



Unified Modeling Language UML

Alberto de la Encina Vara

Departamento de Sistemas Informáticos y Programación
Universidad Complutense de Madrid, España

Contenido

- Introducción
 - Modelado de sistemas
 - Orientación a objetos
 - El lenguaje UML
 - Desarrollo de aplicaciones

Contenido

- Diagramas de UML
 - Diagrama de clases y objetos
 - Diagrama de paquetes
 - Diagrama de casos de uso
 - Diagrama de estados
 - Diagrama de actividad
 - Diagrama de secuencia
 - Diagrama de colaboración
 - Diagramas de UML
 - Diagrama de componentes
 - Diagrama de despliegue
 - Conclusiones

Contenido

- Un ejemplo global
 - Recogida de requisitos
 - Análisis
 - Diseño
 - Desarrollo

Introducción

- Modelado de sistemas
- Orientación a objetos
- El lenguaje UML
- Desarrollo de aplicaciones

Modelado de sistemas

Contenido

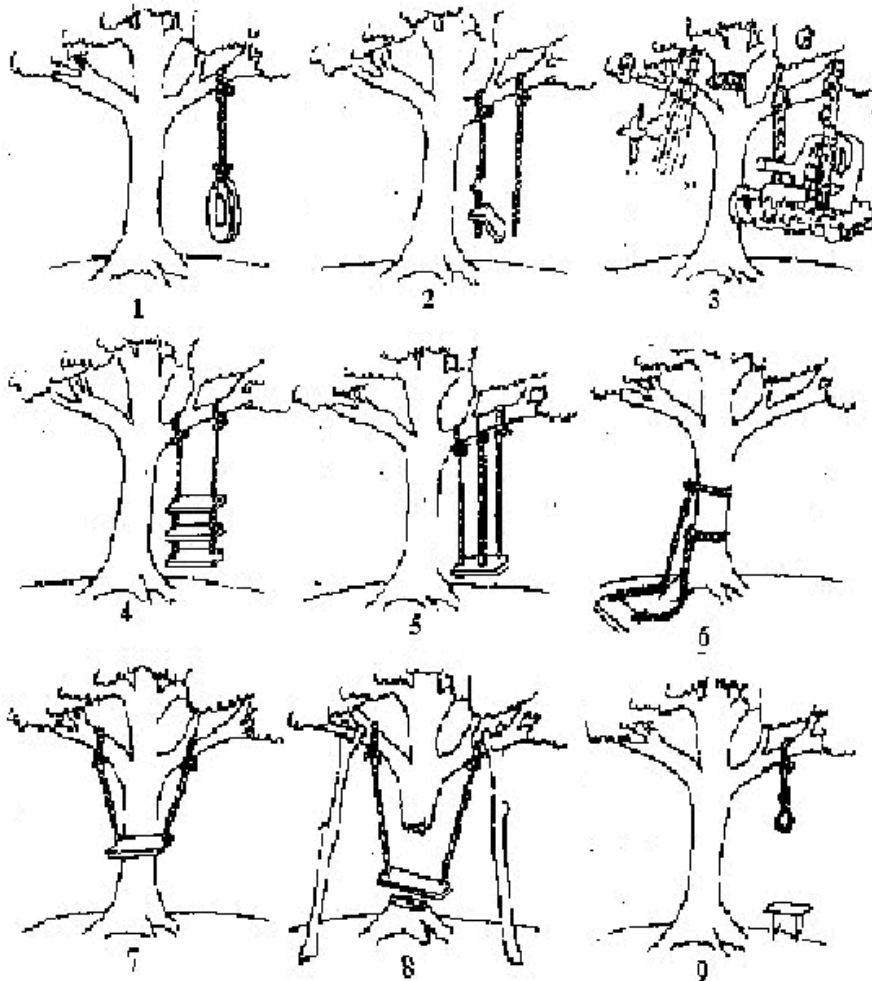
Modelado de sistemas.

- Motivaciones
- Problemas del desarrollo de software y sus soluciones
- Modelado
 - ¿Qué es modelar?
 - ¿Por qué modelar?
 - Principios del modelado
 - Proceso de modelado
 - ¿Cómo modelar?
- Desarrollo del software

Motivación

- Comprender mejor los sistemas software desarrollados.
- Eliminar errores de programación en los sistemas de software.
- Desarrollar las aplicaciones software más rápidamente.
- Mantener las aplicaciones software con un coste menor.
- Hacer comprensible el desarrollo software.
- Tener un método general para que otro desarrollador pueda entender más fácilmente nuestros programas.
- Hacer reutilizable parte de los componentes software.

Problemas en el desarrollo de software



1. **Necesidades.** Lo que realmente quiere el cliente.
2. **Descripción del cliente.** Lo que el cliente es capaz de describir.
3. **El proceso de venta.** Lo que el vendedor promete.
4. **Requerimientos.** La comprensión final de los requisitos descritos por el cliente.
5. **Análisis.** Cómo se planea que funcione el sistema para el cliente.
6. **Diseño.** Cómo realizar el sistema la funcionalidad analizada.
7. **Codificación.** Lo que el programador producirá.
8. **La instalación.** Lo que realmente se instalará en el cliente.
9. **Test.** Lo que el responsable del sistema verá.

Problemas en el desarrollo de software

- El programa no cumple con los requerimientos.
- El programa tiene errores.
- Cuesta aproximadamente el doble de tiempo del esperado el desarrollo de la aplicación.
- Suele costar mucho tiempo el mantenimiento de la aplicación.
- Es muy difícil comprender y mantener el programa desarrollado por otro grupo.
- Etc...

Problemas en el desarrollo de software

- El programa no cumple con los requerimientos.
 - Orígenes del problema:
 - 1.- Dificultades para **determinar** que se quiere.
 - 2.- Dificultades para **representar** que se quiere.
 - 3.- Problemas a la hora de **transmitir** la información.
 - En resumen:
 - 1.- Un problema de **comprensión**.
 - 2.- Un problema de **visualización**.
 - 3.- Un problema de **comunicación**.

Problemas en el desarrollo de software

- El programa tiene errores.
 - Orígenes del problema:
 - 1.- Dificultades para **representar** que se quiere.
 - En resumen:
 - 1.- Un problema de **visualización**.

Problemas en el desarrollo de software

- Cuesta aproximadamente el doble de tiempo del esperado el desarrollo de la aplicación.
 - Orígenes del problema:
 - 1.- Dificultades para **representar** el tiempo que nos llevará.
 - En resumen:
 - 1.- Un problema de **visualización**.

Problemas en el desarrollo de software

- Suele costar mucho tiempo el mantenimiento de la aplicación.
- Es muy difícil comprender y mantener el programa desarrollado por otro grupo.
 - Orígenes del problema:
 - 1.- Programar no es aún una ciencia, sino un **arte**.
 - En resumen:
 - 1.- Necesitamos un **modelo estándar** de desarrollo de software.

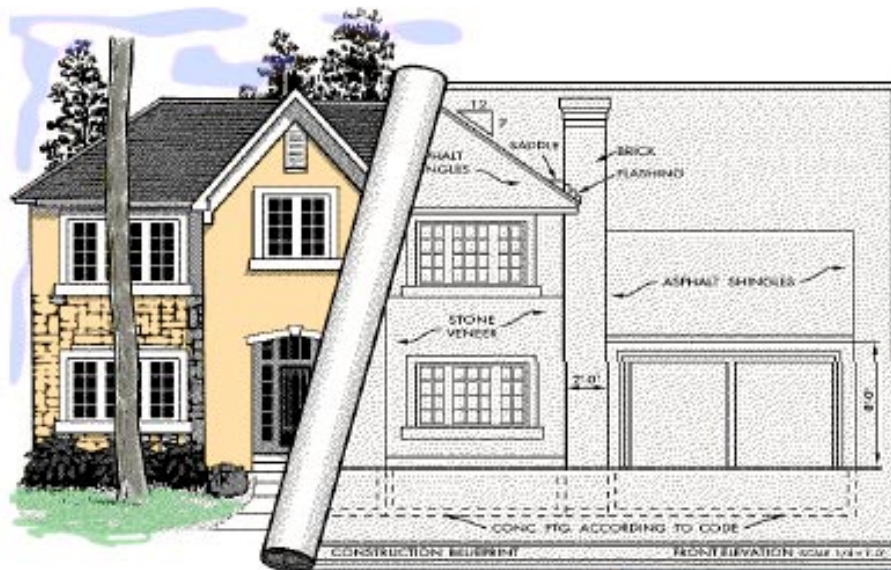
Soluciones a los problemas

- El programa no cumple con los requerimientos.
 - Solución:
Definir unas buenas especificaciones.
- El programa tiene errores.
 - Solución:
Definir correctamente y bajo una metodología el diseño del programa y los test.
- Cuesta aproximadamente el doble de tiempo del esperado el desarrollo de la aplicación.
 - Solución:
Realizar una buena planificación.

Soluciones a los problemas

- Suele costar mucho tiempo el mantenimiento de la aplicación.
- Es muy difícil comprender y mantener el programa desarrollado por otro grupo.
 - Solución:
 - Realizar un modelo previo a la implementación.
 - Usar una notación estándar. (Ej. UML)

¿Qué es modelar?



¿Qué es modelar?

- Modelar consiste en realizar un esquema de la realidad.
- Un modelo es una simplificación de la realidad, de manera exacta y estructurada siguiendo una metodología.

¿Por qué modelar?

- Construimos modelos para que podamos comprender de forma más sencilla el sistema que estamos desarrollando.
- Construimos modelos de sistemas complejos, ya que no podemos comprender estos sistemas de forma global.
- Construimos modelos ya que nos ayudan a:
 - **visualizar** el sistema como es o como queremos que sea.
 - **especificar** la estructura y comportamiento del sistema.
 - **construir** el sistema. Son una guía que nos ayuda a manejar los riesgos y ver los requisitos de forma más clara.
 - **documentar** las decisiones que hemos tomado.

Principios del modelado



SHELBY MUSTANG 500 GT

- Estructura externa e interna.
- El comportamiento desde el punto de vista del usuario.
- Los pasos necesarios para su fabricación.
- Los componentes del coche, motor, radiador, transmisión, distintos circuitos, ...
- El comportamiento de los componentes internos, motor, frenos, airbag, ...

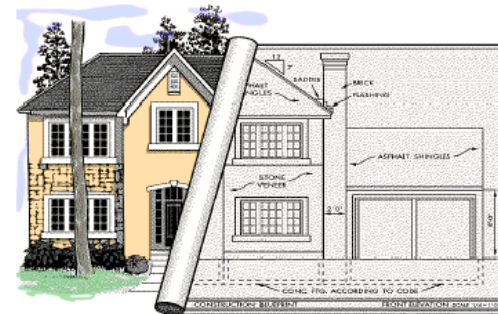
Principios del modelado

- La elección del modelo a crear tiene una influencia profunda en como se va a atacar el problema y como se va alcanzar la solución.
- Cada modelo puede expresar los diferentes niveles de precisión.
- Los mejores modelos son los conectados con la realidad.
- Un modelo sólo no es suficiente. La mejor aproximación a un sistema no trivial es un conjunto de modelos cercanos pero independientes que expresan los diferentes comportamientos.

Principios del modelado

Los modelos hacen que haya una separación clara entre la vista lógica y la vista física del sistema.

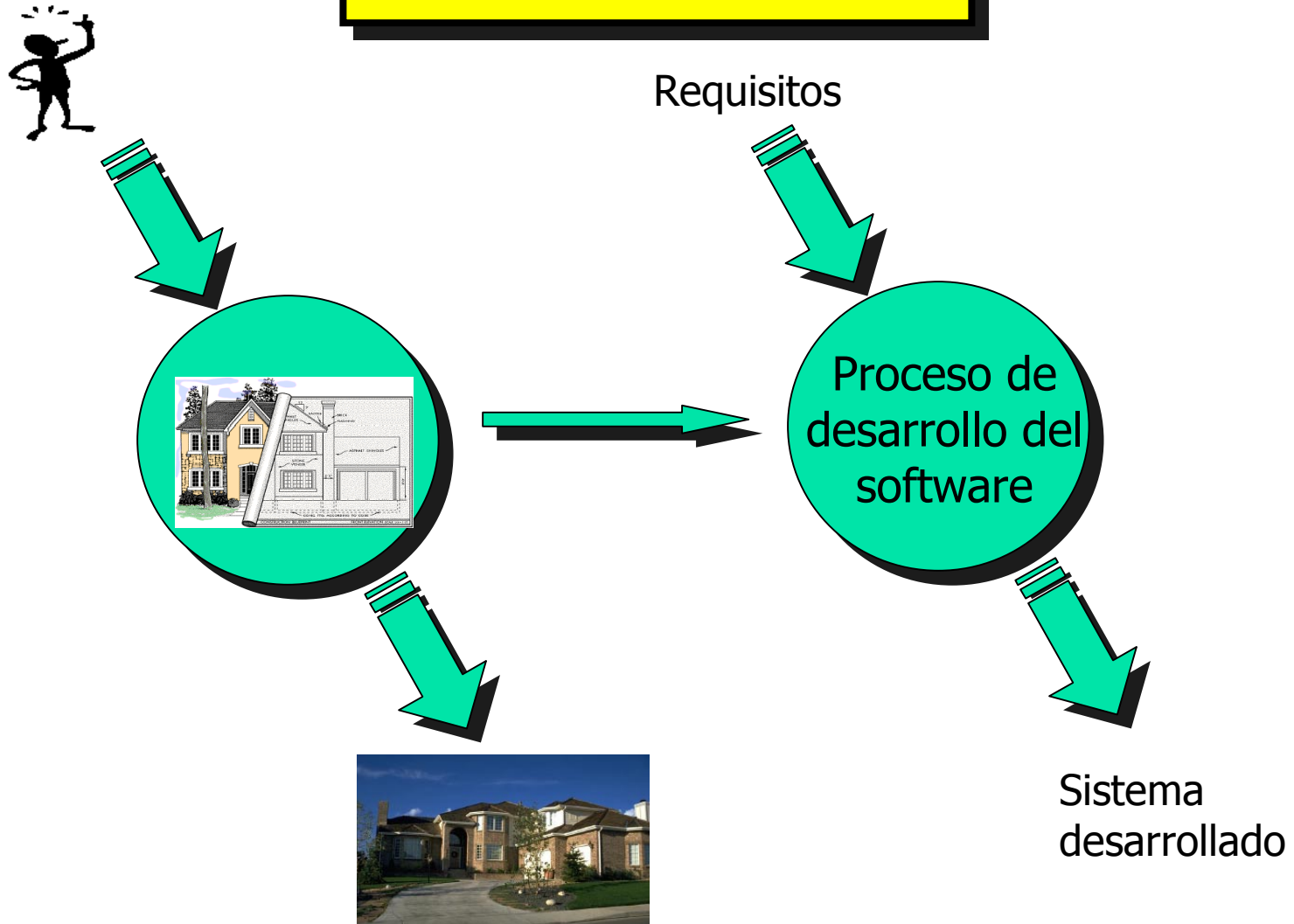
- **Vista lógica del sistema.** Corresponde con los distintos modelos que nos ayudan a especificar, visualizar y documentar el sistema.



- **Vista física del sistema.** Corresponde con el sistema implementado.



Proceso de modelado



¿Cómo modelar?

Visión tradicional:

- El **desarrollo tradicional** del software tomaba una **perspectiva algorítmica**. En esta aproximación la construcción principal eran los procedimientos o funciones.
- Esta visión se centraba en la **descomposición** de los algoritmos en algoritmos más pequeños.
- Este punto de vista hacía que los sistemas **fuera frágiles**, y poco reutilizables.
- Las técnicas de análisis estructurado tenían una clara desconexión entre su **análisis del modelo** y el **diseño del modelo** del sistema. Esto hacía muy difícil mantener a la par el sistema concebido y el sistema implementado.
- Las fases del desarrollo (análisis, diseño, codificación e instalación) se realizaban unas a continuación de las otras. Sólo cuando una había terminado se pasaba a la siguiente.

¿Cómo modelar?

Visión actual:

- El **desarrollo actual** del software toma una **perspectiva orientada a objetos**. En esta aproximación la construcción principal son los **objetos** o las **clases**.
- Esta visión se centra en la **delegación de tareas** a los objetos que contienen los datos.
- Así los sistemas son más fáciles de mantener, modificar y corregir sus errores (sólo es necesario centrarse en el objeto al que le corresponde la tarea).
- Las técnicas de análisis orientado a objetos mantienen una clara conexión entre su **análisis del modelo** y el **diseño del modelo** del sistema.
- Las fases del desarrollo de software se realizan de forma cíclica, interactuando los programadores con los analistas y diseñadores. Esto hace que el software sea más robusto y esté mejor diseñado.

¿Cómo modelar?

- Usa diagramas para modelar tu sistema. Esto te ayudará a comprender, refinar, detectar incoherencias y realizar el sistema.
- Utiliza el conjunto de diagramas necesario para modelar todos los aspectos de tu aplicación, esto te ayudará a hacerte las preguntas adecuadas.
- Utiliza una notación estándar. Así los modelos serán fácilmente comprensibles por otras personas.
- Los diagramas se usan de dos formas:
 - para especificar los modelos del sistema que se está construyendo.
Ingeniería directa.
 - para reconstruir los modelos del sistema ejecutable.
Ingeniería inversa.

Desarrollo del software

¿Qué hacer para construir un sistema?

- Conocer las necesidades. (**Especificaciones del usuario**)
- Representar y formalizar las necesidades. (**Requerimientos**)
- Representar y formalizar el modelo abstracto que describe las necesidades. (**Análisis**)
- Representar y formalizar el modelo concreto que soluciona el problema. (**Diseño**)
- Implementar el modelo en un sistema. (**Implementación**)
- Testar el sistema. (**Testing**)
- Instalar el sistema. (**Instalación**)

Desarrollo del software

Análisis: descripción y modelado del problema.

- Debe mostrar la vista externa del sistema.
- Debe ser comprensible por el cliente.
- Debe suministrar los requerimientos necesarios para el desarrollo del sistema, sin incluir detalles del diseño ni de la implementación.

Diseño: encontrar la solución.

- Está controlado por aspectos de implementación, arquitectura, recursos disponibles, ...
- Debe ser razonablemente eficiente y práctico cuando se implemente.
- Debe manejar detalles de bajo nivel obviados en la fase de análisis. Estructuras de datos, algoritmos, ...
- Durante esta fase se deben optimizar, extender y refinar los modelos de la fase de análisis.

Orientación a objetos

Contenido

Orientación a objetos

- ¿Qué es la orientación a objetos?
- Conceptos básicos en programación OO
 - ¿Qué es un objeto?
 - ¿Qué es una clase?
 - ¿Qué es un interfaz?
 - ¿Cómo se relacionan los objetos?
- Propiedades de la orientación a objetos
- Fases de la metodología orientada a objetos
- Ventajas de la programación OO

¿Qué es la orientación a objetos?

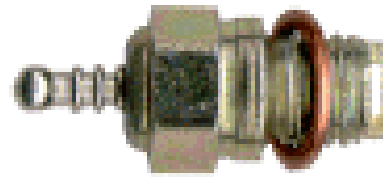
- La orientación a objetos significa que el software está organizado como una colección de objetos que contienen los datos y las funciones para transformarlos. La comunicación entre ellos se realiza a través de mensajes.
 - El objeto se considera una unidad simple e independiente, que contiene sus datos y su propio comportamiento. En contraposición a la programación clásica donde los datos estaban separados de los comportamientos (ej. C).

¿Qué es la orientación a objetos?

- La orientación a objetos es una forma de pensar en la que se trata de imitar el comportamiento del mundo real.

Ej. Tú no vas a la panadería, coges una barra de pan y dejas el dinero en la caja. Sino que vas a la panadería y le pides al panadero que te dé una barra de pan. Él te dice lo que cuesta y te pide el dinero. Tú le pagas y el deja en la caja el dinero que tú le has pagado por la barra.

¿Qué es un objeto?



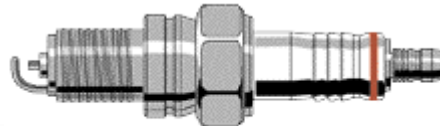
¿Qué es un objeto?

- Una entidad con un comportamiento específico.
- Un objeto es una instancia (un ejemplar, un caso concreto) de una clase (una categoría).
- Un objeto posee un rol según el contexto en el que esté.
- Un objeto tiene una estructura al igual que en el mundo real comparten dos características:
 - **estado:** atributos concretos.
 - **comportamiento:** operaciones que realiza.

¿Qué es un objeto?

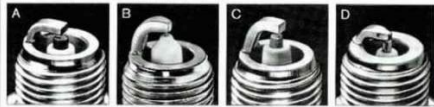
- En programación orientada a objetos un objeto actúa como resultado de una petición externa, mensaje.
- Los atributos pueden ser mutables e inmutables.
- El estado suele influir en el comportamiento.
- El estado de un objeto puede variar según los mensajes que reciba.
- Un objeto corresponde con la parte dinámica del sistema (ejecución).

¿Qué es una clase?

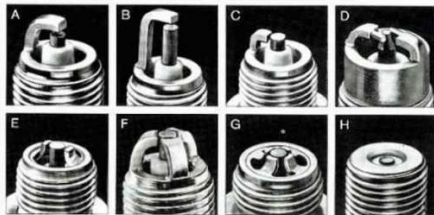


BUJIAS

17) Materiales de los electrodos centrales.
A Material compuesto, B Platino, C y D Plata.

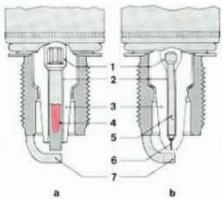


18) Disposiciones de electrodos. Electrodo de una masa (A, C), electrodos de 2, 3 o 4 masas (D, G) y electrodo anular (H).



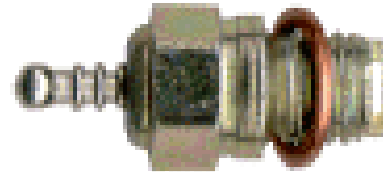
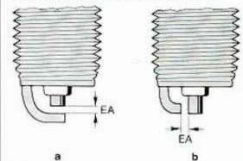
12) Ejemplos de distintas ejecuciones de electrodos en bujías de encendido.

- a con electrodo central compuesto,
- b con electrodo central de platino.
- 1 Masa vitrea conductora,
- 2 Juego para distintas dilataciones térmicas (aumentado),
- 3 Pie del aislador,
- 4 Electrodo central compuesto,
- 5 Clavija metálica de contacto,
- 6 Electrodo central de platino, sinterizado sin entrehierro,
- 7 Electrodo de masa.



13) Disposición y separación de los electrodos (EA).

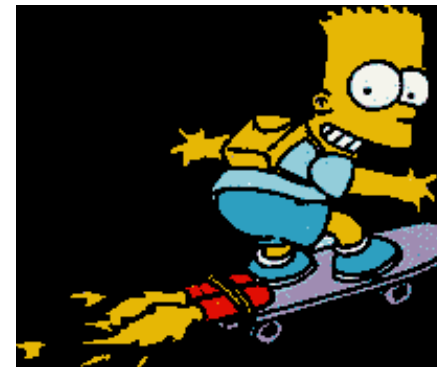
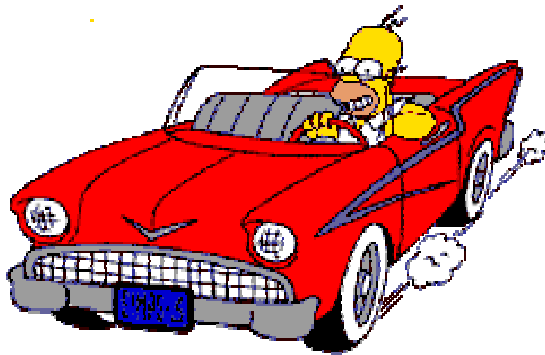
- a Electrodo frontal, b Electrodo lateral.



¿Qué es una clase?

- Una clase es una abstracción de un conjunto de objetos que nos permite comprenderlo.
- Una clase especifica las operaciones y atributos para un conjunto de objetos. Es decir, el comportamiento general que describe el comportamiento de cada objeto de la clase y los atributos generales que se concretaran en cada instancia de la clase.
 - Los atributos son una abstracción de los atributos particulares de los objetos junto con sus tipos.
 - Las operaciones son una abstracción del comportamiento general de los objetos.
- La clase corresponde a la parte estática del sistema

¿Qué es un interfaz?



¿Qué es un interfaz?

- Un interfaz es un conjunto de operaciones que definen una característica concreta del comportamiento de una clase de objetos.
- En programación orientada a objetos corresponde con las cabeceras de los procedimientos.
- Un interfaz puede ser implementado por varias clases a la vez.
- Una clase puede implementar varios interfaces.
- Sirve como mecanismo de abstracción, ya que si un método sólo usa un aspecto del objeto concreto se puede implementar a partir de la interfaz. Así, se consigue que sea valido para todos las clases que implementan dicha interfaz.

¿Cómo se relacionan los objetos?

- Los objetos se relacionan a través de **mensajes**.
- Un **mensaje** es una orden que se le da al objeto. Es una llamada a una de sus operaciones.
- Las operaciones pueden recibir parámetros que definen, cambian o modifican su comportamiento.
- Un mensaje puede provocar un cambio en el estado del objeto.
- Un mensaje consta de tres **componentes**:
 - El objeto al que se dirige el mensaje.
 - El nombre del método a ejecutar.
 - Los parámetros necesarios.

Ej. Le pido al mecánico que me cambie las ruedas delanteras del coche.

Propiedades de la orientación a objetos

- **Abstracción.** La orientación a objetos ayuda a abstraer de forma sencilla los conceptos y mejora su comprensión.
- **Encapsulamiento.** Los datos se encuentran junto con los comportamientos y se ocultan aquellos datos que no se quiere que nadie modifique.
- **Herencia.** Los objetos siguen una jerarquía de clases y heredan de sus padres los atributos y operaciones.
- **Polimorfismo.** Un mismo método vale para diferentes objetos, en concreto para todos aquellas subclases de este.

Propiedades de la orientación a objetos

Abstracción.

- La abstracción corresponde con el proceso de capturar la esencia de un objeto mientras se suprimen o ignoran los detalles irrelevantes. El proceso de la abstracción comienza con la identificación de las características comunes de un conjunto de objetos y continua con la descripción de estas características en lo que se llama una clase. La clase a la que pertenece el objeto.
- La abstracción proporciona un mecanismo crucial que permite que las personas comprendan y se comuniquen.
- Un modo de gestionar la abstracción es mediante clasificaciones jerárquicas. Esta propiedad permite que un sistema se rompa en partes más significativas.

Ej. Los coches tiene unas características generales, cuatro ruedas, un motor, una carrocería y sirven para transportar cosas y personas.

Propiedades de la orientación a objetos

Encapsulamiento.

- El encapsulamiento es el proceso que te permite ocultar los detalles de la implementación. Su ventaja es dual:
 - la encapsulación de los datos en los objetos hace que estos estén protegidos contra accesos inesperados, lo que garantiza la integridad de ellos.
 - la encapsulación hace que el sistema se acople más sencillamente.
 - el encapsulamiento hace que tu aplicación no dependa de la implementación, simplemente de su especificación. Si el comportamiento es el mismo entre ambos objetos se puede sustituir uno por otro.
- Las características de los objetos se describen en la especificación, y los detalles de la implementación no son necesarios para comprender el comportamiento. La especificación se considera un contrato entre las distintas clases.

Propiedades de la orientación a objetos

Herencia.

- Es el proceso mediante el cual un objeto adquiere las propiedades de otro. Por tanto el objeto sólo tiene que describir las características nuevas que lo identifican.
- La herencia permite construir una clase a partir de otras clases ya implementadas, compartiendo todos los atributos, operaciones y asociaciones que ellas posean. Creando con ello una jerarquía de clases.
- La clase hijo hereda las características de la clase padre. Sus atributos y operaciones son accesibles desde la clase hijo como si estuvieran declarados en él.
- Cualquier cambio producido en una superclase es heredado automáticamente por todas sus subclases.

Propiedades de la orientación a objetos

Polimorfismo.

- Es la propiedad que permite que un mismo método se utilice por distintos tipos de objetos, o de que un objeto se vea bajo diferentes formas.
- El polimorfismo permite que una operación pueda existir en diferentes clases, definida de forma distinta para cada una de ellas.
- Permite generar un código más abstracto que el específico para una clase. Por lo tanto se consigue que sea válido en el futuro.
- El polimorfismo se consigue abstrayendo los comportamientos comunes para las distintas subclases de la jerarquía.

Fases de la metodología orientada a objetos

Todas las metodologías orientadas a objetos coinciden en el siguiente esqueleto de fases:

- **Análisis.** Comprensión del problema.
 - **Estático:** Identificar las clases, sus propiedades, sus comportamientos y sus relaciones. Diagrama de clases.
 - **Dinámico:** Identificar los comportamientos, interacciones y escenarios del sistema.
- **Diseño.** Se trata concebir una solución.
 - División del sistema en subsistemas.
 - Asignación de tareas y responsabilidades a las clases.
 - Conseguir las estructuras de datos a través de sucesivos refinamientos.
- **Programación.** Se trata de construir la solución en el lenguaje especificado.

Ventajas de la programación OO

- **Análisis.** Se hace a más alto nivel, ya que nos centramos en los objetos que van a estar en funcionamiento.
- **Comunicación.** La comunicación entre el analista y el usuario es más sencilla.
- **Tolerancia.** Se consigue que los programas sean más tolerantes a fallos ya que es más fácil distribuir las tareas, obligaciones y responsabilidades de cada objeto.

Ventajas de la programación OO

- **Reutilización.** Las clases se pueden utilizar en distintos programas. Esto hace que:
 - La calidad del software mejore.
 - El software sea más robusto, se puede preparar mejor ante errores.
 - El software se pueda extender de forma sencilla.
 - El software se pueda trasladar de un entorno a otro.
- **Verificación.** Es más fácil comprobar que cada objeto cumple con sus requisitos.

El lenguaje UML

Contenido

UML

- ¿Qué es UML?
- ¿Quien creó UML?
- Historia de UML
- Beneficios de usar UML
- Características principales de UML
- Objetivos principales de UML
- Componentes de UML

¿Qué es UML?

- UML es un Lenguaje de Modelado Unificado, basado en una notación gráfica, para crear sistemas software. Es un lenguaje para:
 - **Visualizar** – está dotado de símbolos gráficos con una semántica bien definida.
 - **Especificar** – existen diagramas para describir el comportamiento y las restricciones.
 - **Construir** – un modelo construido en UML puede ser trasladado a un lenguaje de programación como C++ o C#.
 - **Documentar** – a través de los diferentes diagramas es más fácil comprender la aplicación y por tanto documentarla.

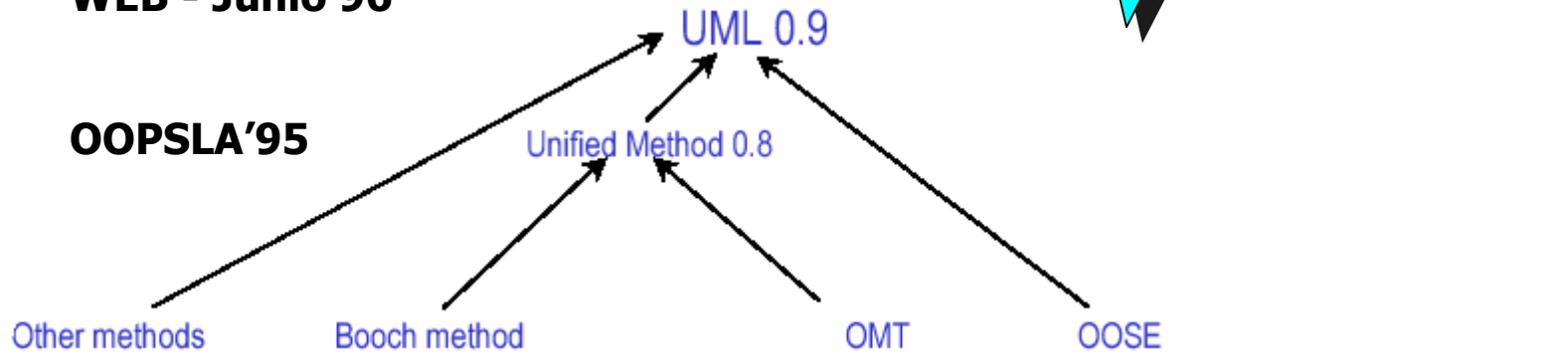
¿Quien creó UML?

- Se origino en el Rational Software Corporation
 - Grady Booch
 - Ivar Jacobson
 - James Rumbaugh
- UML consortium – HP, IBM, MCI, Microsoft, Oracle, Rational, Texas Instruments, etc.
- Fue adoptado como un estándar por el Object Management Group en Noviembre del año 1997.

Historia de UML

OMG Aceptación, Nov'97
Versión final al OMG, Sep'97
Primer envío al OMG, Ene'97
WEB - Junio'96

OOPSLA'95



Beneficios de usar UML

- Tu sistema software se diseña y documenta profesionalmente. Tu sabrás exactamente lo que estás consiguiendo.
- Como el diseño del sistema se realiza antes, el código reutilizable es fácilmente exportado y codificado con la mayor eficiencia. Los costes de desarrollo serán menores.
- Los ‘agujeros’ lógicos pueden ser eliminados en la fase del diseño. Tu software funciona como tú esperas que funcione.
- El diseño del sistema completo te dictará la manera en que se ha de desarrollar. Las decisiones correctas se toman antes de escribir código. De nuevo, tus costes serán menores.
- UML te permite tener una visión general. Por tanto podrás desarrollar sistemas más complejos de forma más eficiente.

Beneficios de usar UML

- Cuando tengamos que modificar el sistema, será más fácil trabajar con un sistema que posee documentación UML. El proceso de estudio y comprensión del sistema a modificar será mucho menor. Por tanto, los costes también.
- Si necesitamos trabajar con otro desarrollador, los diagramas UML le ayudaran a ponerse al día en el desarrollo del sistema de forma rápida. Piensa que son los esquemas de la aplicación en desarrollo.
- Es mucho más sencilla y eficiente la comunicación con el cliente e incluso con los programadores.

Características principales de UML

- UML es un lenguaje de notación.
 - Provee 5 vistas del desarrollo de software.
- Actualmente se encuentra en la Versión 1.5.
- Certificado por el Object Management Group (OMG),
 - como parte de la arquitectura de manejo de objetos.
- Adopta la representación de objetos.
- UML provee diferentes vistas. Vistas para los usuarios finales, analistas, desarrolladores, implantadores, testadores, jefes de proyectos y documentalistas.

Objetivos principales de UML

- Proveer a los usuarios con un lenguaje de modelado visual.
- Soportar extensibilidad y mecanismos de especificación.
- Proveer de una estructura formal para la comprensión del lenguaje de modelado.
- Mejorar las herramientas de manejo de objetos.
- Soportar los conceptos de desarrollo de alto nivel.
- Integrar la notación de las mejores prácticas de la ingeniería del software.

Componentes de UML

- Elementos
- Relaciones
- Diagramas

Elementos de UML

- Elementos estructurales
- Elementos de descripción del comportamiento
- Elementos para agrupar
 - paquetes
- Notas

Elementos estructurales

- Clases.
- Interfaces.
- Colaboraciones.
- Casos de Uso - secuencias de acciones que manejan los resultados observables.
- Clases activas - objetos que poseen uno o más procesadores o hebras de ejecución.
- Componentes - partes físicas de la aplicación que implementan las interfaces.
- Nodos - elementos físicos que representan el hardware.

Elementos de descripción del comportamiento

- **Interacciones** - Un comportamiento que contiene un conjunto de mensajes intercambiados entre un conjunto de objetos para alcanzar un propósito.
- **Estados de la máquina** - Un comportamiento que especifica las secuencias de estados de un objeto o una interacción a través de la que pasa a lo largo de su vida como respuesta a los eventos.

Relaciones de UML

- **Dependencia.**
 - El cambio de un elemento afecta a la semántica del otro.
- **Asociación.**
 - Relación estructural entre los objetos.
- **Generalización.**
 - Relación de herencia.
- **Realización.**
 - Relación de realización entre las interfaces y las clases que las implementan.

Diagramas en UML

- **Diagrama de clases.**
 - Muestra un conjunto de clases y sus relaciones.
- **Diagrama de objetos.**
 - Muestra los objetos y sus relaciones.
- **Diagrama de casos de uso.**
 - Muestra un conjunto de casos de uso, los actores y sus relaciones.
- **Diagrama de secuencias.**
 - Muestra el orden a lo largo del tiempo del paso de mensajes.

Diagramas en UML

- **Diagrama de colaboración.**
 - Muestra la organización estructural de los objetos que envían y reciben mensajes.
- **Diagrama de estados.**
 - Muestra una máquina de estados. Consistente en estados, transiciones, eventos y actividades.
- **Diagrama de actividad.**
 - Máquina de estados que muestra el transcurso de una actividad a otra en el sistema.

Diagramas en UML

- **Diagrama de componentes.**
 - Muestra la organización y dependencias entre un conjunto de componentes del sistema.
- **Diagrama de despliegue.**
 - Muestra la configuración de los nodos en ejecución y los componentes vivos en ellos.

Desarrollo de aplicaciones

Desarrollo de aplicaciones

- **Recoger los requisitos.**
- **Análisis.**
- **Diseño.**
- **Desarrollo.**
- **Despliegue o Implantación.**

Desarrollo de aplicaciones

Recoger los requisitos.

- **El proceso del negocio.**
- **El análisis del dominio.**
- **Identificar el sistema.**
- **Descubrir los requisitos del sistema.**
- **Presentar los resultados al cliente.**

Desarrollo de aplicaciones

Recoger los requisitos.

Esta parte consiste en varias acciones. Es importante comprender lo que el cliente quiere.

- **El proceso del negocio.** El analista obtiene una idea de lo que el cliente le propone. Entrevistando al cliente se consigue una visión del comportamiento del sistema

Producto: Diagrama de actividad.

- **El análisis del dominio.** El analista entrevista al cliente para comprender los elementos de su sistema.

Producto: Diagrama de clases, a un nivel muy alto.

Desarrollo de aplicaciones

Recoger los requisitos.

- **Identificar el sistema.** El equipo de desarrollo identifica el sistema del que dependerá el nuevo sistema.

Producto: Diagrama de despliegue.

- **Descubrir los requisitos del sistema.** En esta sesión se toman las decisiones generales sobre la aplicación, desde el punto de vista de los usuarios, desarrolladores y la organización del cliente.

Producto: Diagrama de paquetes y se refinan los diagramas de clases.

- **Presentar los resultados al cliente.** Se presentan los resultados al cliente y se analizan los posibles cambios.

Desarrollo de aplicaciones

Análisis.

- **Comprender el uso del sistema.**
- **Analizar cada caso de uso.**
- **Refinar el diagrama de clases.**
- **Analizar los cambios de estado en los objetos.**
- **Definir las interacciones entre los objetos.**
- **Analizar la integración con el sistema.**

Desarrollo de aplicaciones

Análisis.

Algunas de las acciones que se realizan aquí se realizan en paralelo con la etapa de recogida de requisitos.

- **Comprender el uso del sistema.** En esta etapa se junta el grupo de desarrollo con los usuarios potenciales para identificar los actores de cada caso de uso y los actores que se benefician de él. (Un actor puede ser un sistema, o una persona.)

Producto: Diagramas de casos de uso .

- **Analizar cada caso de uso.** Continuar el trabajo con los usuarios potenciales tratando de descubrir los paso de cada caso de uso.

Producto: Descripción de los pasos de los casos de uso.

Desarrollo de aplicaciones

Análisis.

- **Refinar el diagrama de clases.** El encargado de modelar los objetos refina el diagrama de clases. Añade los nombres de las asociaciones, clases abstractas, multiplicidades, generalizaciones y agregaciones.

Producto: Diagrama de clases refinado.

- **Analizar los cambios de estado en los objetos.** Se refina el modelo mostrando los cambios del estado de los objetos donde sea necesario.

Producto: Diagrama de estados.

Desarrollo de aplicaciones

Análisis.

- **Definir las interacciones entre los objetos.** El grupo de desarrollo ya tiene un conjunto de diagramas de casos de uso y diagramas de clases, así que ya se pueden definir las interacciones entre los objetos. El encargado del modelado de objetos desarrolla un conjunto de diagramas que incluye el cambio de estado de los objetos

Producto: Diagramas de secuencia y colaboración.

Desarrollo de aplicaciones

Análisis.

- **Analizar la integración con el sistema.** El ingeniero de sistemas, procediendo en paralelo con los pasos anteriores, recoge los detalles necesarios para la integración en el sistema. El tipo de comunicación necesaria, la arquitectura y organización de la red, el tipo de las bases de datos, ...

Producto: Diagrama detallado de implantación y si es necesario modelos de los datos.

Desarrollo de aplicaciones

Diseño.

- Desarrollo y refinamiento de los diagramas de clases.
- Desarrollo de los diagramas de componentes.
- Planear el despliegue.
- Diseño y prototipo de la interfaz de usuario.
- Diseño de test.
- Comienzo de la documentación.

Desarrollo de aplicaciones

Diseño.

En esta etapa el equipo tiene los diagramas generados en la anterior etapa y se diseña la solución. Diseño y análisis suelen producirse en sucesivos ciclos hasta que se encuentra la solución.

- **Desarrollo y refinamiento de los diagramas de clases.** Los programadores con los diagramas de clases generan los correspondientes diagramas de objetos examinando cada operación y desarrollan los diagramas de actividad. Estos diagramas servirán como base en la etapa de desarrollo.

Producto: Diagramas de actividad.

Desarrollo de aplicaciones

Diseño.

- **Desarrollo de los diagramas de componentes.** Los programadores se encargan de visualizar los componentes y mostrar las dependencias entre ellos.

Producto: Diagramas de componentes.

- **Planear el despliegue.** El ingeniero de sistemas con los diagramas de componentes planea el despliegue y la integración con el sistema. Creando los diagramas que muestran donde residirán los componentes.

Producto: Parte del diagrama de componentes.

Desarrollo de aplicaciones

Diseño.

- **Diseño y prototipo de la interfaz de usuario.** Otra nueva sesión con los usuarios. El analista gráfico trabaja con los usuarios para desarrollar el interfaz de la aplicación.
Producto: Prototipos de pantallas.
- **Diseño de test.** Preferiblemente un desarrollador o un especialista en test externo al grupo utiliza los diagramas de casos de uso para el desarrollo de la generación de herramientas de test automáticas.
Producto: Aplicaciones de testing.

Desarrollo de aplicaciones

Diseño.

- **Comienzo de la documentación.** El documentalista junto con los diseñadores desarrolla una estructura de alto nivel de los documentos.

Producto: Estructura de los documentos.

Desarrollo de aplicaciones

Desarrollo.

- Generar el código.
- Testar el código.
- Construir la interfaz de usuario y conectarla con el código y los test.
- Completar la documentación.

Desarrollo de aplicaciones

Desarrollo.

Con las etapas anteriores esta etapa debería ser sencilla y rápida.

- **Generar el código.** Con el diagrama de clases, objetos actividades y componentes los programadores generan el código.

Producto: El código.

- **Testar el código.** Esta acción es cíclica con la anterior, hasta que el código no la pase.

Producto: Resultado de los test.

Desarrollo de aplicaciones

Desarrollo.

- **Construir la interfaz de usuario y conectarla con el código y los test.** Se construye la interfaz de usuario y se conecta con el código. Es necesario testar el buen funcionamiento de de la interfaz.

Producto: Sistema completo funcionado.

- **Completar la documentación.** Durante la etapa de desarrollo se va completando la documentación en paralelo con los otros trabajos.

Producto: Documentación del sistema.

Desarrollo de aplicaciones

Despliegue o Implantación.

- **Planificación de copias de seguridad y recuperación.**
- **Instalación del sistema final en el hardware apropiado.**
- **Testar el sistema instalado.**

Desarrollo de aplicaciones

Despliegue o Implantación.

Cuando el desarrollo se ha completado el sistema se implanta en el hardware adecuado y se integra con el sistema.

- **Planificación de copias de seguridad y recuperación.** El ingeniero de sistema desarrolla un plan con los pasos a seguir cuando el software fallé. Puede empezar en paralelo al desarrollo.

Producto: Plan de recuperación del software.

- **Instalación del sistema final en el hardware apropiado.** Esta acción se realiza por el ingeniero de sistemas con la ayuda de los programadores.

Producto: Despliegue del sistema completo.

Desarrollo de aplicaciones

Despliegue o Implantación.

- **Testar el sistema instalado.** El equipo de desarrollo comprueba que el software instalado funciona.
Producto: Resultado de los test.
- **Celebrarlo.** El equipo se junta y se va celebrarlo junto, vuelve a su casa pronto ya que al día siguiente hay que trabajar.

Producto: Dolor de cabeza a la mañana siguiente.

Vistas

- **La vista funcional. (Requisitos)**
 - Comportamiento del sistema como es visto por el analista, usuarios finales y testadores. Casos de uso.
- **La vista estática. (Diseño)**
 - Clases, interfaces y colaboraciones que forman el vocabulario del problema y su solución.
- **La vista dinámica. (Diseño)**
 - Hilos y procesos que forman el sistema concurrente y su mecanismo de sincronización. Corresponde con el modelado del comportamiento interno de los objetos.

Vistas

- **La vista de interacción. (Diseño)**
 - Abarca la interacción y colaboración entre los distintos objetos del sistema. Corresponde también con el modelado del comportamiento de los objetos pero se refiere a sus relaciones.
- **La vista de implementación. (Desarrollo)**
 - Abarca los componentes y ficheros que se usan para ensamblar y realizar el sistema físico.
- **La vista del despliegue o implantación. (Despliegue)**
 - Los nodos que forman la topología del sistema hardware en el que se ejecuta.

Diagramas de UML

Contenido

- **Diagramas de UML**
 - Diagrama de clases y objetos
 - Diagrama de paquetes
 - Diagrama de casos de uso
 - Diagrama de estados
 - Diagrama de actividad
 - Diagrama de secuencia
 - Diagrama de colaboración
 - Diagrama de componentes
 - Diagrama de despliegue
 - Conclusiones

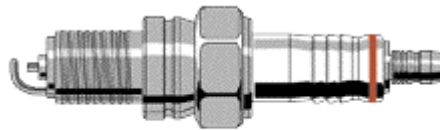
Diagrama de clases y objetos (Vista estática)

Contenido

Diagrama de clases y objetos

- Introducción
- Visualización de una clase
 - Atributos y operaciones
 - Visibilidad
 - Estilo
 - Agrupar clases
- Asociaciones entre clases
 - Herencia y generalización
 - Agregaciones
 - Interfaces y realizaciones
- Metodología

Introducción



BUJIAS



Introducción

- Permite describir las clases, interfaces, objetos y sus relaciones estructurales.
- La descripción que se produce es independiente de la solución técnica elegida: se enfoca hacia el problema.
- Describe el “qué” pero no el “cómo”.
- Unifica lo concerniente a datos y operaciones sobre los mismos (en oposición a los desarrollos estructurados).
- No incluye aspectos relacionados con el comportamiento dinámico de los objetos.
- Es la base sobre la que se construyen las demás vistas.
- Elementos clave: clasificadores (clases, interfaces, etc.) y relaciones.
- Estos diagramas son los más comunes en el modelado de sistemas orientados a objetos y cubren la vista de diseño estática.

Introducción

- Una **clase** es una abstracción de un conjunto de objetos con propiedades (atributos) similares, y comportamiento (operaciones) común.
- Un **objeto** es una instancia de una clase. Una concreción de la clase.
 - Sus características son:
 - Estado
 - Comportamiento
 - Identidad
 - El estado y el comportamiento están unidos:
 - El comportamiento depende del estado actual.
 - El estado puede ser modificado por el comportamiento.

Visualización de una clase

Las clases se representan con un rectángulo. El nombre aparece centrado en la parte superior del rectángulo.

Ej:

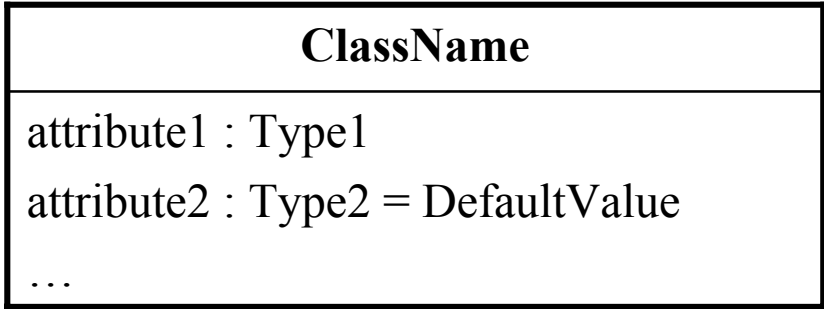


A UML class diagram example showing a single class. It is represented by a rectangle with the text **ClassName** centered in the top portion.

```
classDiagram
    class ClassName
```

Los atributos se representan separados del nombre de la clase por una línea.

Ej:

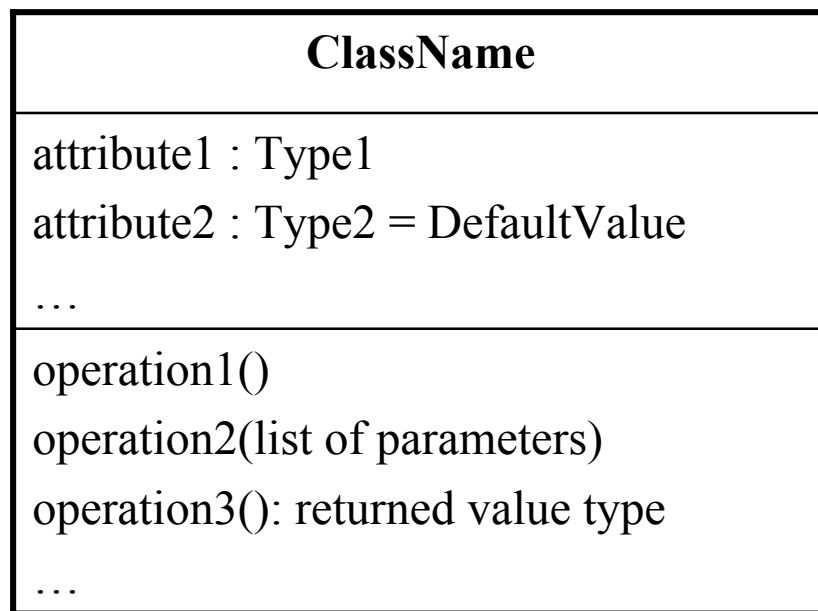


A UML class diagram example showing a class with attributes. The rectangle is divided into two sections. The top section contains the text **ClassName**. The bottom section contains the attributes: `attribute1 : Type1`, `attribute2 : Type2 = DefaultValue`, and `...`.

```
classDiagram
    class ClassName {
        attribute1 : Type1
        attribute2 : Type2 = DefaultValue
        ...
    }
```


Visualización de una clase

Las operaciones se representan separadas de los atributos de la clase por una línea.

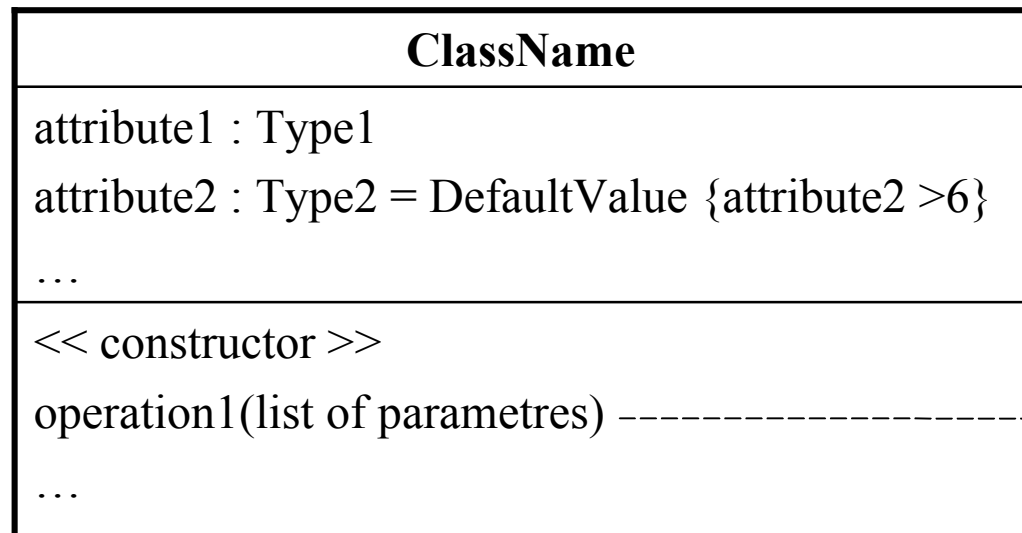


La descripción de la clase se puede ir completando gradualmente durante la transición del análisis al diseño.

Visualización de una clase

Otras cosas que se pueden añadir a los atributos y operaciones son:

- **Estereotipos:** Texto entre << >>.
- **Restricciones:** Texto encerrado entre llaves.
- **Notas:** Se suelen añadir a los atributos y a las operaciones y dan información adicional. Una nota puede contener gráficos y texto. (Se pueden añadir en cualquier diagrama)



Esto es una
nota sobre la
operación

Visualización de una clase

Las **clases abstractas** (clases que no proveen objetos) y **operaciones abstractas** (operaciones sin implementación en la clase) se representan con el nombre en cursiva.

Ej:

<i>AbstractClass</i>

Los métodos y atributos **estáticos** (no necesitan un objeto de la clase para acceder a ellos, pueden ser llamados con el nombre de la clase) se representan subrayados.

Ej:

User
id : int <u>lastUsedID</u>
<u>New()</u> Delete()

Estilo y notación

- Los nombres de las clases son, por convención, una palabra con la primera letra en mayúscula. Si la clase contiene varias palabras, se unen y se pone en mayúscula cada letra de cada palabra.
 - Ej: NombreDeLaClase
- Los nombres de los atributos, las operaciones y los paquetes son, por convención, una palabra escrita en minúscula. Si el nombre consiste en varias palabras, se unen y se pone en mayúscula cada letra de cada palabra excepto de la primera.
 - Ej: nombreDelAtributo

Visibilidad

La visibilidad se aplica a los atributos y operaciones e indica el alcance de uso de esos atributos por otras clases y se sitúa antes de ellos.

