

Desarrollo de Software Orientado a Objeto usando UML

Patricio Letelier Torres
letelier@dsic.upv.es

Departamento Sistemas Informáticos y Computación (DSIC)
Universidad Politécnica de Valencia (UPV) - España



★ www.dsic.upv.es/~uml

1

Contenido

- I. Introducción
 - Modelado de Software
 - UML
- II. Breve Tour por UML
- III. El Paradigma Orientado a Objeto usando UML
 - Fundamentos del Modelado OO
 - Requisitos del software
 - Interacción entre objetos
 - Clases y relaciones entre clases
 - Comportamiento de objetos
 - Componentes
 - Distribución y despliegue de componentes
 - Object Constraint Language (OCL)
- IV. Proceso de Desarrollo de SW basado en UML
- V. Conclusiones

★ www.dsic.upv.es/~uml

2

I Introducción

★ www.dsic.upv.es/~uml

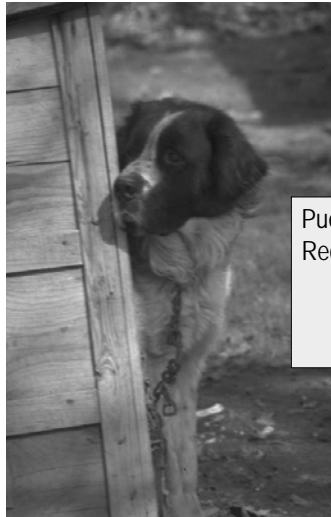
3

Introducción: Modelado de SW

★ www.dsic.upv.es/~uml

4

Construcción de una casa para "fido"



Puede hacerlo una sola persona
Requiere:
Modelado mínimo
Proceso simple
Herramientas simples

Construcción de una casa

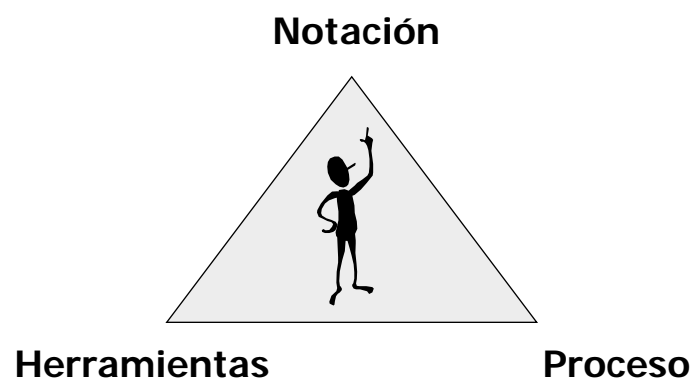


Construida eficientemente y en un tiempo razonable por un equipo
Requiere:
Modelado
Proceso bien definido
Herramientas más sofisticadas

Construcción de un rascacielos



Claves en Desarrollo de SI



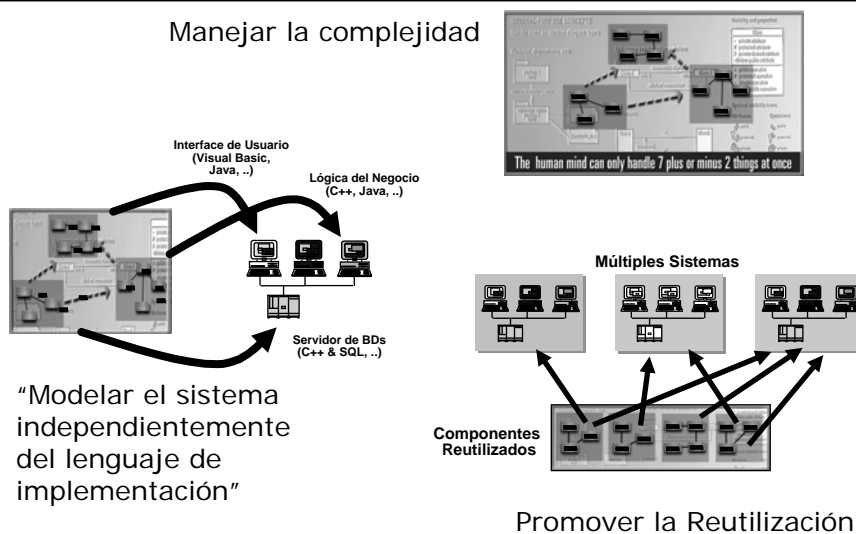
Abstracción - Modelado Visual (MV)

"El modelado captura las partes esenciales del sistema"



II. Notación (Visual) - Beneficios

Manejar la complejidad



Introducción: UML

★ www.dsic.upv.es/~uml

11

I. Introducción: UML

¿Qué es UML?

- UML = Unified Modeling Language
- Un lenguaje de propósito general para el modelado orientado a objetos. Impulsado por el Object Management Group (OMG, www.omg.org)
- Documento "OMG Unified Modeling Language Specification"
- UML combina notaciones provenientes desde:
 - Modelado Orientado a Objetos
 - Modelado de Datos
 - Modelado de Componentes
 - Modelado de Flujos de Trabajo (Workflows)

★ www.dsic.upv.es/~uml

12

Situación de Partida

- Diversos métodos y técnicas OO, con muchos aspectos en común pero utilizando distintas notaciones
- Inconvenientes para el aprendizaje, aplicación, construcción y uso de herramientas, etc.
- Pugna entre distintos enfoques (y correspondientes gurús)

