sistema (cualquier cosa externa que interactúa con el sistema sea ser humano, otro sistema informático, una empresa, una entidad para la autenticación de claves...)

Debe diferenciarse el actor de la forma en que se interactúa con el sistema. Un usuario puede interpretar diferentes roles según la operación que ejecuta. Cada rol es un actor diferente.
Los **actores** no representan a individuos particulares, sino a roles que estos pueden jugar.

Pueden ser primarios (interaccionan con el sistema para explotar su funcionalidad; trabajan directa y frecuentemente con el software), secundarios (soporte del sistema para que los primarios puedan trabajar; precisos para alcanzar un objetivo), iniciadores (no interactúan con el sistema, pero desencadenan el trabajo de otro actor). Si no son iniciados por ningún usuario o elemento software, puede crearse un actor llamado "Tiempo" o "Sistema"

Casos de uso

Especifica una secuencia de acciones, incluyendo variantes, que el sistema puede llevar a cabo y que produce un resultado observable de valor concreto.
El conjunto de los casos de uso forma el **comportamiento requerido** del sistema.

Lo importante no es crear el diagrama, sino la descripción de cada caso porque esto ayudará al equipo de desarrollo a crear el sistema. Para esto se usan otros diagramas que describen la dinámica del caso de uso (como el diagrama de secuencia) y una descripción textual que debe incluir:

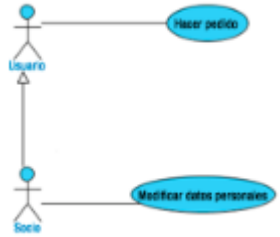
- **Nombre**: Del caso de uso
- **Actores**: Que interactúan con el sistema en el caso de uso
- **Propósito**: Lo que se espera que haga
- **Precondiciones**: Lo que debe cumplirse para que se lleve a cabo el caso de uso
- **Flujo normal**: Flujo normal de eventos que deben cumplirse para ejecutar el caso de uso exitosamente desde el punto de vista del actor y del sistema
- **Flujo alternativo**: Flujo de eventos que se llevan a cabo con casos inesperados o poco frecuentes. No se deben incluir errores como tipos de datos incorrectos o parámetros necesarios omitidos.
- **Postcondiciones**: Se cumplen una vez realizado el caso de uso

Relaciones

Las relaciones representan qué actores realizan tareas de los casos de uso (quiénes las inician) pero también hay relaciones más complejas:

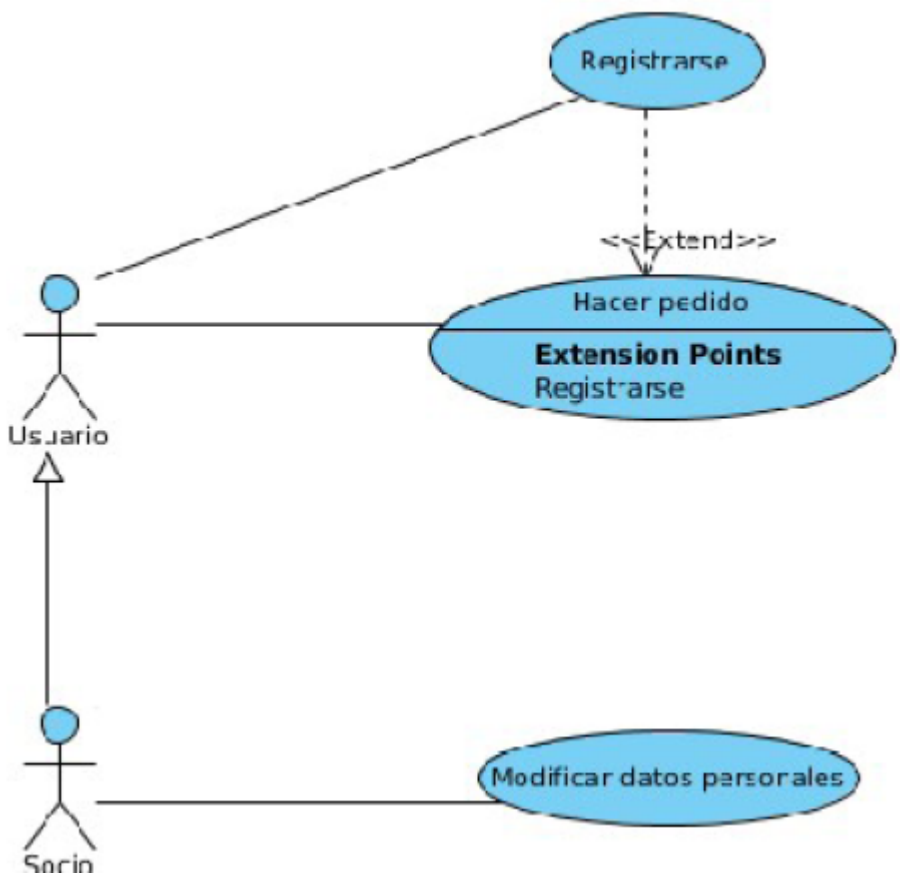
- **Asociación**: Relación entre el actor que interactúa con el sistema para iniciarlo. Línea continua une a un actor con un caso de uso. Ej.: "Usuario" "hacer pedido"
- **Generalización**: Es posible que unos casos de uso tengan comportamientos a otros que los modifican o completan de alguna forma. Se puede definir un caso base de forma abstracta y unos hijos que heredan sus características. (Suele usarse poco en los diagramas de casos de uso). Ej.: La persona puede hacer un pedido y, si quiere, darse de alta. Habría un rol "Usuario"

"hacer pedido" y un rol "Socio" que hereda de "Usuario" y puede "Modificar datos personales" (y además, "hacer pedido")



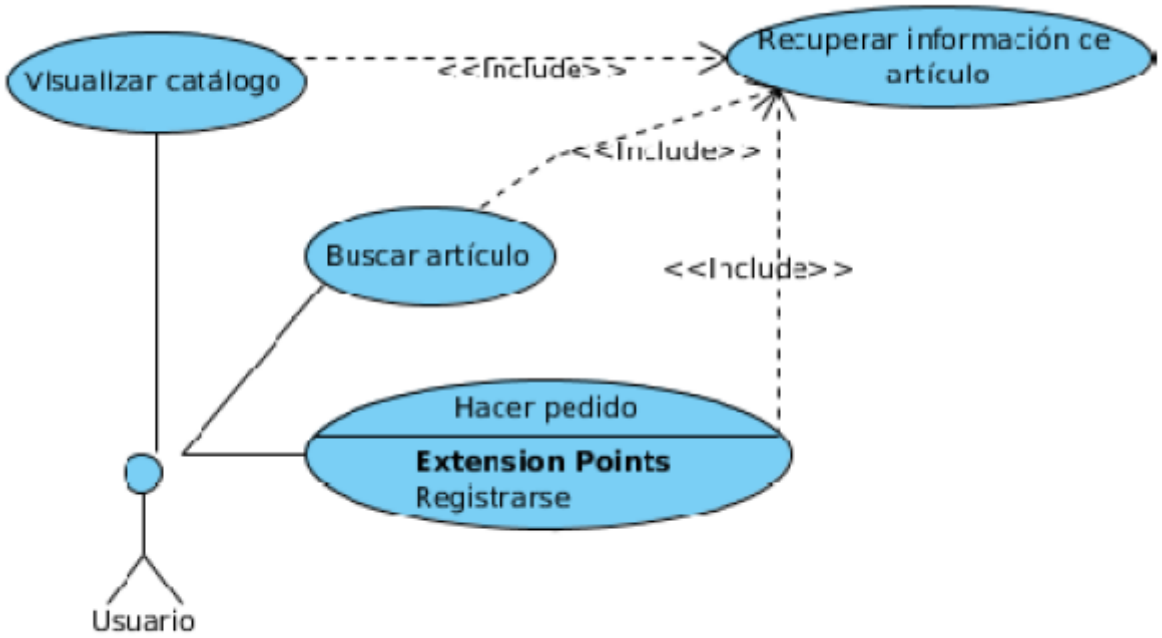
A la hora de usar las siguientes ("extends" e "include") debe tenerse en cuenta que los casos de uso extendidos o incluidos deben representar un flujo de actividad completo desde el punto de vista de lo que el actor espera que el sistema haga por él (caso de uso). No deben usarse estas herramientas para descomponer un caso de uso de envergadura en otros más pequeños.