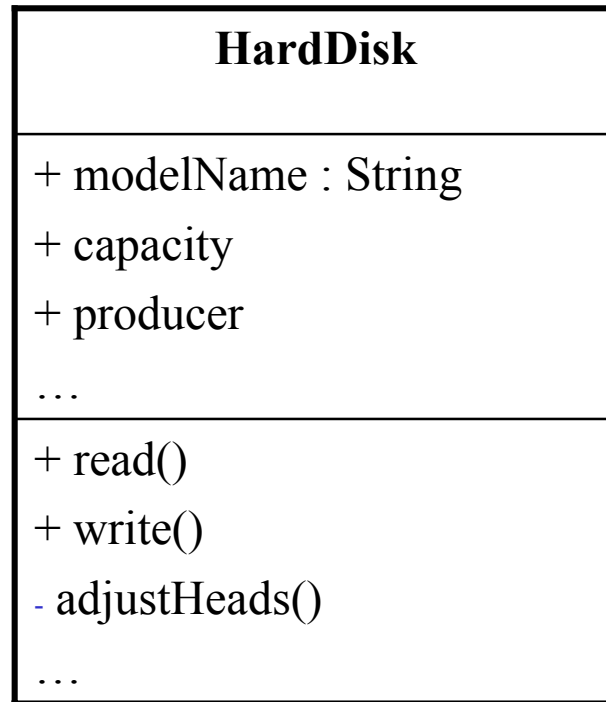
Existen cuatro niveles de visibilidad:

- **público** (+) visible para todas las clases
- **paquete** (~) visible para las clases del mismo paquete
- **protegido** (#) visible para las clases que heredan de esta
- **privado** (-) visible únicamente en la clase original

La ausencia de visibilidad no indica que está indefinida o es pública, sino que no se muestra.

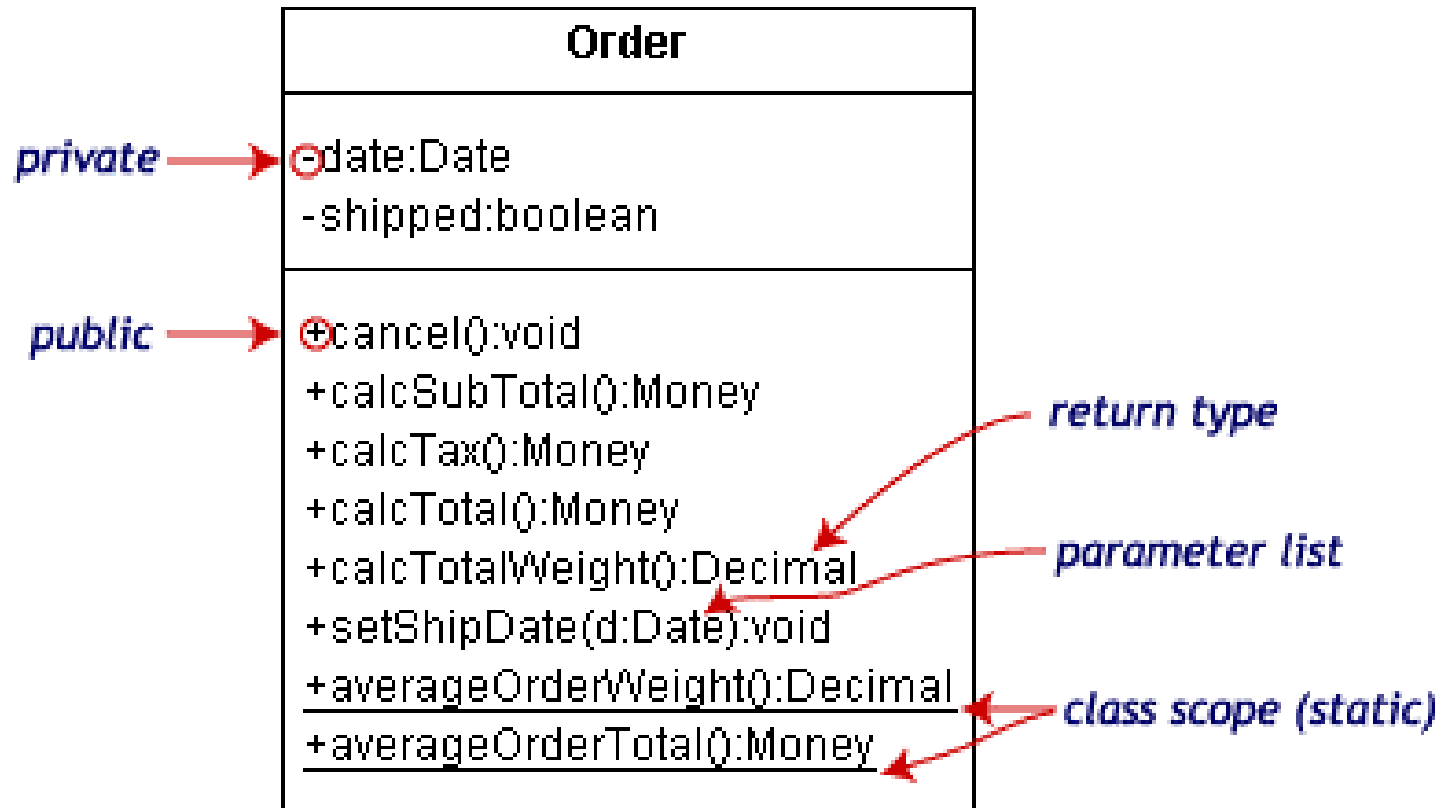
La visibilidad se puede mostrar también con las palabras (*public*, *package*, *protected*, *private*)

Visibilidad- Ejemplo:



Operaciones privadas y públicas dentro de un disco duro

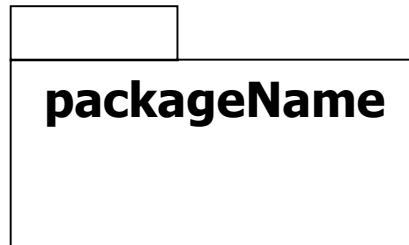
Resumen



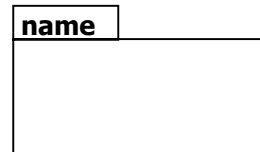
Agrupar clases

Las clases se agrupan en **paquetes**. Estos son simplemente un conjunto de clases. Los paquetes se relacionan entre ellos al igual que las clases.

- La representación de un paquete es la siguiente:



Un paquete con el nombre arriba, significa que es un elemento que modela el paquete.



Cuando queremos indicar que una clase pertenece a un paquete se suele colocar el nombre del paquete antes de la clase separado por ::, o se incluye la clase dentro del paquete.

Ej: datos::Point

¿Cómo derivar las clases?

En la entrevista con el cliente estaremos alerta a los **nombres** que usan para describir las entidades de su negocio. Esos nombres se convertirán en los nombres de nuestras **clases** en el modelo.

Se debe estar también alerta a los **verbos** que el cliente use. Ya que, estos constituirán las **operaciones** de nuestras clases.

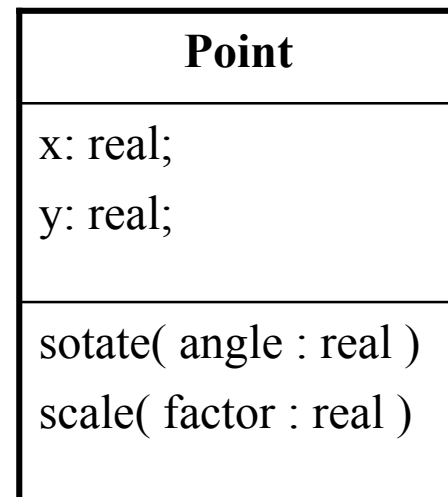
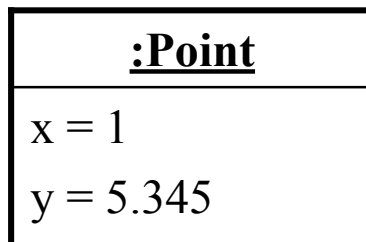
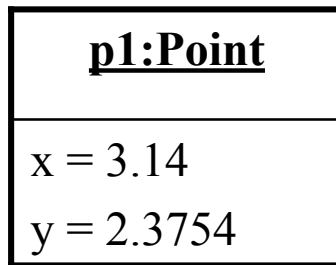
Los **atributos** surgirán como **nombres relacionados** con los nombres de las **clases**.

Las **interfaces** surgen de los clases que comparten **comportamientos**. Es decir, surgen de las operaciones que poseen la misma signatura y los comportamientos son similares. En muchos casos puedes codificar las operaciones en una clase (probablemente abstracta) y rehusarlas en las otras clases.

Objetos

Las **objetos**, al igual que las clases, se representan con un rectángulo. El nombre del objeto aparece en la parte de arriba subrayado, y los valores de los atributos aparecen separados por una línea debajo de este.

Ej:

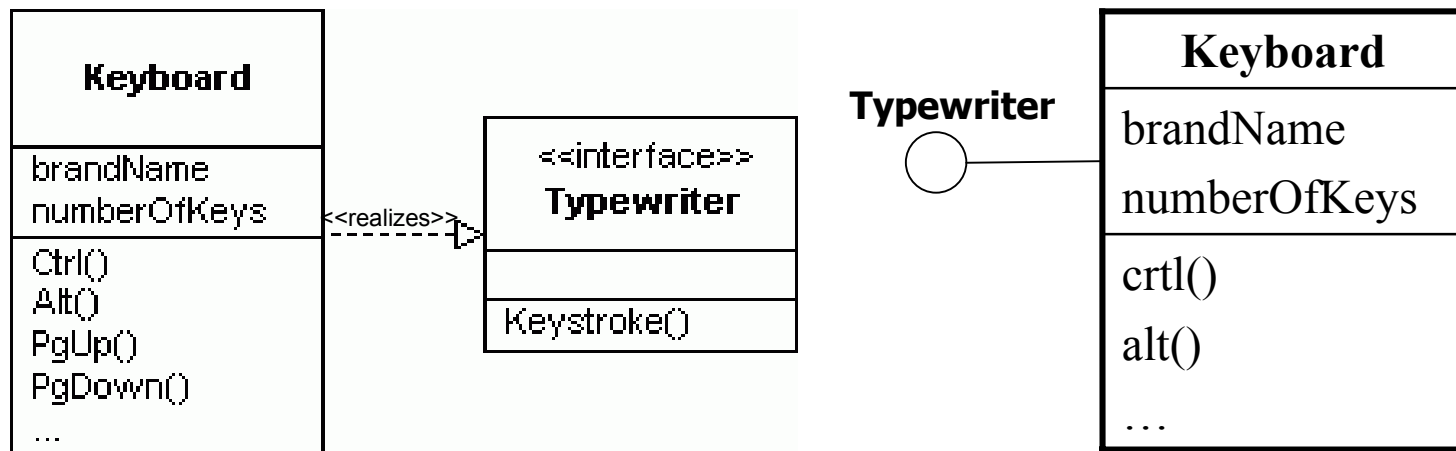


Interfaces e implementación

Un **interfaz** es un conjunto de operaciones que especifican algún aspecto del comportamiento de una clase.

Las **interfaces** se modelan igual que las clases, con el rectángulo, pero no tienen atributos. Otra forma es con un pequeño círculo unido a la clase, esto ofrece menos información.

Ej:



Para distinguir las interfaces de las clases, en el estereotipo ponemos `<<interface>>`.

Relaciones

Las relaciones muestran como se conectan las clases de nuestro modelo y proveen un diagrama del mundo que estamos modelando.

- Representan conexiones semánticas entre los elementos del modelo.
- Las relaciones suelen corresponder con los verbos empleados en las especificaciones del sistema.

Asociaciones



- posee
- multa
- conduce



Asociaciones

- Las uniones que conectan los objetos deben ser mostradas, de forma abstracta, a través de las clases: para cada familia de uniones entre los objetos, existe una unión entre las clases de los objetos.
- Como todos los objetos son una instancia de una clase, las uniones entre los objetos son una instancia de las uniones entre las clases.
- Una asociación es una **abstracción de las uniones** que existen entre las instancias de la clase (objetos).
- La asociación expresa una relación bidireccional entre las clases. La asociación no está contenida en las clases y tampoco está subordinada a ellas. Refleja una conexión que existe entre el domino de aplicación.

Asociaciones

- Las asociaciones se muestran como una línea conectando las dos clases con el nombre de la asociación encima de la línea.



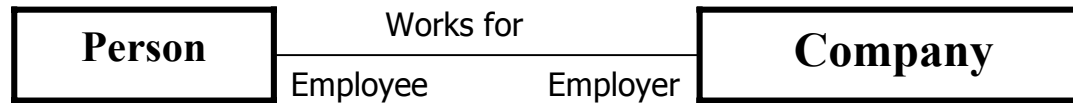
- La dirección en la que la asociación debe ser leída, se puede indicar usando un triángulo apuntando a la clase.



Asociaciones

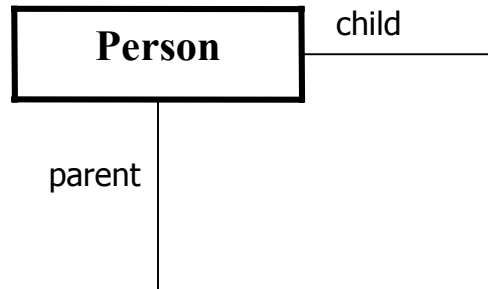
- Cuando una clase se asocia con otra, cada una de ellas juega un **rol** en dicha asociación. Los roles pueden ser mostrados en la asociación escribiéndolos cerca de la clase que juega ese rol.

Ej:



- Cuando una asociación conecta una clase consigo misma, se llama **asociación reflexiva**.

Ej:

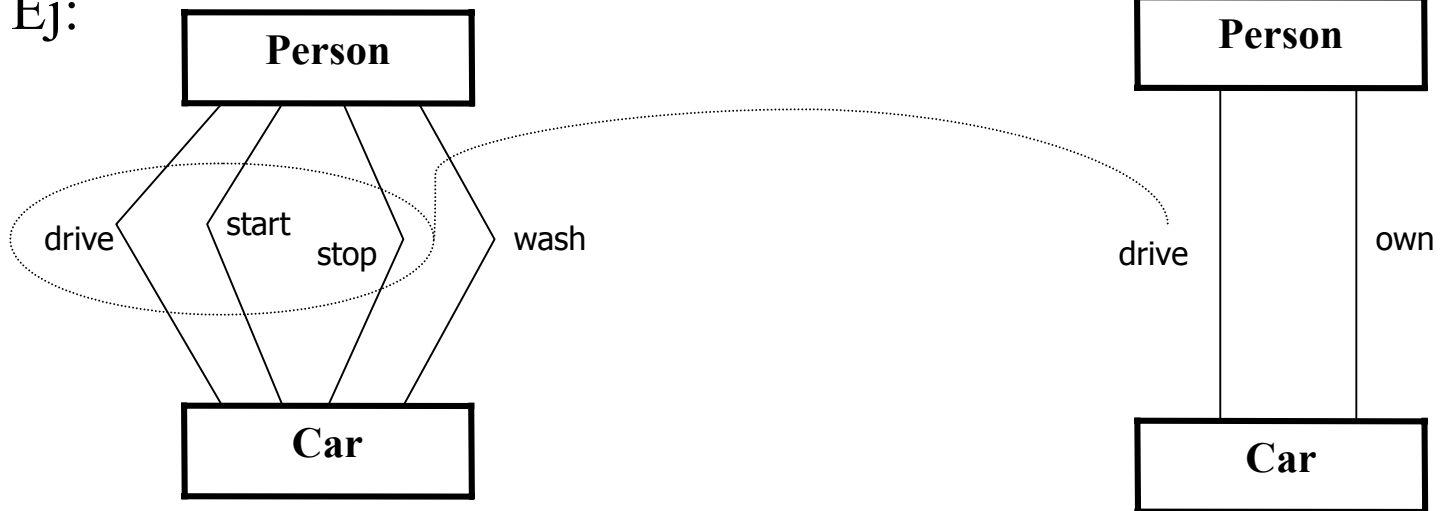


Asociaciones

Problemas:

- La presencia de una lista larga de asociaciones puede ser sospechosa. Se debe pensar que se pueden juntar dichas asociaciones.
- A veces es fácil crear la misma asociación con nombres diferentes.

Ej:



Asociaciones

Multiplicidad

- Cada rol de una asociación tiene una multiplicidad que indica cuantos objetos de la clase se relacionan con la otra.
- La multiplicidad es una información asociada con el rol.
- Expresiones de multiplicidad:
 - 0..n de cero a n;
 - n n y sólo n
 - 0..* cero o más
 - n..* n o más
- La multiplicidad se expresa como un rango de enteros, o una lista de rangos de enteros (separados por comas).

Ej: 1,5, 7..20, 40..*

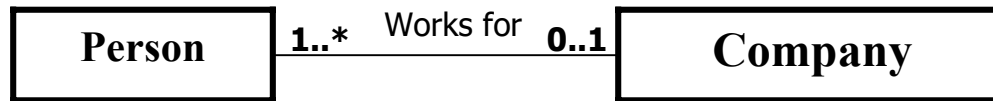
0,2,4,8

Asociaciones

Multiplicidad

- La multiplicidad de una asociación binaria especifica el numero de objetos de una clase que se relacionan con uno o más objetos de la otra.

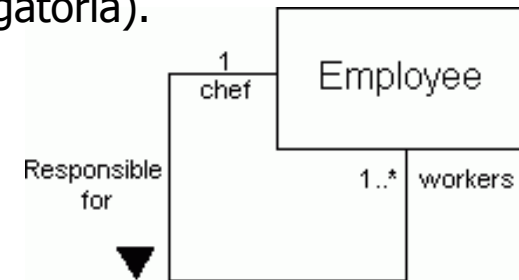
Ej:



Esto indica que una persona sólo puede trabajar a lo sumo en una compañía y no está obligada (participación opcional).

Pero una compañía puede tener una cantidad ilimitada de trabajadores al menos uno (participación obligatoria).

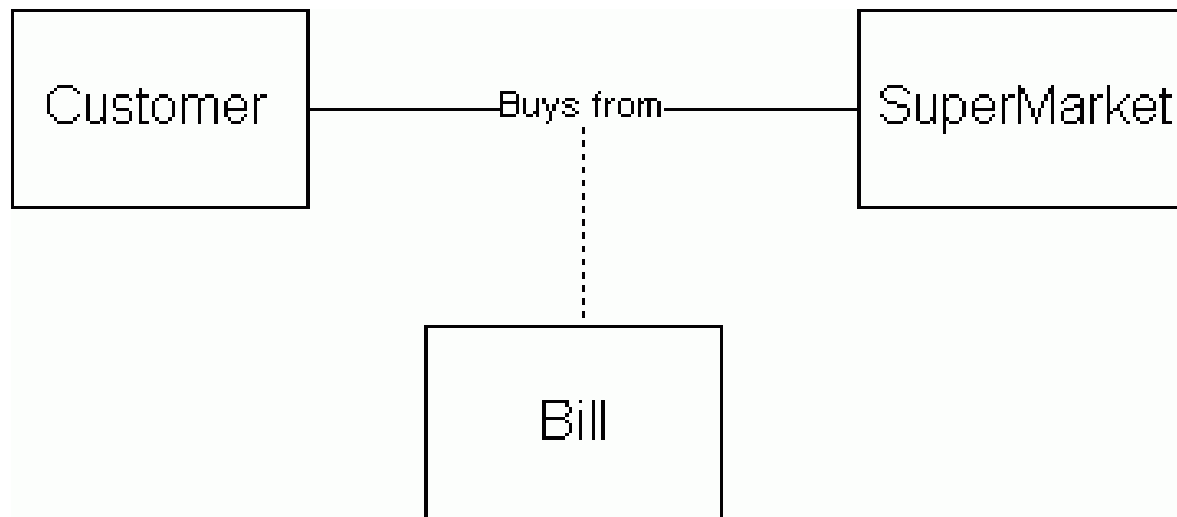
Este segundo ejemplo nos indica que un chef sólo puede existir si existen trabajadores de los que ser responsable.



Asociaciones

- Algunas veces las asociaciones entre dos clases siguen una regla. Se puede añadir esa regla a través de una restricción cerca de la línea de la asociación.
- Las asociaciones pueden tener atributos y operaciones. En este caso se dice que tenemos una **clase asociada**.

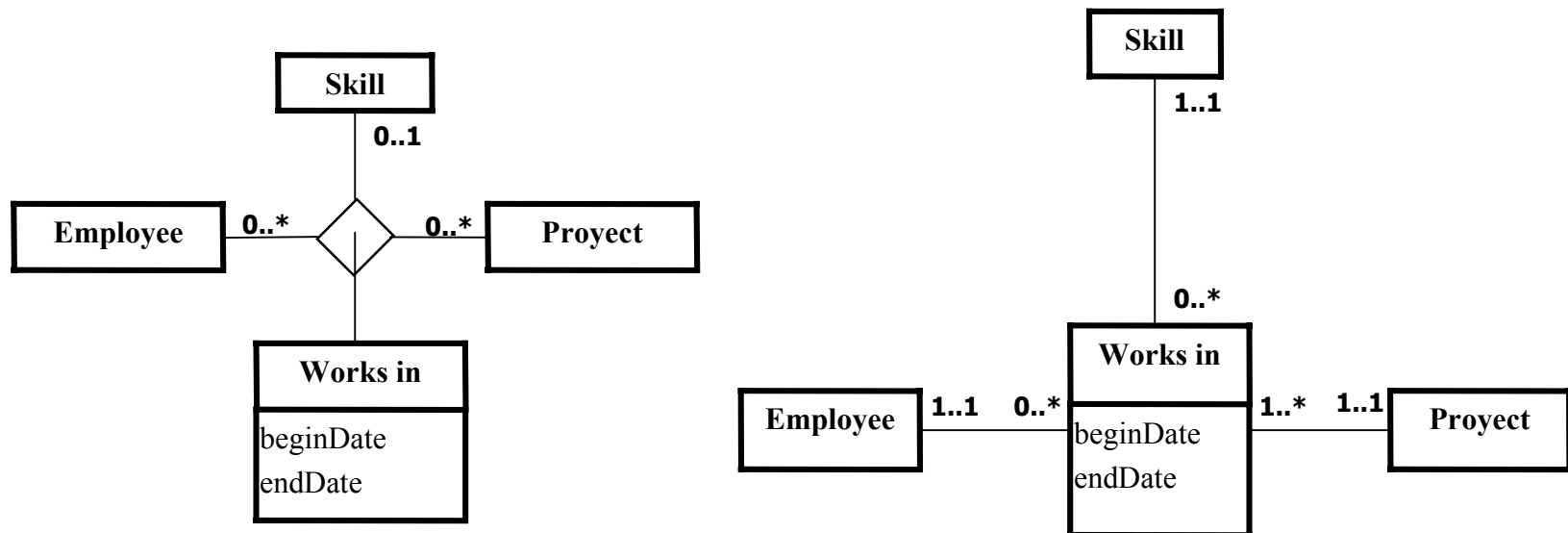
Ej:



Asociaciones

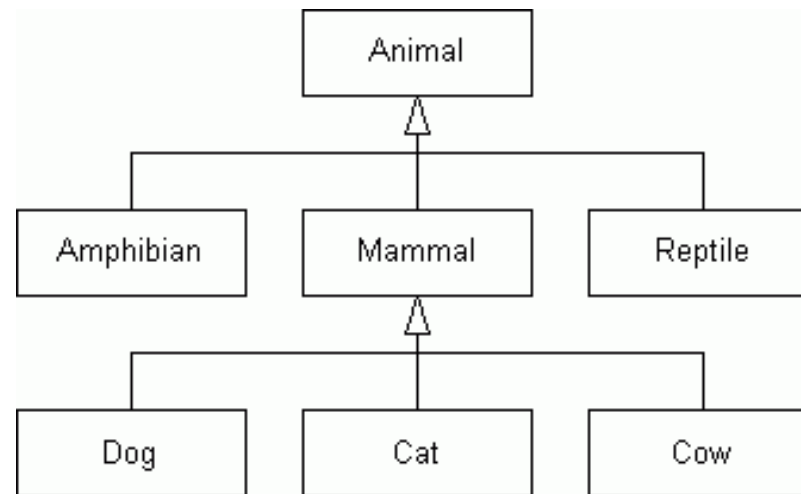
- Las asociaciones pueden ser más complejas que una clase conectada a otras. Una misma clase se puede conectar por varias asociaciones a otra y varias clases pueden conectarse a otra.

Ej:



Herencia y generalización

- **Herencia:** se trata de un proceso mediante el cual un objeto adquiere las capacidades de otro objeto.
- La herencia se representa mediante una flecha con un dardo abierto.
Ej: Un perro es un animal. De ahí, deduces que nace, crece, se alimenta, duerme... Pero un perro también es un mamífero. De ahí, deducimos otras propiedades.



- Una clase (la clase hijo) hereda atributos y comportamiento de otra (la clase padre). Solo hace falta redefinir las operaciones específicas.

Herencia y generalización

¿Cómo descubrir la herencia?

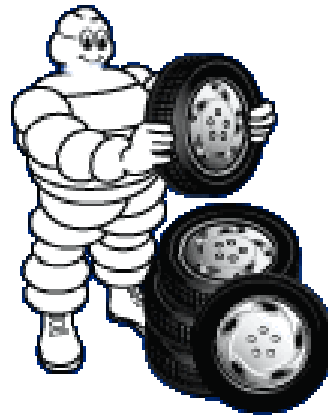
- **Generalización y abstracción.**

- Varias clases comparten unos mismos atributos y comportamientos similares. En este caso suele ser bueno generar una clase abstracta que se sitúa en lo alto de la jerarquía y las demás heredan de ella esos atributos y operaciones.
- Antes de hacer el diseño de las clases abstraemos comportamientos y empezamos a generar el árbol de arriba para abajo.

- **Especialización:** Se trata de especializar las clases en lo alto de la jerarquía con los nuevos comportamientos deseados.

- Una herencia optima se consigue a partir de refinamientos sucesivos en el árbol de herencia.

Agregaciones

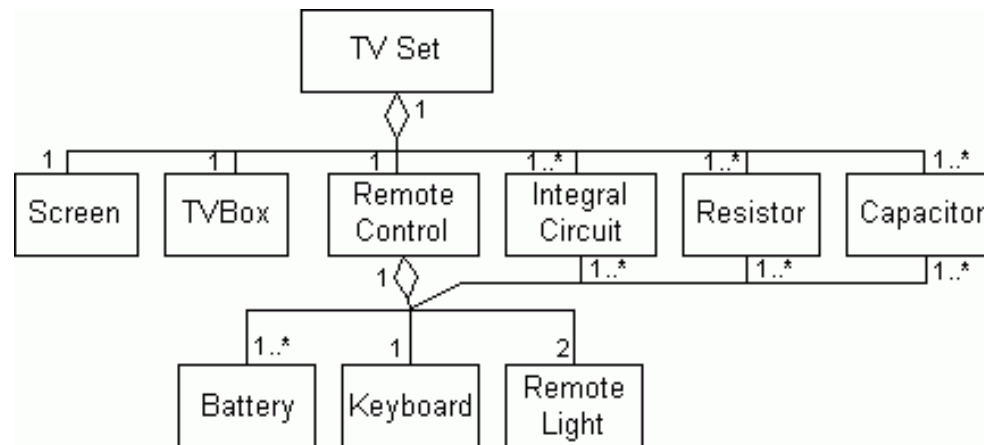


Agregaciones

A veces la clase consiste en un conjunto de componentes de otras clases. Esto es un tipo especial de relación llamada **agregación**.

- Una **agregación** se representa con una línea terminada con un diamante abierto.

Ej:



Cuando algunos componentes son opcionales, se usa una restricción de tipo {OR} y se utiliza una línea de puntos.

Composiciones

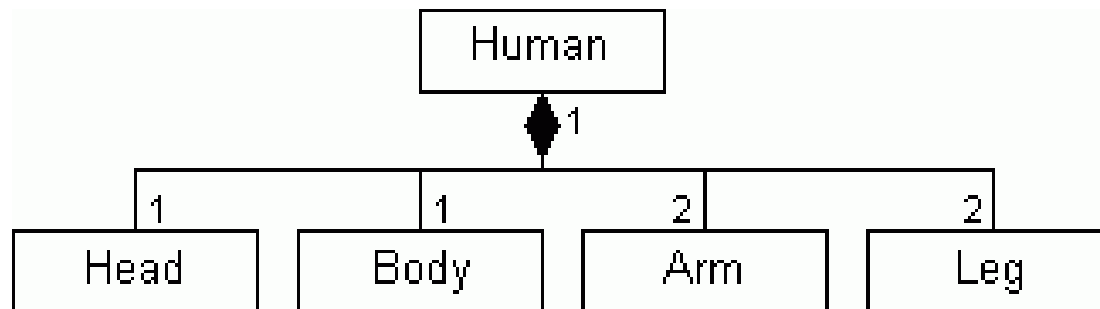


Composiciones

La composición es un tipo de agregación fuerte. Es decir, un objeto está hecho de exactamente los componentes a los que se refiere la composición. Incluso coinciden los tiempos de vida del objeto completo y los componentes. Es decir, si desaparece el objeto desaparecen los componentes.

- Una **composición** se representa con una línea terminada con un diamante cerrado.

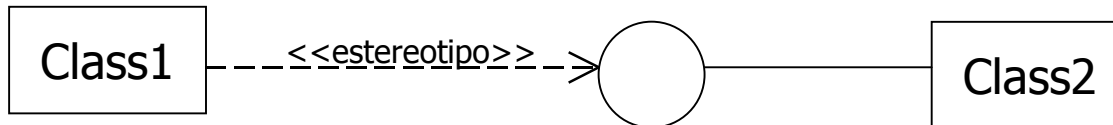
Ej:



Dependencias

- Una relación de **dependencia** se utiliza entre dos clases o entre una clase y una interfaz. Indica que una clase requiere de otra para proporcionar alguno de sus servicios (en realidad la asociación y generalización son dependencias "especiales", con semántica propia muy concreta).
- No requiere un conjunto de objetos para su interpretación.
- Indica que un cambio en el elemento origen de la dependencia puede significar un cambio en el elemento destino de la dependencia.
- Una relación de dependencia se expresa mediante una flecha a trozos con el estereotipo arriba.

Ej:



Dependencias

Tipos de dependencias:

- **Derivación** «derive»: indica que un elemento se puede computar a partir de otro.
- **Traza** «trace»: existe alguna conexión entre elementos de diversos modelos, para no perder de vista que un cambio en un modelo puede afectar a otro.
- **Realización** «realize»: correspondencia entre una especificación y su implementación.
- **Refinamiento** «refine»: relación entre dos versiones de un elemento, uno de ellos suele ser una versión incompleta del otro.

Estas cuatro son dependencias de abstracción: relacionan dos versiones de la misma cosa subyacente.

Dependencias

Tipos de dependencias:

- **Uso** «use»: un elemento requiere la presencia de otro para su correcto funcionamiento (en el mismo modelo). Suele deberse a temas de implementación. Por ejemplo requisitos del compilador que necesita la definición de una clase para compilar otra.
 - **Llamada** «call»: un método en una clase llama a una operación de otra clase.
 - **Instanciación** «instantiate»: un método en una clase crea una instancia de otra clase.
 - **Parámetro** «parameter»: relación entre una operación y sus parámetros.
 - **Envío** «send»: relación entre el emisor de una señal (comunicación asíncrona) y el receptor.

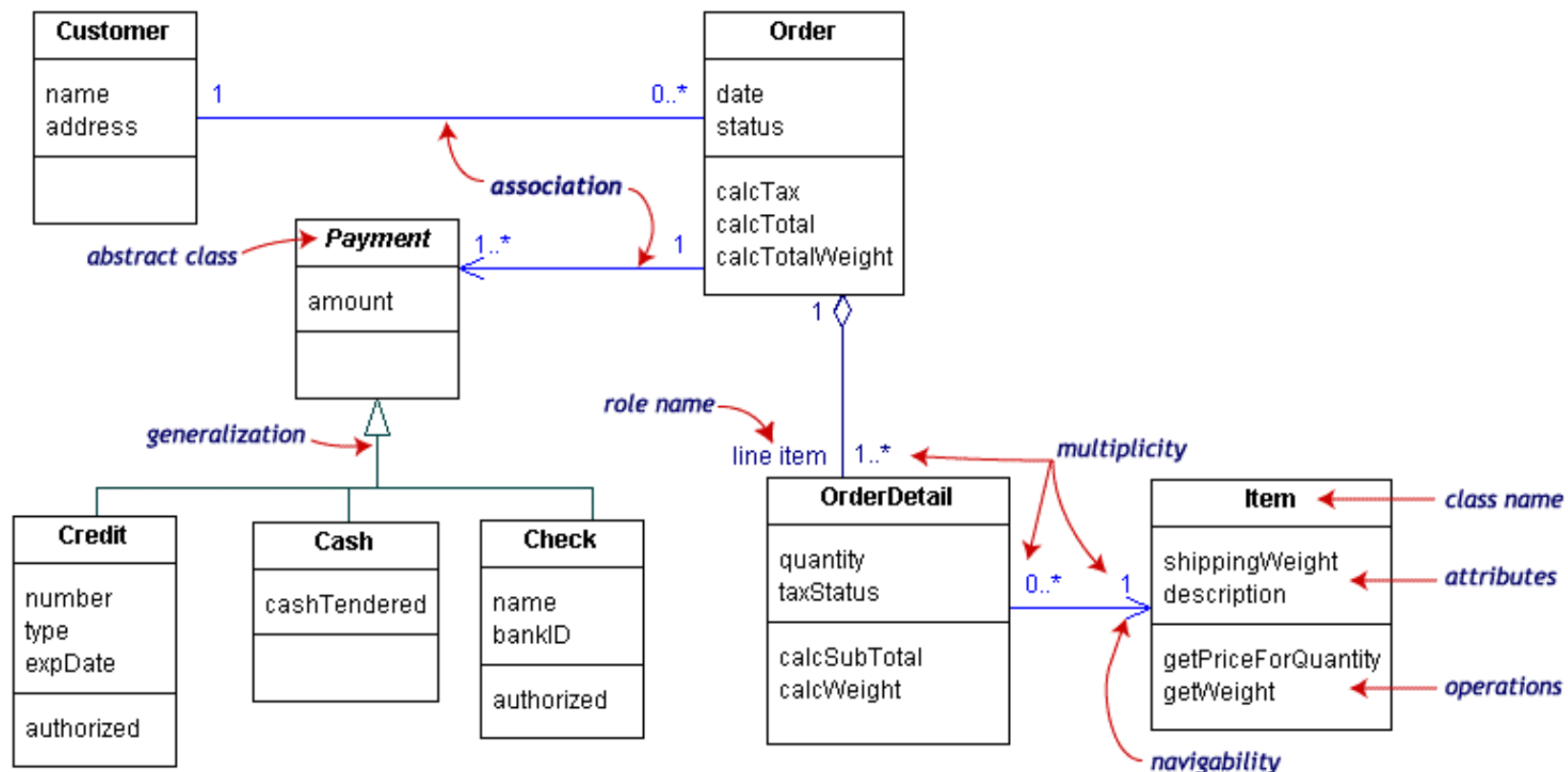
Dependencias

Tipos de dependencias:

- **Uso** «use»

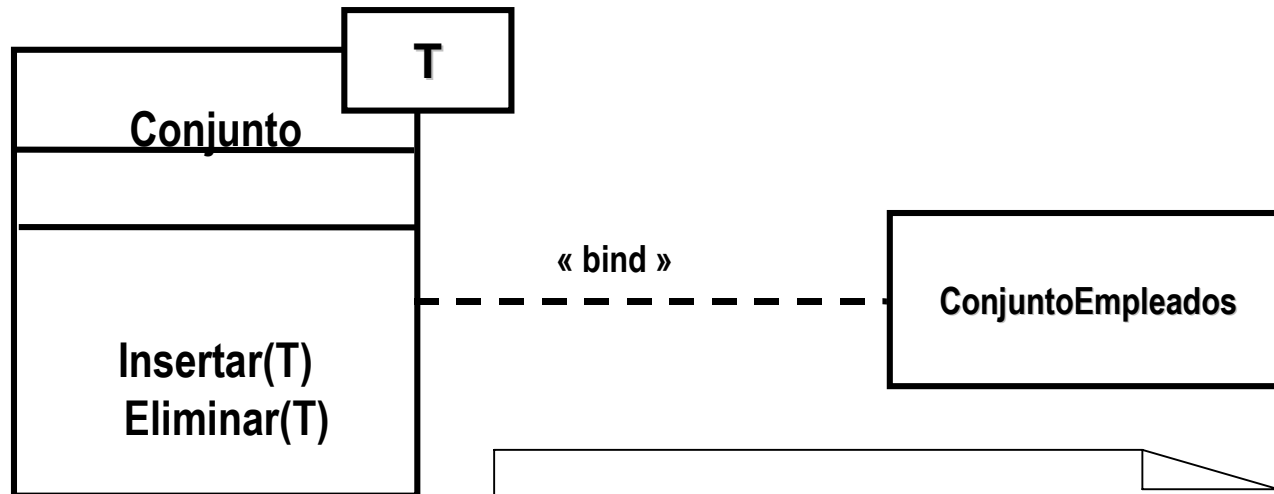
- **Acceso** «access»: es una dependencia de uso que indica que un paquete tiene permiso para acceder a los contenidos de otro paquete:
 - **Amigo** «friend»: dependencia de acceso que además permite que el paquete accedido pueda ver el contenido privado del que accede.
 - **Importación** «import»: dependencia de acceso en la que el paquete accedido puede añadir alias de sus nombres en el espacio de nombres del importador.

Resumen



Conceptos avanzados

Clases con parámetro (=plantillas): clases que representan un conjunto o colección y que contienen las operaciones para gestionarlos.

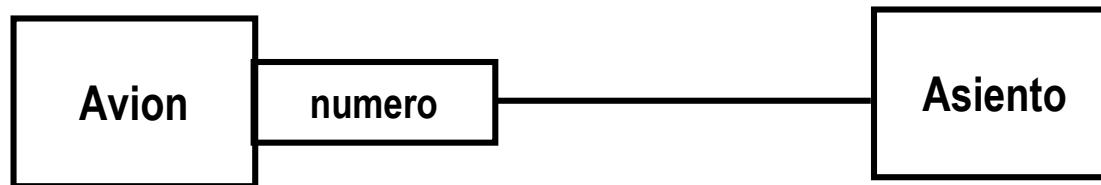


Nota: esto no es una subtipificación: no se puede añadir nada al elemento enlazado, pues es especificado totalmente por su plantilla.

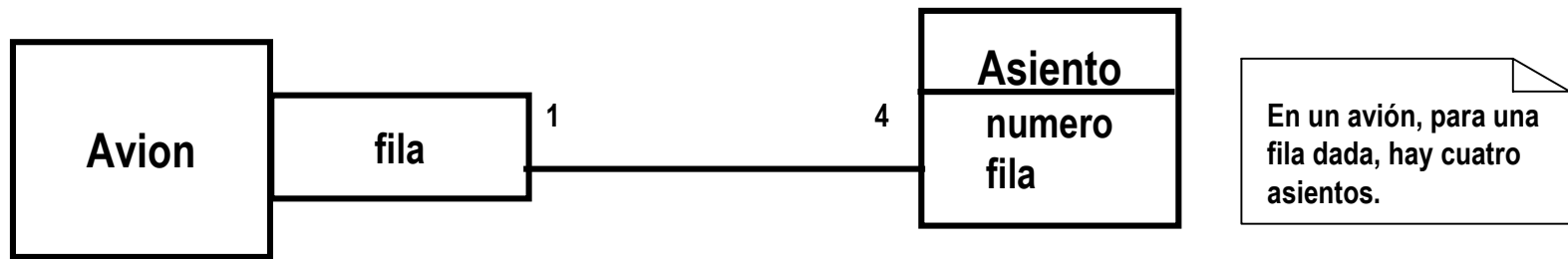
Conceptos avanzados

Asociaciones calificadas: son aquellas que, además de ser clase, tienen un atributo (=calificador) que selecciona un **único** objeto dentro del conjunto de objetos relacionados.

Ej. Vectores y tablas de valores.



Conceptualmente significa que no se pueden tener dos instancias de **Asiento** para el mismo número dentro de un **Avión**. Todos los accesos a **Asiento** se hacen a través del argumento número.



¿Qué hace que un modelo sea bueno?

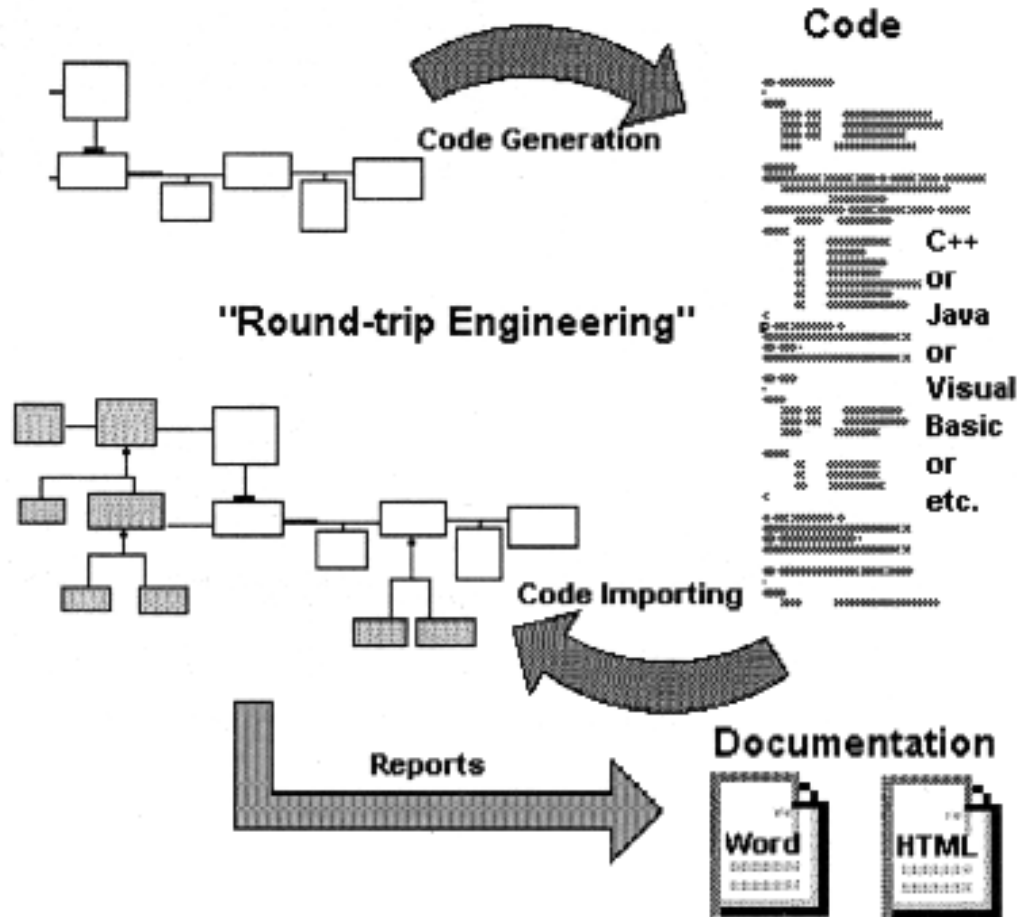
Dos objetivos:

- Construirlo tan rápido y barato como sea posible, satisfaciendo los requerimientos establecidos.
- Construirlo de forma que sea fácil de mantenerlo y adaptarlo a requerimientos futuros.

Para conseguir el primer requisito los comportamientos requeridos por los objetos deben ser generados de forma sencilla por los objetos de las clases elegidos.

Para conseguir el segundo requisito debemos conseguir extraer las características esenciales de los objetos. Las características deben ser suficientemente abstractas para que no dependan de la funcionalidad necesaria en la actualidad. Es decir, que podamos modelar con las clases comportamientos futuros.

Análisis y diseño iterativo



Análisis y diseño iterativo de un diagrama de clases.

Ejemplo

Vamos a modelar la escritura de un documento:

- Supón que estas escribiendo un documento en un procesador de texto, por ejemplo Microsoft Word. Tú puedes comenzar un nuevo documento, o abrir uno ya existente. Tu tecleas el texto a través de un teclado.
- Un documento consiste en un conjunto de paginas, cada página consiste en una cabecera, un cuerpo y/o un pie de página. Tanto en la cabecera, como en el pie de página se pueden añadir la fecha, la hora, el número de página, la localización del fichero...
- El cuerpo del documento consiste en frases. Cada frase se compone de palabras y signos de puntuación. Cada palabra consiste en letras, números y/o caracteres especiales. También se pueden insertar en el texto imágenes y tablas. Las tablas consisten en un conjunto de filas y columnas. Cada celda de la tabla puede contener texto o imágenes.
- Después de finalizar lo puedes guardar o imprimir.

Ejemplo

Esta es una explicación simple de la creación de un documento. La lista de nombres que se puede extraer de el modelo anterior son:

procesador de texto, Microsoft Word, teclado, documento, página, cabecera, cuerpo, pie de página, fecha, hora, numero de página, localización del fichero, cuerpo del documento, frase, palabra, signos de puntuación, caracteres, caracteres especiales, números, imágenes, tabla, fila, columna, celda

Los nombres en rojo son las clases y atributos candidatos en nuestro modelo.

Comencemos por el documento. Nuestra clase central en nuestro diagrama ha de ser un documento, ya que trabajamos con ellos. Los documentos constan de varias páginas, por tanto un atributo de los documentos es el *numeroDePaginas* de las que consta. Las páginas son también candidatas a ser clases.

Ejemplo

Document
numberOfPages
open() save() print() new()

La clase documento

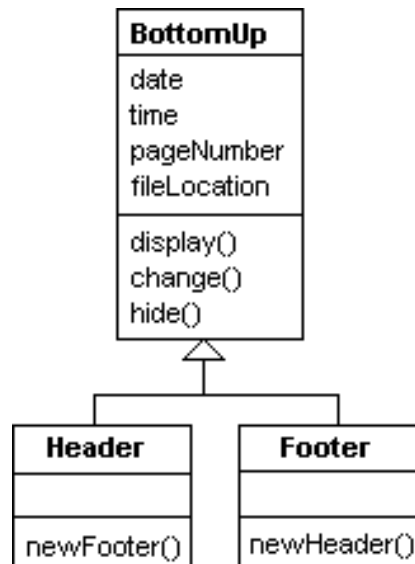
La clase **página** mantendrá en numero de la página como atributo, y las operaciones permitidas serán: *newPage()*, *hideHeader()*, *hideFooter()*... Las operaciones para la cabecera y el pie de página nos indican que estos también pueden ser clases.

Page
pageNumber
newPage() hideHeader() hideFooter() insertPicture() insertTable()

La clase página

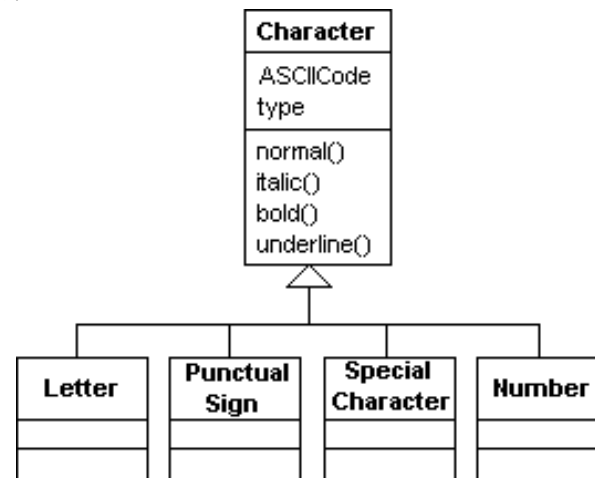
Ejemplo

Las clases cabecera y pie de página tienen atributos comunes: fecha, hora, número de página, y localización del fichero. Estos atributos son opcionales para cada cabecera y pie de página, y el usuario puede configurarlos. Esto nos indica que podemos introducir una clase común de la que podemos heredar los comportamientos y atributos. La clase padre la llamaremos **BottomUp** y contendrá los atributos y operaciones comunes.



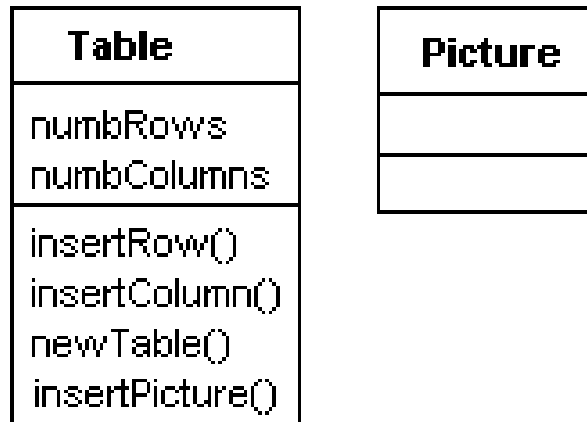
Ejemplo

Ahora nos centraremos en la creación de texto. El texto del documento está formado por frases. Las frases se forman de palabras, que se forman de caracteres. Si consideramos las palabras como arrays de caracteres y las frases como arrays de palabras, entonces podemos considerar una frase, también, como un array de caracteres. Por tanto para hacer un documento utilizaremos la clase carácter. Debemos tener en cuenta que hay distintos tipos de caracteres, por tanto este también es un buen punto para considerar la herencia.



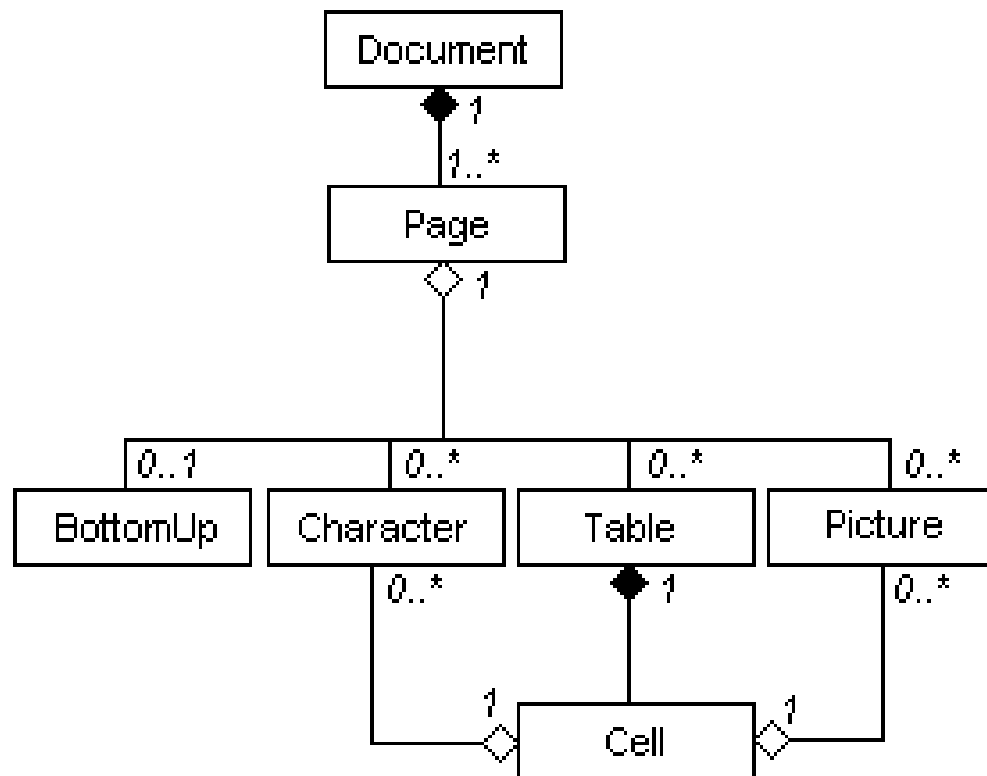
Ejemplo

Un documento puede contener tablas e imágenes. Por tanto debemos considerar esas clases. Las tablas tendrán un número de filas y columnas, y tendrán operaciones que permitan añadir filas y columnas.

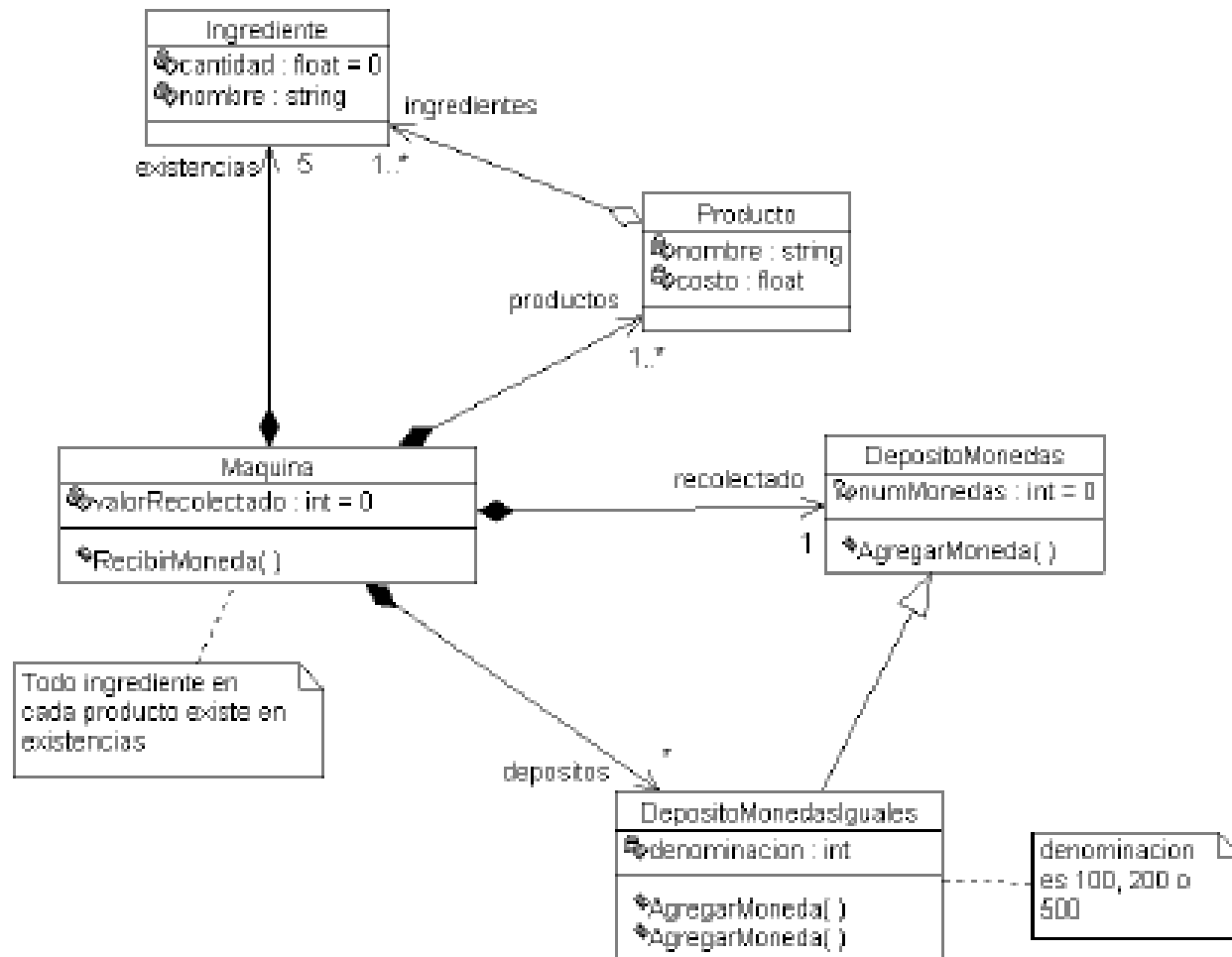


Ejemplo

Teniendo todo esto en la mente ahora ya podemos dibujar las asociaciones existentes entre nuestras clases.