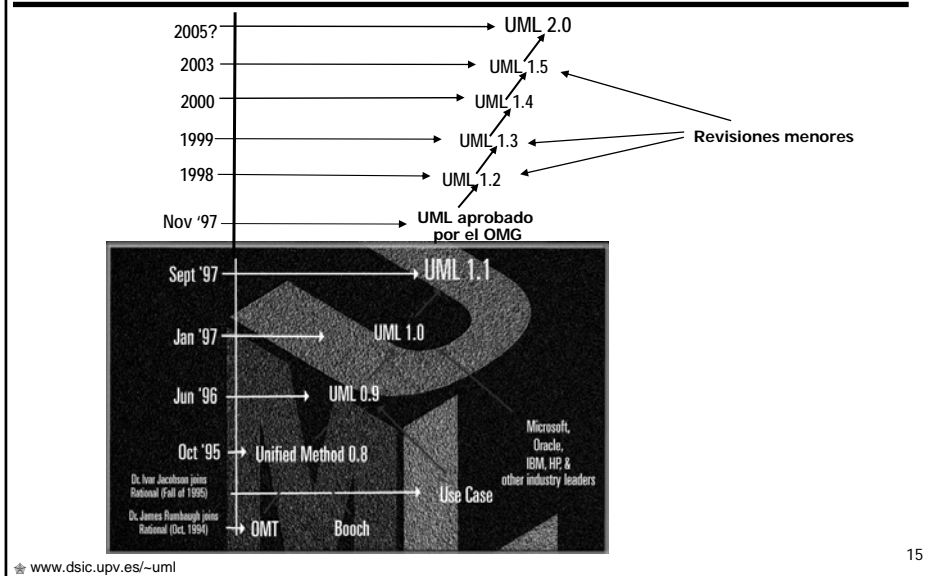


Establecer una notación estándar

Historia de UML

- Comenzó como el “Método Unificado”, con la participación de Grady Booch y Jim Rumbaugh. Se presentó en el OOPSLA’95
- El mismo año se unió Ivar Jacobson. Los “Tres Amigos” son socios en la compañía Rational Software. Herramienta CASE Rational Rose

Historia de UML



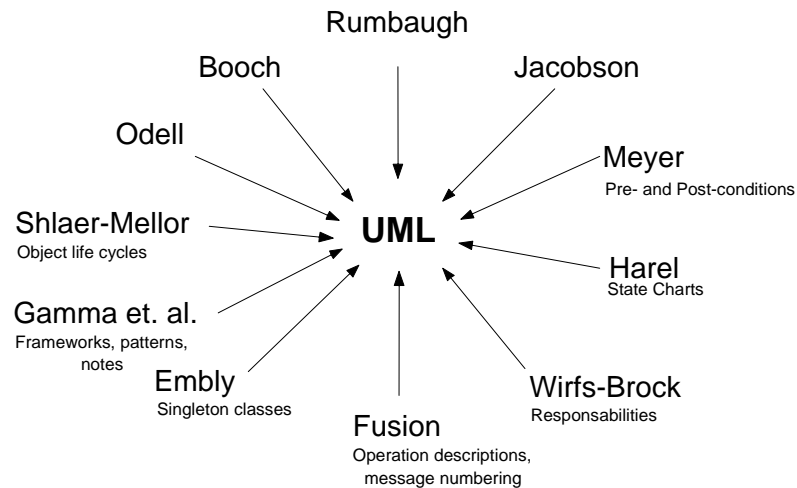
15

Participantes en UML 1.0

- Rational Software (Grady Booch, Jim Rumbaugh y Ivar Jacobson)
- Digital Equipment
- Hewlett-Packard
- i-Logix (David Harel)
- IBM
- ICON Computing (Desmond D'Souza)
- Intellicorp and James Martin & co. (James Odell)
- MCI Systemhouse
- Microsoft
- ObjecTime
- Oracle Corp.
- Platinum Technology
- Sterling Software
- Taskon
- Texas Instruments
- Unisys

16

UML "aglutina" enfoques OO



Aspectos Novedosos

- Definición semi-formal del Metamodelo de UML
- Mecanismos de Extensión en UML:
 - *Stereotypes*
 - *Constraints*
 - *Tagged Values*

Permiten adaptar los elementos de modelado,
asignándoles una semántica particular

Inconvenientes en UML

- Definición del proceso de desarrollo usando UML. UML no es una metodología
- No cubre todas las necesidades de especificación de un proyecto software. Por ejemplo, no define los documentos textuales
- Ejemplos aislados
- "Monopolio de conceptos, técnicas y métodos en torno a UML y el OMG"

Perspectivas de UML

- UML es el lenguaje de modelado orientado a objetos estándar predominante ahora y en los próximos años
- Razones:
 - Participación de metodólogos influyentes
 - Participación de importantes empresas
 - Estándar del OMG
- Evidencias:
 - Herramientas que proveen la notación UML
 - "Edición" de libros (más de 300 en www.amazon.com)
 - Congresos, cursos, "camisetas", etc.

II Breve Tour por UML

Modelos y Diagramas

- Un **modelo** captura una vista de un sistema del mundo real. Es una abstracción de dicho sistema, considerando un cierto propósito. Así, el modelo describe completamente aquellos aspectos del sistema que son relevantes al propósito del modelo, y a un apropiado nivel de detalle.
- **Diagrama**: una representación gráfica de una colección de elementos de modelado, a menudo dibujada como un grafo con vértices conectados por arcos

... Modelos y Diagramas

- Un proceso de desarrollo de software debe ofrecer un conjunto de modelos que permitan expresar el producto desde cada una de las perspectivas de interés
- El código fuente del sistema es el modelo más detallado del sistema (y además es ejecutable). Sin embargo, se requieren otros modelos ...



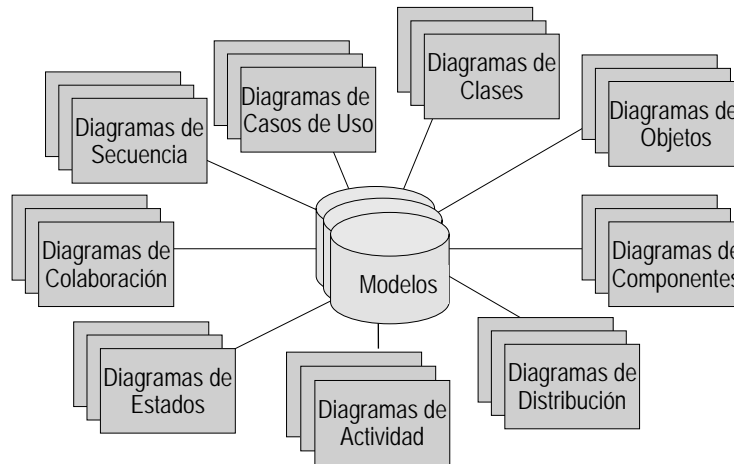
- Cada modelo es completo desde su punto de vista del sistema, sin embargo, existen relaciones de trazabilidad entre los diferentes modelos

