Introducción a UML

Lenguaje para modelar objetos

Juan Manuel Cueva Lovelle
Catedrático de E.U. de Lenguajes y Sistemas Informáticos
Departamento de Informática
Universidad de Oviedo (España)
Octubre 1999

Tabla de contenidos

Bibliografía

- [Bock/Odell 94] C. Bock and J. Odell, "A Foundation For Composition," *Journal of Object-oriented Programming*, October 1994.

- [Booch 94] G.Booch. *Object-oriented analysis and design with applications.* Benjamin Cummings (1994). Versión castellana: *Análisis y diseño orientado a objetos con aplicaciones.* 2ª Edición. Addison-Wesley/ Díaz de Santos (1996).

- [Booch 99] G.Booch., J. Rumbaugh, I. Jacobson *The unified modeling language. User guide.* Addison-Wesley (1999). Versión castellana: *El lenguaje Unificado de Modelado.*Addison-Wesley (1999).

- [Cook 94] S. Cook and J. Daniels, *Designing Object Systems: Object-oriented Modelling with Syntropy,* Prentice-Hall Object-Oriented Series, 1994.

- [Eriksson 98] H-E Eriksson & M. Penker. *UML Toolkit. Wiley, 1998.*

- [Fowler 97] M. Fowler with K. Scott, *UML Distilled: Applying the Standard Object Modeling Language*, ISBN 0-201-32563-2, Addison-Wesely, 1997.

- [Jacobson 92] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard. *Object-Oriented Software Enginneering. A use Case Driven Approach.*, ISBN 0-201-54435-0, Addison-Wesely, 1992.

- [Lee 97] R. C.Lee & W. M. Tepfenhart. *UML and C++, Prentice-Hall, 1997*

- [Rumbaught 91] Rumbaught J., Blaha M., Premerlani W., Wddy F., Lorensen W. *Object-oriented modeling and design.* Prentice-Hall (1991). Versión castellana: *Modelado y diseño orientado a objetos. Metodología OMT.* Prentice-Hall (1996)

- [UML] UML Notation Guide, Version 1.1, Sep-97, www.rational.com/uml

Método y metodologías

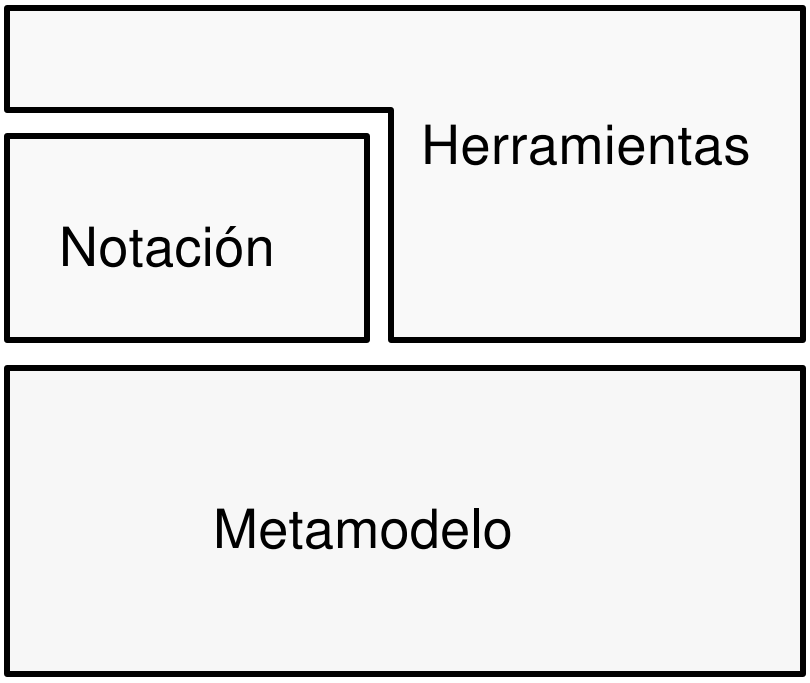
- *Un **método** es un proceso disciplinado para generar un conjunto de modelos que describen varios aspectos de un sistema de software en desarrollo, utilizando alguna notación bien definida*
- *Una **metodología** es una colección de métodos aplicados a lo largo del ciclo de vida del desarrollo de software y unificados por alguna aproximación general o filosófica*
  - La mayor parte de las metodologías puede catalogarse en uno de los grupos siguientes:
    - Diseño estructurado descendente
      - Yourdon y Constantine
      - Wirth
      - Dahl, Dijkstra y Hoare
    - Diseño dirigido por datos
      - Jackson
      - Warnier y Orr
    - Diseño orientado a objetos son las que siguen el modelo de objetos
      - Booch
      - OMT (Rumbaugh et al.)
      - Objectory (Jacobson et al.)
      - Schlaer-Mellor
      - Coad/Yourdon
      - Fusion (Coleman et al.)

Notación

- *Una **notación** es un conjunto de diagramas normalizados que posibilita al analista o desarrollador el describir el comportamiento del sistema (análisis) y los detalles de una arquitectura (diseño) de forma no ambigua*
  - La acción de dibujar un diagrama no constituye ni análisis ni diseño
  - Una notación no es un fin por si misma
  - El hecho de que una notación sea detallada no significa que todos sus aspectos deban ser utilizados en todas las ocasiones
  - La notación son como los planos para un arquitecto o un ingeniero
  - Una notación no es más que un vehículo para capturar los razonamientos acerca del comportamiento y la arquitectura de un sistema
  - Las notaciones deben ser lo más independientes posibles de los lenguajes de programación, sin embargo facilita el proceso de desarrollo que exista una equivalencia entre la notación y los lenguajes de programación
  - El propósito de este tema es describir la sintaxis y semántica de la notación que se utiliza para el análisis y diseño orientado a objetos

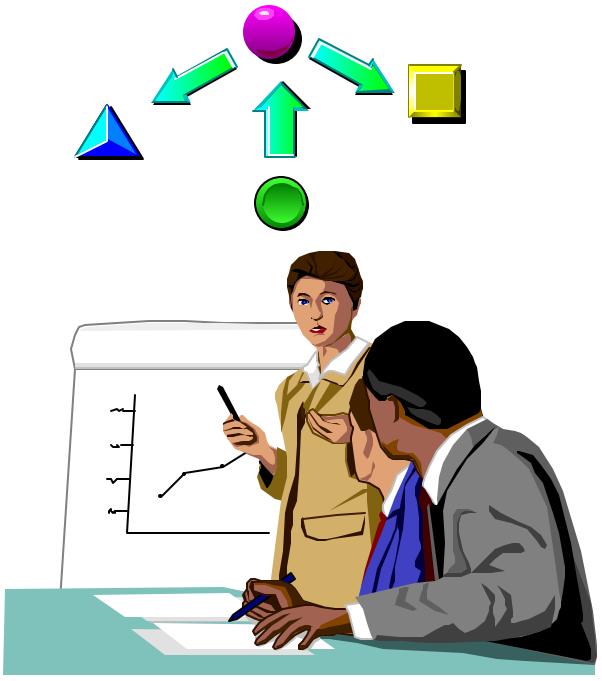
Modelos, metamodelos y herramientas

- *Un **metamodelo** es la descripción de un modelo*
- *Un **modelo** es el resultado del análisis y diseño*
- *Una **herramienta** es el soporte automático de una notación*

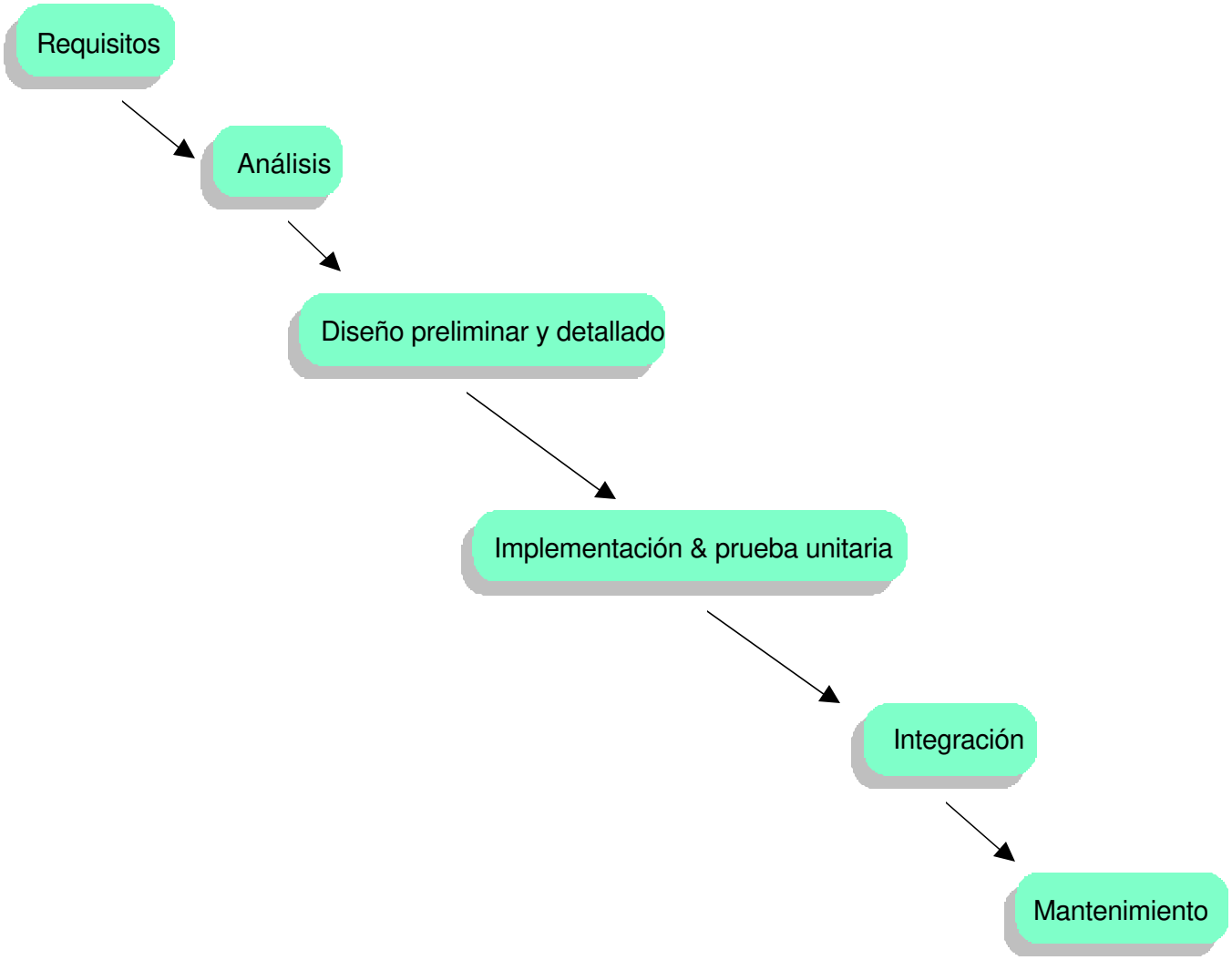


Comparación de metodologías

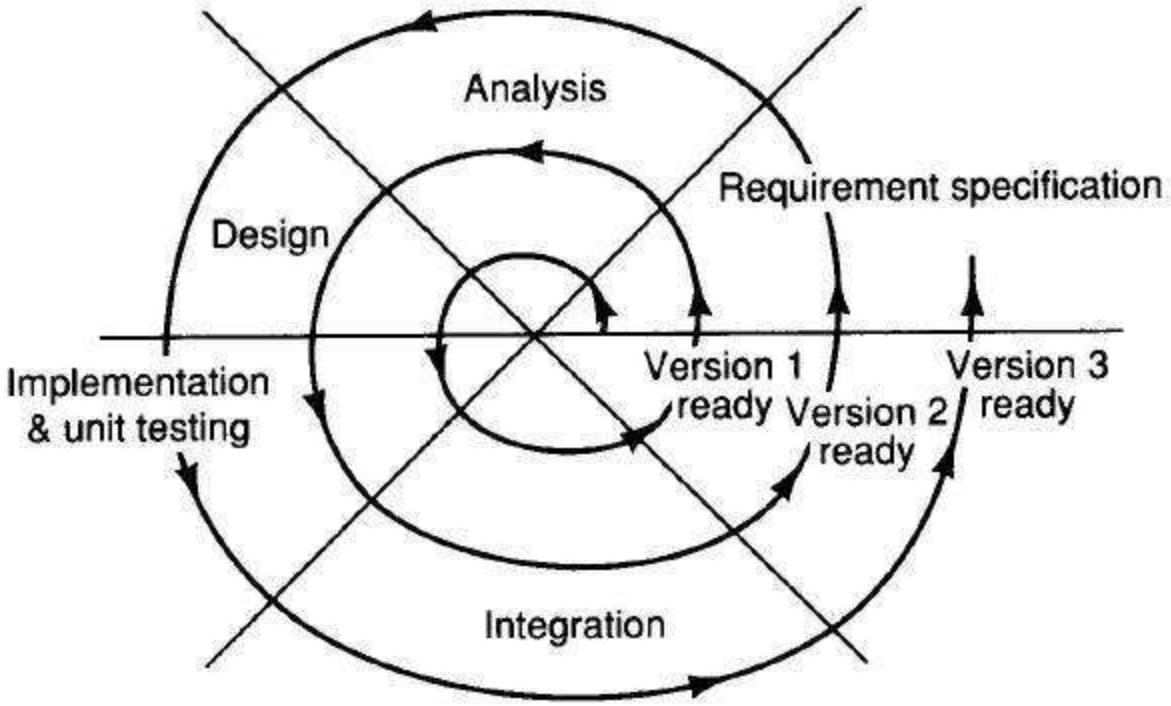
- Con el paso del tiempo las metodologías de análisis y diseño orientado a objetos han ido convergiendo en
  - Conceptos de modelado similares
  - Procesos similares
- Las principales diferencias están en la notación
  - Diferentes iconos para la representación de los artefactos del diseño
  - Diferentes diagramas para expresar las relaciones entre clases y objetos
- La experiencia indicará que es lo que se debe utilizar y lo que se debe dejar de lado



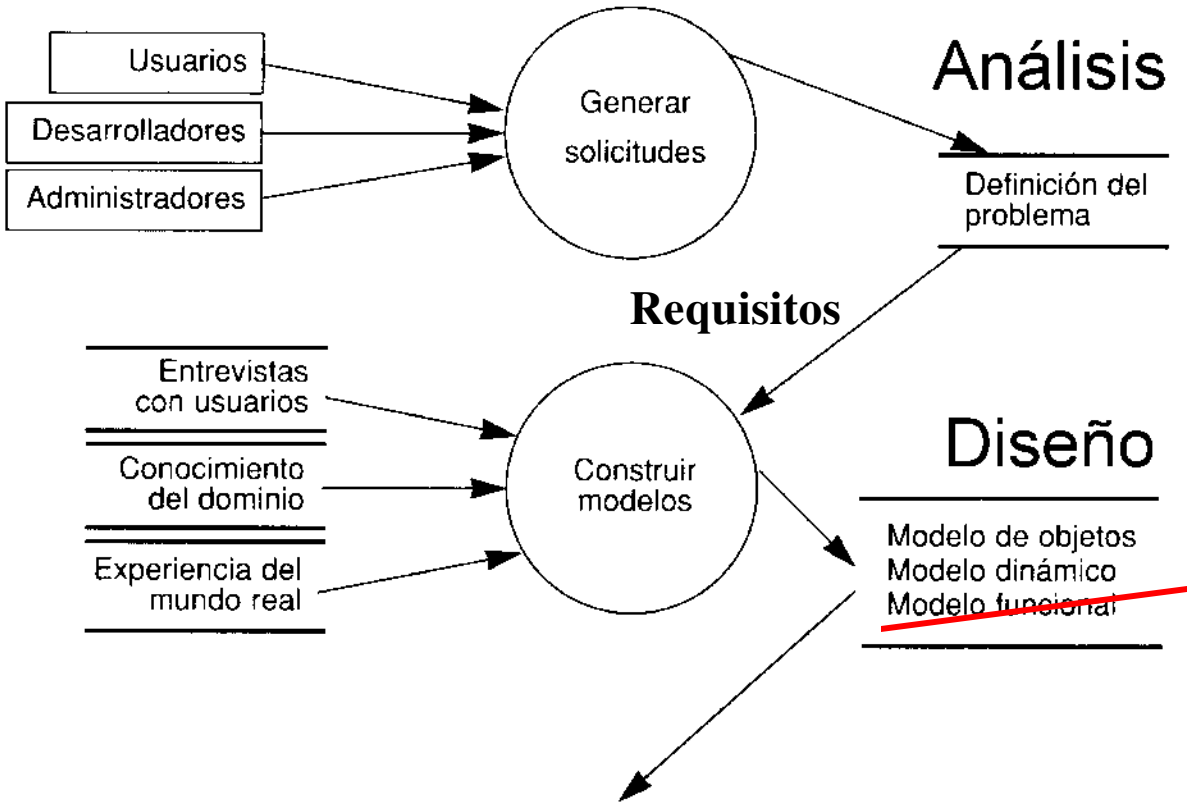
Desarrollo de software
Modelo en cascada



Desarrollo de software
Modelo en espiral



Análisis y diseño orientado a objetos



Análisis Orientado a Objetos

Preguntas que se deben plantear

- ¿Cuál es el comportamiento que se desea en el sistema?
- ¿Qué objetos existen en el sistema?
- ¿Cuáles son las misiones de los objetos para llevar a cabo el comportamiento deseado del sistema?

Diseño Orientado a Objetos

Preguntas que se deben plantear

Modelo lógico

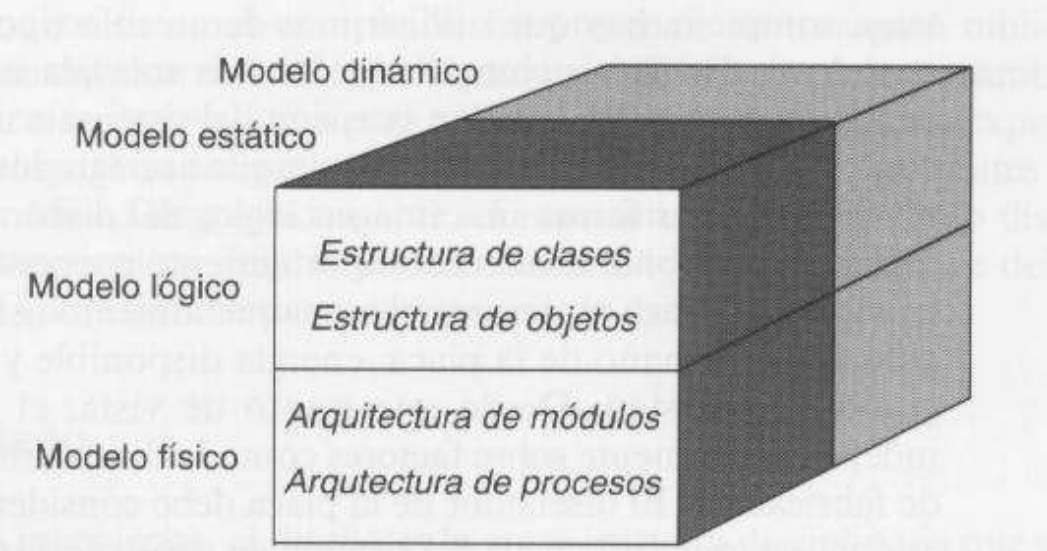
- ¿Qué clases existen y como se relacionan estas clases?
- ¿Qué mecanismos se utilizan para regular la forma en que los objetos colaboran?

Modelo físico

- ¿Dónde debería declararse y construirse cada clase y objeto?
- ¿A qué procesador debería asignarse un proceso, y para un procesador dado, cómo deberían planificarse sus múltiples procesos?

El modelo del desarrollo orientado a objetos

- Se definen distintas dimensiones del modelo
  - *El **modelo estático** describe la estructura estática del sistema*
    - *El **modelo lógico** define la arquitectura del sistema desde el punto de vista de las abstracciones principales y mecanismos*
      - *Diagramas de clases*
      - *Diagramas de objetos*
    - *El **modelo físico** define la arquitectura del sistema desde el punto de vista de la composición concreta hardware y software*
      - *Diagrama de módulos*
      - *Diagrama de procesos*
  - ***Modelo dinámico** describe la evolución dinámica y las interacciones entre objetos*
    - *Diagramas de transición de estados*
    - *Diagramas de interacción o de seguimiento de sucesos*
  - *OMT añade el **modelo funcional (eliminado posteriormente)***
    - *Diagrama de flujo de datos*
- Por cada dimensión se define una serie de diagramas que denotan una vista de los modelos del sistema
- Cuando el modelo es estable cada uno de los diagramas permanece semánticamente consistente con el resto de los diagramas



Los modelos del desarrollo orientado a objetos.

Proceso de Análisis y Diseño Preliminar

Aunque el proceso es iterativo los pasos fundamentales son los siguientes:

1. Título de la aplicación

El título de una aplicación debe reflejar de la mejor forma posible sus fines y su funcionalidad

2. Documentos de análisis

- Son la documentación que aporta el cliente que encarga la aplicación
- También es la documentación elaborada de forma informal en reuniones de trabajo para entender las solicitudes del cliente

3. Especificación de Requisitos o Requerimientos

- Es la especificación más técnica y elaborada de los documentos de análisis
- Es importante codificar los requisitos para poder seguirlos a lo largo del proceso de construcción del software

4. Diagramas de Casos de Uso

- Es un diagrama que muestra sistemas, casos de uso y actores
- Es una documentación que describe cada caso de uso, cada sistema y cada actor
- Es importante codificar cada caso de uso
- Los casos de uso sólo muestran aspectos muy generales

5. Escenarios y sub-escenarios

- A cada caso de uso le corresponden varios escenarios donde se pueden mostrar los detalles
- Cada escenario puede dividirse en sub-escenarios para bajar más el nivel de detalle
- Los escenarios se codifican siguiendo los valores de los casos de uso

6. Diagramas de Secuencia

- Se corresponden con los escenarios y sub-escenarios pero con mucho más detalle
- Siguen la misma codificación que los escenarios y sub-escenarios
- Algunos diagramas de secuencia pueden refinarse más en la fase de diseño detallado

7. Diccionario de datos

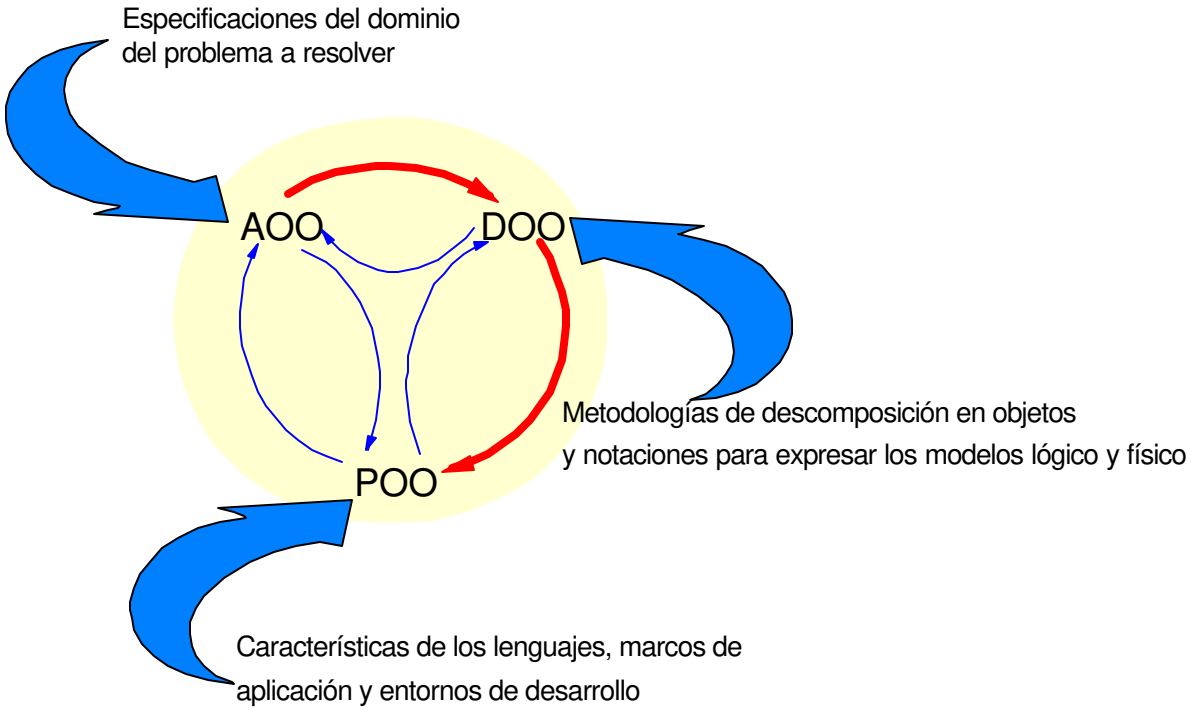
- Contiene todas las clases
- Se pueden ir definiendo los miembros de las clases (datos y métodos)
- Se iré refinando paso a paso en cada iteración
- Las herramientas lo van construyendo automáticamente
- **También se pueden utilizar fichas CRC para obtener el diccionario de datos**

Proceso de Diseño Detallado

Es continuación del análisis y diseño preliminar, pero como es un proceso iterativo en muchos casos será necesario volver al análisis y diseño preliminar

8. **Diagramas de Colaboración**
9. **Diagramas Estáticos**
10. **Diagramas de Actividad**
11. **Diagramas de Estados**
12. **Diagramas de implementación**

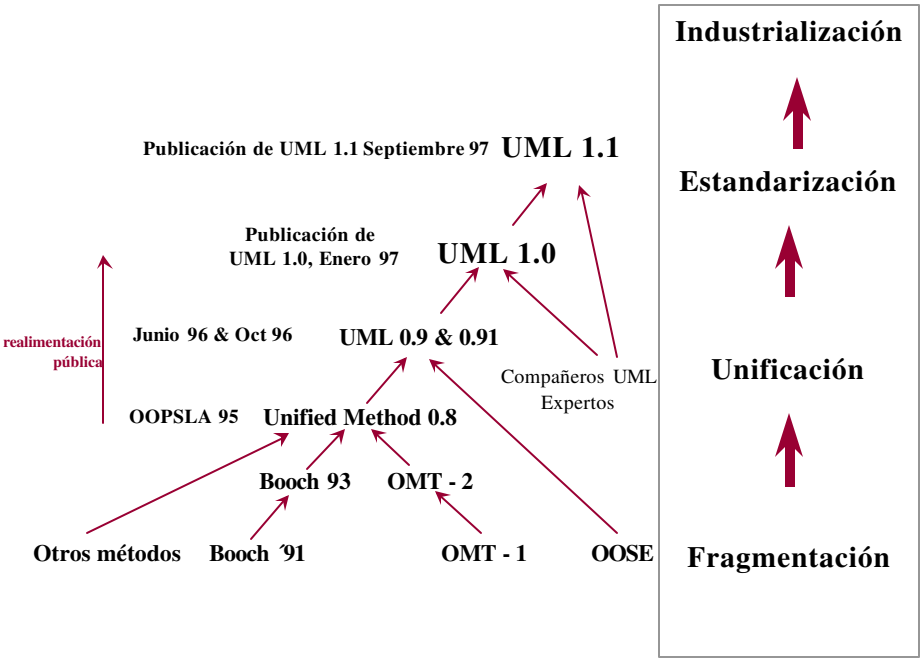
• En algunos casos en el **diseño preliminar** es necesario hacer algunos diagramas del diseño detallado pero de forma sencilla
• El **diseño preliminar** puede ser interesante para poder dar una idea de los costes y de los tiempos de desarrollo de una aplicación



Unified Modeling Language (UML)

[UML]

- UML es el sucesor de la ola de métodos de A y DOO que aparecieron a finales de los 80 y principios de los 90
- UML unifica principalmente los métodos de Booch, Rumbaught (OMT) y Jacobson. Pero pretende dar una visión más amplia de los mismos
- UML está en proceso de estandarización por el OMG (Object Management Group) [OMG]
- UML es un **lenguaje de modelado**, no un método.
- Un método incluye
  - ***Lenguaje de modelado***: *Es la notación (en su mayoría gráfica) que utilizan los métodos para expresar los diseños.*
  - ***Proceso****: Son los pasos que se aconsejan dar para realizar un diseño*



- # Ejemplo de título
  - **TRASGU: Gestión de hardware y software**
    - Consta de nombre propio
    - Breve descripción de la aplicación

- # Ejemplo de Documento de Análisis

Se debe realizar un sistema capaz de mantener una base de datos con todos los equipos hardware y software de una empresa, de manera que se pueda obtener información acerca del número de licencias instaladas y de los equipos en los que están instaladas dichas licencias.

Además debe ser posible controlar el hardware, las modificaciones efectuadas en los equipos, las averías de dichos equipos, la composición de cada uno de los ordenadores y el software que está instalado en ellos.

Por otro lado cada equipo y cada software que posee la empresa tiene asociados una serie de manuales de los que es necesario seguir la pista, pudiendo, en cada momento, saber qué manuales tiene cada equipo y también cada programa.

Por tanto existen tres elementos importantes implicados en el sistema:

1.El software.

2.El hardware.

3.Los manuales.

Es necesario seguirles la pista a estos tres elementos y saber en todo momento las relaciones entre ellos para poder localizar, mediante el ordenador el manual de un componente instalado en un ordenador.

Para el Software es necesario saber el nombre del producto, la versión, la marca o casa que lo fabrica, la fecha de compra, el precio de compra, el proveedor, el soporte (disquetes, CD-ROM, etc.), el número de elementos del soporte, la localización física del soporte, el número de instalaciones, los equipos en los que está instalado, el número de licencias adquiridas, los manuales que acompañan el producto y la localización física de dichos manuales.

Las localizaciones físicas pueden ser sustituidas por los códigos si se codifican tanto los soportes físicos como los manuales.

El sistema debe ser capaz de contestar a las preguntas:

1.Licencias existentes, en uso y necesarias (si procede) de cada una de las aplicaciones que se estén usando en la empresa.

2.Ordenador u ordenadores (si hay varios) en que reside una aplicación.

3.Composición del paquete original (disquetes, CD-ROM, etc. y manuales).

4.Proveedor que sirvió el programa, fecha y precio de adquisición.

Un detalle importante a tener en cuenta es que existe la posibilidad de que exista software llave-en-mano, y en este caso además hay que saber si se dispone o no de los códigos fuente, la casa que lo desarrolló, quién posee los derechos de copia, el precio, el tiempo de desarrollo y el nombre de la persona responsable del proyecto.

Los software se quedan obsoletos y, por tanto, es necesario actualizarlos. Se debe tener en cuenta que es necesario que el sistema ofrezca una reseña histórica del producto (versiones anteriores) y por tanto es necesario saber el estado de cada uno de ellos (activo, actualizado, desechado, en preparación, pedido, etc.)

En el caso de los antivirus y otros programas similares, es necesario obtener regularmente una adaptación, por lo que es importante que el sistema nos avise de la inminencia de la caducidad de dichos sistemas.

Ejemplo de Documento de Análisis (continuación ...)

De cada ordenador se necesita saber su composición (monitor, teclado, ratón y unidad central). De esta última es necesario saber su composición (VGA, disco duro, disquete, placa madre, procesador(es), memoria RAM, memoria caché, etc.).

Cada uno de los cuatro componentes principales estará codificado adecuadamente para permitir el intercambio de dichos equipos entre los diferentes puestos de ordenador, de manera que la asociación no sea fija. De ellos es necesario saber (cuando exista) la marca, el modelo, el número de serie, y otras características particulares (por ejemplo, del monitor la resolución, si es o no *Energy*, etc.)

Además de ordenadores existen otros equipos: impresoras, plotters, scaners y unidades de almacenamiento (ZIP, CD y Magneto-ópticos) de los cuales es necesario saber, al menos, la marca, el modelo, el número de serie y una breve descripción de sus características.

De cada uno de los equipos es necesario tener un reseña histórica de sus averías y cambios, así como una estimación de su precio (en función del precio de compra de cada uno de sus componentes).

En el caso de los ordenadores es necesario saber el software que tienen instalado (comenzando por el sistema operativo) y debe ser posible seguir la pista de los manuales de cada una de sus partes componentes.

De los manuales sólo es necesario controlar su código, su ISBN (si lo posee), su precio (si es aparte del paquete) y su título. Pero es imprescindible poder obtener información del software y hardware que está relacionado con ellos.

Los manuales deben de ser actualizados según se vaya cambiando el software y el hardware .

El programa debe ser capaz de gestionar el sistema desde diferentes puntos de vista: responsable de informática, mantenimiento y usuario.

El responsable de informática es el encargado de comprar todos los componentes (equipos, software y manuales). Además da estos equipos de alta y debe poder apoyarse en el sistema para gestionar los pedidos.

Para esta última labor debe poder anotar en el sistema que ha pedido un componente a un proveedor (por tanto que está pendiente de recibir), confirmando en la recepción este pedido, además tendrá la capacidad de poder anular un pedido o si se da el caso anularlo

Ejemplo de Documento de Análisis (continuación ...)

Además debe poder obtener informes de inventario de los equipos tasados por el precio de compra menos una amortización del 25% anual (que dejaría al equipo sin valor pasados cuatro años) para los procesadores y del 10% anual para el resto de los equipos.

Además debe poder obtener informe de la composición de cada equipo, del estado de disponibilidad de cada uno de ellos y de el estado con respecto a la garantía del equipo.

El responsable de informática es la única persona que puede dar de alta, modificar y dar de baja los equipos.