- **Extensión**: Se puede usar relación "extends" para indicar que el comportamiento del caso de uso es diferente según alguna circunstancia. Así se simplifica el flujo de casos de uso complejos. Cuando un caso de uso extendido se ejecuta se indica en la especificación del caso de uso como un punto de extensión. Ej.: Cuando el usuario hace un pedido, si no es socio, se le da la posibilidad de darse alta en el sistema en ese momento, aunque puede realizar el pedido aunque no lo sea.



- **Inclusión**: Cuando la ejecución del caso de uso incluido se da en la rutina normal del caso que lo incluye se usa "include". Es útil cuando se desea especificar un comportamiento en común con dos o más casos de uso. Aunque es un error frecuente usar esta técnica para hacer subdivisión de funciones (debe tenerse cuidado). Ej.: Cuando se hace un pedido, se busca información de los artículos para obtener el precio. Pero esto también forma parte de otros como visualizar el catálogo de productos y buscar un artículo concreto. Al tener entidad por sí solo se separa del resto de casos de uso y se incluye en los otros tres. Ej2.: Empleado "Servir pedido" y "Consultar stock" (Consultar stock es "extends" porque tiene entidad suficiente como para ser caso de uso por sí mismo, que podrá usarse en otros casos y además siempre debe

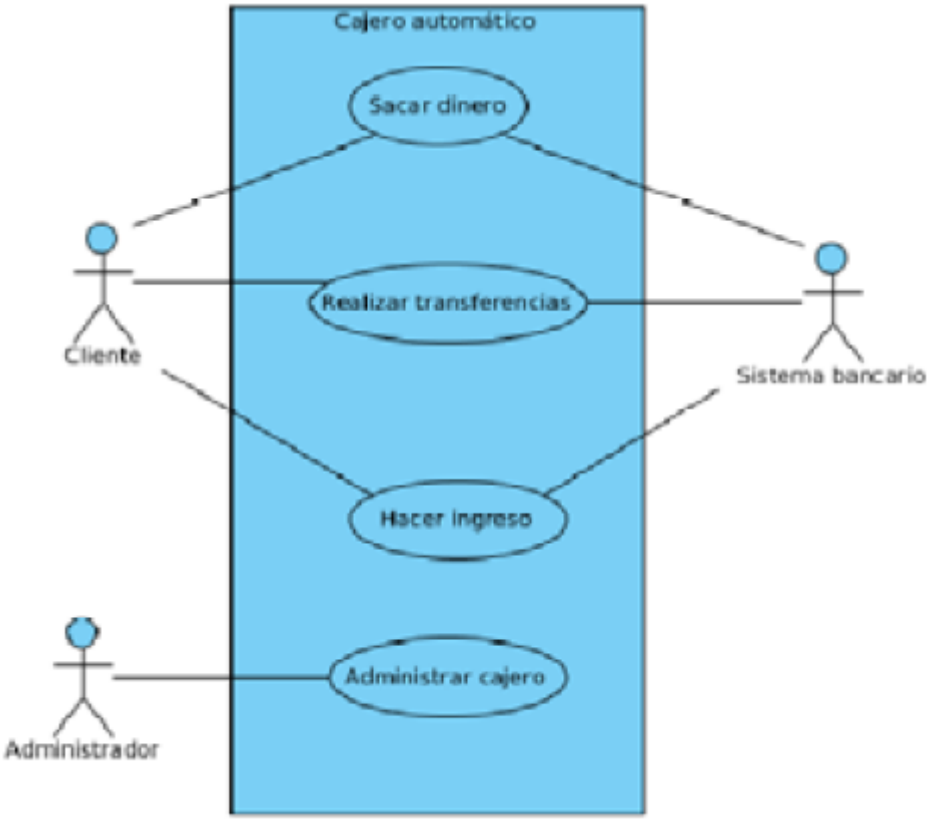
hacerse).
Ventajas: Descripciones de casos de uso cortas y se entienden mejor. Identificación de funcionalidad común puede identificar reutilización de componentes ya existentes.
Inconvenientes: Los diagramas son más difíciles de leer por los clientes.