Diagramas de UML 1.5

- Diagrama de Casos de Uso
- Diagrama de Clases
- Diagrama de Objetos
- Diagramas de Comportamiento
 - Diagrama de Estados
 - Diagrama de Actividad
- Diagramas de Interacción
 - Diagrama de Secuencia
 - Diagrama de Colaboración
- Diagramas de implementación
 - Diagrama de Componentes
 - Diagrama de Despliegue

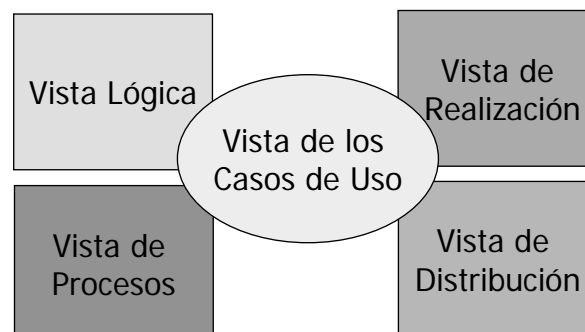
... Diagramas de UML

Los diagramas expresan gráficamente partes de un modelo



Organización de Modelos

4+1 vistas de Kruchten (1995)



Este enfoque sigue el browser de Rational Rose

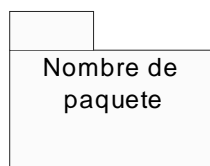
... Organización de Modelos

Propuesta de Rational Unified Process (RUP)

- M. de Casos de Uso del Negocio (Business Use-Case Model)
- M. de Objetos del Negocio (Business Object Model)
- M. de Casos de Uso (Use-Case Model)
- M. de Análisis (Analysis Model)
- M. de Diseño (Design Model)
- M. de Despliegue (Deployment Model)
- M. de Datos (Data Model)
- M. de Implementación (Implementation Model)
- M. de Pruebas (Test Model)

Paquetes en UML

- Los paquetes ofrecen un mecanismo general para la organización de los modelos/subsistemas agrupando elementos de modelado
- Se representan gráficamente como:

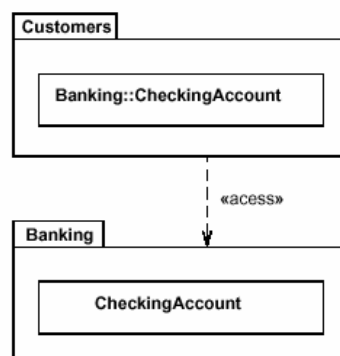


... Paquetes en UML

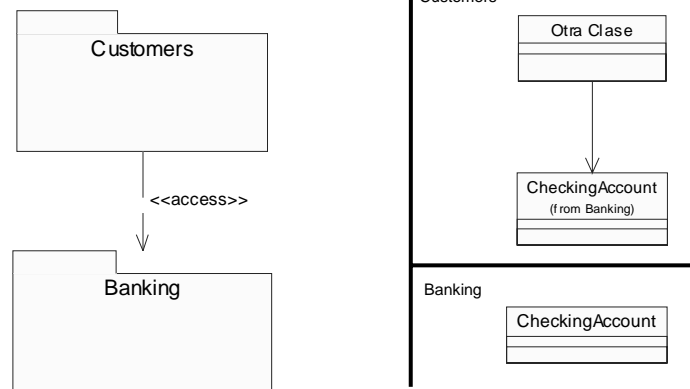
- Cada paquete corresponde a un submodelo (subsistema) del modelo (sistema)
- Un paquete puede contener otros paquetes, sin límite de anidamiento pero cada elemento pertenece a (está definido en) sólo un paquete
- Una clase de un paquete puede aparecer en otro paquete por la importación a través de una relación de dependencia entre paquetes

... Paquetes en UML

- Todos los elementos no son necesariamente visibles desde el exterior del paquete, es decir, un paquete encapsula a la vez que agrupa
- El operador "::" permite designar una clase definida en un contexto distinto del actual



...Paquetes en Rational Rose



... Paquetes en UML

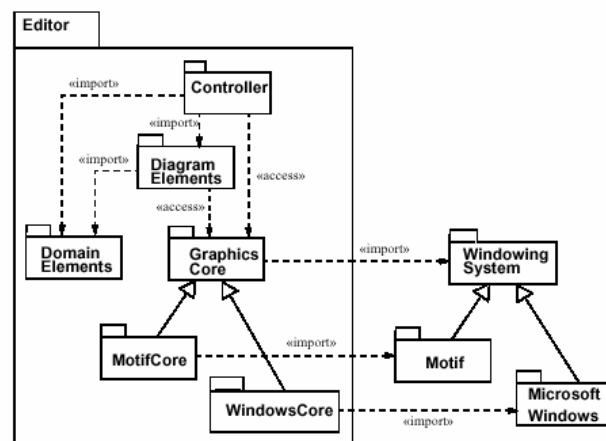


Diagrama de Casos de Uso

- Casos de Uso es una técnica para capturar información respecto de los servicios que un sistema proporciona a su entorno
- No pertenece estrictamente al enfoque orientado a objeto, es una técnica para captura y especificación de requisitos

... Ejemplos

Ejemplo:

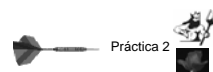
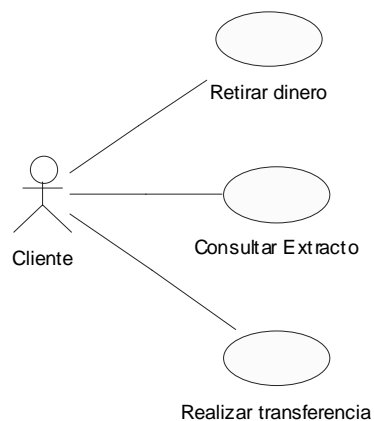