La baja de un equipo se dará en el momento en que se avise de la avería y si el equipo no tiene arreglo lo daremos de baja permanente.

Además debe poder obtener toda la información que tienen el resto de los usuarios del sistema (responsable de mantenimiento y usuarios), y tendrá acceso a un buzón de sugerencias sobre el sistema.

El responsable de informática se encarga además de reservar un equipo cuando se solicita por cualquier usuario, para ello tiene que obtener los informes de disponibilidad y composición de equipos.

El responsable de mantenimiento debe poder anotar en todo momento las averías de cada equipo (fecha, hora de comienzo, hora de fin, nombre del mecánico y descripción). Debe poder anotar que un equipo no tiene reparación. Además debe poder obtener informes acerca del tiempo de inactividad de los equipos y acceso al buzón de sugerencias.

El usuario debe poder obtener información de los manuales, software y hardware asociado y disponibilidad de un equipo en concreto . Tendrá acceso al buzón de sugerencias.

Se debe tener en cuenta que:

* El sistema trabajará en un entorno multiusuario.
* Las bases de datos serán un modelo estándar.
* Cada equipo estará compuesto por un monitor, un teclado, un ratón, y una unidad central.
*La unidad central estará formada por una serie de componentes que se describirán de manera textual en un campo al efecto.

Ejemplo de Especificación de Requisitos

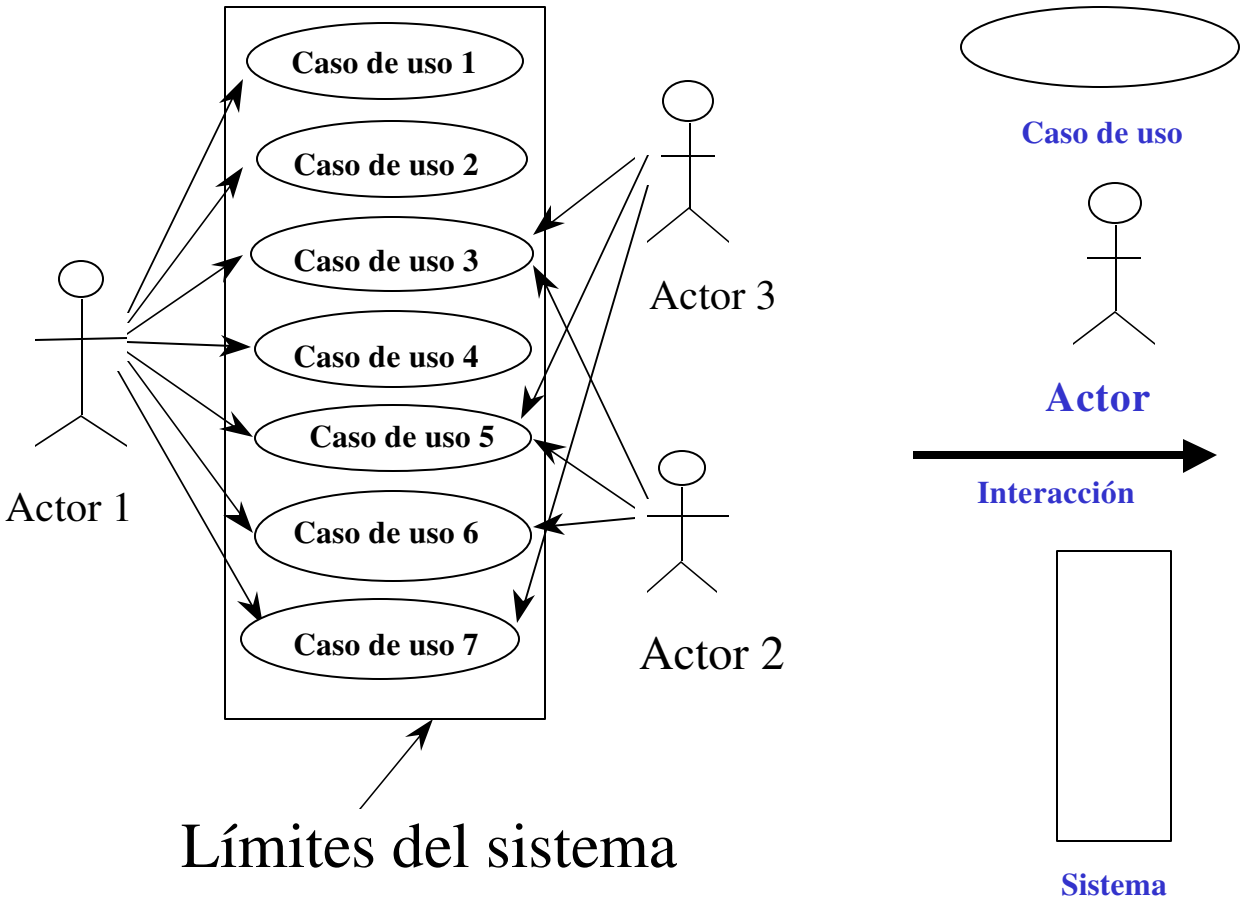
- R1. El sistema gestiona una base de datos con todos los equipos hardware y programas software de la empresa
- R2. La gestión de equipos hardware incluirá las altas, bajas, modificaciones de los equipos, las averías, la composición de cada equipo y el software instalado
  - R2.1 Los manuales correspondientes a cada equipo y cada programa deben de poder localizarse
- R3. La gestión del software debe incluir altas, bajas, y consultas
  - R3.1 Sobre cada programa software deberá conocerse los manuales, soportes magnéticos (disquetes, CD-ROM), número de licencias, número de instalaciones, localización física de los manuales.
  - R3.2 Ligado a cada programa software debe conocerse el proveedor, fecha y precio de adquisición
  - R.3.3 Puede existir software llave-en-mano y deberá conocerse si se dispone del código fuente y donde esta situado, así como la empresa que lo desarrolló, los derechos de copia, el precio, el tiempo de desarrollo, la herramienta de desarrollo, y el nombre de la persona responsable de l desarrollo
- R4. Las localizaciones físicas de equipos, programas y manuales se codificarán
- ...

Análisis Orientado a Objetos

- **Casos de uso**
  - Un caso de uso es una técnica de modelado utilizada para describir lo que un nuevo sistema debe hacer o lo que un sistema existente ya hace.
  - Un modelo de casos de uso se construye mediante un proceso iterativo durante las reuniones entre los desarrolladores del sistema y los clientes (y/o los usuarios finales) conduciendo a una especificación de requisitos sobre la que todos coinciden.
  - Un caso de uso captura algunas de las acciones y comportamientos del sistema y de los actores
  - El modelado con casos de uso fue desarrollado por Ivar Jacobson [Jacobson 92]
- **Escenarios**
  - A veces se utiliza escenario como sinónimo de caso de uso
  - Sin embargo en UML un escenario se refiere a los pasos que se desarrollan dentro de un caso de uso
- **Fichas CRC**
  - Son muy útiles para la enseñanza del AOO, DOO y POO
  - No están contempladas en UML

Análisis mediante *Casos de Uso*

- El sistema que se desea modelar se representa encerrado en un rectángulo
- Los actores son los que interactúan con el sistema. Representan todo lo que necesite intercambiar con el sistema.
  - Un actor es una clase
- Se diferenciará entre actores y usuarios.
  - Un usuario es una persona que utiliza el sistema
  - Un actor representa el papel (rol) que una persona desempeña
  - Por ejemplo una persona puede ser usuario y administrador en un sistema, unas veces actuará como usuario y otras como administrador, pero deben contemplarse ambos actores.
- Los Casos de Uso es un camino específico para utilizar el sistema
- Para cada Caso de Uso, Actor y Sistema se realiza una descripción detallada
- Los Casos de Uso tan sólo indican opciones generales
- El diagrama de Casos de Uso es un diagrama sencillo que tiene como finalidad dar una visión global de toda la aplicación de forma que se pueda entender de una forma rápida y gráfica tanto por usuarios como por desarrolladores



Diagramas de casos de uso en UML

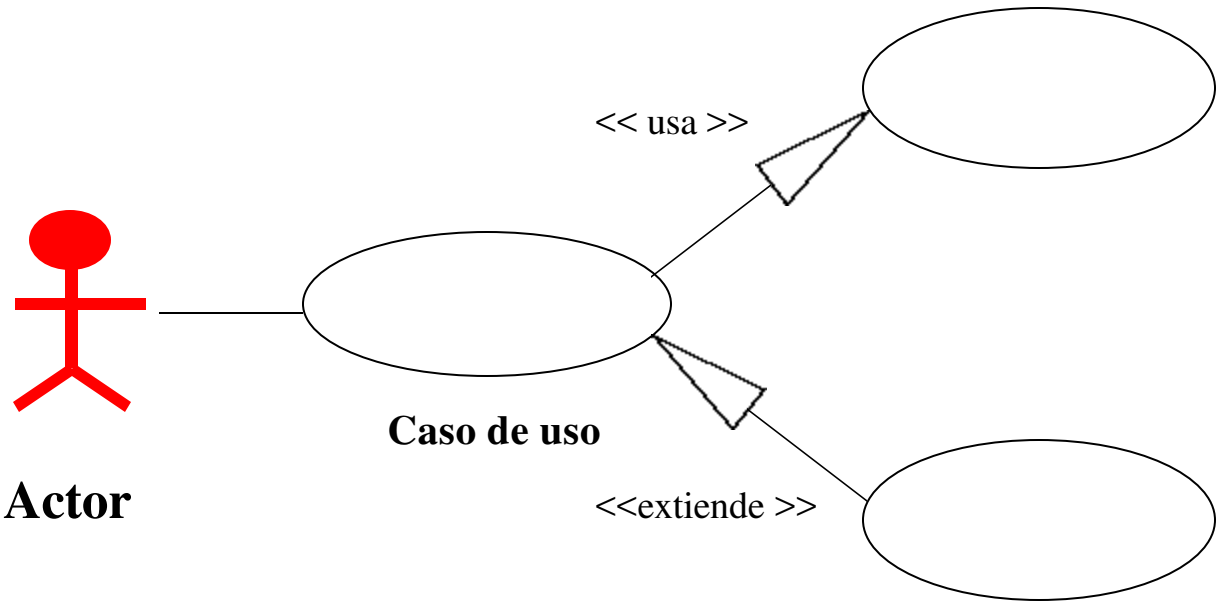
• El **diagrama de casos de uso** es parte de UML

• Un **caso de uso** es la típica interacción entre un usuario y un sistema informático

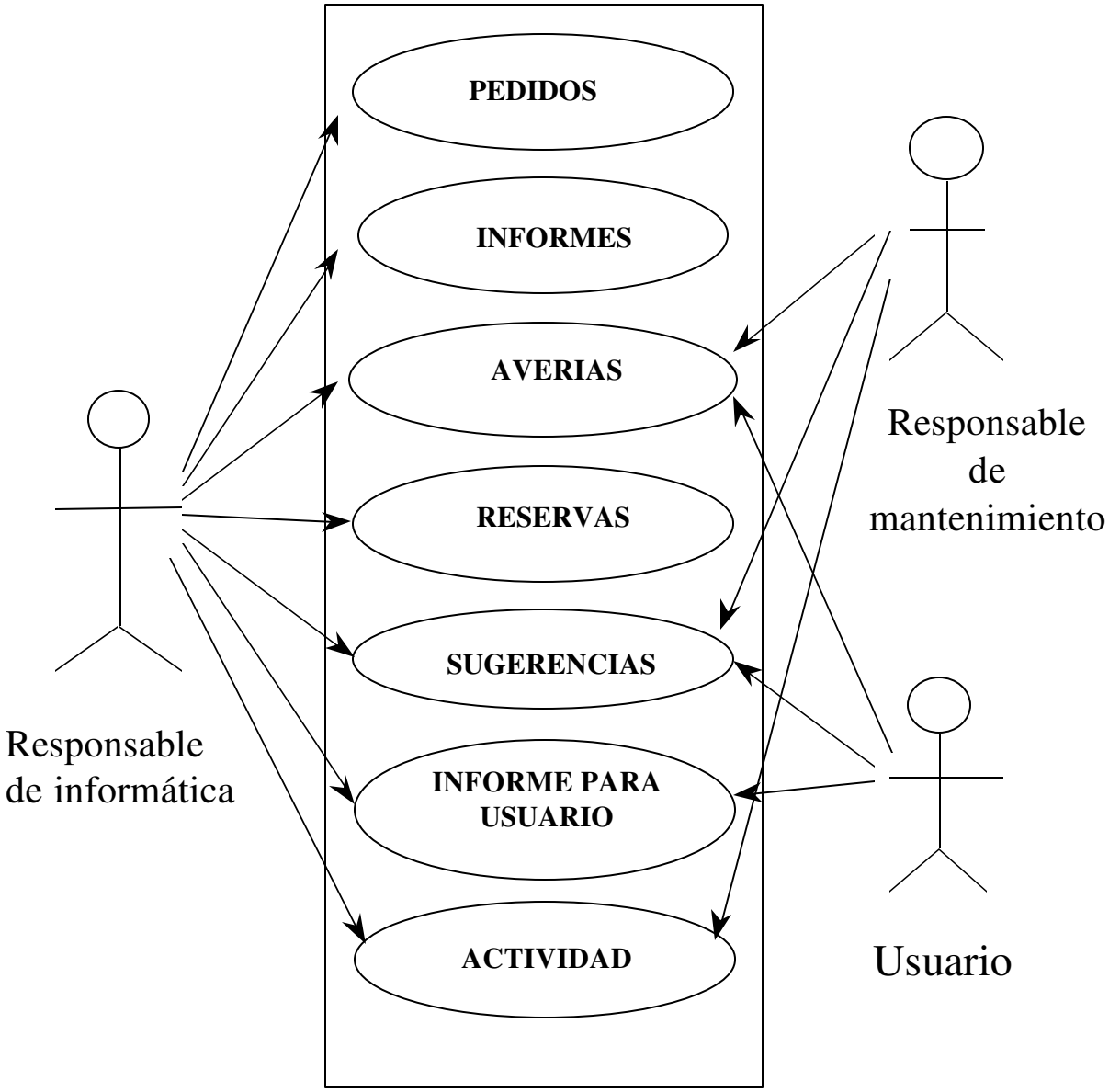
• Un **actor** es el papel que el usuario juega con respecto al sistema. Un actor no tiene que ser un humano, puede ser por ejemplo otro sistema externo que pide información al sistema actual

• La relación ***<<extiende>>*** se utiliza cuando un caso de uso es similar a otro caso de uso pero se le añade alguna característica nueva

• La relación ***<< usa >>*** se utiliza cuando se tiene una parte del comportamiento común a más de un caso de uso, y no se desea almacenar una copia en cada caso de uso de la descripción de este comportamiento.



Ejemplo de Casos de Uso



Descripción de los Casos de Uso

1.PEDIDOS

Escenario general donde se realizan todas las operaciones relativas a pedidos: hacer, recibir, anular y devolver pedidos. Todo es realizado por el Responsable de Informática.

2.INFORMES

Todos los informes que son necesarios para el funcionamiento de la empresa: informe de pedido, de amortizaciones, de inactividad, de composición de equipos básicos, de composición de otros equipos, de inventario software y manuales, de garantías y de disponibilidad. Estos informes son realizados para el Responsable de Informática.

3.AVERIAS

Engloba todas ls operaciones relativas a las averías tanto el aviso que puede ser realizado por cualquier actor (Responsable de Informática, de Mantenimiento o Usuario) , como el parte de avería que es realizado por el Responsable de Mantenimiento y entregado al Responsable de Informática.

4.RESERVAS

Es tanto la petición de reserva de un equipo con unas características determinadas, que puede ser realizada por cualquier usuario al Responsable de Informática, como la concesión de una reserva que realiza este último a un usuario.

5.SUGERENCIAS

Es una línea de comunicación entre los diferente agentes que interaccionan con el sistema.

6.INFORMES PARA EL USUARIO

Es un informe especialmente realizado para el usuario donde este puede encontrar toda la información que pueda necesitar en un momento determinado sobre un equipo, su disponibilidad, software o un manual.

7.ACTIVIDAD

Realizado por el Responsable de Informática engloba todo lo relativo al buen funcionamiento del material de la empresa: dar de baja temporalmente un equipo cuando esta en reparación, dar de baja permanentemente un equipo cuando no tiene arreglo y actualizar tanto software como los manuales.

Análisis mediante escenarios

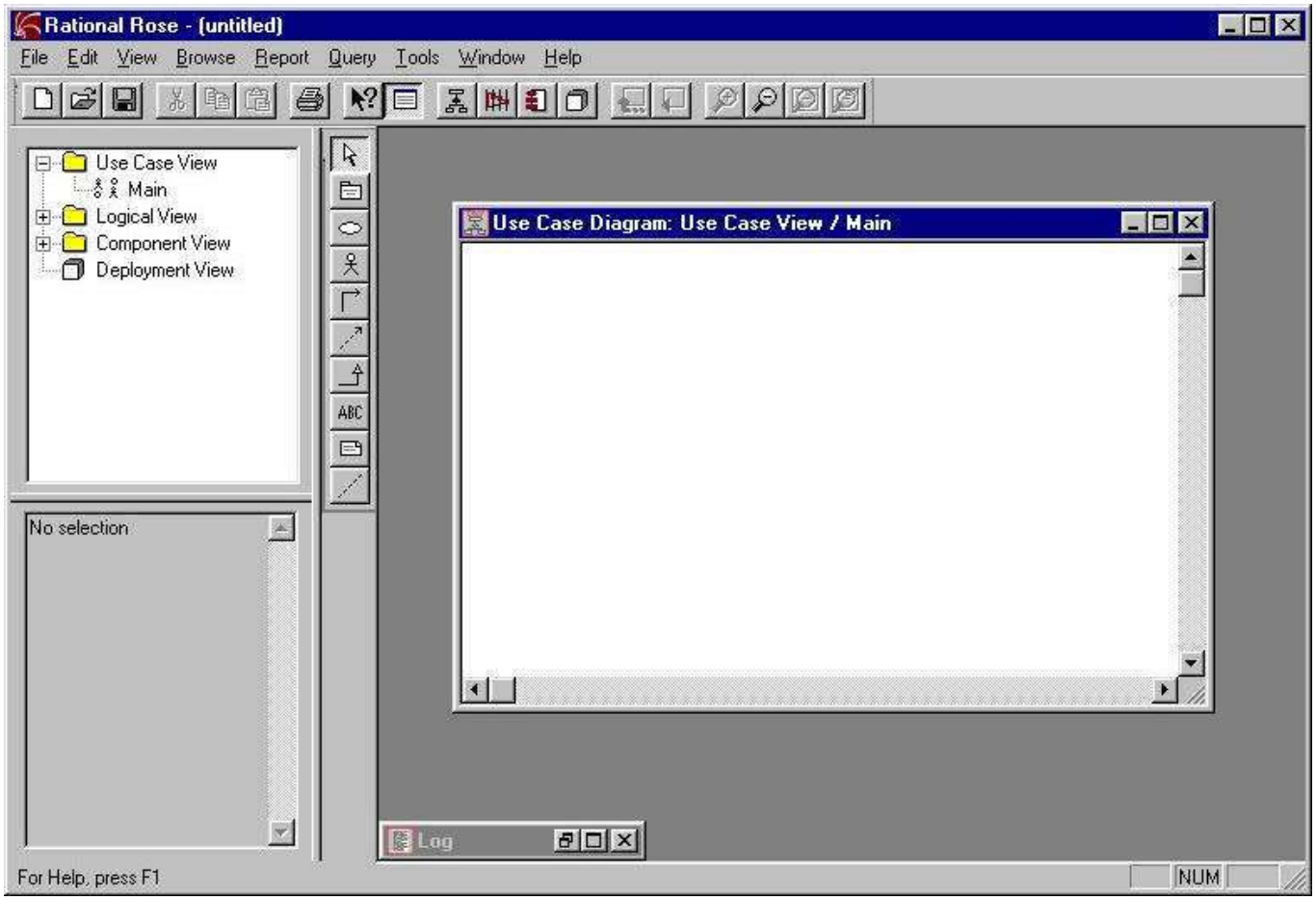
- Se enumeran escenarios fundamentales para el funcionamiento del sistema
- Se estudia cada escenario utilizando guiones como los que se usan en el cine
- Cada equipo que pasa por un escenario identifica los objetos y sus responsabilidades, así como los mecanismos que relacionan los objetos
- De los escenarios iniciales se puede pasar a otros escenarios secundarios
- Los escenarios también se pueden utilizar para probar el sistema en la fase de pruebas
- El estudio de los escenarios con detalle permite ir enriqueciendo el Diccionario de datos
- No es necesario hacer todos los escenarios y sub-escenarios posibles si se observa que no enriquecen el diccionario de datos

Numeración: 1.1.
Titulo: Hacer pedido
Precondiciones: Sugerencias de compra, caducidad de licencias, bajas permanente hardware, …
Quien Lo Comienza: Responsable de Informática.
Quien Lo Finaliza: Responsable de Informática.
Postcondiciones:
Descripción: Son las operaciones de compra de todos los componentes (hardware, software y manuales) que realiza el responsable de informática. Además da estos equipos de alta y debe apoyarse en el sistema para gestionar los pedidos correctamente para lo que debe anotar en el sistema que ha pedido un componente a un proveedor (por tanto que está pendiente de recibir).

Numeración: 1.2.
Titulo: Anular pedido
Precondiciones: Cambio de precio, cambio de necesidades de la Empresa.
Quien Lo Comienza: Responsable de Informática
Quien Lo Finaliza: Responsable de Informática
Postcondiciones:
Descripción: Esta operación la realiza el responsable de informática cuando toma la decisión de anular un pedido que había realizado con anterioridad.

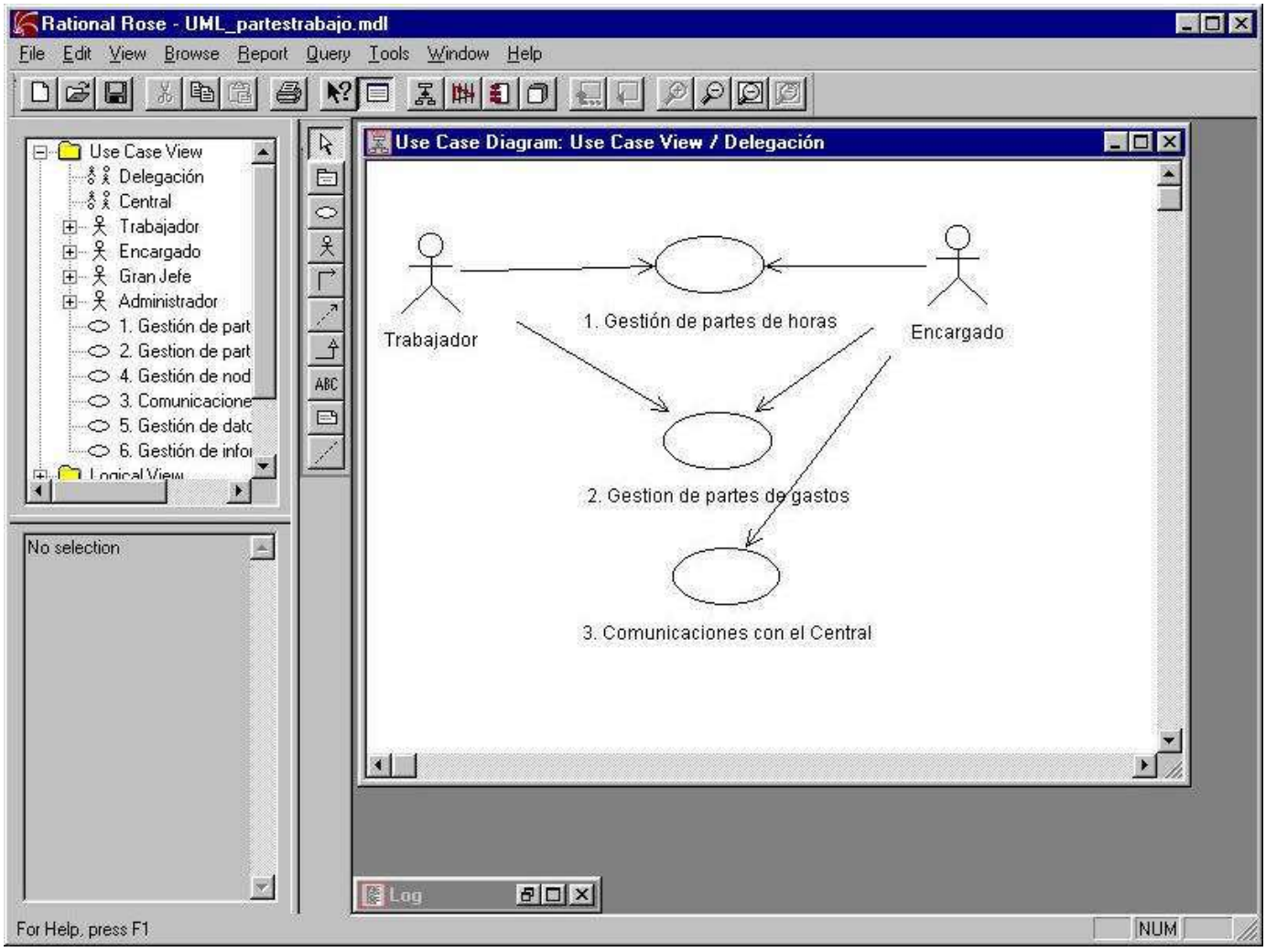
Numeración: 1.3
Titulo: Recibir pedido
Precondiciones: Haber realizado la petición de un pedido.
Quien Lo Comienza: Responsable de Informática
Quien Lo Finaliza: Responsable de Informática
Postcondiciones:
Descripción: El responsable de informática confirma la recepción de los pedidos.

Herramienta Rational Rose
[ROSE]



- Tiene una sección para ir introduciendo los Casos de Uso (*Use Case View*)
- Permite el manejo de actores, que se traducirán al sistema como clases
- Cada sistema recibe un nombre (no aparece el rectángulo)

Ejemplo de Casos de Uso con Rational Rose



EJERCICIOS PROPUESTOS

- 5.1 Realizar el análisis y el diseño preliminar del juego del ajedrez. Se puede jugar dos personas entre sí o una persona contra el ordenador. En este último caso debe ser posible seleccionar el nivel de dificultad entre una lista de varios niveles. El juego de ajedrez permitirá al jugador elegir el color de las piezas. La aplicación deberá permitir detectar los movimientos ilegales de las piezas, tiempos utilizados cada jugador, registro de jugadas y piezas perdidas. También determinará si se alcanzan tablas, y permitirá abandonar la partida a un jugador.
- 5.2 Realizar el análisis y diseño preliminar de una aplicación que realiza estudios de mercado para situar grandes superficies (hipermercados). Se supone que cada gran superficie necesita un mínimo de población que pueda acceder a dicho hipermercado en menos de un tiempo dado. La red de carreteras se puede representar mediante un grafo.
- 5.3 Realizar el análisis y diseño preliminar para gestionar los fondos bibliográficos y de socios de una biblioteca.
- 5.4 Realizar el análisis y diseño preliminar para gestionar un centro de enseñanza

Diagramas de Secuencia

- Se denominan
  - Diagramas de Secuencia en UML
  - Diagramas de Interacción en Booch
  - Diagramas de seguimiento de sucesos en OMT
- Se hace un diagrama de secuencia por cada escenario
- Permiten en las fases iniciales de análisis y diseño
  - Razonar más en detalle como es el comportamiento de un escenario
  - Obtener nuevas clases y objetos en el escenario (enriquecimiento del diccionario de datos)
  - Detectar cuales son los métodos de las clases, al observar como se relacionan los objetos entre sí para llevar a cabo la tarea encomendada en el escenario
- Se utilizan en las fases de prueba para validar el código
- Si se desea más detalle se utilizan los diagramas de Colaboración

Diagramas de interacción

Traza de la ejecución de un escenario

Notación de Booch

- *Un diagrama de interacciones se usa para realizar una traza de la ejecución de un escenario en el mismo contexto que un diagrama de objetos de la notación Booch.*
- Los diagramas de interacción toman los elementos esenciales de los diagramas de objetos y los reestructuran para enfocarlos a la lectura de los mensajes en orden
- *El **orden** se indica mediante la posición vertical,* siendo el primer mensaje el de la parte superior y el último el de la parte inferior. Por tanto no necesitan números de secuencia
- Los diagramas de interacción son mejores que los diagramas de objetos para capturar la semántica de los escenarios en un momento temprano del ciclo de desarrollo
- Según se avanza se pone más énfasis en los diagramas de objetos

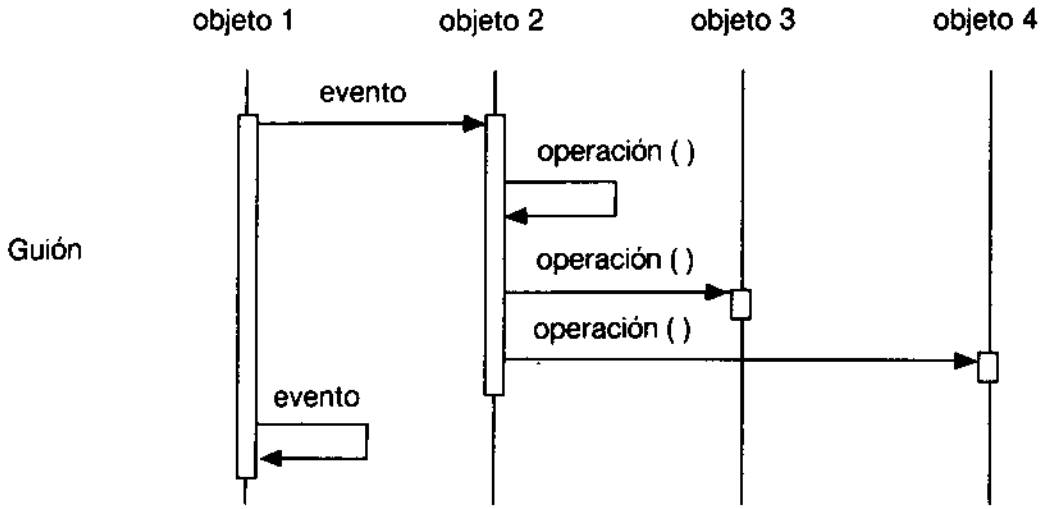
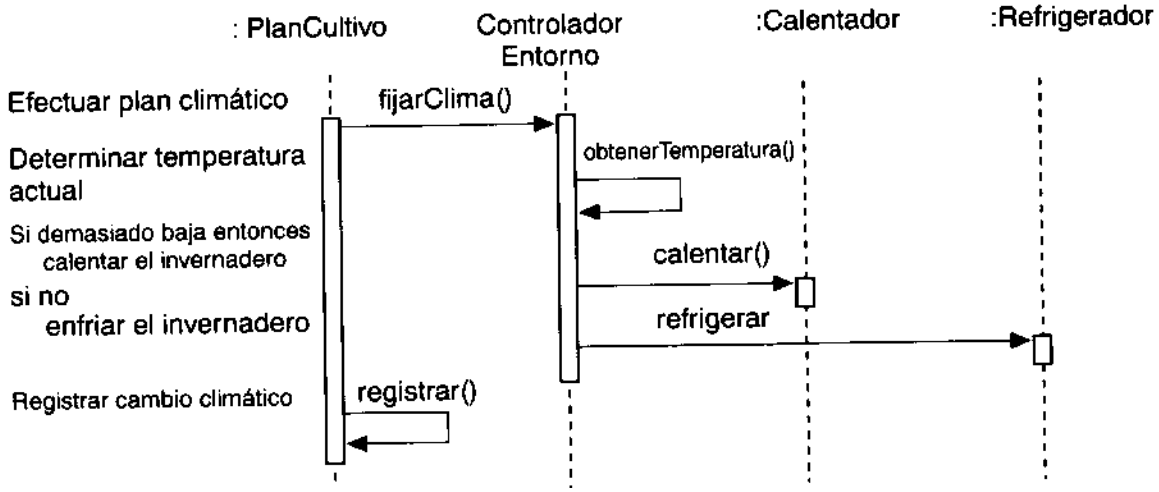
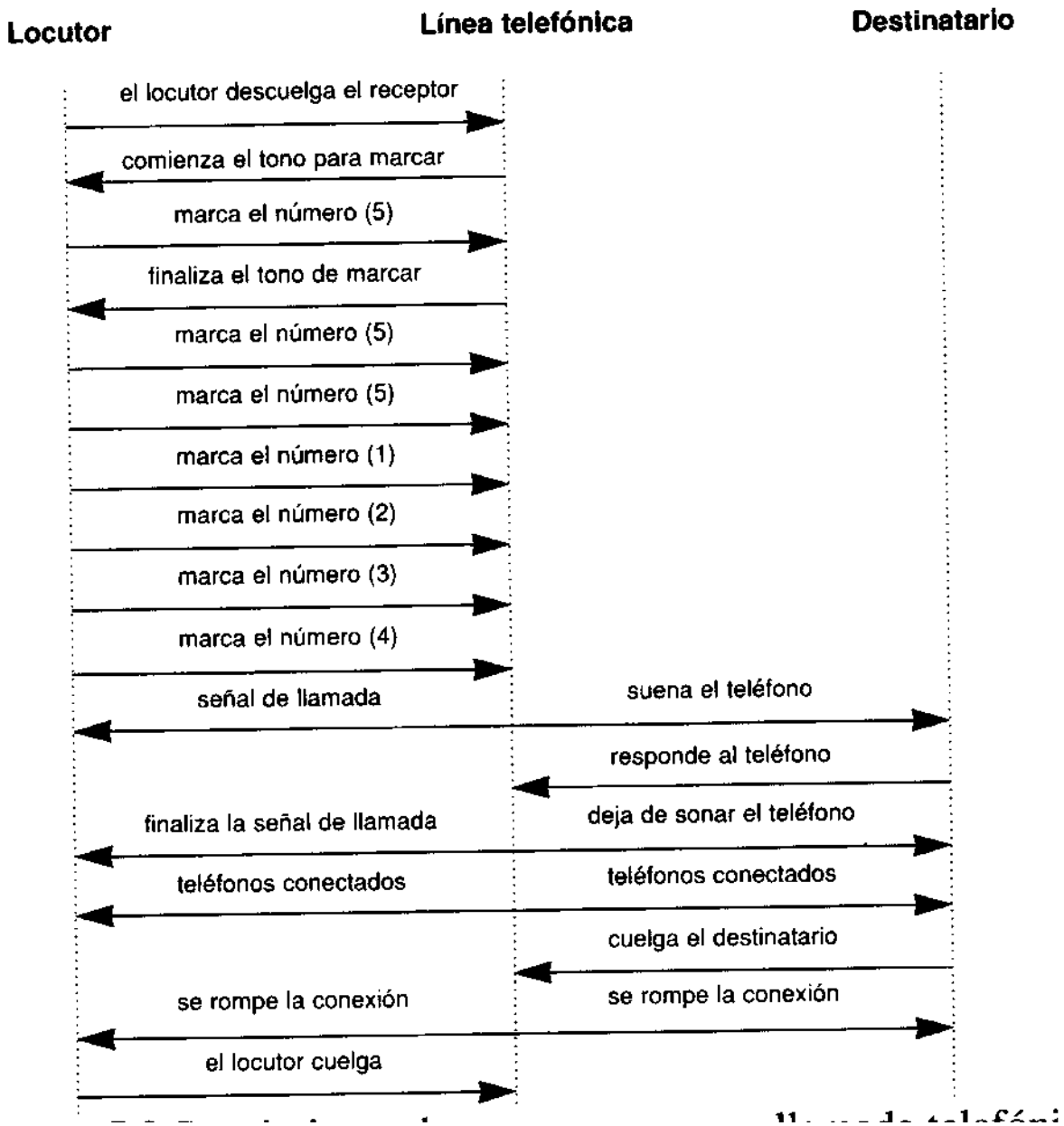