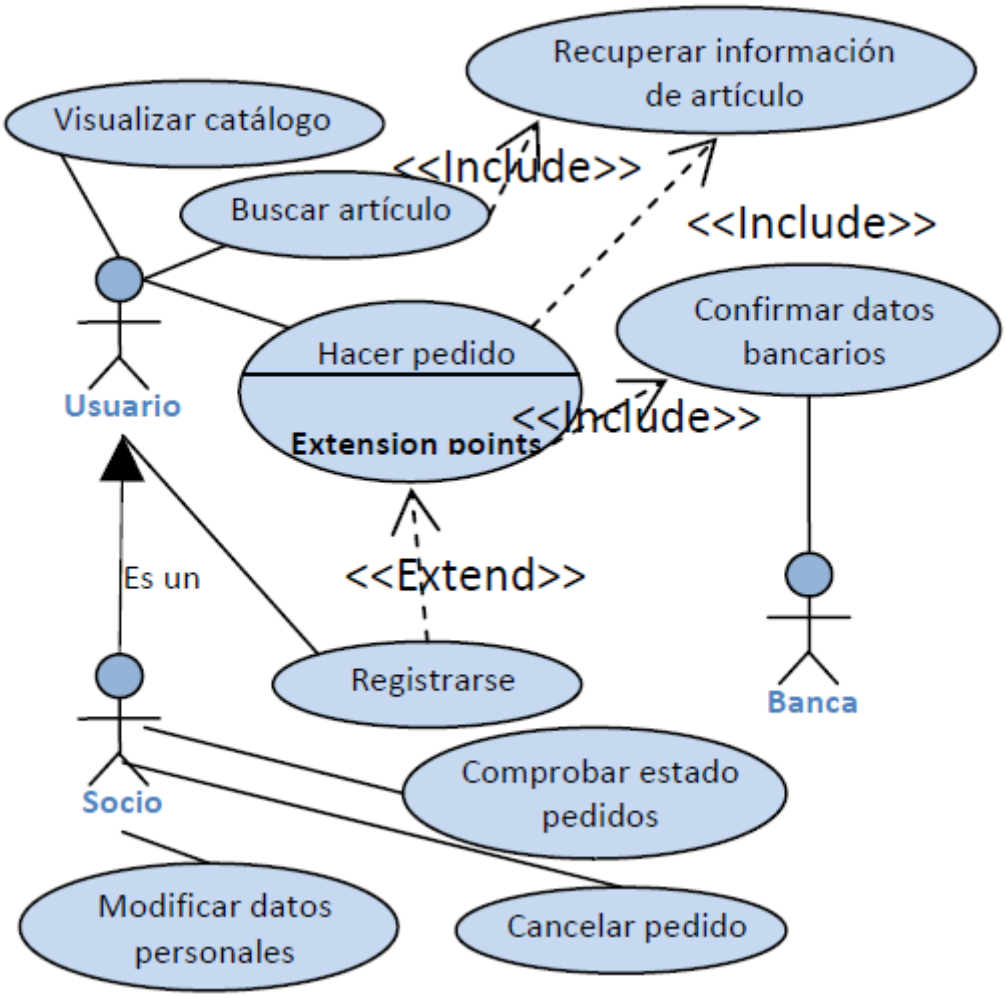


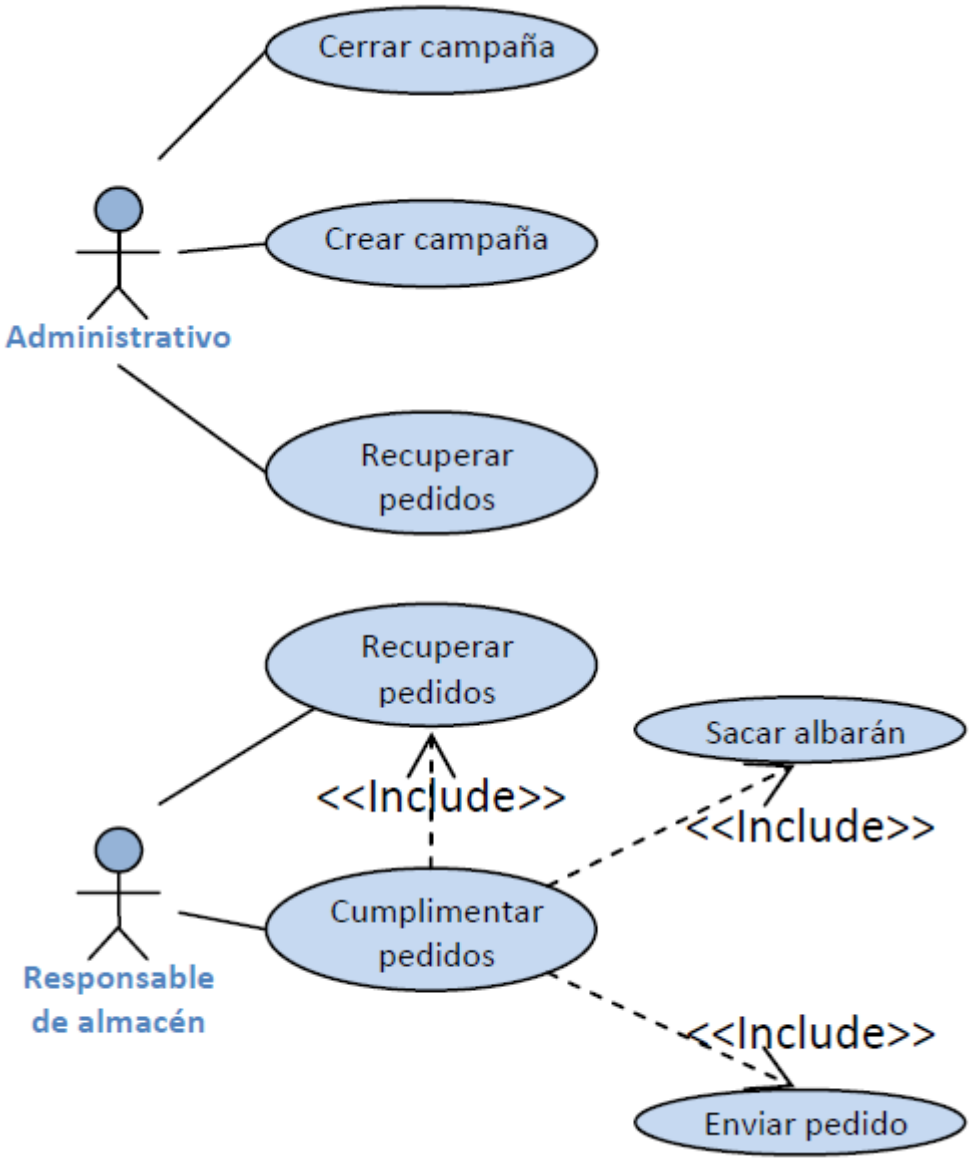
Elaboración de casos de uso

Identificar actores
Identificar funcionalidades
Funcionalidad de cada uno

Se parte de una descripción lo más detallada posible del problema, se detecta quién interactúa con el sistema para obtener los actores, se busca qué tareas realizan esos actores para descubrir casos de uso más genéricos. Se refina el diagrama añadiendo casos de uso más generarles para detectar casos por inclusión (aparecen en dos o más generales), exclusión y generalización. Al diagrama generado se le llama **diagrama frontera** (diagrama que incluye los casos de uso genéricos del sistema, que podrán ser desglosados en nuevos diagramas que los describan si es necesario. Se especifica enmarcando los casos de uso en un recuadro que deja a los actores fuera)


Cajero automático
Sacar dinero
Realizar transferencias
Hacer ingreso
Administrar cajero
Cliente
Sistema bancario
Administrador


Recuperar información de artículo
Visualizar catálogo
<<Include>>
Buscar artículo
<<Include>>
Confirmar datos bancarios
Hacer pedido
Extension points
<<Include>>
Usuario
Es un
<<Extend>>
Registrarse
Banca
Socio
Comprobar estado pedidos
Modificar datos personales
Cancelar pedido



Luego cada caso de uso debe documentarse (es lo más importante). Por ejemplo el caso de uso "Hacer pedido" (y sus posibles variantes):

Brief Description	EL usuario selecciona un conjunto de artículos, junto con la cantidad de los mismos, para crear el pedido. Cuando se formaliza se comprueba que el usuario sea socio. A continuación se comprueban los datos bancarios, se realiza el cobro y se crea el pedido.		
Preconditions	Existe un catálogo de productos disponibles para pedir. El usuario está registrado. Los datos bancarios son correctos.		
Post-conditions	Se crea un pedido con los datos del usuario que lo realiza y los artículos solicitados.		
Flow of Events		**Actor Input**	**System Response**
	1	Inicia el pedido.	
	2		Se crea un pedido en estado "en construcción".
	3	Selecciona un artículo.	
	4	Selecciona una cantidad.	
	5		Recupera la información del artículo para obtener el precio y modifica el precio total del pedido.
	6	El proceso se repite hasta completar la lista de artículos.	
	7	Se acepta el pedido.	
	8		Se comprueba si el usuario es socio, si no lo es se le muestra un aviso para que se registre en el sitio.
	9		Se comprueban los datos bancarios con una entidad externa.
	10		Se genera: calcula el total, sumando los gastos de envío.
	11		Se realiza el pago a través de la entidad externa.
	12		Se almacena la información del pedido con el estado "pendiente".

Flujo alternativo para el caso de uso Hacer Pedido cuando el usuario no está registrado.			
Author	usuario		
Date	26-ago-2011 17:56:35		
Brief Description	Cuando el usuario hace un pedido si no está registrado se abre la opción de registro para que se dé de alta.		
Preconditions	El usuario no está registrado. Existe un catálogo de artículos para hacer pedido. Los datos bancarios son correctos.		
Post-conditions	El usuario queda registrado. Se crea un pedido con los datos del usuario que lo realiza y los artículos solicitados.		
Flow of Events		**Actor Input**	**System Response**
	1	Inicia el pedido.	
	2		Se crea un pedido en estado "en construcción".
	3	Selecciona un artículo.	
	4	Selecciona la cantidad.	
	5		Recupera la información del artículo para obtener su precio y modifica el precio total a pagar por el pedido.
	6		Añade la información al pedido en creación.
	7	EL proceso se repite hasta completar la lista de artículos del pedido.	
	8	Acepta el pedido.	
	9		Se comprueba si el usuario es socio.
	10		Se invoca el registro de usuario.
	11		Se comprueban los datos bancarios con una entidad externa.
	12		Se genera: calcula el total, sumando los gastos de envío.
	13		Se realiza el pago a través de la entidad externa.
	14		El pedido queda almacenado con el estado "pendiente".

Flujo alternativo para hacer pedido cuando los datos bancarios no son correctos.			
Author	usuario		
Date	26-ago-2011 18:14:35		
Brief Description	Una vez que se han seleccionado los artículos y se han introducido los datos del socio, al hacer la comprobación de los datos bancarios con la entidad externa se produce algún error, se da la posibilidad al usuario de modificar los datos o de cancelar el pedido.		
Preconditions	Existe un catálogo de productos disponibles para pedir. El usuario está registrado. Los datos bancarios no son correctos.		
Post-conditions	Se crea un pedido con los datos del usuario que lo realiza y los artículos solicitados.		
Flow of Events		**Actor Input**	**System Response**
	1	Inicia el pedido.	
	2		Se crea un pedido en estado "en construcción".
	3	Selecciona un artículo.	
	4	Selecciona una cantidad.	
	5		Recupera la información del artículo para obtener el precio y modifica el precio total del pedido.
	6	El proceso se repite hasta completar la lista de artículos.	
	7	Se acepta el pedido.	
	8	Acepta el pedido.	
	9		Se comprueban los datos bancarios con una entidad externa, fallando la comprobación.
	10		Se solicitan los datos de nuevo.
	11	Introduce los datos de nuevo.	
	12		Se repite el proceso hasta que se acepten los datos bancarios o se cancele la operación.
	13		Se genera: calcula el total, sumando los gastos de envío.
	14		Se realiza el pago a través de la entidad externa.
	15		Se almacena la información del pedido con el estado "pendiente".

El resto de casos se uso se documentan de forma similar hasta completar la descripción formal de la funcionalidad del sistema.

Escenario

Es una ejecución particular de un caso de uso que se describe como una secuencia de eventos. Cada caso de uso es una generalización de un escenario.