Diagrama de Secuencia

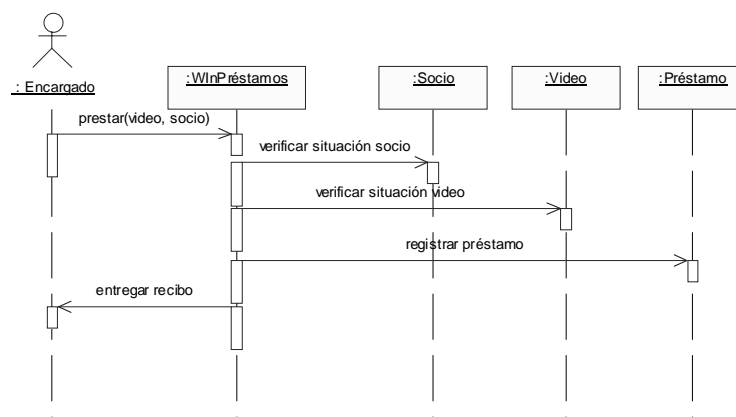


Diagrama de Colaboración

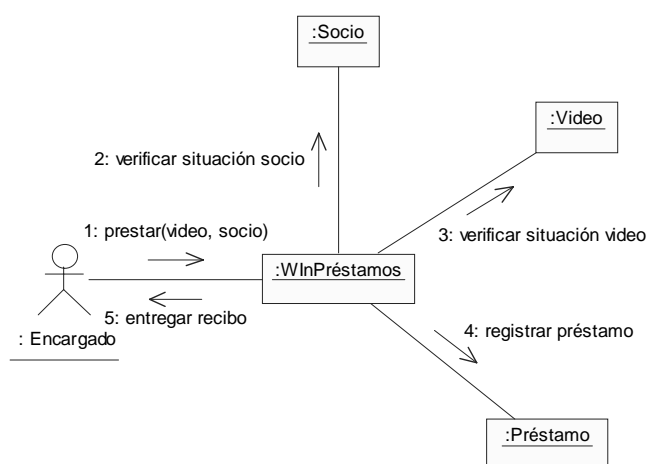
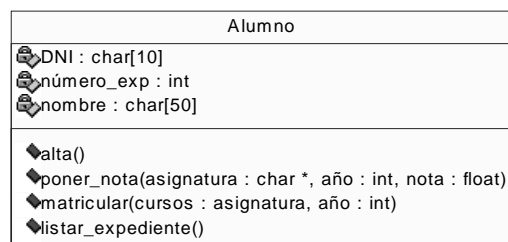


Diagrama de Clases

- El Diagrama de Clases es el diagrama principal para el análisis y diseño del sistema
- Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia
- La definición de clase incluye definiciones para atributos y operaciones
- El modelo de casos de uso debería aportar información para establecer las clases, objetos, atributos y operaciones

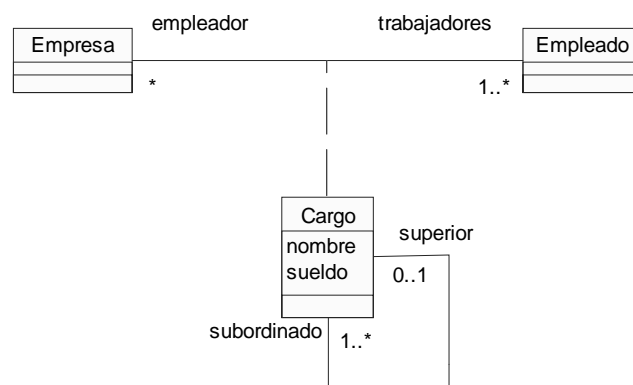
Ejemplos (Clase y Visibilidad)



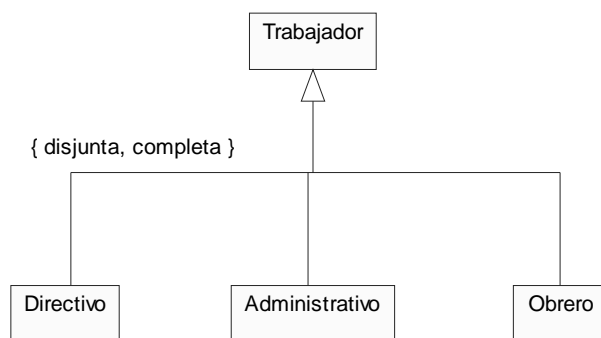
... Ejemplos (Asociación)



... Ejemplos (Clase Asociación)



... Ejemplos (Generalización)



... Ejemplos

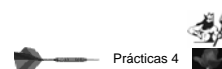
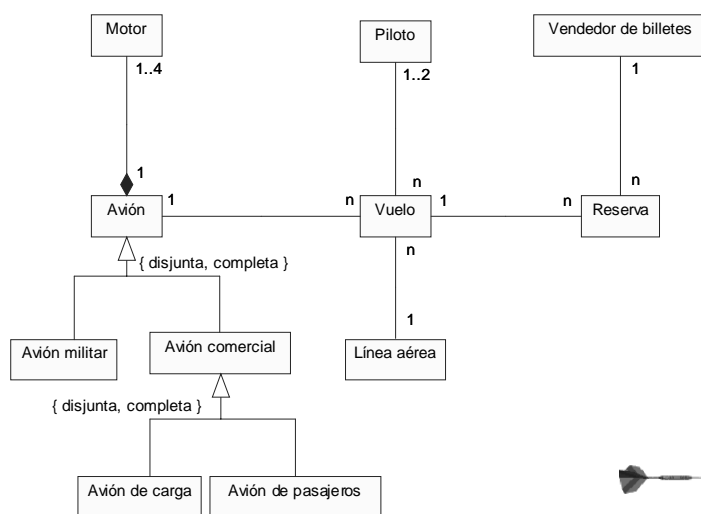