Ejemplo 2



Ejemplo 3



Cuestión

Cada estudiante del colegio TS es supervisado por un profesor. Algunos profesores supervisan varios alumnos, y otros ninguno. ¿Cuál de los siguientes diagramas de clases modeliza esa relación?

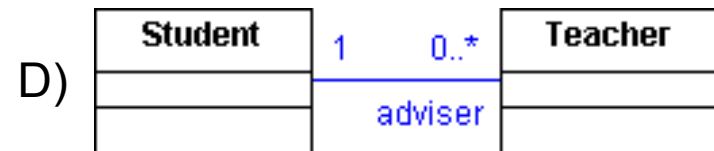
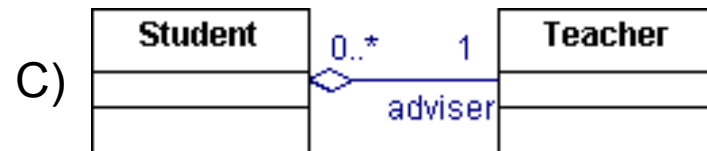
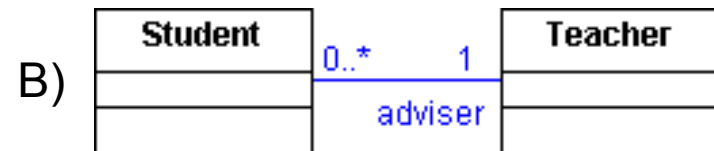
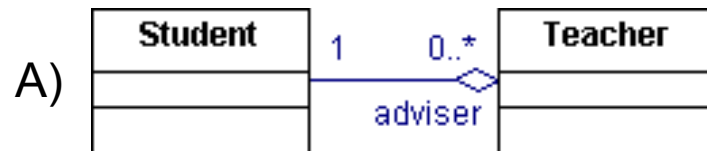


Diagrama de Paquetes (Vista estática)

Contenido

Diagrama de paquetes.

- Introducción
- Dependencias
- Resumen
- Ejemplos

Introducción

Sonido



Introducción

Muestra el sistema organizado “por partes” para manejar sólo la información necesaria en un momento dado.

- Se puede aplicar a cualquier vista del modelo (diagramas de clases o de casos de uso fundamentalmente).
- Consiste en fragmentar un sistema en subsistemas para facilitar la gestión y manipulación del mismo.
- La herramienta es el diagrama de paquetes, que muestra la descomposición en paquetes del sistema y las dependencias entre los mismos.

Introducción

Paquete: mecanismo de propósito general para organizar en grupos los elementos y/o diagramas del sistema.

- Contienen elementos como clases y sus relaciones, máquinas de estado, diagramas de casos de uso, interacciones y colaboraciones.
- Los paquetes pueden anidarse y organizarse jerárquicamente.
- La organización en paquetes puede seguir un criterio funcional, de una vista concreta o cualquier otro que elija el desarrollador.

Dependencias

Dependencia entre paquetes: resume las dependencias entre los contenidos de los paquetes implicados, es decir, se deriva de las dependencias entre los elementos individuales contenidos (dependencias de alto nivel).

- Representa una o varias dependencias entre elementos contenidos en los paquetes implicados.
- En general un paquete es opaco a otro, salvo que la dependencia que los enlaza sea de acceso o de importación. En estos casos la visibilidad de los elementos contenidos en el paquete debe ser la adecuada.

Dependencias

- En el caso de la **importación** se concede permiso en un único sentido.
- La diferencia con la dependencia **de acceso** es que en esta última no se añade el contenido del paquete destino al espacio de nombres del paquete origen.
- La **exportación** del contenido de un paquete se realiza a través de la visibilidad “pública”.
- Puede existir **generalización** entre paquetes, lo que nos conduce a la existencia de paquetes abstractos de manera similar a las clases abstractas.

Resumen

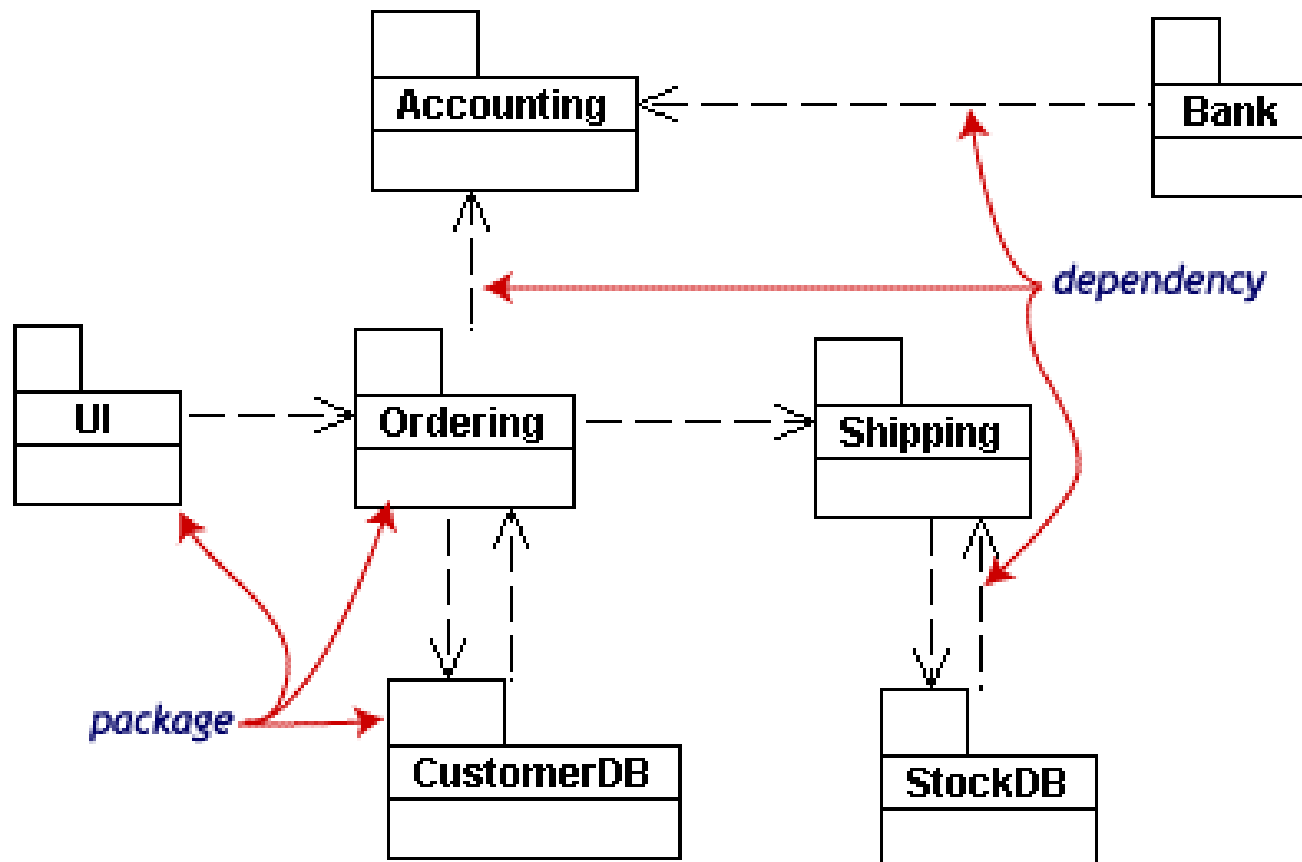
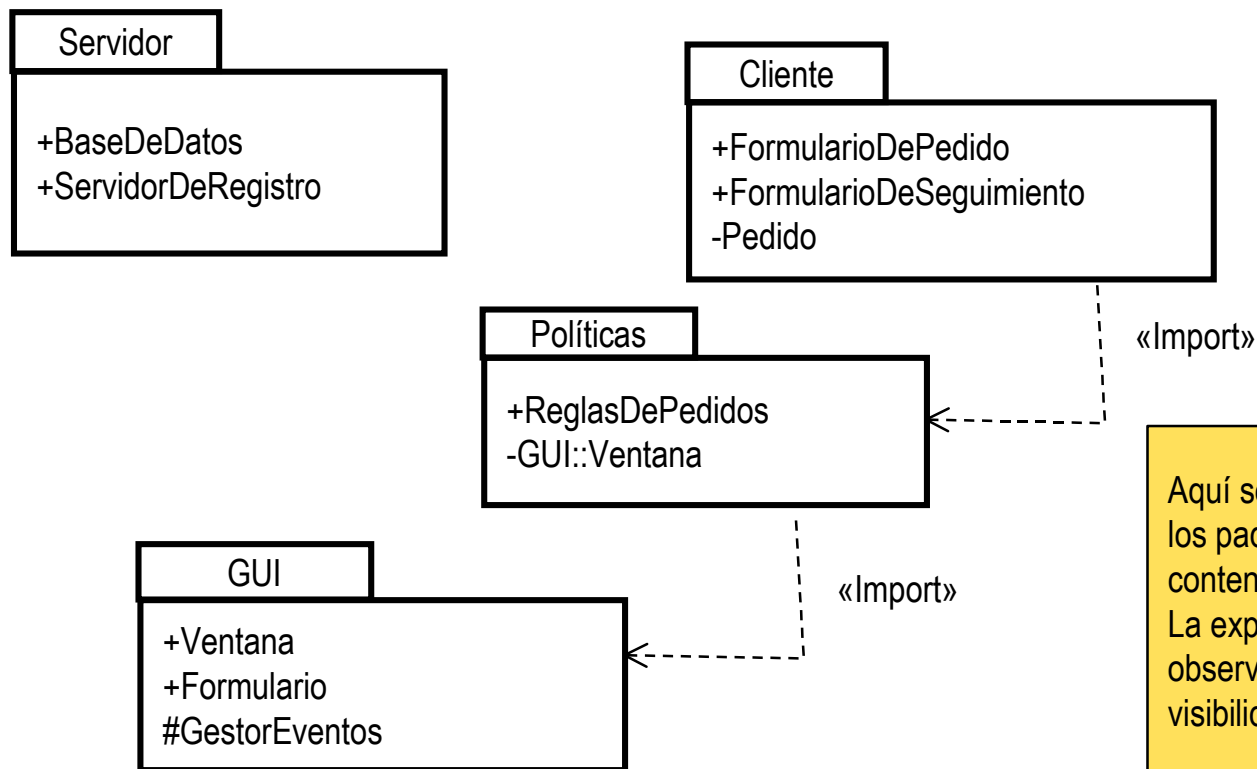


Diagrama de paquetes

Ejemplo



Aquí se muestran los paquetes conteniendo clases. La exportación se observa según es la visibilidad + (pública)

Ejemplo

En el ejemplo anterior:

- Todo lo que un paquete exporta es visto por otro paquete que lo importe explícitamente.
- Políticas importa del paquete GUI dos clases :
 - GUI::Ventana y GUI::Formulario pero GUI::GestorDeEventos no será visto desde Políticas por tener visibilidad protegida.
 - El paquete Servidor no ve nada de GUI porque no establece importación explícita.
- Las dependencias de importación no son transitivas:
 - Cliente no importa Ventana ni formulario, ni GUI aunque sí lo haga Políticas.

Ejemplo 2

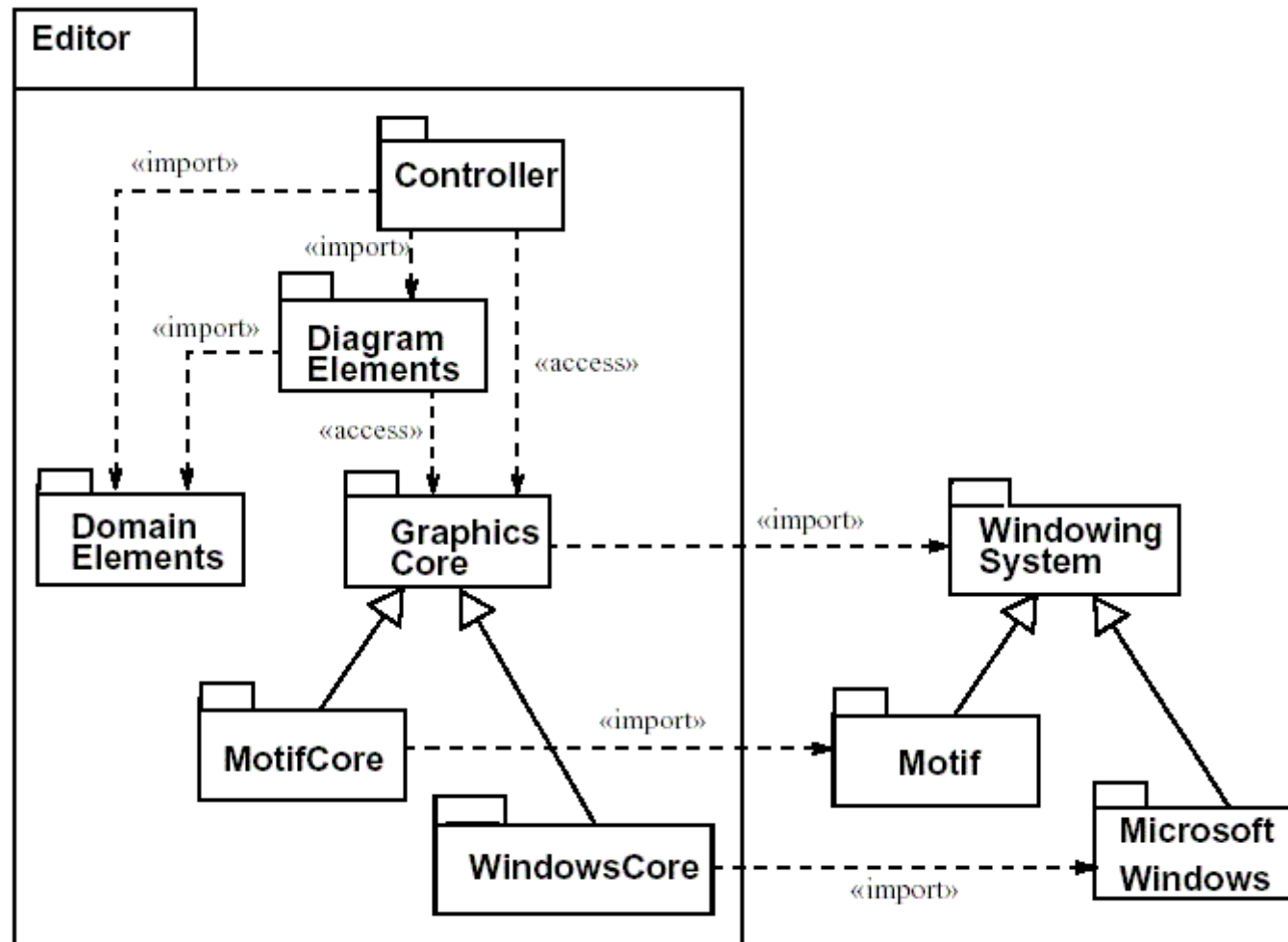


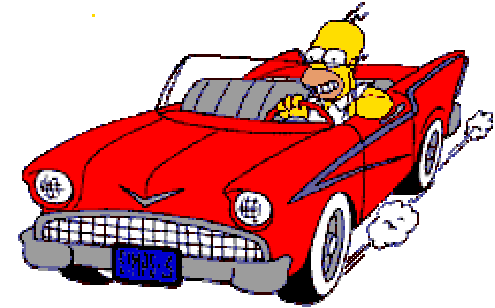
Diagrama de casos de uso (Vista funcional)

Contenido

Diagrama de casos de uso

- Introducción
- Notación
- Escenarios
- Relaciones
- Ejemplos
- Beneficios
- Construcción
- Dificultades

Introducción



Caso de uso el coche funciona perfectamente y no lo tengo que llevar al taller.

Otros casos:

- Coche sin gasolina
- Coche que no arranca

Introducción

Los diagramas de casos de uso representan una vista dinámica del sistema y muestran, por tanto, la evolución del sistema a lo largo del tiempo.

Son un buen punto de partida para analizar a grandes rasgos el comportamiento del sistema.

- Se representa la funcionalidad del sistema desde la percepción que tienen del él los usuarios (actores).
- Los diagramas de casos de uso se van refinando partiendo de un esquema de comportamiento general hasta llegar a escenarios y secuencias detalladas.
- Son colecciones de comportamientos del sistema. Cada escenario describe una secuencia de eventos. Cada secuencia es iniciada por un actor.

Notación

Diagramas de Casos de Uso

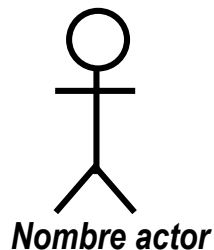
- Un **Caso de Uso** es una unidad coherente de funcionalidad, externamente visible, proporcionada por una unidad del sistema y expresada por secuencias de mensajes intercambiados por dicha unidad y uno o más actores.
- Un caso de uso sigue el paradigma de caja negra.
- Su perspectiva dinámica se puede detallar mediante otros diagramas, normalmente DIAGRAMAS DE INTERACCIÓN.
- Un caso de uso se puede factorizar en casos de uso más simples.
- Símbolo :



Notación

Diagramas de Casos de Uso

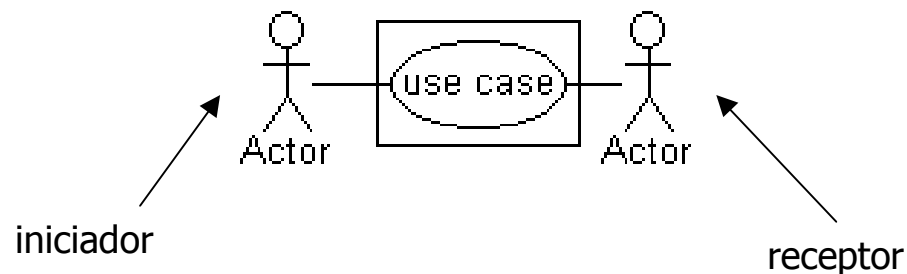
- Un **Actor** puede ser una persona, software o hardware que percibirán las interfaces del sistema. No tienen porque conocer la estructura interna del mismo. Se identifica con un rol, no con una persona física o sistema.
- Dos tipos de actores : los que usan el sistema (primarios) y los que lo administran (secundarios).
- Permiten obtener una vista funcional del sistema orientada al usuario.
- Los actores participan en uno o más casos de uso, intercambiando mensajes. Son los iniciadores de los casos de uso.
- Símbolo:



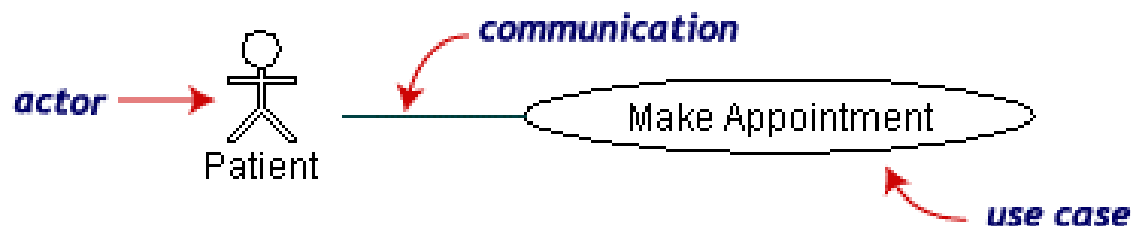
Notación

El diagrama de casos de uso se inicia por un actor, y otro actor (posiblemente el que lo ha iniciado pero no necesariamente) recibe algo del diagrama de caso de uso.

Ej:



El nombre del diagrama de caso de uso puede aparecer en la elipse o debajo de ella.



Escenarios

Un diagrama de caso de uso esta compuesto de distintos escenarios.

Un **escenario** es una secuencia de pasos. Cada escenario de cada caso de uso debe realizarse en una página independiente. En el escenario deben aparecer los siguientes detalles:

- **Actor que inicia** el caso de uso.
- **Precondición** para el caso de uso.
- **Pasos** que se producen en el escenario.
- **Postcondición** cuando el escenario se completa.
- **Actor que se beneficia** del caso de uso.

Ejemplo: para comprar unas mercancías, tendremos un caso de uso con varios escenarios asociados: uno en el que todo va bien, otro donde no hay suficientes mercancías, otro en el que nuestro crédito es rechazado...


Relación


- Una **RELACIÓN** (o **INTERACCIÓN**) muestra la conexión y su semántica entre un actor y un caso de uso, o entre dos o más casos de uso.

- Tipos de relaciones :

Asociación: **comunicación entre un caso de uso y un actor.**  « communicates »

Extensión: **generación de un nuevo caso de unos añadiendo pasos a otra ya existente.**  « extends »

Generalización: **relación de herencia entre un caso de uso más general y otro más específico o entre actores.** 

Inclusión: **inserción de un comportamiento adicional en un caso de uso que describe explícitamente esta inserción: incorpora el comportamiento de otros casos de uso como fragmentos de su propio comportamiento. En UML 1.3 y anteriores « uses ».**  « include »

Ejemplo

En este ejemplo modelaremos una máquina de servicios de productos. La principal función de la máquina es permitir al usuario comprar un producto de la máquina (chocolate, galletas, zumos...).



Este escenario se puede detallar más. Primero se inserta una moneda, se selecciona un producto y la máquina nos da el producto. Pero si la máquina no contiene el producto, o si la cantidad de dinero no es correcta la máquina nos devolverá el dinero o nos dará un mensaje de error indicandonos que la cantidad de dinero es incorrecta.

Este caso de uso es desde el punto de vista de un usuario, pero hay otros actores que interactúan con la máquina. El responsable de reponer los productos agotados tiene otra visión.

Ejemplo

El reponedor tiene la siguiente visión:

Al cabo de un tiempo estipulado por la empresa (precondición). Mira si hay algún producto agotado y si lo hay, desbloquea la máquina, abre la puerta, rellena los productos agotados, cierra la puerta y bloquea la máquina. La postcondición es que la máquina ha quedado completa.

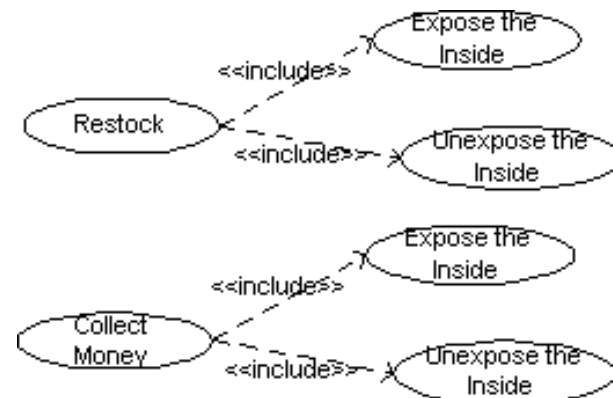


Pero esta visión es la misma visión que la que tiene el recolector de monedas. Excepto que él obtiene como postcondición que la máquina se ha quedado sin monedas y él se lleva lo recaudado.

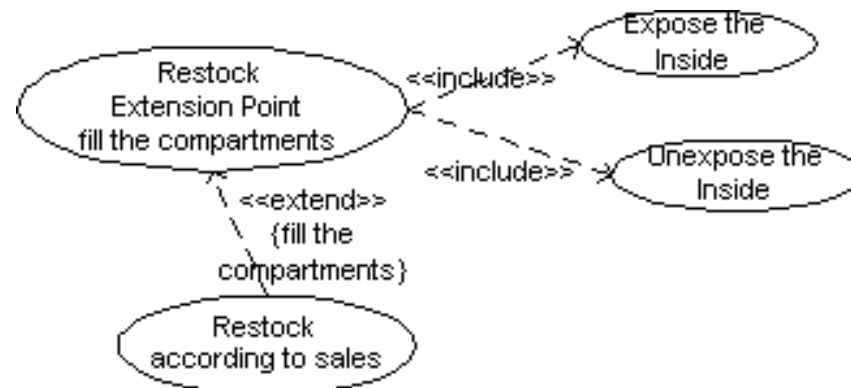


Ejemplo

Podemos ver ciertos pasos en detalle:

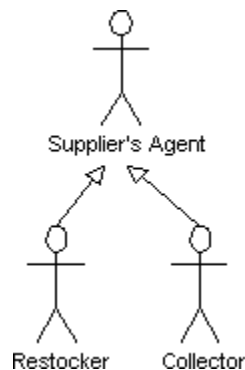


Puede que queramos que el comportamiento del reponedor se modele según la venta de los productos y por tanto no llegue a llenar los compartimentos. Esto representa una extensión del reponedor original.

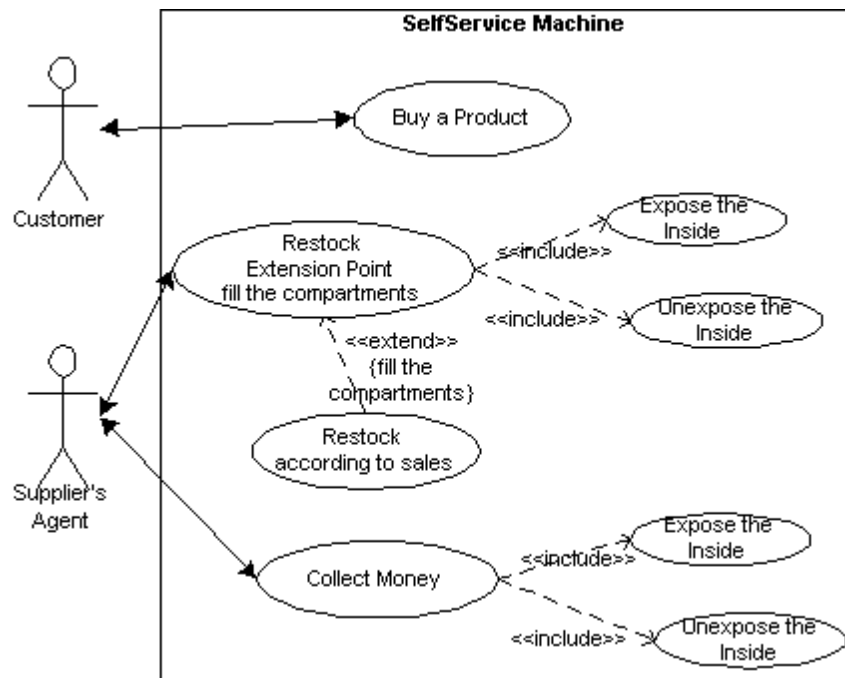


Ejemplo

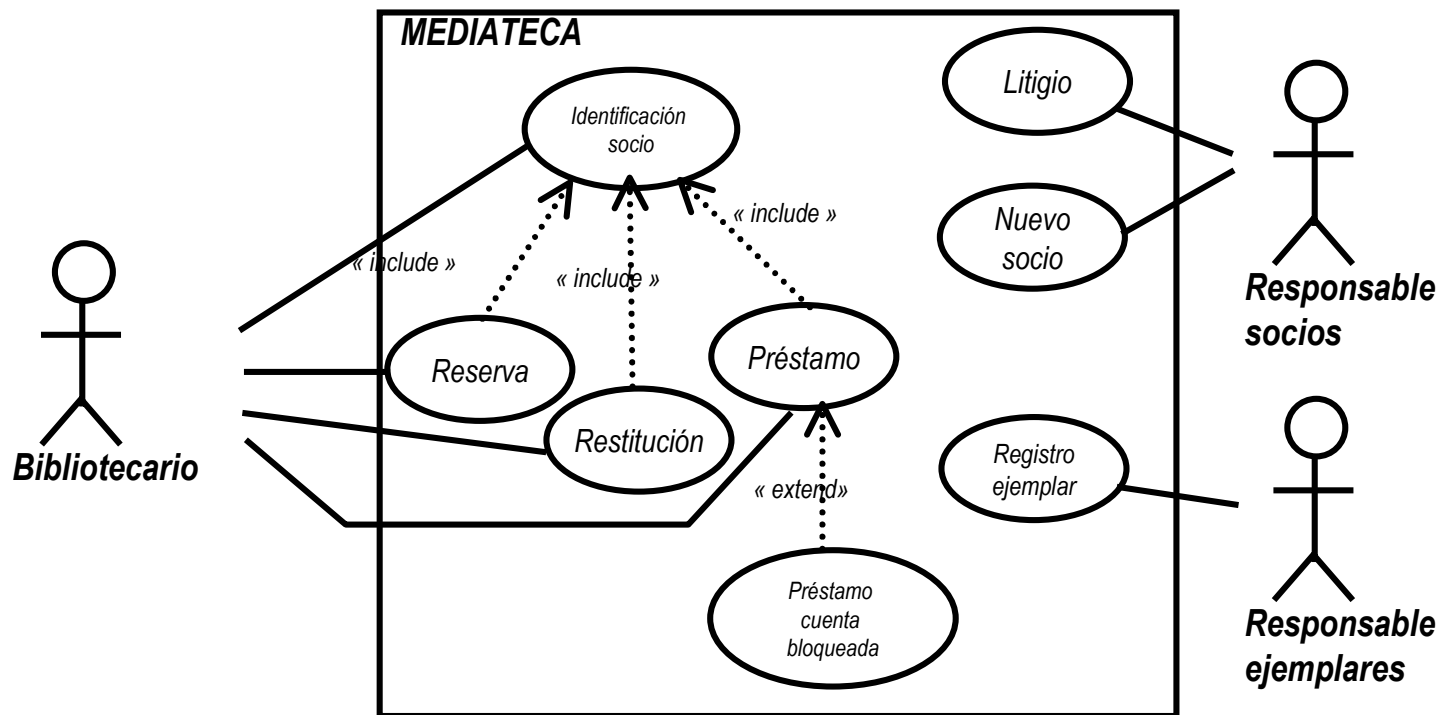
Podemos ir un poco más allá y considerar que los pasos entre el reponedor y el encargado del dinero son similares y por tanto podemos generalizar el usuario.



Esquema general de la máquina.

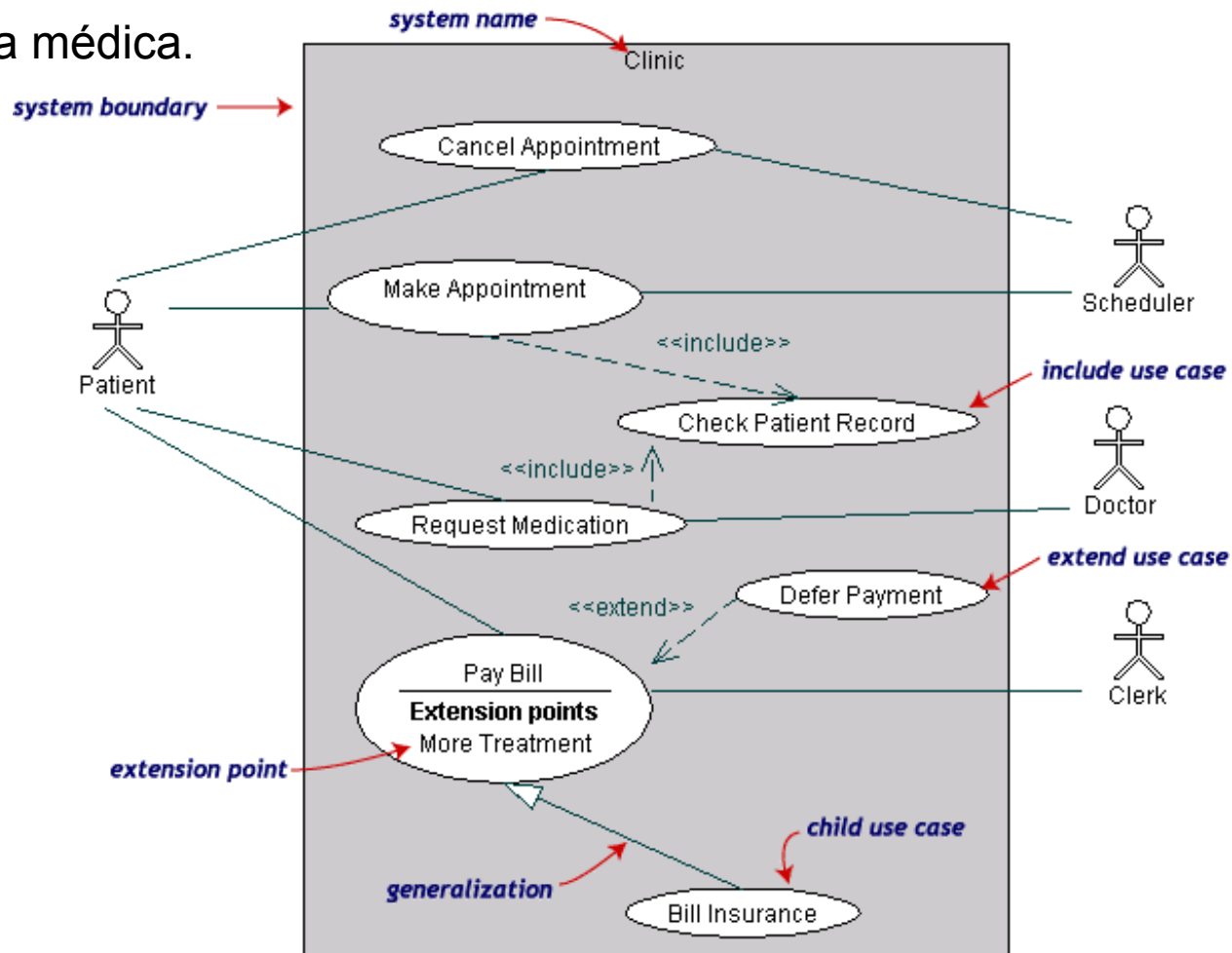


Ejemplo 2



Ejemplo 3

Clínica médica.



Beneficios

- La determinación y comprensión de los requerimientos del sistema es a menudo difícil, ya que los participantes en el proyecto tienen una gran cantidad de información.
- Los requerimientos son a menudo expresados de forma **desestructurada** y son incoherentes; frecuentemente pasa que unos contradicen a otros, se olvidan cosas, se mantienen errores y el análisis del sistema se diseña mal. Cuando los requerimientos cambian es muy difícil apreciar el impacto y el coste de las modificaciones.
- La organización de la aproximación se hace de acuerdo a las interacciones de una categoría simple de usuarios. Este particionamiento de los requisitos reduce considerablemente la complejidad de los requerimientos necesarios.

Beneficios

- La terminología utilizada en los diagramas de casos de uso es la misma que la empleada por los usuarios en la vida real. Por tanto, se facilita el análisis de los requisitos necesarios.
- Los casos de uso centran sus esfuerzos en los **requerimientos reales**. El caso de uso ayuda a los usuarios a estructurar y articular sus deseos. Les obliga a:
 - Definir la forma en la que quieren interactuar con el sistema
 - Especificar la información que esperan recoger
 - Describir el comportamiento esperado para obtener el resultado
- Si tu realizas lo que el cliente te pide, pero el resultado no corresponde a las necesidades reales del cliente, probablemente te lo reproche. Los requerimientos reales deben estar separados de la implementación en el diseño.

Construcción

- Los diagramas de casos de uso centran el desarrollo en los requerimientos del usuario, por tanto, ayudan a construir sistemas. Sin embargo, el caso de uso no define como construir el sistema de forma correcta, y tampoco especifica todo el desarrollo software.
- Un caso de uso debe ser simple, entendible y describir de forma clara y concisa el sistema.
- Durante la construcción de un caso de uso es necesario pensar los siguientes puntos:
 - ¿Cuales son las tareas del actor?
 - ¿Qué información debe ser creada, salvada, modificada, borrada o simplemente leída por el actor?
 - ¿Debe notificar el actor al sistema de modificaciones externas?
 - ¿Debe notificar el sistema al actor de modificaciones internas?
 - ¿Qué se debe hacer? y no ¿Cómo?

Dificultades

- La principal dificultad tiene que ver con el nivel de detalle, en vez de con la formulación del caso de uso. Los casos de uso son una abstracción de un conjunto de comportamientos que están ligados funcionalmente con ellos.
- Suelen cometerse errores de describir partes del sistema a un detalle muy bajo, mientras que otras se mantienen en un alto nivel de abstracción.
- La presencia de una gran cantidad de detalles identifica un escenario, más que un caso de uso.
- Un gran numero de casos de uso es un signo de una mala abstracción.
- En cualquier sistema, independientemente de su complejidad y tamaño, existen relativamente pocos casos de uso, pero muchos escenarios.
- Hay que resistir la tentación de describir detalles internos del comportamiento del sistema.

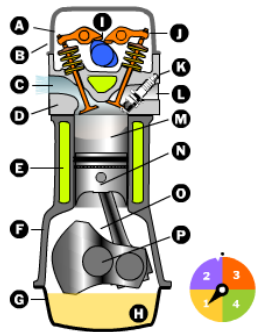
Diagrama de Estados (Vista dinámica)

Contenido

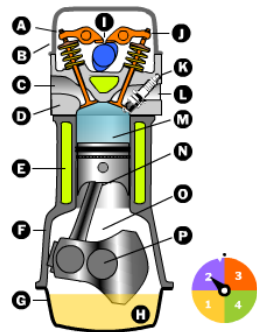
Diagrama de estados.

- Introducción
- Máquinas de estados
 - Representación
 - Estados
 - Acciones
 - Transiciones
 - Eventos
- Resumen
- Estados y transiciones especiales
- Utilidad
- Ejemplos

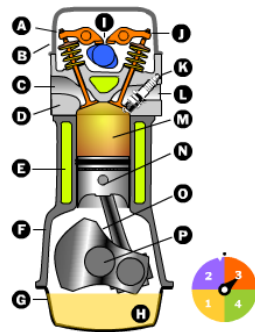
Introducción



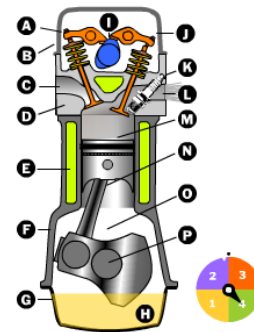
Aspiración



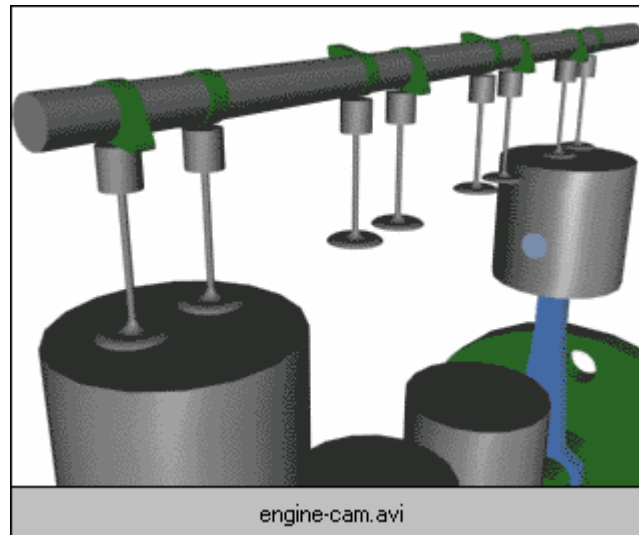
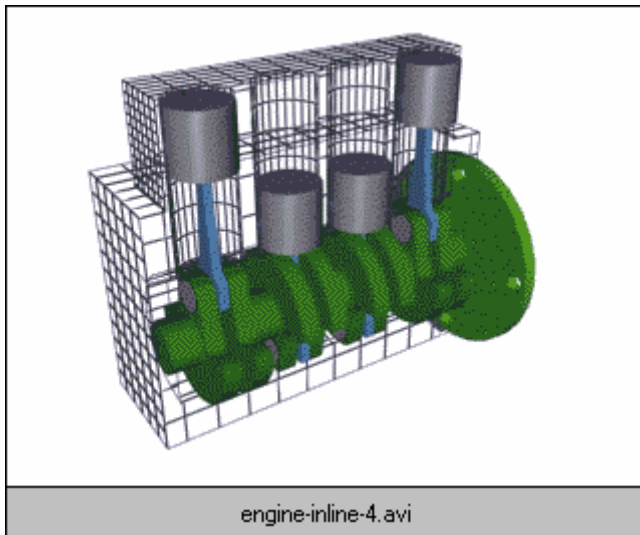
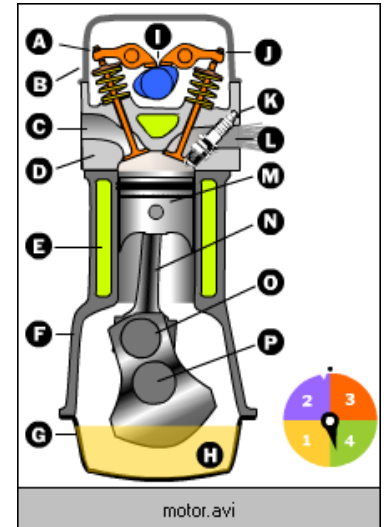
Compresión



Combustión



Escape



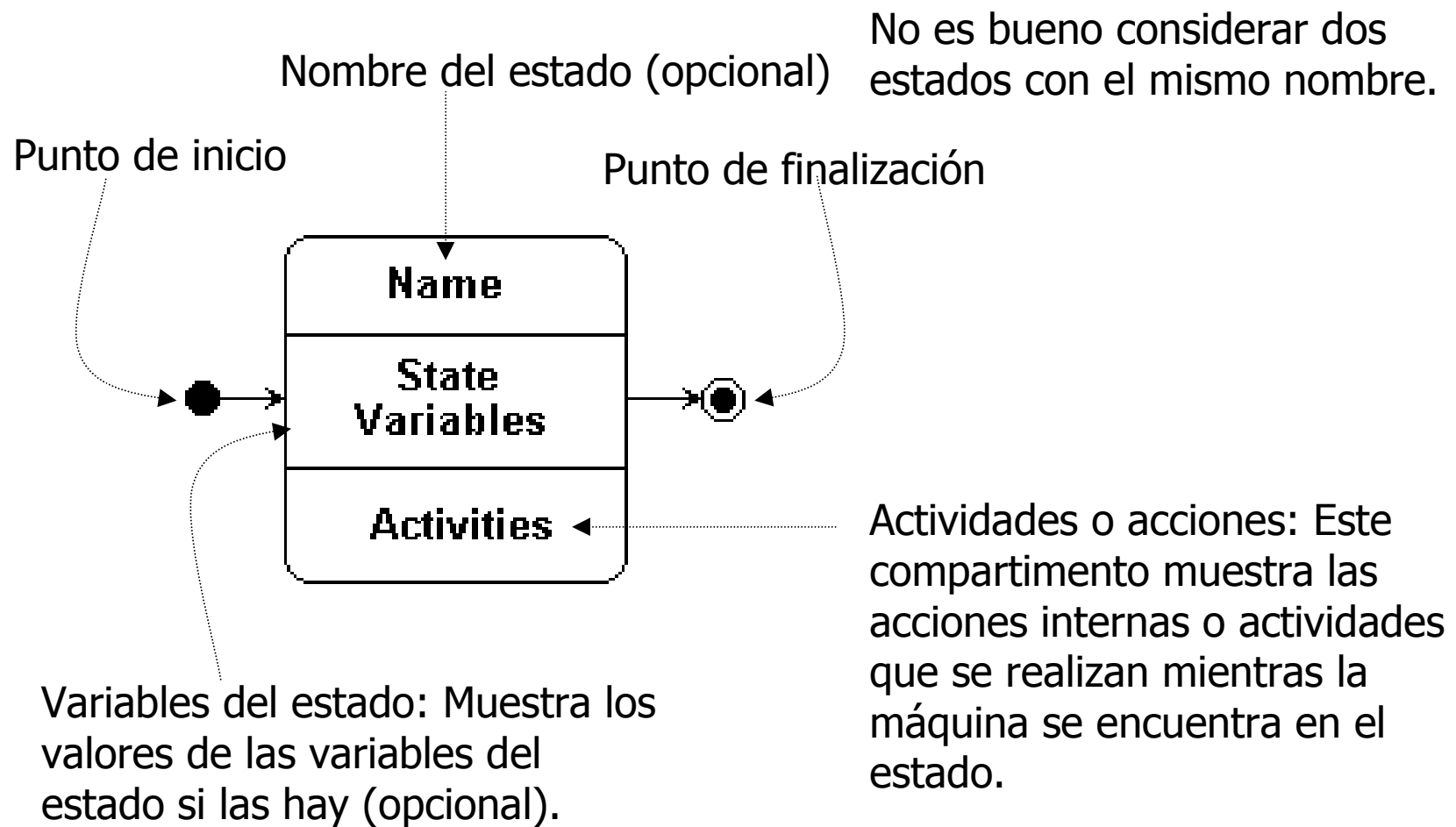
Introducción

Este diagrama se centra en el comportamiento. Sirve para modelar los cambios en el sistema.

- Describe el comportamiento dinámico de los objetos (es decir, de las clases), en un cierto plazo y su ciclo de vida.
- Como interactúan los estados internos de los objetos.
- Muestra el estado que tiene el objeto entre las distintas transiciones.
- También muestra el punto inicial y final de partida.
- También pueden describir el comportamiento dinámico de un caso de uso, una colaboración o de un método.
- **Máquina de estados:** gráfico que refleja los estados, eventos, transiciones y acciones resultantes de los cambios de estado.

Ej: Cuando presionamos un interruptor una luz se apaga o enciende.

Representación



Estados

Estado:

- Conjunto de valores de un objeto (para una clase dada) que producen la misma respuesta ante un evento.
 - Período de tiempo en espera de cierto/s evento/s.
 - Período de tiempo durante el cual el objeto realiza alguna actividad.
-
- Los objetos en distintos estado, pueden reaccionar de distinta forma ante un mismo evento.
 - Un estado es descrito por sus acciones asociadas.
 - Una ME conecta estados mediante transiciones.

Estados

Tipos de estados:

- Estado **simple**: un estado "normal", sin estructura.
- Estado **inicial**: un pseudo estado que indica cuál es el estado de inicio de una ME o de un estado compuesto.
- Estado **final**: un estado especial que indica fin de la actividad de la ME o de un estado.
- Estado **compuesto**: es un estado formado por varios subestados. (Los veremos más adelante.)

Acciones

Acción: asociada a una transición, puede ser un cómputo, enviar una señal a otro objeto, llamar a una operación, crear o destruir un objeto, una lista de acciones más simples, etc.

- Se realizan sin posible interrupción por otro evento, u otras acciones simultáneas.
- Son breves en comparación con la reacción al evento que las dispara.
- El sistema puede estar ejecutando varios hilos de acciones simultáneamente.
- Pueden emplear los parámetros del evento que las dispara y atributos del objeto poseedor.
- Las acciones suelen ir etiquetadas. Las etiquetas identifican las circunstancias en las que se realiza la acción.

Sintaxis:

etiqueta ‘(‘ lista de parámetros separados por comas ’)’ ‘[‘ condición de ejecución’]’ ‘/’ acción

Acciones

- Etiquetas de acciones:
 - **De entrada:** acción que se realiza en un estado destino de una transición nada mas entrar en él.
 - Sintaxis: **entry** / acción
 - **De salida:** acción que se realiza en un estado antes de una transición de salida del mismo (y por tanto, antes de la acción, la transición y la acción de entrada ,si la hay, del nuevo estado).
 - Sintaxis: **exit** / acción
 - **De ejecución:** acción que se realiza mientras la máquina se encuentre en ese estado, o hasta que la computación de la acción se termine (si se dispara una transición se termina).
 - Sintaxis: **do** / acción
 - **De inclusión:** acción que invoca a otra máquina.
 - Sintaxis: **include** / acción

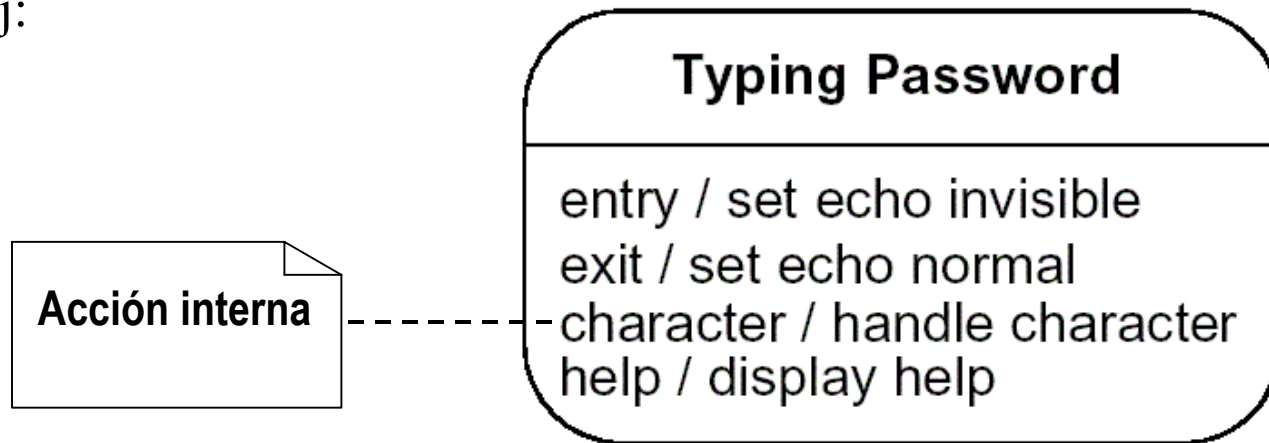
Acciones

- Etiquetas de acciones:

Es posible definir nuevas etiquetas de acciones.

Todas las etiquetas nuevas se consideran transiciones internas y se consideran transiciones propias del estado. La máquina no cambia de estado para realizar estas transiciones, ni siquiera se sale y se vuelve a entrar en el estado cuando se terminan de ejecutar.

Ej:



Acciones

- Tipos de acciones:
 - **Asignación:** asigna el valor de una variable.
 - Sintaxis: destino:=expresión
 - **Creación:** crea un nuevo objeto.
 - Sintaxis: newNombreClase(arg,arg)
 - **Destrucción:** destruye un objeto.
 - Sintaxis: objeto.destroy()
 - **Envío:** crea una instancia de una señal y la envía a un objeto u objetos destino.
 - Sintaxis: Nombres(arg,arg)

Acciones

- Tipos de acciones:
 - **Llamada:** llama una operación en un objeto destino y espera a que termine su ejecución. Puede devolver un valor.
 - Sintaxis: nombreop(arg,arg)
 - **No interpretada:** Acción específica del lenguaje, tal como un bucle, una condición...
 - Sintaxis: depende del lenguaje.
 - **Retorno:** Especifica los valores a devolver al llamador.
 - Sintaxis: return valor
 - **Terminación:** Autodestrucción del objeto.
 - Sintaxis: Terminate

Transiciones

Transición: La transición corresponde con el paso de un estado a otro de la máquina de estados.

- Se producen desde un estado de un objeto como respuesta ante un evento que la activa (evento disparador), si la condición se satisface (condición de ejecución), y que motiva un cambio hacia otro estado (o el mismo) y la realización de las acciones previstas.
- Los eventos pueden por tanto contener variables que son accesibles desde las acciones y estados específicos.

Sintaxis:

evento'[' condición de ejecución ']' '/' acción



Transiciones

- Algunas veces, un evento causa una transición sin tener asociada una acción.
- Otras veces, sucede una acción ya que el estado completa una actividad (en vez de generarse por un evento). Este tipo de transiciones se llama transiciones disparadoras.
- Si se pueden realizar varias transiciones con el mismo evento y no existe ningún tipo de prioridad o guarda que las distinga se elige una de ellas al azar.
- La condición de ejecución puede contener parámetros relacionados con el estado que inicia la transición. Esta condición puede involucrar a las variables de otros estados que se están ejecutando a la vez. (en el Estado1 se da...)
- La acción se ejecuta de forma completa antes de entrar en el nuevo estado. Puede ser de cualquiera de los tipos vistos anteriormente.

Ej: right-mouse-down(location)[location in window]/object := pick-object(location).

Transiciones

- Tipos:
 - **Transición de finalización:** asociada a un estado de un objeto, carece de evento disparador y es accionada por la terminación de la actividad del estado del que sale. Puede tener condición de guarda que se evalúa en el momento en que TERMINA la actividad en ese estado.
 - **Transición interna:** una respuesta a un evento que causa la ejecución de una acción pero no un cambio de estado ni ejecución de acciones de entrada ni de salida. Ej: desplegar una ayuda (es una especie de “interrupción”).
 - **Transición externa:** respuesta a un evento que causa un cambio de estado del objeto (puede ser hacia si mismo), y que conlleva una acción asociada (a veces, de entrada y/o salida también).

Eventos

Evento: es la acción que provoca los cambios en un objeto.

- Tipos de eventos:
 - Recepción de llamadas de un objeto a otro.
 - Recepción de señales de un objeto a otro.
 - Cambio de ciertos valores.
 - Paso de cierto intervalo de tiempo.

Sintaxis:

nombre del evento'(' lista de parámetros separados por comas ')'

Eventos

Tipos de eventos:

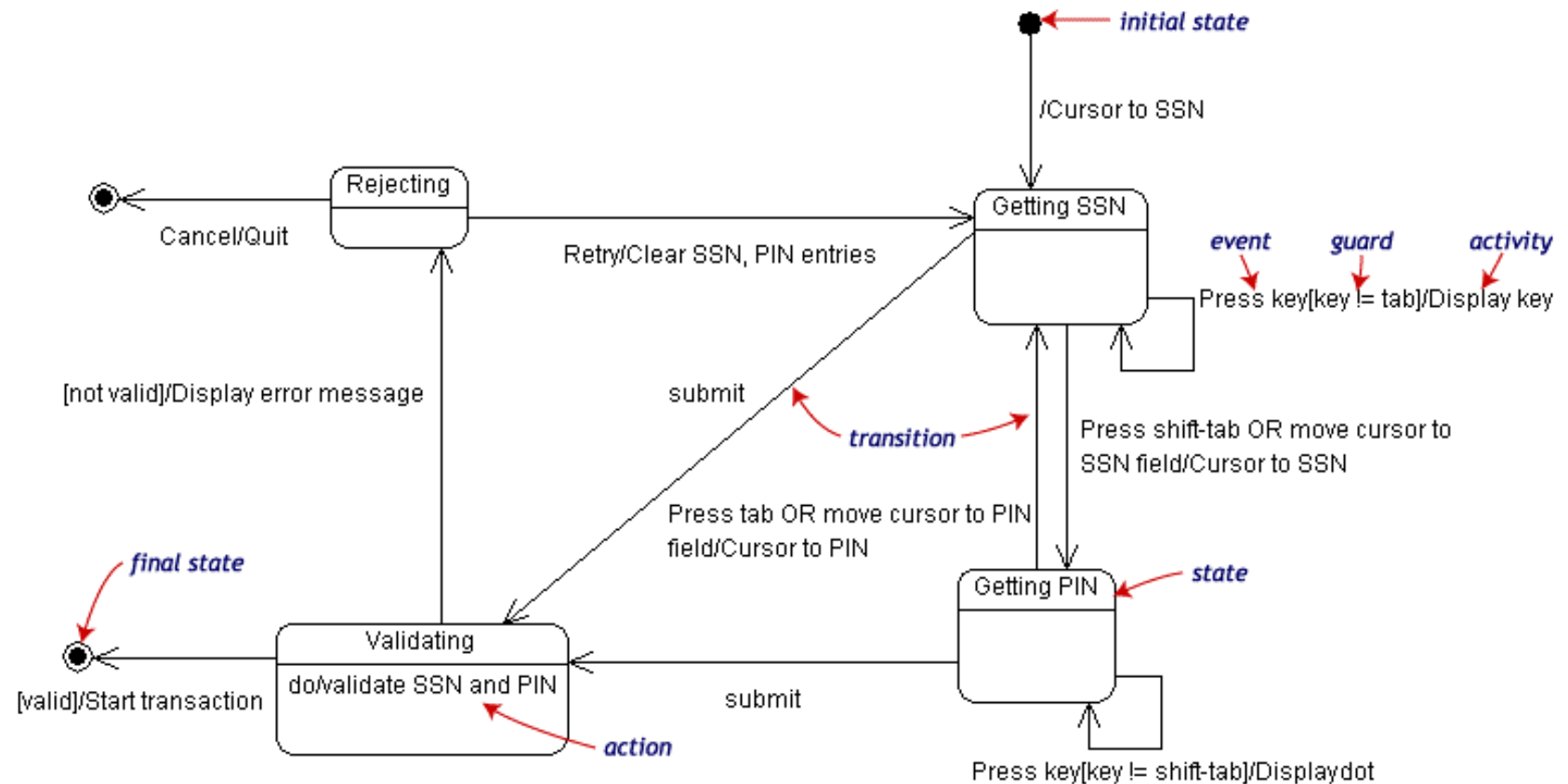
- **Evento de señal:** Recepción de una petición asíncrona explícita y con nombre entre objetos
 - El objeto remitente no espera respuesta del destinatario (pueden ser el mismo objeto).
 - Ejemplo: pulsar botón de ratón, teclear un carácter
- **Evento de llamada:** Recepción de una petición explícita síncrona entre objetos que esperan por una respuesta.
 - Normalmente es la solicitud de ejecución de una operación.
 - Una vez procesado el evento de llamada, el control se retorna al objeto llamador. Pero a diferencia de una llamada ordinaria, el receptor puede continuar su propia ejecución en paralelo al llamador.