Diagrama de interacción
Guiones y centros de control
Ejemplo en notación de Booch

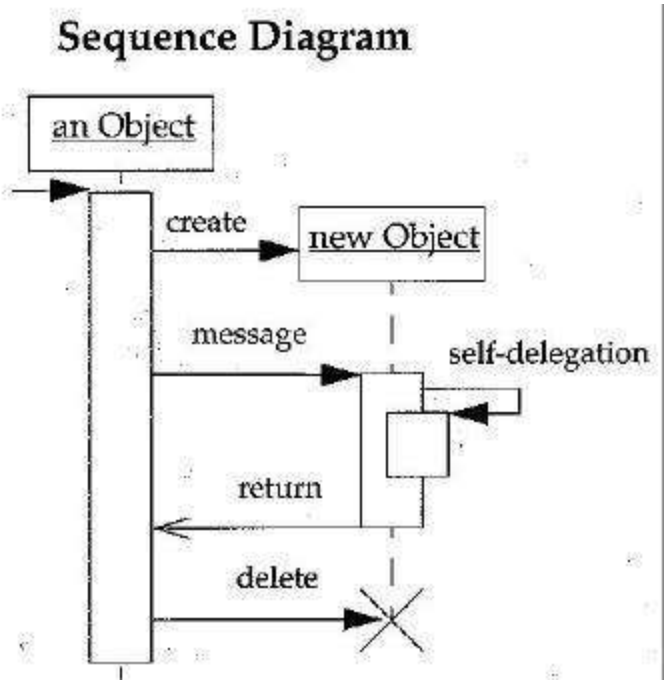
- *Un **guión** son las instrucciones generales de cada mensaje del diagrama de interacción*
- Los guiones se escriben a la izquierda del diagrama de interacción en lenguaje natural (español) o en un lenguaje de programación
- Ni los diagramas de objetos ni los diagramas de interacción indican por sí solos el centro de control a medida que pasan los mensajes
- En el diagrama de interacción se utilizan ***rectángulos*** en las líneas verticales de cada objeto *para indicar el tiempo relativo durante el cual el flujo de control está centrado en ese objeto*

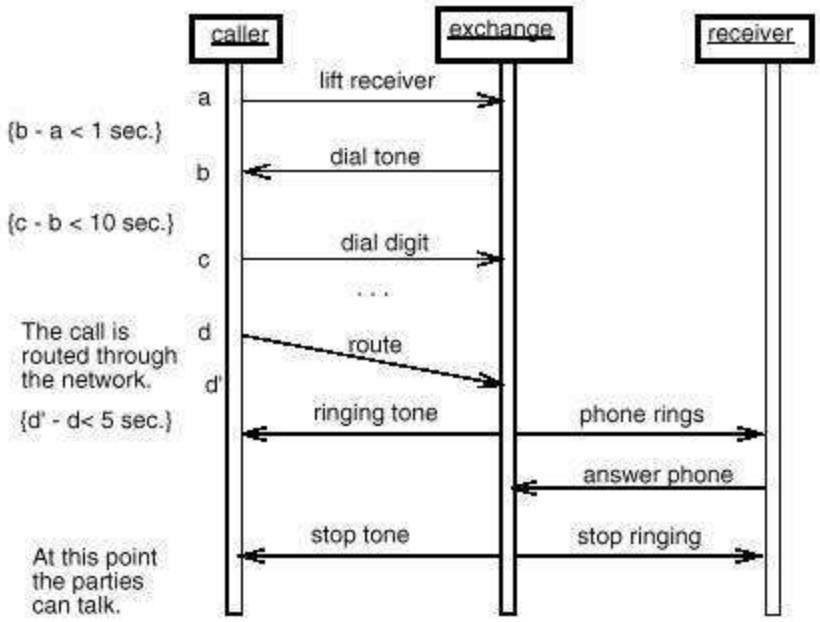


Escenarios y seguimiento de sucesos
Notación OMT

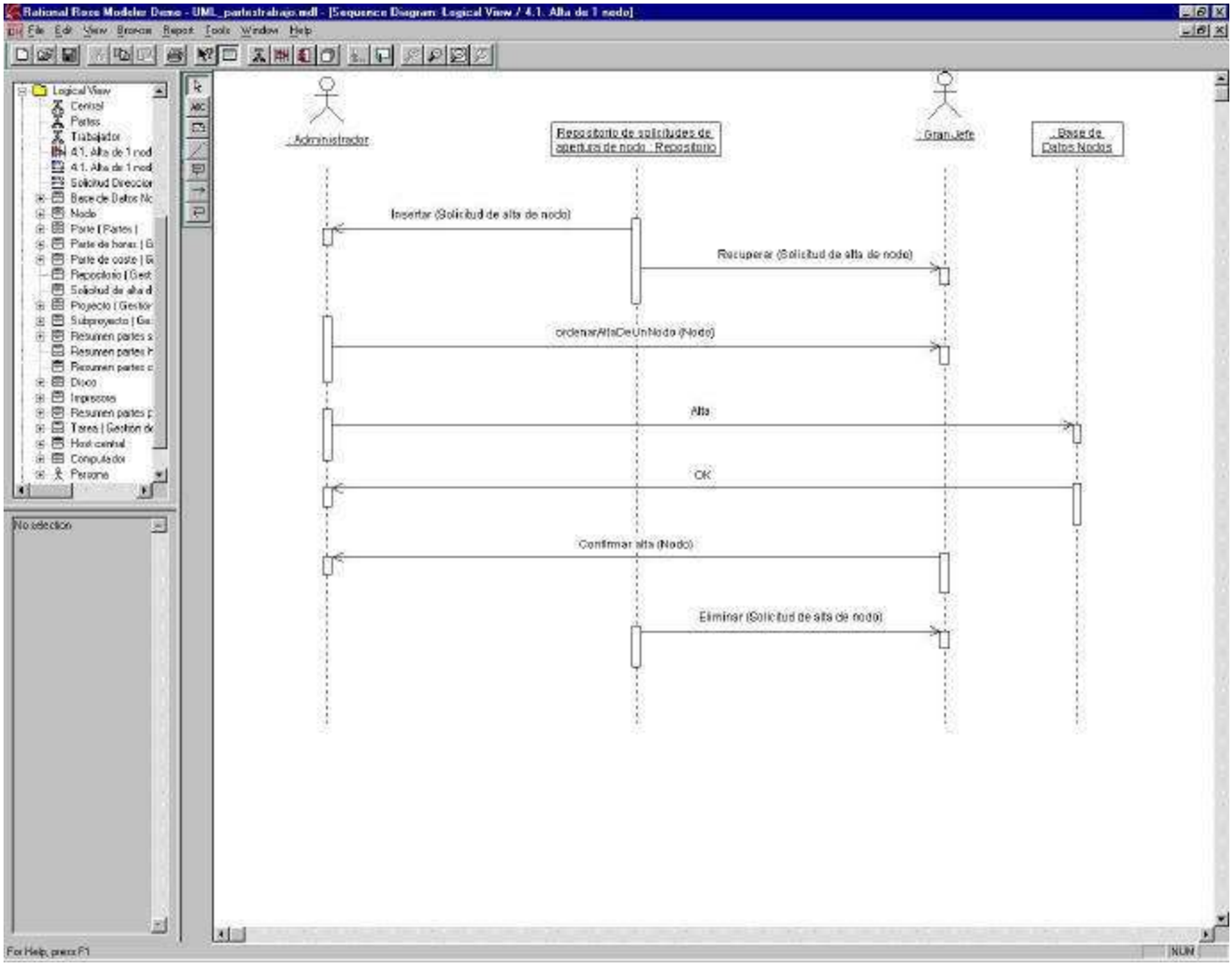


Diagramas de secuencia en UML





Ejemplo de diagrama de secuencia con Rose



Diagramas de Colaboración

- Se denominan
  - Diagramas de Colaboración en UML
  - Diagramas de objetos en Booch
- Permiten profundizar en el nivel de detalle en los diagramas de Secuencia
- Expresan las colaboraciones de los objetos en tiempo de ejecución

Diagramas de objetos

Muestra la existencia de objetos y sus relaciones en la vista lógica de un sistema

Notación de Booch

- En el ***análisis*** se usan los diagramas de objetos para indicar la semántica de los escenarios primarios y secundarios que proporcionan la traza del comportamiento del sistema
- En el ***diseño*** se usan diagramas de objetos para ilustrar la semántica de los mecanismos del diseño lógico
- Cada objeto se denota con su nombre y opcionalmente con sus atributos
- Los objetos interaccionan a través de enlaces con otros objetos
- ***Objeto cliente*** *es el objeto que invoca una operación*
- ***Objeto servidor*** *es el objeto que suministra la operación*
- Con una flecha se indica la dirección de la invocación, así como la operación, y un número de orden (opcional)

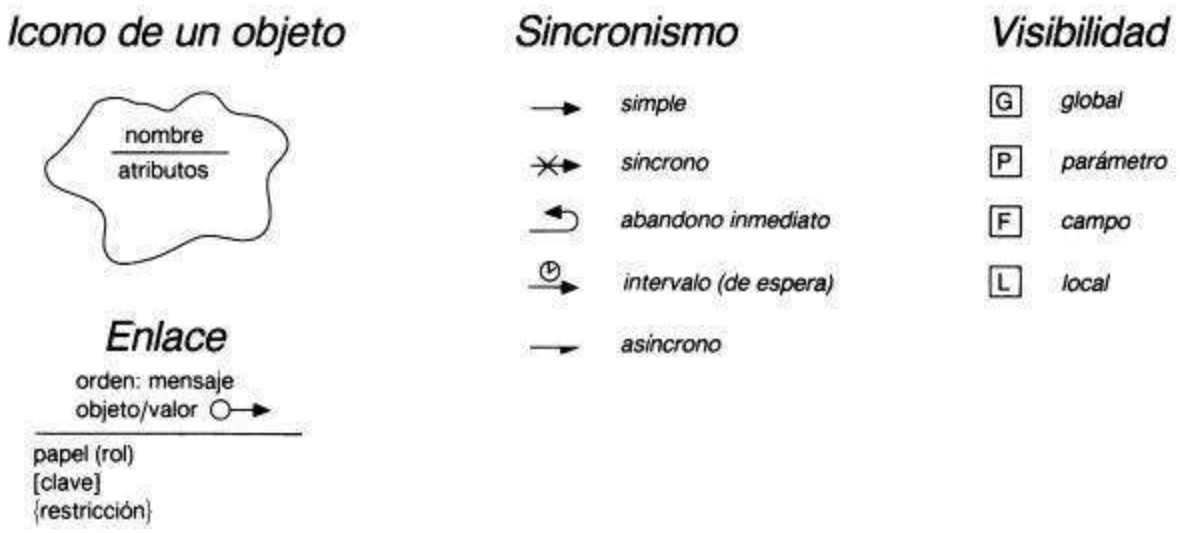
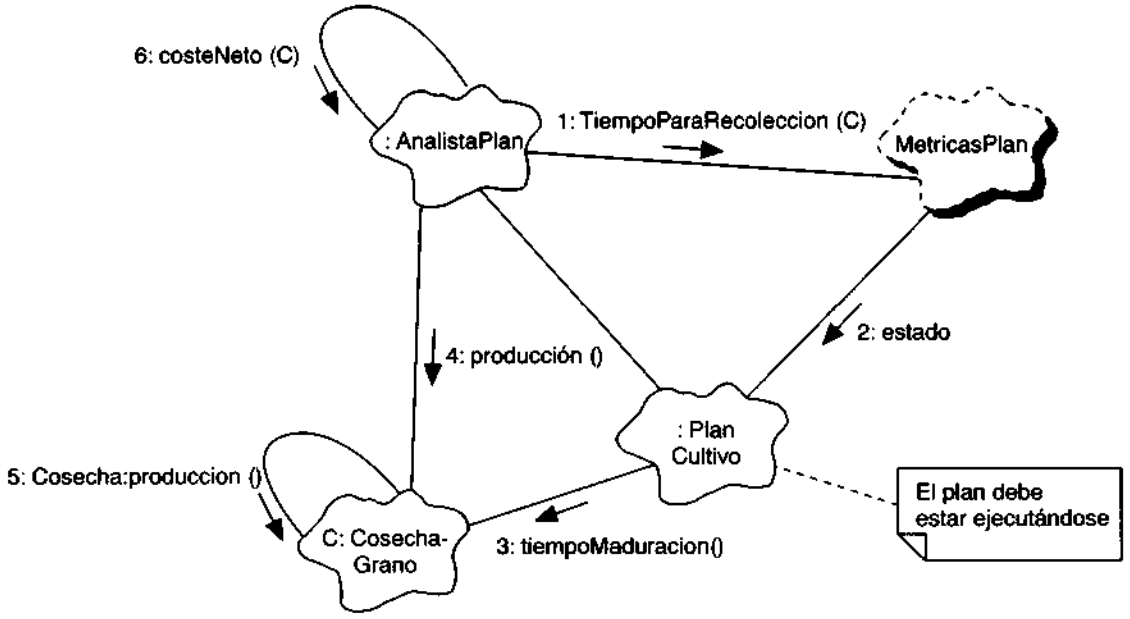


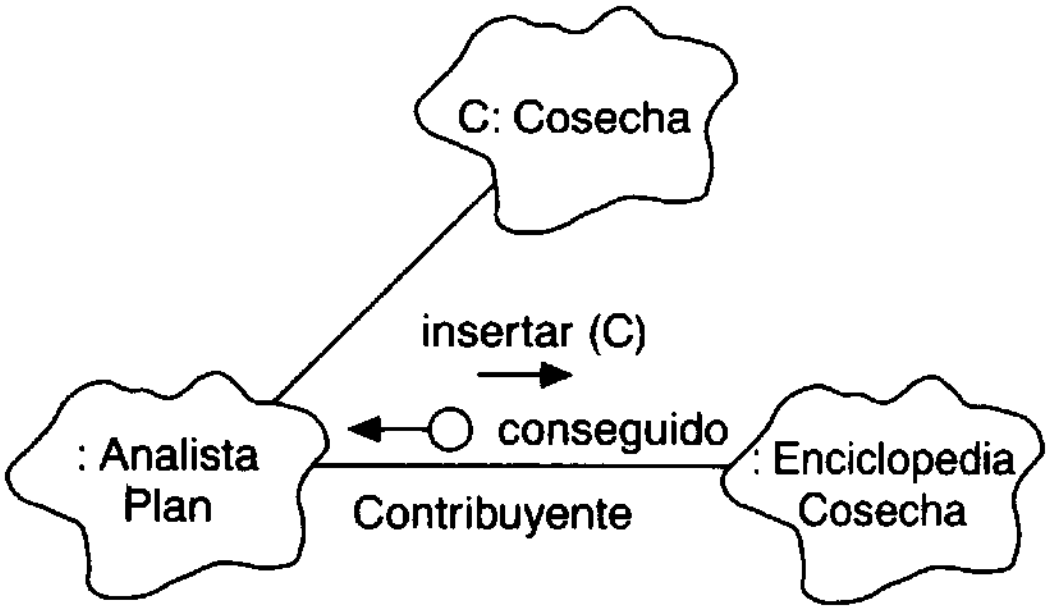
Diagrama de objetos
Ejemplo en notación de Booch

- Se ilustra el escenario que muestra la traza de ejecución de la determinación del coste neto de recolección de una cosecha específica
- El orden de las acciones está numerado
- Puede haber relaciones que no se utilicen para pasar mensajes en un escenario determinado



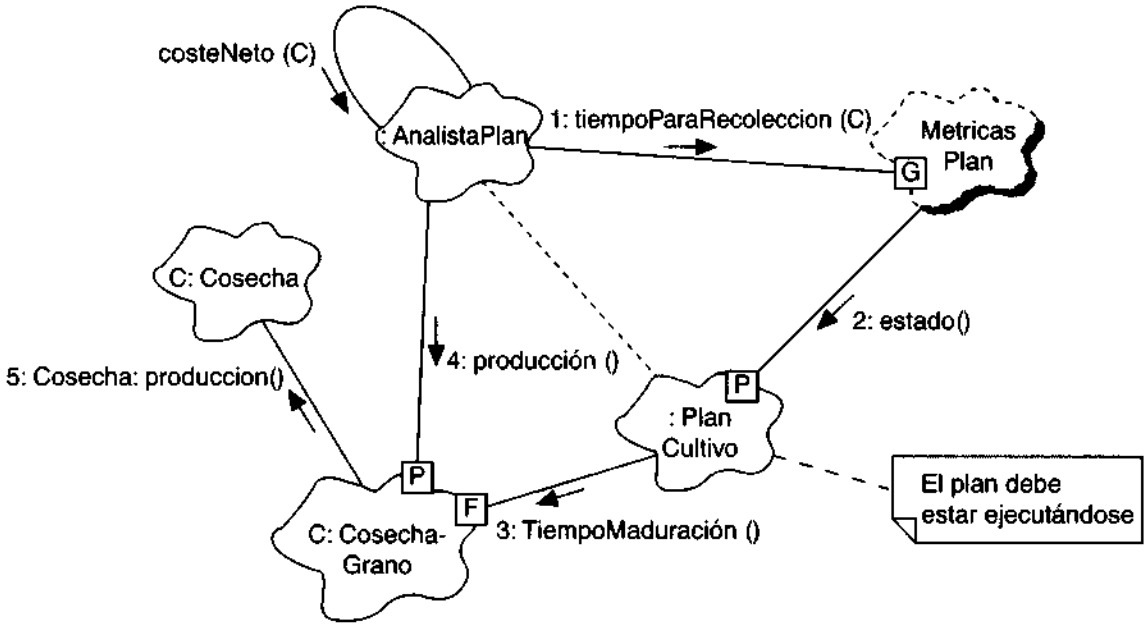
Diagramas de objetos
Papeles (roles) y flujo de datos
Notación de Booch

- Se pueden especificar los ***papeles*** (roles) del diagrama de clases en el diagrama de objetos
- Se pueden indicar también las ***claves*** y ***restricciones*** del diagrama de clases en el diagrama de objetos
- Se muestra explícitamente la ***dirección del flujo de datos*** que *ayuda a explicar la semántica de un escenario particular*
- Se indica la dirección del flujo de datos con una flecha y un círculo
- Se puede usar tanto un objeto como un valor para el flujo de datos



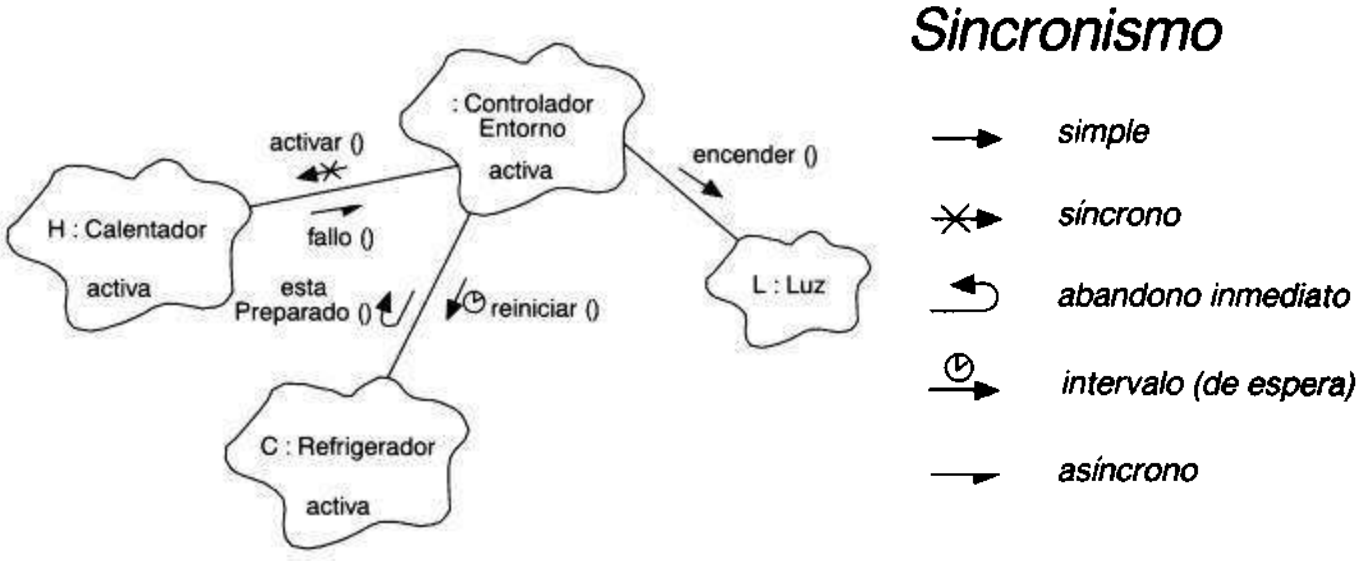
Diagramas de objetos
Visibilidad
Notación de Booch

- En ciertos escenarios es útil reflejar la forma exacta en que un objeto tiene ***visibilidad*** sobre otro
- Las ***marcas de visibilidad*** son
  - ***G -*** El objeto servidor es ***global*** al cliente
  - ***P*** - El objeto servidor es ***parámetro*** de alguna operación del cliente
  - ***F -*** El objeto servidor es parte del cliente ***(field, campo)***
  - ***L -*** El objeto servidor es un objeto declarado ***local*** en el ámbito del diagrama de objetos
- La ausencia del adorno de visibilidad indica que está sin especificar



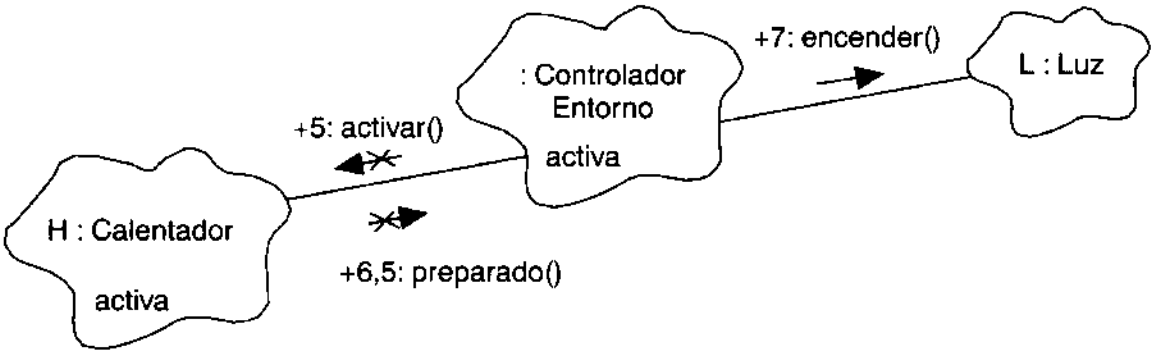
Diagramas de objetos
Objetos activos y sincronización
Notación de Booch

- Los objetos son activos si incorporan su propio hilo de control
- Todos los objetos son ***secuenciales*** a menos de que se diga lo contrario
  - ***sincronismo simple****: sólo hay un único hilo de control*
- Se puede indicar el estilo de concurrencia de un objeto:
  - ***Síncrono****: el cliente espera hasta que el servidor acepte el mensaje*
  - ***Paso de mensajes con abandono inmediato****: el cliente abandona el mensaje si el servidor no lo atiende inmediatamente*
  - ***Sincronización de intervalo (de espera o timeout)****: el cliente abandona el mensaje si el servidor no puede atenderlo dentro de un espacio de tiempo determinado*
  - ***Asíncrono****: el cliente envía el evento al servidor para que lo procese, el servidor pone el mensaje en una cola, y el cliente continúa su actividad sin esperar al servidor*



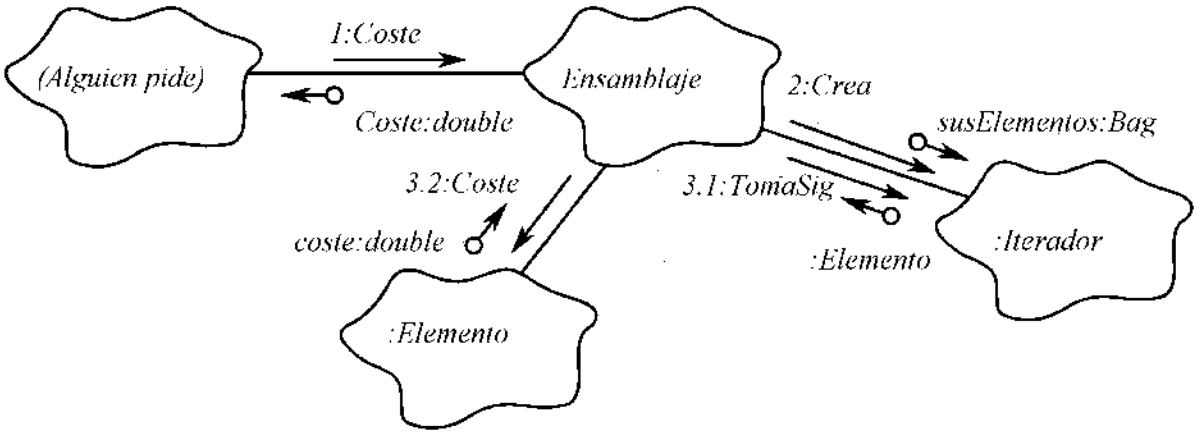
Diagramas de objetos
Presupuestos de tiempo y especificaciones
Notación de Booch

- Para aplicaciones críticas respecto al tiempo se pueden trazar escenarios indicando tiempos exactos
- Se usan ***valores de tiempo en vez de números de secuencia*** que indican tiempo relativo (por ejemplo en segundos) con el signo más delante
- ***Especificaciones***. Al igual que para los diagramas de clases cada entidad del diagrama de objetos puede tener su especificación en texto
  - Los diagramas de objetos deben especificar su contexto
    - `Contexto: global | categoría | clase | operación`

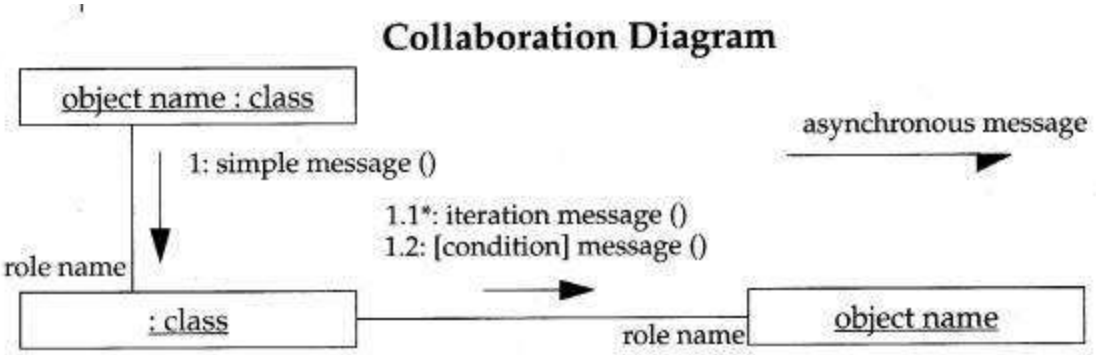


Ejemplo: Control de inventario
Diagrama de objetos
Notación de Booch

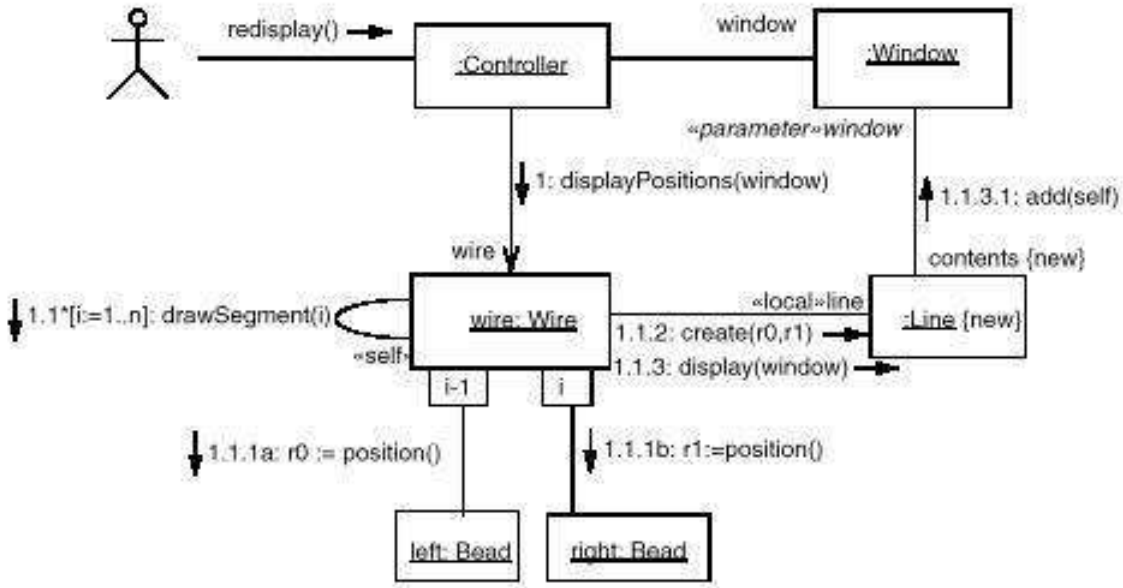
- 1 (*Alguien pide) **Coste*** de un *Ensamblaje*
  - *Ensamblaje* debe devolver *Coste*
- 2 *Ensamblaje **Crea*** un *Iterador*
  - *Ensamblaje* envia *susElementos:Bag*
- *3 Comienza un bucle*
  - 3.1 *Ensamblaje **TomaSig***-uiente *Elemento*
    - *Iterador* envia *Elemento*
  - *3.2 Ensamblaje* pide *Coste* del *Elemento*
    - *Elemento* envia *coste*