Diagrama de Estados

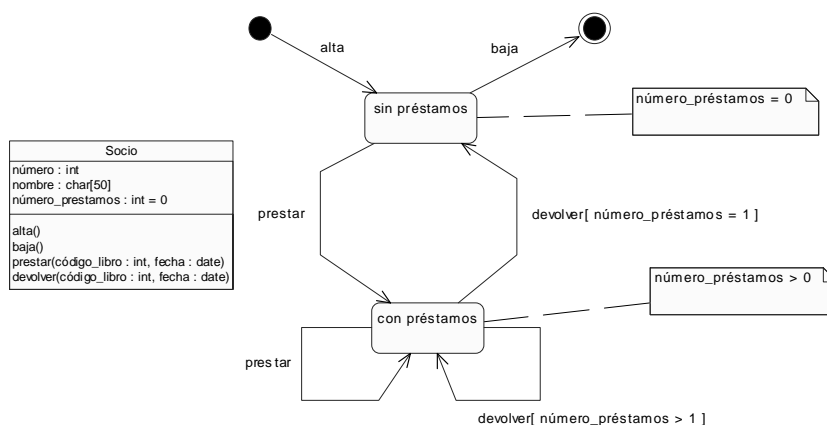


Diagrama de Actividad

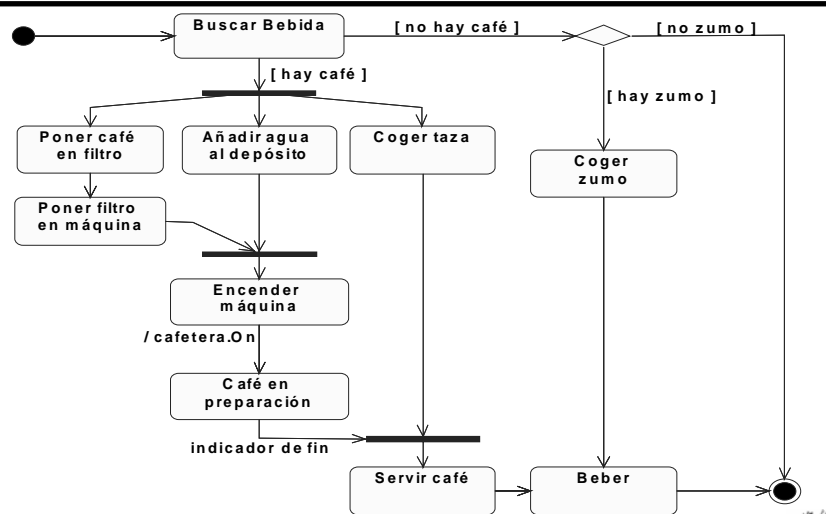


Diagrama Componentes

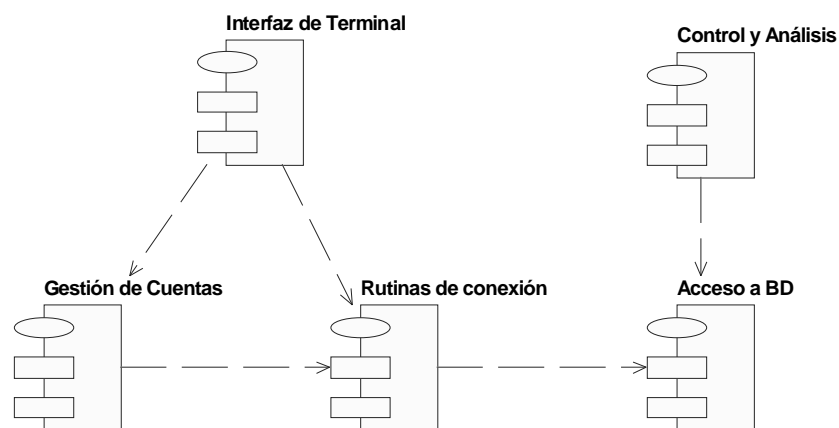


Diagrama de Despliegue

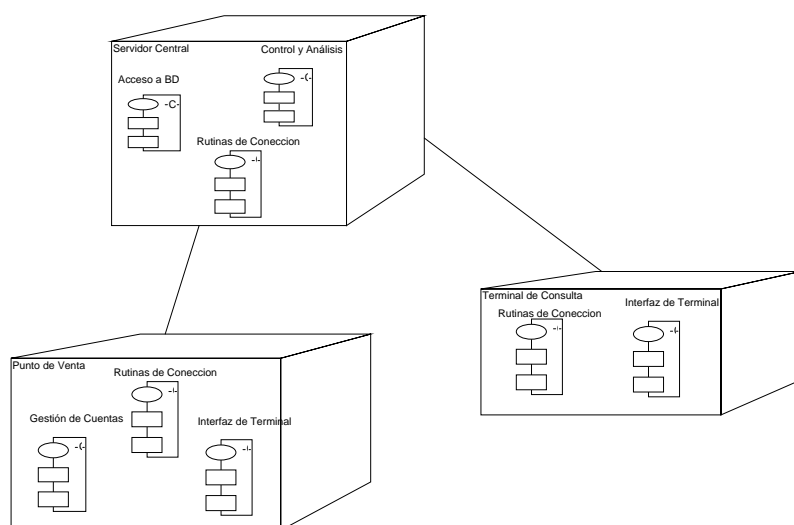
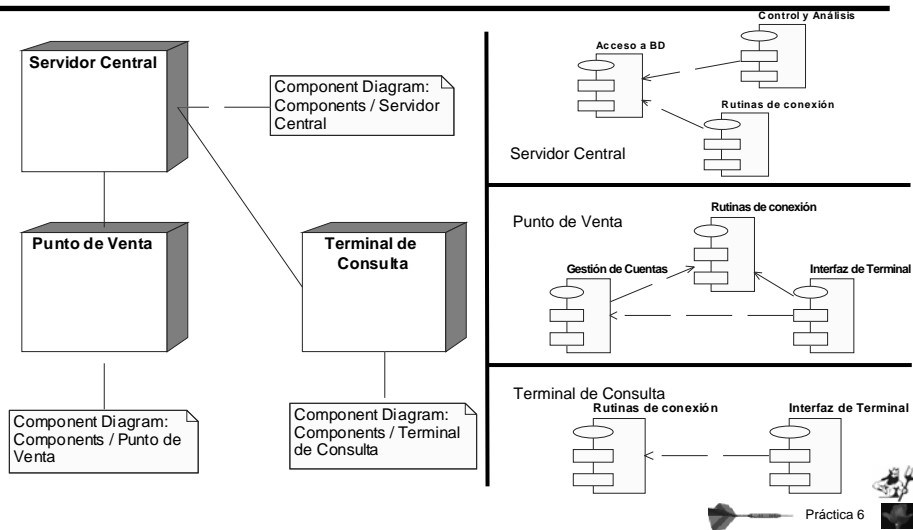