El caso de uso debe especificar un comportamiento deseado pero no imponer cómo se llevará a cabo (debe decir QUÉ pero no CÓMO).

Los escenarios se documentan en los diagramas de secuencia

4.2. Diagramas de secuencia

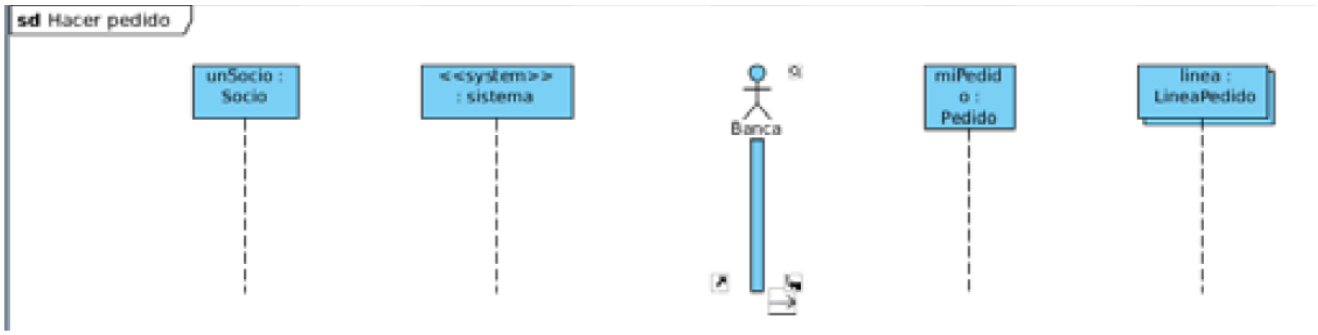
Los diagramas de secuencia formalizan la **descripción de un escenario representando mensajes que fluyen en el sistema, quién los envía y quién los recibe**.

(No se incluyen los eventos por ejemplo... aquí nos centramos en los mensajes)

Los **objetos y actores** del escenario se representan con **rectángulos distribuidos horizontalmente en la parte superior del diagrama** a los que se asocia una,**línea de vida**, línea punteada vertical que representa el paso del tiempo y que el objeto existe. De ella salen en orden los mensajes que se pasan entre ellos. (Así puede hacerse una idea el equipo de desarrollo de las diferentes operaciones que deben ocurrir al ejecutarse y el orden; de qué objetos participan en el caso de uso y cómo interaccionan a lo largo del tiempo)

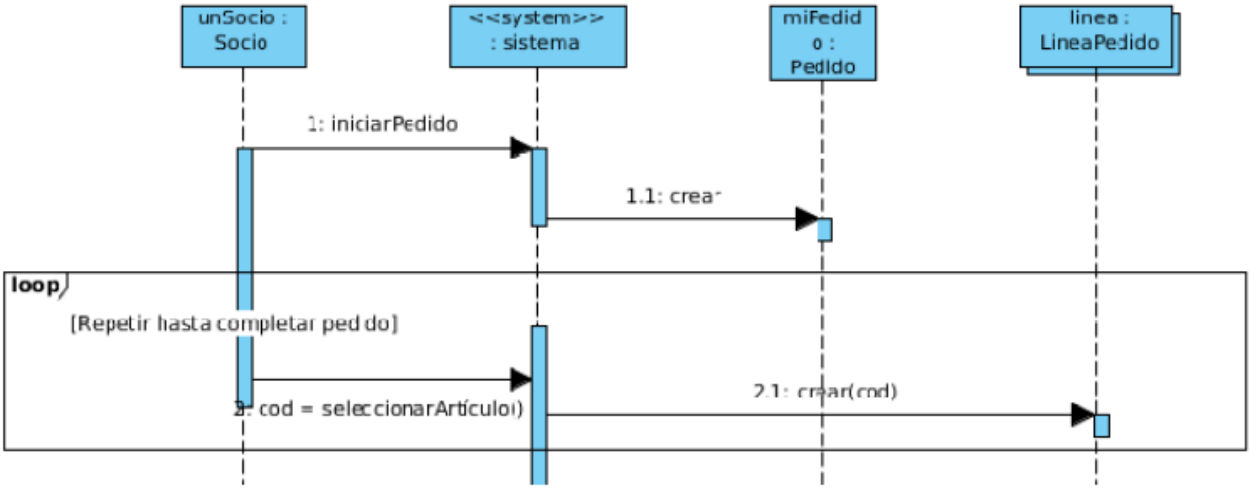
En el rectángulo de la línea de vida se anota el nombre del objeto (opcional), dos puntos y el nombre de la clase a la que pertenece.
Se usará el sistema para representar solicitudes al mismo como pulsar un botón para abrir una ventana o llamada a una subrutina.

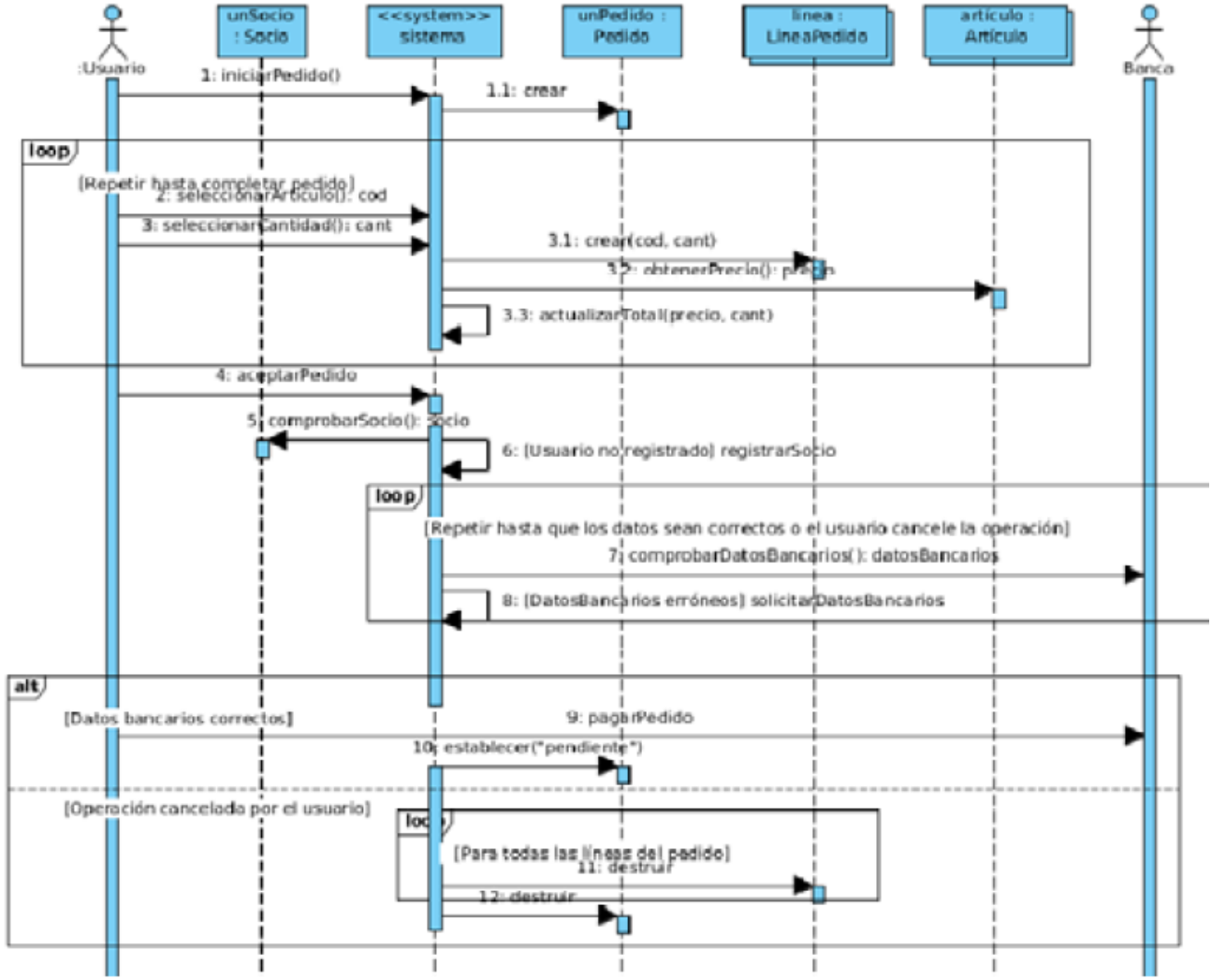
Si el objeto puede tener varias instancias aparecerá como un rectángulo sobre otro (como las líneas de pedido, que pueden ser varias)



La **invocación de métodos** (mensajes) son flechas horizontales que van de una línea de vida a otra, indicando la dirección del mensaje del que envía al que recibe. Ambos pueden ser el mismo objeto y su orden viene determinado por su posición vertical: mensaje debajo de otro indica que se envía después, no es necesario una secuencia.