Eventos

Tipos de eventos:

- **Evento de cambio:** un cambio en el valor de una expresión booleana sobre atributos del objeto.
 - Se evalúa constantemente (a diferencia de las condiciones de guarda que se evalúan asociadas a una transición).
- **Evento de Tiempo:** la conclusión de un tiempo absoluto (hora) o el transcurso de un tiempo relativo (time-out).
 - También se evalúa constantemente para cada objeto que tenga descritos estados que lo tengan asociado.
- **Evento diferible:** evento cuya aparición en el estado queda pospuesta, si no disparan una transición hasta que lo hagan o bien hasta que el sistema haga una transición a un estado en el que no se puedan retrasar, momento en el que serán tratados.
 - Se implementan mediante una cola interna de eventos.

Resumen



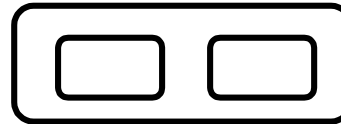
Estados especiales


Composición de estados

- La idea es estructurar los estados de una ME, aprovechando los conceptos de herencia: se heredan acciones, transiciones, etc., así como permitir la modelización de la concurrencia de estados.
- También se puede optar por definir submáquinas de estados, con propósitos de reutilización, haciendo referencia a ella desde un estado en otra máquina, lo cual activaría el estado inicial de la submáquina referenciada
(si se quiere entrar en la submáquina a otro estado distinto del inicial, se coloca un estado externo en el estado que referencia a la submáquina)

Estados especiales

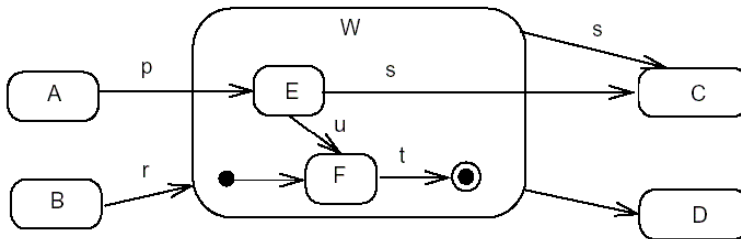
- **Estado compuesto:** estado formado por varios subestados secuenciales o concurrentes.
 - **Secuenciales:** Son estados que suceden unos detrás de otros. En definitiva, en secuencia.
- **Concurrentes:** Son estados que suceden a la vez. Esto se representa con una línea a puntos entre los estados adecuados.



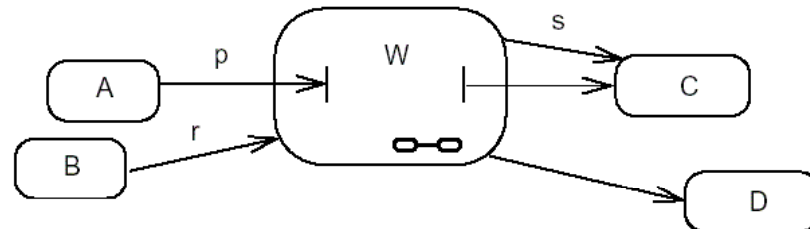
UML provee de un símbolo que muestra que un estado compuesto recuerda su subestado activo cuando se están realizando transiciones externas a dicho estado. Ese símbolo es:  conectado con una flecha al subestado correspondiente

Estados especiales

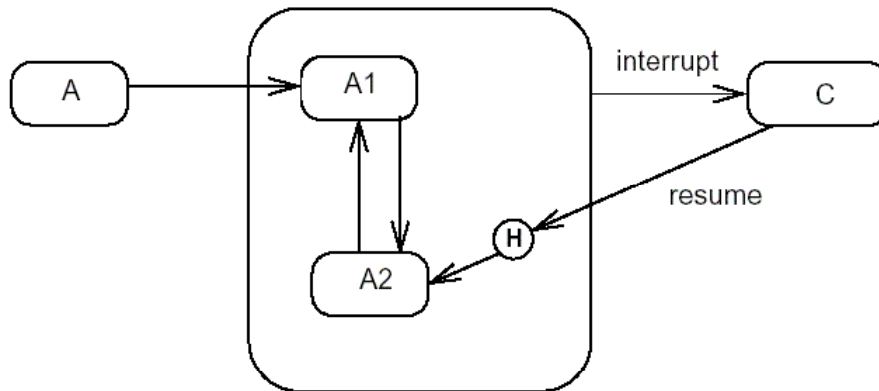
Ejemplos:



Estados compuestos



Estados contraídos



Indicador de situación

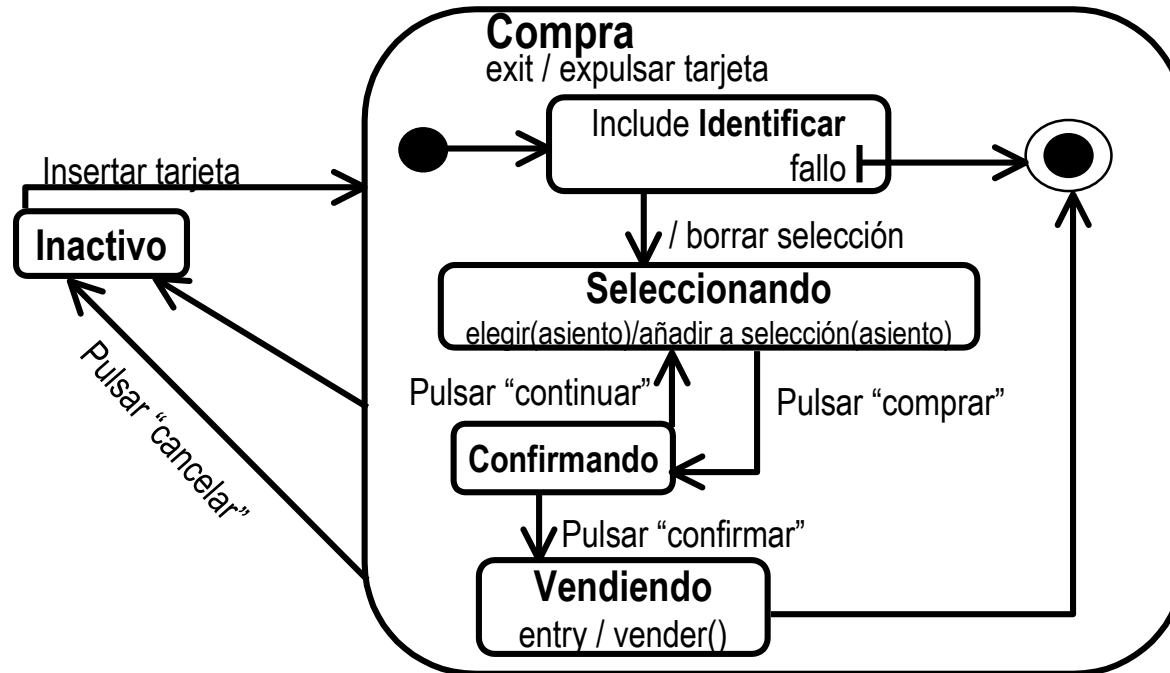
Estados especiales

- **Estado de referencia a submáquina:** un estado que referencia a una submáquina, que está implícitamente insertada en el lugar del estado de referencia a submáquina.

Sintaxis:

Include E

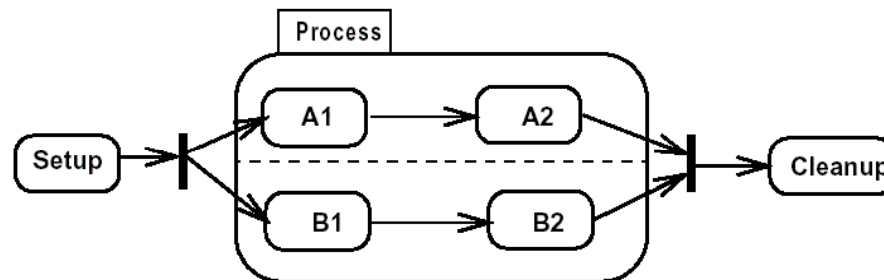
Ej:



Transiciones especiales

- Tipos:
 - **Transición concurrentes:** son transiciones que involucran a varios estados a la vez, tanto de partida, como de llegada.
 - **Paralelismo:** son transiciones que provocan que varios estados se ejecuten en paralelos.
 - **Sincronización:** son transiciones que provienen de varios estados y hacen que estos terminen la ejecución y se sincronicen.
 - **Mixta:** provoca la sincronización de los estados de los que recibe la transición y el paralelismo entre los que ejecutan.

Ej:

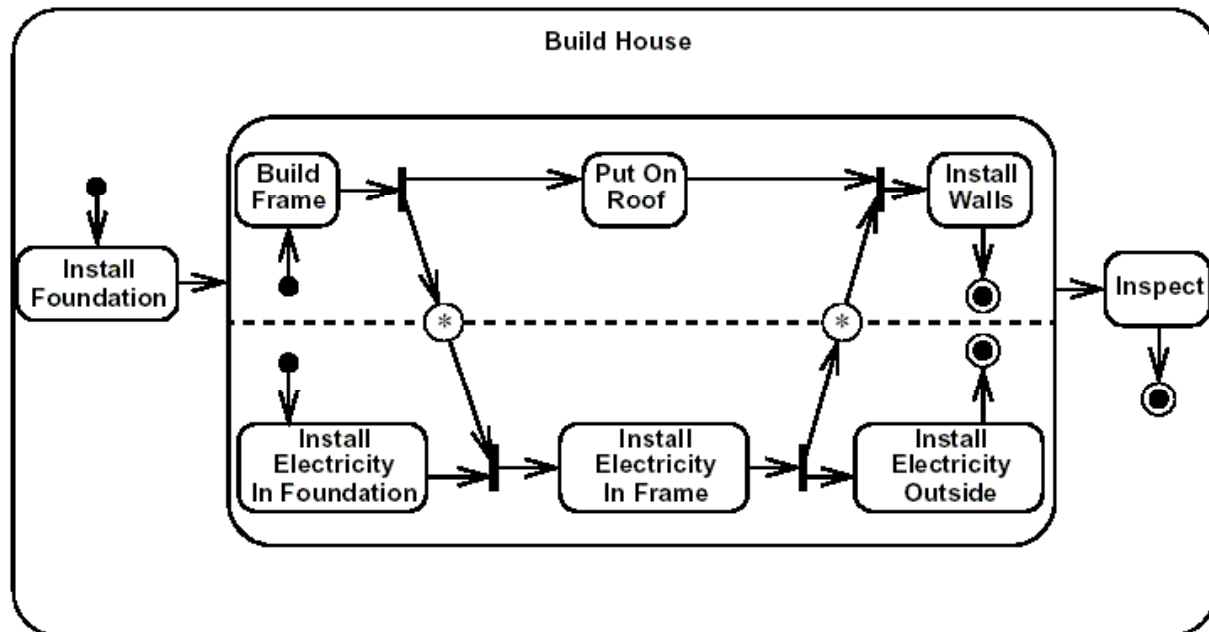


Estados especiales

- **Estados de sincronización:** Son estados que sincronizan las regiones de concurrentes de un estado.

Sintaxis: Se representan como un círculo con un numero dentro, que significa el limite de tiempo que se deja para la sincronización, o un * que significa que se espera hasta que se sincronicen.

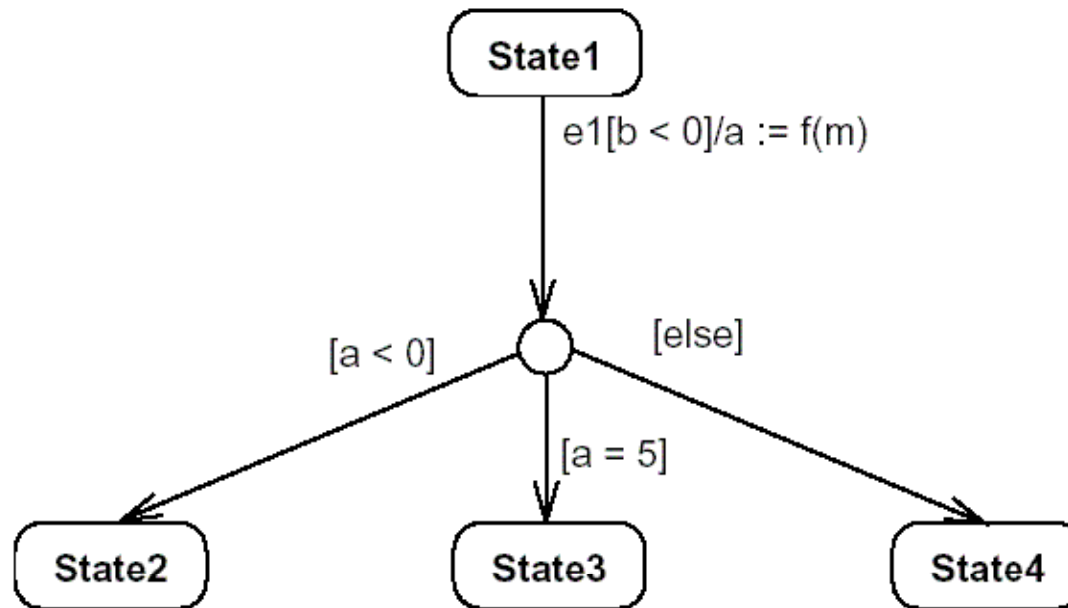
Ej:



Transiciones especiales

- Transiciones **factorizadas**. Se trata de compartir segmentos de una transición para simplificar el gráfico correspondiente.
 - **Puntos de conjunción del computo.** Sólo se realiza la transición si se cumplen ambos eventos. (También puede ser representado con un rombo.)

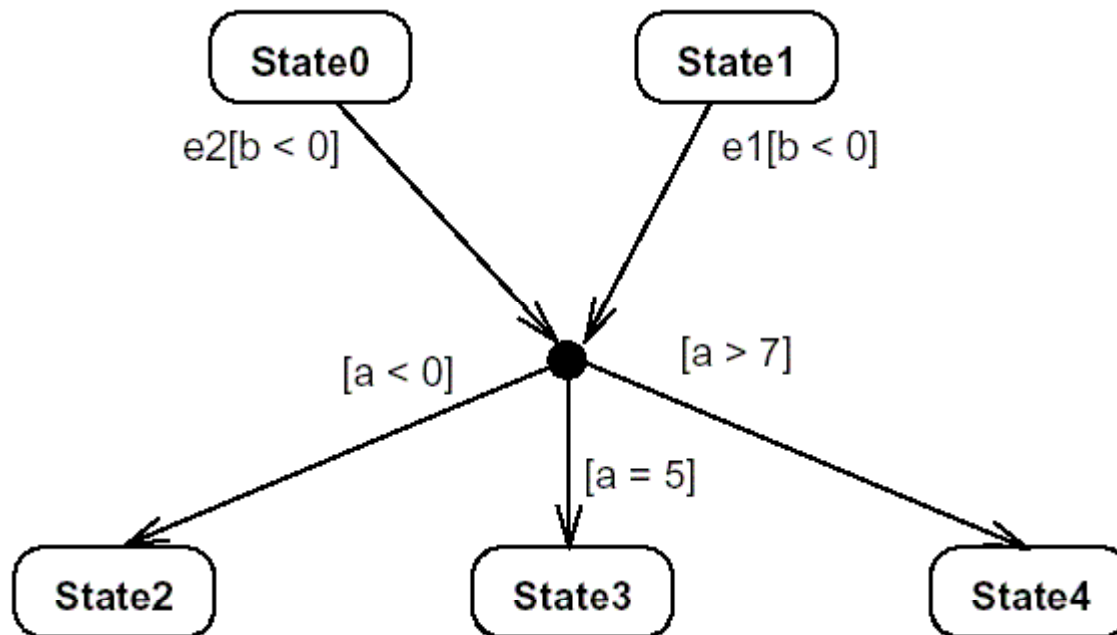
Ej:



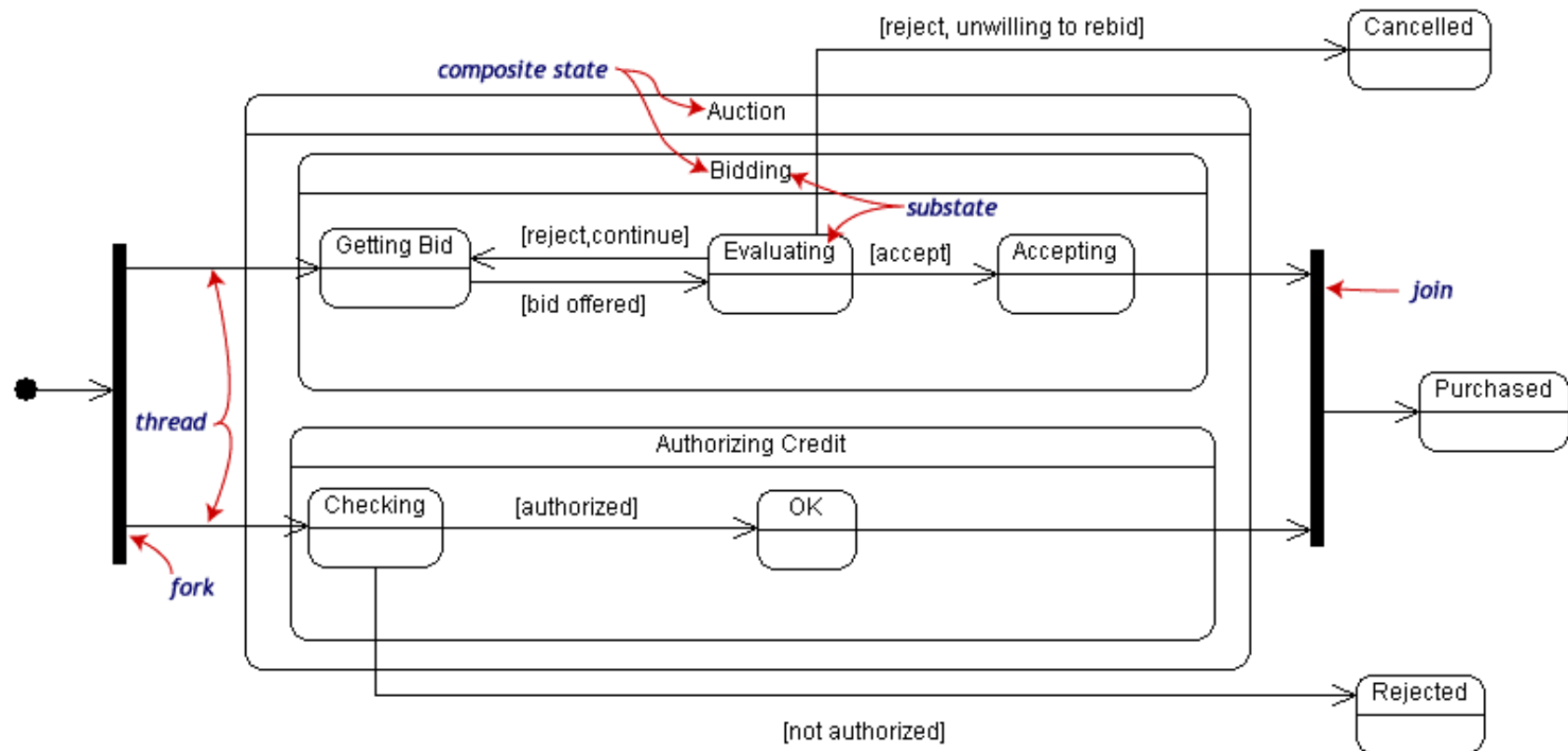
Transiciones especiales

- Transiciones **factorizadas**.
 - **Puntos de elección dinámicos.** Se ejecuta la acción de salida del estado y luego se mira si se puede realizar la siguiente.

Ej:



Resumen



Utilidad

Los diagramas de estados ayudan a los analistas, diseñadores y desarrolladores a comprender el comportamiento de los objetos del sistema.

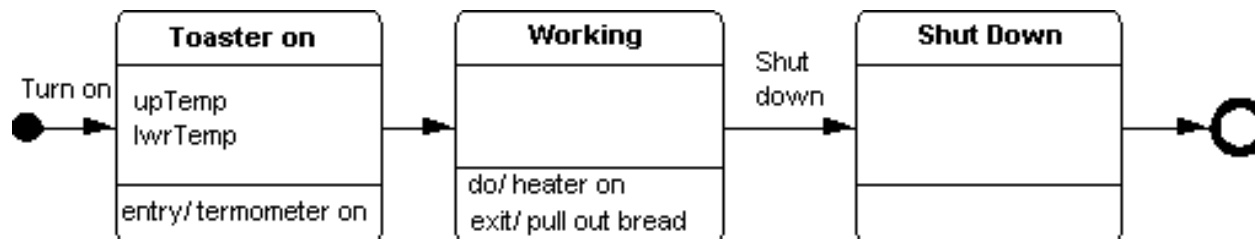
Los desarrolladores tienen que conocer cual es el comportamiento de los objetos del sistema, ya que son ellos los implementadores de sus comportamientos. No es suficiente con implementar un objeto. Los desarrolladores tienen que darle funcionalidad.

Ejemplo

Vamos a diseñar un tostador. Sólo nos vamos a concentrar en el diagrama de estados.

- ¿Cómo se comporta un tostador?
 - Se enciende.
 - Se pone a trabajar durante varios minutos.
 - Se apaga.

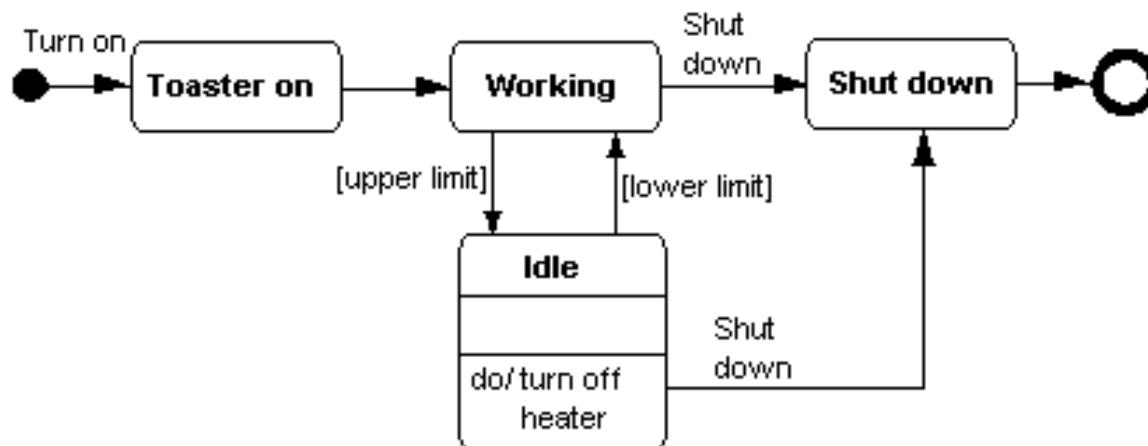
El diseño inicial que se nos puede ocurrir es el siguiente:



Ejemplo

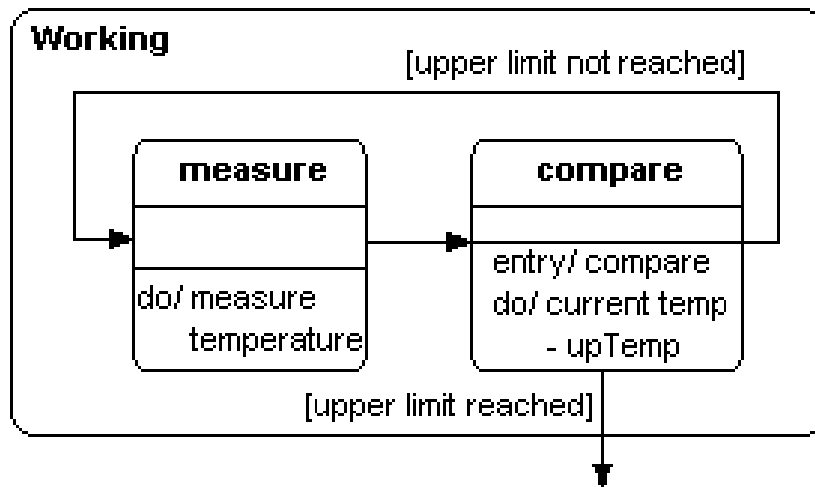
Pero este no es el diagrama de estado definitivo. Ya que, para que el tostador no se queme, lleva unos valores de temperatura máxima y mínima, entre los que funciona. El termostato hace que cuando sobrepase el valor máximo el tostador se pare, y que cuando se alcance el mínimo se encienda.

Por tanto, el modelo que surge es el siguiente:

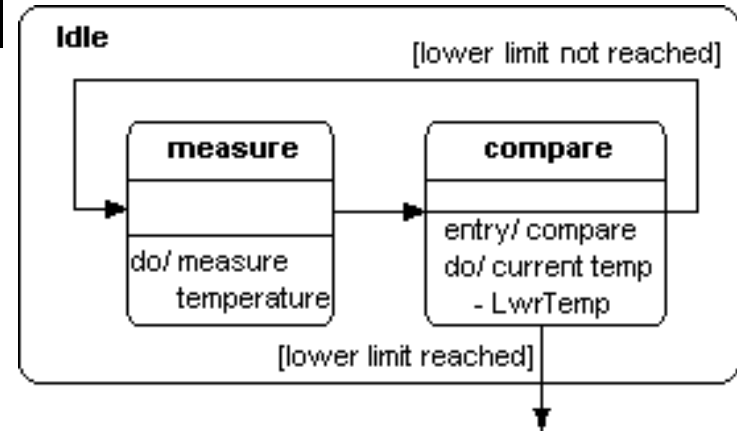


Ejemplo

Las transiciones del termostato no se muestran en detalle. Para verlas mejor debemos añadir:

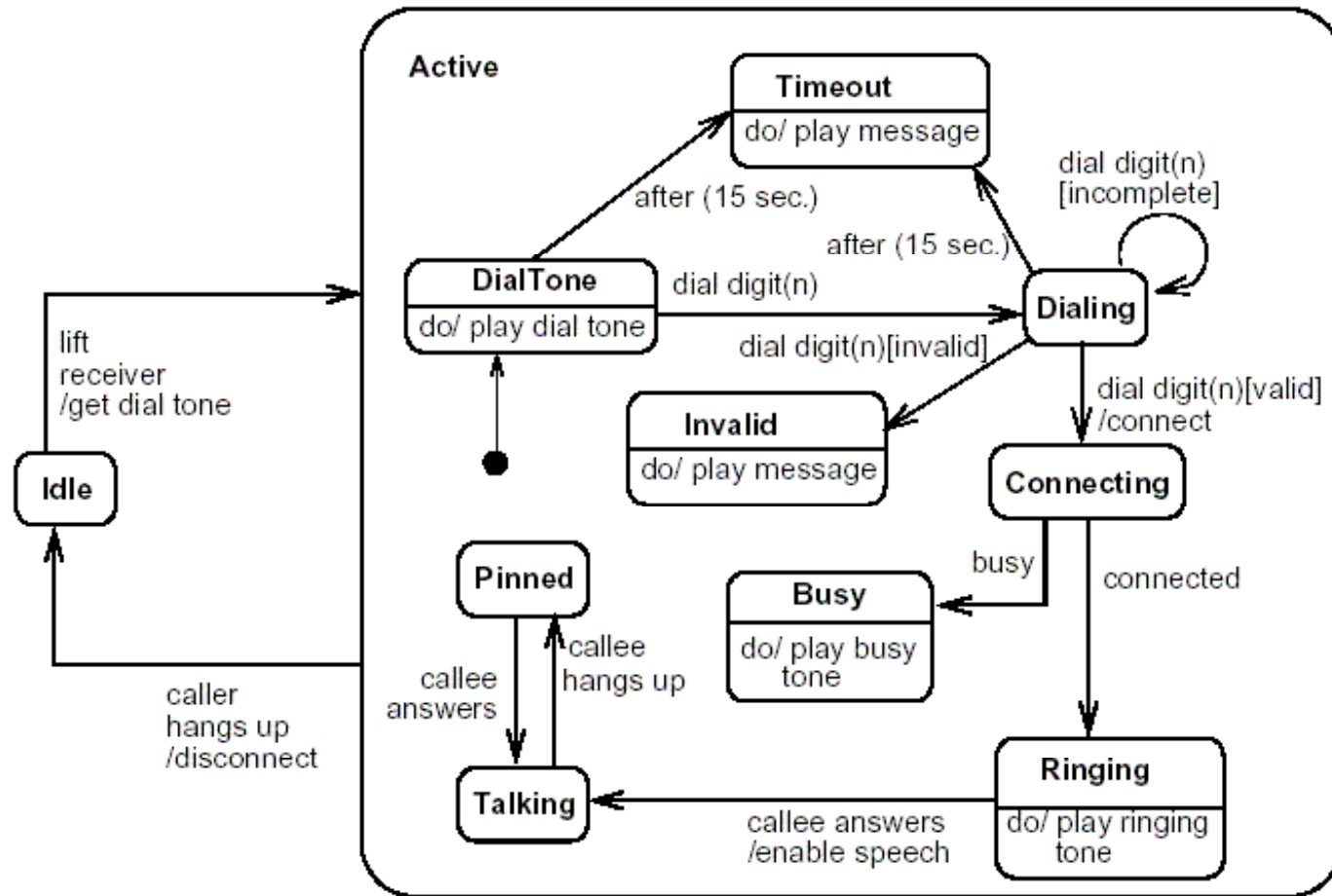


Ambos subestados son muy similares pero se diferencian en la comparación de la temperatura.



Ejemplo 2

Teléfono



Ejemplo 3

Venta automática de entradas.

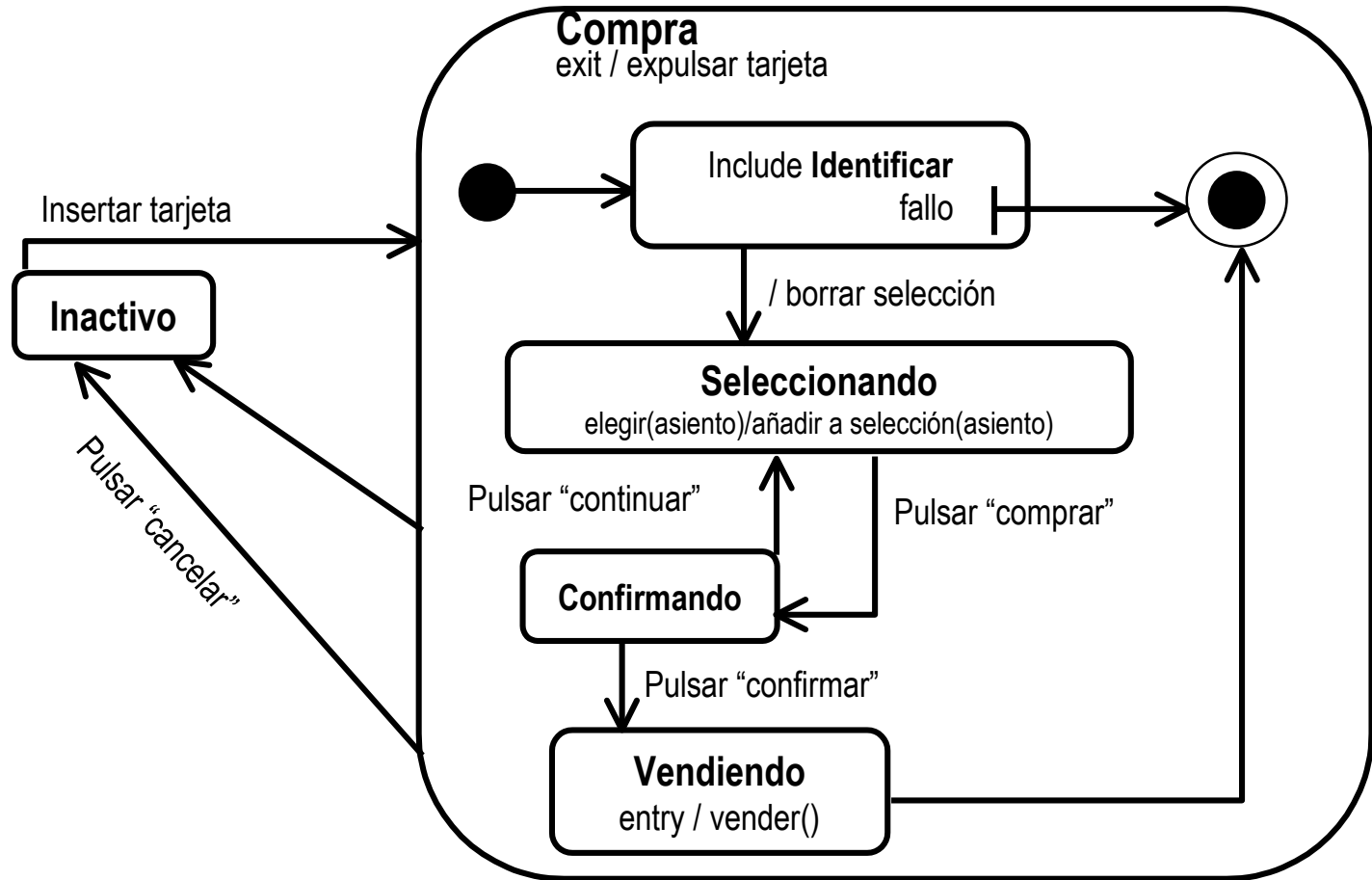


Diagrama de Actividad (Vista dinámica)

Contenido

Diagrama de actividad.

- Introducción
- Componentes
- Resumen
- Ejemplos

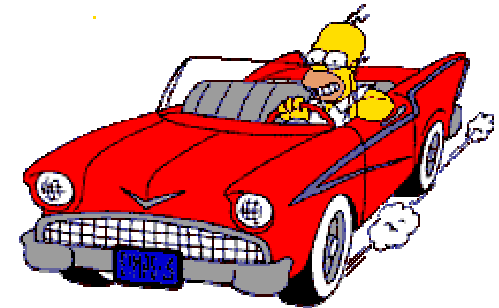
Introducción



Repostar



Contemplar



Conducir



Lavar



Multar

Introducción

Los diagramas de actividad se centran en las relaciones entre las actividades necesarias para la realización de un proceso. Muestra la dependencia entre ellas.

- Es una forma especial de ME, prevista para modelar cálculos y flujos de trabajo.
- Es utilizada para describir una secuencia de acciones, las cuales pueden corresponder a distintos niveles de abstracción de un sistema: modelización del negocio, algoritmo de una operación en una clase, la especificación de un caso de uso
- Es una “mezcla” de ME, Redes de Petri y Diagramas de Flujo tradicionales.

Introducción

- Pueden involucrar a objetos de varias clases y están orientados a las acciones, no a los estados.
- Permiten expresar la concurrencia de diversas acciones y su secuenciamiento.
- Están formados por:
 - Estados (de actividad y de acción)
 - Eventos disparadores
 - Transiciones de terminación
 - Bifurcaciones (actividades de decisión)
 - Barras de sincronización
 - Condiciones de guarda
 - Calles / Roles
 - Mensajes.

Componentes

Estado de actividad: representa la ejecución de una sentencia en un procedimiento o el funcionamiento de una actividad en un flujo de trabajo.

- Solo esperan por su terminación y no por otro evento.
- Sus transiciones no suelen llevar asociadas eventos.
- Puede ser terminado externamente por un evento que fuerce la transición de salida del estado., pero eso no significa que esté esperando por ese evento
- Puede hacer referencia a una submáquina (otro grafo de actividades).

Componentes

Estados de acción: similares a los de actividad pero son atómicos y no permiten transiciones mientras están activos.

- Su propósito es ejecutar una acción y luego realizar una transición de finalización a otro estado.
- No puede tener transiciones disparadas por eventos, ni una estructura interna.

Componentes

Eventos disparadores: causan las transiciones entre las distintas actividades. Puede tener condición de **guarda** que se evalúa en el momento en que TERMINA la actividad o acción en ese estado.

Transición de terminación: asociada a un estado carece de evento disparador y es accionada por la terminación de la actividad o acción del estado del que sale. Puede tener condición de **guarda** que se evalúa en el momento en que TERMINA la actividad o acción en ese estado.

Componentes

Bifurcación (actividad de decisión): asociadas a las transiciones plantean diversos flujos de trabajo asociadas a una condición de guarda.

Barras de sincronización: permiten diversificar o reunificar diversos flujos de control concurrentes.

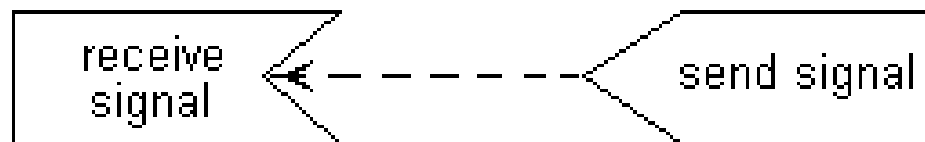
Componentes

Calles: partición de los grafos de actividades que organiza las responsabilidades de las actividades según diversas unidades organizativas de modelos de negocio.

Roles: se colocan en cada calle indicando el rol del objeto que realiza la actividad.

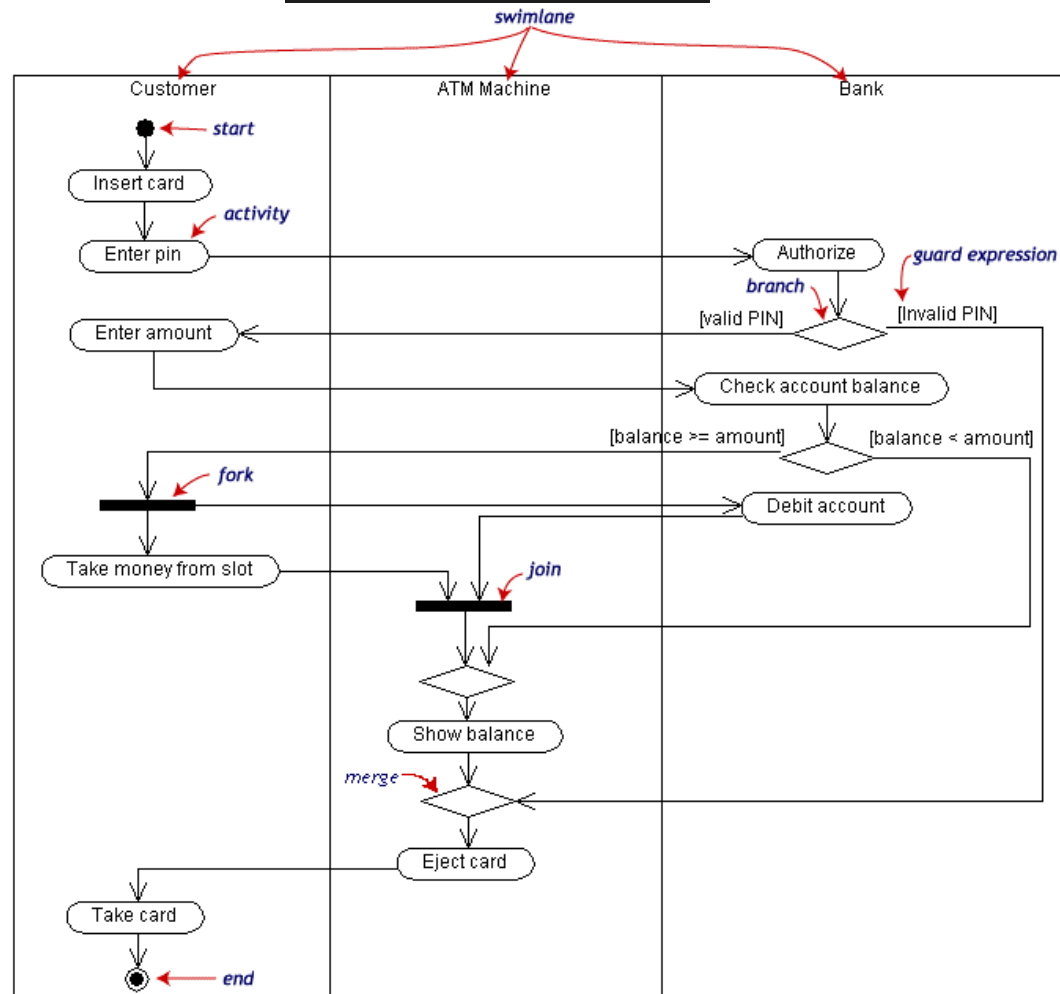
Mensajes: señales de comunicación entre los procesos

Sintaxis:



Resumen

Cajero



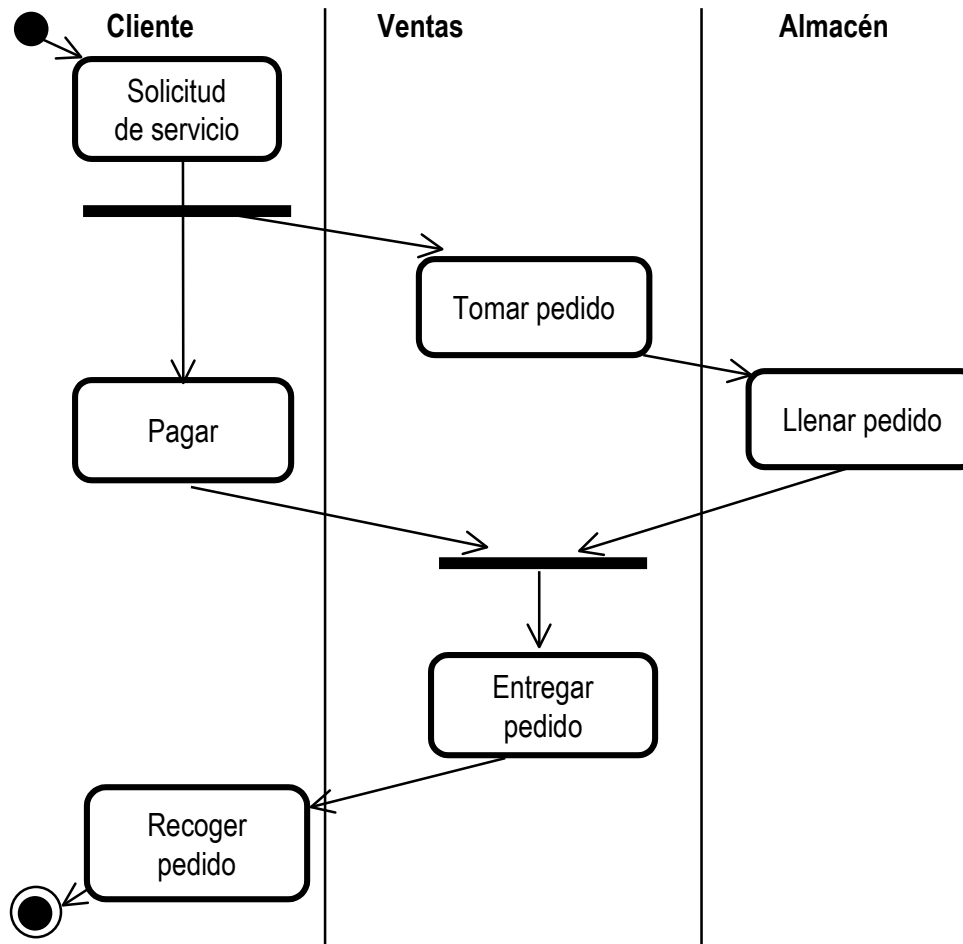
Ejemplo

Pedido de un cliente:

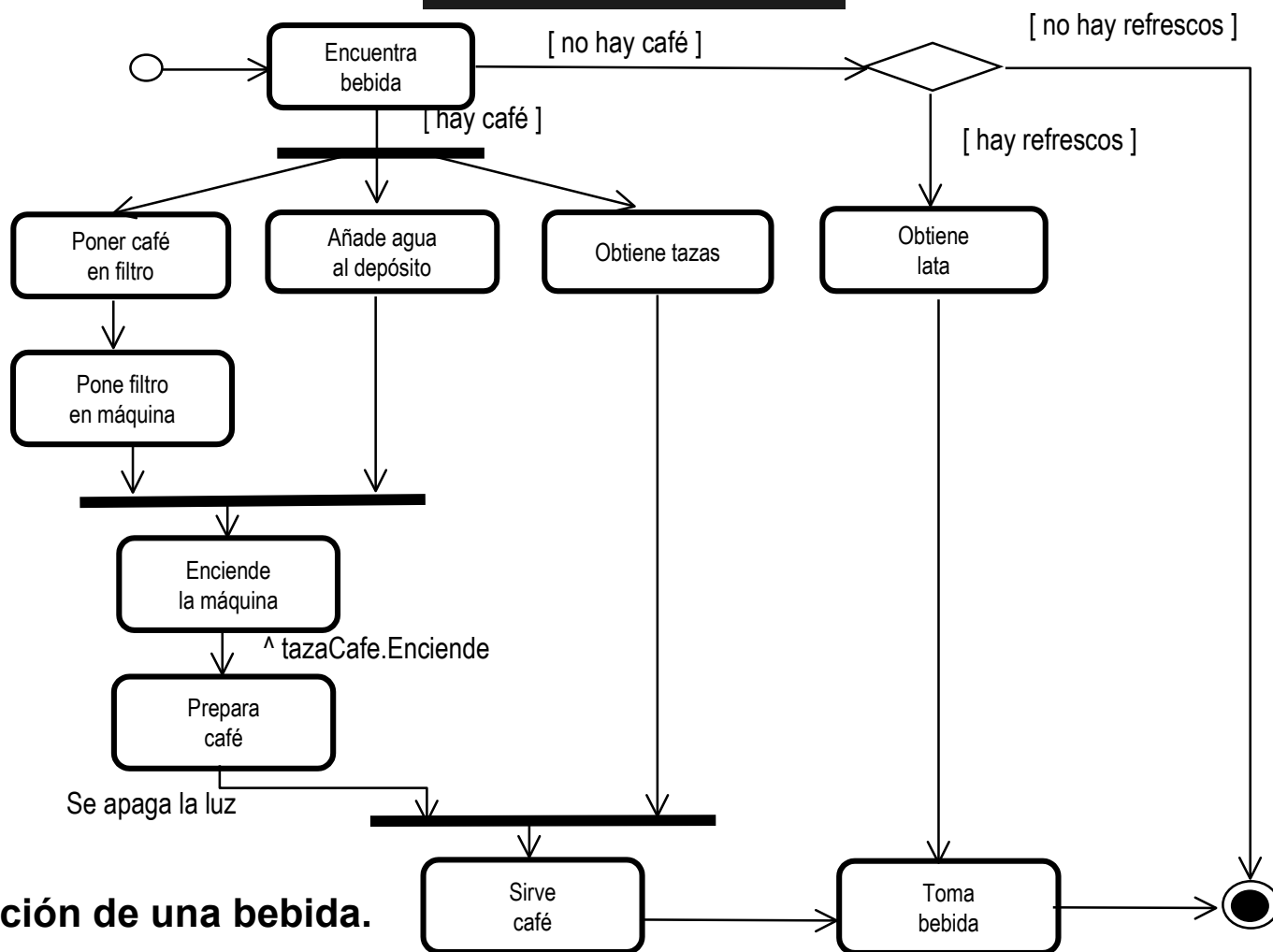
- Un cliente se pone en contacto con nuestro departamento de Ventas para formalizar un pedido.
- En ventas, se registran los datos del pedido y se notifica a Almacén, en donde se atiende. (por simplificar, supongamos que siempre hay existencias)
- Una vez que el cliente ha pagado, Ventas se encargará de entregar el pedido, que será recogido por el cliente.

Diagrama de actividad

Ejemplo



Ejemplo 2



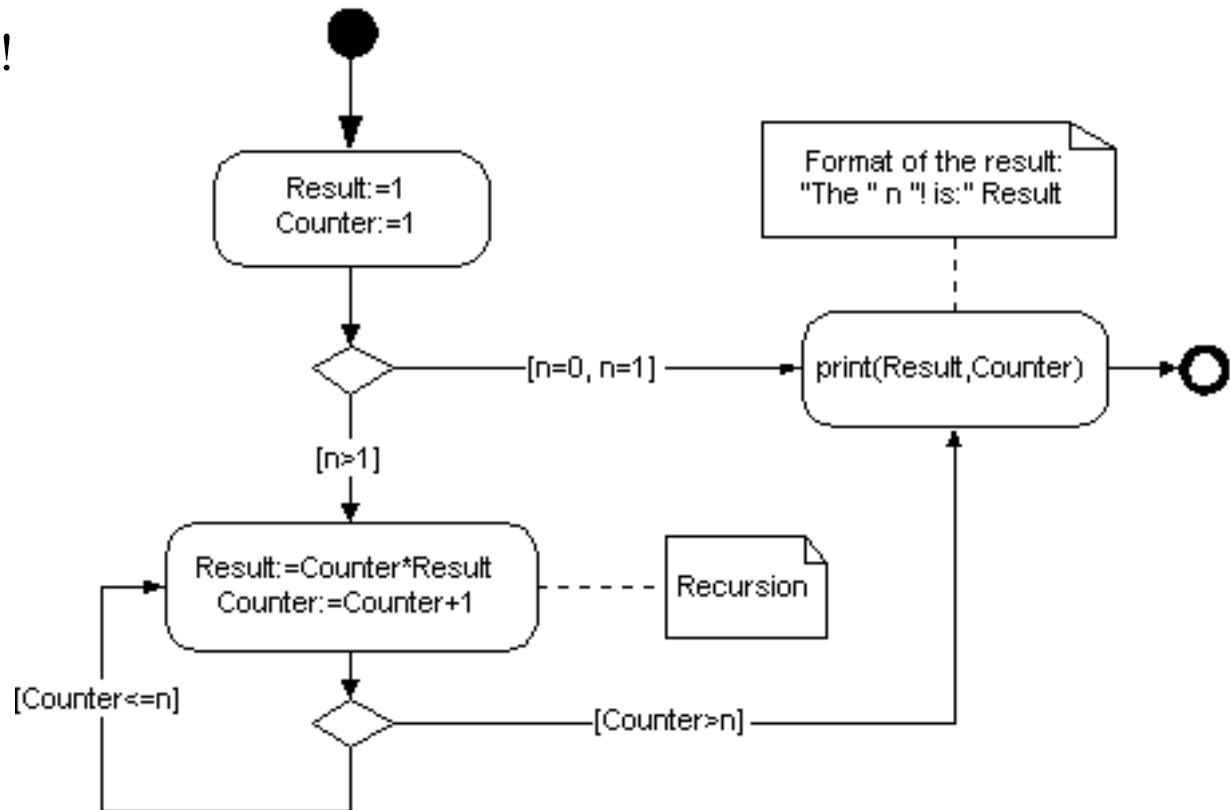
Preparación de una bebida.

Ejemplo 3

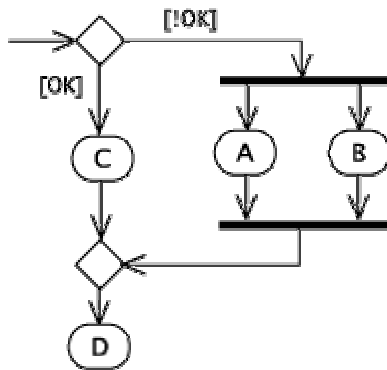
Modelado del factorial de un número:

$$0! = 1$$

$$n! = n * (n-1)!$$



Cuestión



¿Cuál de los pseudo-códigos implementa el diagrama de actividad anterior?