Diagrama de Despliegue en Rational



Resumen

- UML define una notación que se expresa como diagramas sirven para representar modelos/subsistemas o partes de ellos
- El 80 por ciento de la mayoría de los problemas pueden modelarse usando alrededor del 20 por ciento de UML-- Grady Booch*

III

El Paradigma Orientado a Objeto

★ www.dsic.upv.es/~uml

49

III. El Paradigma OO

¿Por qué la Orientación a Objetos?

- Proximidad de los conceptos de modelado respecto de las entidades del mundo real
 - Mejora captura y validación de requisitos
 - Acerca el “espacio del problema” y el “espacio de la solución”
- Modelado integrado de propiedades estáticas y dinámicas del ámbito del problema
 - Facilita construcción, mantenimiento y reutilización

★ www.dsic.upv.es/~uml

50

¿Por qué la Orientación a Objetos?

- Conceptos comunes de modelado durante el análisis, diseño e implementación
 - Facilita la transición entre distintas fases
 - Favorece el desarrollo iterativo del sistema
 - Disipa la barrera entre el “qué” y el “cómo”
- Sin embargo, existen problemas ...

Problemas en OO

“...Los conceptos básicos de la OO se conocen desde hace dos décadas, pero su aceptación todavía no está tan extendida como los beneficios que esta tecnología puede sugerir”

“...La mayoría de los usuarios de la OO no utilizan los conceptos de la OO de forma purista, como inicialmente se pretendía. Esta práctica ha sido promovida por muchas herramientas y lenguajes que intentan utilizar los conceptos en diversos grados”

--Wolfgang Strigel

... Problemas en OO

- Un objeto contiene datos y operaciones que operan sobre los datos, pero ...
- Podemos distinguir dos tipos de objetos degenerados:
 - Un objeto sin datos (que sería lo mismo que una biblioteca de funciones)
 - Un objeto sin "operaciones", con sólo operaciones del tipo crear, recuperar, actualizar y borrar (que se correspondería con las estructuras de datos tradicionales)
- Un sistema construido con objetos degenerados no es un sistema verdaderamente orientado a objetos

"Las aplicaciones de gestión están constituidas mayoritariamente por objetos degenerados"

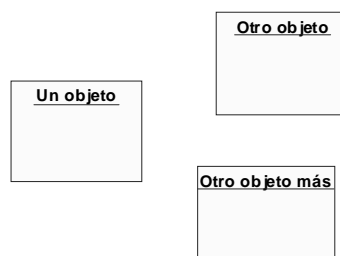
Fundamentos de Modelado OO

Objetos

- Objeto = unidad atómica que encapsula estado y comportamiento
- La encapsulación en un objeto permite una alta cohesión y un bajo acoplamiento
- Un objeto puede caracterizar una entidad física (coche) o abstracta (ecuación matemática)

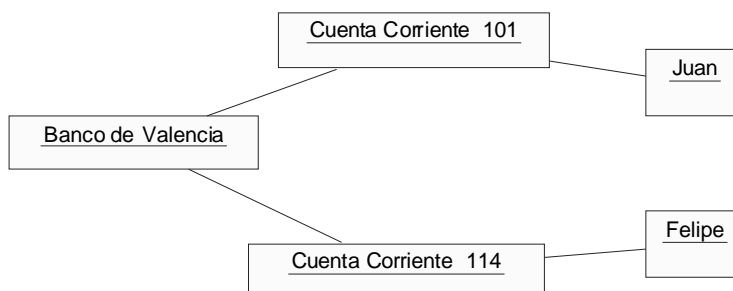
... Objetos

- En UML, un objeto se representa por un rectángulo con un nombre subrayado



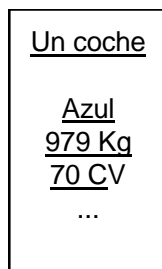
... Objetos

- Ejemplo de varios objetos relacionados:

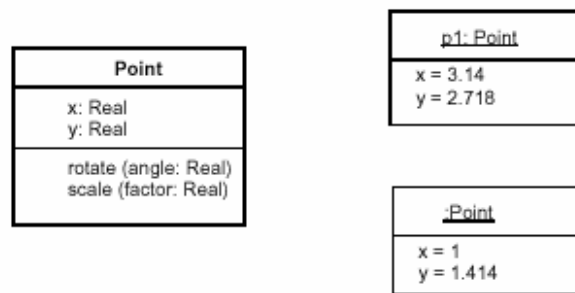


... Objetos

- Objeto = Identidad + Estado + Comportamiento
- El estado está representado por los valores de los atributos
- Un atributo toma un valor en un dominio concreto

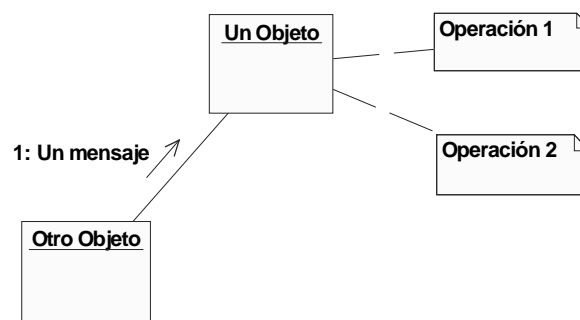


Clases y Objetos



Comportamiento

- Ejemplo de interacción:



... Comportamiento

- Los mensajes navegan por los enlaces, a priori en ambas direcciones
- Estado y comportamiento están relacionados
- Ejemplo: no es posible aterrizar un avión si no está volando. Está volando como consecuencia de haber despegado del suelo