Se pueden representar **iteraciones** usando marcos (normalmente se nombra el marco con el tipo de bucle a ejecutar y la condición de parada) y **condicionales** en función de un valor determinado,. También se pueden incluir etiquetas y notas en el margen izquierdo.





Elaborar diagramas de secuencia

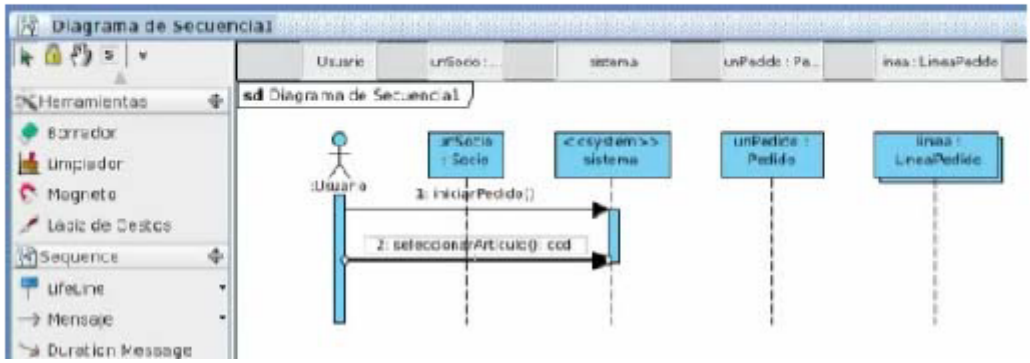
Incluir entidades participantes

Actores.
El primer paso es incluir las entidades participantes. Tenemos dos actores, el usuario que inicia el caso de uso y la banca que es un actor secundario que realiza una operación por nosotros como es comprobar que los datos de la tarjeta sean correctos.

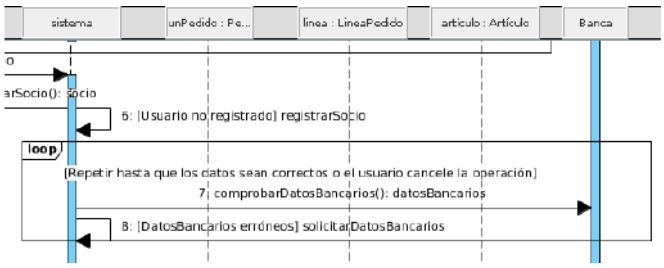
Objetos.
También necesitamos que intervengan cuatro objetos de nuestro sistema: un socio, un pedido con sus líneas de pedido, y los artículos a pedir.
Para **nombrar** estas entidades en la especificación indicamos su nombre y seleccionamos en el clasificador de base, que muestra la lista de clases del sistema la clase a la que pertenece, así aparce automáticamente su nombre en el formato adecuado.
Indicaremos que las líneas de pedido y los artículos son un conjunto de ojetos marcando en la especificación la casilla **"Multi-object"**, en la zona inferior del cuadro de diálogo.

Sistema.
Por último también interviene el sistema en si mismo, que utilizaremos para las operaciones relacionadas con la interfaz gráfica y las que se lanzan directamente para crear objetos, recuperar información del sistema como los datos de un artículo, o comunicarnos con entidades externas como la banca.
Se añade como una línea de vida más, a la que en su especificación indicamos que su nombre es sistema, y en la pestaña **Estereotipos** seleccionamos **System**.

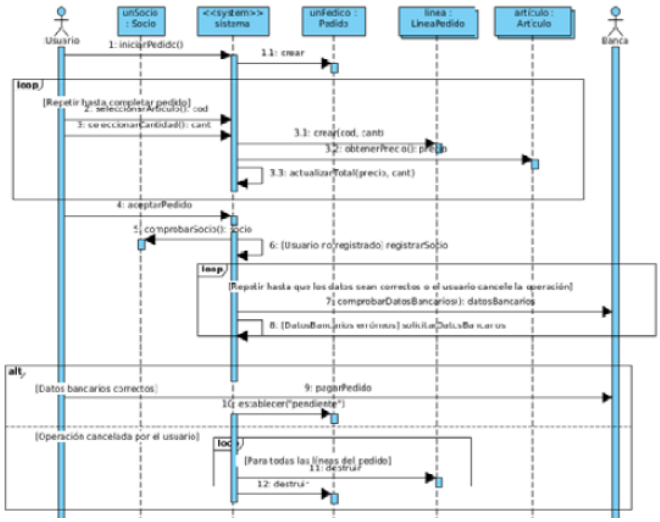
Los mensajes pueden devolver un valor, que podemos escribir en la especificación del mismo en la opción return value.



Añadiremos condiciones de guarda cuando queramos indicar que un mensaje se enviará sólo si se cumple cierta condición, por ejemplo, sólo registraremos a un usuario si no es socio ya.



También se pueden incluir condiciones más elaboradas que implique una bifurcación en el flujo de eventos, como es el caso de la comprobación de la tarjeta, si todo marcha bien se finalizará la creación del pedido, si no, el usuario puede cancelar la operación sin guardar nada.

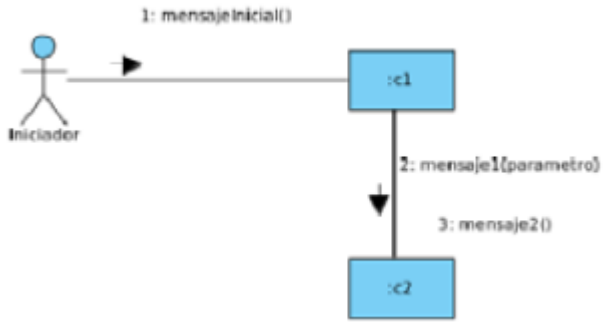


4.3. Diagramas de colaboración

Los diagramas de colaboración complementan a los de secuencia, mostrando las interacciones entre objetos del diagrama mediante el paso de mensajes entre ellos. Los nodos son objetos y las aristas enlaces entre objetos. Los objetos se conectan mediante enlaces y se comunican mediante los mensajes.

No se incluyen líneas de vida, sino que los mensajes se numeran para determinar su orden.

Por ejemplo: Actor iniciador manda mensaje a objeto c1 que inicia escenario; objeto c1 envía mensaje1 que lleva parámetro a objeto c2 y el mensaje2 al objeto c2.



En cierto modo, son semejantes a los de secuencia. (Representan misma información, representan entidades del sistema y los mensajes que circulan entre ellas y es fácil detectar la secuencialidad)

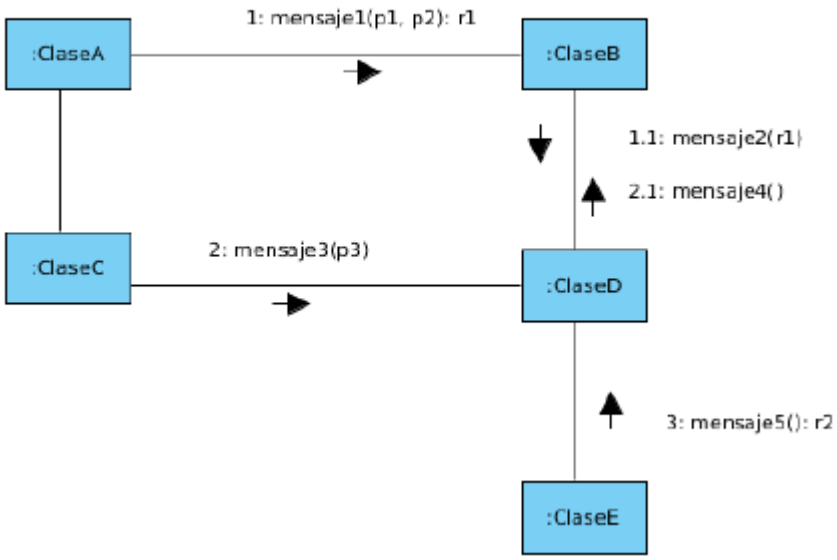
En los rectángulos aparece como "NombreClase" "NombreObjeto" ":nombreClase" (objeto genérico de la clase) "NombreObjeto:nombreClase"

Para que sea posible el paso de mensaje debe haber una asociación entre los objetos, quedando garantizada la navegación y la visibilidad entre ambos.