A) If (OK) then
do C
else
do A and B //either order
do D

C) if (OK) then
do C
else
do A OR B not both
do D

B) do C
do A and B
do D

D) if (OK) then
do C
else
do A
do B
do D

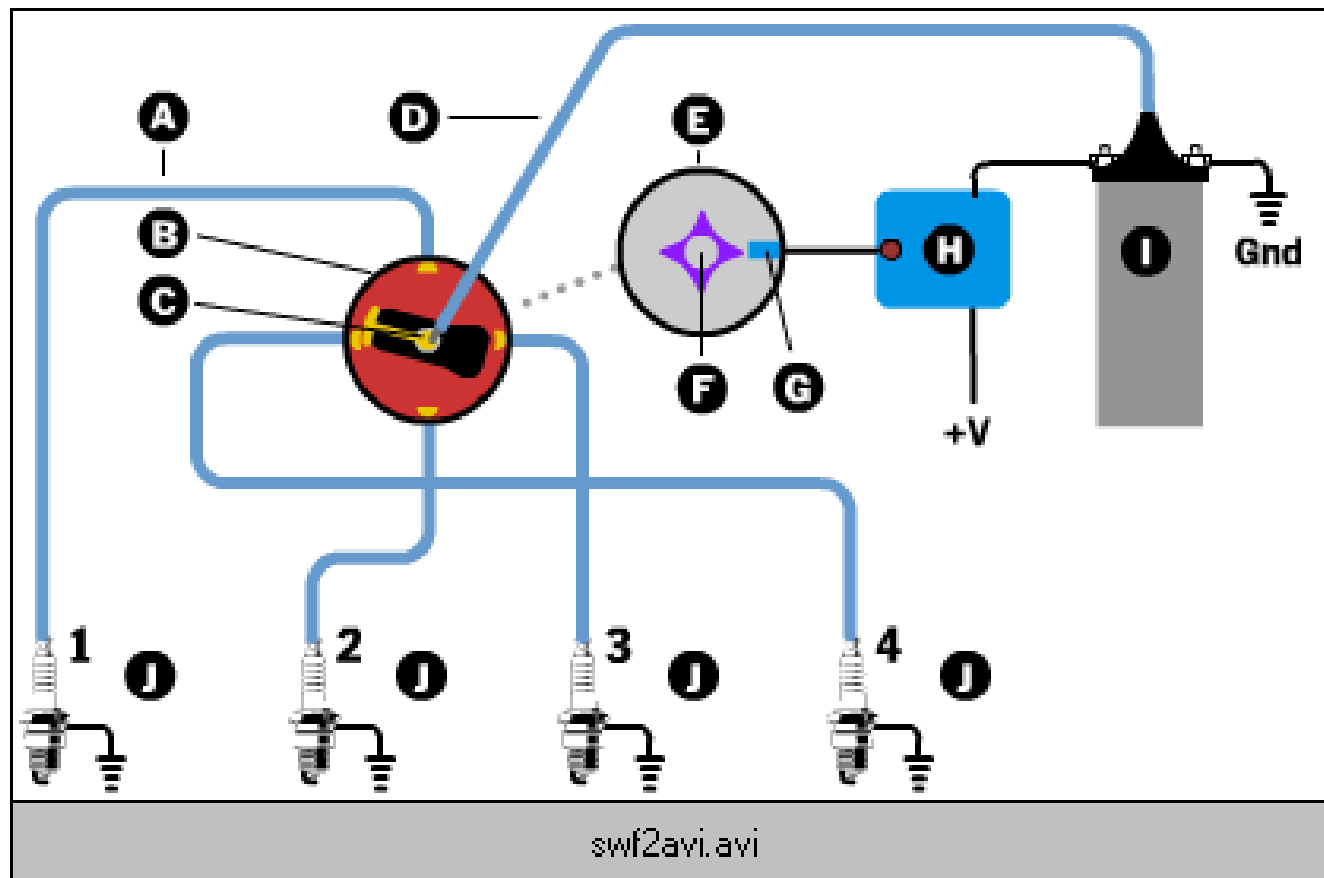
Diagrama de secuencias (Vista de interacción)

Contenido

Diagrama de secuencias.

- Introducción
- Mensajes
- Detalles
- Tipos
- Resumen
- Ejemplos

Introducción



Introducción

Este diagrama se centra en la comunicación entre los objetos. El diagrama de estados se centraba en el comportamiento del objeto.

- Muestra como se comunican e interactúan los diferentes objetos del sistema.
- Como interactúan unos objetos con otros, se sincronizan, ...
- La principal idea es que las interacciones entre los objetos siguen una secuencia específica.
- La dimensión principal en este tipo de diagramas es el tiempo.
- En este diagrama se muestran, por tanto los mensajes que se pasan los objetos a lo largo del tiempo. Estos mensajes quedan ordenados según el momento de tiempo en que se producen.

Introducción

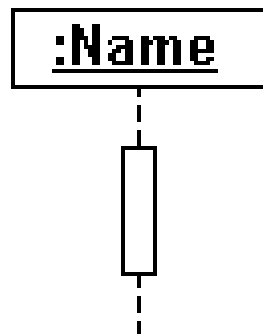
Los diagramas de secuencias poseen dos dimensiones:

- Vertical: Representa el paso del tiempo.
- Horizontal: Representa las instancias de las clases, o sea los objetos.

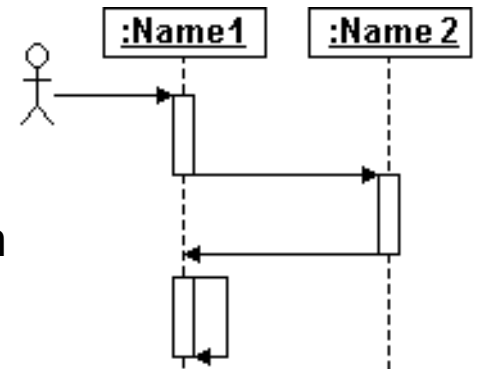
En los diagramas de secuencia se muestran los mensajes entre los diferentes objetos.

El orden horizontal de los mensajes no es importante.

La representación de un objeto es la siguiente:

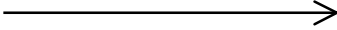
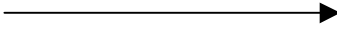
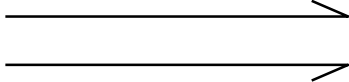
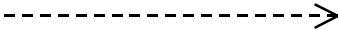


Los símbolos básicos del diagrama son siguientes:



Mensajes

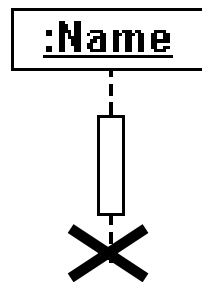
Los mensajes se representan a través de flechas. Los diferentes tipos de mensajes son los siguientes:

- **Simples:** La ejecución termina cuando se envía el mensajes. Es decir, se transfiere el control de un objeto a otro.

- **Síncronos:** El objeto que ha mandado el mensaje se espera a recibir la respuesta antes de seguir ejecutando.

- **Asíncronos:** El objeto que ha mandado el mensaje continua su ejecución sin esperar la respuesta del otro.

- **Retorno:** Retorno de una llamada a una operación.


Detalles

La multiplicidad se puede añadir a los mensajes. La notación es la misma que se a utilizado en los diagramas de clases. La multiplicidad se añade antes del mensaje e indica las veces que se manda esa señal.

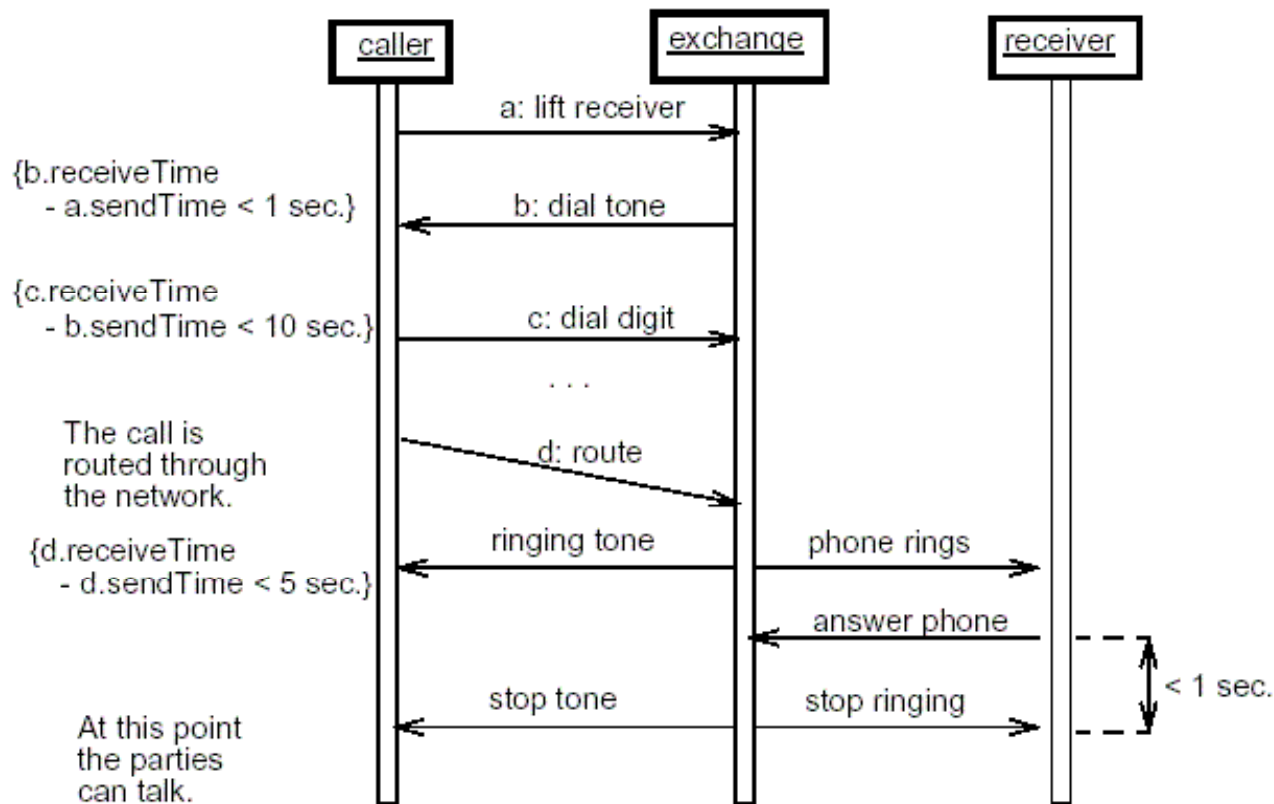
La vida de un objeto es una línea vertical dividida. Se puede representar la destrucción de un objeto añadiendo una cruz al final de la línea de vida del objeto.



Detalles

Las restricciones se pueden añadir a lo largo de la línea vertical indicando el tiempo necesario que ha de transcurrir entre distintos eventos.

Ej:



Detalles

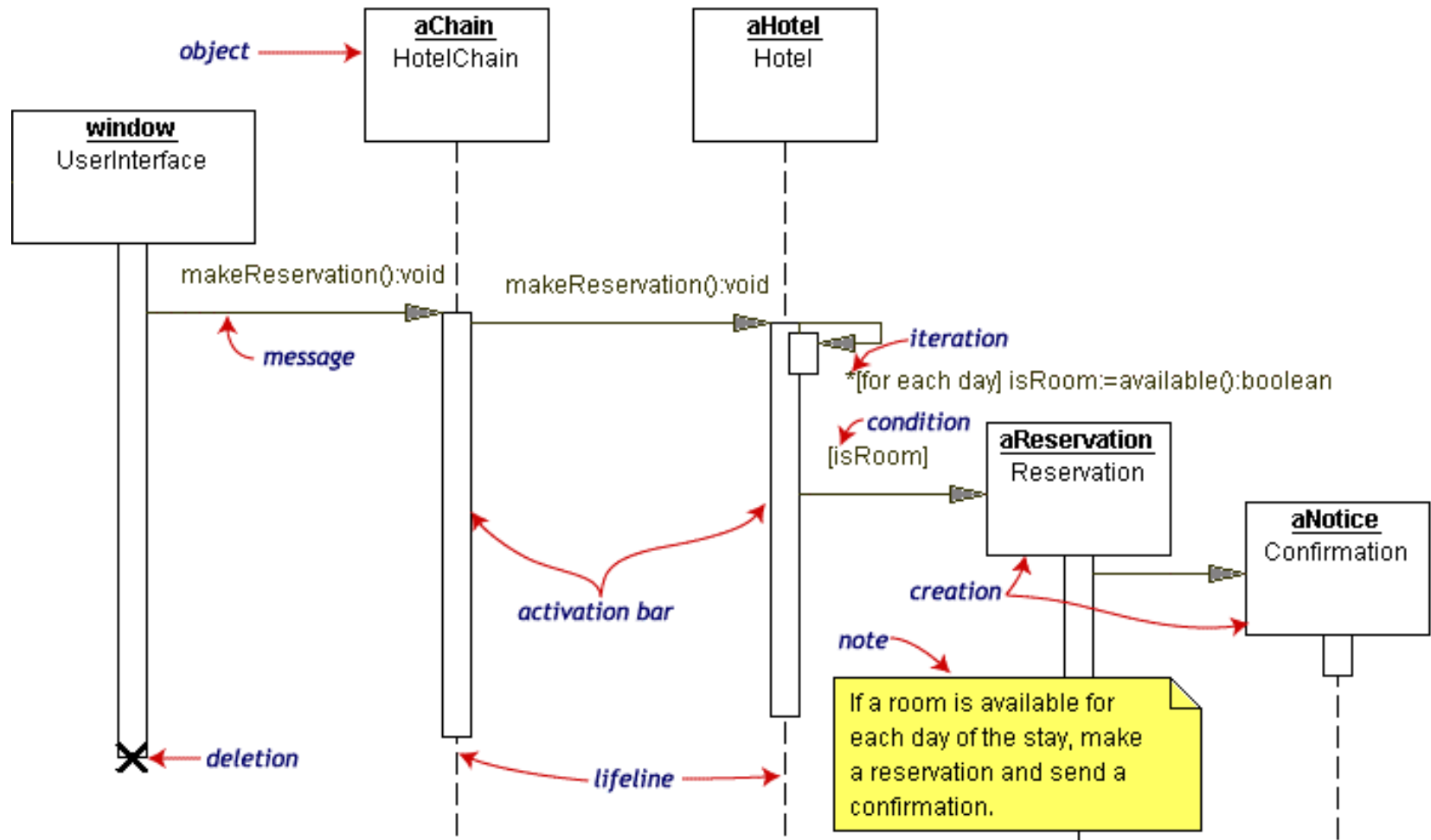
- La autodelegación es un mensaje que un objeto se envía a sí mismo.
- Un regreso es un retorno de un mensaje, no un nuevo mensaje. Su flecha es punteada.
- Si un objeto (que no aparece, inicialmente en el diagrama) es creado durante la interacción, su línea de vida comienza con la recepción del mensaje estereotipado create.
- Análogamente, la destrucción de un objeto se realiza mediante el mensaje estereotipado destroy.
- Si el objeto se llama a si mismo recursivamente se superpone otra copia de la línea continua de activación.
- Se pueden enviar simultáneamente dos mensajes, pero no recibirlos, porque no se puede garantizar esa simultaneidad.
- Pueden existir bifurcaciones en la línea de vida de un objeto. Ahora bien, un escenario no representa “decisiones por tomar” sino “ya tomadas”.

Tipos

Los diagramas de secuencias se clasifican de acuerdo con su generalidad. Esta clasificación es la siguiente:

- **Instancias:** se considera que son una instancia cuando sólo modelan un escenario posible en la situación global entre los objetos.
- **Genéricos:** se consideran genéricos cuando incorporan todos los posibles escenarios que se pueden producir entre los objetos que se encuentran en el modelo.

Resumen



Ejemplo

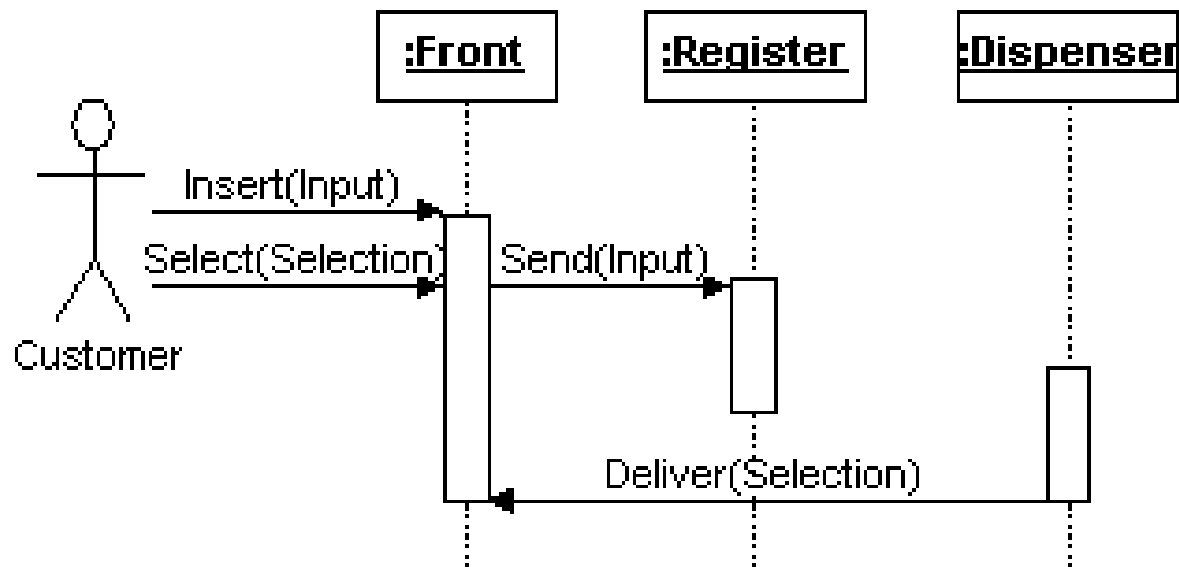
Vamos a modelar la máquina de servicio.

- Supongamos que la máquina posee tres objetos para hacer el trabajo:
 - El **frontal**: representa la interfaz con el comprador.
 - El **registro de dinero**: representa la parte encargada de recoger las monedas.
 - El **dispensador**: representa la parte encargada de dar el producto seleccionado por el cliente.

Ejemplo

- El diagrama de secuencias se guiara a través de las siguientes acciones:
 - El cliente inserta las monedas.
 - El cliente hace una selección.
 - El dinero va hasta el registro.
 - El registro comprueba que el producto deseado se encuentra en el dispensador.
 - El registro actualiza su caja.
 - El registro indica al dispensador que entregue el producto a través del frontal.

Ejemplo



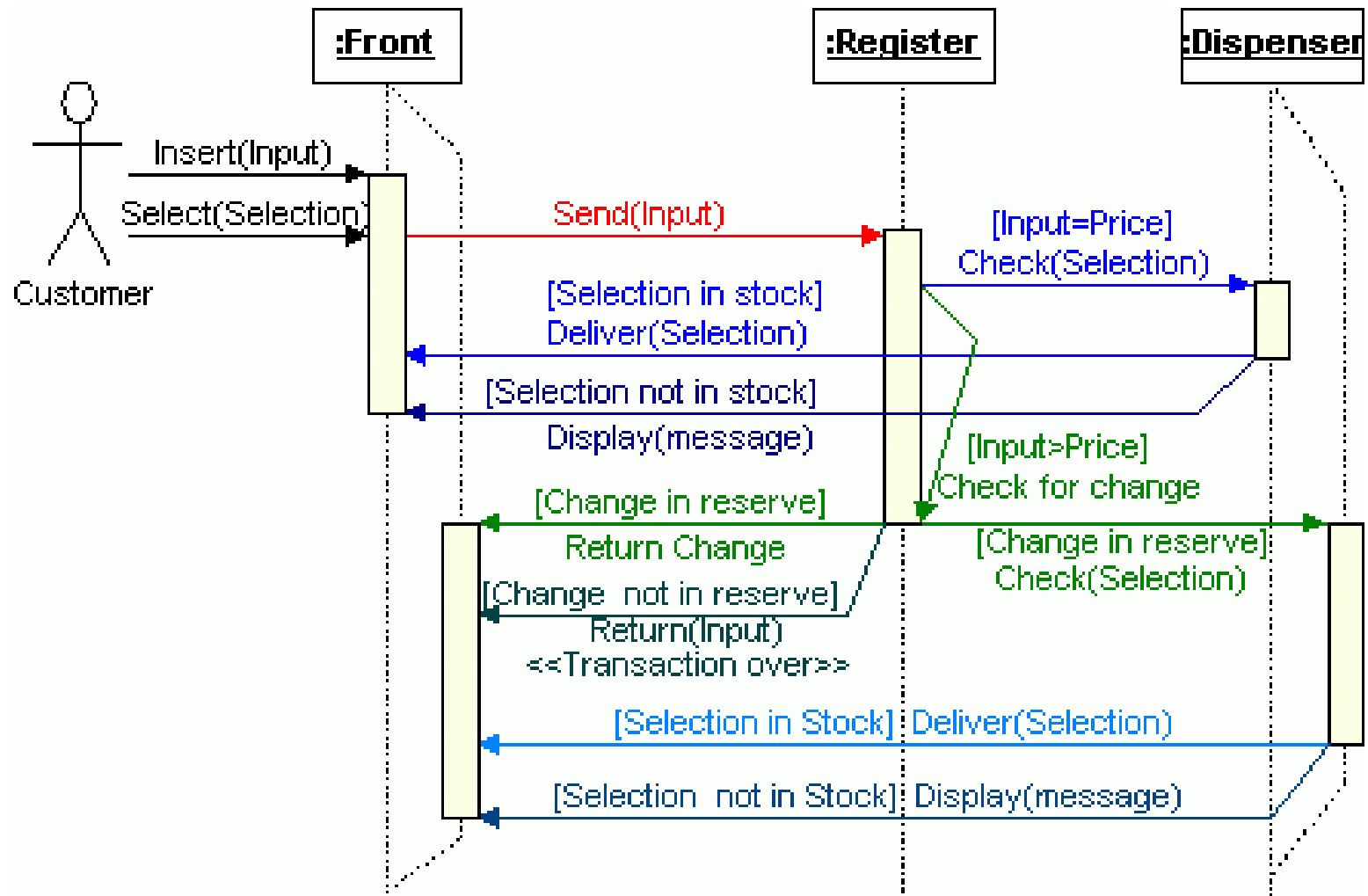
Como este diagrama sólo modela el caso mejor se considera que es un diagrama de tipo **instancia**.

Ejemplo

Si consideramos todos los casos posibles que se pueden dar podemos generar un diagrama genérico.

- El escenario relativo a la inexistencia de producto implica la siguiente secuencia de pasos:
 - Después de seleccionar un producto inexistente la luz indicadora de la inexistencia parpadea.
 - Se pide al cliente otra elección.
 - El cliente tiene la oportunidad de pedir que le devuelvan el dinero.
 - Si el cliente selecciona otra opción todo funciona como en el mejor caso si la cantidad introducida es correcta. Si no la máquina se va al escenario relativo a cantidad de dinero incorrecta.
 - Si el cliente selecciona otro producto inexistente el proceso se repite hasta que el cliente selecciona un producto existente o solicita que le devuelvan el dinero.

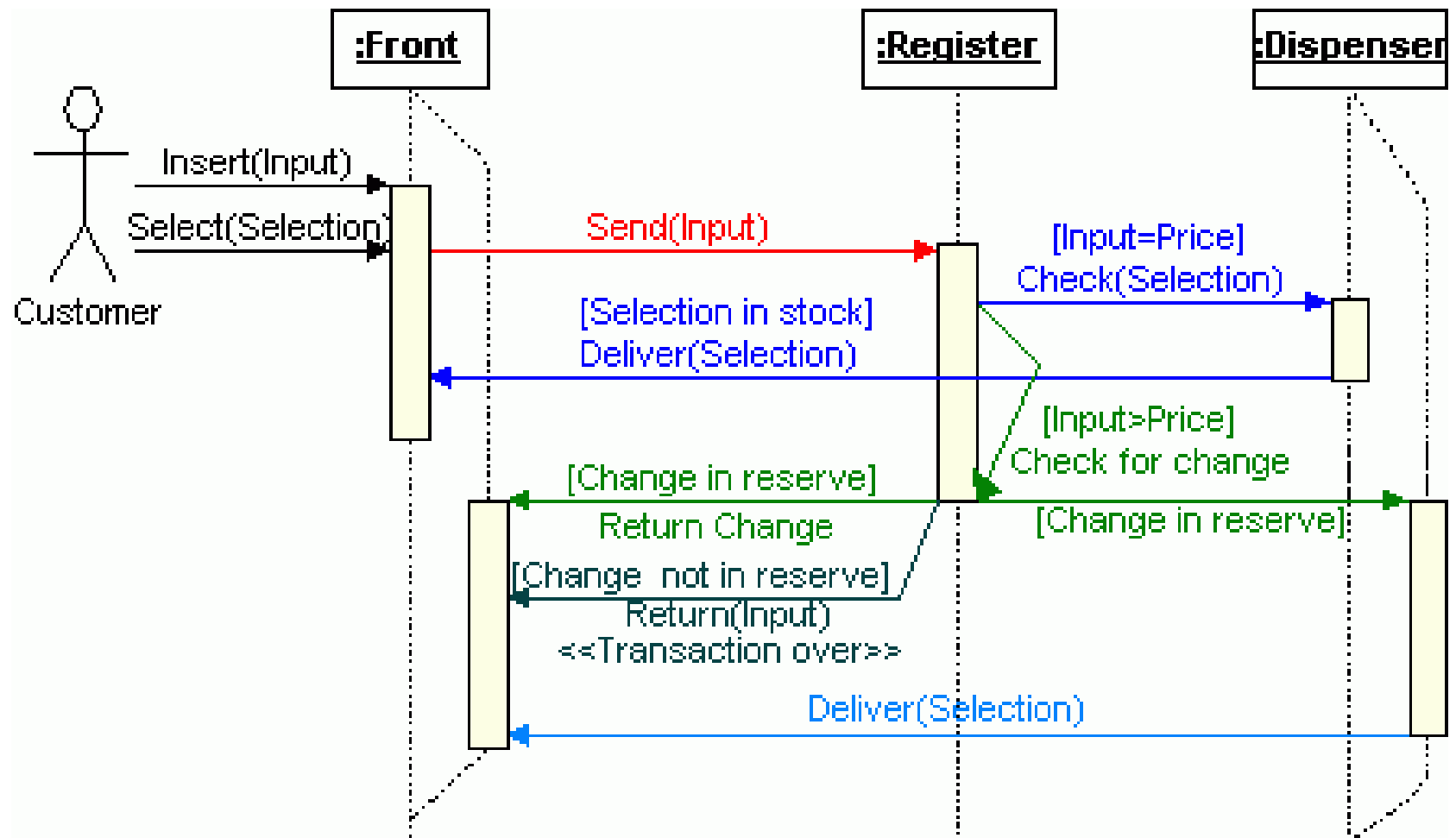
Ejemplo



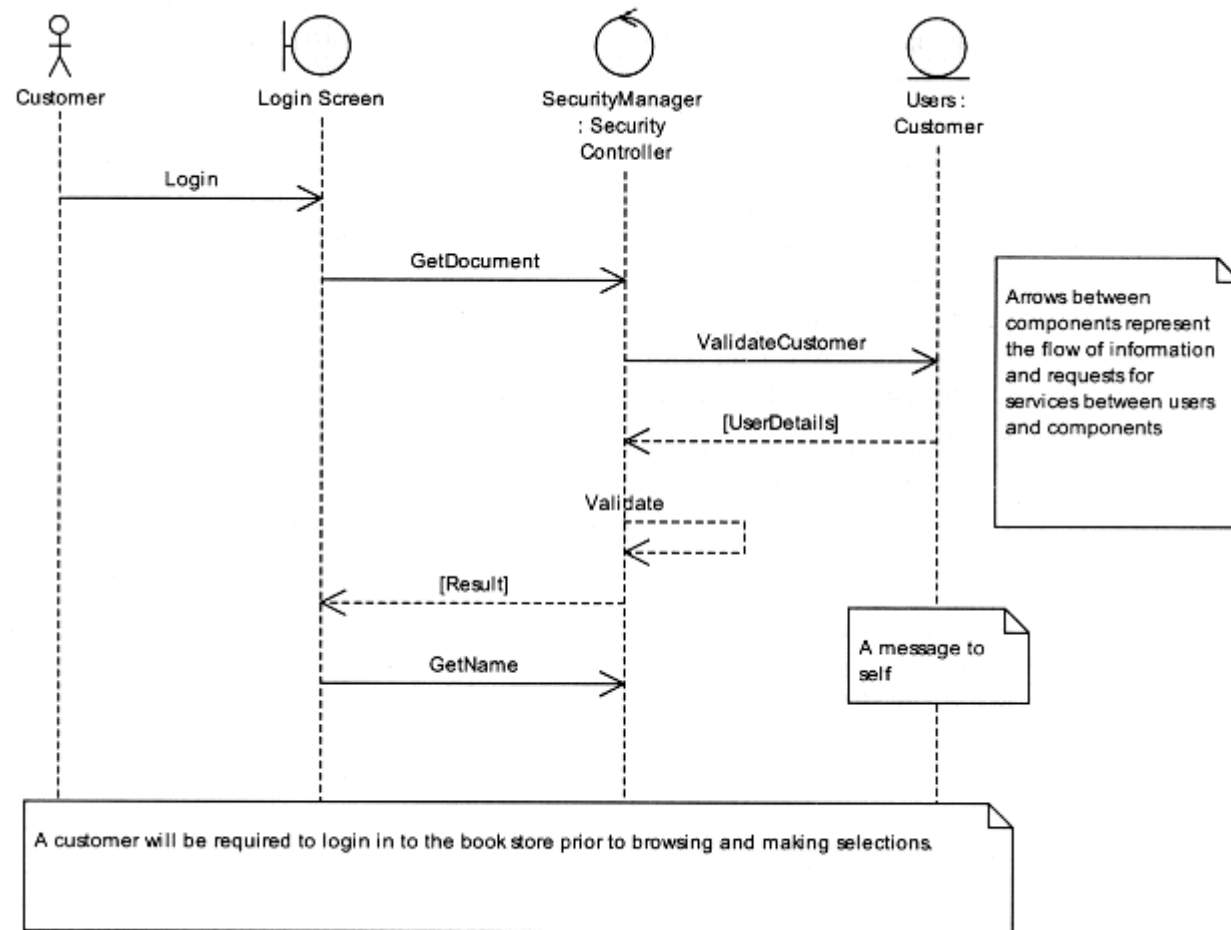
Ejemplo

- El escenario relativo a la cantidad incorrecta de dinero implica la siguiente secuencia de pasos:
 - El registro comprueba que la cantidad de dinero es correcta con respecto al producto elegido por el cliente.
 - Si la cantidad es superior, se calcula la diferencia y se mira si se posee cambio.
 - Si se posee cambio el registro retorna el cambio y todo sucede como en la mejor situación.
 - Sino, el registro devuelve la cantidad e indica al cliente que no posee cambio.
 - Si la cantidad es menor que el precio, el registro no hace nada y la máquina espera.

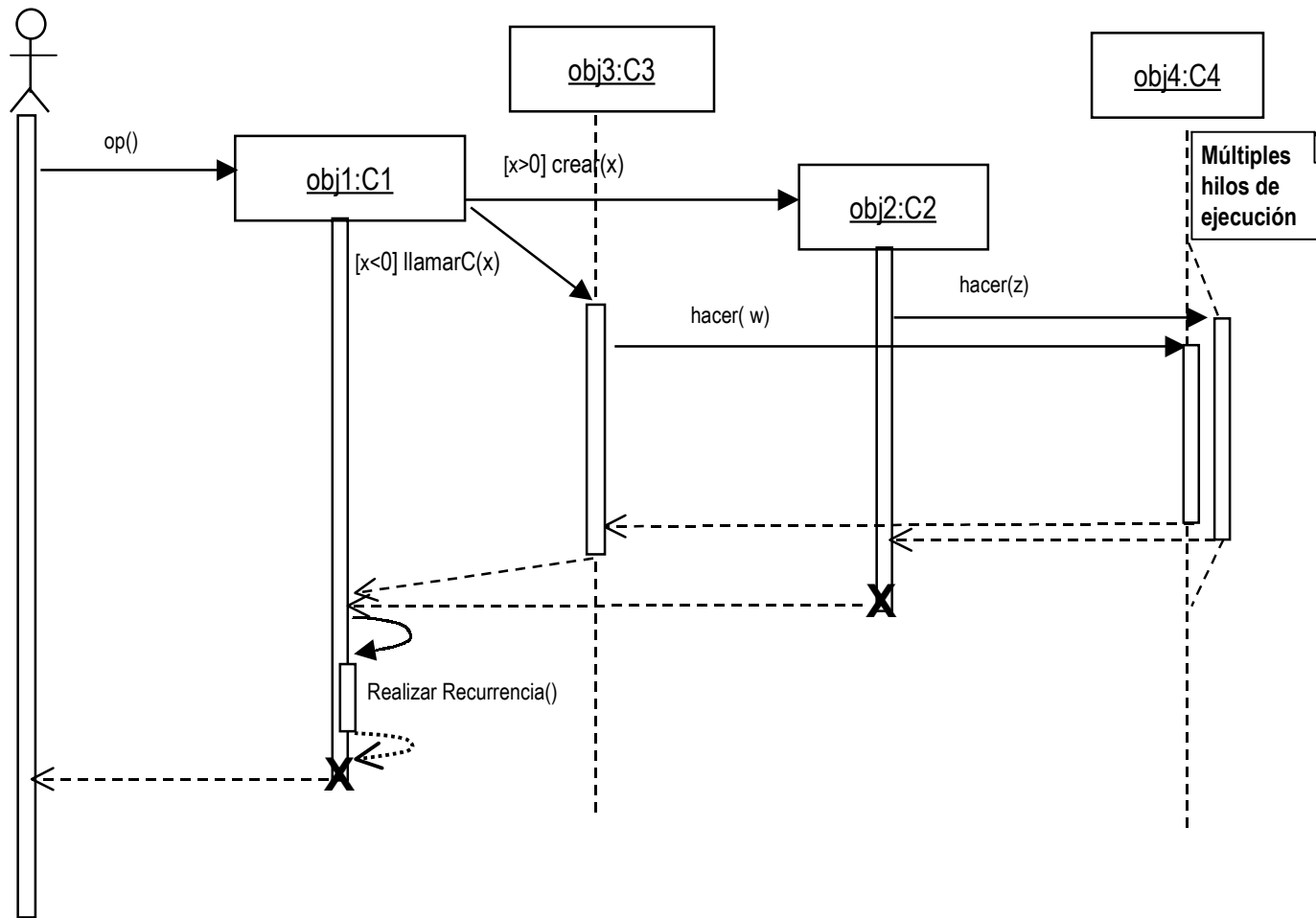
Ejemplo



Ejemplo 2



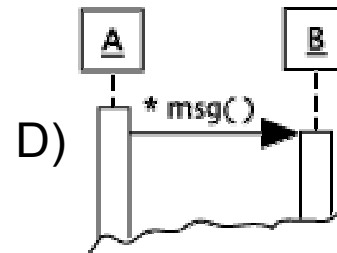
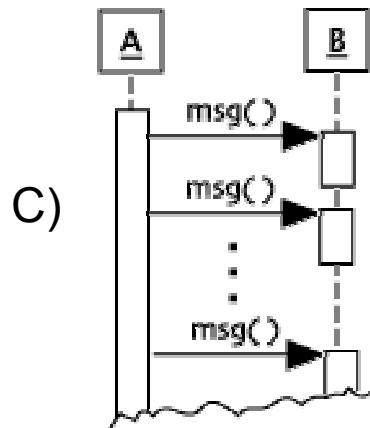
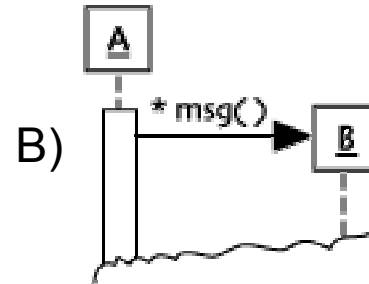
Ejemplo 3



Cuestión

¿Cuál de los siguientes diagramas modela la siguiente acción?

“El objeto **A** envía la señal *msg* al objeto **B** varias veces”



Diagramas de Colaboración (Vista de Interacción)

Contenido

Diagrama de colaboración

- Introducción
- Representación
- Resumen
- Ejemplos
- Otros usos

Introducción

Este tipo de diagrama, al igual que el diagrama de secuencia, muestra como interactúan los objetos a través de los mensajes que se pasan.

El diagrama de secuencia y el de colaboración son similares. Son semánticamente equivalentes (presentan la misma información) y se puede pasar de uno a otro. La principal distinción es: el diagrama de secuencia se fija en el tiempo, y el de colaboración se fija en la memoria.

El diagrama de colaboración es una extensión del diagrama de objetos. En vez de mostrar las relaciones entre los objetos muestra los mensajes que estos se envían.

Los mensajes se muestran etiquetados con el orden que les corresponde en la secuencia de mensajes. Los números se separan por comas.

Cuando modelamos una expresión condicional debemos añadir un decimal al número del mensaje.

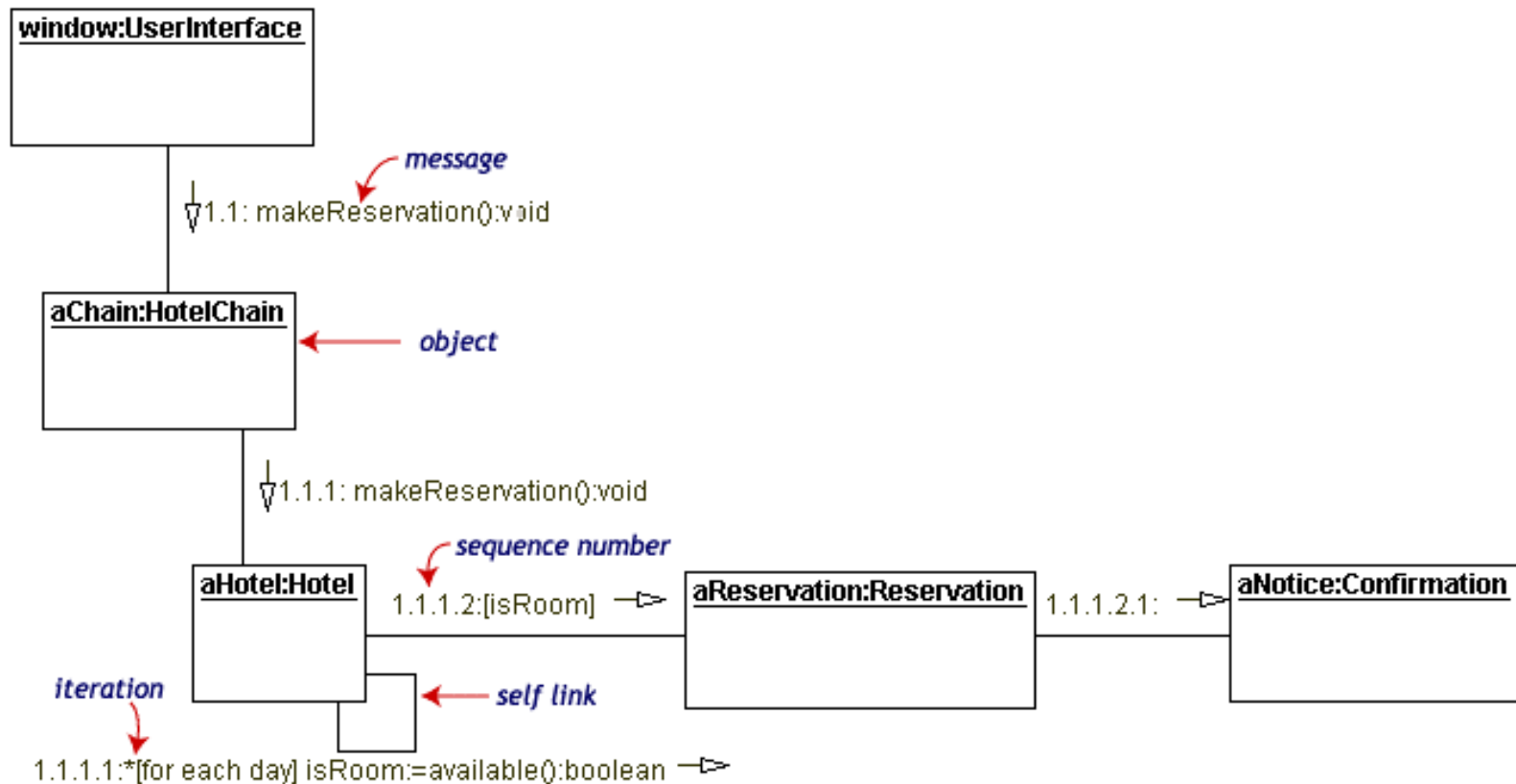
Representación

Representación de:

- **Cambio de estado de un objeto.** Solo es necesario añadir otro objeto al diagrama. Conectar ambos objetos y etiquetar la línea con el estereotipo <<become>>.
- **Creación de un objeto.** Añades el objeto que se crea y etiquetas la transición con el estereotipo <<create>>.
- **Objetos múltiples.** Se representan a través de varios rectángulos.



Resumen

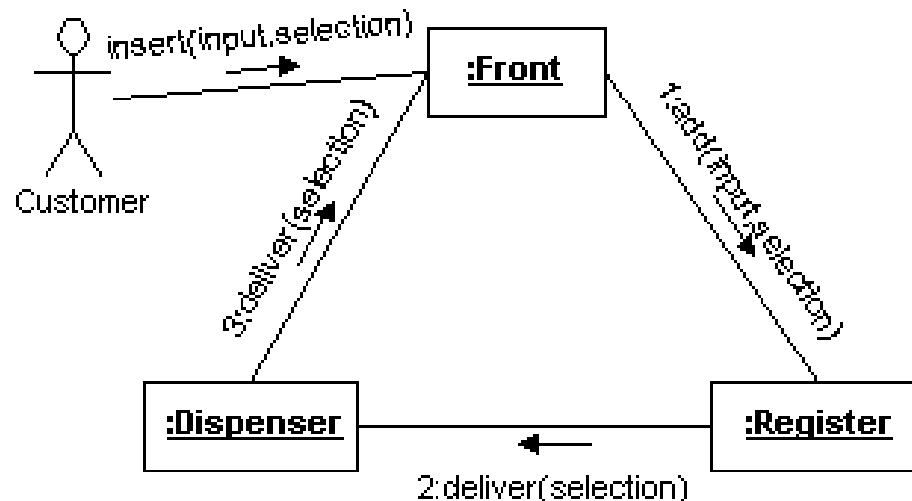


Ejemplo

Como es lógico volvemos a centrarnos en la máquina dispensadora.

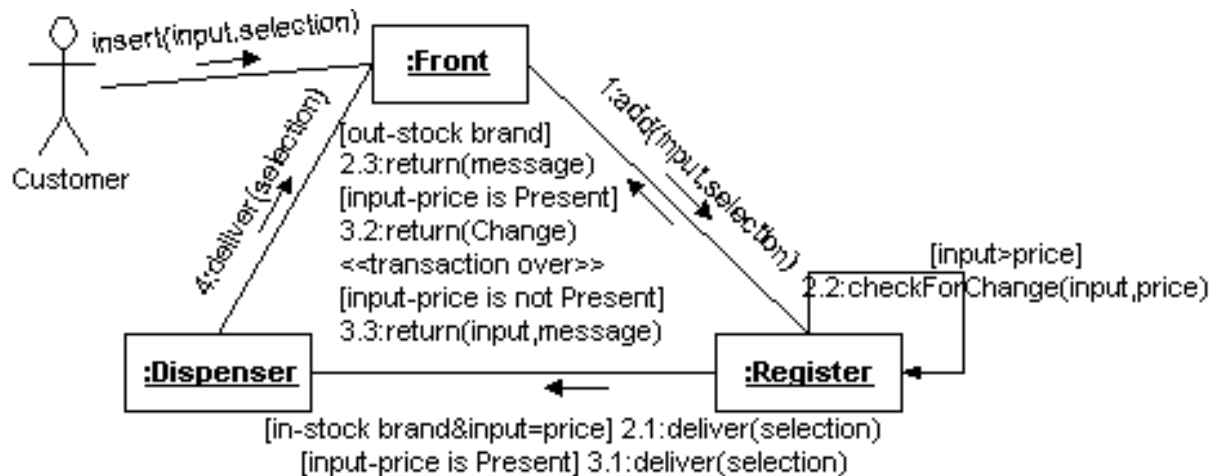
El mejor caso consistía en:

- El cliente inserta las monedas y realiza una selección.
- Cuando el registro obtiene el dinero, se ordena al dispensador que entregue el producto deseado.
- El dispensador da el producto.



Ejemplo

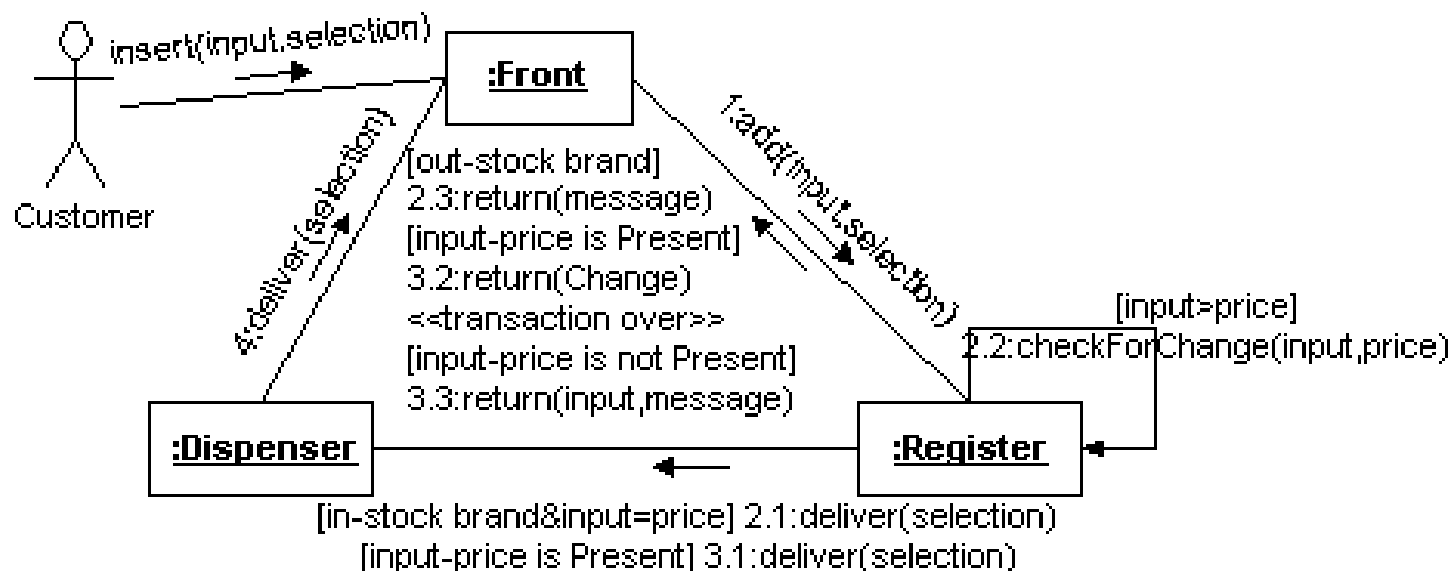
Cuando examinamos el comportamiento nos dimos cuenta de otros escenarios. Como el escenario relativo a la inexistencia del producto.



De este escenario se pueden deducir dos escenarios el relativo a producto inexistente y el relativo a tener el producto. Si consideramos que se producen detrás del mensaje número 1, el número del mensaje será el 2.1 para el escenario relativo a tener el producto y 2.2 para el relativo al producto inexistente.

Ejemplo

El escenario relativo a no tener el cambio correcto se modela con el siguiente diagrama de colaboración:



En este escenario hay dos posibilidades para el mensaje 2 y tres posibilidades para el mensaje 3.

Otros usos

El diagrama de colaboración se puede utilizar a un nivel de especificación del sistema expresando los roles y las asociaciones existentes entre las clases.

Ej:

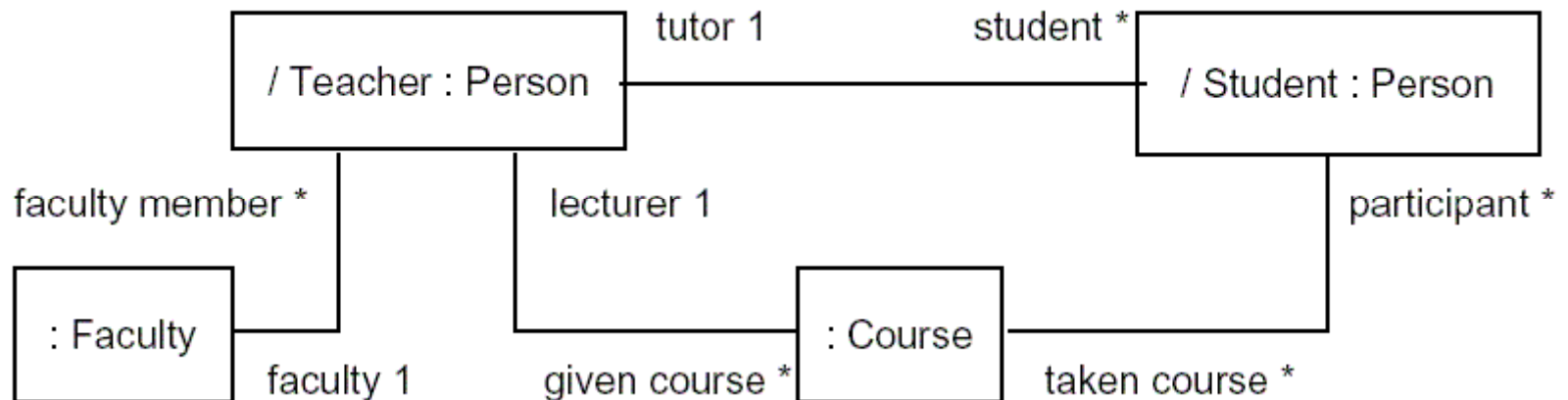


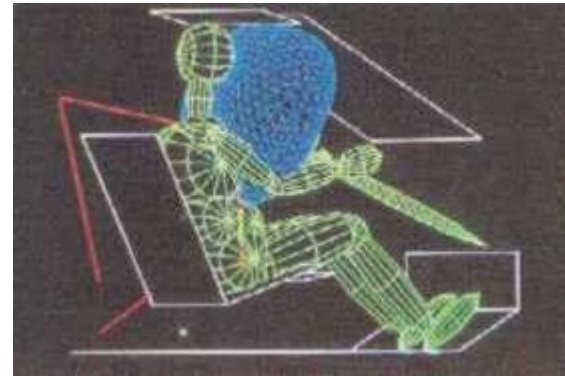
Diagrama de Componentes (Vista de Implantación)

Contenido

Diagrama de componentes

- Introducción
- Representación
- Relaciones
- Tipos
- Ejemplos

Introducción



Introducción

Este tipo de diagrama muestra los componentes software existentes en un sistema.

- Muestra el empaquetado físico de las partes reutilizables del sistema en unidades denominadas componentes.
- Muestra la implementación de los elementos del diseño mediante componentes, interfaces y sus dependencias.
- Muestra la organización del código y demás archivos del sistema (fuentes, ejecutables, archivos de librería...) y sus dependencias.

Introducción

Se modelan componentes software y sus relaciones. Por tanto:

- El cliente puede ver la estructura del sistema terminado.
- Los implementadores tienen una estructura sobre la que trabajar.
- Los escritores técnicos que tiene que realizar la documentación y los ficheros de ayuda pueden comprender sobre lo que están trabajando.
- Es fácilmente reutilizable.

Piensa en un componente como la implementación de una clase. La clase representa una abstracción del conjunto de atributos y operaciones. Un componente puede ser la implementación de más de una clase.

Cuando se trabajan con componentes es necesario trabajar con las interfaces que ellos implementan.

Introducción

Un diagrama de componentes esta formado por:

- Componentes.
- Interfaces.
- Relaciones.
- Otros tipos de símbolos.

Introducción

Componente: pieza de reutilización de alto nivel a partir de la cual se construyen sistemas. Es una unidad física con interfaces bien definidas que puede ser reemplazable en un sistema.

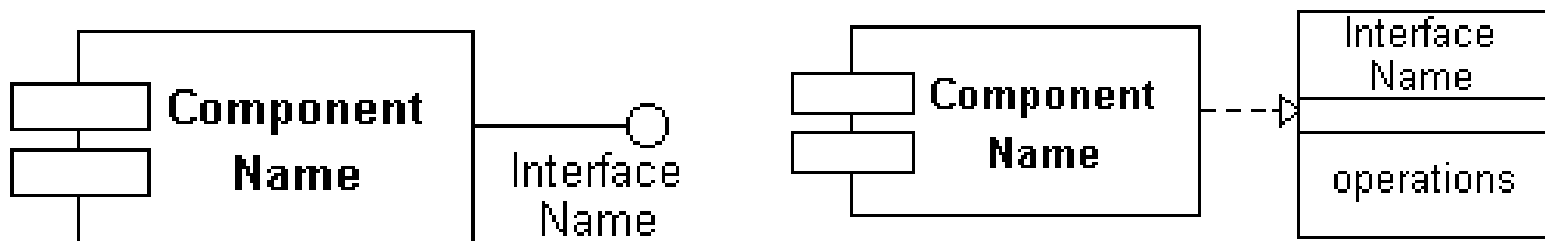
- Cada componente incorpora la implementación de un grupo de clases del sistema.
- Un componente depende de las interfaces de los demás: por tanto es reemplazable por otro que ofrezca una interfaz similar.
- Se puede reutilizar un componente en otro sistema si el sistema nuevo puede reutilizar el componente accediendo a través de su interfaz.
- Las dependencias se implementan mejor a través de las interfaces en vez de forma directa como enlace.

Representación

- Un componente se representa de la siguiente forma:

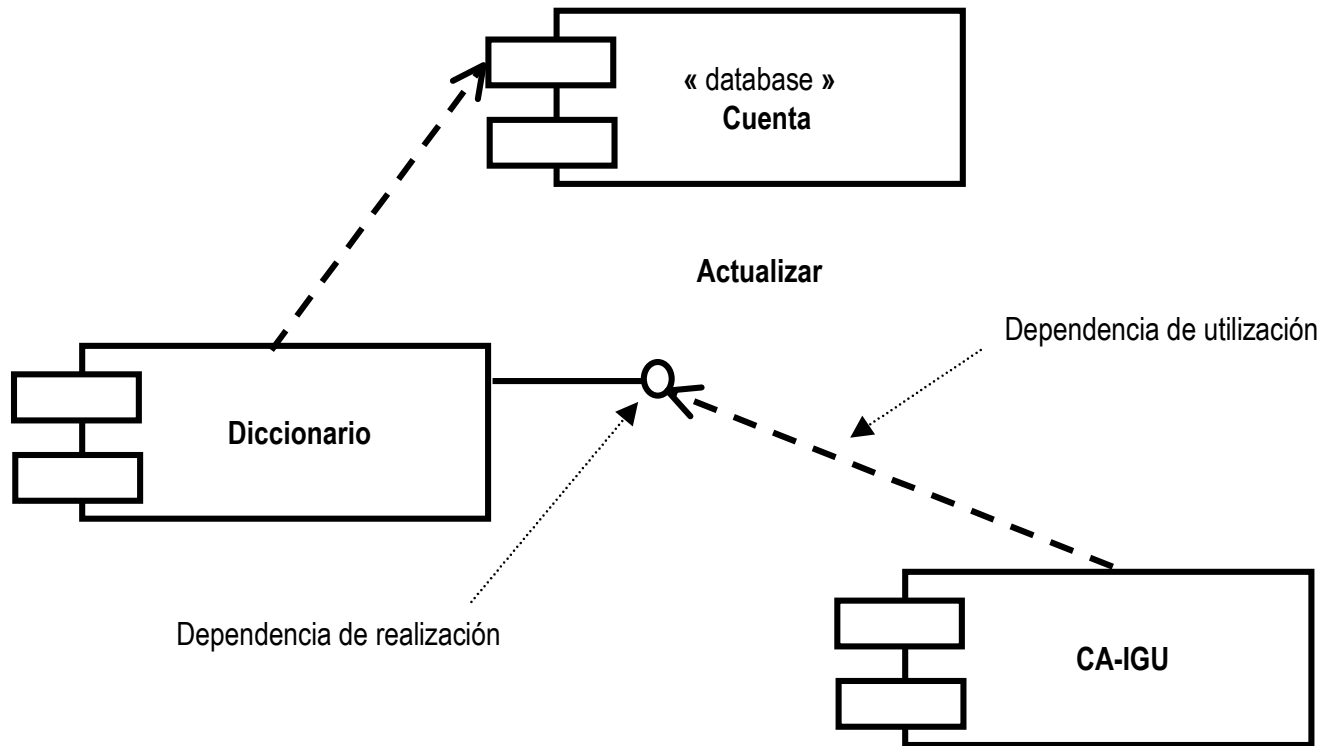


- Un componente junto con la interfaz que implementa se muestra de la siguiente forma:



Relaciones

Las dependencias directas entre componentes se muestran de la siguiente forma:



Tipos

Tipos de componentes:

- **Componentes de desarrollo**, que forman la base del sistema ejecutable (DLL, ejecutables, controles ActiveX, JavaBeans).
- **Componentes del trabajo del producto**, desde los que se han creado los componentes de desarrollo (ficheros de datos y ficheros de código de programa).
- **Componentes de ejecución**, creados como resultado de la ejecución del sistema.