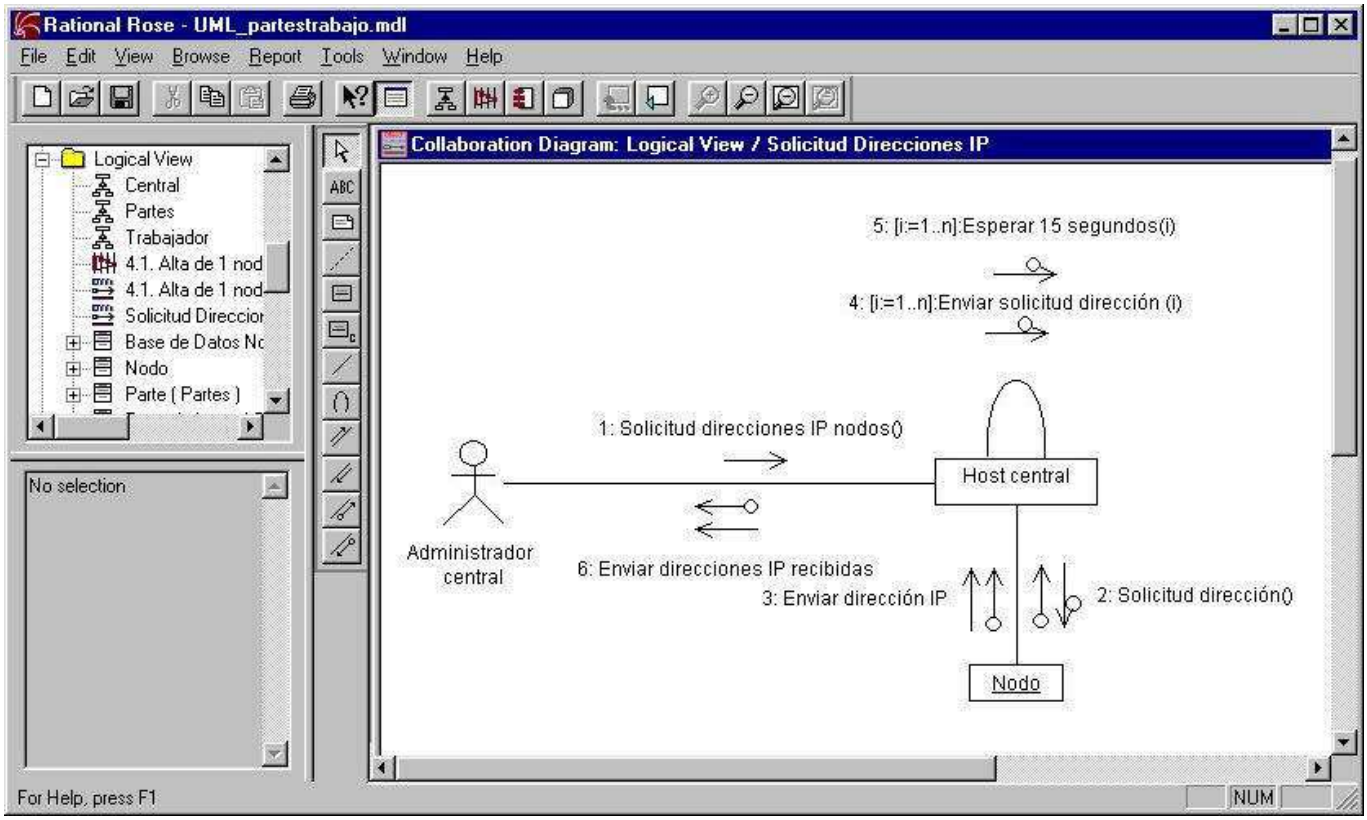
Diagramas de colaboración UML



Ejemplo de diagrama de colaboración en UML



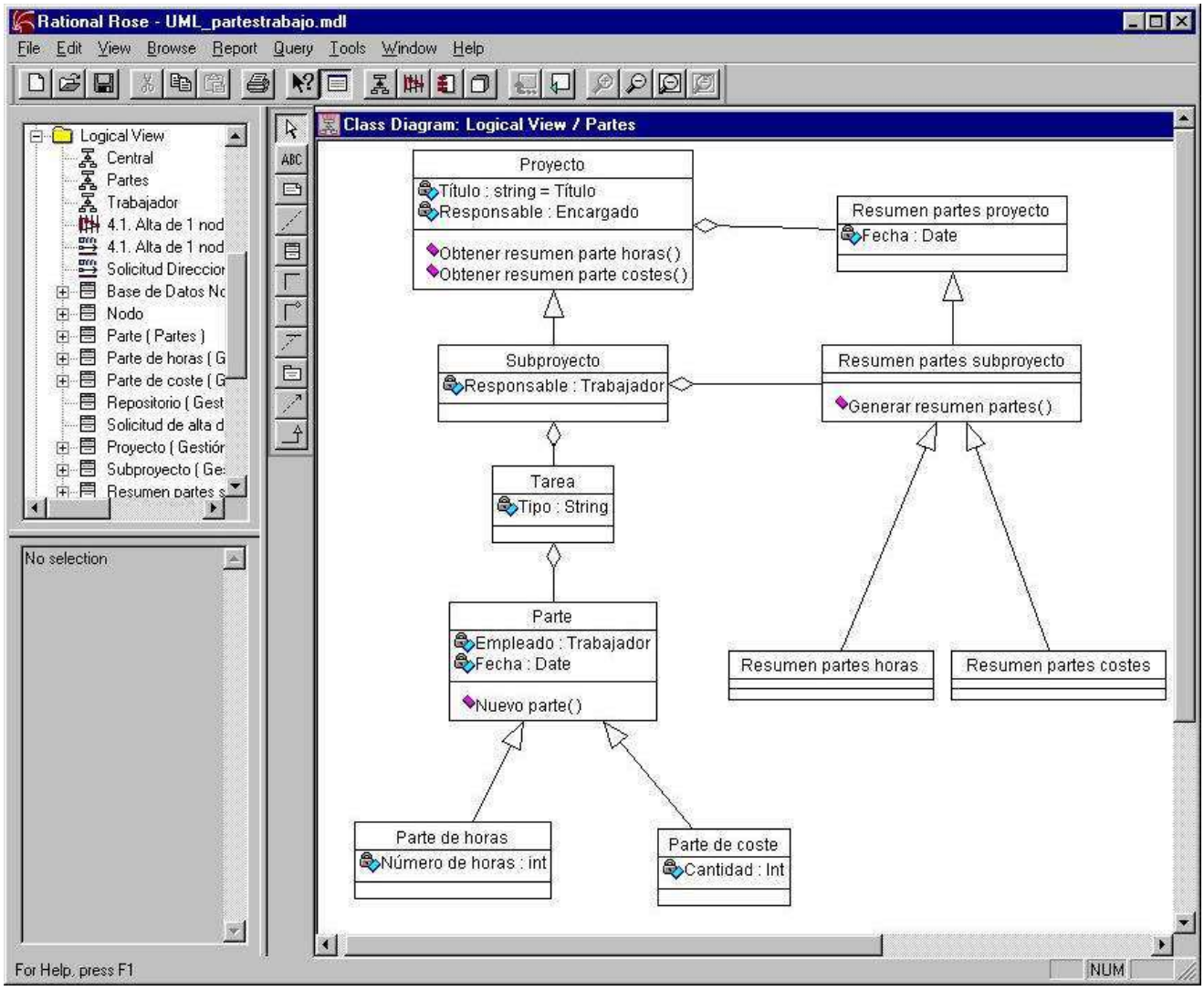
Ejemplo de diagrama de colaboración en Rose



Diagramas Estáticos

- Se denominan
  - Diagramas Estáticos en UML
  - Diagramas de Clases en Booch
  - Diagrama de Clases en OMT
- Estos diagramas son los más importantes del diseño orientado a objetos, son la *piedra angular* de nuestro diseño.
- Contienen toda la información de todas las clases y sus relaciones con otras clases
- Aunque son los más importantes no se llega a ellos directamente dado que tienen un gran nivel de abstracción dado que contemplan el modelo globalmente sin particularizarse en ningún escenario concreto
- Cuando se construyen los diagramas anteriores (Casos de uso, Secuencia, Colaboración) las herramientas van obteniendo nombres de clase y generando los atributos y operaciones de cada clase siguiendo las indicaciones dadas por las especificaciones de requisitos en los casos de uso y escenarios
- En el momento de hacer el primer diagrama estático ya se tiene una lista de clases con algunos de sus atributos y operaciones. Sin embargo es necesario reflexionar y abstraer sobre la organización de esas clases estudiando las relaciones de herencia, agregación, etc.
- El diagrama Estático se refinará en las sucesivas iteraciones del modelo

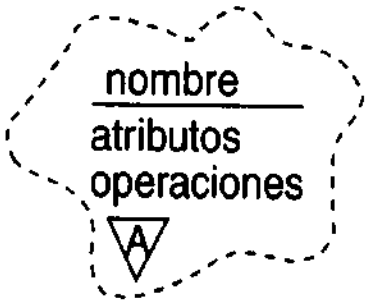
Ejemplo de Diagrama Estático con Rose



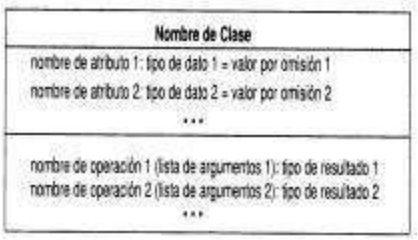
Clases

- Booch representa las clases por una nube con contorno discontinuo, colocándose en su interior el nombre de la clase y separados por una línea están los atributos, las operaciones, las restricciones y los adornos
- La nube significa que las fronteras de una abstracción no son nítidas
- Las líneas discontinuas indican que los clientes sólo operan sobre objetos y no sobre la clase
- Los atributos y las operaciones pueden ser visibles o no según el detalle deseado
- Un atributo es un dato que se almacenará en los objetos que son instancias de la clase
- Un método es la implementación de una operación para la clase
- Un adorno de clase es una propiedad por ejemplo indicar que la clase es abstracta (no puede tener instancias, sólo se puede heredar de ella)
- En UML las clases se pueden representar de tres formas:
  - Sin detalle
  - Detalles a nivel de análisis y diseño
  - Detalle a nivel de implementación
- Los atributos en UML se pueden describir como:
  visibilidad : *tipo* = *valor-inicial* { *propiedad* }
    - donde *visibilidad* puede ser:
      - + publico
      - # protegido
      - - privado

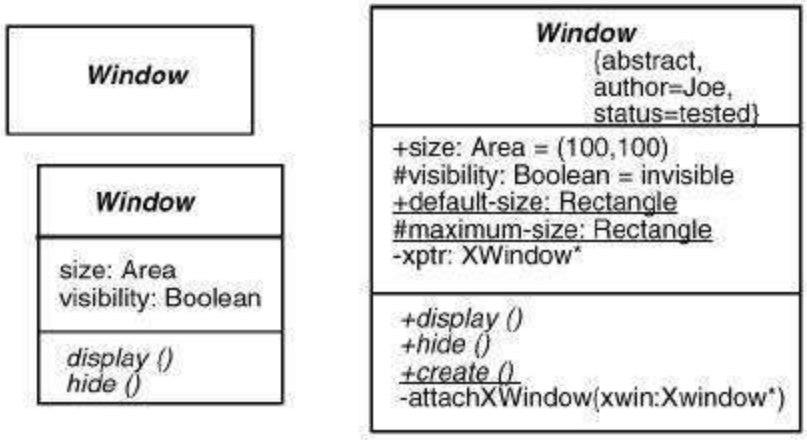
Booch



OMT



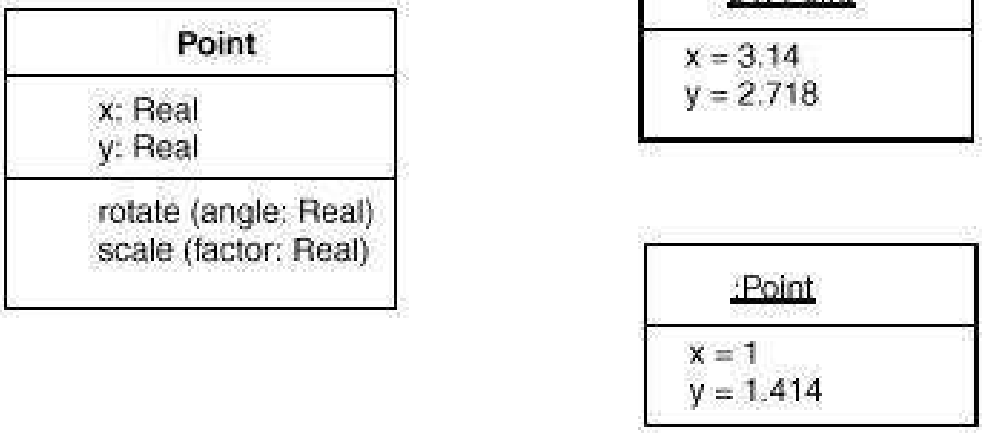
UML



Clases y objetos en UML

Objetos

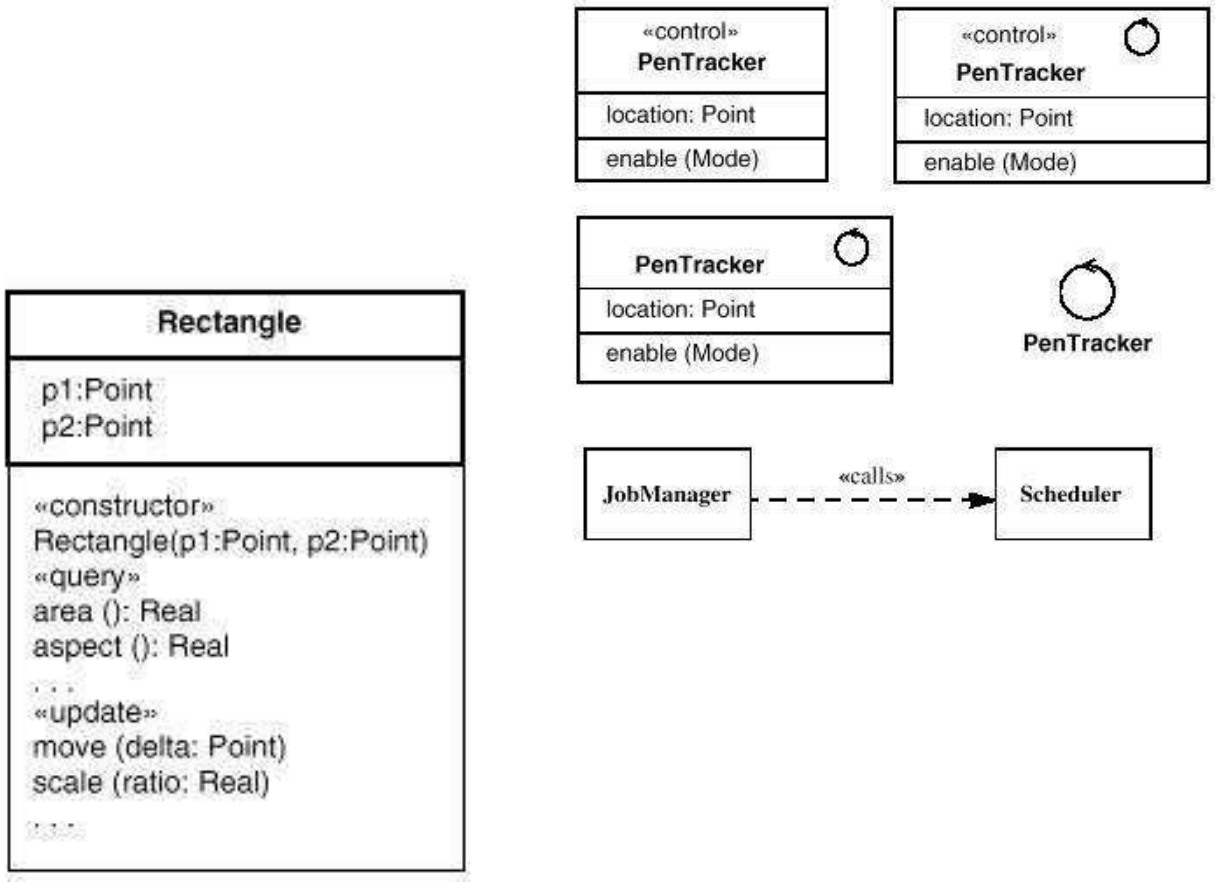
Clase



Los compartimentos de una clase pueden tener nombres

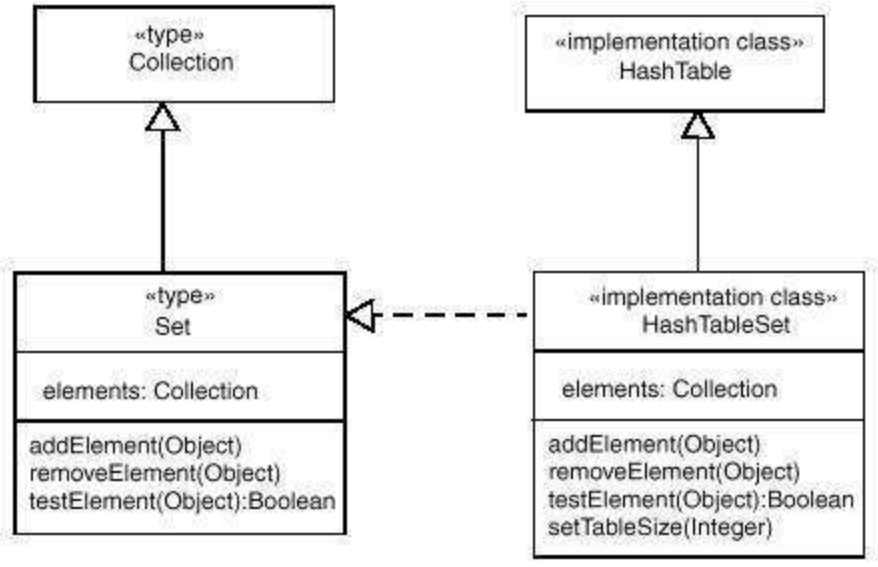
Estereotipos en UML

• Un estereotipo es una forma de clasificar las clases a alto nivel
• Los estereotipos se muestran con un texto entre doble ángulo << y >>, por ejemplo: <<*control*>>
• Los estereotipos también se pueden definir con un icono
• Muchas extensiones del núcleo de UML se pueden describir mediante una colección de estereotipos
• También se puede razonar que los tipos clase, asociación, y generalización son subtipos de estereotipos del metamodelo
• Los estereotipos también se pueden colocar a grupos de la lista de elementos de una clase



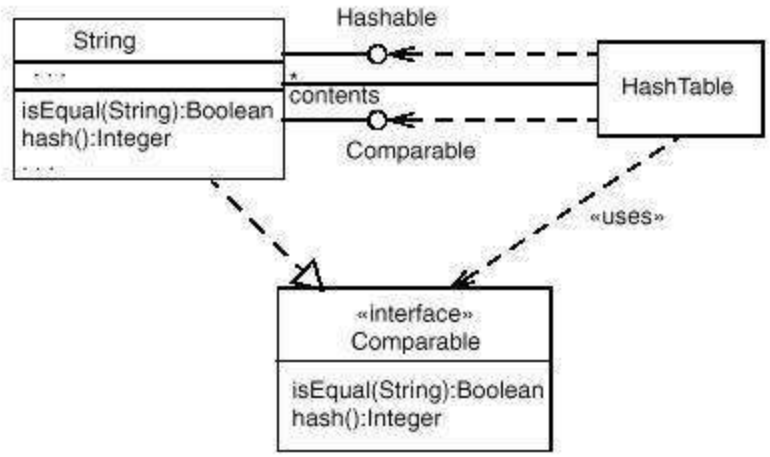
Tipos versus Clases de implementación en UML

• Las clases se pueden especializar usando estereotipos en Tipos y Clases de implementación
• Un ***Tipo*** caracteriza un papel (rol) cambiante que un objeto puede adoptar y después abandonar
• Una ***Clase de implementación*** define físicamente la estructura de datos y procedimientos que se implementará en un lenguaje de programación (C++, Java,...)
• Un objeto puede tener múltiples tipos (que pueden cambiar dinámicamente) pero sólo una Clase de Implementación (que es fija).
• Un Tipo se muestra con el esterotipo ***<<type>>***
• Una Clase de implementación se muestra con el estereotipo ***<<implementation class>>***
• La implementación de un tipo por medio de una clase de implementación se modela por medio de una "relación **Realiza**", que se indica con una flecha hueca con trazos discontinuos. Este símbolo implica herencia de operaciones pero no de estructura.



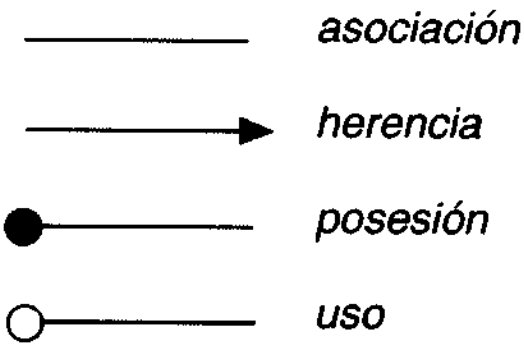
Interfaces en UML

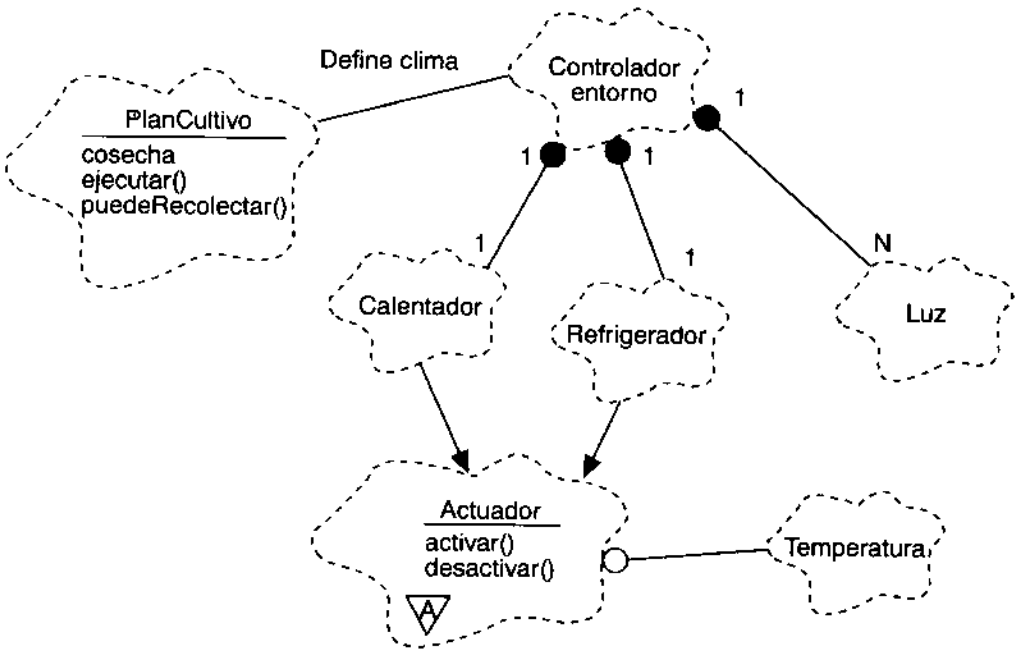
- *Una **Interfaz** es una especificación para las operaciones externas visibles de una clase, componente, o otra entidad (incluyendo unidades globales como los paquetes), pero siempre sin especificar la estructura interna*
- Cada interfaz especifica a menudo sólo una parte limitada del comportamiento de una clase real
- Una interfaz no tiene implementación
- Una Interfaz es formalmente equivalente a una clase abstracta sin atributos y sin métodos, con sólo operaciones abstractas
- Un interfaz se puede representar de dos formas:
  - Circulo unido por una línea a una clase con las operaciones de interfaz al lado del círculo
  - Rectángulo con la palabra reservada ***<<interface>>*** y la sección de operaciones
- Una clase que usa las operaciones de una interfaz se une por medio de una flecha de línea discontinua a los círculos o al rectángulo de la interfaz
- También se puede utilizar la "***relación Realiza***" desde una clase a una interfaz que soporta



Relaciones entre clases
Booch

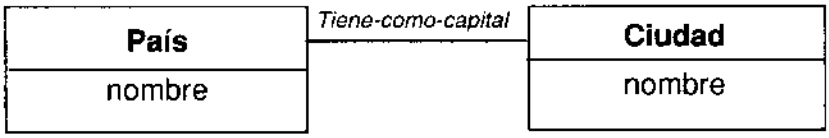
- Las asociaciones se pueden etiquetar con un nombre o frase
- Una clase puede asociarse consigo misma (asociación reflexiva)
- La cardinalidad o multiplicidad especifica el número de asociaciones entre dos clases
  - 1 (exactamente uno)
  - N (cero o más)
  - 1..N (uno o más)
  - 3..7 (rango especificado)
  - 7 (número exacto)
- Si no se indica se considera sin especificar



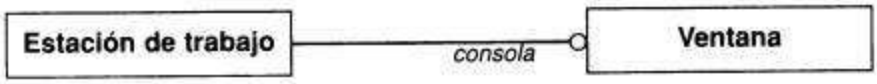


Relaciones entre clases (OMT)

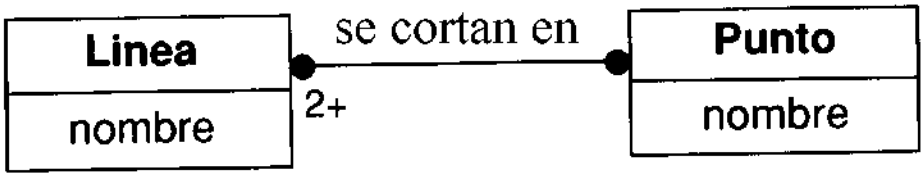
Un a línea sin símbolos de multiplicidad denota asociación uno a uno



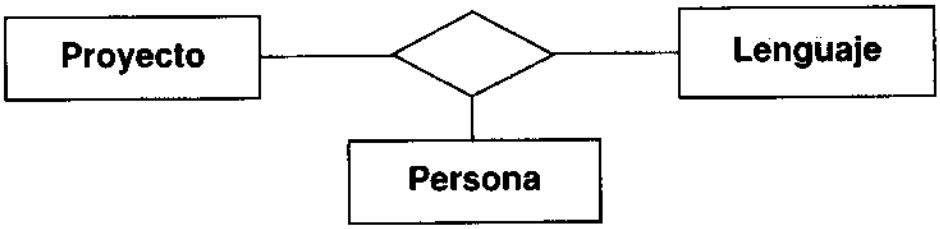
Un círculo blanco indica cero o uno



Un círculo negro indica cero o más
También se puede utilizar un número 7 (valor exacto), 2+ (2 o más), 3-5 (un intervalo).

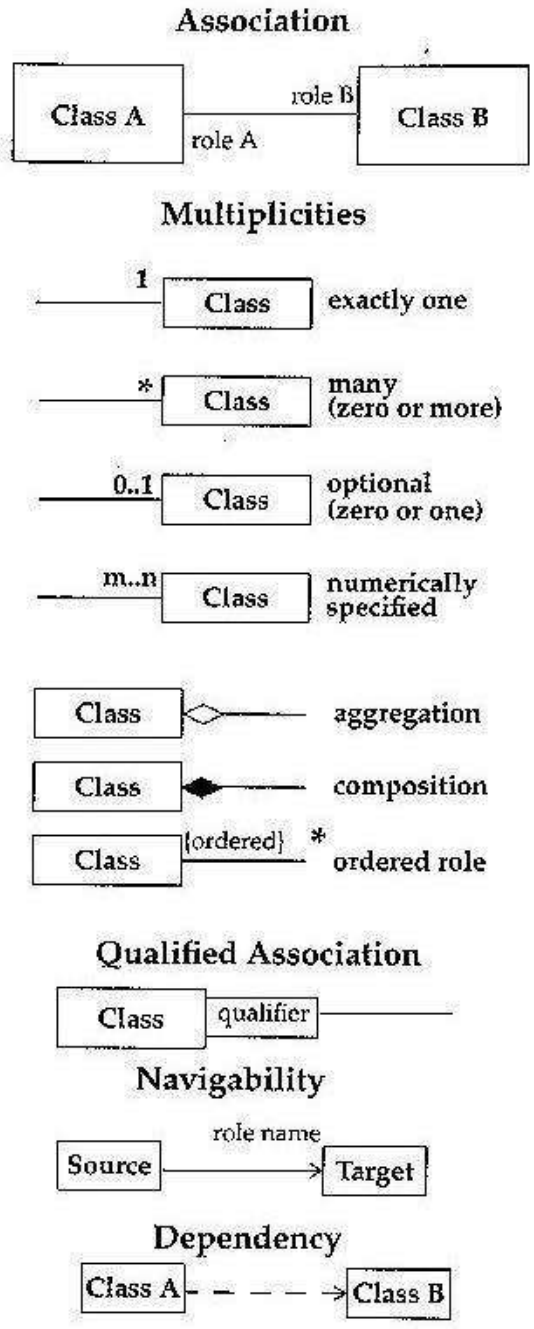


Asociación ternaria



La notación de círculos blancos y negros no se usa para asociaciones n-arias

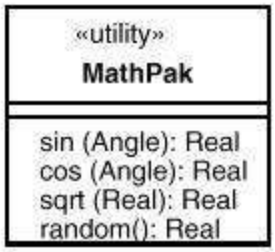
Asociaciones entre clases (UML)



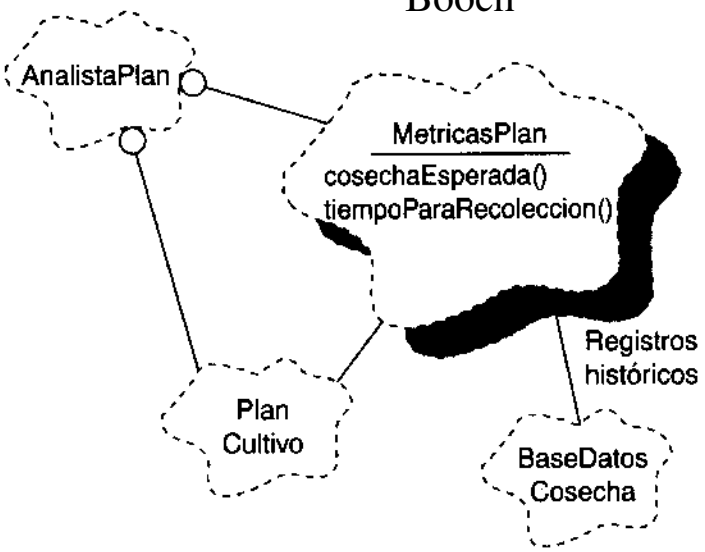
Utilidades de clase

- Las utilidades de clase representan la encapsulación de abstracciones no orientadas a objetos que proporcionan servicios algorítmicos
- Pueden estar constituidas por los servicios de subprogramas libres, servicios de manejo de sistemas de gestion de bases de datos (SQL inmerso), clases estáticas de C++, llamadas a funciones API de Windows,…
- Las clases pueden asociarse y utilizar utilidades de clase
- En algunos casos pueden parametrizarse y ser instanciadas
- ***¡ NO SE PUEDE HEREDAR DE LAS UTILIDADES DE CLASE !***
- En UML se coloca el ***esterotipo <<utility>>*** en el diagrama de clase
- En Booch la utilidad de clase lleva sombra

UML



Booch

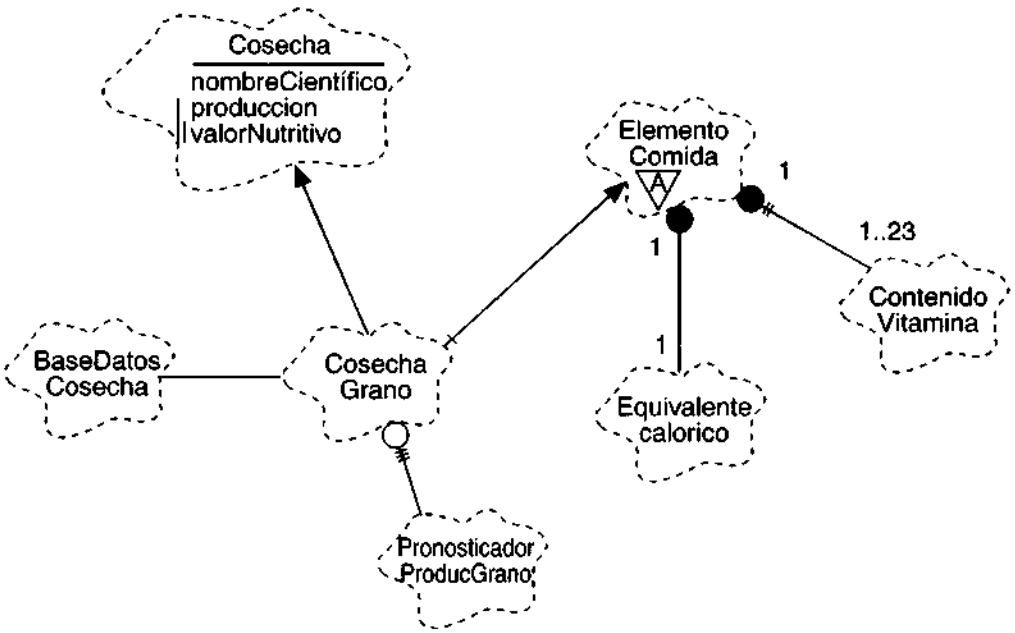


Control de exportación (control de acceso)

- La mayor parte de los lenguajes OO permiten especificar los permisos de acceso a los datos y a los métodos de una clase
- Los accesos se pueden marcar en la relación apropiada con los siguientes símbolos

Booch	**UML**	
<sin marcas>	+	*Acceso público*
|	#	*Acceso protegido*
||	-	*Acceso privado*
|||	{implementation}	*Se utiliza en la implementación*

Booch

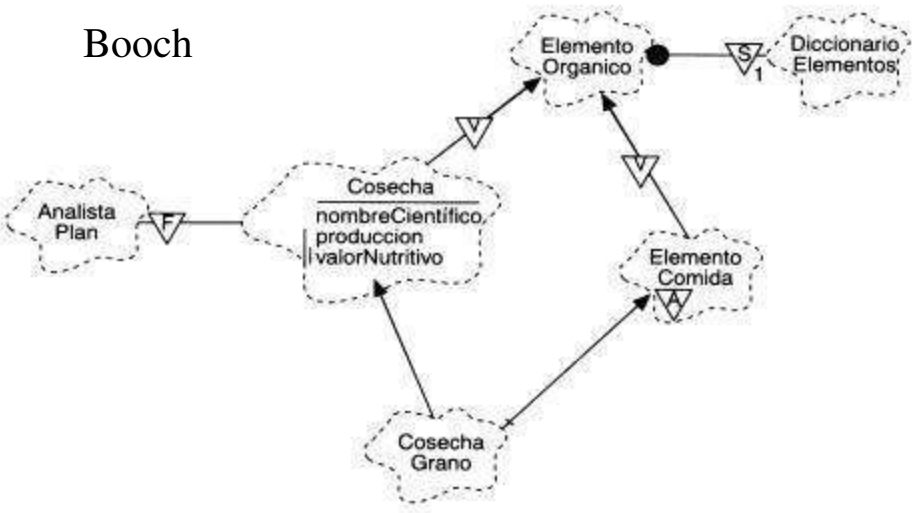


Propiedades

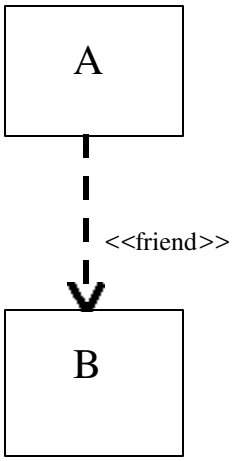
Califican la relación entre clases, en C++ hay tres propiedades:

Booch	UML	
S	*<<static>>*	La designación de un objeto o función de una clase
V	*<<virtual>>*	La designación de una clase base compartida en una trama de herencias con forma de rombo
F	*<<friend>>*	La designación de una clase que concede a otra derechos de acceso a sus partes no públicas

Booch

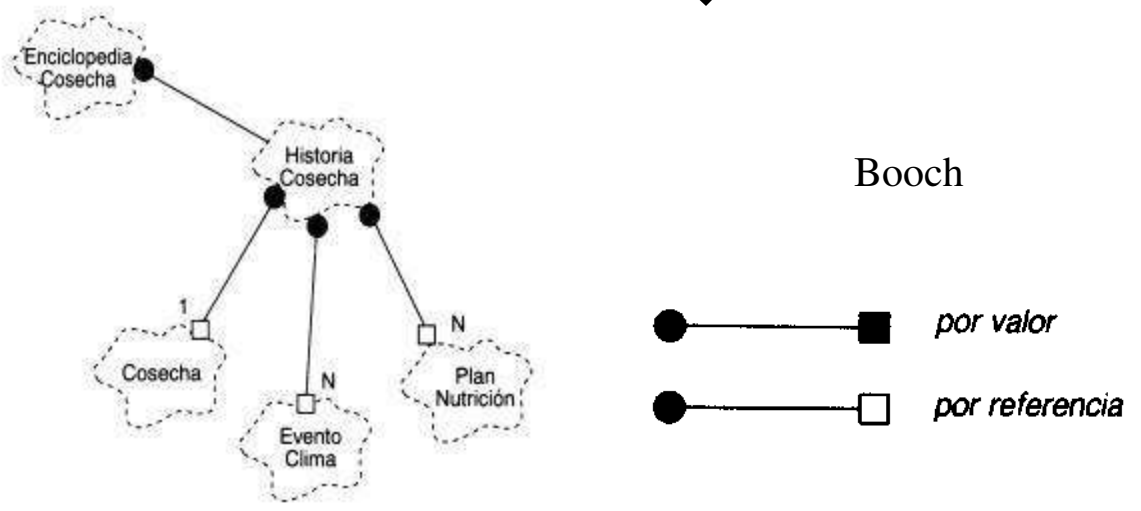


UML



Contención física

- La agregación es un caso particular de la asociación que denota posesión
- La agregación es una relación “todo/parte”
- La contención física puede ser de dos tipos:
  - ***por valor*** *indica que la construcción y destrucción de estas partes ocurre por construcción y destrucción del agregado. Asegura que los tiempos de vida del todo y su parte son iguales*
  - ***por dirección o referencia*** *indica que sólo contiene un puntero. Los tiempos de vida del todo y su parte son independientes*



Ejemplo: Control de inventario
Modelo estático

Versión preliminar 1.0

Código en C++ y diagrama de clases en notación Booch

```
#include "string.h"
#include "set.h"
class Num_elem;
class Elemento{
      public:
           virtual double Coste() const=0;
      private:
           const Num_elem& suNumeroElemento;
           String          suDescripcion;
     };
class Ensamblaje: public Elemento{
     public:
           virtual double Coste() const;
     private:
           Set<Elemento*> susElementos;
     };
class Pieza: public Elemento{
     public:
           virtual double Coste() const;
     private:
           double suCoste;
     };
```

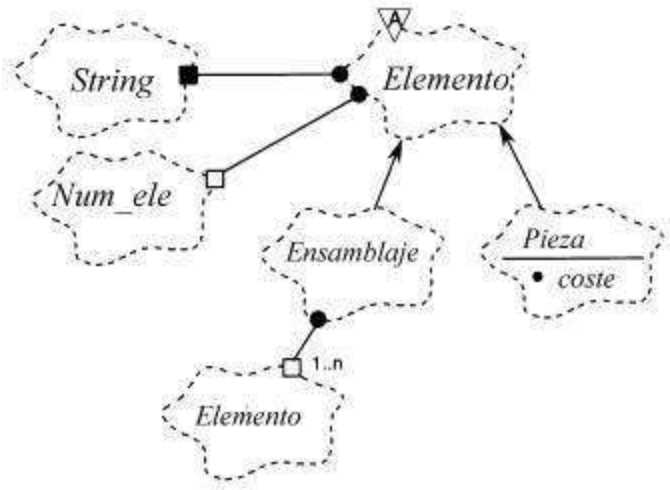
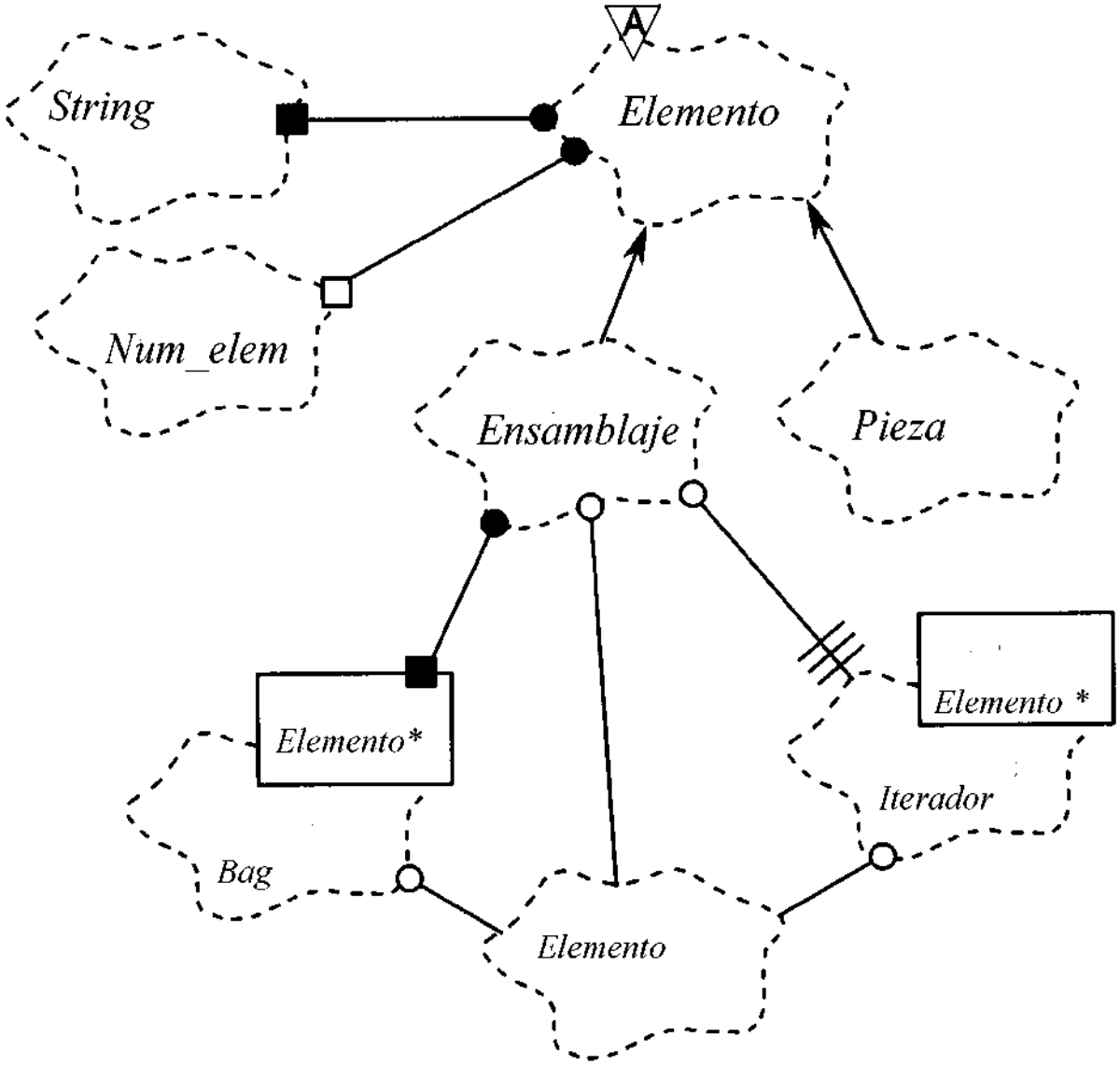


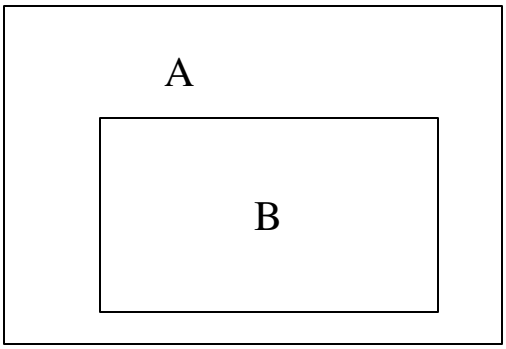
Diagrama de clases detallado
Versión 2.0



Anidamiento de clases

- Las clases pueden anidarse físicamente en otras clases
- La mayor parte de los lenguajes OO no permiten heredar las clases anidadas
- El anidamiento es una decisión táctica de diseño
- Rara vez existe una razón que obligue a anidar clases o categorías de clases hasta profundidades de uno o dos niveles
- No hay que confundir anidamiento con agregación

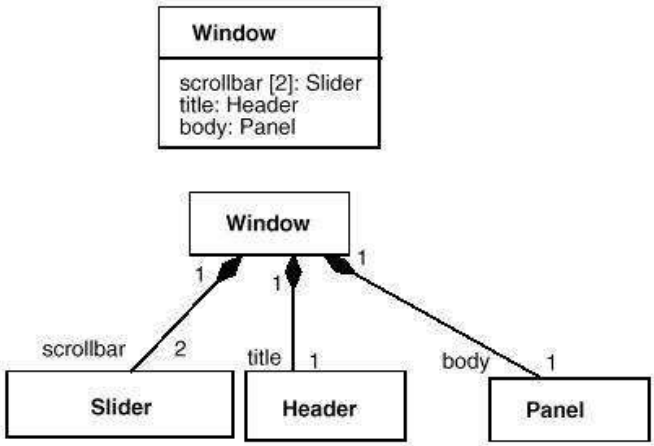
UML

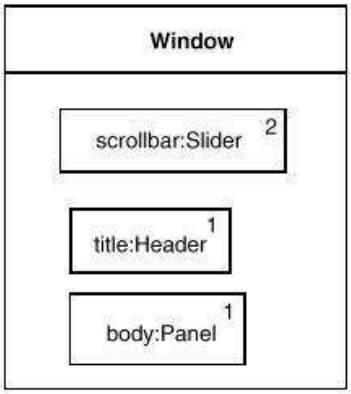


Booch

Composición en UML

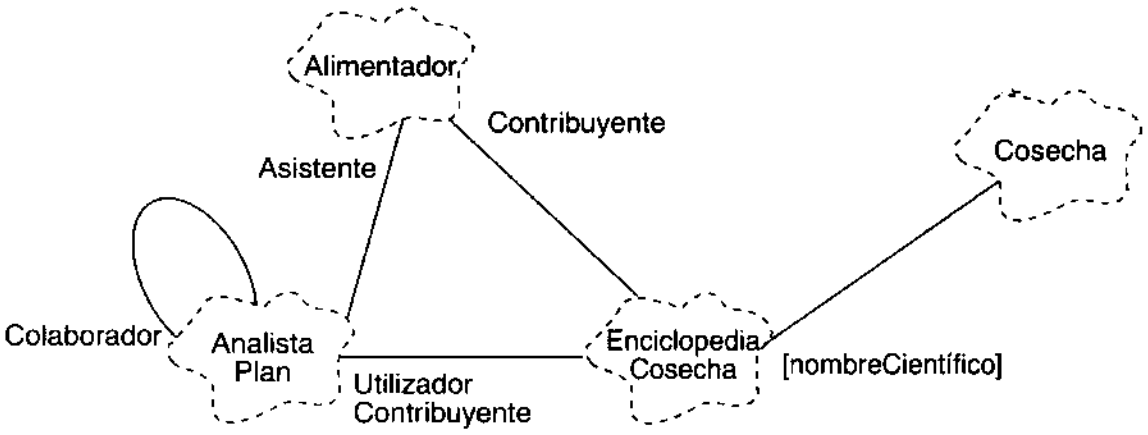
- Además de la agregación UML añade la composición
- Con la composición las partes sólo se puede pertenecer a un todo
- Si se elimina el todo se eliminan las partes
- Distintas formas de mostrar la composición en UML





Papeles (roles) y claves

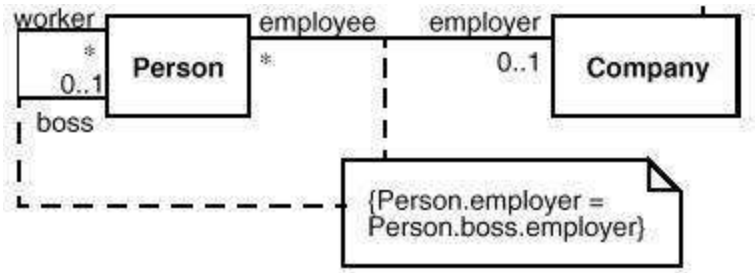
- *Un **papel (o rol)** denota el proposito o capacidad por la que una clases se asocia con otra*
- Se nombra el papel de una clase mediante una etiqueta ligada a una asociación, colocada adyacentemente a la clase que ofrece ese papel
- *Una **clave** es un atributo de la asociación entre clases cuyo valor identifica un objeto de manera única*
- Las claves aparecen como identificadores entre corchetes
- Una clave debe ser un atributo del objeto situado en el extremo destino de la asociación
- Son posibles las claves múltiples, pero los valores de clave deben ser únicos



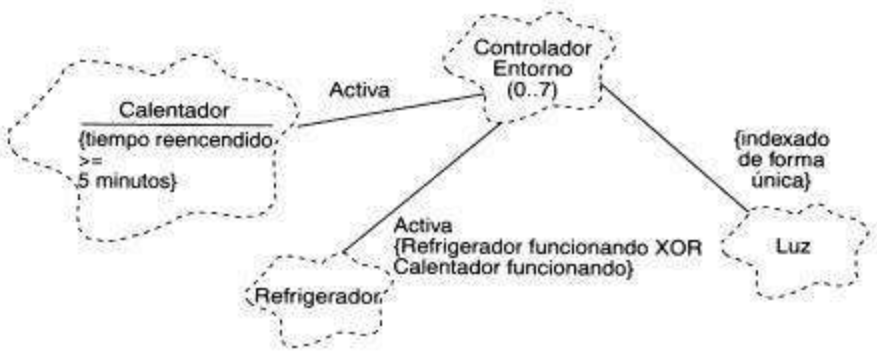
Restricciones

- *Una **restricción** es la expresión de alguna condición semántica que se debe preservar*
- *Una **restricción** es un invariante de clase o de relación que hay que cumplir mientras el sistema esté en un estado estable*
- La notación es una expresión entre llaves ({exp.}) adyacente a la clase o relación a la que se aplica la restricción
- Pueden escribirse expresiones de restricción que nombren a otras asociaciones o elementos de clases

UML



Booch

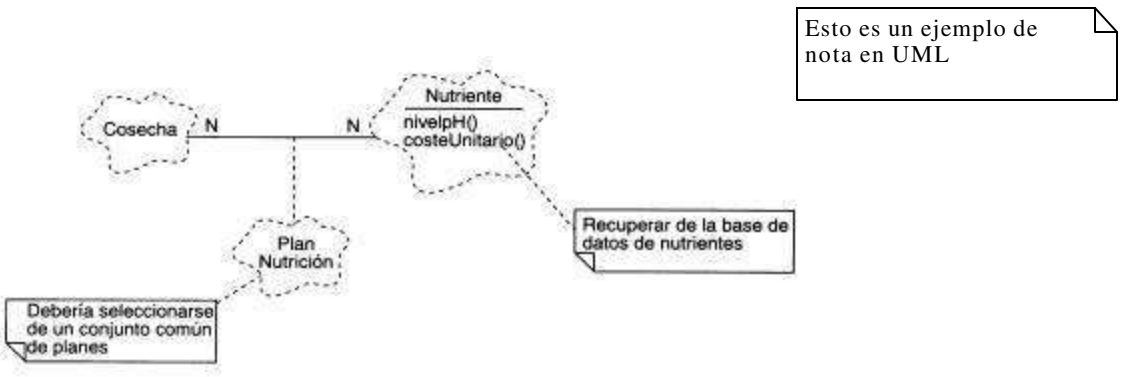


Asociaciones atribuidas y notas

- *Una **asociación atribuida** es una asociación que tiene una clase como atributo*
- Estas clases atributo permiten modelar permiten modelar las propiedades de las asociaciones
- La clase atributo se une mediante una línea discontinua a la línea que representa la asociación
- *Una **nota** es un comentario que se puede añadir a cualquier elemento del diagrama con el objetivo de recordar suposiciones y decisiones del desarrollador*
- El icono en forma de nota se une al elemento que afecta mediante una línea discontinua
- Es interesante especificar en las notas los requisitos a los que responde la implementación, permitiendo el seguimiento de los requisitos en el ciclo de vida del software

Booch

UML



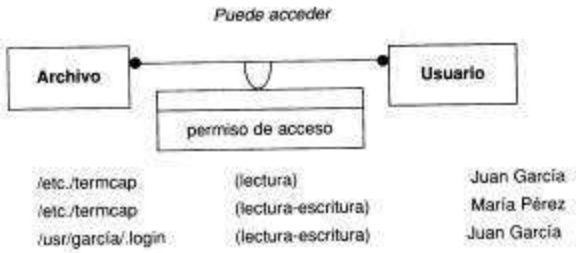
Especificaciones

- *Una **especificación** es una forma no gráfica que se usa para proporcionar la definición completa de una entidad de la notación*
- Una entidad de la notación puede ser una clase, una asociación, una operación individual o un diagrama completo
- **Elementos comunes**
  - Nombre: identificador
  - Definición: texto
- **Especificaciones de clases**
  - Responsabilidades: texto
  - Atributos: lista de atributos
  - Operaciones: lista de operaciones
  - Restricciones: lista de restricciones
  - Máquina de estados: referencia a la máquuina de estados
  - Control de exportación: public |implementación
  - Cardinalidad: expresión
  - Parámetros: lista de parámetros
  - Persistencia: transitorio |persistente
  - Concurrencia: secuencial | vigilada |síncrona |activa
  - Complejidad espacial: expresión
- **Especificaciones de operaciones**
  - Clase de retorno: referencia a una clase
  - Argumentos: lista de argumentos
  - Calificación: texto
  - Control de exportación: public|private|protected|implementación
  - Protocolo: texto
  - Precondiciones: texto|referencia cód. fuente|ref. diag. objetos
  - Postcondiciones: texto|referencia cód. fuente|ref. diag. objetos
  - Excepciones: lista de excepciones
  - Concurrencia: secuencial | vigilada | síncrona
  - Complejidad espacial: expresión
  - complejidad temporal: expresión

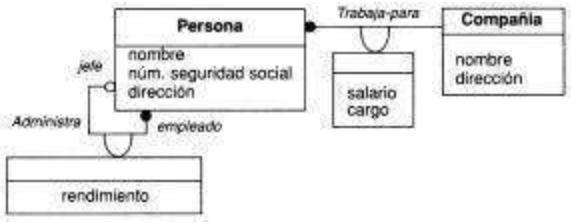
Atributos de enlaces y asociaciones
Ejemplos OMT

- *Un **atributo de enlace** es una propiedad del enlace de una asociación*
- Los atributos del enlace son una propiedad de la asociación y no de las clases enlazadas
- También puede haber atributos de enlace para asociaciones ternaria
  - En el ejemplo se relacionan las clases Jugador, Equipo y Partido. Así un **jugador** juega a lo largo de su vida profesional en distintos **equipos** en distintos **partidos**. En esta relación ternaria se le pueden asociar los atributos goles y tarjetas
- Atributos de enlace versus atributos de clases
  - Es preferible que los atributos estén en el enlace que en la clase pues permite más flexibilidad a cambios del modelo de clases

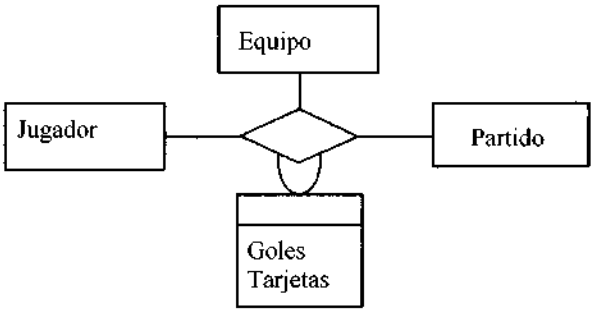
Asociación muchos-a-muchos con atributos de enlace

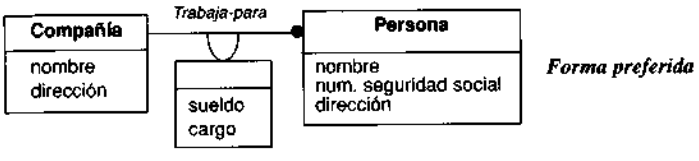


Atributos de enlace para asociaciones uno-a-muchos

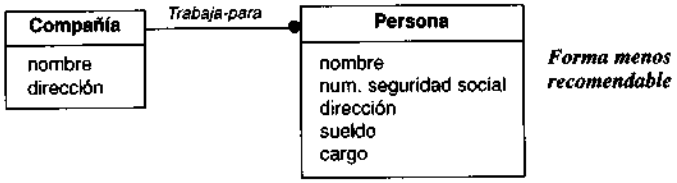


Atributos de enlace para una asociación ternaria



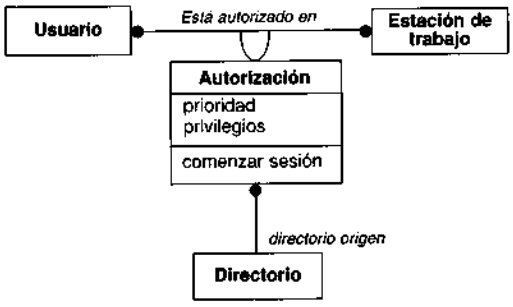


Forma preferida

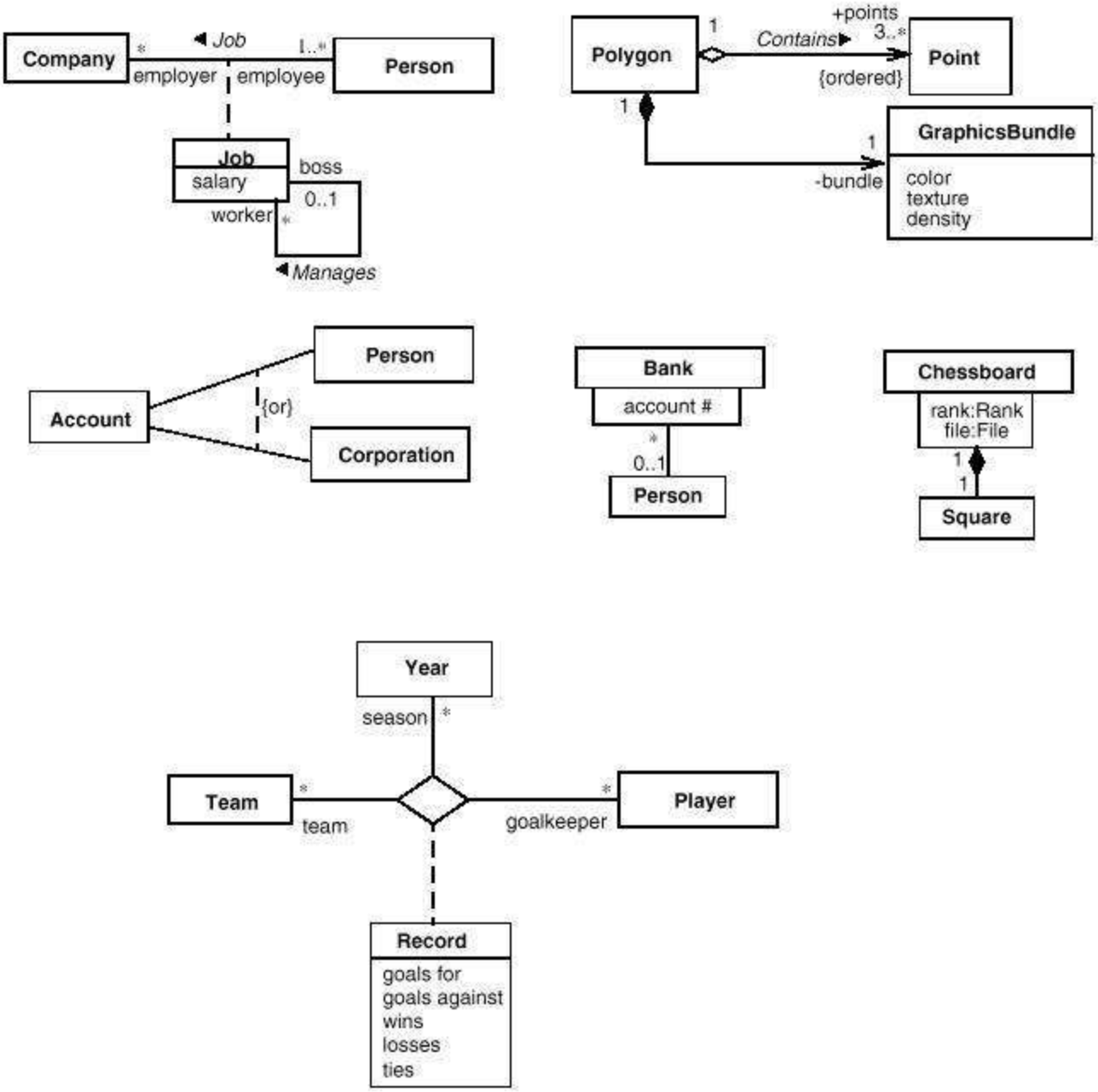


Forma menos recomendable

Atributos de enlace modelados como clases

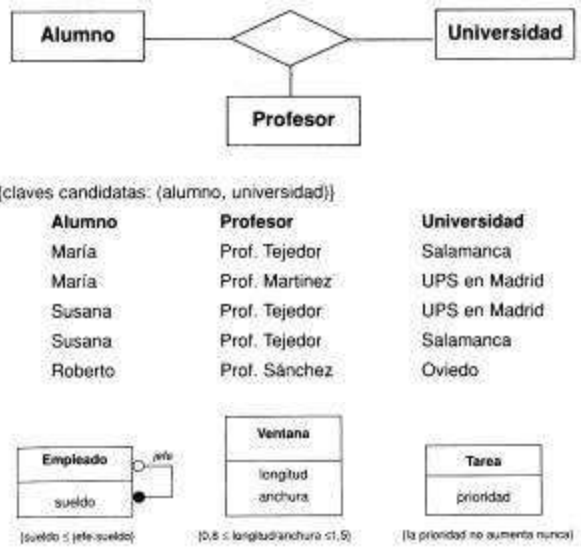


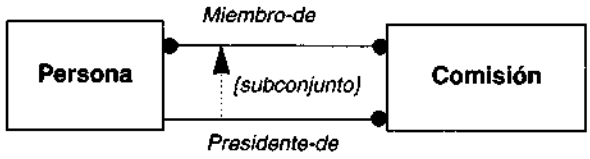
Ejemplos de asociaciones UML

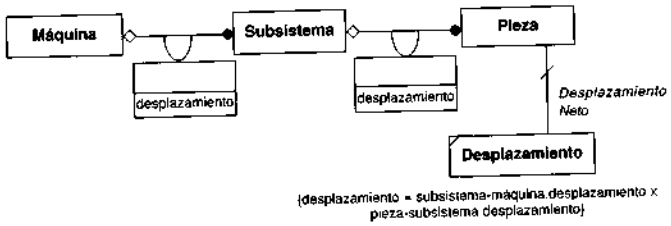


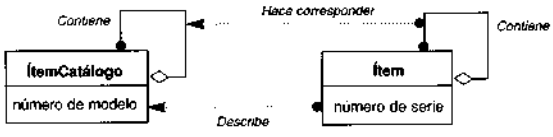
Claves, restricciones, atributos derivados y homomorfismos
Ejemplos OMT

- *Una **clave candidata** es un conjunto mínimo de atributos que define de forma única un objeto o enlace*
  - El identificador de un objeto es siempre una clave candidata de una clase
  - Para las asociaciones son claves candidatas una o más combinaciones de objetos relacionados
  - El uso de claves es típico del manejo de bases de datos, pero es un concepto lógico en los diagramas de clases
  - En el ejemplo *Alumno+Universidad* es la clave candidata
- *Las **restricciones** limitan los valores que pueden tomar las entidades*
  - Son necesarias varias iteraciones para darse cuenta de las restricciones del modelo
- *Las **restricciones generales** se expresan por medio del lenguaje natural o ecuaciones*
  - Se representan por una línea discontinua que une las clases o relaciones, colocándose un comentario entre llaves
  - En el ejemplo una asociación es un subconjunto de otra
- *Los **atributos derivados o calculados** se obtienen a partir del cálculo de otros atributos que también pueden ser derivados o calculados*
  - Se marcan con /
- *Las **clases derivadas o calculadas** se obtiene a partir de otras clases que a su vez pueden ser también derivadas o calculadas*
  - Se indican con una marca en la esquina superior izzquierda
- *Las **asociaciones derivadas o calculadas** se obtienen a partir de otras asociaciones que asu vez pueden ser también derivadas o calculadas*
  - Se indican con una marca en la línea de la asociación
- *Un **homomorfismo** establece una correspondencia entre dos asociaciones*



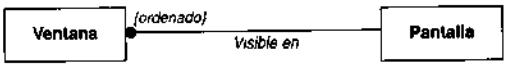




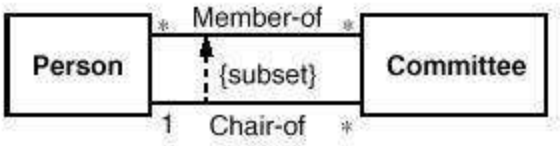


Papeles (roles), ordenación y calificación
Ejemplos OMT

- En una relación entre clases cada una de ellas juega un papel que se identifica con un nombre situado de forma adyacente a la clase en cada uno de los extremos de la línea que expresa la relación
- Puede haber más de una relación entre dos clases, y cada clase jugar distintos papeles para cada relación
- Cuando la parte “muchos” de una asociación está ordenada se puede indicar con el símbolo {ordenado} al lado del papel que juega la clase
- Una asociación entre clases puede tener restricciones a la multiplicidad por medio de atributos calificados