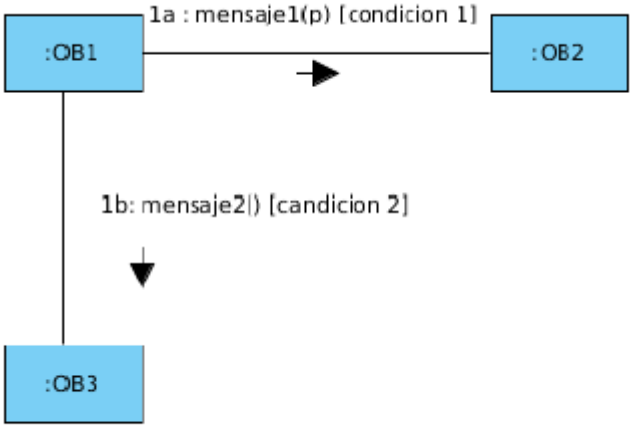
El mensaje es la especificación de una comunicación entre objetos que transmite información y desencadena una acción en el destinatario. La sintaxis es la siguiente:

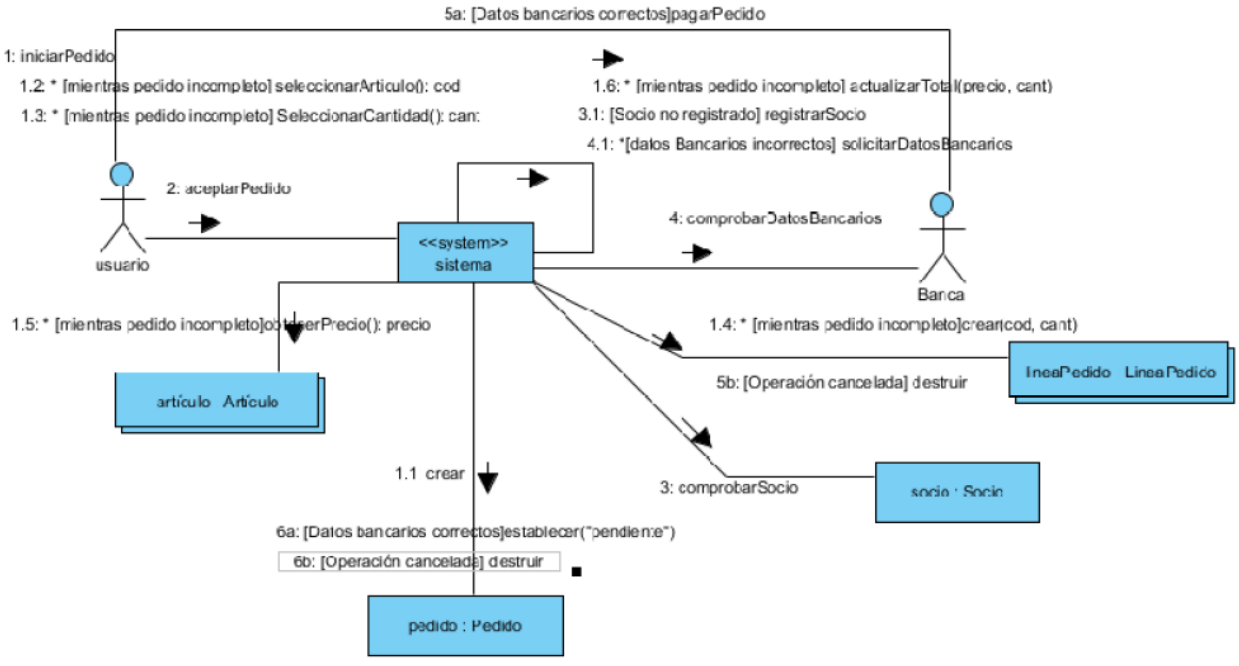
```
[secuencia][*][Condición de guarda]{valorDevuelto} : mensaje (argumentos)
```

Secuencia: Nivel de anidamiento del envío del mensaje dentro de la interacción. Mensajes se numeran para indicar el orden en que se envían y si es necesario se puede anidar con subrangos
* Mensaje iterativo
Condición de guarda: Debe cumplirse para que el mensaje sea enviado
ValorDevuelto: Lista de valores devueltos por el mensaje
Mensaje: Nombre del mensaje
Argumentos: Parámetros que se pasan al mensaje



Cuando hay una condición se repite el número de secuenmcia y se añaden las condiciones necesarias; enviándose uno u otro pero no ambos:





- Actividades que se repiten o pueden repetirse se marcan con asteriscos y su condición

- Condiciones de guarda se escriben en el mismo nombre que el mensaje
- Flujo alternativo de eventos si el usuario cancela el pedido o no, obliga a modificar la secuencia pasando a 4a 6a, 5b 6b.
- Objeto sistema con estereotipo system

4.4. Diagramas de actividad

Es una especialización del diagrama de estado, organizado respecto de las acciones. Se compone de actividades y representa cómo se pasa de una a otras. Las actividades se enlazan por transiciones automáticas (cuando termina una, se desencadena otra).
Es fundamental para el flujo de control entre actividades distinguiéndose cuáles se pueden llevar a cabo secuencialmente y cuáles conurrentemente. Define la lógica de control:

- En el modelado de los procesos del negocio
- Análisis de un caso de uso
- Compresión del flujo de trabajo a través de varios casos de uso
- Expresar aplicaciones multihilo.

Es un grafo conexo en el que los nodos son estados que pueden ser de actividad o de acción y los arcos son transiciones entre estados. El grafo parte de un nodo inicial que representa el estado inicial y termina en un nodo final.

Elementos del diagrama

Estados

Estados de actividad: Elemento compuesto que se descompone en otros estados de actividad y de acción
Estados de acción: Estado que representa la ejecución de una acción atómica que no se puede descomponer ni interrumpir (invocación de operación). Se considera que la ejecución conlleva tiempo insignificante
Es posible definir **Estado inicial** y **Estado final**