Ejemplo

Vamos a modelar un software para oír música desde un CD-ROM. Para ello utilizaremos un lenguaje visual. Una posible vista del sistema es la que sigue a continuación:



Ejemplo

Como se puede observar el software para escuchar la música necesita los siguientes controles:

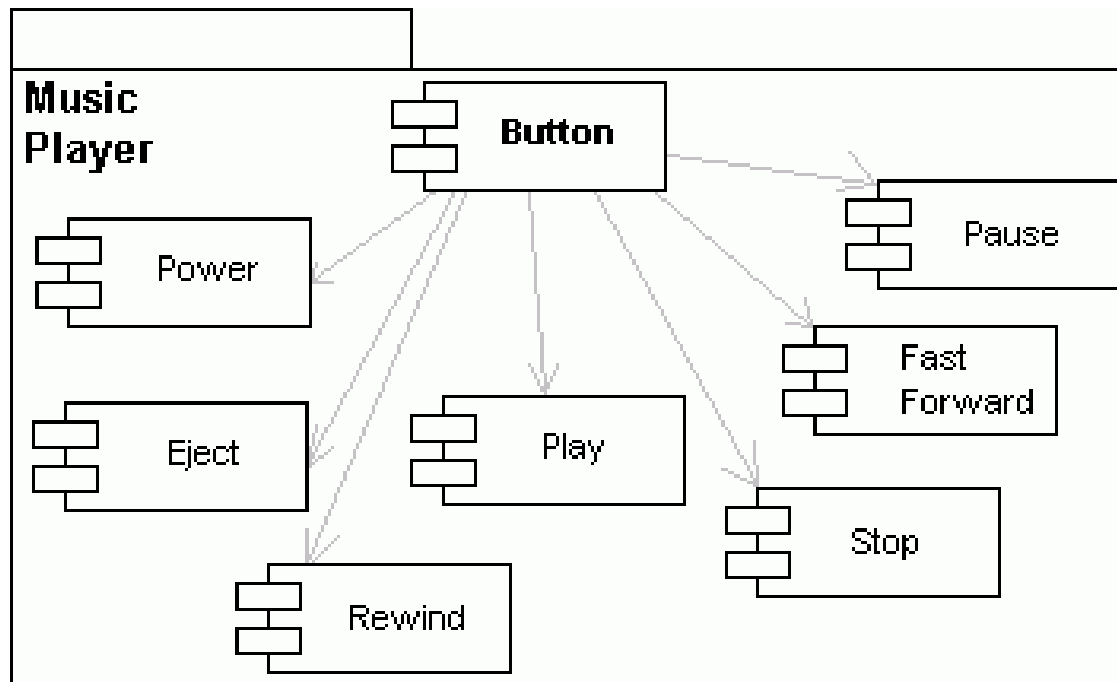
- play
- stop
- eject
- pause
- fast forward
- rewind
- power



Estos controles se implementaran como botones.

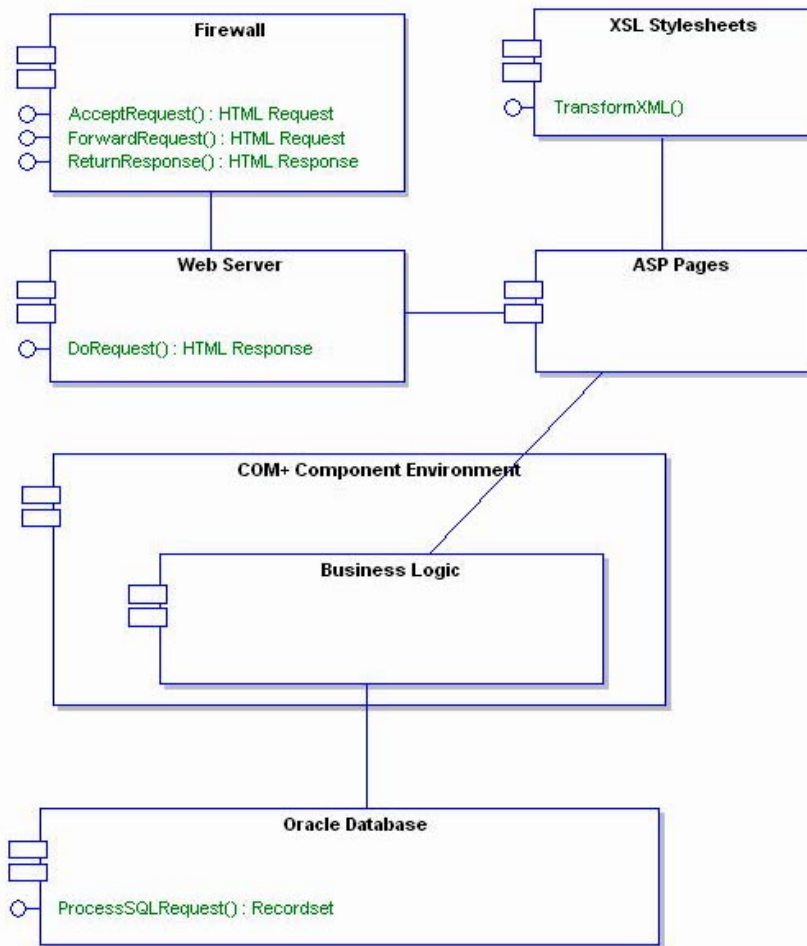
Ejemplo

Por tanto debemos tener un componente de tipo button que los modele. El diagrama de componentes nos queda de la siguiente forma:



Todos los componentes del diagrama anterior pertenecen al componente global (button) pero como realizan diferentes acciones son diferentes.

Ejemplo 2



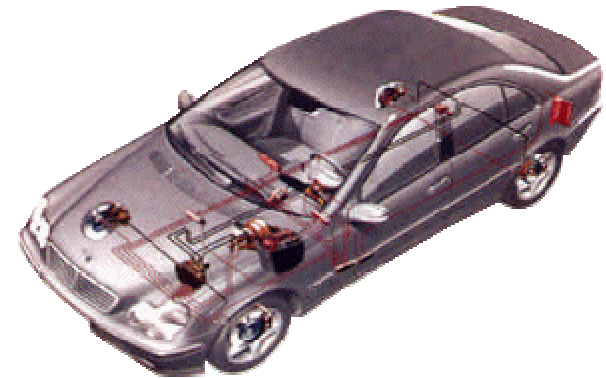
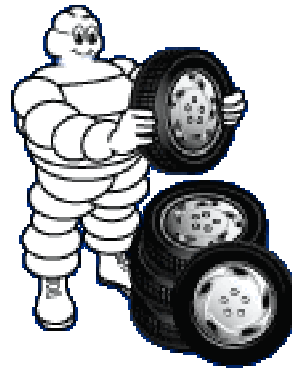
Diagramas de Despliegue (Vista de Implantación)

Contenido

Diagrama de despliegue

- Introducción
- Representación
- Resumen
- Ejemplos

Introducción



Introducción

Este tipo de diagrama muestra los componentes hardware existentes en un sistema.

- Muestra la disposición física de los recursos de ejecución tales como ordenadores y sus conexiones.
- Se muestran las relaciones físicas entre los componentes software y hardware para un sistema distribuido o empotrado.

Introducción

Un diagrama de despliegue esta formado por:

- **Nodos** (componentes hardware).
- **Conexiones** (relaciones de dependencia y asociación) entre los mismos que participan en la ejecución.

Modelizan la vista de implantación estática del sistema, incluyendo la topología del hardware del sistema.

Introducción

Nodo: una unidad de cómputo. Generalmente un elemento hardware con capacidad de proceso y almacenamiento.

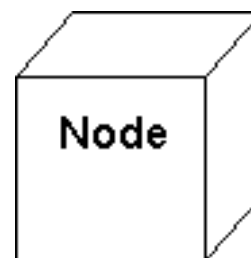
- Pueden contener objetos e instancias y, sobre todo, los componentes del sistema.

Representación

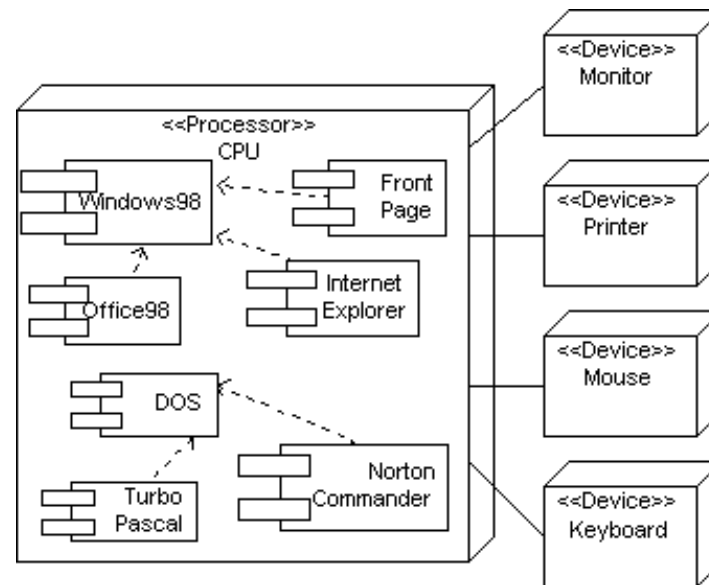
- Un nodo se representa de la siguiente forma:

Sintaxis:

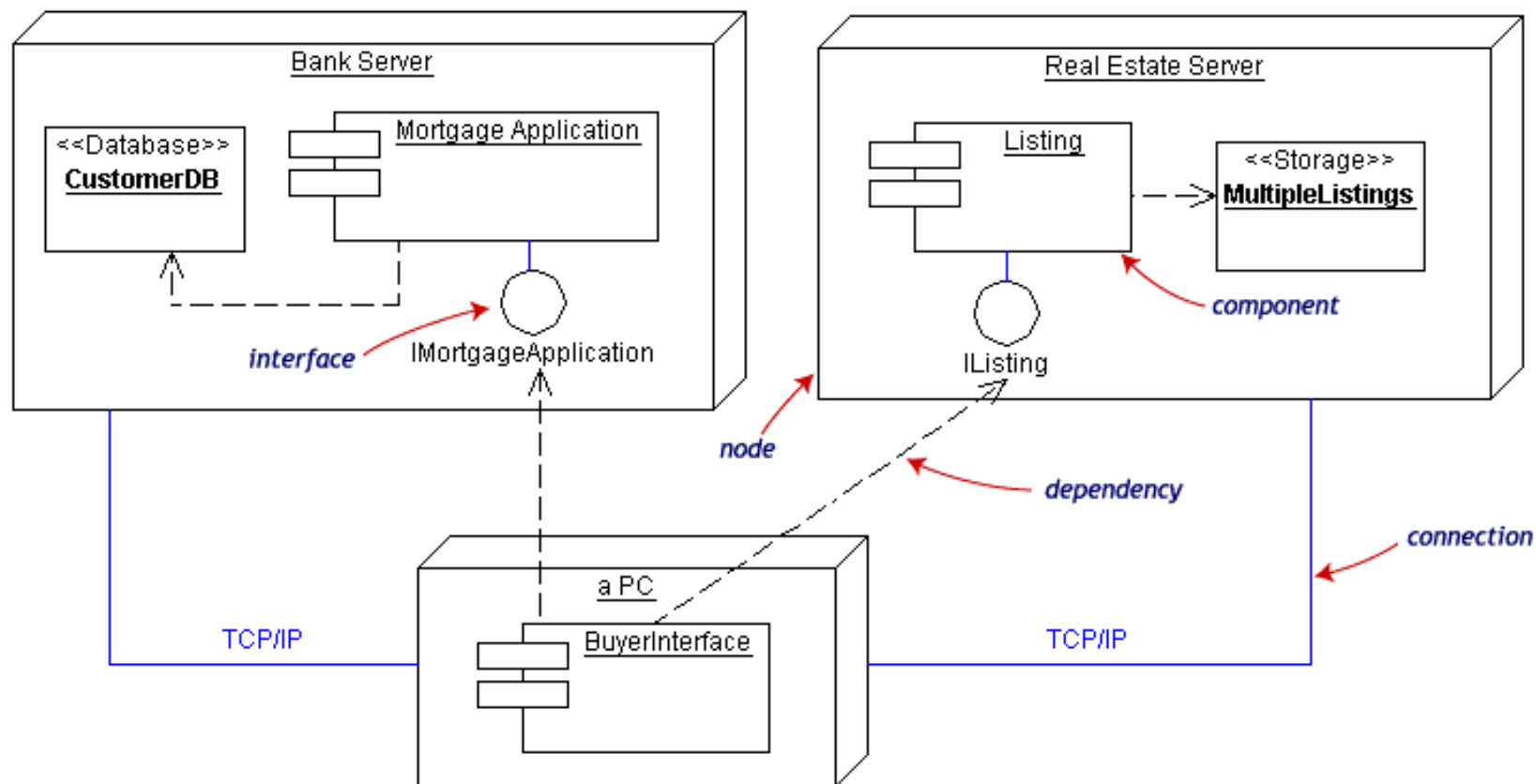
nombre del nodo : Tipo del nodo



- Las relaciones entre los nodos se representan de la siguiente forma:

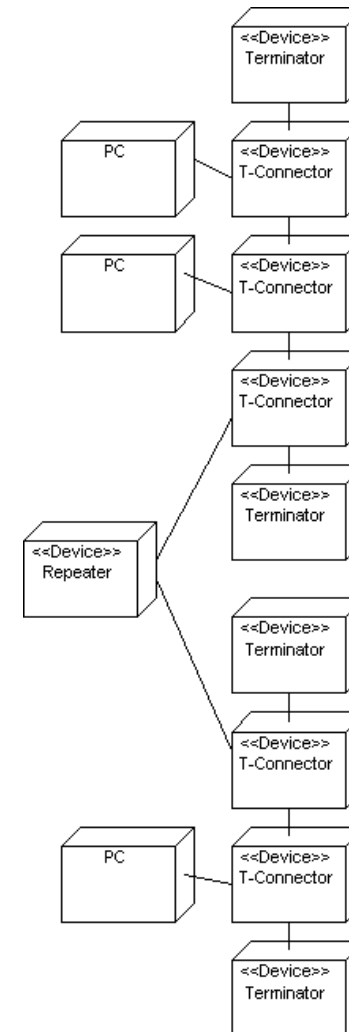


Resumen

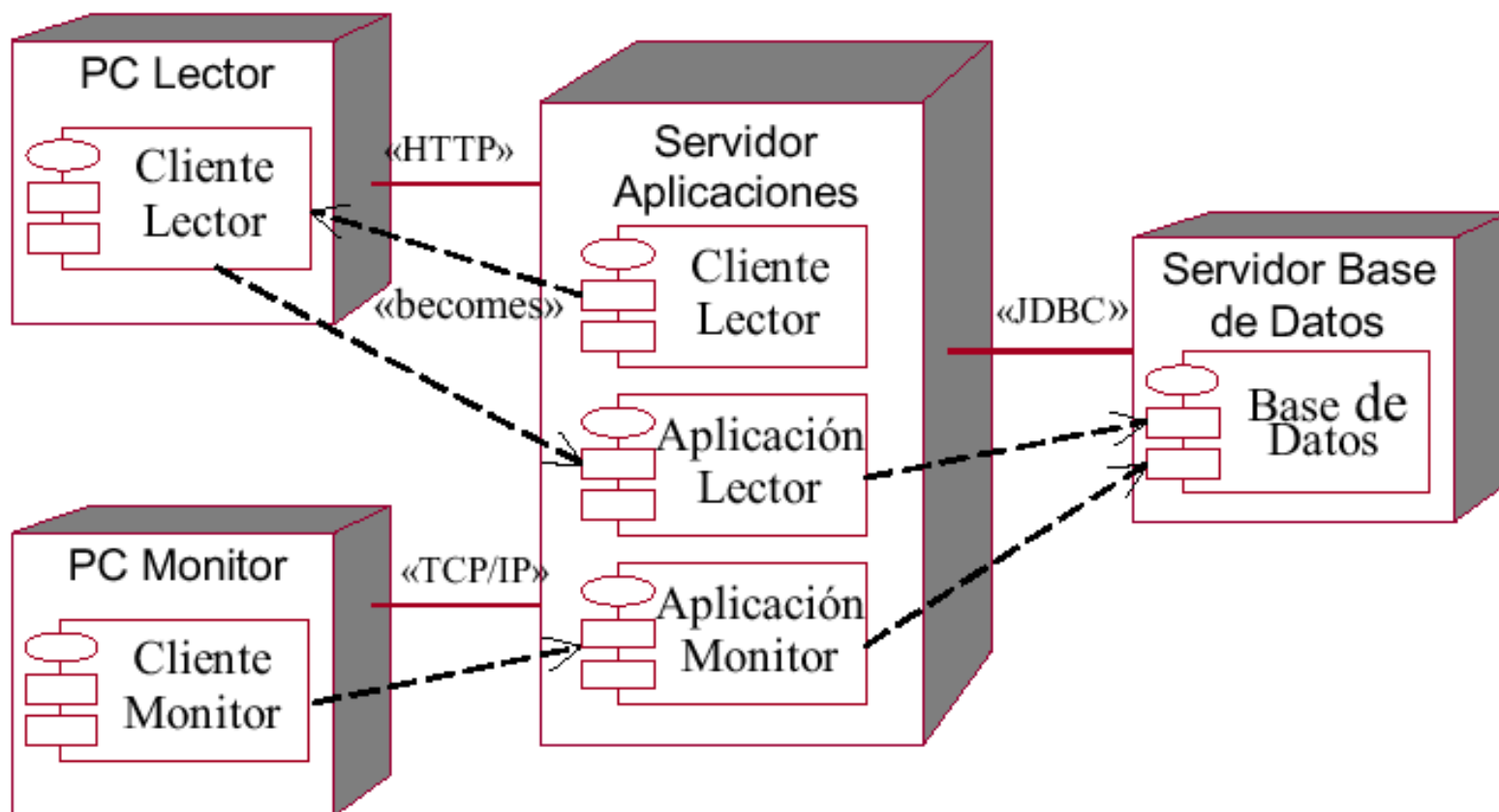


Ejemplo

El diagrama de despliegue correspondiente a este modelo es el siguiente:



Ejemplo 2



Conclusiones

Resumen

- Diagrama de clases:
 - Muestra las categorías de las cosas.
 - Proveen representación utilizada por los desarrolladores.
- Diagrama de objetos:
 - Muestra los objetos concretos.

Resumen

- Diagrama de casos de uso:
 - Descripción del comportamiento del sistema desde el punto de vista del usuario.
 - Desde el punto de vista de los desarrolladores es una herramienta muy útil ya que describe los requisitos del sistema desde la perspectiva de los usuarios.

Resumen

- Diagrama de estados:
 - Descripción del objeto en un estado particular.
 - Representan los estados y los cambios del objeto a lo largo del tiempo.

Resumen

- **Diagrama de actividad:**
 - Descripción de las actividades que suceden entre los diagramas de casos de uso o entre los objetos.
 - Representan la secuencia en que se producen estas actividades.
 - Es una extensión del diagrama de actividades, se centra en las actividades que se desarrollan.

Resumen

- **Diagrama de secuencias:**
 - Descripción de la interacción, a lo largo del tiempo, entre los objetos que componen el sistema.
 - Representan los estados y los cambios del objeto a lo largo del tiempo.
- **Diagrama de colaboración:**
 - Los elementos del sistema trabajan juntos para componer el sistema, este diagrama representa la colaboración entre los distintos objetos del sistema.
 - Muestran los mensajes que se pasan los objetos.

Resumen

- Diagrama de componentes:
 - Representa los componentes que forman el sistema y la relación entre ellos. Muchos de estos componentes han podido ser desarrollados por distintos grupos.

Resumen

- Diagrama de despliegue o implantación:
 - Muestra la arquitectura física del sistema.
 - Puede representar los computadores, sus conexiones con otros y el software existente en cada máquina.

Resumen

- Un **caso de uso** se puede detallar mediante:
 - Un Diagrama de Casos de Uso.
 - Un Diagrama de Máquina de Estados.
 - Un Diagrama de Actividades.
 - Un Diagrama de Interacción.
- Una **clase** se puede detallar mediante:
 - Un Diagrama de Clases. (parte estática)
 - Un Diagrama de Máquina de Estados. (comportamiento)

Resumen

- Un **estado** se puede detallar mediante:
 - Un Diagrama de Máquina de Estados (submáquina).
- Una **operación** se puede detallar mediante:
 - Un Diagrama de actividades.
 - Un Diagrama de Máquina de Estados.
- Una **actividad** se puede detallar mediante:
 - Un Diagrama de Actividades.

Resumen

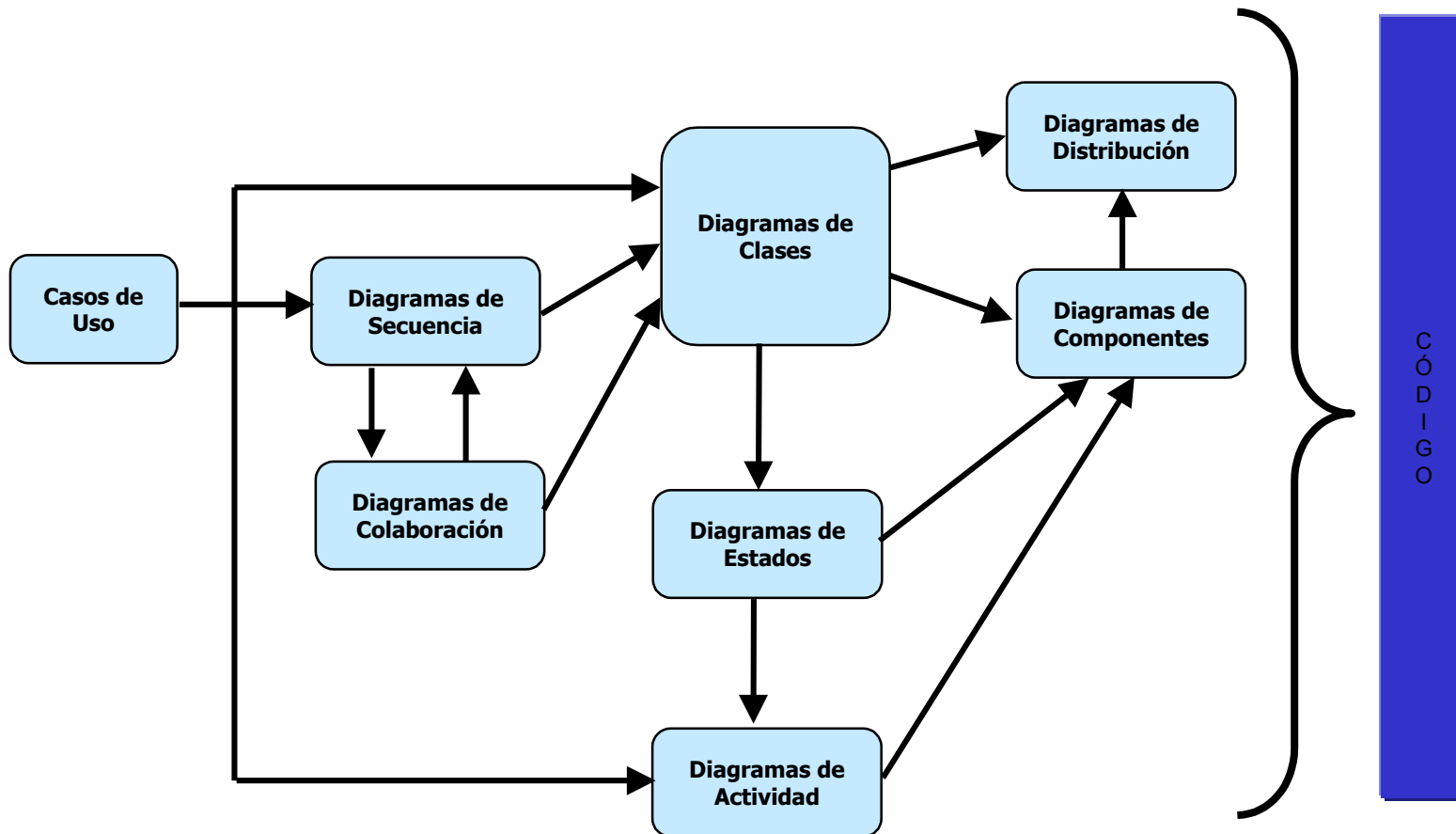
- Una **colaboración** se puede detallar mediante:
 - Un Diagrama de Interacción
 - Un Diagrama de Máquina de Estados.

- Un **proceso de negocio** se puede detallar mediante:
 - Un Diagrama de Actividades

Resumen

- Un **componente** de un Diagrama de Componentes incorpora la implementación de un grupo de clases
- Un **nodo** en un Diagrama de Despliegue contiene componentes.
- Un **paquete** contiene normalmente clases, pero puede agrupar cualesquiera elementos del modelo.

Relación entre diagramas



Ejemplo global

Pasos a seguir:

- **Recoger los requisitos.**
- **Análisis.**
- **Diseño.**
- **Desarrollo.**
- **Despliegue o Implantación.**

Pasos a seguir:

- **Recoger los requisitos.**
 - **El proceso del negocio.**
 - **El análisis del dominio.**
 - **Identificar el sistema.**
 - **Descubrir los requisitos del sistema.**
 - **Presentar los resultados al cliente.**

Pasos a seguir:

- **Análisis.**
 - **Comprender el uso del sistema.**
 - **Analizar cada caso de uso.**
 - **Refinar el diagrama de clases.**
 - **Analizar los cambios de estado en los objetos.**
 - **Definir las interacciones entre los objetos.**
 - **Analizar la integración con el sistema.**

Pasos a seguir:

- **Diseño.**
 - **Desarrollo y refinamiento de los diagramas de clases.**
 - **Desarrollo de los diagramas de componentes.**
 - **Planear el despliegue.**
 - **Diseño y prototipo de la interfaz de usuario.**
 - **Diseño de test.**
 - **Comienzo de la documentación.**

Pasos a seguir:

- **Desarrollo.**
 - **Generar el código.**
 - **Testar el código.**
 - **Construir la interfaz de usuario y conectarla con el código y los test.**
 - **Completar la documentación.**

- **Despliegue o Implantación.**
 - **Planificación de copias de seguridad y recuperación.**
 - **Instalación del sistema final en el hardware apropiado.**
 - **Testar el sistema instalado.**

Contenido

- **1ª Parte**
 - Recogida de los requisitos
 - Análisis

- **2ª Parte**
 - Análisis
 - Diseño
 - Desarrollo

Ejemplo global

Vamos a realizar una práctica global. La idea va a ser modelar una biblioteca digital. La información la obtendremos a través de la entrevista con el cliente y de ahí iremos viendo como se desarrolla todo el modelado del sistema.

- La estructura clásica de una biblioteca es un edificio con un bibliotecario y con una clasificación propia. Para coger un libro tenías que estar físicamente allí. Si el libro no se encontraba y querías algo similar dependías del acierto del bibliotecario.
- Internet ha cambiado el mundo y actualmente existen varias bibliotecas digitales. Este tipo de bibliotecas es accesible a través de la red, por tanto todo el mundo puede acceder a ellas. La clasificación de libros es mucho mayor que en una biblioteca normal y por tanto, se pueden encontrar libros similares, muy rápidamente.

Empezaremos a modelar la Biblioteca Digital a través de una entrevista con el cliente.

Contenido (1ª parte)

- **Modeling of Digital Library**
- Discovering Business Processes
- Domain Analysis
- JAD Session

El proceso del negocio

In the following text *A* will stand for Analyst and *C* for the client.

- *A* Let's start with the work. Tell me what's happening when a customer enters in your library?
- *C* He or she goes to the corridor where data search can be done. After selection of which books or journals to lend, the customer goes to the place where he or she can take the sample of that book or journal.
- *A* You said data search. What kind of data?
- *C* There is main data record for every book, author or journal, sorted alphabetically. Every record consist of different data according to type of the product: (book or journal). These records are printed on paper cards, and they are putted on different folders. Access to these records can be done also by computer terminals which can be found in the corridor.

El proceso del negocio

- *A* That means that your library is computerized. Why using a computers and a paper records?
- *C* As we live in hi-technological era, we must to follow time steps. Computers were introduced few years ago, but paper records are here since the library was built. Also some impatient customers, can use them if all of the computer terminals are occupied.
- *A* Are the same data put in the paper records and in the computer records?
- *C* Yes. All of old data from paper records is converted to computer records, and new data that we write in paper cards also is written in electronic format.
- *A* Where is your database located on?
- *C* Our database is located on a computer server. This server is a heart of our local area network - all of the terminals located in our library.

El proceso del negocio

- **A** Let's backtrack to the data. You said that same data is stored in the electronic format and in the paper format. Next in our conversation we can refer to these with same name, let's say index.
- **C** I agree.
- **A** When we talk about index, what kind of information is necessary to perform a search?
- **C** Data for the books and for the journals is very similar. Both have ISBN-the unique number for every book/journal, title, publishing year, publishing house, but books have author and genre, and journals have volume and area. For the data search are useful ISBNs, titles, authors, genres, volumes and areas. Retrieved results specifies the type of the books or journals.
- **A** Just a second. What are the kinds of books and journals?
- **C** Except paper books and journals, electronic books and journals are supported in our library.

El proceso del negocio

- **A** Electronic books and journals. How do you manage these types of items (books / journals)?
- **C** The server we had is used for storage of these types of items. We produced some of the electronic items, but also in our database we have Internet addresses from electronic books or journals from other libraries we used to cooperate.
- **A** You have a lot of amount of data. Is everyone concerned with data input and data maintenance?
- **C** We have employees to do this.

El proceso del negocio

- **A** You said employees. I think that they are not parts from this conversation for now. We will talk about them later. The customers are not finished yet. Before going to the process of lending let's assume what we have at this point of interview, things come clearer to me now. Every customer can perform a data search for the books or journals. Electronic formats of books retrieved from the search, can be accessed immediately. Paper formats can be lent if there is a free sample. Now you may go to the process of lending.
- **C** At the beginning, I've stress out that user who is in the library after data search can perform a lending. Lending can be done if books or journals are in paper format, and it's done in the lending area in the library. Searched books or journals can be given to the user if free samples of them are in the library.

El proceso del negocio

- **A** Let's look at the both cases - present and not present sample of the book/journal. What is the flow of events?
- **C** When sample is not present, the user can make a reservation for it. After that library must inform the user for a free sample, when one is set free. User, than should go the library and make a lending for that sample. I must say that free sample is kept for the user few weeks after informing it.
- **A** What if there is a free sample?
- **C** In that case employee who is concerned with lending (let's say - lender), must check the users data to see if user is a library member or not. If the user is a library member, he or she takes the ISBN of lent books or journals and personal data for the user, and writes them in the lending record. Also a date is written in this record, which is stored in the lending database. Every user has a limit of books/journals to lend, and also there is a limit for time keeping of books/journals.

El proceso del negocio

- **A** Do you allow employees to lend books or journals?
- **C** Yes we do. Every employee in our library has the same record as members. That we have lending records for employees too.
- **A** You said that lending records are stored in a lending database. Is this a work of "lender" or someone else do this?
- **C** Our local network doesn't consists only from search terminals. A "lender" also has a terminal from which he or she can check for user data and free samples. Also a quick reports can be done to see which users have books to return.

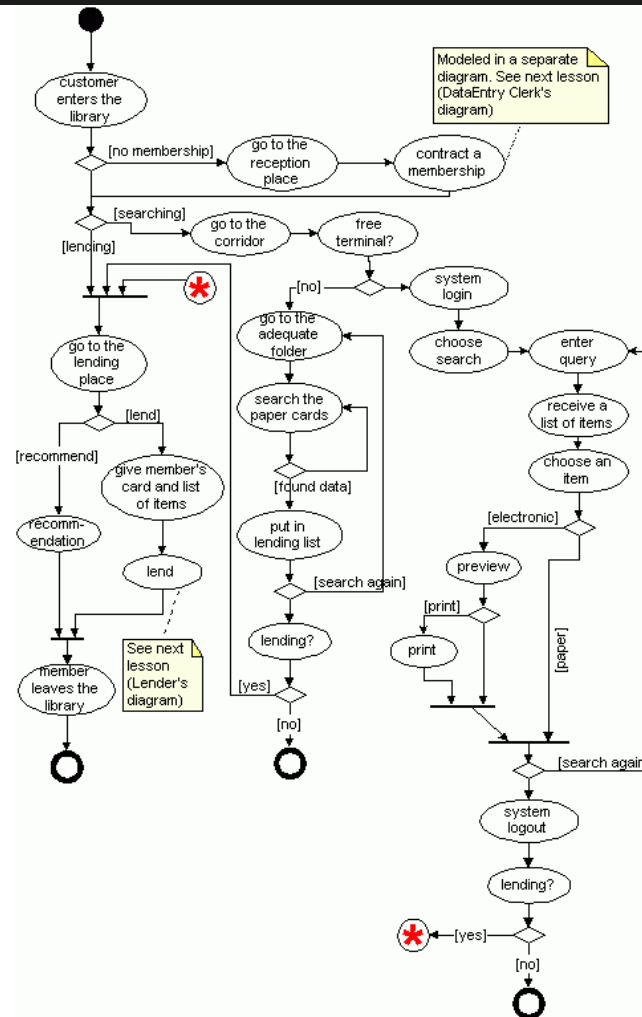
El proceso del negocio

- **A** Wait a minute. Our conversation gone in other direction. We started to talk about employee's duties, we didn't finished with the customers. Let's see to made a lending a free sample of the book or journal can be present in the library, and a lending record is produced after it.
- **C** That's it. I'll stress again that only paper formats can be lent, because they have a samples. Electronic formats of books and journal doesn't have a samples. In other words they have infinite number of samples, or they are present in the library everytime.

El proceso del negocio

- **A** Is it possible to have no present samples of books or journals from searched user's list? Can users ask a lender to recommend them a book or journal?
- **C** Certainly lender can take a brief look at the user's history of lent books and journals, and if he or she had read a books or journals from the user's interest areas, then he or she can recommend something to the user. User can accept that or not.
- **A** This will be all for now. I think that user's diagram is done, and we can take a look at it:

El proceso del negocio



El proceso del negocio

Now we'll give you the interview with the client - explanation of work of the employees in the library. Let's see at all given details and analyze the produces activity diagram.

- **A** In previous interview we talked about customers, and all the activities related to them, like lending, reservation, recommendation, and search. In lending and recommendation process interaction with the employees was introduced. Let's turn to the employees now, and talk about employees which interact with clients, books and journals.
- **C** Allright. The lenders were introduced in previous interview, and we see that their work is to take care of lending and returning of books and journals. They post reports to the members who have books to return, and members who have reserved books and their samples are present in the library.

El proceso del negocio

- *A* Is it all for the lenders?
- *C* Yes.
- *A* When we talk about a customers, we mentioned that for huge amount of data, some of the employees are employed. Can you tell me more about them?
- *C* After buying a computers, a part of our old employees, involved in process of writing paper cards, were trained to work with computers. These DataEntry clerks now operate with some of terminals for data entering and maintenance. Also for maintenance of our databases DatabaseSpecialists were employed.

El proceso del negocio

- *A* If I want to become a member to your library. What I'm supposed to do?
- *C* First of all every new member must go to the reception place to conclude his or her membership. Employee that enters new memberships, opens a new record with a name and last name of the member, address and place of living and Personal Registration Number added to it. Also telephone e-mail and date of birth are stored in this record. After registration, a member's card is given to the member, and a paper record is printed for the library's documentation.
- *A* How do you make data input of books and journals?
- *C* Books and journals have a plenty of attributes in common. That are ISBN, title, year of publishing and publishing house. Also every item has a number of samples. Sometimes a multimedia data are supported with the item. If that is a propaganda material in paper form we convert it into electric form and put it into the database.

El proceso del negocio

- *A* Do you support multimedia data only for electronic items, or paper items has multimedia too?
- *C* The form of the book/journal is not important. Both for paper and electronic formats we support multimedia data. Popular books in the world, can be more popular if a multimedia is supported for them. Every user is curious to meet with these types of information.
- *A* If I perform a data search, can I browse the multimedia data supported for the item? And what kind of multimedia is supported?
- *C* It's true. During computer's searching in returned results if an item has a multimedia data, user can browse that data. Types of multimedia which are supported are Text, Audio, Video and Picture.

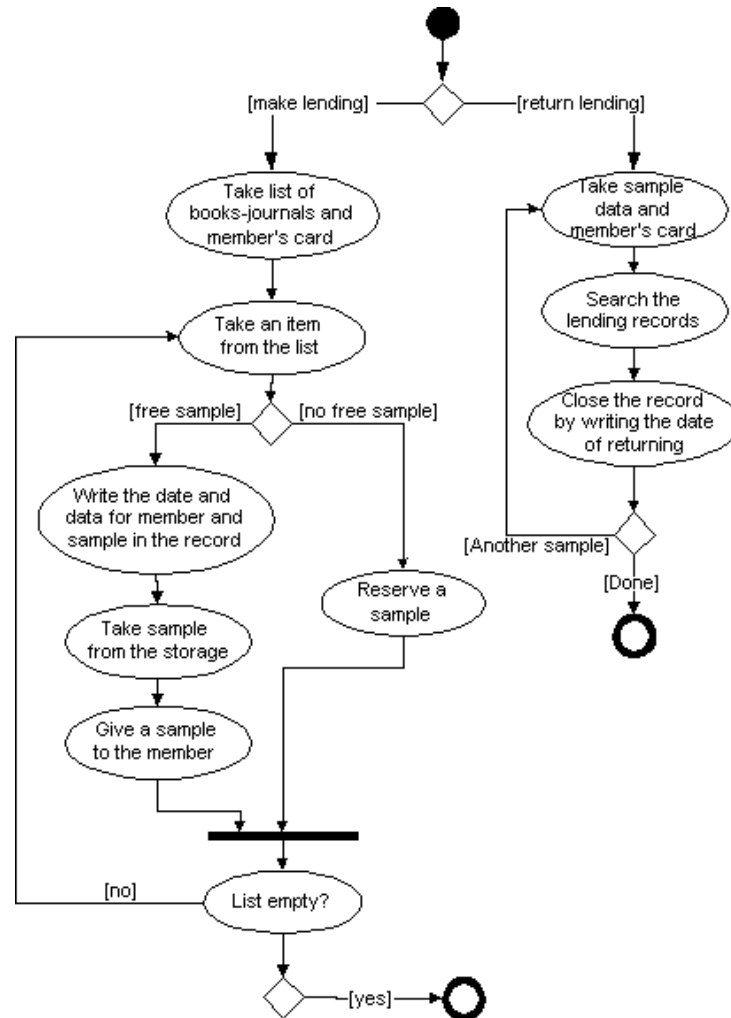
El proceso del negocio

- *A* How do you enters a data for the book's author?
- *C* I'm sorry. I forgot to tell you that authors are entered separately. For the authors name and last name, year and place of birth are inputted. During the process of book's input if the book's author is in the entered list of authors, then he or she is chosen from that list. Otherway we used an author creation process. Also for books their genre is inputted.
- *A* What about a journals?
- *C* For the journals volume and area are entered.
- *A* You said that a parallel evidence is made (paper and electronic). How do you represent multimedia data in your paper cards?
- *C* Multimedia data is entered only in electronic format. I said that paper cards are printed from electronic records, but only necessary information for cards is printed.

El proceso del negocio

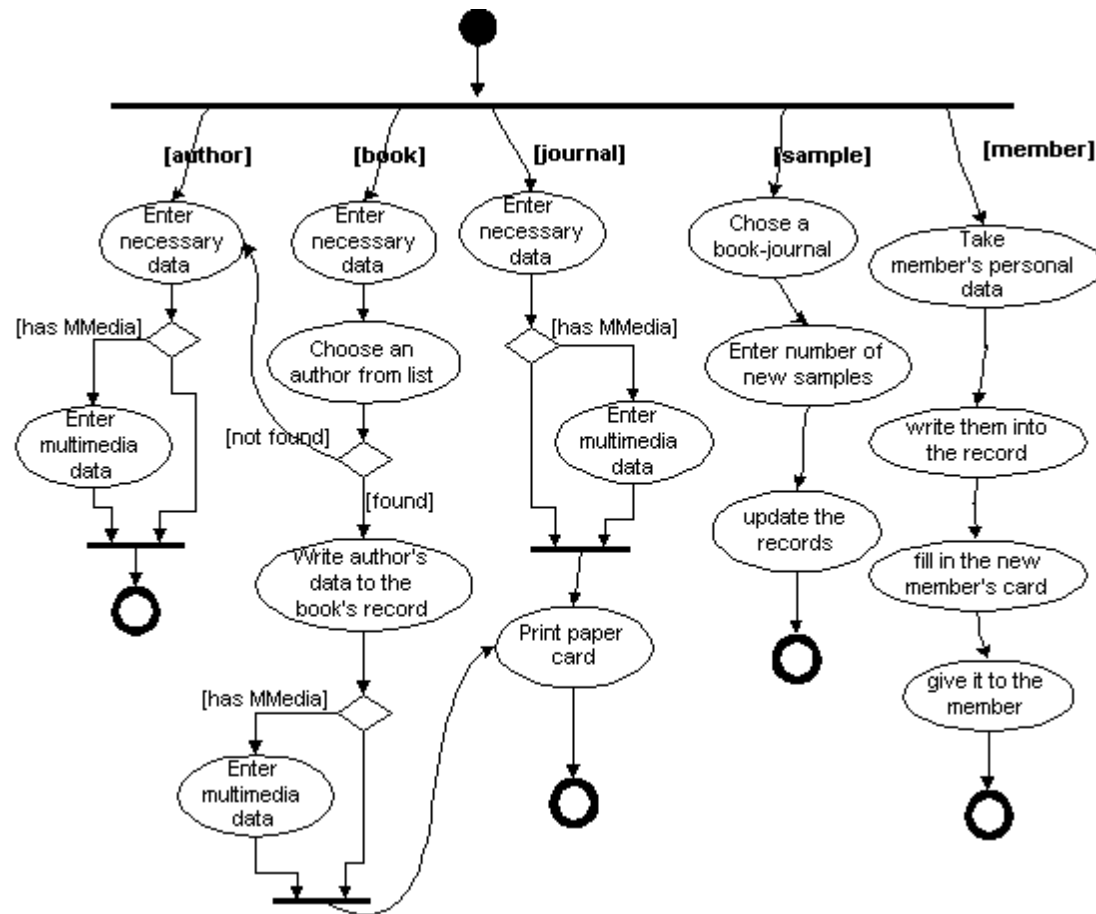
- *A* We examined a work of lenders, employees who enters a data for authors, books, journals and members. But I don't see here a sample input. In lending process you said that free samples can be lent. How do you perform sample input?
- *C* I noticed when we talk about book entering, that process of book's input also consists of inputting a number of new samples. But it is possible new samples to arrive in library, and a record for the book to exist in database. In that case only sample data is entered. This data consists of sample's signature, which only concerns the lenders (information about stored place of sample in the storage room).
- *A* Do you think that we have finished the employee's analysis?
- *C* Sure. We reached the end. I hope that all my spelling out all the steps in library work, makes clearer view to you about libraries like ours.

El proceso del negocio



El proceso del negocio

- *The Data Entry Clerk's Activity diagram extracted from the interview*



Análisis del dominio

Before having additional interviews, the development team first must work within the context of the business-process interview. This is the job of an *object modeller*. The objective is to produce initial class diagram.

The object modeller looks for nouns, verbs and verb phrases. Some of the nouns will become classes in the model, some will become attributes. The verb and verb phrases can become operations or the labels of associations.

Análisis del dominio

corridor, data record, storage folder, local network, computer server, computer terminal, paper book, paper journal, electronic book, electronic journal, item, date, limit of time, limit of copies, book, journal, sample, author, paper card, library, database, ISBN, title, publishing year, genre, publishing house, volume, area, reservation, employee, lending, lending area, lender, user, member, lending record, report, number of samples, DataEntry clerk, DatabaseSpecialist, reception place, name, lastname, address, place of living, PersonalRegistrationNumber, telephone, e-mail, date of birth, member's card, library's documentation, multimedia data, information, text, audio, video, picture, signature

- ¿Cuales de los nombres anteriores pueden ser eliminados como atributos de las clases? ¿Cuales son sinónimos? ¿Cuales son los que representan las clases en nuestro modelo?

Análisis del dominio

corridor, data record, storage folder, local network, computer server, computer terminal, paper book, paper journal, electronic book, electronic journal, item, date, limit of time, limit of copies, book, journal, sample, author, paper card, library, database, ISBN, title, publishing year, genre, publishing house, volume, area, reservation, employee, lending, lending area, lender, user, member, lending record, report, number of samples, DataEntry clerk, DatabaseSpecialist, reception place, name, lastname, address, place of living, PersonalRegistrationNumber, telephone, e-mail, date of birth, member's card, library's documentation, multimedia data, information, text, audio, video, picture, signature

- En rojo los atributos.
- En negro los nombres de las posibles clases.
- En amarillos sinónimos.

Análisis del dominio

- And these verbs:

go, search, lend, take, sort, print, put, do, has, have, convert, write, manage, access, read, preview, take a sample, reserve, give, inform, make, set free, check the data, recommend, post, buy, operate, maintain, conclude, browse, input

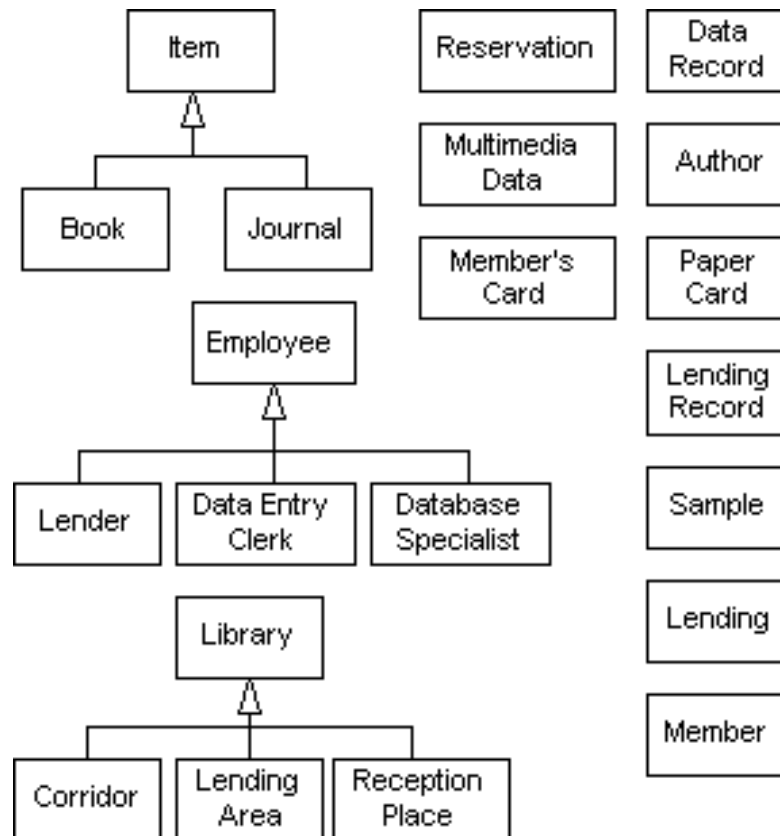
Análisis del dominio

Now we'll try to form some meaningful groups of nouns.

- One group consists of *people*: member, author, employee, lender, DatabaseSpecialist, DataEntry clerk. This group could stand some subdivision because everyone except the member and author are employees.
- Another group consists of *library's items* like books, journals, samples, reservation, lending.
- Third group consists of *areas within the library*: corridor, lending area, reception place.
- In forth group may be these nouns: member's card, data record, paper card, multimedia data, lending record.

Análisis del dominio

- This is the initial class diagram, next we'll discover associations between classes and produce more descriptive class diagram than this one.



Formando asociaciones

Now we'll create and label associations between some of the classes. The verbs and verb phrases can help us with the labelling, but we won't limit ourselves to the ones from the interview.

- Labels that are somewhat more descriptive might suggest themselves.

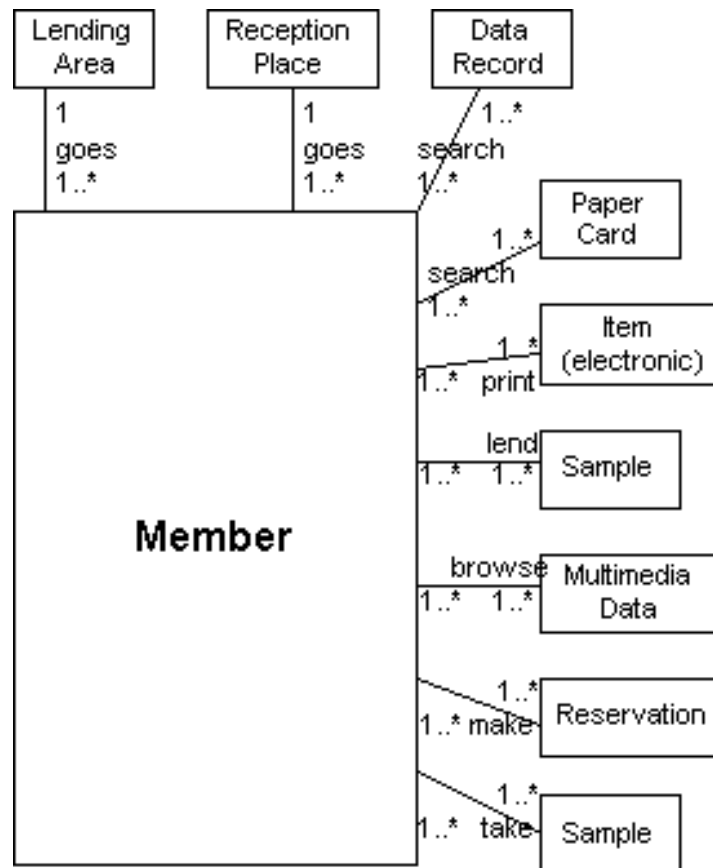
Formando asociaciones

Let's start with the member first. Let's label the associations by generating phrases that characterize the associations. Here are some phrases that immediately come to mind:

- The member searches the paper cards or data records.
- The member goes to the lending place.
- The member lends a sample.
- The member makes a reservation.
- The members takes a sample.
- The member browse the multimedia data.
- The member goes to the reception area.
- The member prints the electronic item.

Formando asociaciones

When we labeled out the associations, we can put multiplicities into associations lines. After this customer's class will look like this:



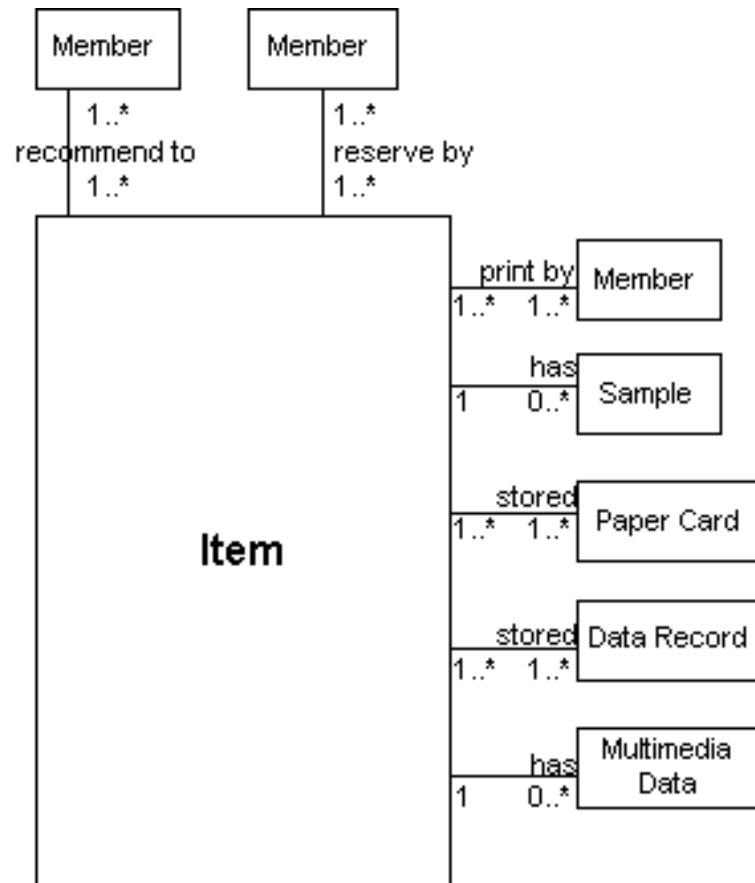
Formando asociaciones

The associations for the item's (here item is standing for both book and journal) class might be the phrases from the following list:

- The item can be recommended to the member.
- The item is reserved by a member.
- The item (electronic) is printed by a member.
- The items are sorted alphabetically in paper cards and data records.
- The item has sample(s).
- The item has multimedia data.