Persistencia

- La persistencia de los objetos designa la capacidad de un objeto trascender en el espacio/tiempo
- Podremos después reconstruirlo, es decir, cogerlo de memoria secundaria para utilizarlo en la ejecución (materialización del objeto)
- Los lenguajes OO no proponen soporte adecuado para la persistencia, la cual debería ser transparente, un objeto existe desde su creación hasta que se destruya

Comunicación

- Un sistema informático puede verse como un conjunto de objetos autónomos y concurrentes que trabajan de manera coordinada en la consecución de un fin específico
- El comportamiento global se basa pues en la comunicación entre los objetos que la componen

... Comunicación

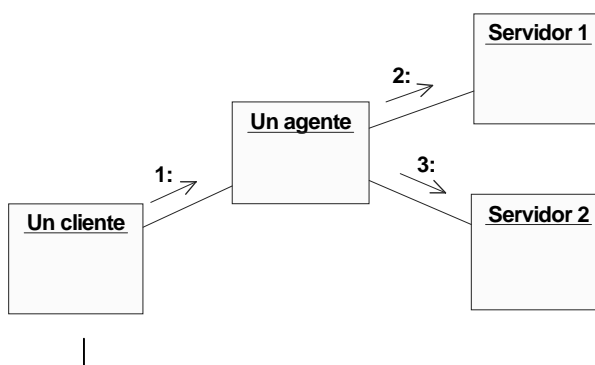
- Categorías de objetos:
 - Activos - Pasivos
 - Cliente – Servidores, Agentes
- Objeto Activo: posee un hilo de ejecución (*thread*) propio y puede iniciar una actividad
- Objeto Pasivo: no puede iniciar una actividad pero puede enviar estímulos una vez que se le solicita un servicio
- Cliente es el objeto que solicita un servicio. Servidor es el objeto que provee el servicio solicitado

... Comunicación

- Los agentes reúnen las características de clientes y servidores
- Son la base del mecanismo de *delegación*
- Introducen indirección: un cliente puede comunicarse con un servidor que no conoce directamente

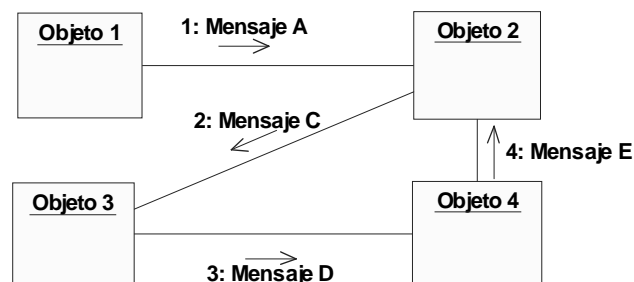
... Comunicación

- Ejemplo con objeto agente:



El Concepto de Mensaje

- La unidad de comunicación entre objetos se llama mensaje



Mensaje y Estímulo

- Un estímulo causará la invocación de una operación, la creación o destrucción de un objeto o la aparición de una señal
- Un mensaje es la especificación de un estímulo
- Tipos de flujo de control:
 - Llamada a procedimiento o flujo de control anidado \longrightarrow
 - Flujo de control plano \longrightarrow
 - Retorno de una llamada a procedimiento $- - - - \rightarrow$
 - Otras variaciones
 - Esperado (*balking*)
 - Cronometrado (*time-out*)

Requisitos del software

Casos de Uso

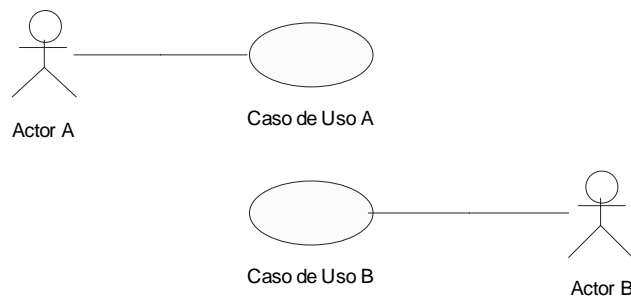
- Los Casos de Uso (Ivar Jacobson) describen bajo la forma de acciones y reacciones el comportamiento de un sistema desde el p.d.v. del usuario
- Permiten definir los límites del sistema y las relaciones entre el sistema y el entorno
- Los Casos de Uso son descripciones de la funcionalidad del sistema independientes de la implementación
- Comparación con respecto a los Diagramas de Flujo de Datos del Enfoque Estructurado

... Casos de Uso

- Los Casos de Uso cubren la carencia existente en métodos previos (OMT, Booch) en cuanto a la determinación de requisitos
- Los Casos de Uso particionan el conjunto de necesidades atendiendo a la categoría de usuarios que participan en el mismo
- El usuario debería poder entenderlos para realizar su validación

... Casos de Uso

- Ejemplo:



... Casos de Uso

Actores:

- Principales: personas que usan el sistema
 - Secundarios: personas que mantienen o administran el sistema
 - Material externo: dispositivos materiales imprescindibles que forman parte del ámbito de la aplicación y deben ser utilizados
 - Otros sistemas: sistemas con los que el sistema interactúa
- La misma persona física puede interpretar varios papeles como actores distintos
 - El nombre del actor describe el papel desempeñado

... Casos de Uso

- Los Casos de Uso se determinan observando y precisando, actor por actor, las secuencias de interacción, los escenarios, desde el punto de vista del usuario
- Un escenario es una instancia de un caso de uso
- Los casos de uso intervienen durante todo el ciclo de vida. El proceso de desarrollo estará dirigido por los casos de uso

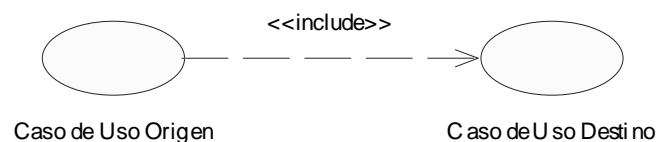
Casos de Uso: Relaciones

- UML define cuatro tipos de relación en los Diagramas de Casos de Uso:
 - Comunicación**