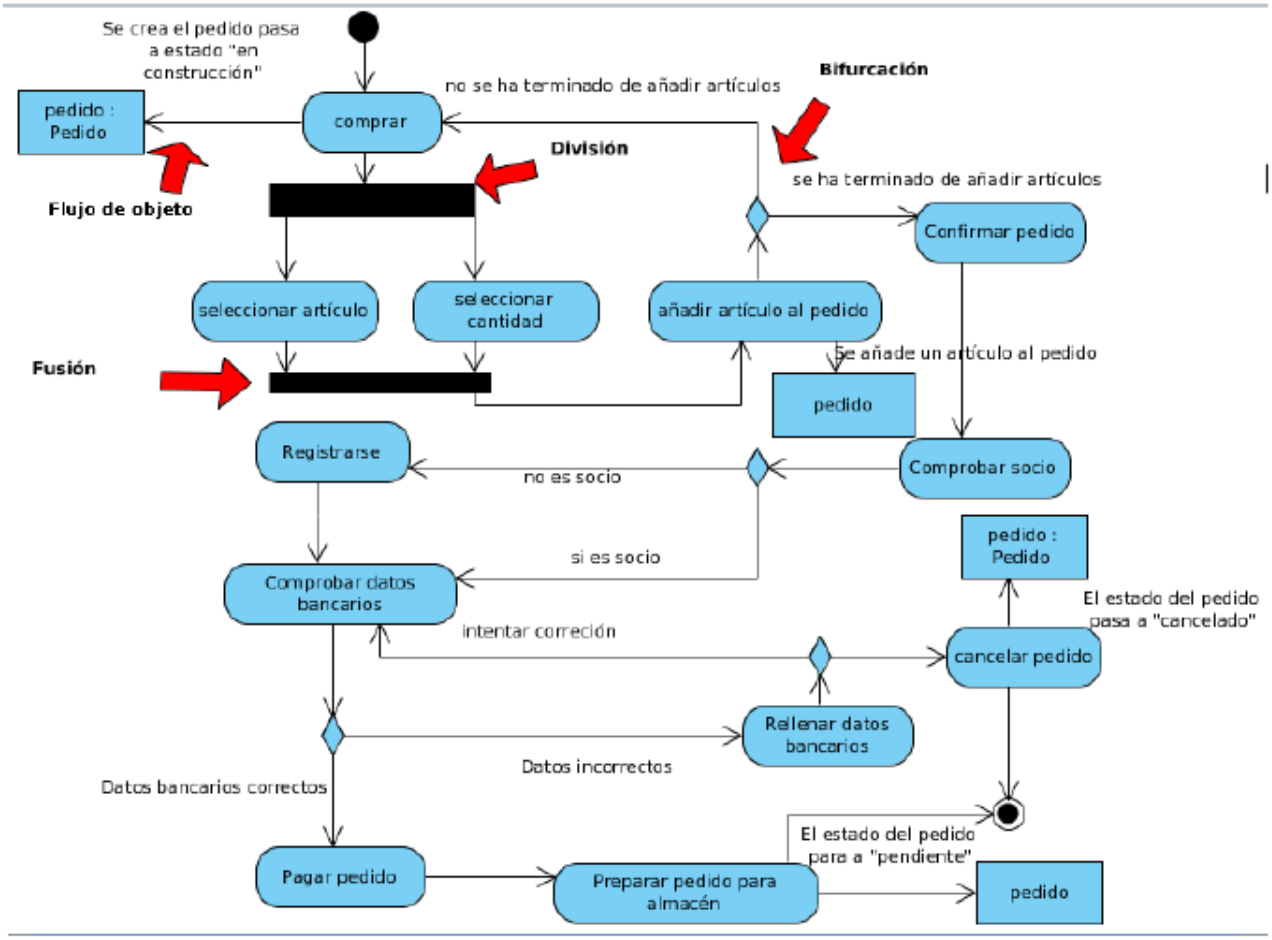
Transiciones

Relación entre dos estados que indica que un objeto en el primer estado hará ciertas acciones y pasará al segundo estado cuando ocurra un evento específico y satisfaga ciertas condiciones. Se representa por línea dirigida de un estado inicial al siguiente. Pueden encontrarse varias transiciones:

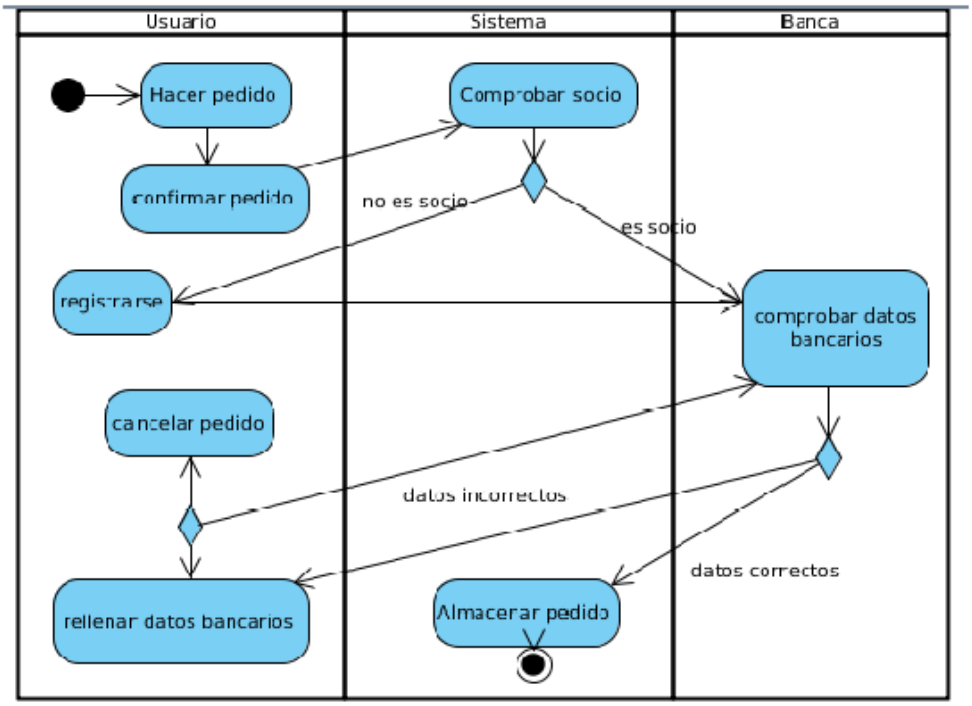
- **Secuencial** o sin disparadores: Al completar una acción se ejecuta la acción de salida y pasa al siguiente estado
- **Bifurcación** (Decision node). Caminos alternativos según el valor de una expresión booleana. Condiciones de salida no deben solaparse y deben cubrir todas las posibilidades (puede ejecutarse la palabra clave else)
- **Fusión** (Merge node): Redirigen varios flujos de entrada a un único flujo de salida. Sin espera ni sincronización.
- **División** (Fork node) Sincronización o ejecución paralela de actividades. Son concurrentes.
- **Unión** (Join node): Los flujos entrantes se sincronizan, es decir, cada uno espera hasta que han alcanzado la unión.

Objetos: Cuando interviene un objeto no se usan los flujos de eventos habituales sino flujos de objetos (representados con flechas) que permiten visualizar qué objetos están dentro del flujo de control asociados a un diagrama de actividades. Junto a ello puede indicarse cómo cambian los valores de sus atributos, su estado o sus roles.

Se usan **carriles o calles** para ver QUIENES son los responsables de hacer las distintas actividades (qué parte de la organización es responsable de una actividad): Cada calle tiene un nombre único. Puede ser implementada por una o varias clases. Las actividades de cada calle se considera independiente y se ejecutan concurrentemente a las de otras calles.



En este otro diagrama se simplifican las acciones a realizar y se eliminan los objetos para facilitar la inclusión de calles que indican quien realiza cada acción:



4.5. Diagramas de estados

Expresa el comportamiento de un objeto como una progresión a través de una serie de estados provocada por eventos y las acciones relacionadas que puedan ocurrir.

Representan máquinas de estados (autómatas de estados finitos) para modelas el comportamiento dinámico basado en la respuesta a determinados eventos de aquellos objetos que requieran su

especificación (normalmente por su comportamiento significativo en tiempo real y su participación en varios casos de uso). El resto de objetos se dice que tienen un único estado.

Con el diagrama de estados se cumple que:

- Un objeto está en un estado concreto en un cierto momento determinado por los valores de sus atributos
- La transición de un estado a otro es momentánea y se produce con un evento
- Una máquina de estados procesa un evento cada vez y termina con todas las consecuencias del evento antes de procesar otro. Si ocurren dos eventos simultáneos se procesan como si se hubieran procesado en cualquier orden, sin pérdida de generalidad.

Estados

Estado: Situación en la vida de un objeto en la que satisface cierta condición, realiza alguna actividad o espera algún evento.

Elementos de un estado:

- Nombre
- Acciones E/S
- Actividad a realizar
- Subestados cuando el estado sea complejo y necesite diagrama
- Eventos diferidos
  Hay dos estados especiales el "estado inicial" (punto de partida para una transición cuyo destino es el límite de un estado compuesto). El estado inicial del estado de nivel más alto representa la creación de la instancia de clase y el "estado final" estado espacial de un estado compuesto que, cuando está activo, indica que la ejecución del estado compuesto ha terminado y que una transición de finalización que sale del estado compuesto está activada.

Eventos

Un evento es un acontecimiento que ocupa un lugar en el tiempo y en el espacio y que funciona como un estímulo que dispara una transición en una máquina de estados. Existen eventos externos e internos, según quién los produzca.

Tipos de eventos:

- **Señales (Excepciones)**: La recepción de una señal (entidad a la que se le ha dado nombre, estereotipada, explícitamente prevista para la comunicación explícita), prevista para la comunicación explícita y asíncrona entre objetos. Enviada por un objeto a otro objeto o conjunto de objetos. Las señales con nombre que puede recibir un objeto se modelan designándolas en un compartimento extra de la clase de ese objeto. Normalmente una señal es manejada por la máquina de estados del receptor y puede disparar una transición a la máquina de estados
- **LLamadas:** Recepción de una petición para invocar una operación. Normalmente el evento de llamada es modelado como una operación del objeto receptor, manejado como un método del receptor e implementado como una acción o transición de la máquina de estados.