Formando asociaciones

After filling out associations names and multiplicities, the item's class will be:



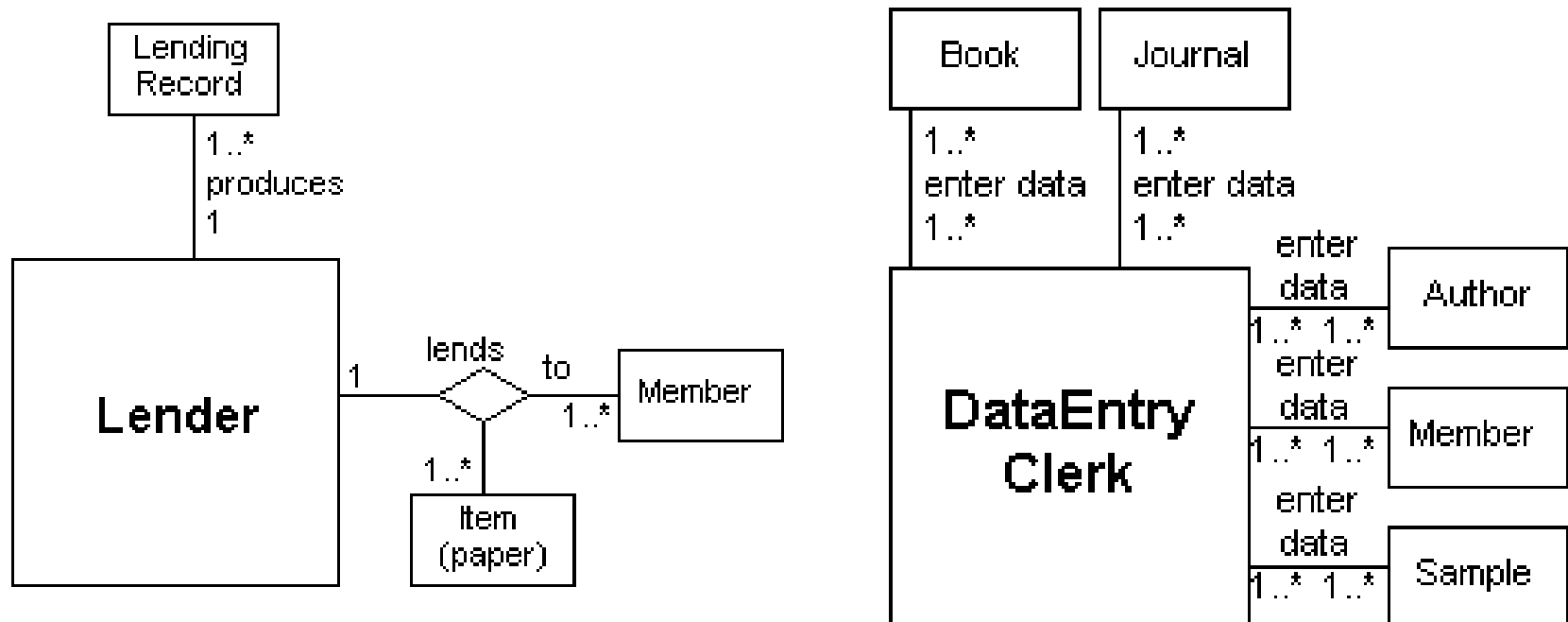
Formando asociaciones

Interesting, also is the process of forming associations and multiplicities for the employee's class. Bear in mind that employees are also a members of the library, thus all the member's class associations can be putted into the employee's class, but some of the employees perform some actions that members are not allowed:

- The lender produces a lending record.
- The lender lends an item to the member.
- The DataEntry clerk enters data for the books.
- The DataEntry clerk enters data for the journals.
- The DataEntry clerk enters data for the authors.
- The DataEntry clerk enters data for the members.
- The DataEntry clerk enters data for the samples.

Formando asociaciones

After filling out associations names and multiplicities, the lender's and DataEntry clerk's classes will be:

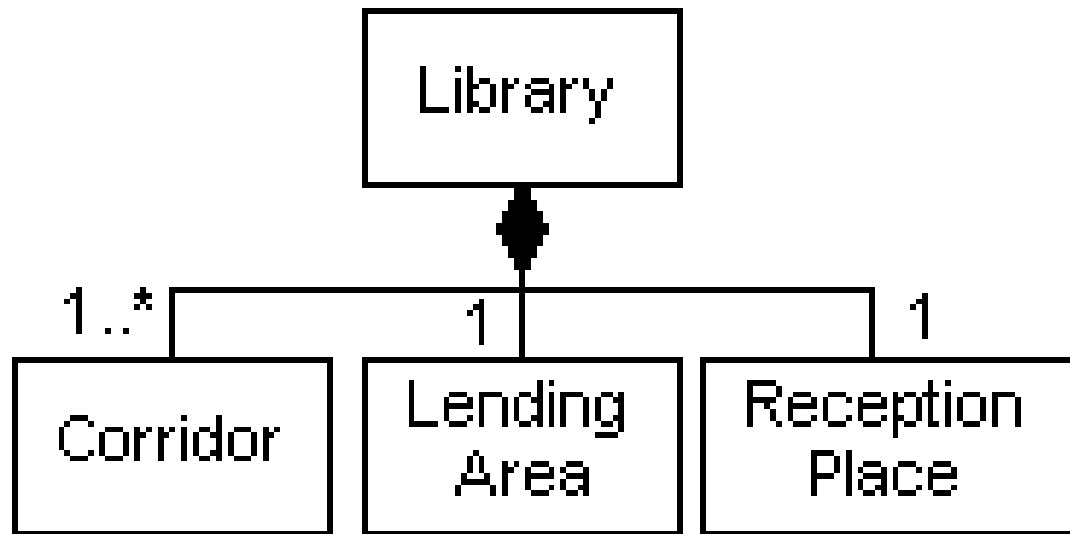


Formando asociaciones

Try drawing out the associations with the Author's class and Sample's class.

Formando asociaciones

We've been forming and naming abstract classes and associations, and another organizational dimension awaits. The next step will be finding out classes that are components of other classes. A library for instance consists of a set of parts, but for as interesting will be: corridor, lending area and reception area. Next picture shows this composition.



Descubriendo las clases

We can begin the refinement now by adding some attributes and operations.
Our most important classes appear to be Member, Author, Employee, Book, Journal, and Sample.

Descubriendo las clases

The Member

- In nouns list some of the nouns are appropriate to be used as member's attributes. And from verbs list we can chose some of the possible operations for the member. These two lists are given in the following table:

List of attributes	List of operations
<i>name</i>	<i>newMember()</i>
<i>lastname</i>	
<i>address</i>	<i>search()</i>
<i>date of birth</i>	
<i>place of living</i>	<i>lend()</i>
<i>PersonalRegistrationNumber</i>	
<i>telephone</i>	<i>reserve()</i>
<i>e-mail</i>	
<i>memberID</i>	

Descubriendo las clases

The Member

- The member class now will look as:

Member
name lastname address dateOfBirth placeOfLiving PersonalRegistrationNumber telephone e-mail memberID
newMember() search() reserve() lend()

Descubriendo las clases

The Author

- Lists of attributes and operations for the author will be:

List of attributes

name
lastname
date of birth

List of operations

createAuthor()
bookReference()

Descubriendo las clases

The Author

- The author class now will look as:



Descubriendo las clases

The Employee

- Lender and the DataEntry clerk are children of the abstract class Employee. First we'll assign attributes to the Employee class and the children classes will inherit them. Also we must bear in mind that every employee has a member record. This will guide us that in the Employee class list of member's attributes may appear, plus an attribute to differ employees from members. Also DataEntry clerk class will have new attribute which determines the type of data entry performing that clerk. List of attributes and operations (for Lender and DataEntry clerk) are given in following table:

Descubriendo las clases

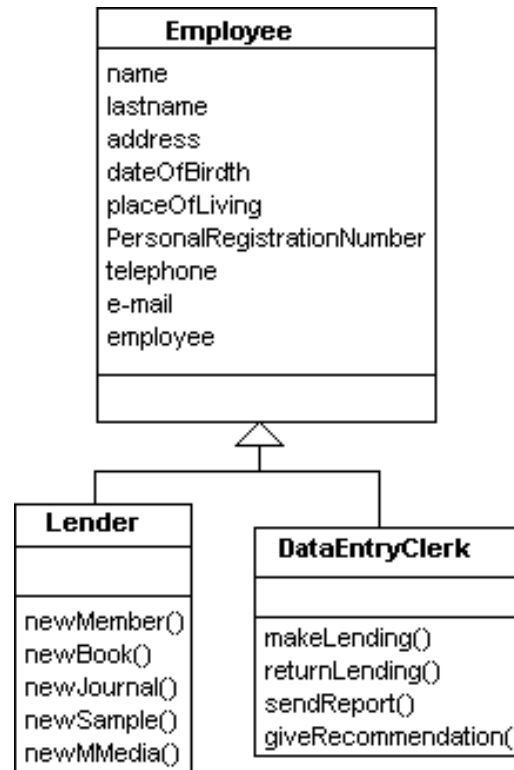
The Employee

List of attributes for the Employee class	List of operations for the Lender class	List of operations for the DataEntryClerk class
<i>name</i>		<i>newMember()</i>
<i>lastname</i>	<i>makeLending()</i>	
<i>address</i>		<i>newBook()</i>
<i>date of birth</i>		<i>newJournal()</i>
<i>place of living</i>	<i>returnLending()</i>	
<i>personalRegistrationNumber</i>		<i>newSample()</i>
<i>telephone</i>		<i>newMultimedia()</i>
<i>e-mail</i>	<i>sendReport()</i>	
<i>memberID</i>		
<i>employee</i>		

Descubriendo las clases

The Employee

- The structure of the Employee class with its subclasses is given on the following picture:



Descubriendo las clases

Try finding out an attributes and operations for the Book class.

Descubriendo las clases

The Journal

- List of attributes and operations for the Journal class are following:

List of attributes

title
volume
area
publishing year
publishing house
ISBN
status
numberOfSamples

List of operations

freeSamples()
newJournal()
makeLending()
makeReservation()

Descubriendo las clases

The Journal

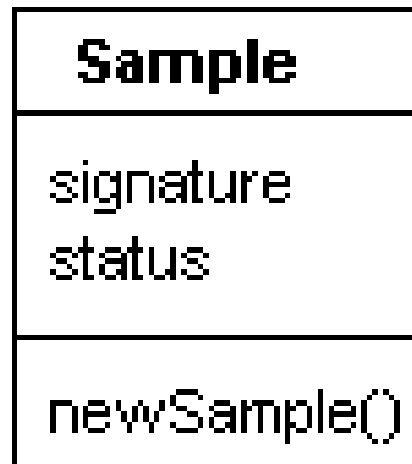
- The "status" attribute indicates the type of the journal: electronic or paper. The Journal class will be:

Journal
title volume area publishingYear publishingHouse ISBN numberOfSamples
freeSamples() newJournal() makeLending() makeReservation()

Descubriendo las clases

The Sample

- The lending process may produce lent book or journal only if a free sample of that book or journal is present in the library. For that purpose a Sample class must be present. The sample class will have only two attributes (in this moment) - *signature* and *status* (lent or free), and one operation *newSample()*. The Sample class is present on next picture:



Descubriendo las clases

When you're putting together interview results, business processes, and domain analyses, keep a model dictionary. This is a glossary of all the terminology in the model. It will help you maintain consistency and avoid ambiguity.

Also it's not a good idea to have every detail of your class model in one huge diagram. You'll need a master diagram that shows all the connections, associations and generalizations, but it's the best to elide attributes and operations from this picture. You can turn the spotlight on selected classes by putting them in separate diagrams.

Desarrollando la visión

Now it's time for the team to work on the technical backbone for the Digital Library. They've got business processes and class diagrams. Can they begin with the coding now, or not? They're not even close to writing a program, first, they have to develop a vision of the system.

- The team will use its business process knowledge and newly acquired domain knowledge to see where infusion of technology enhances the library work. The players in this conversation are an analyst, a modeller, a librarian, a lender, a DataEntry clerk, a member and a system engineer. A facilitator runs the meeting. The facilitator distributes copies of the business process diagrams developed in previous lessons.

Desarrollando la visión

This conversation is about information movement. Some of the processes in the library business depends on the movement of the information. If we can speed that movement along - something technology is really good at - we'll meet our goal. From this conversation was concluded that information movement takes place when:

- The member lends a book's sample
- The member lends a journal's sample
- The lender recommends a book to the member
- The lender recommends a journal to the member
- The member contracts the membership with the DataEntry clerk

Desarrollando la visión

Next will be given a part from this conversation. Here modeler has some interesting ideas about adding new attributes in the class diagram. Let's see what is happening:

- **Analyst** Let's go to the process of lending. It was mentioned that members has time limit and copies limit for a lending. How the lender knows how many books or journals to give to the member?
- **Lender** First of all I take a look at member's lending record, and count how many items he or she has lent. Also I check the lending date to see if there is a books or journals with overloaded time limit.

Desarrollando la visión

- **Modeler** I've been working on my class diagrams while I've been listening to all of you. And I have some questions to ask. In the Member class, we may add new attributes like timeLimit and copiesLimit which are common for every member. Also an attribute that holds the number of lent samples - samplesLent may be introduced, and a derived attribute - numberToLend, which is the difference between copiesLimit and samplesLent.
- **Lender** That's a nice idea. Then I'd know the number of allowed samples to lend to the member.
- **Analyst** What about timeLimit?
- **Modeler** We have an attribute that holds the lending time. If we add timeLimit to it, a result is aimed. Then if the result is greater than current date, we can inform the member that he or she has a time pass over for a lent book or journal.

Desarrollando la visión

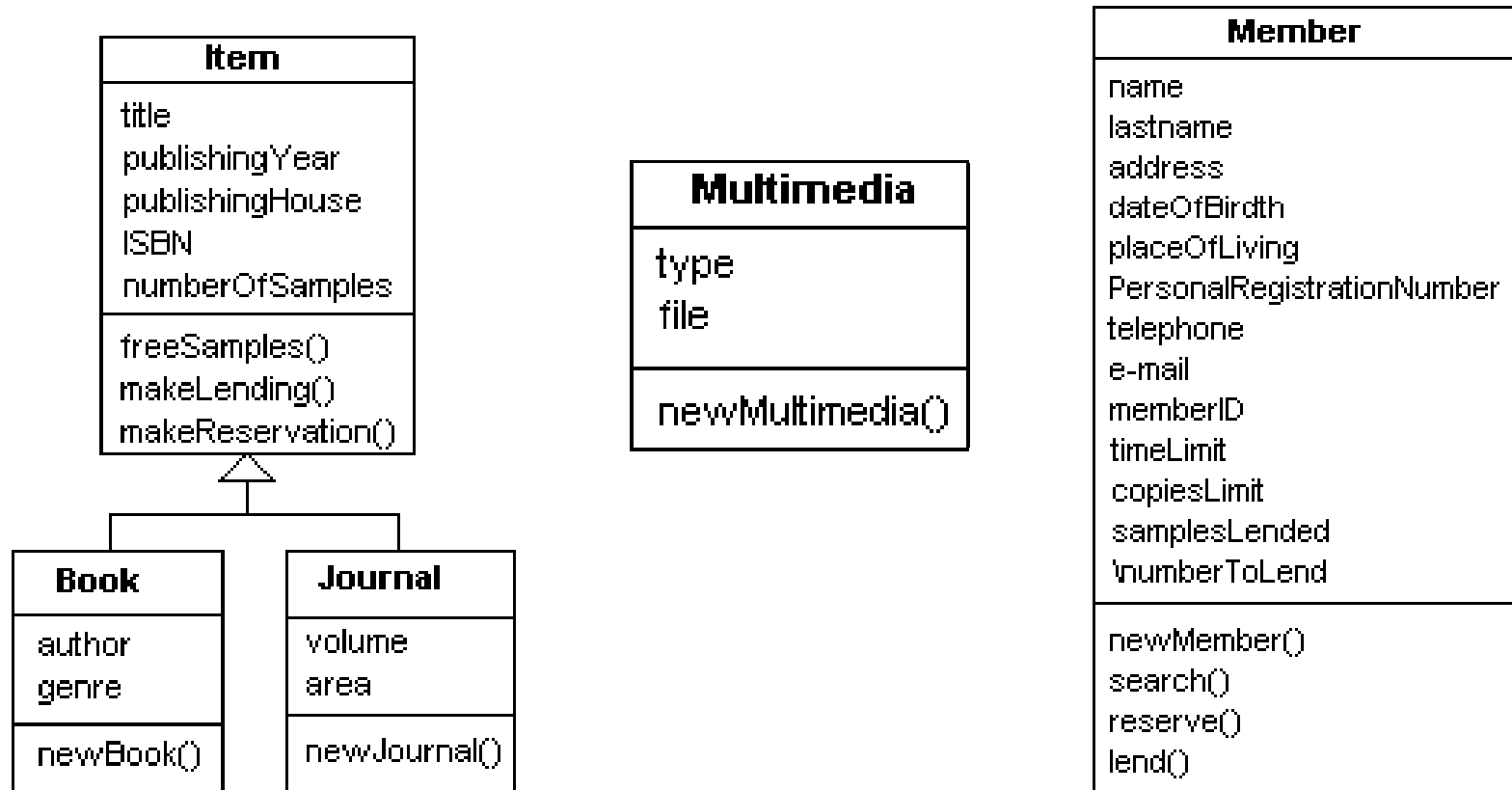
- **Analyst** What about multimedia data? It was said that for some of the books, journals or authors multimedia data is supported.
- **Modeler** It's a perfect time to introduce a new class in our class diagram - Multimedia. For that purpose, Multimedia class will have two attributes *type* (text, audio, video or picture) and *file* (physical location of the multimedia file) and one operation - *newMultimedia()*.
- **Facilitator** Analyst are you satisfied?
- **Analyst** I agree. Now things comes clearer to me. Every part of the puzzle is putted on its place.

Desarrollando la visión

- **Modeler** Here's another possibility. Look carefully at the Book and the Journal classes. Both are having common attributes like ISBN, title, publishing year, publishing house, number of samples. They are differing in author and genre data for the book, and volume and area data for the journal. This is perfect situation for a new inheritance. Common attributes can be putted in new parent class, let's say **Item**. Also in this new class some of the operations can take place like: `freeSamples()`, `makeLending()` and `makeReservation()`. With this we can save memory space in our database.
- **Facilitator** I'm glad that we're making some optimization. Before going to the next steps, san I suggest you to look at new classes?
- (All agree)

Desarrollando la visión

Changes in the Member class. New Multimedia class and hierarchy between books and journals, the parent class is the new one - Item class

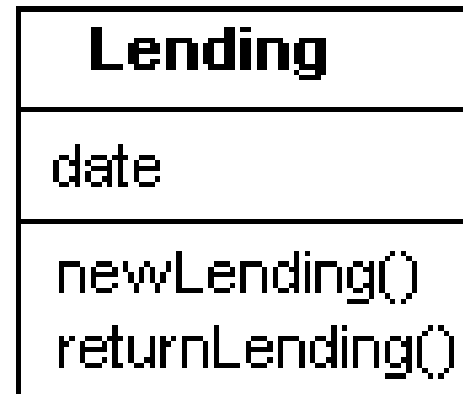
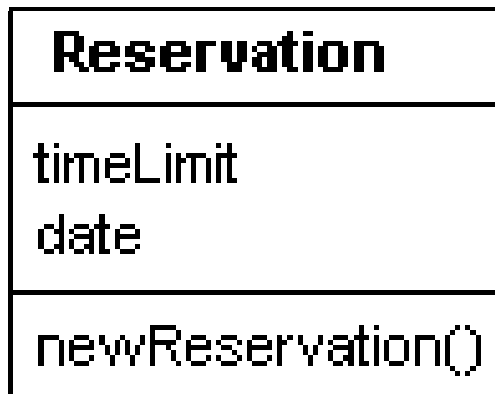


Desarrollando la visión

- **Analyst** During reservation process or lending process, current date must be somewhere stored. In our class diagram I don't see place in the classes where this data is stored. Please tell me where do you put information about date of lending and date of reservation?
- **Modeler** According to many-to-many association between the Sample class and the Member class, we are supposed to add a new class for that association holding the intersection data for that association. Right here we can add two new classes, the Reservation and the Lending. Here date of reservation and date of lending can take place. Also in the Reservation class an attribute for timeLimit of the reservation can be added. This is necessary for calculating the time period of holding a reservation to the member. These new classes are given in the following pictures

Desarrollando la visión

New classes added in our class diagram are the Reservation class and the Lending class



Desarrollando la visión

- **Analyst** Alright. For this moment I think that we've resolved all the problems about classes. Now, can we move on to some ideas about what the system should specifically do?
- **Facilitator** Sure. Ideas everyone?
- **Member** I think that this system will be more productive if we, the members, can access to all the data in the library from home. Sometimes, when I go to the library with thought to lend some book that I really need, I hear the Lender's words: "Sorry, but we're out of free samples for that book. Can I recommend you something else?". In that moment I'm saying into me: "My chef will kill me. I spent so much time and I didn't finished my work. I hope that there will be a little bit crowd in traffic."

Desarrollando la visión

- **Analyst** Yes. Our system has to somehow keep the members from walking to the library so much. Obviously they have to go to the library when they made a reservation from their home and a free sample is holded for them. Also a lending process can be done if a member is present in the library.
- **System Engineer** I think we're onto something. We can allow members to access to the library's LAN directly from their homes or work offices, using the global network - Internet. Then the information would move around very quickly. Also I heed to tell you that our system will cooperate with library's LAN, but new software will be implemented everywhere.

Desarrollando la visión

- **Analyst** The system you're talking about would resolve a number of issues. Like data search from home and making a reservation for a book or journal from your home or office. I'll have a feeling that I'm in the library and using their local area network and I don't care for a free terminal.
- **Member** That's beautiful. When I have a lot of work to do, I can turn on my computer, make a connection with the library and search the data to find out a free sample of books or journals I need to lend. But from library's network I can't make a reservation for a book or journal. How can I make a reservation from my home or office?

Desarrollando la visión

- **System Engineer** We are going to implement a Web interface that allows you to access data stored in the library's database, and allows you to make a reservation from your home or office by clicking a button that sends a message from your computer with your personal data and book's data that you want to reserve. This message will open a new reservation record in our database holding the data until you get your reservation, or time limit is overstepped.
- **Member** But how can my computer know my personal data?
- **System Engineer** First of all, Web interface can be used only by library's members. In our class diagram we have memberID - number printed on your member's card with all the necessary personal data. When you search the data you make a selection of books/journals to reserve. This list is also sent when you click the button. But if you aren't a member in the library, first that you must do is to make a library's membership.

Desarrollando la visión

- **Librarian** Is it possible some of our employees to access the system if they aren't present in the library too?
- **System Engineer** Of course. Data entry clerks can do their job even if they are not present in their work offices. They can access the database data from every place in the world that has a computer and Internet connection.
- **Librarian** How are you going to make this happen?
- **System Engineer** Let's don't worry about that right now. Very soon you'll find out how this will be realized.
- **Facilitator** So we're all set, then? Our system will incorporate the library's local area network and Internet. Now is time to give a name for our system.

Desarrollando la visión

- **System Engineer** How about Digital Library.
- **Facilitator** Can we all agree with Digital Library?
- (All agree)
- **Facilitator** Okay. I think our work here is done.

Sesión JAD

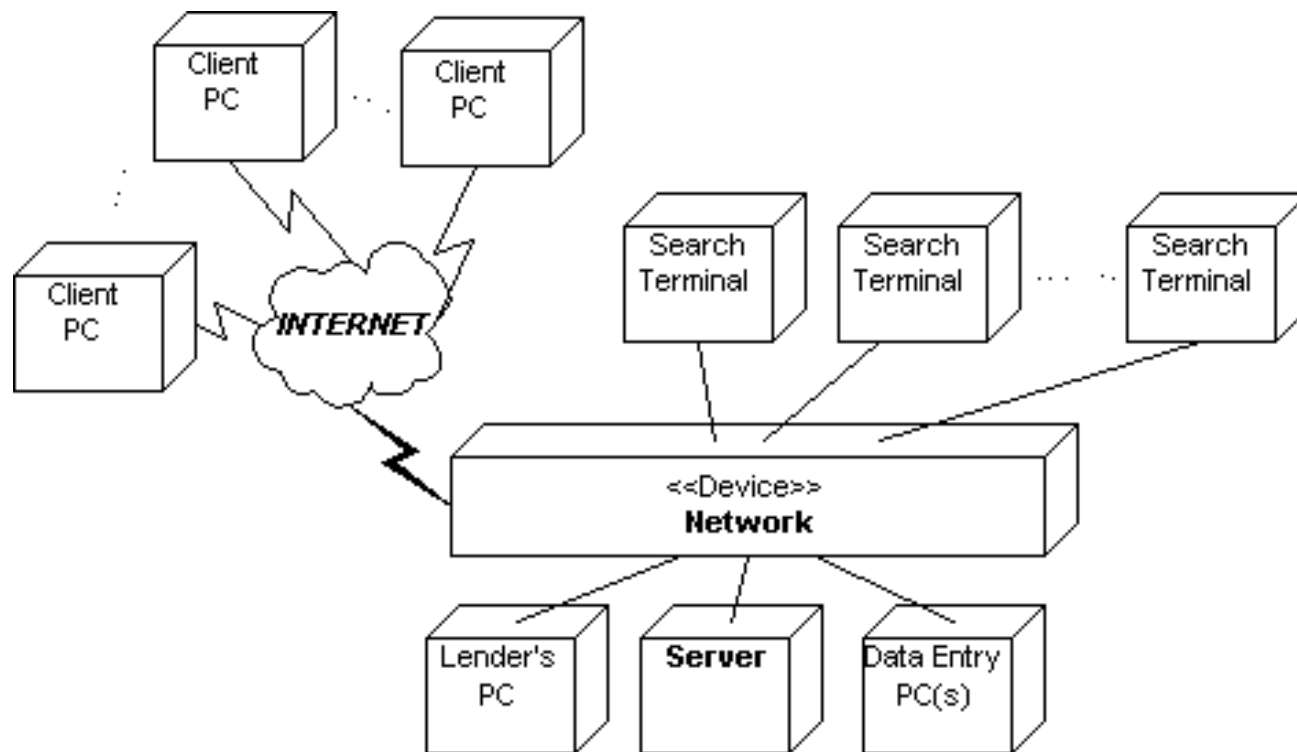
Now that the team has a vision for the system, can the programmers program and the system engineers engineer? Absolutely not. The team must center the Digital Library system around the user's needs, not around nifty technology. Although they have a few insights from the team meeting, they haven't exposed the Digital Library concept to a group of employees and end users - the members, to get feedback and ideas from the user's point of view. The next GRAPPLE action does just that. In a **Joint Application Development session**, the team will gather and document system requirements.

Sesión JAD

- The JAD session takes place in conference room. Led by a facilitator, it's called a *joint* because it includes members of the development team along with potential system users and domain experts. The development team members are analyst, a modeler, two programmers and a system engineer. The potential users are two members, a lender, two data entry clerks and a librarian. This meeting will produce a package diagram that shows the major functionality of the system.

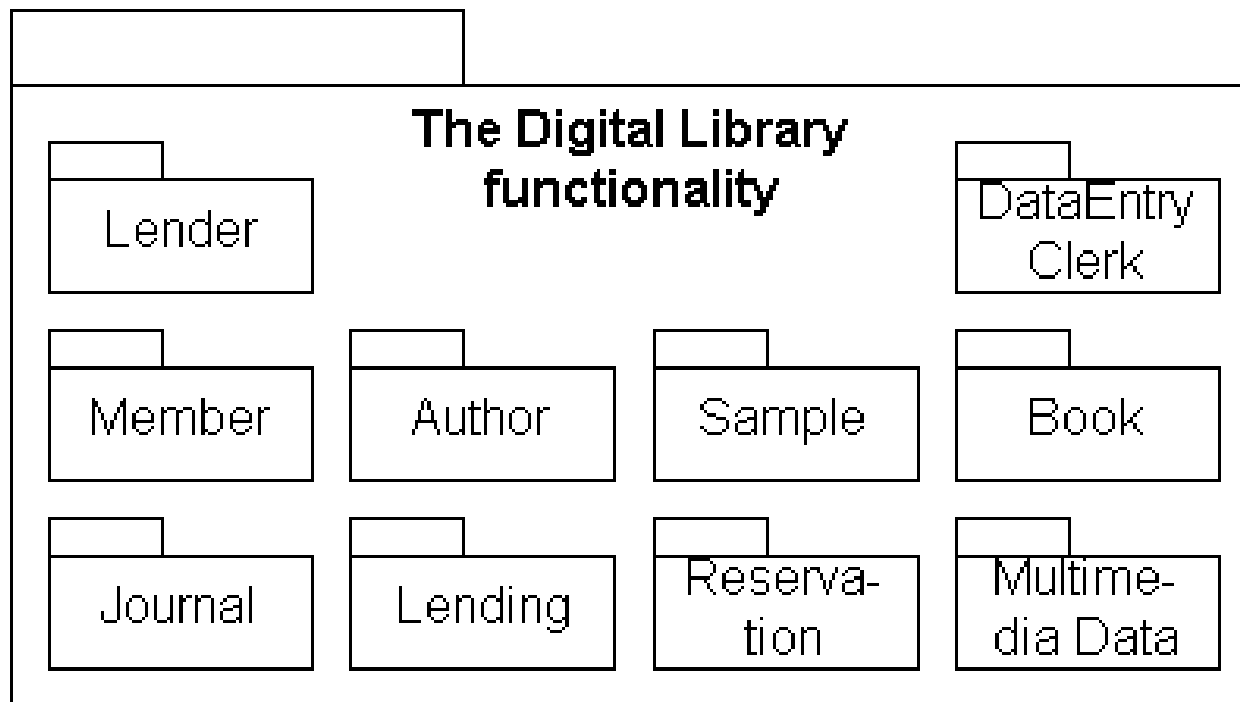
Sesión JAD

This session starts with a facilitator's speech about the Digital Library concept. The structure of a Digital Library system is represented on this picture, which is given to all the participants in the meeting.



Sesión JAD

The group starts their conversation by figuring out what the major pieces of functionality should the system had. It was decided to use following functionality of the system:



Sesión JAD

Next will be given a part of the meeting conversations between some of the participants:

- **Facilitator** Now that we have the major pieces, does anyone have a preference as to where to start?
- **Member1** How about the Member part?
- **Facilitator** Sounds good. Alright, what kinds of functionality would you want to see in this package? Remember, group, just because we're doing a piece that happens to not coincide with your particular job, you can still participate. everyone's insights are welcome.
- **Member1** I like to perform a search from my home and selected books and journals reserve until I go to the library and lend them.
- **Facilitator** Okay. What else?
- **Member2** I want to print electronic books and journals directly from my office.

Sesión JAD

- **Analyst** By the way I don't see any information about electronic items in our class diagram. How can we manage these items?
- **Modeler** You're right. In the Item class we must add new attributes for an electronic books/journals. A *status* will be one. With this one we can check the type of the item (electronic or paper). Also we must add an attribute that holds the location of the electronic item (on the local storage device or on Internet). This attribute will be named as *link*.
- **Facilitator** Has anyone had another kind of functionality to add?
- **Member1** I want to take a list of recommended books and journals onto my computer.
- **Facilitator** OK. You'll notice that I'm writing these in labeled ellipses. We refer to these as *use cases*. We'll be asking some of you to come back and help us analyze those use cases, but that's another meeting.

Sesión JAD

When JAD session has finished, participants had a set of requirements that appear as use cases arranged in the packages. The list of uses cases for every package is given to you.

The member package use cases:

▶ enter data for searching	▶ transmit the reservation to the library's database
▶ transmit the data to the library's database	▶ receive notification from library's database
▶ retrieve a list of items	▶ take a list of recommended books
▶ make a reservation for some of them	▶ take a list of recommended journals

Sesión JAD

The book package use cases:

▶ select an author	▶ check for reservations
▶ receive an author data	▶ receive number of reservations
▶ enter data for a new book	▶ find out the oldest reservation
▶ edit data for an existing book	▶ make a reservation
▶ enter multimedia data	▶ receive notification for reservation
▶ receive notification from multimedia adding	▶ make a lending
▶ check for free samples	▶ transmit lending data
▶ receive number of free samples	▶ receive notification for lending

The sample package use cases:

▶ add new sample	▶ receive a request for free samples
▶ set it free	▶ return number of free samples
▶ edit data for an existing sample	

Sesión JAD

The book package use cases:

▶ select an author	▶ check for reservations
▶ receive an author data	▶ receive number of reservations
▶ enter data for a new book	▶ find out the oldest reservation
▶ edit data for an existing book	▶ make a reservation
▶ enter multimedia data	▶ receive notification for reservation
▶ receive notification from multimedia adding	▶ make a lending
▶ check for free samples	▶ transmit lending data
▶ receive number of free samples	▶ receive notification for lending

The sample package use cases:

▶ add new sample	▶ receive a request for free samples
▶ set it free	▶ return number of free samples
▶ edit data for an existing sample	

Sesión JAD

The reservation package use cases:

▶ receive a request for reservation	▶ transmit a reservation result to the member
▶ perform a reservation	▶ delete a reservation

The lending package use cases:

▶ receive a request for new lending	▶ receive a request for returning a lending
▶ perform a lending	▶ return a lending
▶ transmit a lending result to the lender and book	▶ transmit a result to the lender
▶ delete lending	▶ check for reservations
▶ receive oldest reservation	▶ transmit information for reservations

Sesión JAD

The author package use cases:

- | | |
|--------------------------|-------------------------------------|
| ▶ enter new author | ▶ receive request for author's data |
| ▶ edit old author's data | ▶ transmit data to the book |

The lender package use cases:

- | | |
|-------------------------------------|---|
| ▶ enter new lending | ▶ perform a lending return |
| ▶ receive notification from lending | ▶ receive a result from returning a lending |

Sesión JAD

- For exercise try adding the use cases in DataEntry clerk package, Multimedia package, and to the Journal package.

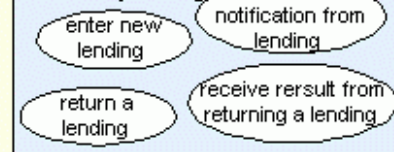
Sesión JAD

The Digital Library functionality (uncompleted)

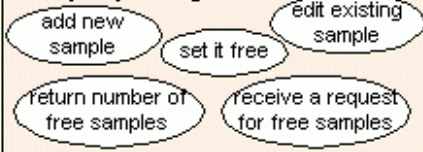
Book package



Lender package



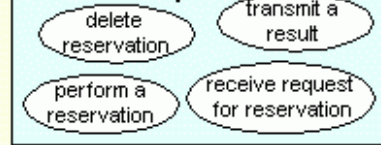
Sample package



Author package



Reservation package



Lending package



Member package



Sesión JAD

In a JAD session, the development team meets with potential users and domain experts to gather the requirements for the system. The result is a package diagram in which each package represent a major piece of functionality. Use cases inside a package elaborate on the functionality.

Preguntas y respuestas

- **1 Can you give a definition for a Digital Library, and why is it so important nowadays?**
- The Digital Library is a library which is not located on a building, but is located on computer servers.
Book's content is in electronic format, containing a lot of multimedia data (ordinary books don't have this kind of data) Access to the books is done by Internet.
Digital Libraries take a great progress today, because a lot of paper materials are converted into electronics format.
- **2 How do we make use of nouns in the interview with an expert?**
- Nouns became candidates for class names and class attributes.

Preguntas y respuestas

- **3 How do we use the verbs and verb phrases?**
- Verbs and verbs phrases become candidates for operations and for names of associations.

- **4 How are we representing system requirements?**
- We're using the UML package diagram along with use cases to represent system requirements.

- **5 Does class modeling stop after the domain analysis?**
- No it doesn't. Class modeling continues to evolve after the domain analysis.

Contenido (2ª Parte)

- **Analysis of the Digital Library**
 - The use cases analysis
 - Interactions in the Digital Library
 - Integration with other systems
- **Design of the Digital Library**

Análisis del sistema

- We've finished the requirements gathering process for the Digital Library model. During this day an analysis of the Digital Library model will be done.
- First we'll meet with use case analysis, after what a use case diagram for every package will be gained. Next we're going to discover and analyze an interactions among objects in the Digital Library system, and we'll produce collaboration diagrams. And at the end of the day we'll examine the interactions between our Digital Library with existing systems, like Library's local network, and we'll produce detailed deployment diagram, and a few user interfaces will be introduced.
- Let's start with the work.

Desarrollando los Casos de Uso

- The use cases from the package diagram, give a good picture of what the system will have to do. The team will have to flesh out each one. They've moved gradually from understanding the domain to understanding the system. The use cases have provided the bridge.
- At no point in the JAD session did the development team discuss how the system would accomplish all the activities specified in the panology of use cases. The idea was just enumerate all the possible use cases. As the use cases become fleshed out in this hour, notice how the components of the Digital Library system start to materialize. At this point in the development effort, the system begins to take center stage.

Desarrollando los Casos de Uso

- To analyze the use cases, we have to run another JAD session. The discussion in this JAD session is intended to derive analysis for each use case.
- The use case JAD session is usually the most difficult one, as it calls for the participants - potential users of the finished system - to become analysts. In their own niche, each one is a domain expert, and you have to tap into their expertise. Typically, they're not used to either verbalizing or analyzing what they know. It's also probably true that they haven't been part of a system design effort before, and they may be uncomfortable with trying to specify what a system should do to help them do their work.

Desarrollando los Casos de Uso

- **The system-related steps in the scenario are extremely important. They'll show how the system is supposed to work. When the JAD session participants tell us these steps, they're telling us, in effect, what the system will ultimately look like. After this JAD session, we should have a good idea about the components of the system.**

Desarrollando los Casos de Uso

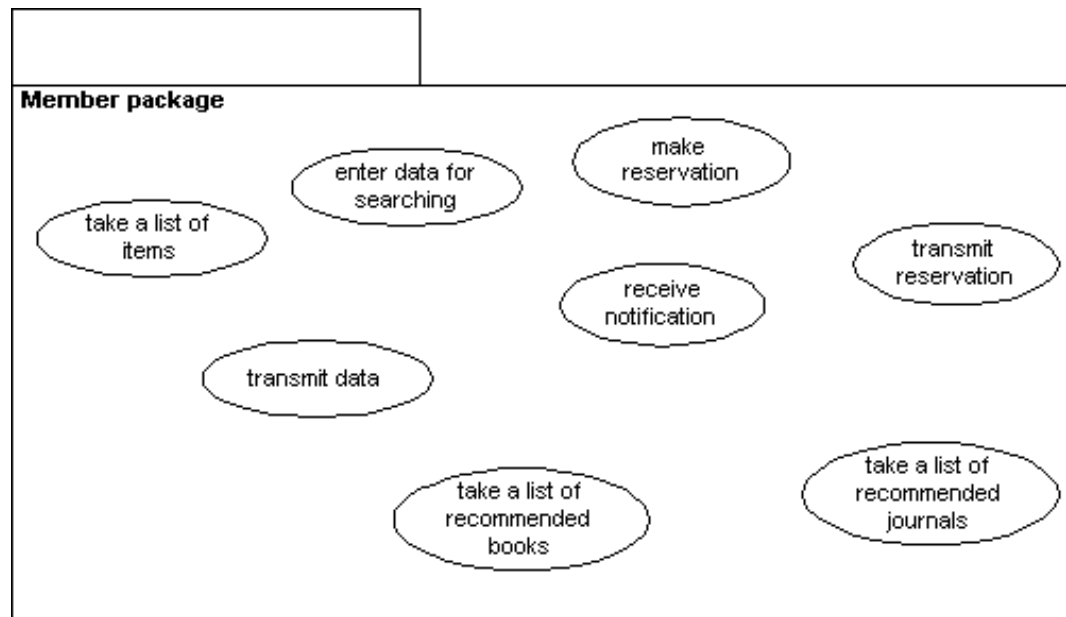
Each use case is a sequence of scenarios, and each scenario is a sequence of steps. For each scenario in each use case, we'll want to show:

- A brief description of the scenario
 - Assumptions for the scenario
 - The actor who initiates the use case
 - Preconditions for the use case
 - System-related steps in the scenario
 - Postconditions when the scenario is complete
 - The actor who benefits from the use case
-
- Use cases that we're going to examine are numerous. That's why when we're performing an use case analysis number of them will be examined in detail, and the rest of them will be given to you as an exercise. Now let's dive into process of analysis the use case.

El análisis del Caso de Uso 1

We'll meet with the process of analysis of the use cases. We're going to examine a member's package of use cases and a book's package of use cases. The rest of them are given to you, to try yourself in the use case analysis.

- **The Member package**



El análisis del Caso de Uso 1

- Enter data for searching

The good-one sentence description for this use case may be: *The member enters a data for searching into search form.*

Here initialing actor will be Member, which is also a benefiting actor.

The assumptions are that a member wants to find out books or journals. Another assumption is that the Digital Library's interface has a form dedicated for search data entry.

The precondition is that a member has a necessary data to make a search. The postcondition is that a member enters necessary data into Digital Library.

The steps in the use case are:

- ▶ From the Library's LAN or from home/office, the member activates the user interface for search entry.
- ▶ The search form appears on the screen.
- ▶ The member enters necessary data patterns into search form.

El análisis del Caso de Uso 1

- Transmit data for searching

The description for this use case may be: *Take a data pattern entered in search form, and send it to the Digital Library's database search engine.*

The assumptions are that we have communication via the Internet or library's LAN, and we have a search form in the member's interface.

We must make repetitions of some of the assumptions, because each use case will eventually appear on a separate page in the design document, which will serve as a reference about the system.

The precondition is entered data pattern in search form, the postcondition is that the data pattern has arrived in the Digital Library database search engine.

The steps in the use case are:

- ▶ Click on the SUBMIT button on the search form.
- ▶ Activate the transmitting mechanism for data in network(s).
- ▶ The data pattern arrives in Digital Library database search engine.
- ▶ On the member's screen appears a message that data has been sent, and to wait a moment during receiving the data.

Because this use case is related to the previous one, we must make a change to the Member package. It has to show <<include>> dependency between this one and "Enter data for searching" use case.

El análisis del Caso de Uso 1

- Take a list of searched items

This one use case is the last part for completion the searching process. Description here is: *Receive a list of book or journals, after entering a data in search form and send it to the Digital Library database.*

The assumptions are that a member uses a search form, and a communication via network(s) is good.

The precondition is clicked SUBMIT button on search form, and the postcondition is printed list of items on the member's screen. The actor here again is Member.

The steps of this use case are:

- Wait until search is performed (in paralell a Digital Library search engine performs a search).
- Digital Library database transmits a list of items to the member's interface via the network(s).
- The member receives a list of items satisfying entered search-data pattern.

Inclusion also appears in this use case. <<include>> joins this use case with the previous one.

El análisis del Caso de Uso 1

- Make reservation

Description for this use case will be: *The member wants to make a reservation for the item gained from searching process or recommendation process.*

As you can see this use case depends on two different use cases - search and recommendation. Thus reservation process may take place only when one of these two processes take turn. That's because an inclusion is presented between these three use cases.

The assumptions are a list of items, and a button for reservation.

The precondition is clicked RESERVE button, and a postcondition is transmitted reservation record to the Digital Library.

The steps of this use case are:

- The member receives a list of items (from searching or from recommendation)
- The member clicks the RESERVE button for the items that he or she wants to reserve
- The member fills in the information in the form that appears for the reservation

- Transmit reservation

This use case wouldn't be explained in detail, because it is almost the same with the "transmit the data for searching" use case. Only we'll say that this use case is related with "make reservation" and "receive notification" use cases. The connection is <<include>>.

El análisis del Caso de Uso 1

- Receive notification

Description is: *After pressing the SUBMIT button in the reservation form, a notification from the Digital Library must arrive that the reservation has been accepted, and inform the member with the status of reservation.*

Here status of reservation is for having or not a free sample for reserved item. If a free sample is present how many weeks it'll be holded for the member.

The assumptions are good communication in the network(s) and clicked SUBMIT button.

The precondition is clicked SUBMIT button, and a postcondition is received status of reservation

The steps of this use case are:

- Wait until notification is received
- Transmit the necessary information via the network to the member's interface
- Members receives the status of reservation

El análisis del Caso de Uso 1

- Take a list of recommended items

Description is: *The member activates the recommendation part of the system, and gets a list of recommended items*

The assumptions are clicking the RECOMMEND button, and a working recommendation algorithm.

The precondition is clicked RECOMMEND button, and a postcondition is received list of items.

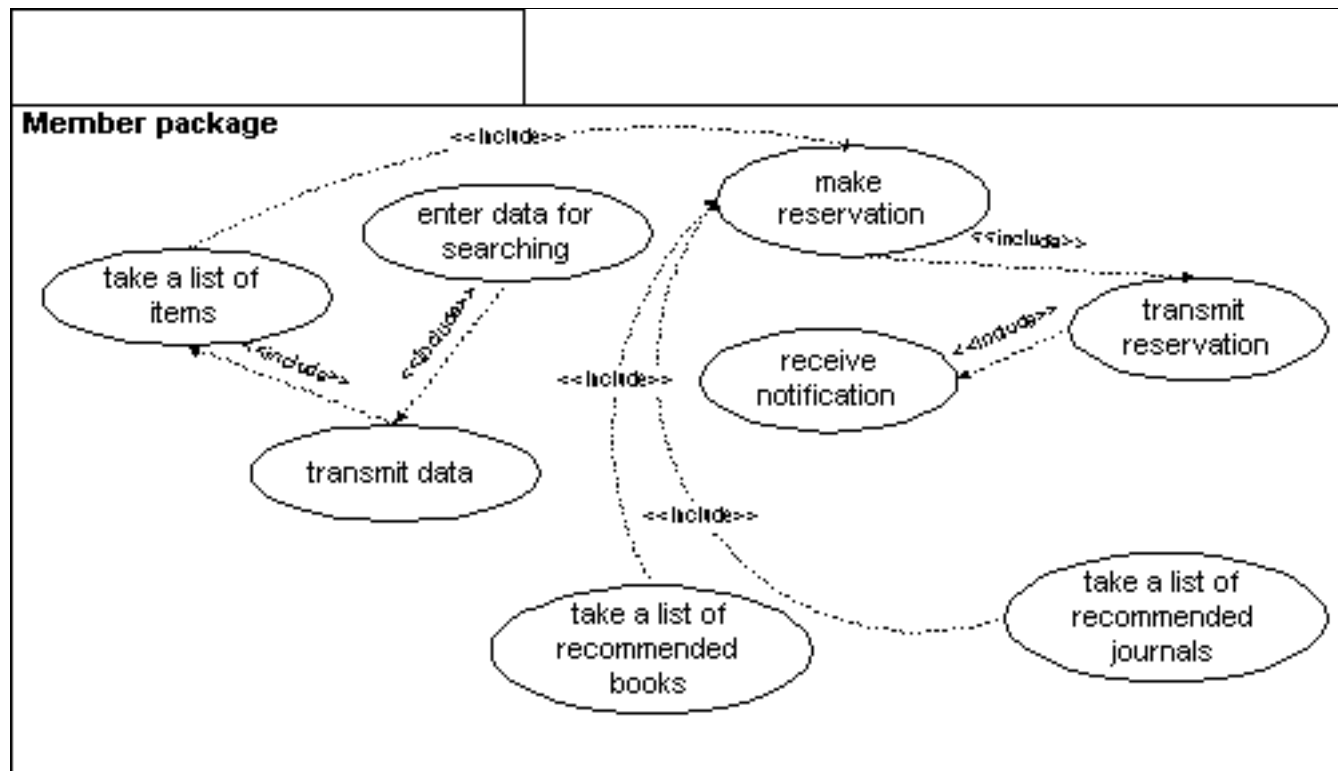
The initiating and benefiting actor is Member.

The steps of this use case are:

- ▶ Activate the recommendation part from the user's interface.
- ▶ The Digital Library activates a recommendation algorithm which produces a list of items.
- ▶ Print this list of items to the member's screen.

El análisis del Caso de Uso 1

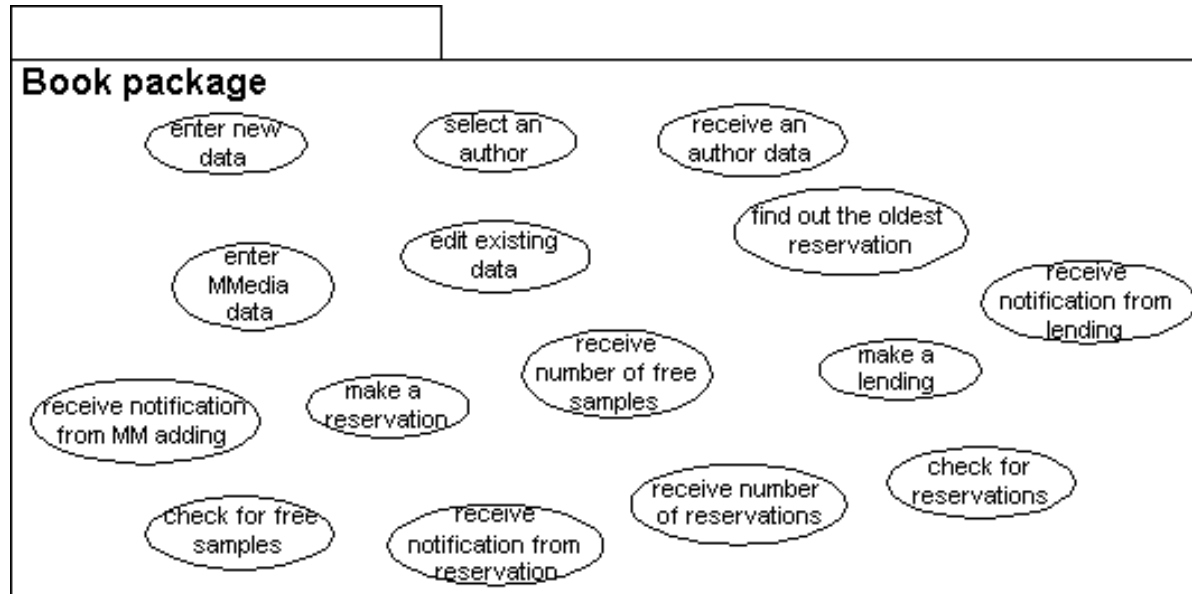
The updated use case diagrams for the Member package



El análisis del Caso de Uso 2

Now let's take look at the Book package use cases, and analyze them separately.

- **The Book package**



The Book package of use cases developed in earlier stages of the system analysis

El análisis del Caso de Uso 2

- Enter data for a new book

Description: *Fill in the data in book's entry form.*

Initiating actor is DataEntry clerk.

Assumptions: Activated entry form in the DataEntry clerk's interface, and entered information.

Precondition is new book's data.

Postcondition is entered data into Digital Library database.

The steps are:

- From the library's LAN or from internet the DataEntry clerk activates an interface for a book's entry.
- The form appears on the screen.
- The clerk enters necessary information.

El análisis del Caso de Uso 2

- Select an author

Description: *Select a book's author from the author list.*

Initiating actor is DataEntry clerk.

Assumptions: There are author records in the database, and user makes a selection from that list.

Precondition is book's author data.

Postcondition is selected author's data.

The steps are:

- ▶ In data entry form for the book, a clerk selects one author from the list of authors.
- ▶ This information is send to the database.
- ▶ Searching is performed into the database for that information.

An inclusion may be introduced here. This use case includes steps from "edit data for an existing book" use case and from "enter data for a new book" use case.

El análisis del Caso de Uso 2

- Retrieve author's data

Description: *Receive all the necessary information about the selected author in previous use case.*

Assumptions: User has made a selection from the author's list.

Precondition is selected author.

Postcondition is retrieved author's information.

The steps are:

- ▶ After searching, database selects the record for that author and extracts necessary information about the author.
- ▶ Sends that information to the book's data entry form.
- ▶ Prints the information onto the user's screen.

This use case includes the "select an author" use case.

El análisis del Caso de Uso 2

- Enter multimedia data

Description: *If the book has a multimedia data, then it is entered into the book's record.*

Assumptions: Book has a multimedia data, and a multimedia file(s) is(are) saved somewhere in the server.

Precondition is book has a multimedia data and a MMEDIA button is clicked.

Postcondition is entered multimedia data.

The steps are:

- If book has a multimedia data, a MMEDIA button is clicked, and a entry form appears on the screen.
- User enters type of multimedia data, and a location on the server where that data is saved.
- SUBMIT button is clicked and multimedia record is transmited to the database.

An inclusion may be introduced here. This use case includes steps from "edit data for an existing book" use case and from "enter data for a new book" use case.

El análisis del Caso de Uso 2

- Receive notification from multimedia adding

Description: *After filling the multimedia entry form, database sends a message how successful was performed a process of adding new multimedia data.*

Assumptions: Filled multimedia data entry form, good communication via network, and existing multimedia file.

Precondition is transmitted multimedia entry form to the database.

Postcondition is received status of this transaction.

The steps are:

- ▶ The multimedia data arrives into the database, and a process of creation of new record is started.
- ▶ This process may finish with a result, or may end with failure.
- ▶ The type of ending of the creation process is returned to the book's data entry form.

This use case is inclusion of previous one - "enter multimedia data" use case.

El análisis del Caso de Uso 2

- Edit data for an existing book

Description: *If a wrong information is entered for some book, it is possible to change that information.*

Assumptions: Existing book's record for changeing, and clicked EDIT button.

Precondition is false entered information into book's record.

Postcondition is changed book's record.

The steps are:

- User clicks the EDIT button and filled form with the old book's information appears on the screen.
- False information is selected and changed.
- If an author's information or multimedia information has to be changed, the same process like entering new data is repeated.

El análisis del Caso de Uso 2

- Make a reservation

Description: *Member wants to reserve a sample for the book, and after that a lending can be performed in the library.*

Assumptions: Book must be present in the retrieved list of books during the search or recommendation process, books must have samples present in the library.

Precondition is Member wants to make a reservation for a book.

Postcondition is book's necessary data and member's necessary data are transferred to the Digital Library.

The steps are:

- Member clicks the RESERVE button and fills in the necessary information to perform a reservation.
- The Digital Library database receives request for reservation.
- The steps of checking the data with database records are made.
- Digital Library checks for free samples, and performs a new reservation.
- After that a wait period is introduced, while the notification from the reservation is achieved.

El análisis del Caso de Uso 2

- Receive notification from reservation

Description: *After clicking the RESERVE button, and filling the form, Digital Library receives a request for a reservation, and after performing a reservation process, a status message is received.*

Assumptions: RESERVE button, and sent request for reservation.

Precondition is clicked RESERVE button.

Postcondition is reservation status.

The steps are:

- ▶ The status of performing a new reservation is achieved.
- ▶ This status is sent through the network to the user.
- ▶ Message about status of reservation is printed on the screen.

This use case includes "make a reservation" use case.

El análisis del Caso de Uso 2

- Check for free samples

Description: *Count how many samples from that book are available, and return that number to the reservation process.*

Assumptions: request for reservation, sample(s) for the book.

Precondition is new reservation.

Postcondition is number of free samples

The steps are:

- Receive a request for a number of free samples.
- Count the available samples (those one with status="available").
- Sent that number to the Digital Library.

This use case includes "make a reservation" use case.

El análisis del Caso de Uso 2

- Return number of free samples

Description: *Gives a number of free samples.*

Assumptions: samples of that book, request for number of free samples.

Precondition is request for number of samples.

Postcondition is transmitting the number of free samples

The steps are:

- Receive the transmitted number of free samples.
- Check the result to determine the reservation status.
- Transmit the reservation status to the Digital Library.

This use case includes "check for free samples" use case.

El análisis del Caso de Uso 2

- Make a lending

Description: *Member wants to lend a book.*

Assumptions: user must be present in the library, book must be in paper form, a free sample for the book must be available.

Initiatin actor is Lender and benefiting actor is Member.

Precondition is request for lending.

Postcondition is transmited lending request to the Digital Library.

The steps are:

- Lender activates interface for makeing and returning a lending.
- Lender enters data for the sample and the member in the lending data entry form.
- After pressing the LEND button a new lending process is activated.