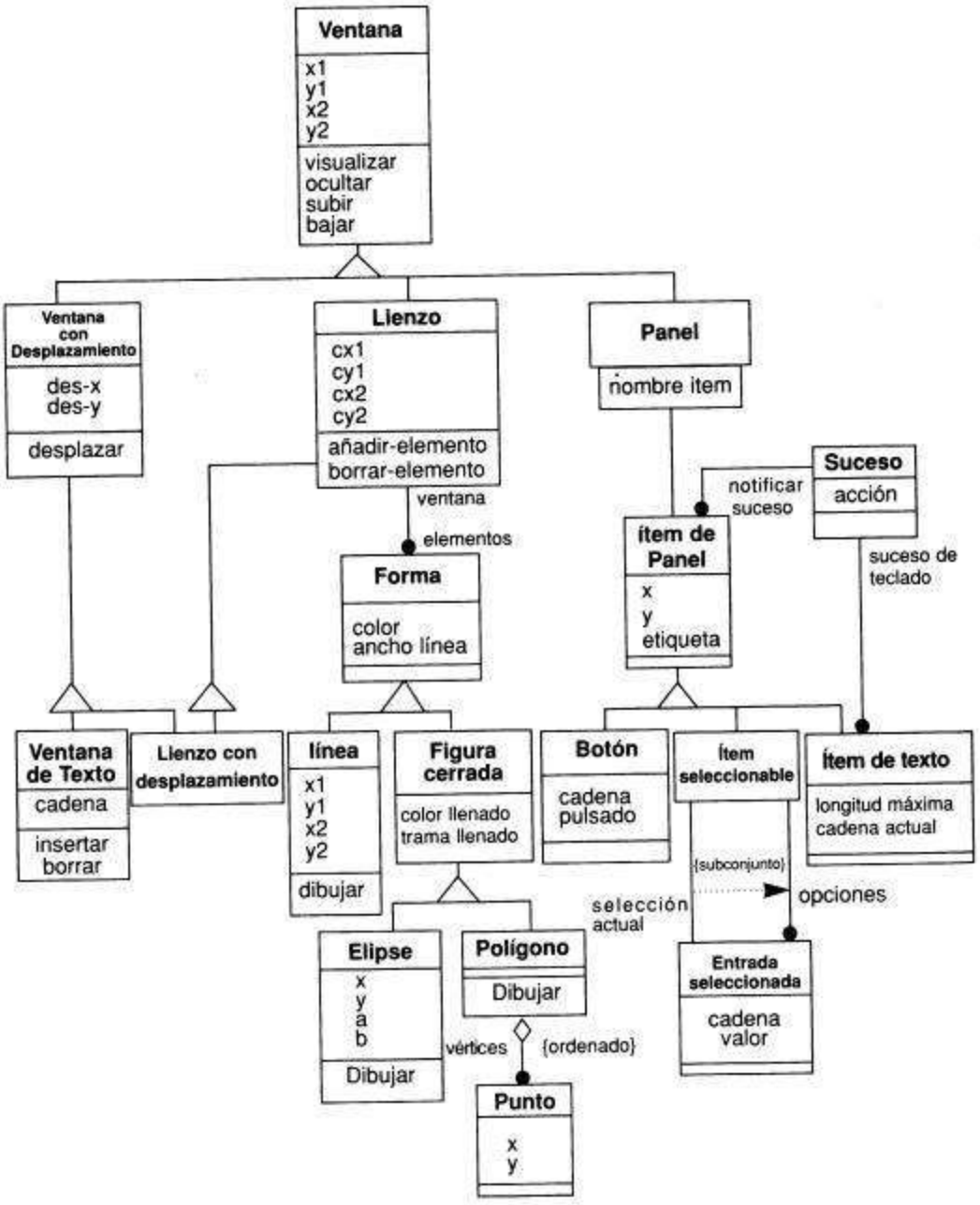


Restricciones en UML



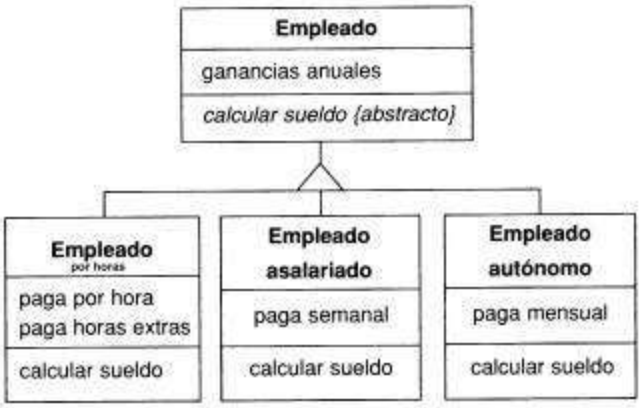
Ejemplo OMT

Diagrama de clases de un sistema de ventanas

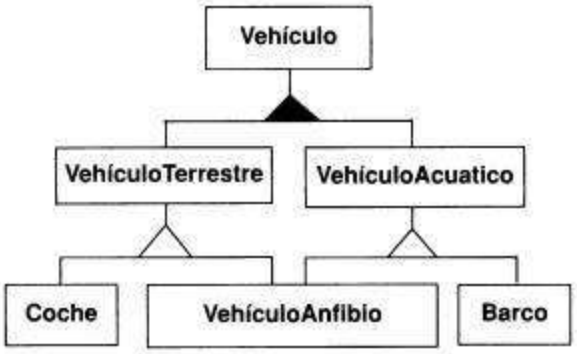


Clases abstractas y herencia múltiple
Ejemplos OMT

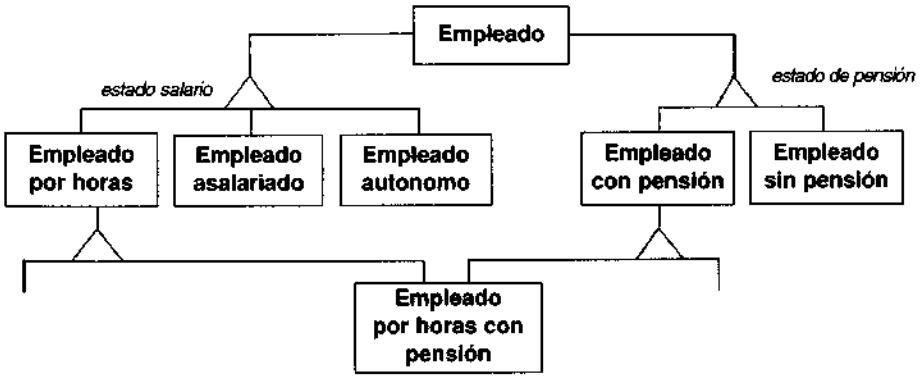
Ejemplo de clase abstracta y operaciones abstractas



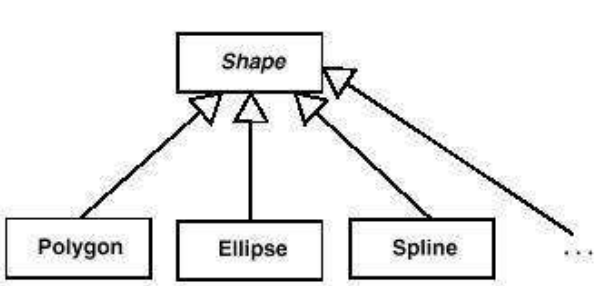
Herencia múltiple con clases solapadas

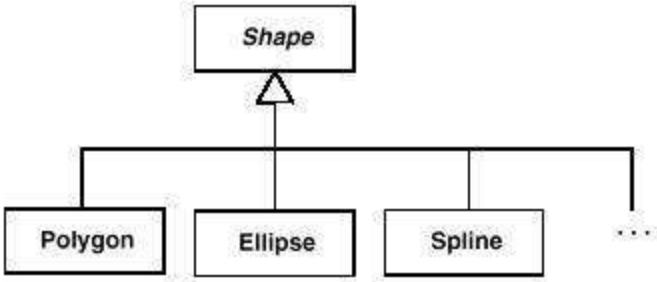


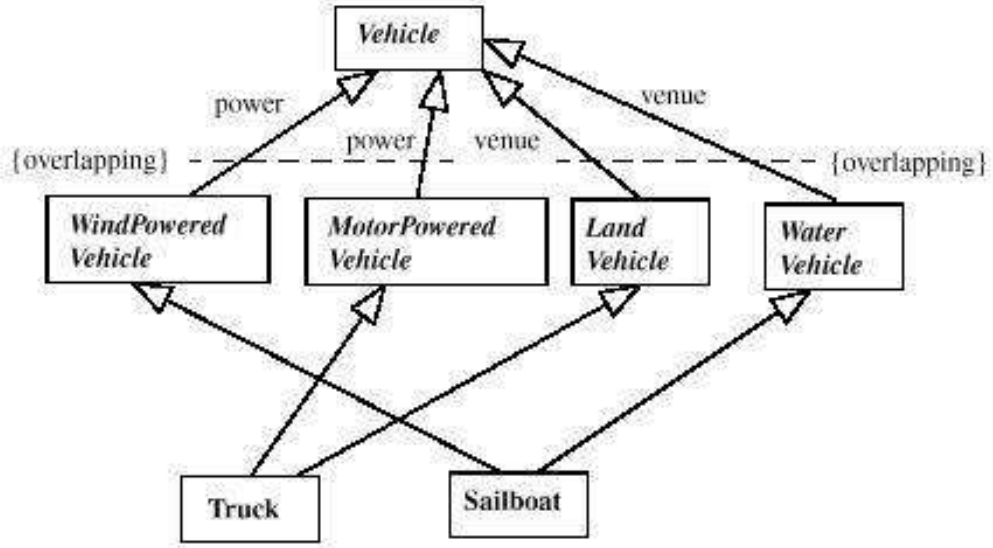
Herencia múltiple a partir de clases disjuntas



Herencia en UML







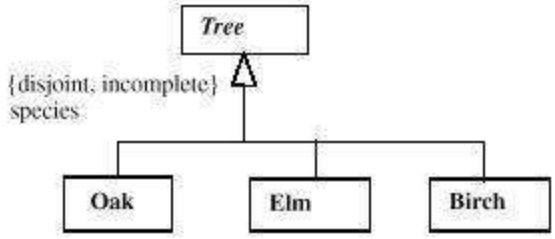


Diagrama de clases (Booch)

Muestra la existencia de clases y sus relaciones en el modelo lógico del sistema

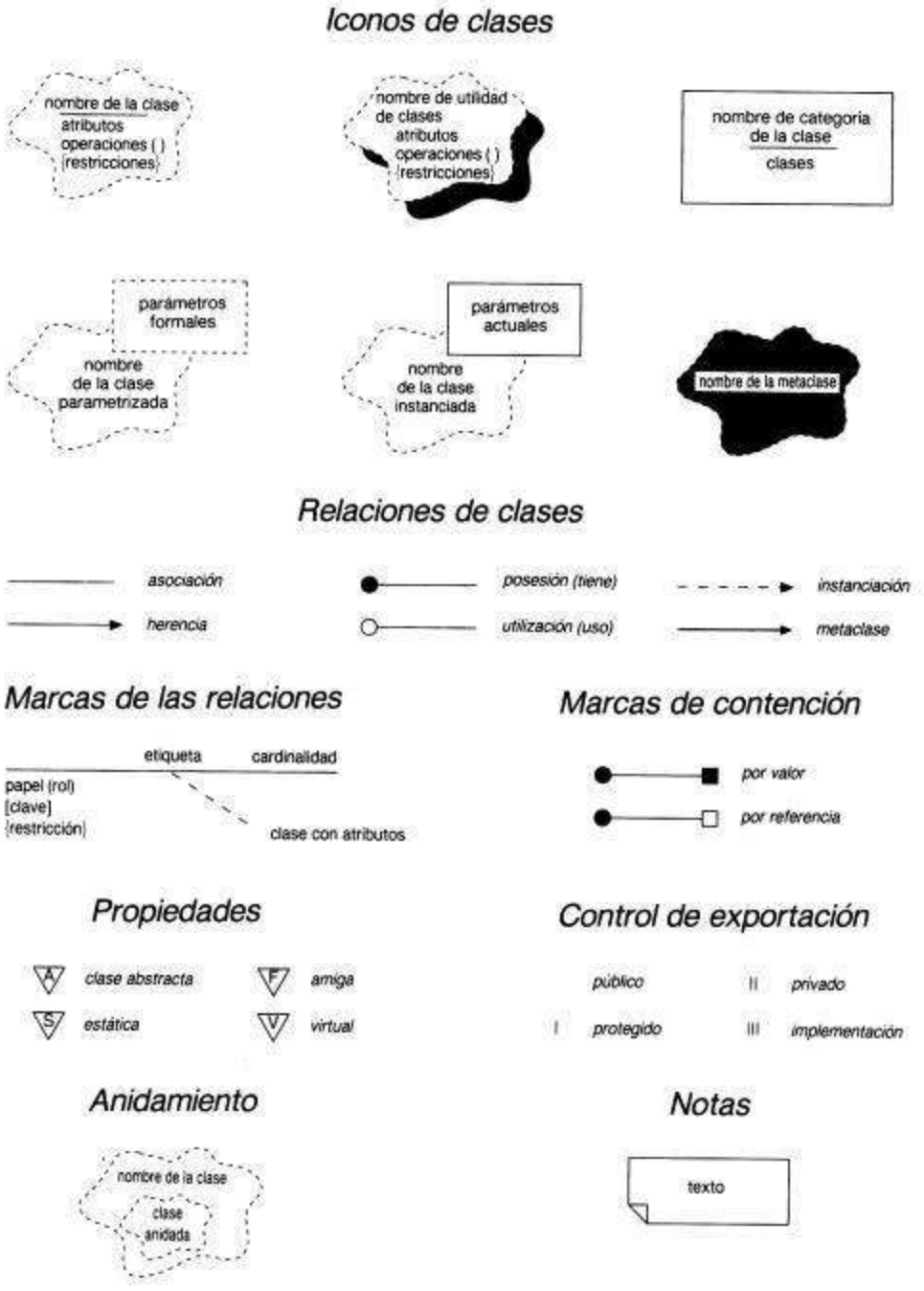


Diagrama de clases (OMT)
Básico

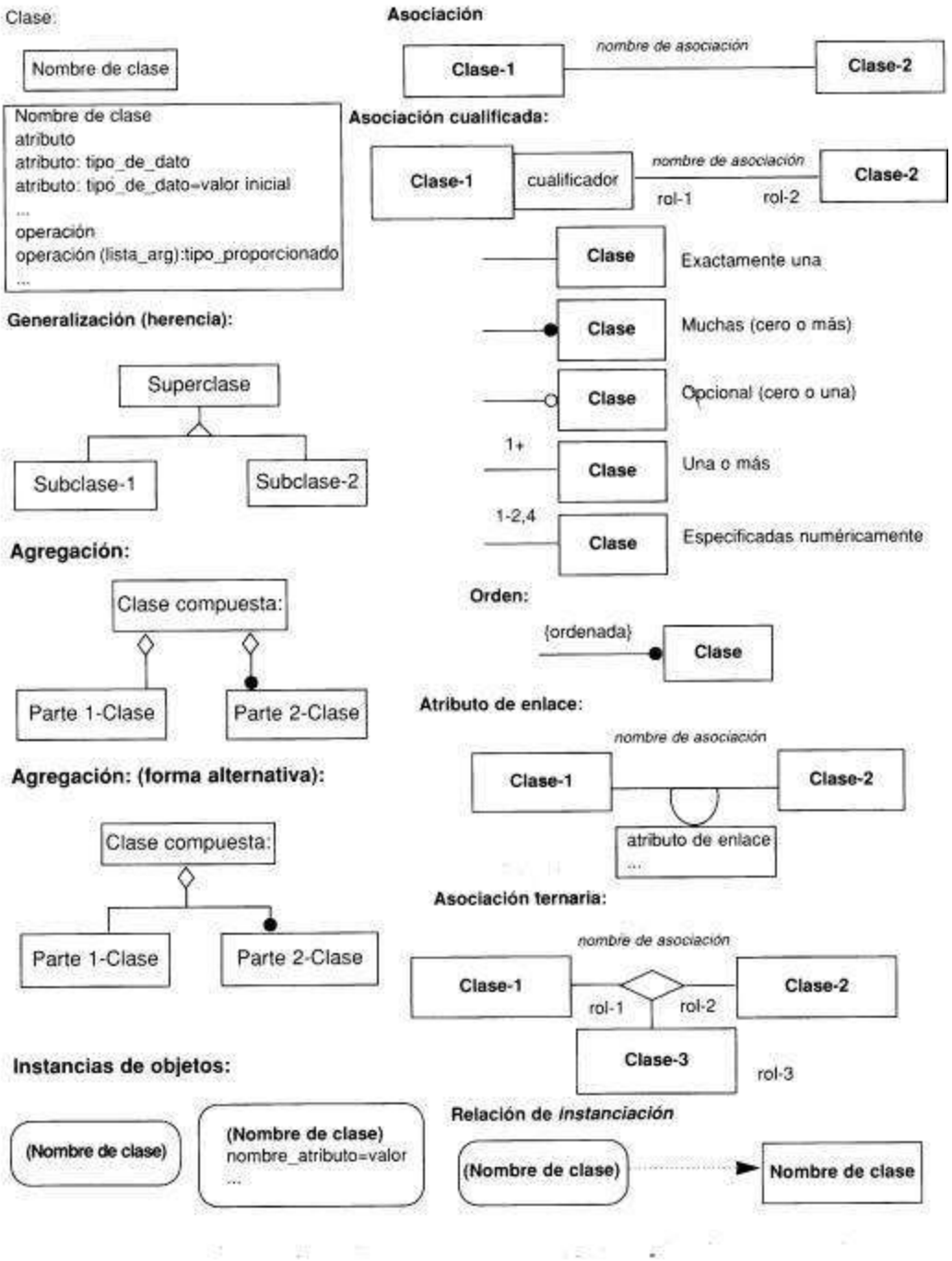
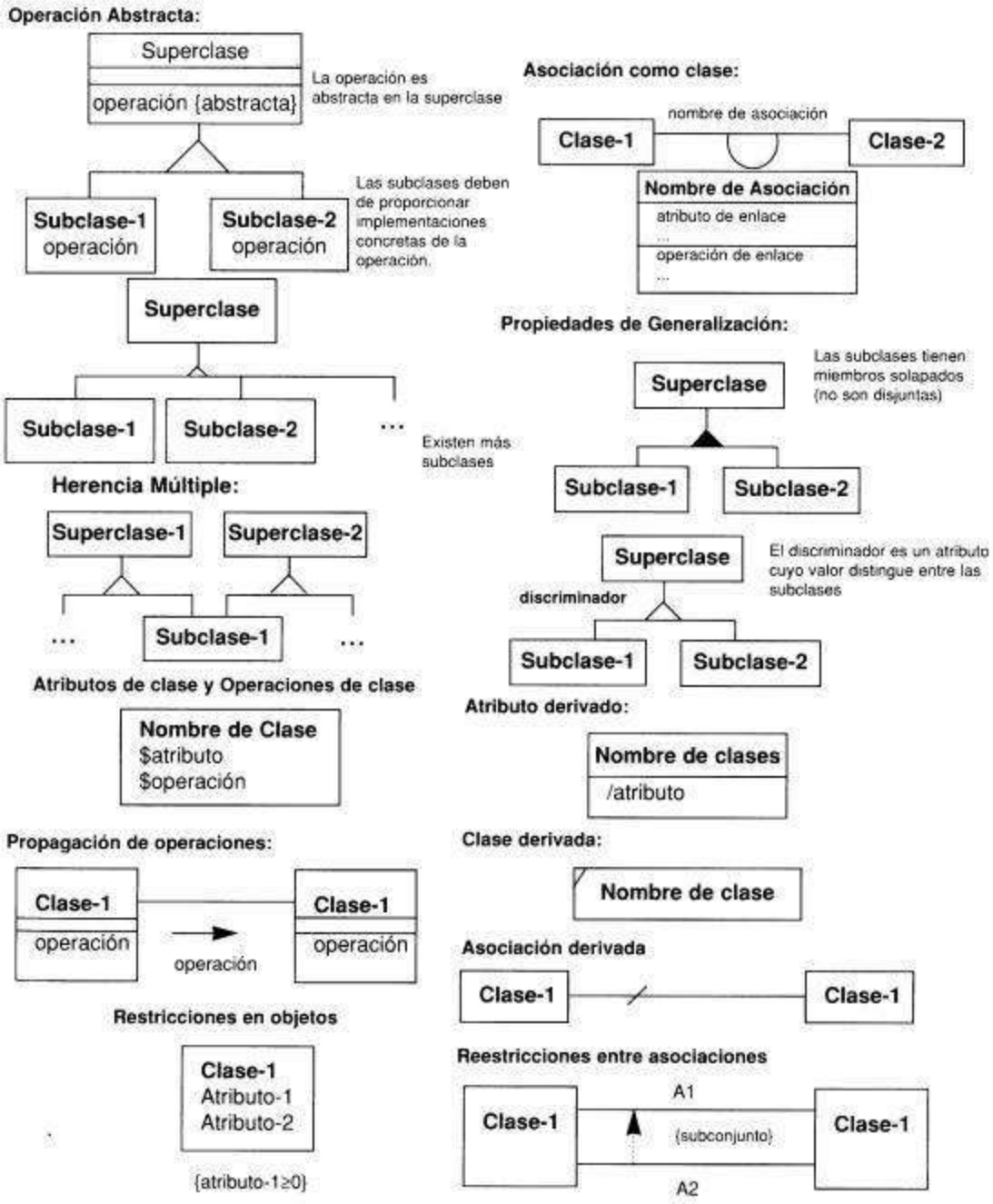
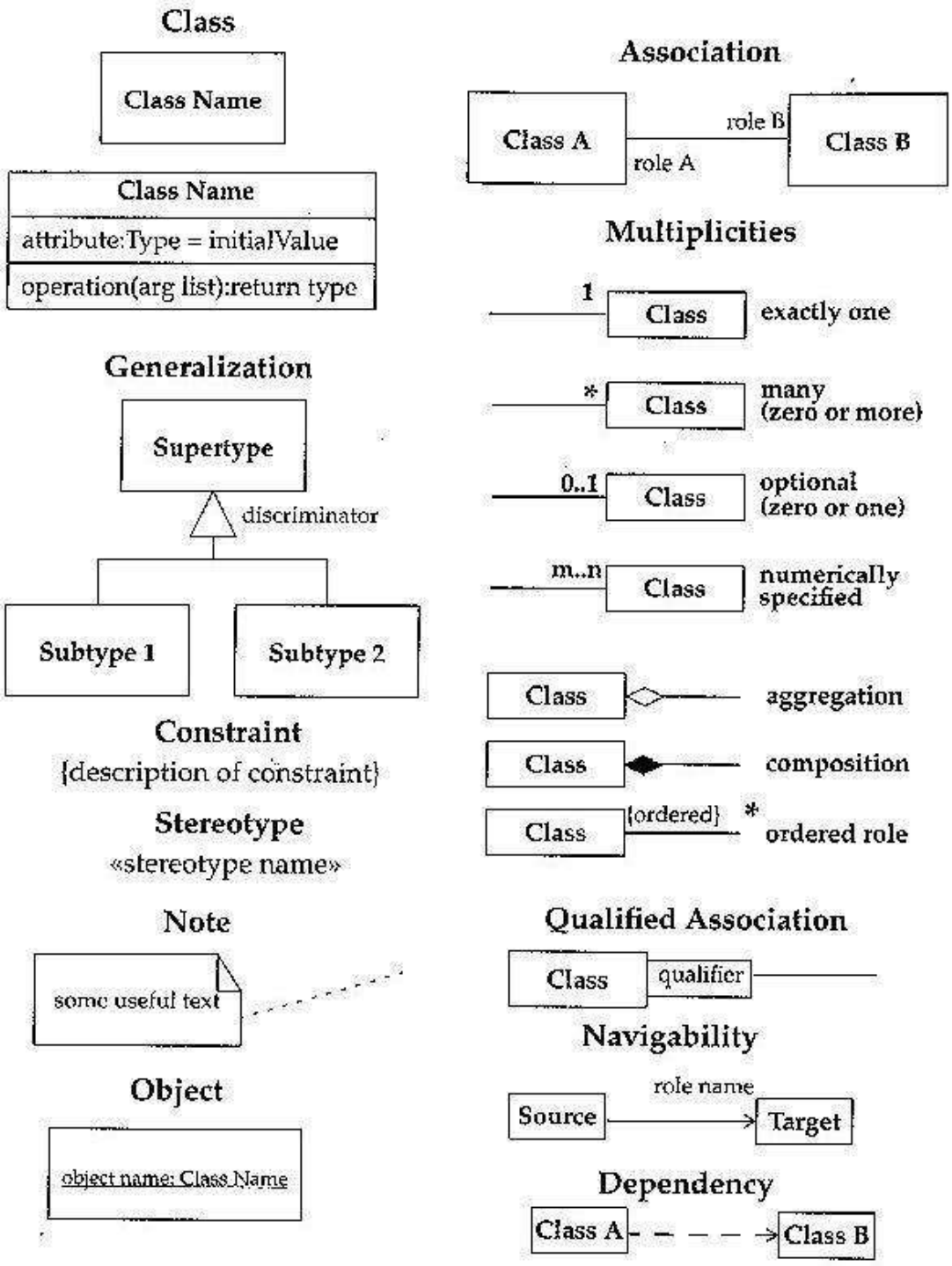


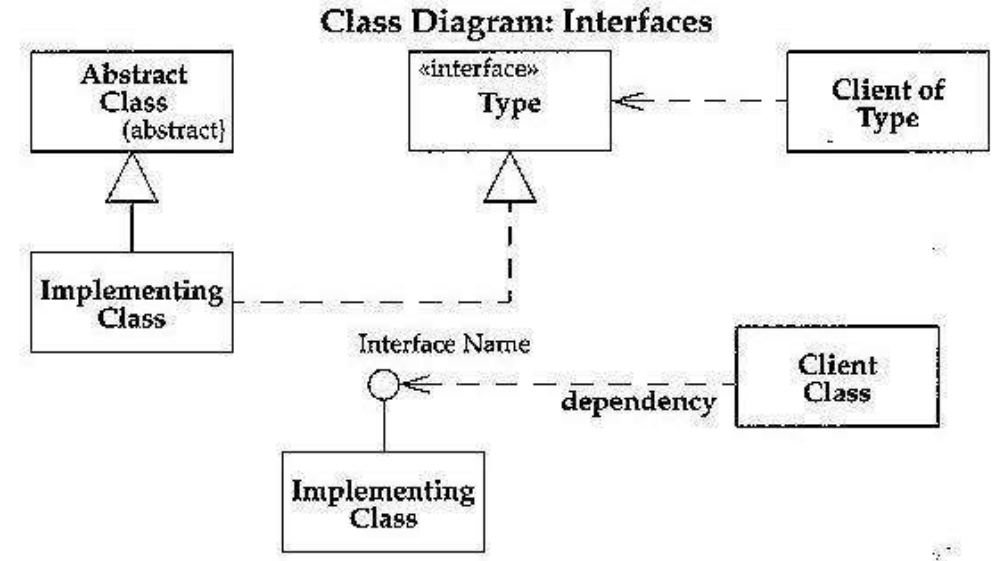
Diagrama de clases (OMT) Avanzado

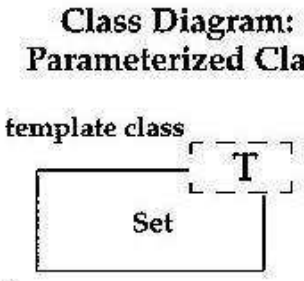


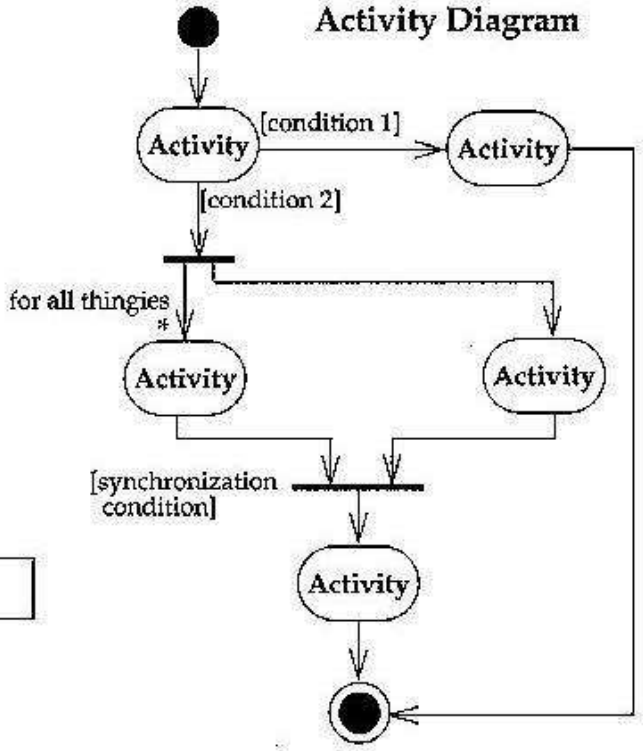
UML Resumen de la notación (a)

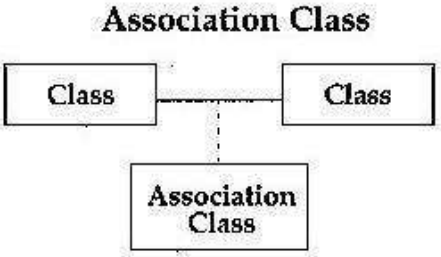


UML Resumen de la notación (b)





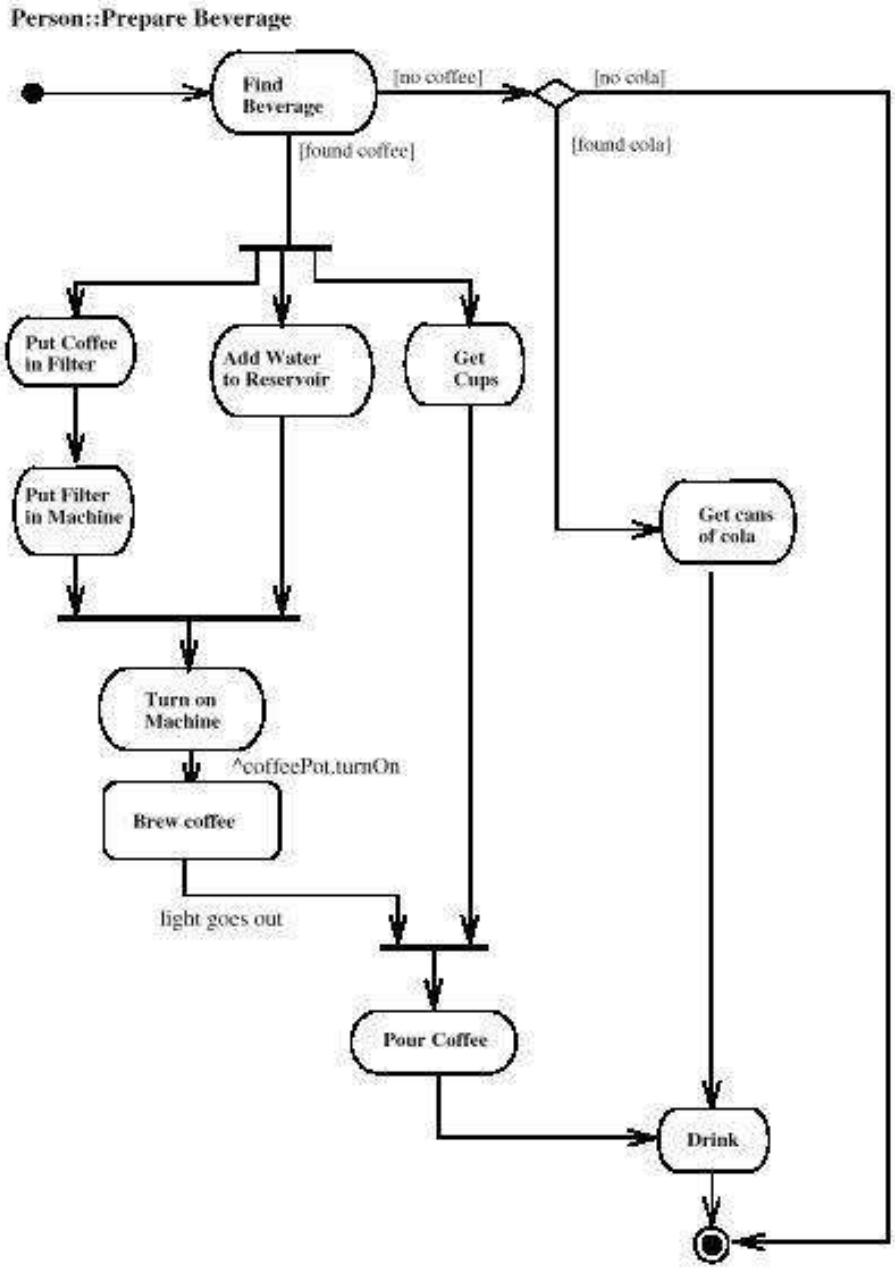




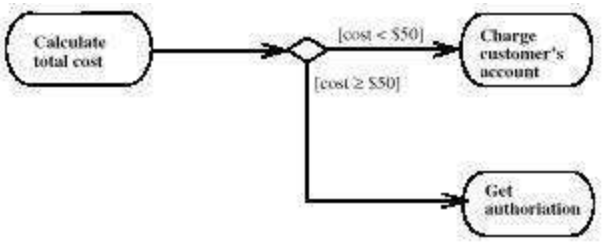
Diagramas de Actividad

- *Los **Diagramas de Actividad** representan como se dirigen los flujos de los procesos internos (en oposición a los eventos externos)*
- No están disponibles en Booch ni en OMT
- Cada diagrama de actividad se corresponde con los flujos que ocurren dentro de
  - Una clase
  - Una operación de una clase
  - Un caso de uso
- Se utilizarán
  - **Diagramas de Actividad** en situaciones donde todos o la mayoría de los eventos representan la totalidad de acciones generadas internamente (es decir procedimientos de control de flujo)
  - **Diagramas de Estados** en situaciones donde ocurren eventos asíncronos
- Los elementos del diagrama de actividad se denominan ***Actividades*** y se representan por un rectángulo redondeado