- **Paso del tiempo:** Representa el paso del tiempo (tiempo absoluto respecto de reloj real o virtual o paso de cantidad de tiempo dada desde que el objeto entra en el estado). Palabras clave: after(2 segundos); after 1 ms desde la salida del devInactivo...
- **Cambio de estado**: Representa el cambio en el estado o el cumplimiento de una condición. Palabras clave: when, seguida de una expresión booleana que puede ser de tiempo ode otra clase: when (hora = 11:30) when (altitud < 1000)

Transiciones

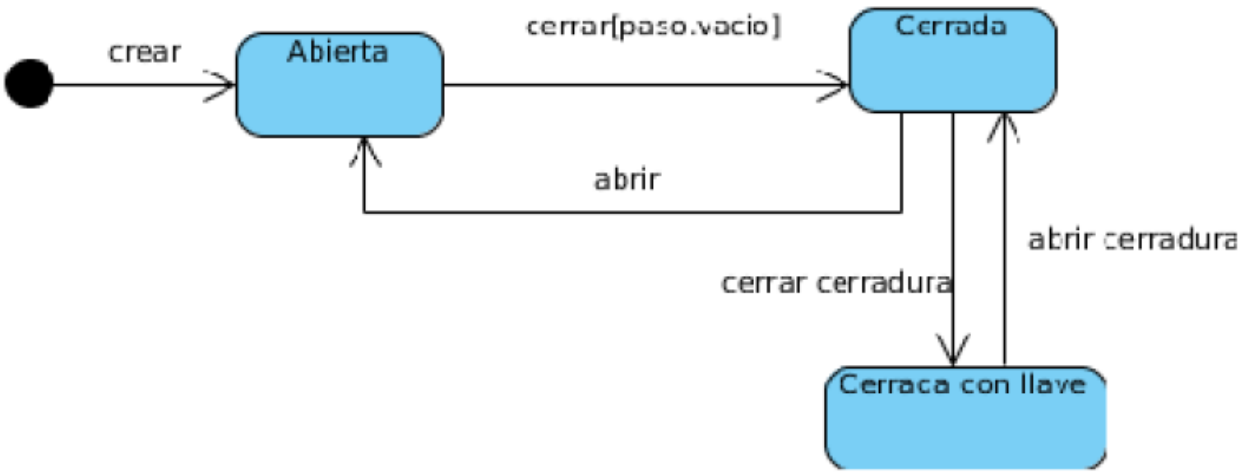
Una transición de un estado A a un estado B se produce cuando se origina un evento asociado y se satisface cierta condición especificada, en cuyo caso se ejecuta la acción de salida de A, la acción de entrada a B y la acción asociada en la transición.

La signatura de la transición es:

```
Evento(argumentos) [condicion] / accion
```

Elementos de una transición:

- **Estados origen y destino:** Se dispara la transición si, estando en el estado origen, se produce el evento de disparo y se cumple la condición de guarda (si la hay), pasando a ser activo el estado final.
- **Evento de disparo**: Cuando se produce un evento, afecta a todas las transiciones que lo contienen en su etiqueta. Todas las apariciones de un evento en la misma máquina de estados deben tener la misma signatura. Los tipos de eventos se vieron más arriba.
- **Condición de guarda:** Expresión booleana. Si es falsa, la transición no se dispara. Si no hay otra transición etiquetada con el mismo evento que pueda dispararse, este se pierde.
- **Acción**: computación atómica ejecutable. Puede incluir llamadas a operaciones del objeto que incluye la máquina de estados (o sobre otro visibles), creación o destrucción de objetos o envío de una señal a otro objeto.



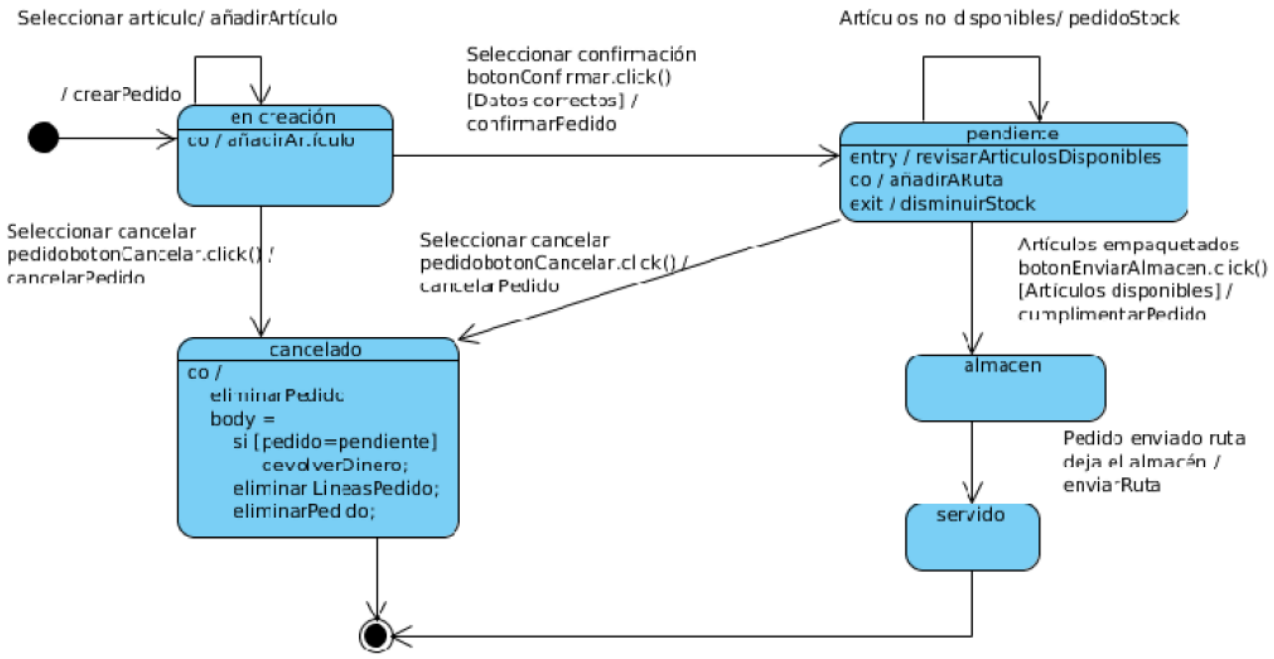
Aquí `cerrar[paso.vacio]` implica que para cerrar la puerta el paso debe estar vacío. No se podrá cerrar hasta que no lo esté.

Vamos a ver el diagrama de estados asociado a un objeto "pedido" ya que este tiene un comportamiento significativo en tiempo real, su situación física y en el sistema va evolucionando con el tiempo y participa en varios casos de uso.

Se definen como estados del pedido:

- En creación
- Pendiente
- En almacén
- Servido
- Cancelado

Las transiciones entre estados se producen por llamadas a procedimientos, no interviniendo ni cambios de estado o tiempo, ni señales:



5. Ingeniería inversa

Proceso de analizar código, documentación y comportamiento de una aplicación para identificar sus componentes actuales y dependencias para extraer y crear una abstracción del sistema e información del diseño. No se altera el sistema en estudio, solo se produce un conocimiento adicional del mismo.

Tipos de ingeniería inversa:

- **Ingeniería inversa de datos**: Sobre código de bases de datos (aplicación SQL) para obtener modelos relacionales o sobre el modelo relacional para obtener el diagrama E-R.
- **Ingeniería inversa de lógica o de proceso**: Sobre el código de un programa para averiguar su lógica (reingeniería) o sobre un documento de diseño para obtener documentos de análisis o de requisitos
- **Ingeniería inversa de interfaces de usuario**: Busca mantener la lógica interna del programa para obtener modelos y especificaciones que sirvieron de base para la construcción de la misma, para tomarlas como punto de partida en ingeniería directa para modificar la interfaz.

En herramientas como VP-UML se le daría a "Update UML Model" o a Herramienta >> CodigoI >> Reverse Code, indicándole dónde se localiza el código fuente. Obtiene clases y relaciones de herencia, el resto de relaciones deben establecerse a mano.