El análisis del Caso de Uso 2

- Receive notification for lending

Description: *After pressing the LEND button in the lending form, a new reservation record is written to the database, and the result of this transaction is returned to the lender's interface.*

Assumptions: communication via network, and LEND button present in the interface.

Precondition is entered data in lending form.

Postcondition is status of performed transaction.

The steps are:

- ▶ Wait until lending process ends, and receive a lending result status.
- ▶ Transmit the status via network to the lender's screen.
- ▶ If there are free samples, member gets one.

This use case includes "make a lending" use case.

El análisis del Caso de Uso 2

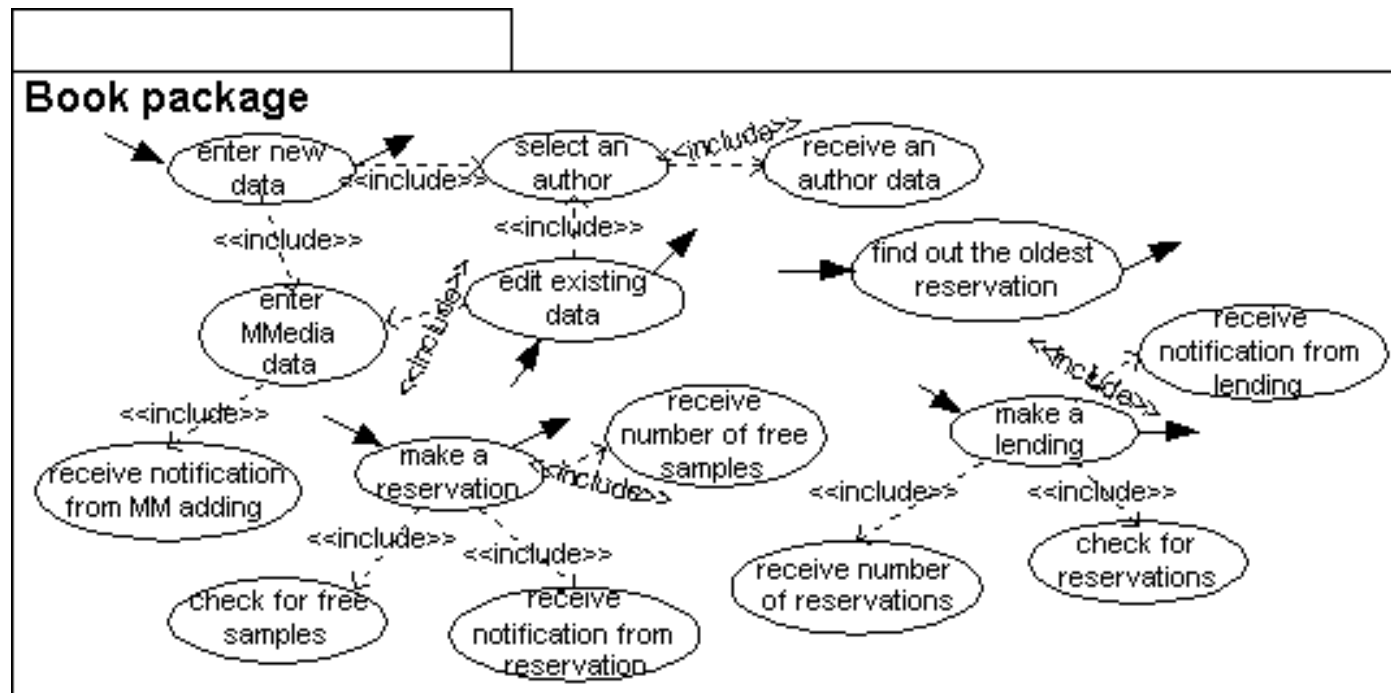
Another use cases related to the lending process are:

▶ Receive number of reservations	This use case take place when new lending record is created. If a book has a pending reservations (free samples with status="reserved" are available), and no free samples with status="available" are present, then lending process must fail.
▶ Check for reservation	If a member has a reservation for a book's sample, then if a holded sample is available, the member takes that sample, else a lending process will fail.
▶ Find out the oldest reservation	This use case is used in the process of updating reservation records. When a member make a request for a reservation and there is no free sample, then when only one sample is set free, this use case take place and finds out the oldest reservation for that book, and sets the sample's status from "available" to "reserved". Sample can be set free when it is returned from a user who lends it.

Inclusions that appears here are between: "make a lending" and a "check for reservation" use cases; "make a lending" and a "receive number of reservations" use cases.

El análisis del Caso de Uso 2

- *The updated use case diagrams for the Book package*



El análisis del Caso de Uso 2

Analyzing the use cases has helped conceptualize the working parts of the system. Although we know a lot about the use cases, we still have to model how those working parts will interact with one another and how (and when) they change state. Passing this information to the programmers will make their jobs a lot easier. They will have a vision of how to code classes and make them both together.

Refinar el diagrama de clases

Now is the time to complete our class diagram. We will analyze every class gained from previous analysis.

It is obvious to have a deleting process invert to the creation process. This is said because in almost every class we have a creating operation (in Object Oriented programming known as *constructor*), therefore an operation for deleting an object must be obtained in every class (in Object Oriented programming this is known as *destructor*).

Refinar el diagrama de clases

A new attributes must be added for a Member class. They are *alias*, *password* and *status*. Alias and password will serve as an attributes for logging onto the system. It is obvious that Internet interface will allow a lot of people to access the Digital Library, but only those who have contracted membership with the library, can browse the data, make reservations and printing the electronic items. The Status attribute is introduced to distinguish between ordinary member and library employee-member. We said that every employee also can be a member of a library.

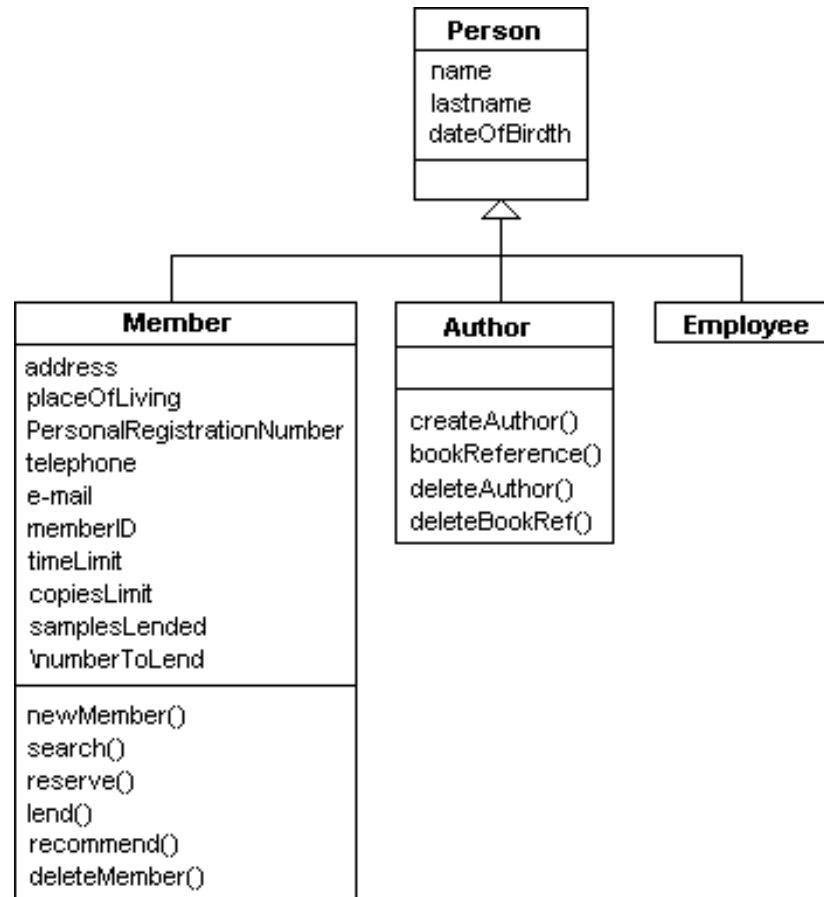
New operations for the Item class can be introduced according to the previous use case analysis. This operations are *findOutTheOldestReservation()*, *checkForReservation()* and *receiveNumberOfReservations()*, and they are described in previous lesson.

Refinar el diagrama de clases

If we take a look at Member class attributes, Employee class attributes and an Author class attributes, we'll see that most of them are common for these three classes. Therefore an inheritance may be introduced. We will create new class *Person* which will hold name, lastname and date of birth attributes.

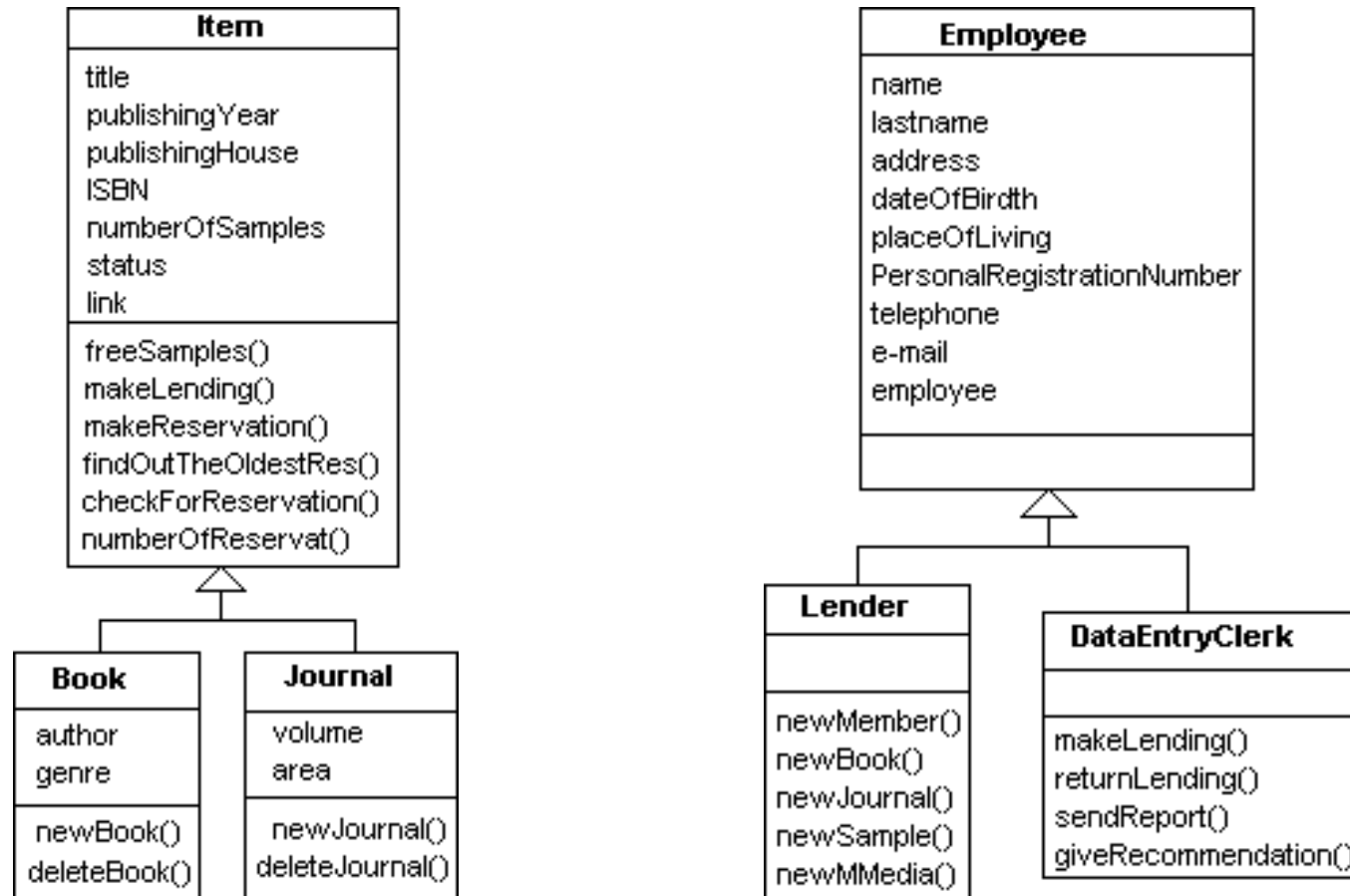
Description for classes is not longer necessary, because you are going to see all the classes with their last and final state. They are presented in the following list of classes.

Refinar el diagrama de clases



The parent class Person with it's children

Refinar el diagrama de clases



The Item class hierarchy

The Employee class and its children

Refinar el diagrama de clases

Multimedia
type file
newMultimedia() deleteMultimedia()

The Multimedia class

Reservation
timeLimit date
newReservation() deleteReserv()

The Reservation class

Sample
signature status
newSample() deleteSample()

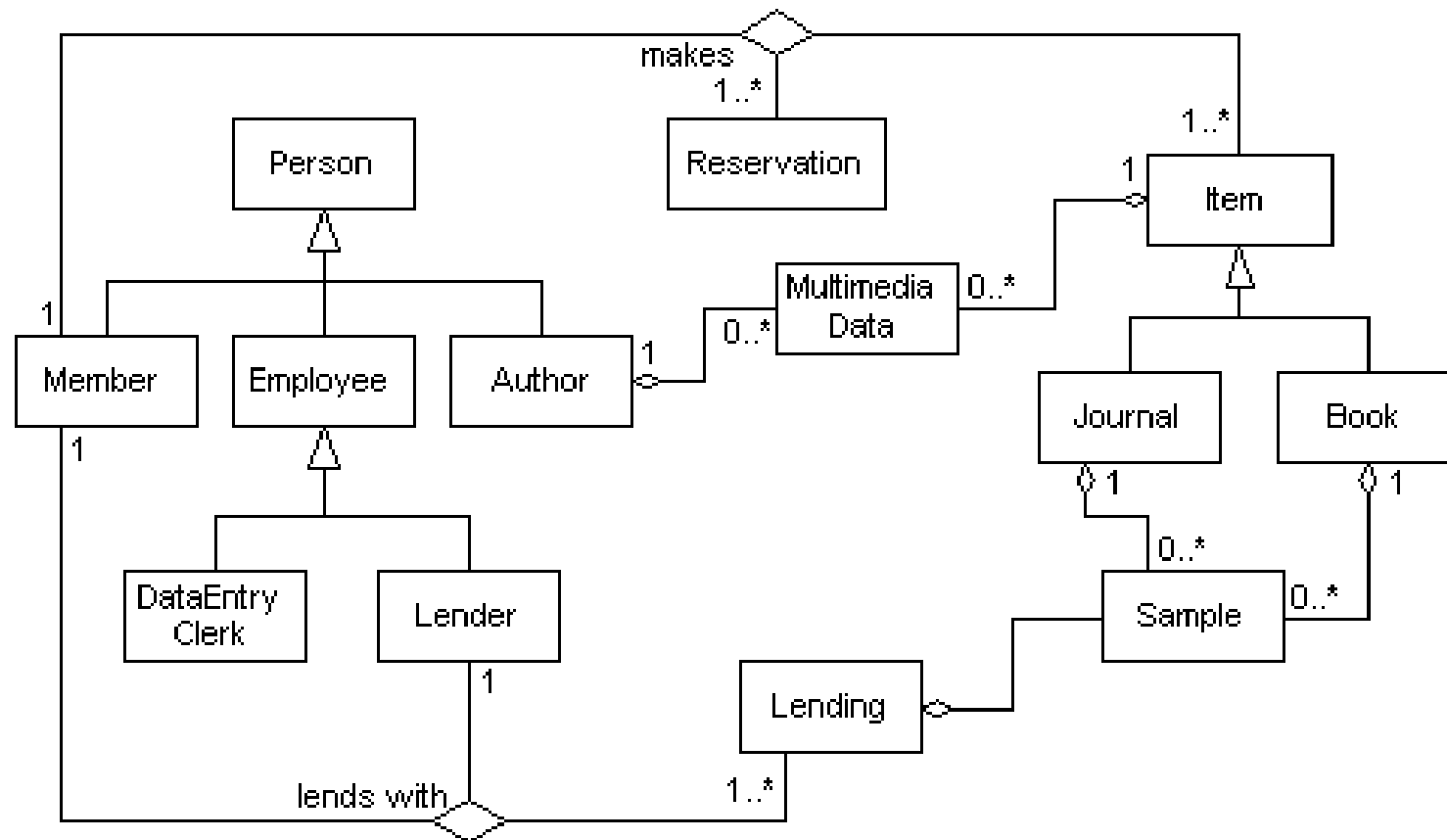
The Sample class

Lending
date
newLending() returnLending() deleteLending()

The Lending class

Refinar el diagrama de clases

The class diagram associations



Interacciones del Sistema 1

- One way to find out the interactions in the system is to enumerate the system components suggested in each package of use cases. The task to perform is to show how the system components interact in order to complete each use case.
- Although we didn't explicitly analyze all the use cases in all the packages in the previous lessons, we can still extract the system components those use cases assume. Let's start to reanalyze the packages that were analyzed in previous lessons. That were the Member package and the Book package. We're going to find interactions between components of these packages, but only for reservation and lending process.
- We'll analyze only these two interactions, and a rest of them you'll need to do.

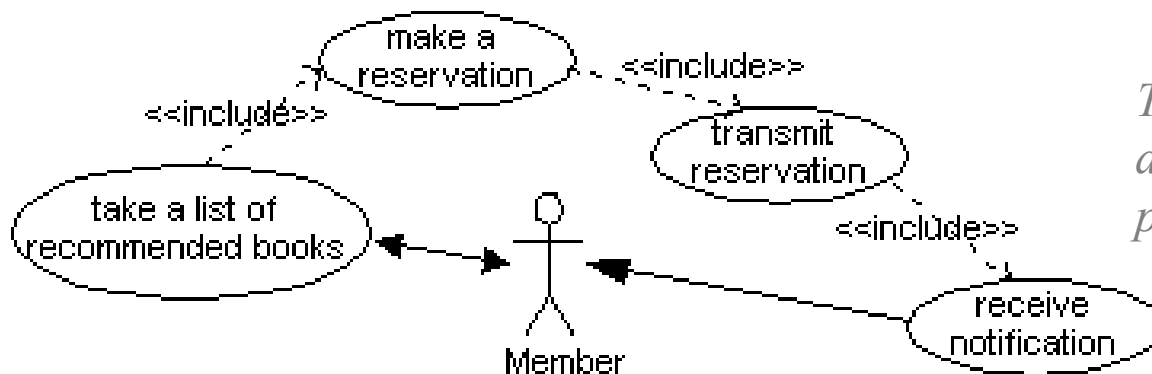
Interacciones del Sistema 1

■ The Member package

The member package use cases:

▶ enter data for searching	▶ transmit the reservation to the library's database
▶ transmit the data to the library's database	▶ receive notification from library's database
▶ retrieve a list of items	▶ take a list of recommended books
▶ make a reservation for some of them	▶ take a list of recommended journals

Let's examine again the process of reserving the book. Suppose that you have to reserve a book from retrieved list of recommended books. Now let's look at the steps necessary to perform these operations.



The part of use case diagram in the Member package

Interacciones del Sistema 1

- Take a list of recommended books

The steps of this use case are:

- 1 Activate the recommendation part from the member's interface.
- 2 After clicking the RECOMMEND button on the interface, a signal is sent to the Digital Library, to activate the recommendation algorithm.
- 3 The Digital Library activates a *recommendation algorithm** which produces a list of recommended books.
- 4 The Digital Library sends this list of books to the member's interface.
- 5 Print this list of items to the member's screen.

* For the purpose of Digital Library a special algorithm for recommendation is developed. If you need some detailed information about this algorithm please refer to the webmaster of this course.

Interacciones del Sistema 1

To activate the reservation process, a RESERVE button must be obtained beyond every item in the list. Pressing this button another use case sequence is activated. This sequence includes "make reservation" use case, "transmit reservation" use case, and "receive notification" use case. The steps for everyone use case are given in following tables:

- Make reservation

- | | |
|---|--|
| 1 | The member receives a list of items (from searching or from recommendation) |
| 2 | The member clicks the RESERVE button for the items that he or she wants to reserve |
| 3 | The form for acknowledging a reservation appears |
| 4 | The member fills in the information in the form |

- Transmit reservation

- | | |
|---|---|
| 1 | Click on the SUBMIT button on the reservation form. |
| 2 | Digital Library transmits the reservation request to the Reservation database. |
| 3 | The request arrives in the Reservation database. |
| 4 | On the member's screen appears a message indicating "request sent", and a message " wait to receive acknowledge". |

Interacciones del Sistema 1

Before going to the "receive notification" use case we'll take a look at Reservation package use cases. A "new reservation" and "transmit notification" use cases will take part at this process of interactions.

- New reservation

- 1 A request arrives in the Reservation database, and the process of creation new reservation is activated.
- 2 Result of this process is status of reservation, which is sent to the Digital Library.
- 3 Send the status of reservation to the Digital Library.

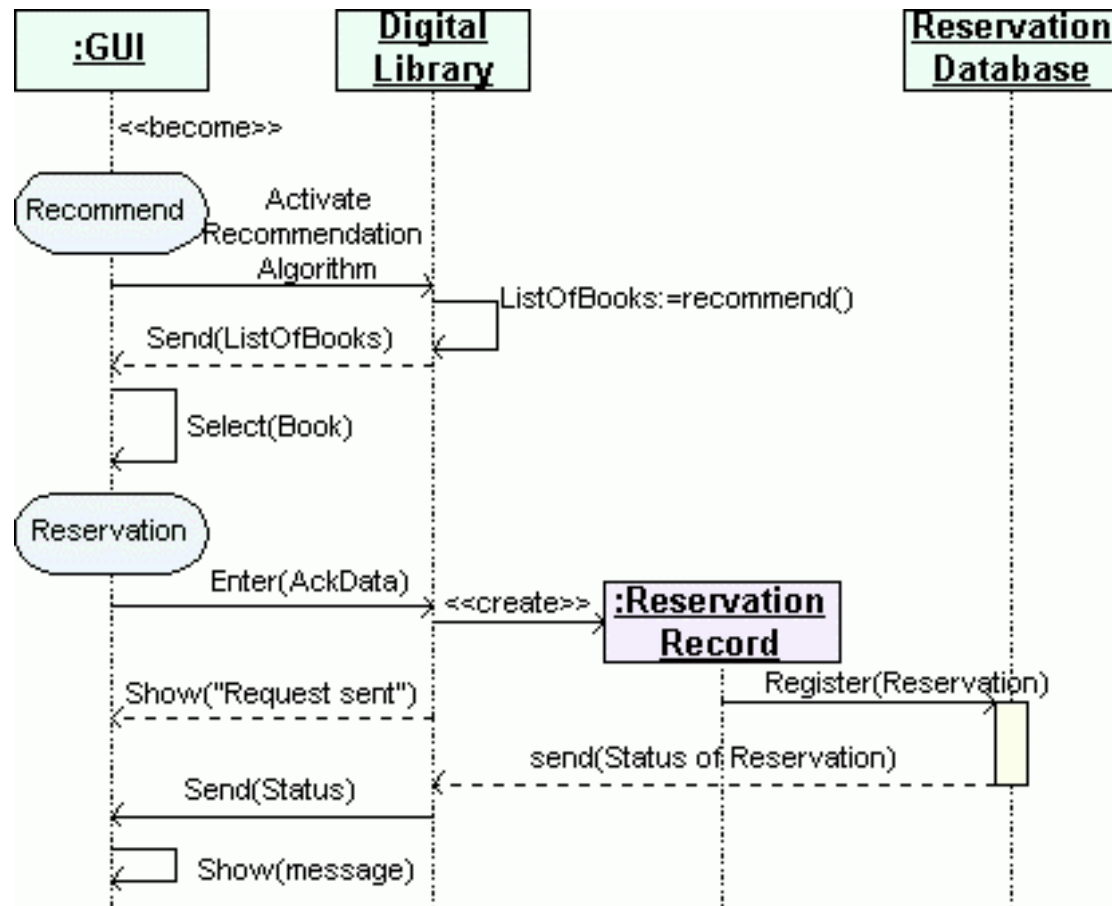
- Transmit notification

- 1 Digital Library sends the status of reservation to the member's interface.
- 2 Status arrives in the member's side.

- Receive notification

- 1 Member's interface receives arrived status.
- 2 Status is decoded.
- 3 A message is printed on the member's screen according to the decoded status of reservation.

Interacciones del Sistema 1



The sequence diagram for reservation process of recommended book

Interacciones del Sistema 1

As you can see some of use case steps are modified. These modifications are example of the way one phase of a project can influence another. Note also the state changes on the first lifeline. It's intended to clarify how the user interface sets up to handle a particular kind of activity. We could include all the possible state changes as separate state diagrams, but this would be overkill. Putting them on the sequence diagrams appears to be more economical

Interacciones del Sistema 2

Suppose that member wants to lend a book. First of all, member must be present in the library, the book for lending must be in paper format, and a free sample must be present in the library's storage. If all of these conditions are realized then a sample of book can be lent to the member.

To avoid going to the library and search for free sample, the member can perform a reservation to the Digital Library system, and after receiving a notification that a free sample for the reservation is present in the library, can go to the library and make his or hers lending.

Interacciones del Sistema 2

Let's look at use cases in the Book package necessary for this operation.

■ The Book package

The book package use cases:

▶ select an author	▶ check for reservations
▶ receive an author data	▶ receive number of reservations
▶ enter data for a new book	▶ find out the oldest reservation
▶ edit data for an existing book	▶ make a reservation
▶ enter multimedia data	▶ receive notification for reservation
▶ receive notification from multimedia adding	▶ make a lending
▶ check for free samples	▶ transmit lending data
▶ receive number of free samples	▶ receive notification for lending

Interacciones del Sistema 2

For process of lending we'll need these use cases: "make a lending", "transmit lending data", "receive number of reservations", "check for reservation" and "receive notification from lending". Let's take a look at steps of every use case.

- Make a lending

- | | |
|---|--|
| 1 | Activate the make/return lending part from the lender's interface. |
| 2 | A lending form appears on the lender's screen. |
| 3 | The lender enters the necessary data. |
| 4 | Lending data are transmitted to the Digital Library |

- Transmit lending data

- | | |
|---|--|
| 1 | The lender clicks the SUBMIT button on the lending form, and lending data is sent to the Digital Library |
| 2 | The Digital Library receives this data, and a request for new lending is sent to the Lending database |
| 3 | The request is sent to the Lending database, and an acknowledge is waited to receive |

Interacciones del Sistema 2

When data is transmitted to the Lending database, a mechanism for adding a new record in the database is activated. This operation incorporates these use cases located in the Lending package: "receive request for new lending", "perform a lending" and "transmit lending result". When performing a lending, also results from the Book package use cases are obtained. This process needs results for pending reservation made by that member for that book, and this will be done only if a result from "receive number of reservations" use case is not equal to zero. Checking for pending reservations is necessary, because if a reservation is made for a book (a sample for that book is holded for the member), then when a lending is done, that reservation from Reservation database must be deleted.

Interacciones del Sistema 2

- Receive request for new lending

- 1 The Lending database receives a request for performing a process of creating a new record
- 2 This activates the process of creating and adding a new record to the database

- Perform a lending

- 1 Database has activated its mechanisms for creating a new record
- 2 The result of creation a new record is aimed
- 3 This result is sent to the Digital Library

- Transmit lending result

- 1 Result from lending is sent to the Digital Library
- 2 This result is transmitted to the lender's interface

Interacciones del Sistema 2

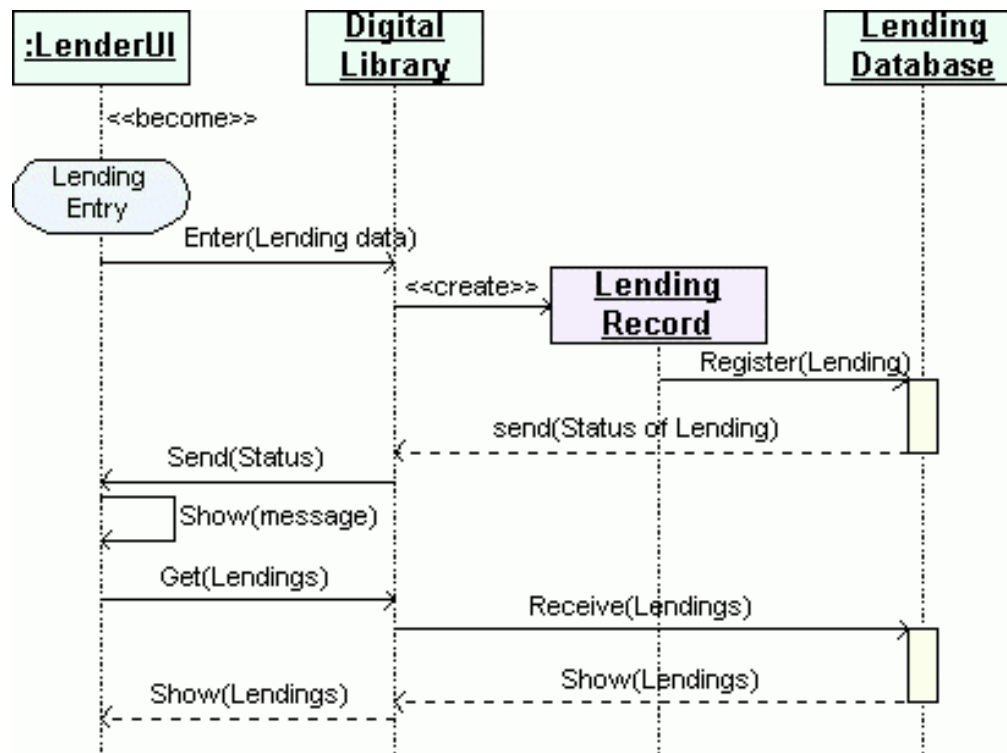
Now we can continue with use cases from the Book package:

- Receive notification from lending

- | | |
|---|--|
| 1 | Lender's interface has received result from lending |
| 2 | This result is decoded, and a message is printed on a screen |
| 3 | A request for current state in the member's lending record is sent to the Lending database |
| 4 | A list of records is retrieved from the Lending database |
| 5 | This list is printed on the lender's screen |

Interacciones del Sistema 2

The sequence diagram for these use cases steps is shown on the next picture:



The sequence diagram for "Make a lending"

Interacciones del Sistema 2

As you can see some of use cases in the given list were not used for this purpose. They are internally used in the activation of object in the Lending database. The operation for creation a record incorporates these use cases, and after that returns a result, that is dotted line message going out from the activation in the LendingDB lifeline.

Interacciones del Sistema 2

For an exercise try drawing out the sequence diagrams for "return a lending" use case from the Book package, and for "enter data for a new journal" use case from the Journal package.

Interacciones del Sistema 2

After you model interaction among components, the system is much closer to becoming reality. As you model the interactions, you may find that it's appropriate to modify the use cases at the base of these interactions. The results of this analysis should make it easy for programmers to code the system objects and how they communicate with one another.

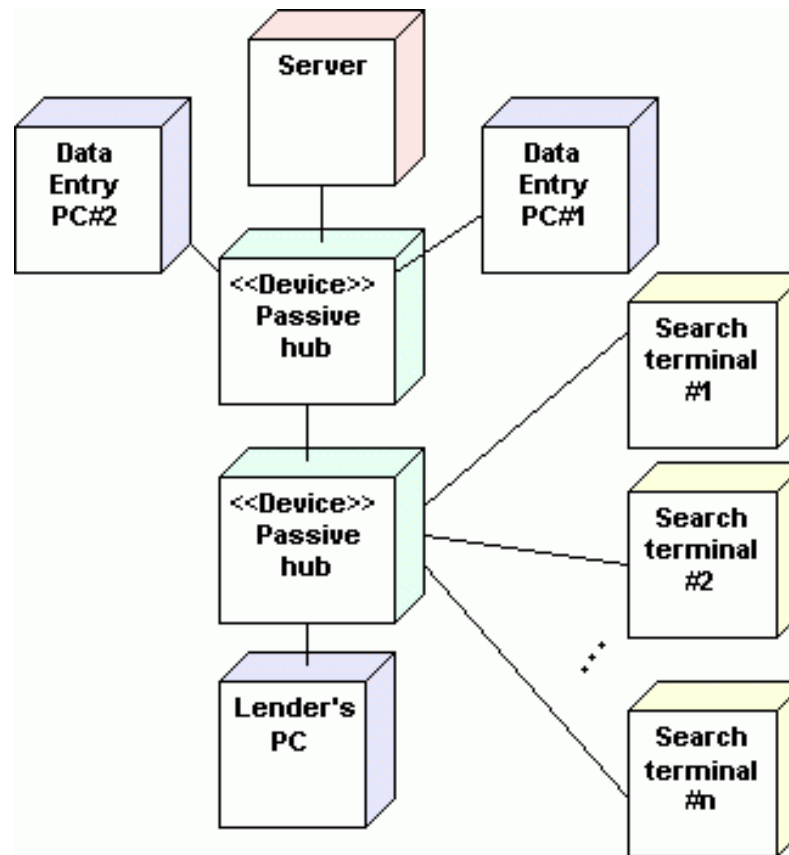
Analizando la integración con el sistema

After the GRAPPLE analysis segment has produced the general concept of the Digital Library system, a system engineer will start thinking about how the physical structure should look. He or she will start considering alternative network topologies and how to implement them, and will start figuring out which software components belong on which nodes in the network.

- The development team have made decision that a library local network will take part in the Digital Library system, and that an Internet interface will be implemented on the LAN's server.

Analizando la integración con el sistema

First let's look at the physical architecture of the library's local area network.



The Deployment diagram for the library LAN

Analizando la integración con el sistema

As you can see from this deployment diagram, library LAN consists from these hardware components:

Component name	Amount
Server	1
DataEntry PC	2
Lender's PC	1
Search terminal	6
Passive hub	2

- Server and DataEntry PCs are located in the same room, also one Passive hub is located in that room. A cable from this passive hub goes out from the room, to the corridor. There another passive hub is located. From this hub cables goes to the Search terminals located in the corridor, and to the Lender's PC located in the Lending area.

Analizando la integración con el sistema

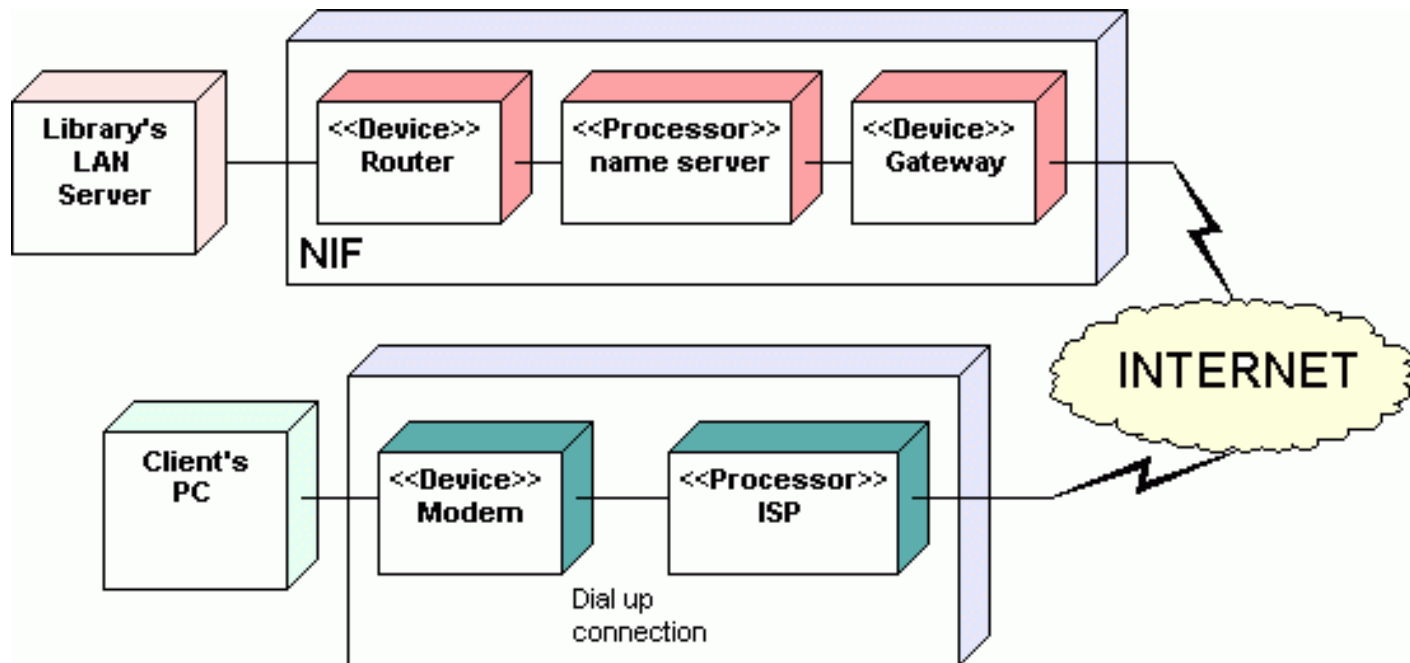
When we have a physical architecture in front of us, let's look at possibility of implementing an Internet interface on the Server. This interface will be used locally from the Search terminals, from Lender's PC and from DataEntry PCs and through the Internet from member's homes or offices. Therefore a new hardware must be implemented in the library's LAN.

Library's LAN will have the same structure. New hardware will be added only to the Server's side, and that is a Network Interconnection Facility (NIF). The NIF consists of a *name server* (a database that validates connections), a *router* (a device for linking networks together), and a *gateway* (a device for translating information from one communication protocol to another). The gateway allows Internet members to access the Digital Library from home or from their office.

On the member's side only a computer with Internet connection, and Internet browser is necessary. Also a member must have an alias and password to make access to the Digital Library's Web page.

Analizando la integración con el sistema

To illustrate the deployment, the System Engineer delivers the deployment diagram shown on the following picture:



The Deployment diagram for the Digital Library system

Diseñando la vista (GUI)

You've come through a lot of use-case driven analysis. Now you're going to look some of aspects of system analysis. They are traceable to use cases, and both are extremely important to the final product. Graphical User Interfaces (GUIs) determine system usability. Let's dive into GUI design process.

Diseñando la vista (GUI)

Some General Principles of GUI Design

User interface design, equals parts art and science, draws upon the vision of the graphic artist, the findings of the human factors researcher, and the intuitions of the potential user. Some general principles in GUI design are:

- 1. Understand what the user has to do. User interface designers typically perform a *task analysis* to understand the nature of the user's work. Our use case analysis roughly corresponds to this.
- 2. Make the user feel in control of the interaction. Always include the capability for the user to cancel an interaction after it's started.
- 3. Give the user multiple ways to accomplish each interface-related action (like closing a window or a file) and forgive user errors gracefully.

Diseñando la vista (GUI)

Some General Principles of GUI Design

- 4. Because of cultural influences, our eyes are drawn to the upper-left corner of a screen. Put the highest priority information there.
- 5. Take advantage of spatial relationships. Screen components that are related should appear near one another, perhaps with a box around them.
- 6. Emphasize readability and understanding. Use active voice to communicate ideas and concepts.
- 7. Limit the number of colors you use. Limit that number severely. Too many colours will distract the user from the task at hand.
- 8. If you're thinking of using color to denote meaning, remember it's not always easy for a user to see an association between a colour and a meaning.

Diseñando la vista (GUI)

Some General Principles of GUI Design

- **9.** As is the case with the color, limit your use of fonts. Avoid italics and ornate fonts.
- **10.** Try to keep components (like the buttons and list boxes) the same size as much as possible. If you use different-size components, a multiplicity of colors, and a variety of fonts, you'll create a patchwork that GUI specialists call a *clown-pants design*.
- **11.** Left-align components and data fields - line them up according to their left-side edges. This minimizes eye movements when the user has to scan the screen.
- **12.** When the user has to read and process information and then click a button, put the buttons in a column to the right of the information or in a row below and to the right of the information. This is consistent with the natural tendency to read left to right. If one of the buttons is a default button, highlight it and make it the first button in the set.

Diseñando la vista (GUI)

The GUI JAD Session

For this session, you recruit potential users of the system. For the Digital Library we'd recruit members, lenders and data entry clerks. The users' participation in the session is a two-part affair. In the first part, they derive the user interface screens. In the second, they approve prototypes generated by the development team.

- How the users derive the screens? The facilitator suggests a use case to start from, and the users discuss ways to implement that use case via the system. When they're ready to start talking at the level of a specific screen, the users work with paper mock-ups. The facilitator provides a large sheet of easel paper in landscape view to represent the screen. Post-it note stickies represent the GUI components. The users' task is to work as a group to position the components appropriately.

Diseñando la vista (GUI)

The GUI JAD Session

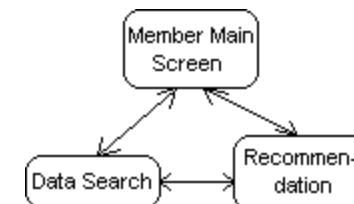
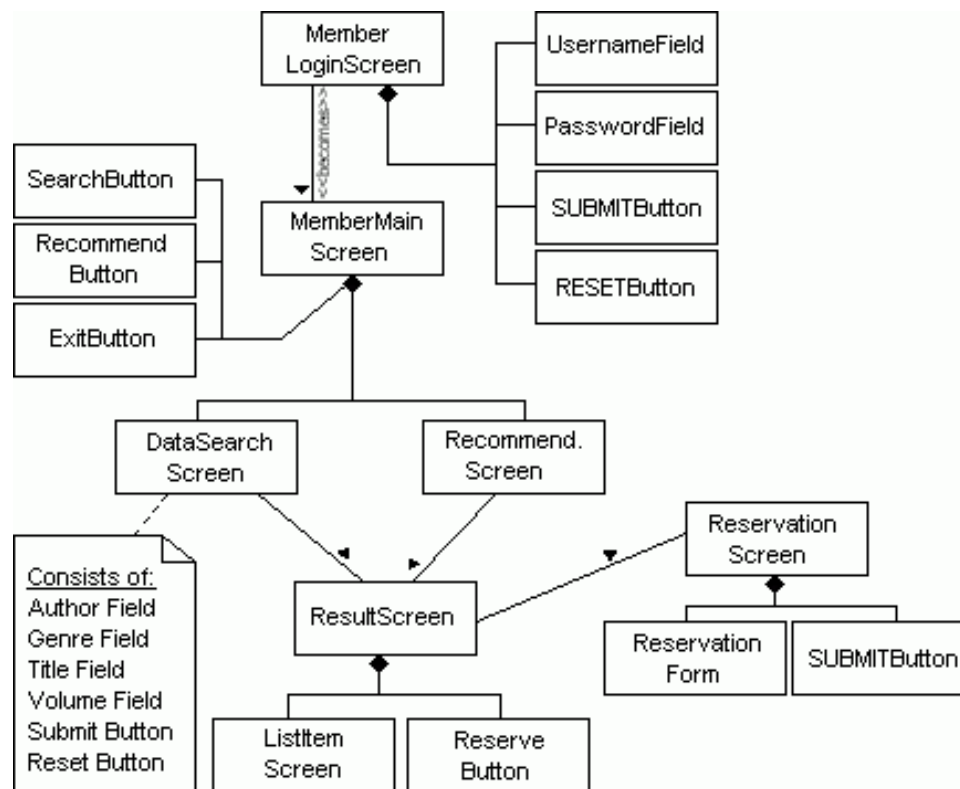
- When they reach agreement on which components should be on a screen and where those components should be located, development team members create prototype screens. As they work, they use appropriate GUI principles. Then, they present those screens on computers, and the users make any necessary modifications.

Diseñando la vista (GUI)

- It's your turn to draw out the users' screens. Let your imagination and GUI principles guide you when constructing these screen shots. To make your work easier a set of UML diagrams for the member, lender and data entry clerk screens are given to you. Look at these diagrams and draw out necessary screen shots. DataEntry clerk's diagrams are too large and complex, and if you want you may draw out these screen shots or not.

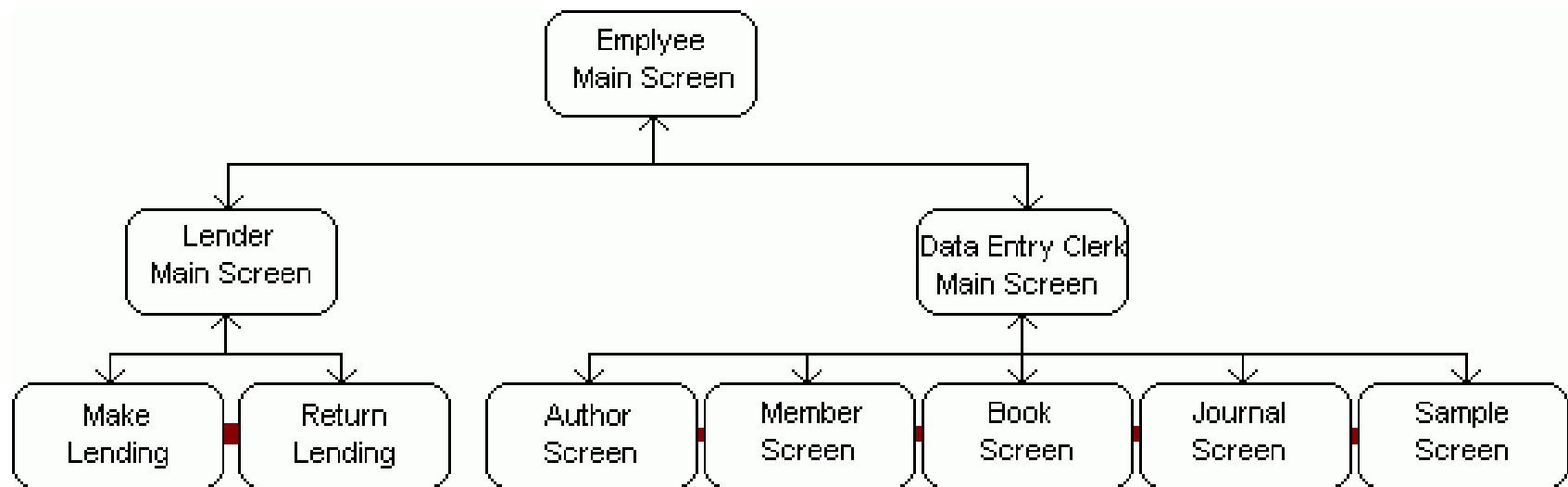
Diseñando la vista (GUI)

The UML state diagram for the Member interface



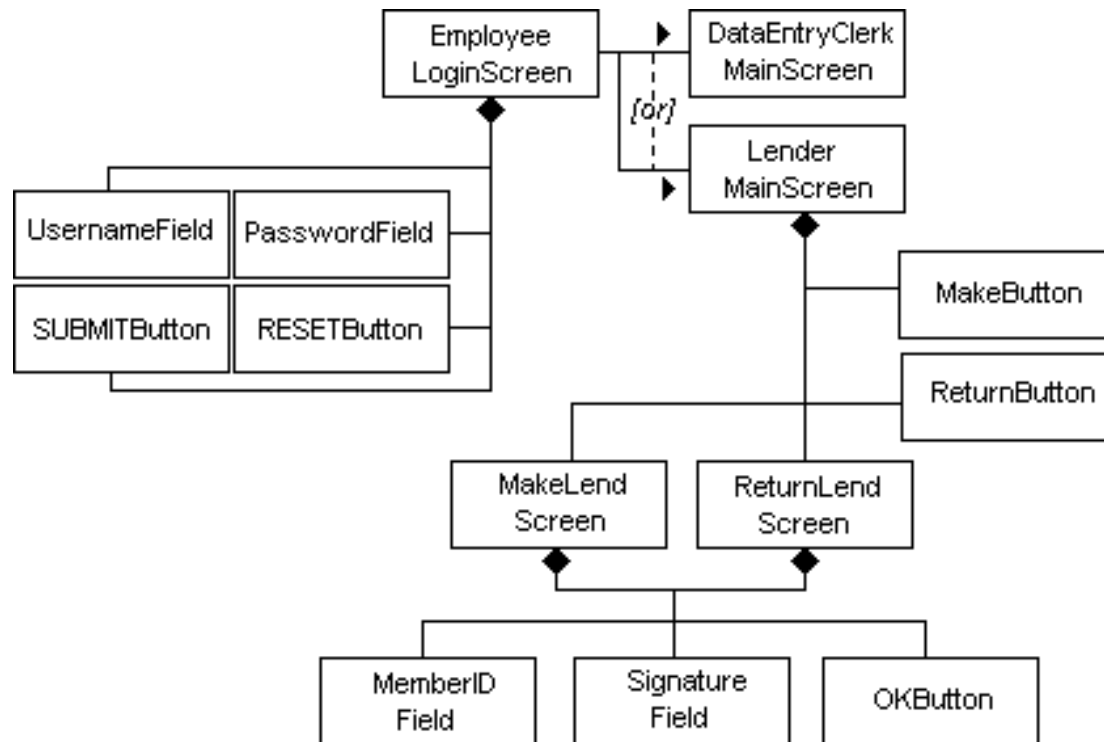
The UML composite diagram for the Member screen hierarchy

Diseñando la vista (GUI)



The UML state diagram for high-level screen flow in the Employee interface

Diseñando la vista (GUI)



The UML composite diagram for the Employee-Lender screen hierarchy

Diseñando la vista (GUI)

Something About Implementation of Digital Library

- When all appropriate analysis and design parts of GRAPPLE are complete, the team compiles its result into a design document and had off copies to the client and to the programmers. It then falls to the programmers to start turning the design into code. After writing the code it will be tested, rewritten according to the results of the tests, and retested - a process that will continue back and forth until the code passes all tests. The use case analysis forms the basis of the tests.

Diseñando la vista (GUI)

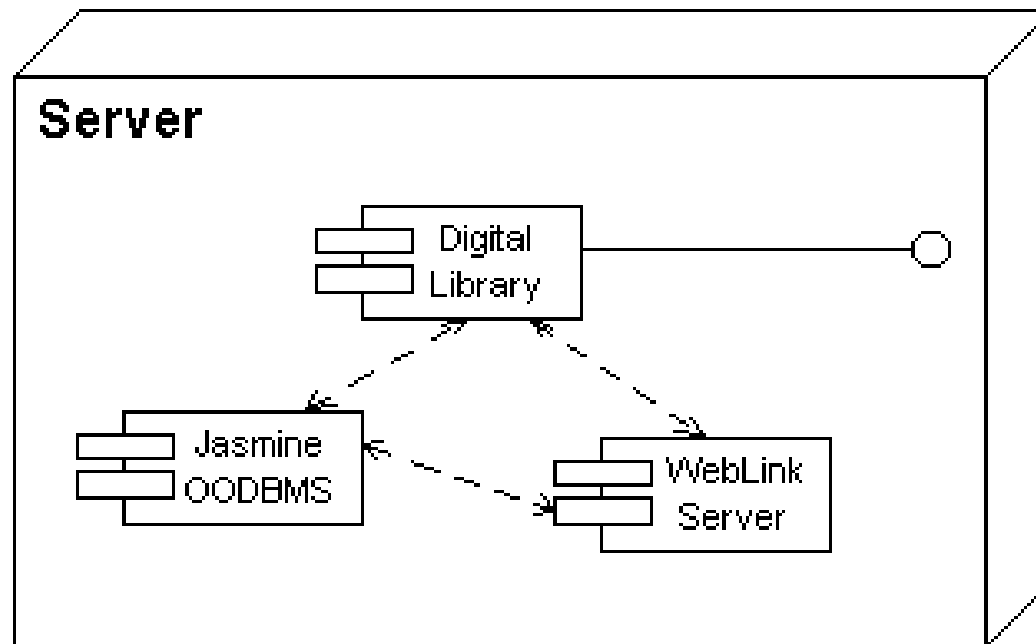
Something About Implementation of Digital Library

- The one possible implemented Digital Library that you'll visit after today's workshop, is coded in Jasmine OODBMS. Jasmine is used because it is Object Oriented language, has possibility of adding Multimedia data into your database, and allows you to create stand alone applications, or applications that have Web interfaces. Other advantages of using Web interface are client software shouldn't be installed to the client's machine (client only should have an Internet browser) and Web software doesn't depend on the hardware platform that runs on.

Diseñando la vista (GUI)

Something About Implementation of Digital Library

The Server side in the deployment diagram developed for the purpose of Digital Library, now will look like



:* The Jasmine OODBMS and WebLink are parts from the Jasmine package. With Jasmine OODBMS we created necessary classes, and with WebLink we implemented the web interface.

Diseñando la vista (GUI)

Something About Implementation of Digital Library

- Document specialists begin creating documentation for the system, and they create training materials as well. A good document creation effort should proceed like a good system development effort, with careful planing, analysis, and testing, and should begin early in the development process.

Conclusión

The main idea of modeling one system is to focus intense efforts on analysis and design so that implementation confronts as few challenges as possible and the result of the project is a system that fully meets the client's needs.

Preguntas y respuestas

1 What we want to show for every scenario in the use case, to perform use case analysis?

- For each scenario in use case, we'll want to show:
 - A brief description of the scenario
 - Assumptions for the scenario
 - The actor who initiates the use case
 - Preconditions for the use case
 - System-related steps in the scenario
 - Postconditions when the scenario is complete
 - The actor who benefits from the use case.

2 What are the parts of a typical use case diagram?

- The parts of every use case diagram are:
 - The initiating actor
 - The use case and
 - The benefiting actor.

Preguntas y respuestas

- 3 Why would the original use cases fail to capture all the nuances in the first place?**
 - Because they're results of JAD sessions with system users, not system developers. You'll notice all the additions and changes were system-related, not business-related. After you finish the sessions with the potential users and have a chance to analyze the use cases, it's not uncommon that modifications like these emerge.
- 4 In a sequence diagram, how do you show an "activation", and what does it represent?**
 - An activation is represented as a small narrow rectangle on an object's lifeline. It represents the time period during which the object performs an action.
- 5 What UML diagrams are useful for describing user interfaces?**
 - State diagrams and Composite diagrams (these are parts of Class diagrams) are appropriate when designing the possible user interfaces.

Bibliografía

- Grady Booch, James Rumbaugh, Ivar Jacobson. **The unified modeling language user guide**. Addison-Wesley, 1999
- Grady Booch, James Rumbaugh, Ivar Jacobson. **The unified modeling language reference manual**. <http://www.omg.org/technology/uml/>
- Muller Pierre-Alain. **Instant UML**. Wrox Press, 1997
- Perdita Stevens, Rob Pooley. **Using UML, Software engineering with objects and components**. Addison-Wesley, 2000
- Artículos del Object Management Group.
<http://www.omg.org/technology/documents/formal/>
- Martin Fowler, Kendall Scott. **UML Distilled, Applying the standard object modelling language**. Addison-Wesley, 1997.
- Popkin Software. **Modeling Systems with UML**. 1998

Direcciones web interesantes

- <http://www.rational.com/university/description/13.jsp>
- <http://www.rational.com/uml/resources/documentation/index.jsp>
- <http://www.omg.org/technology/uml/>
- <http://www.omg.org/technology/documents/formal/>
- <http://www.can.ibm.com/services/learning/course/N9010E.html>
- <http://www.gendev.com/education/read/oo-business-process-modeling.html>
- <http://www.essentialstrategies.com/publications/modeling/uml.htm>
- <http://www.sparxsystems.com.au/>