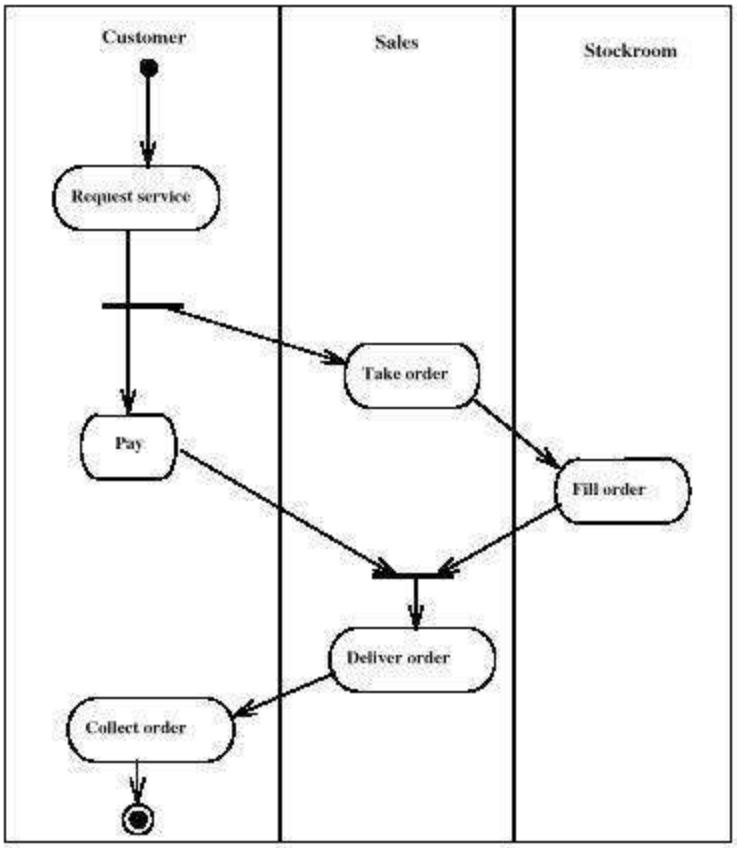
Ejemplo de Diagrama de Actividad



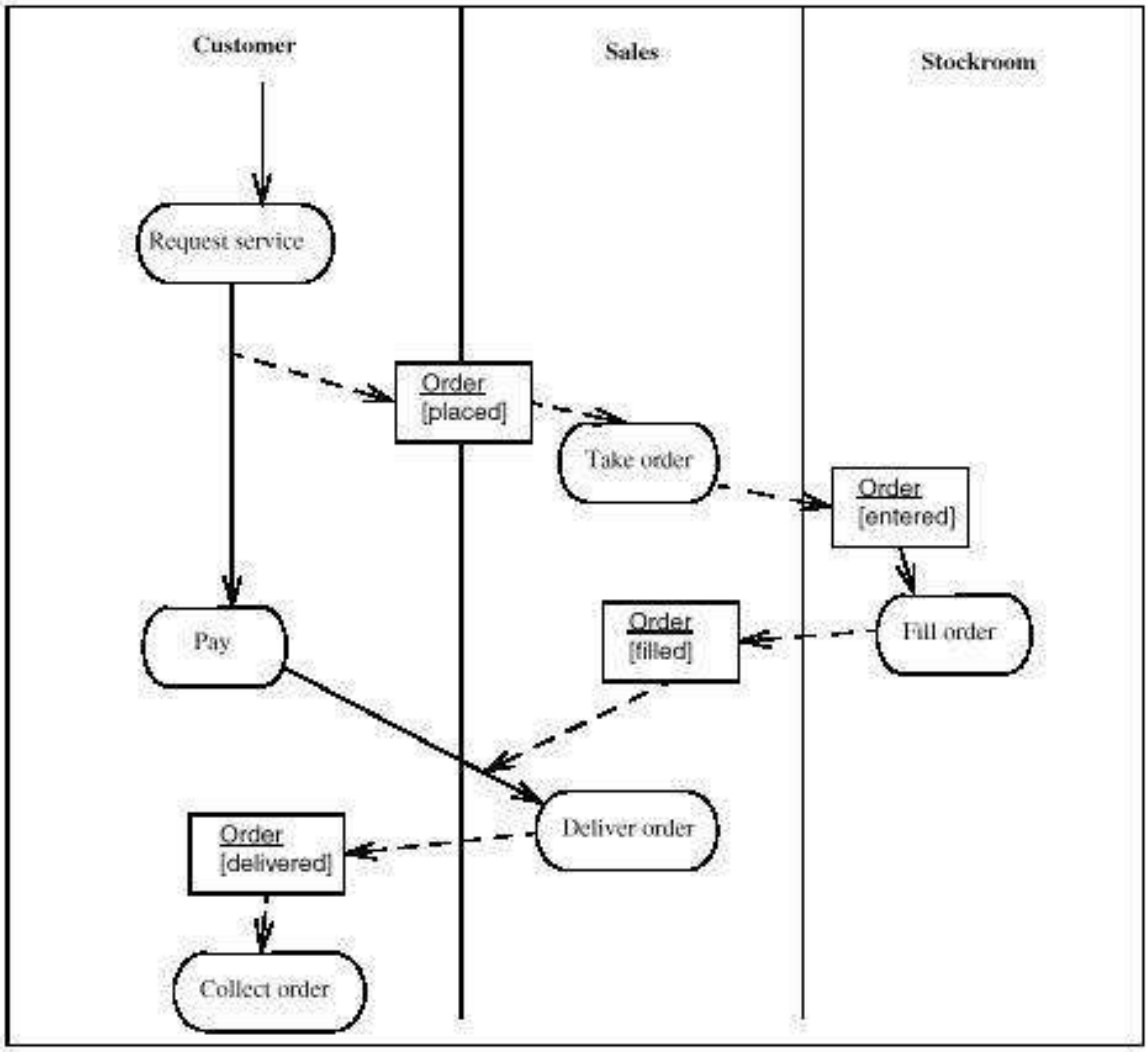
Decisiones



Calles o pasillos de actividades *(swinlanes)*

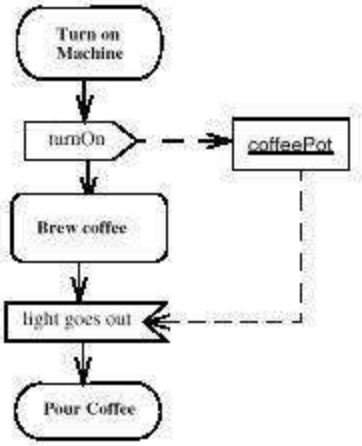


Relaciones de flujo acciones-objetos



Iconos de control

A) Símbolos de envío y recepción de señales



B) Eventos aplazados

Diagramas de Estados

- Se denominan
  - Diagramas de Transición de Estados en Booch
  - Diagramas de Estados en OMT
  - Diagramas de Estados en UML
- Los diagramas de estados muestran la secuencia de estados por los que pasa un objeto durante su vida y que se corresponden con los estímulos recibidos, junto con sus respuestas y acciones
- Cada diagrama de estados se corresponde con una clase o con un método

Diagramas de transición de estados

Muestran a las clases desde la lógica del sistema

Notación de Booch

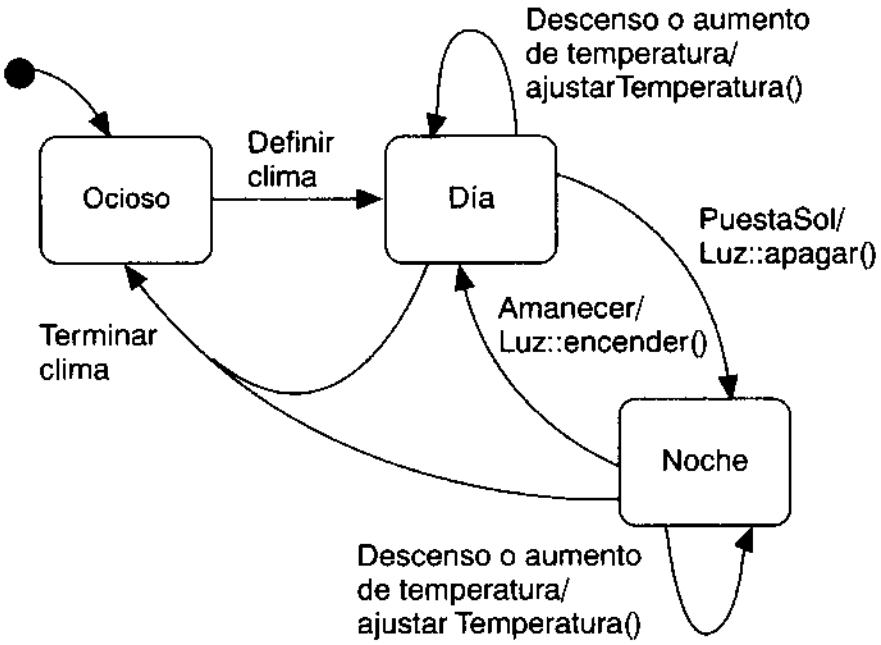
- *Un **diagrama de transición de estados** muestra el espacio de estados de una clase determinada, los eventos que provocan una transición de un estado a otro y las acciones que resultan de ese cambio de estado*
- *Un **estado de un objeto** representa los resultados acumulados de su comportamiento*
- Es necesario un nombre para cada estado. Además se pueden poner las acciones asociadas a cada estado
- *Un **evento** es algún suceso que puede causar un cambio de estado en el sistema*
- Un evento puede disparar una acción
- Hay un estado inicial por defecto
- Puede haber o no estado final o de parada
- La máquina de estados deja de existir cuando se destruye el objeto que la contiene

Diagramas de transición de estados

Ejemplo en la notación de Booch

Diagrama de estados de la clase ControladorEntorno

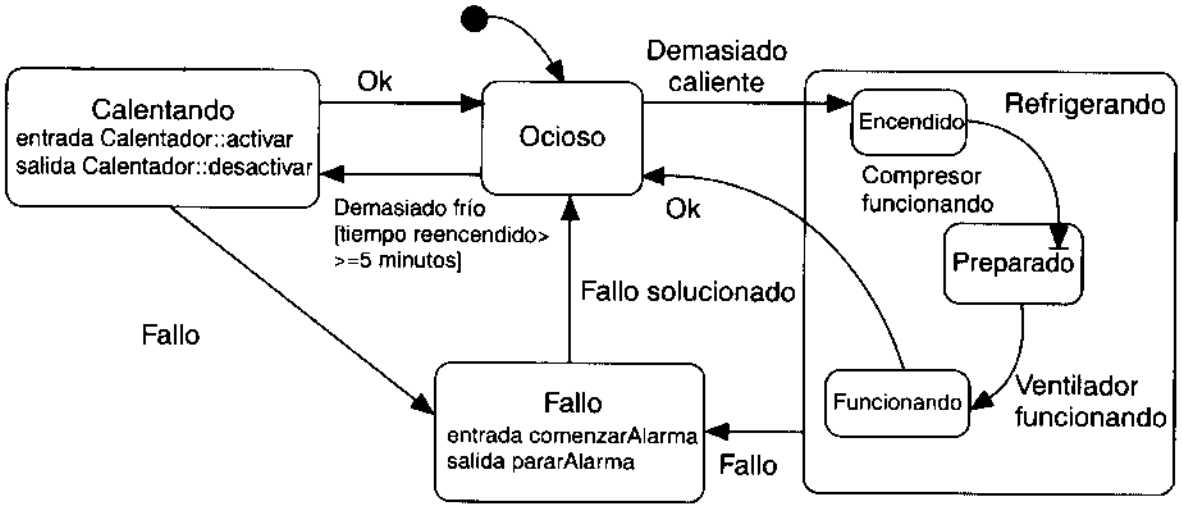
- Se comienza con el estado *Ocioso*
- Con el evento *Definir-Clima* hay un cambio de estado
  - Se ha supuesto que este evento sólo puede ocurrir de día
- Se alterna entre los estados *Dia* y *Noche*
- Se controla la temperatura tanto de dia como de noche
- Si se recibe el evento *Terninar-Clima* se vuelve al estado *Ocioso*



Diagramas de transición de estados

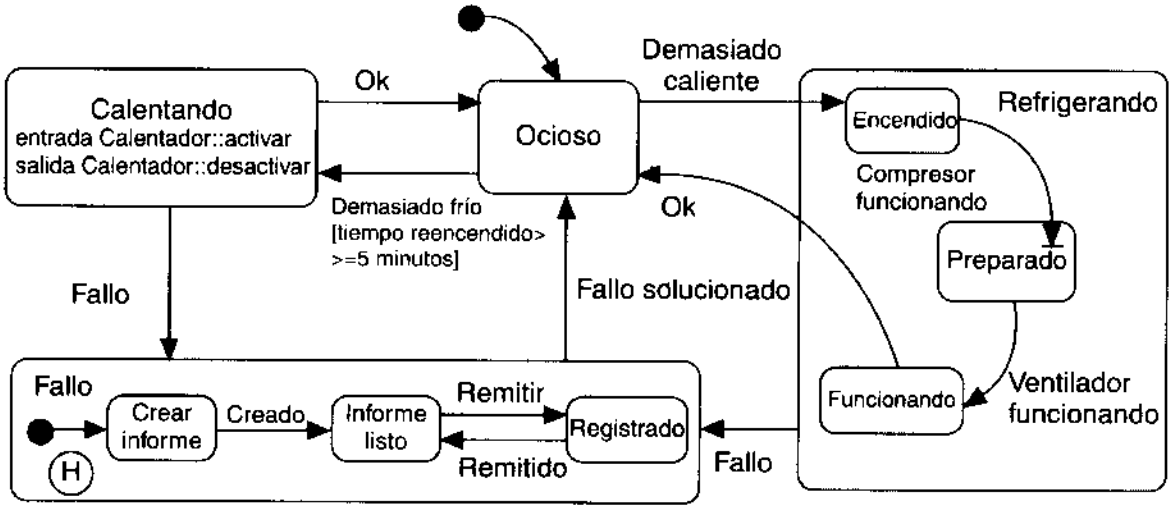
Acciones, transiciones condicionales y estados anidados
Ejemplo en notación de Booch

- Se pueden *asociar acciones a los estados* indicando la acción que ocurre cuando se entra o sale del estado
  - Ejemplo: en el estado *Calentando*
    - entrada Calentador::activar
    - salida Calentador::desactivar
- Se pueden representar ***transiciones de estado condicionales*** (o vigiladas) por medio de una expresión booleana entre llaves
  - Ejemplo: en la transición *Demasiado-frio*
    - *[tiempo de encendido>=5 minutos]*
- Se pueden detallar los estados utilizando ***estados anidados***
  - Los estados continentes se llaman *superestados*
  - Los estados anidados se llaman *subestados*
  - Las transiciones desde y hacia subestados dento de un superestado se hacen utilizando *flechas romas*
    - Ejemplo: Transición de *Encendido* a *Preparado*
- *No debería denominarse **Fallo** un estado y un evento*

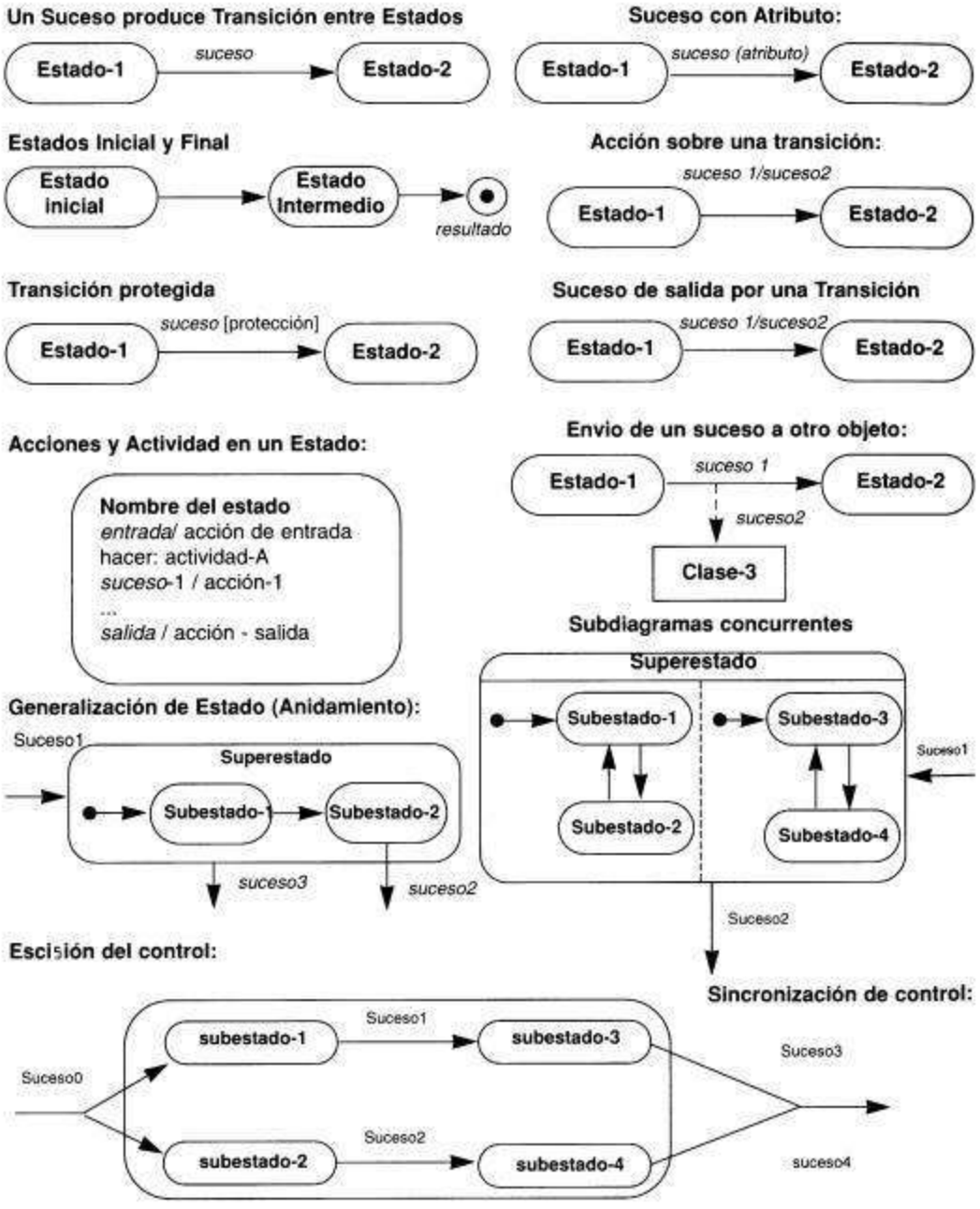


Diagramas de transición de estados
Historia, estados ortogonales y especificaciones
Ejemplo en notación de Booch

- **Historia**. *Se utiliza cuando se quiere pasar por un estado una sóla vez dentro de un superestado con subestados*
  - En el ejemplo
    - Por el estado *Crear-Informe* sólo se pasa la primera vez
  - Se utiliza una H en el superestado y un círculo relleno con una flecha que señala al estado por el que se pasa sólo una vez
- **Estados ortogonales** o concurrentes. La notación de Booch no los utiliza pues ese comportamiento se refleja en los diagramas de objetos
- **Especificaciones**. También se pueden especificar en modo texto la definición completa de un diagrama de transición de estados. Aunque en general no deben de añadir información nueva



Notación del modelo dinámico OMT



Diagramas de estados
Ejemplos en OMT

Diagrama de estados simplificado de la clase ajedrez

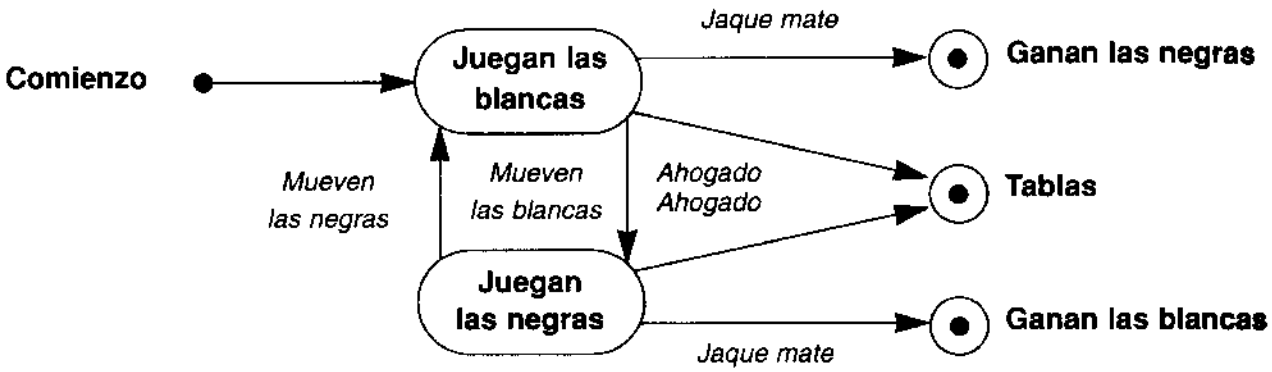
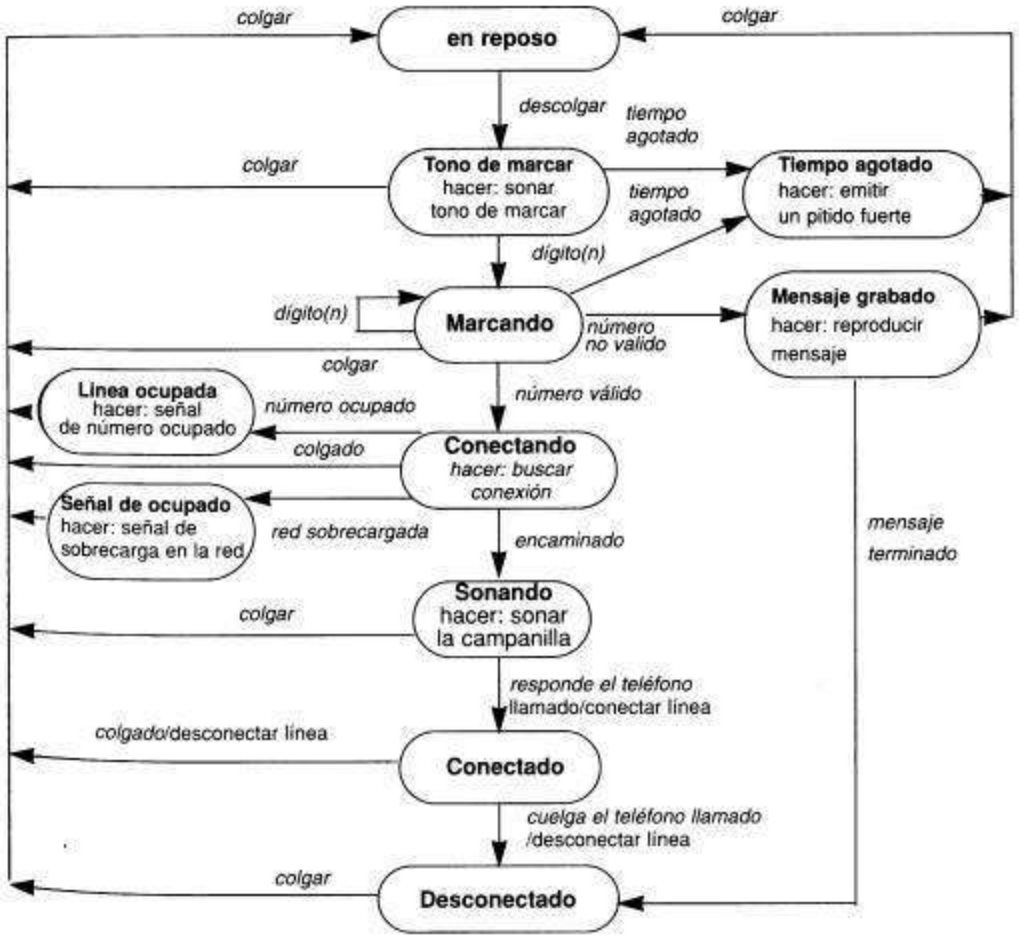
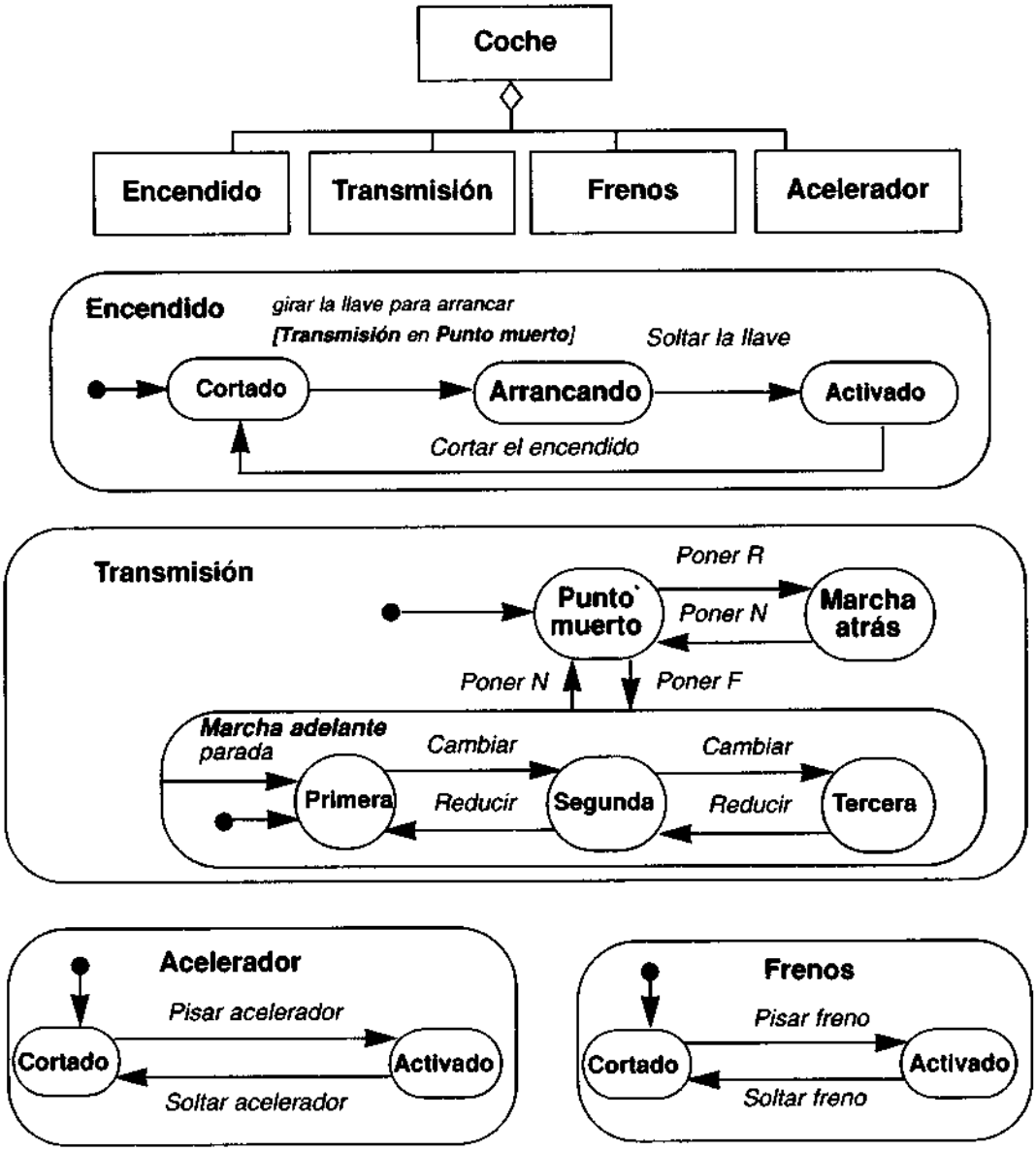


Diagrama de estados de la clase teléfono

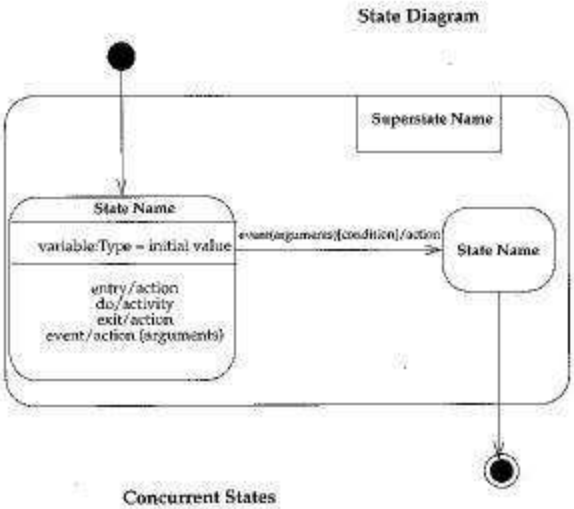


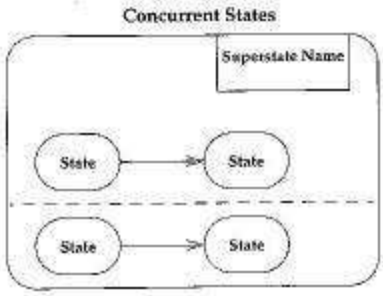
Diagramas de estados
Concurrencia de agregación (OMT)

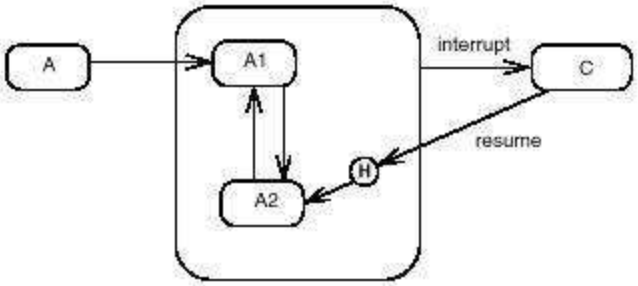
Cada estado componente experimenta transiciones en paralelo a todos los demás. Los diagramas de estados de los componentes son independientes, excepto si hay expresiones de protección como la que se muestra en la clase encendido que obliga a encender con [Transmisión en punto muerto]

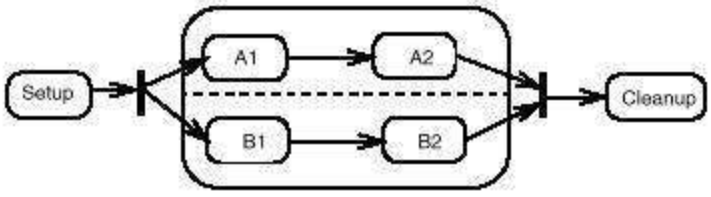


Diagramas de estados en UML

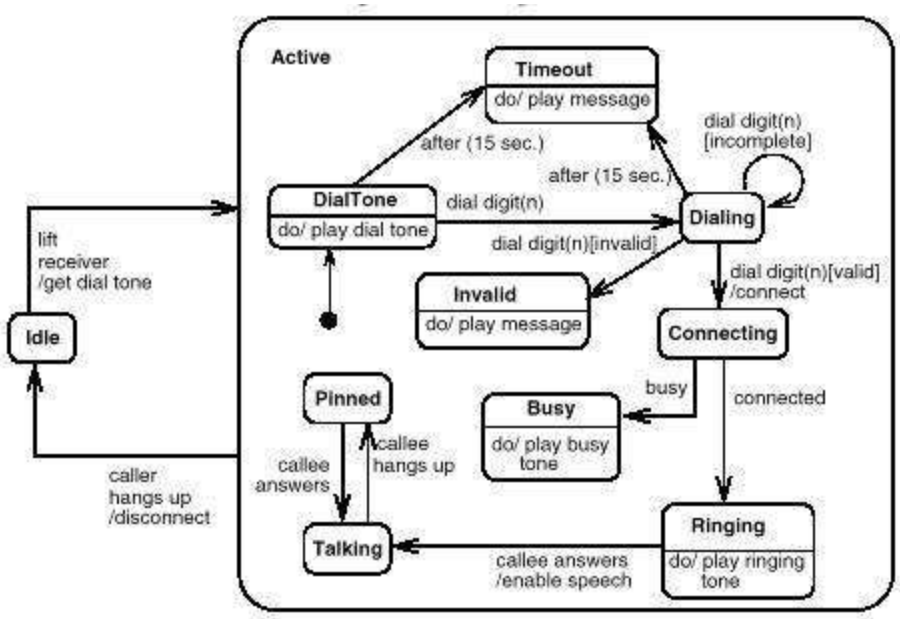


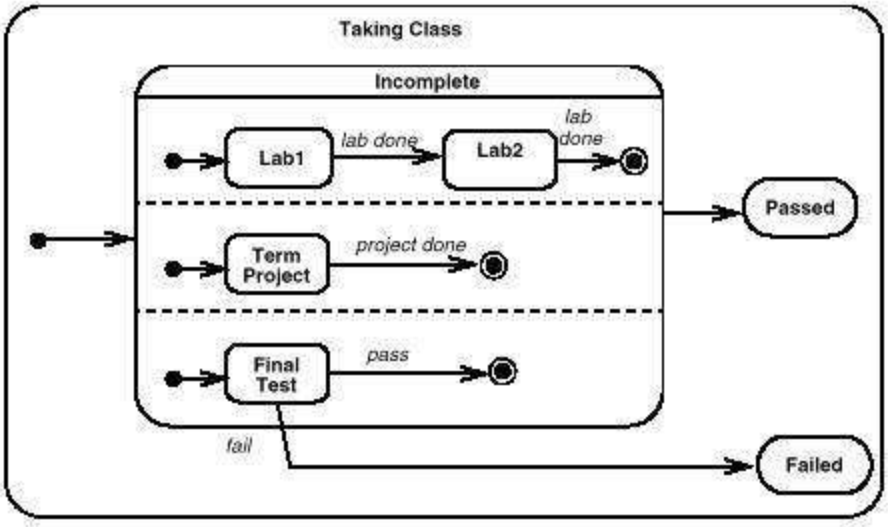




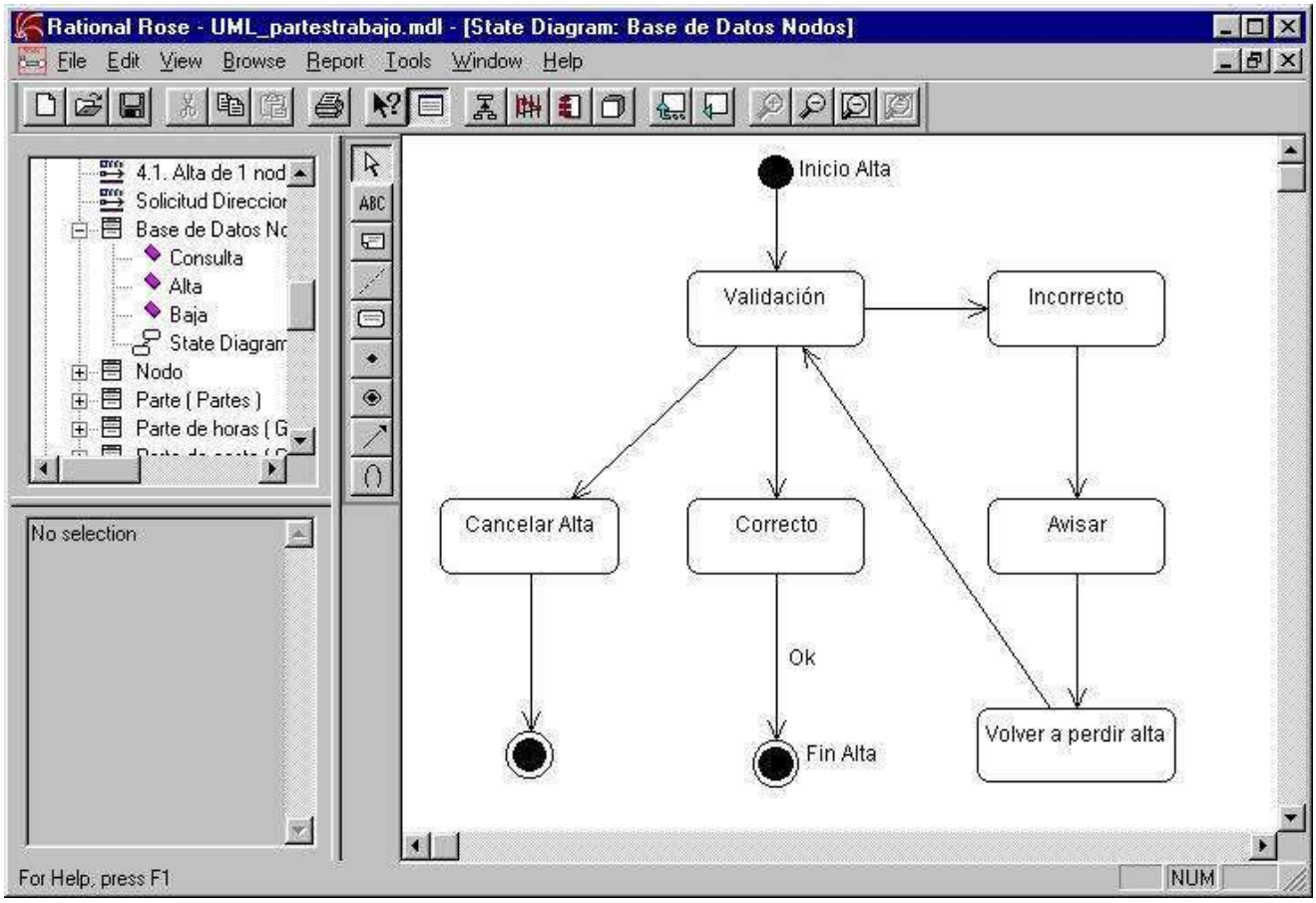


Ejemplos de diagramas de estados en UML





Ejemplo de Diagrama de Estados en Rose



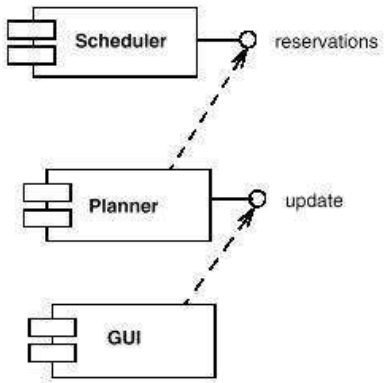
Diagramas de Implementación

- Los diagramas de implementación como su propio nombre indica representan aspectos relativos a la implementación incluyendo la estructura del código fuente y otras características propias del tiempo de ejecución
- UML tiene dos tipos de diagramas de implementación
  - ***Diagramas de Componentes***
    - Muestran la estructura del código fuente
    - Se corresponden con
      - Categorias de clases en Booch
      - Paquetes de UML
      - Diagramas de Módulos en Booch
  - ***Diagramas Desplegables*** *(Deployment Diagrams)*
    - Muestran la estructura del sistema en tiempo de ejecución
    - Se corresponden con los Diagramas de Procesos de Booch

Diagramas de Implementación UML

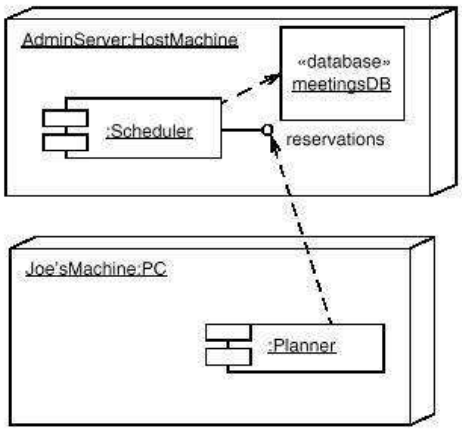
a) Diagrama de Componentes

Muestran la dependencia entre los componentes software



b) Diagramas Desplegables

Son grafos de nodos conectados por asociaciones de comunicación

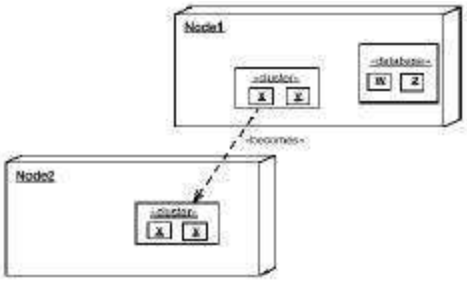


c) Nodos

• Son objetos físicos en tiempo de ejecución que representan algún recurso que se pueda procesar.

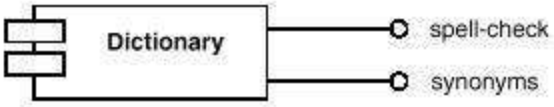
• Los nodos incluyen periféricos

•Pueden representarse como tipos e instancias



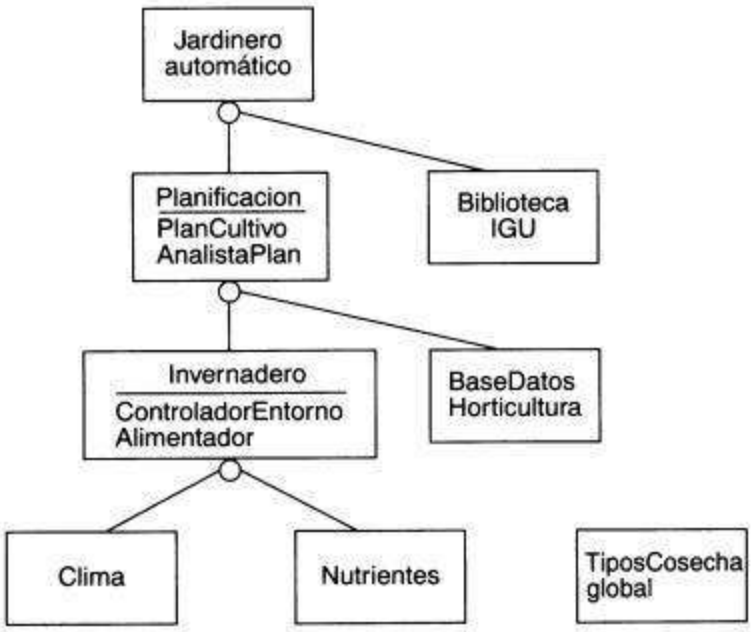
d) Componentes

Representa un elemento de implementación que se puede redistribuir



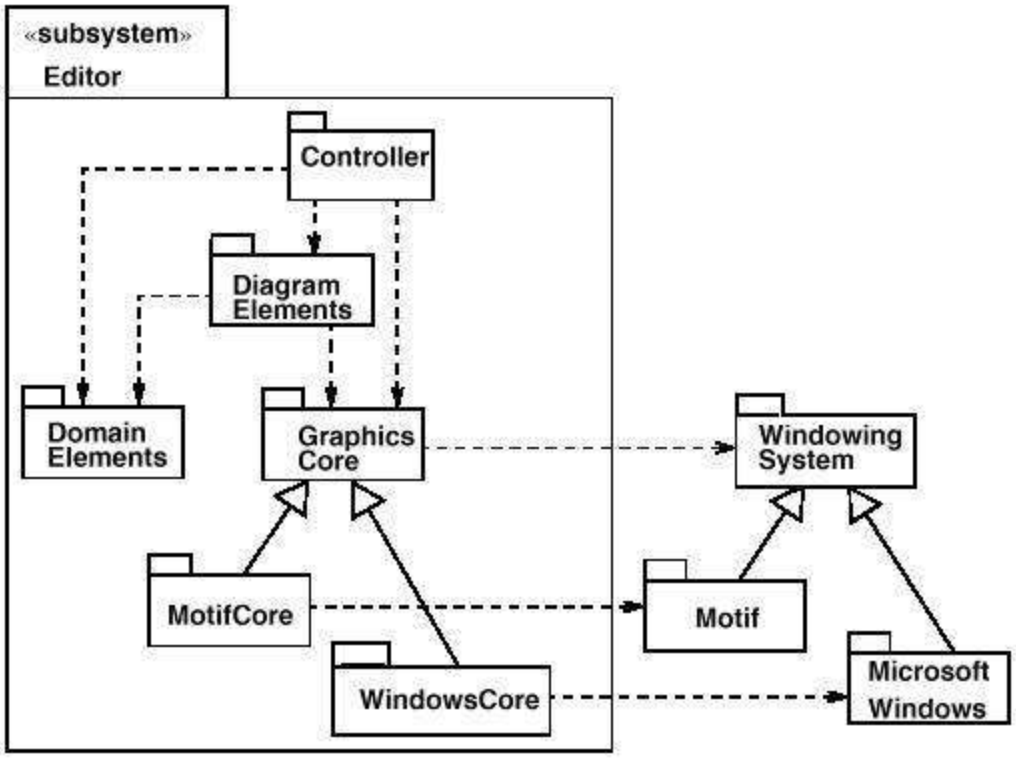
Categorías de clases
Booch

- *Una **categoría de clases** es un agregado que contiene clases y otras categorías de clases*
- Sirven para dividir el modelo lógico de un sistema
- Se da un nombre a cada categoría de clases
- No es necesario mostrar todas las clases contenidas, puede mostrarse las más representativas o ninguna
- Una categoría de clases puede utilizar otras categorías de clases. Se utiliza el mismo símbolo de la relación de uso.
- Las categorías de clases de más alto nivel representan la arquitectura de alto nivel del sistema



Paquetes en UML

• Un paquete (package) es un grupo de elementos del modelo
• Los paquetes pueden tener anidados otros paquetes, así como paquetes subordinados
• Algunos paquetes pueden ser Subsistemas o modelos

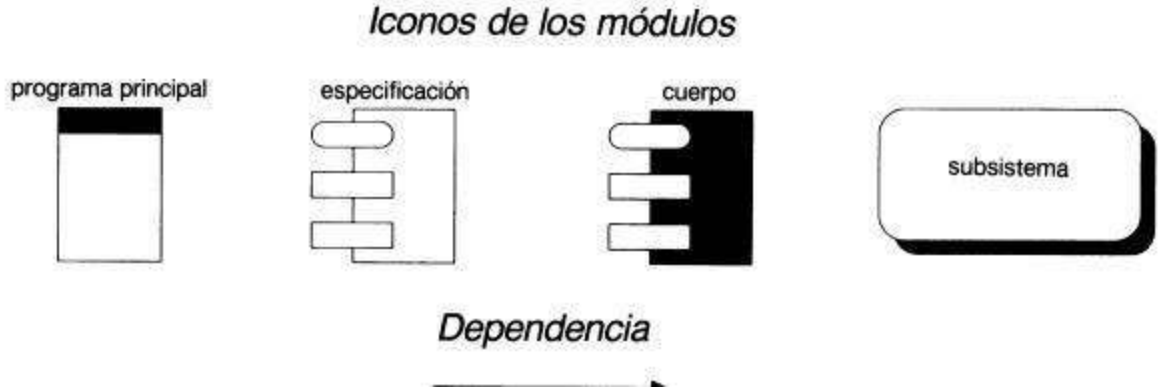


Diagramas de módulos

Muestran la asignación de clases y objetos a módulos en la vista física de un sistema

Notación Booch

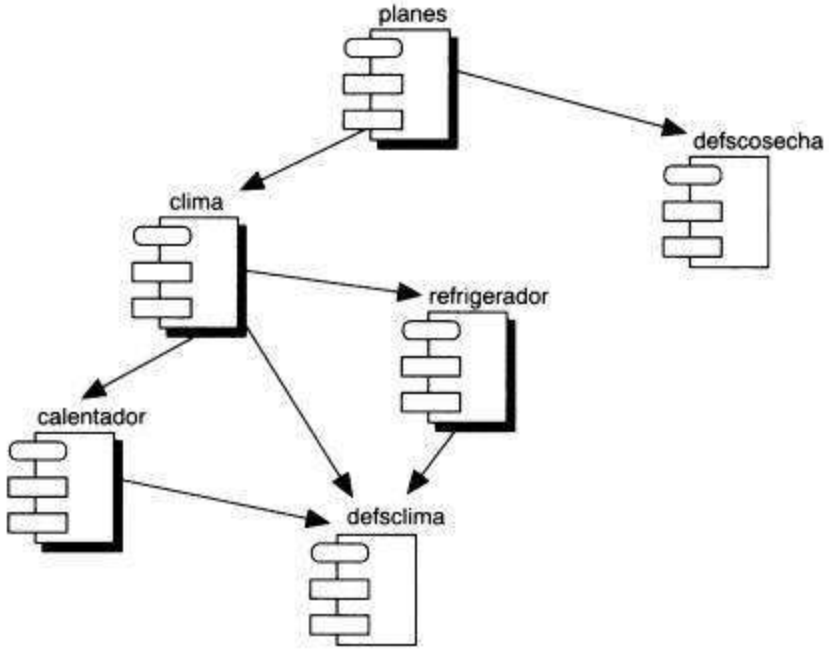
- Los primeros tres iconos denotan ficheros
  - *icono **programa principal** denota el fichero que contiene la raíz del programa*
  - *icono **especificación** denota el fichero cabecera del módulo*
  - *icono **cuerpo** denota el fichero cuerpo de cada módulo*
    - Estos dos últimos iconos pueden agruparse en un icono especificación con sombra
    - Todo nombre de fichero debe ser único
- *Las **dependencias** entre módulos se refieren a dependencias de compilación*
  - Se representan por una flecha que apunta al módulo respecto al cual existe la dependencia
- *Los **subsistemas** son agrupaciones de módulos relacionados lógicamente*



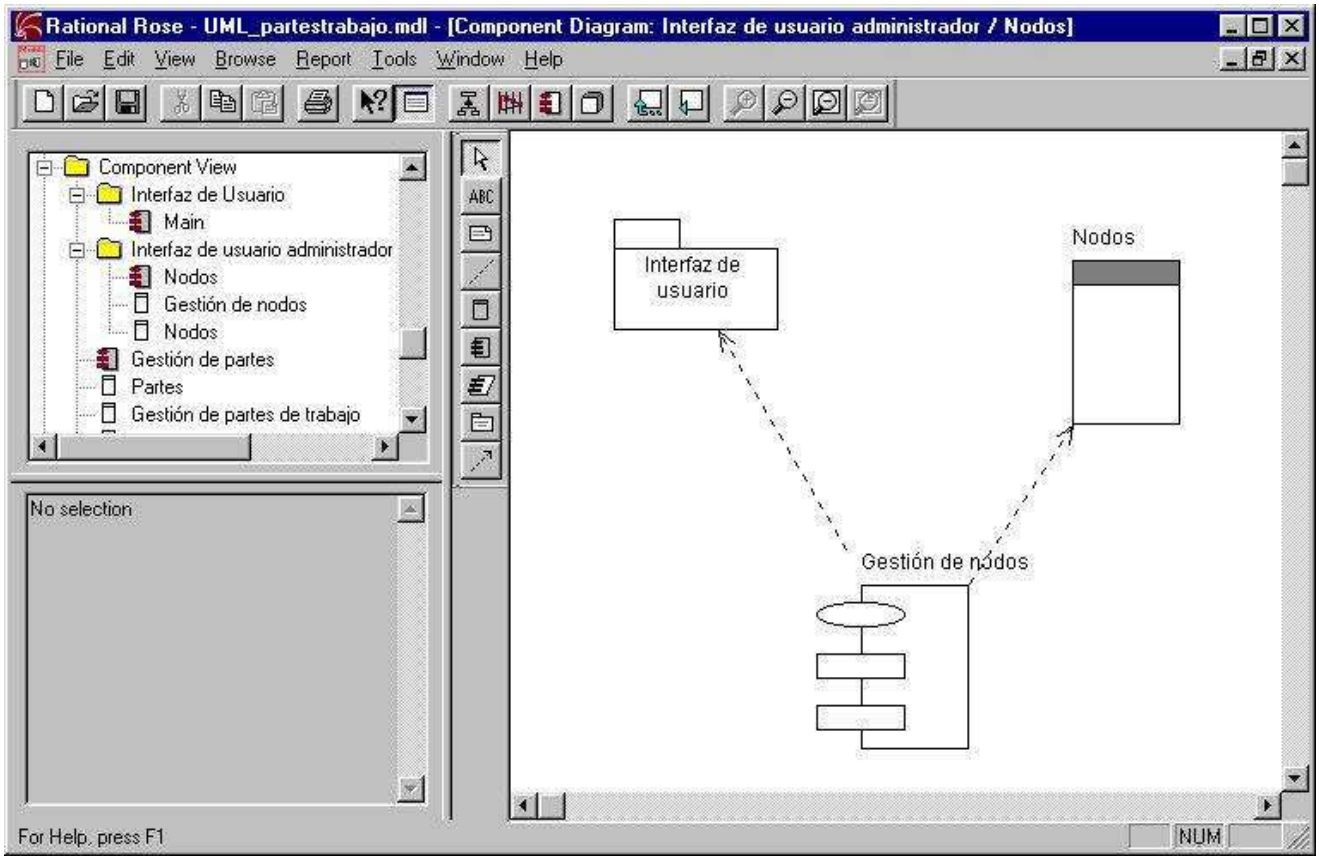
Diagramas de módulos

Ejemplo en la notación de Booch

- Cuatro módulos se muestran con sus especificaciones y cuerpos agrupados
- Dos módulos son solamente especificaciones y sirven para definir tipos y constantes comunes



Ejemplo de diagrama de componentes en Rose

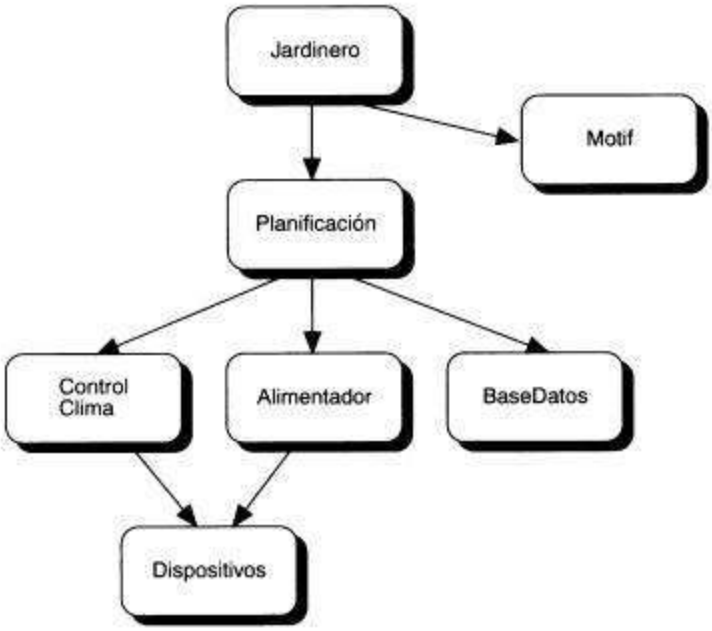


Diagramas de módulos

Subsistemas

Ejemplo en la notación de Booch

- Un subsistema es un agregado que contiene otros módulos y otros subsistemas
- Se necesita un nombre para cada subsistema
- Algunos de los módulos englobados en un subsitema se consideran públicos, que quiere decir que pueden ser utilizados por otro módulo fuera del subsistema
- Un subsistema puede tener dependencias respecto de otros susbsistemas o módulos. Se usa la flecha dirigida para indicarlo como en el caso de los módulos

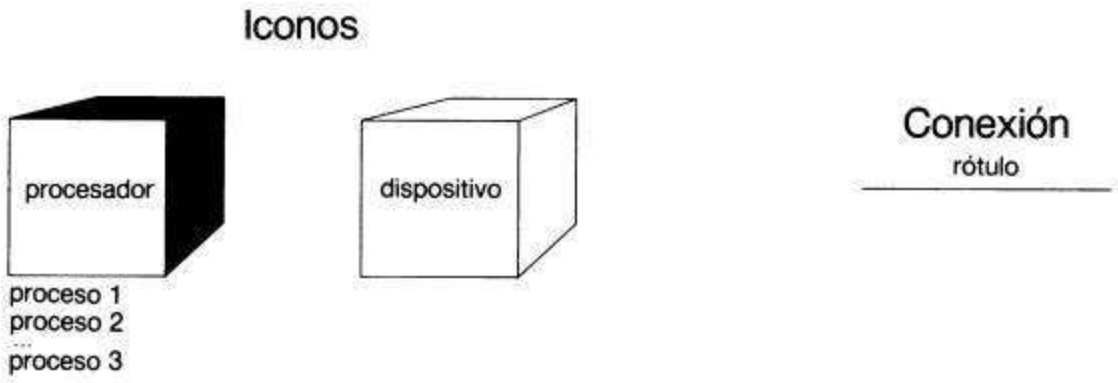


Diagramas de procesos

Muestran la asociación de procesos a procesadores en la vista física de un sistema

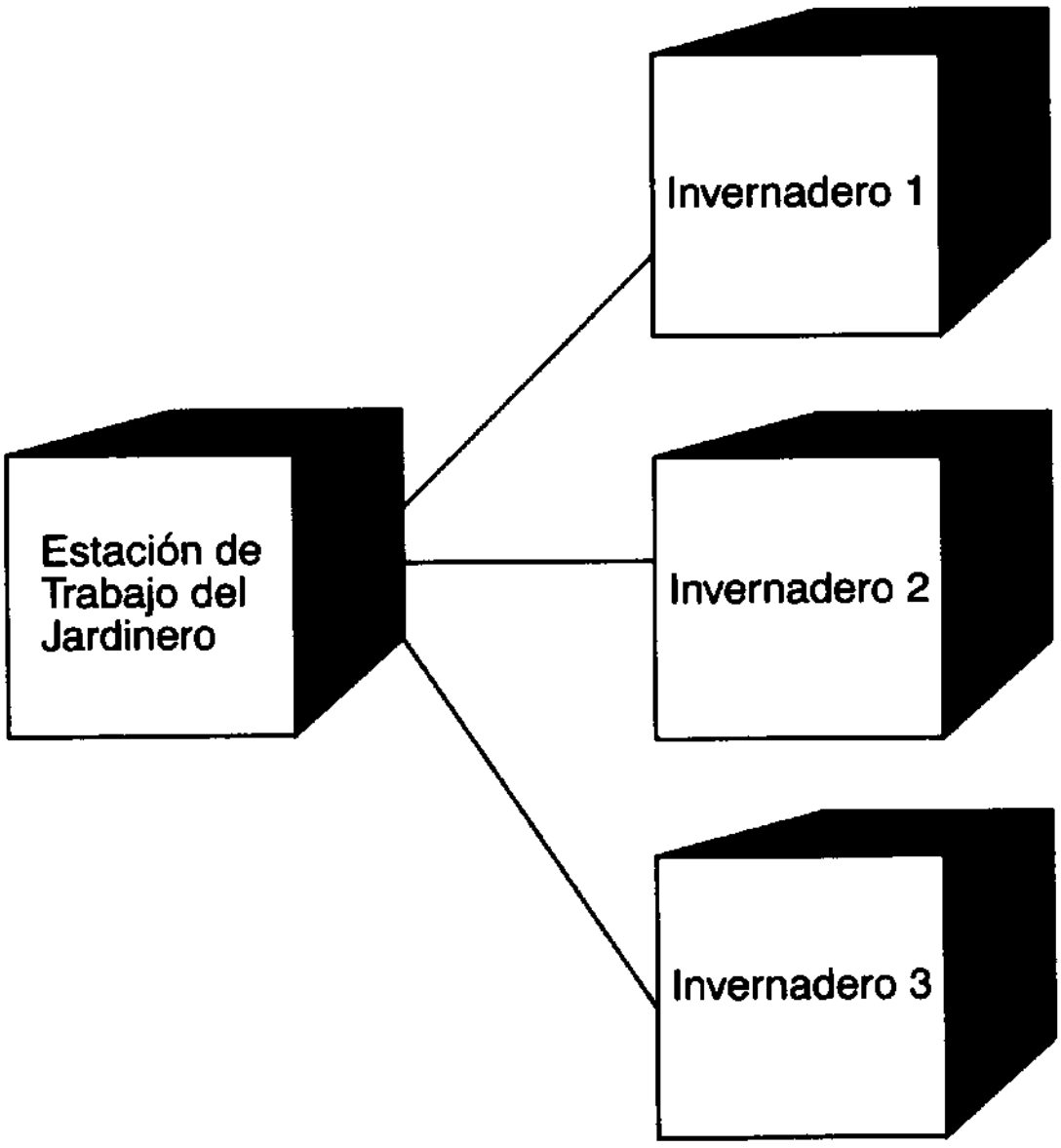
Notación Booch

- Se utilizan diagramas de procesos para mostrar la asignación de procesos a procesadores en el diseño físico de un sistema
- Los tres elementos básicos de un diagrama de procesos son
  - **Procesadores** *: Elementos hardware capaces de ejecutar programas*
  - **Dispositivos** *: Elementos hardware incapaces de ejecutar programas*
  - **Conexiones** *: Comunicación entre procesadores y dispositivos*
- Pueden definirse iconos específicos para los elementos del diagrama de procesos
- Puede haber anidamiento de procesadores y dispositivos
- Se puede adoptar alguna política de planificación de la ejecución de procesos en un procesador
- Pueden añadirse especificaciones en modo texto

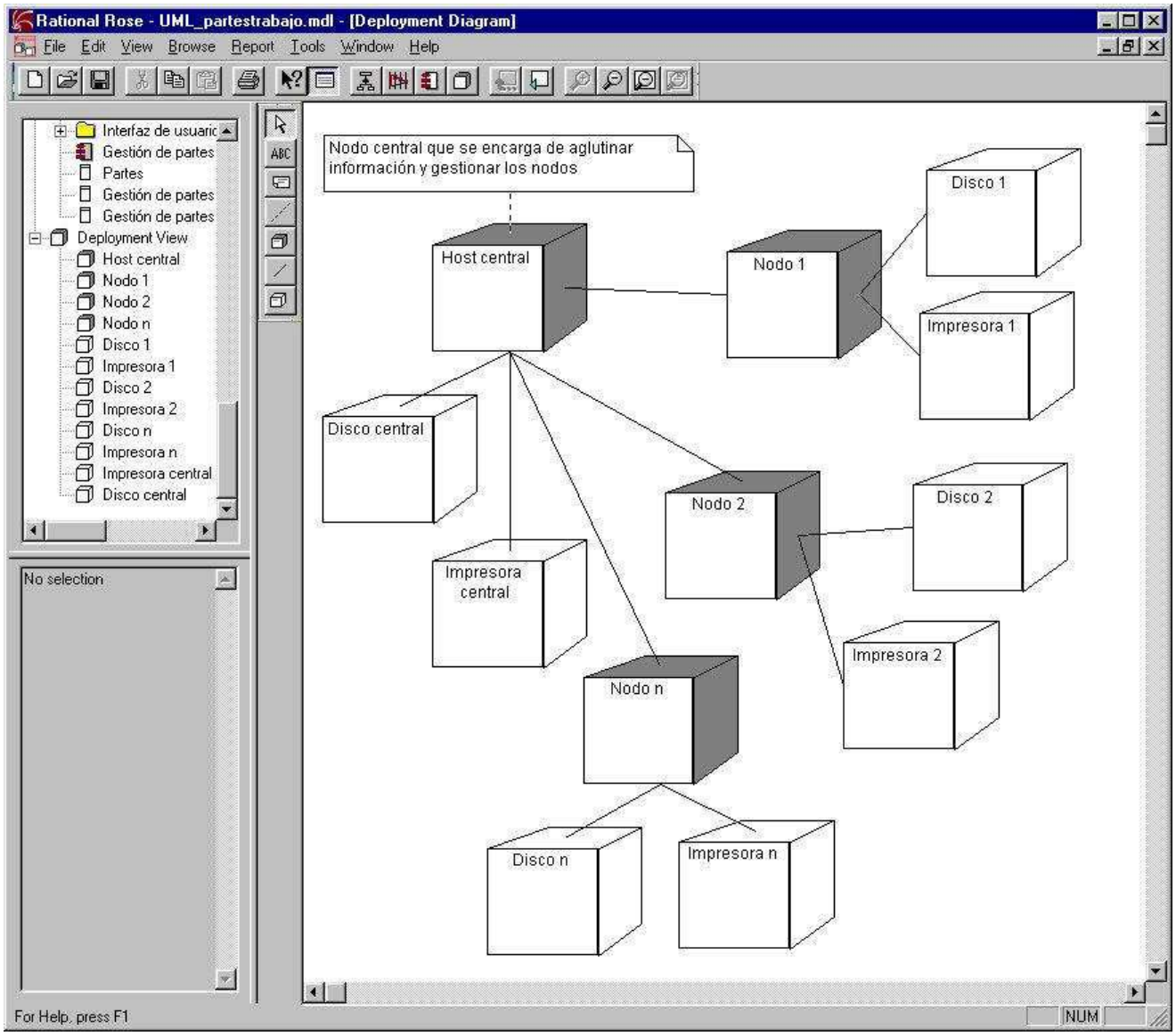


Diagramas de procesos

Ejemplo en la notación de Booch

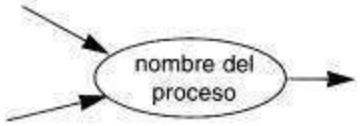


Ejemplo de Diagrama Desplegable en Rose



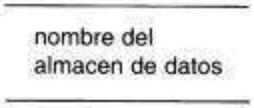
Notación del modelo funcional OMT

Proceso:

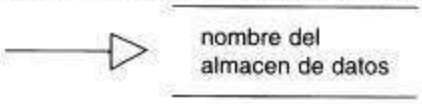


Flujo de datos entre procesos:

Almacen de Datos u Objeto Archivo:

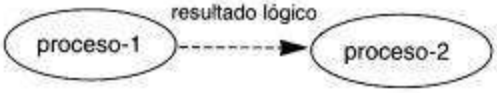


Flujo de Datos que da lugar a un almacén de datos

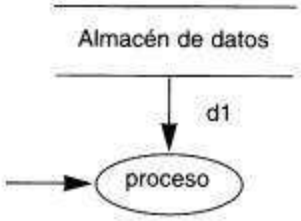


Objetos Actor (como Fuente o Sumidero de Datos)

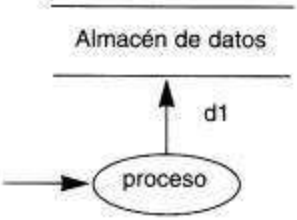
Flujo de Control:



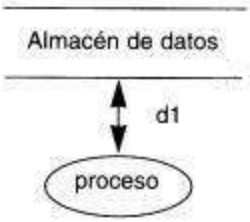
Acceso a un valor de una Almacén de Datos



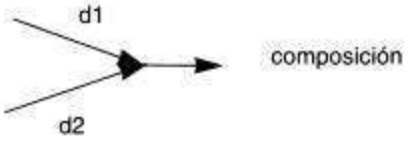
Actualización de un valor de un Amacén de Datos



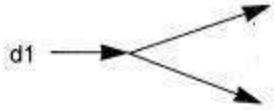
Acceso y actualización de un valor de un almacén de datos



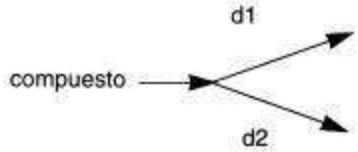
Composición de un valor de datos:



Duplicación de un valor de datos:



Descomposición de un valor de datos:



Aplicación de las notación

- Las herramientas para el soporte automático del diseño orientado a objetos
  - Las herramientas no hacen diseño
- Las herramientas se pueden aplicar a sistemas grandes y pequeños
- Notación independiente del lenguaje

Criterios de selección de herramientas de AyDOO

- Soporte completo de una o más metodologías
  - Soporte completo de cada notación
- Interfaz de usuario amigable
  - Consistente con la notación
  - Debe realizar comprobaciones mientras se está diseñando
    - Ejemplo: No debe permitir que una clase padre herede de una clase hija
- Generación de código automático
  - Normalmente generan declaraciones y algunos métodos básicos como constructores y destructores
- Ingeniería inversa a partir de código legado o código generado
  - A partir de ficheros con programas se obtiene los esquemas de la notación
  - Cambios en el código implican cambios en la notación
- Seguimiento de los requisitos
  - Reflejar en el diseño y en el código los requisitos
- Generación automática de la documentación de la aplicación
  - La documentación producida debe ser
    - Completa
    - Bien diseñada, cómoda de utilizar y de usar
    - Personalizable (se genera en formatos estándar que pueden ser manejados por herramientas de maquetación estándar)
- Comprobación de la consistencia y completud entre todas las vistas del sistema
- Ayuda en línea completa
- Buena documentación de la herramienta