... Casos de Uso: Relaciones

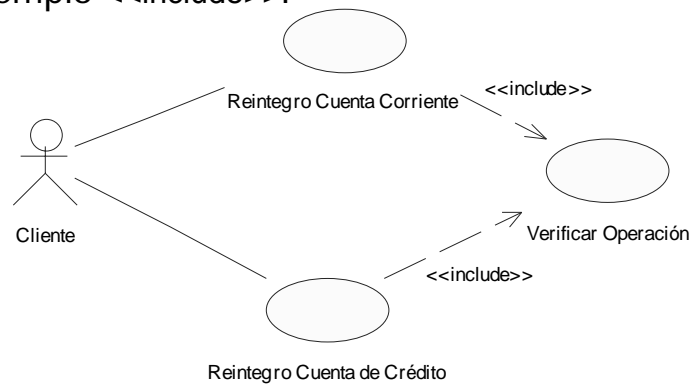
- Inclusión** : una instancia del Caso de Uso origen incluye también el comportamiento descrito por el Caso de Uso destino



<<include>> reemplazó al denominado <<uses>>

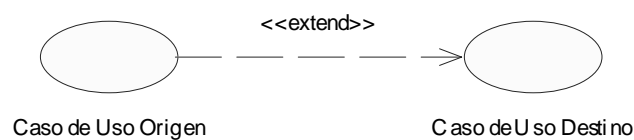
... Casos de Uso: Relaciones

▪ Ejemplo <<include>>:



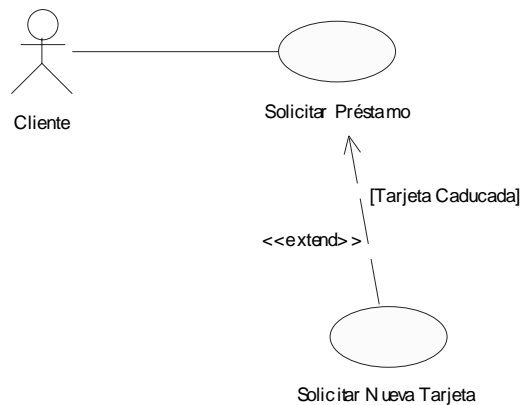
... Casos de Uso: Relaciones

- **Extensión** : el Caso de Uso origen extiende el comportamiento del Caso de Uso destino



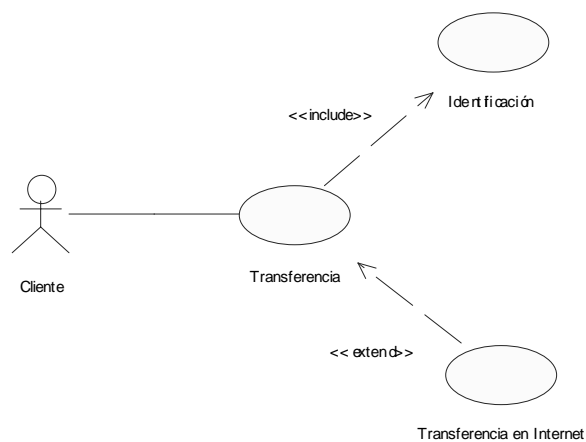
... Casos de Uso: Relaciones

- Ejemplo <<extend>>:



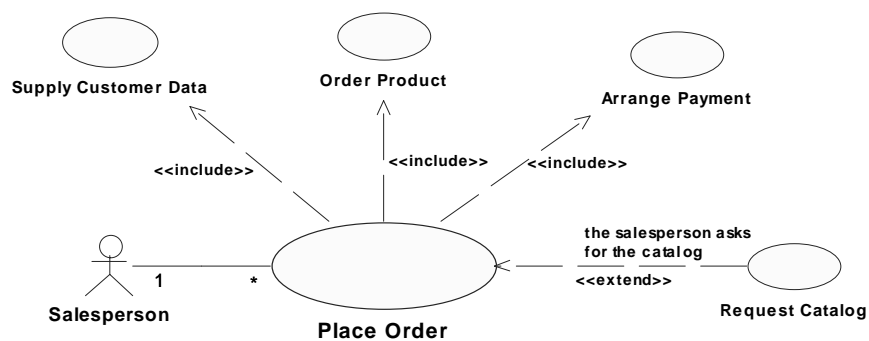
... Casos de Uso: Relaciones

- Ejemplo <<include>> y <<extend>>:



... Casos de Uso: Relaciones

- Otro ejemplo <<include>> y <<extend>>:



... Casos de Uso: Relaciones

- **Herencia** : el Caso de Uso origen hereda la especificación del Caso de Uso destino y posiblemente la modifica y/o amplía



Casos de Uso: Construcción

- Un caso de uso debe ser simple, inteligible, claro y conciso
- Generalmente hay pocos actores asociados a cada Caso de Uso
- Preguntas clave:
 - ¿cuáles son las tareas del actor?
 - ¿qué información crea, guarda, modifica, destruye o lee el actor?
 - ¿debe el actor notificar al sistema los cambios externos?
 - ¿debe el sistema informar al actor de los cambios internos?

... Casos de Uso: Construcción

- La descripción del Caso de Uso comprende:
 - el inicio: cuándo y qué actor lo produce?
 - el fin: cuándo se produce y qué valor devuelve?
 - la interacción actor-caso de uso: qué mensajes intercambian ambos?
 - objetivo del caso de uso: ¿qué lleva a cabo o intenta?
 - cronología y origen de las interacciones
 - repeticiones de comportamiento: ¿qué operaciones son iteradas?
 - situaciones opcionales: ¿qué ejecuciones alternativas se presentan en el caso de uso?



III. El Paradigma OO: Requisitos		
Identificador	CU-<id-requisito>	
Nombre	<nombre del requisito funcional>	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso (concreto cuando <evento de activación> , abstracto durante la realización de los casos de uso <lista de casos de uso>)	
Precondición	<precondición del caso de uso>	
Secuencia Normal	Paso	Acción
	1	{El <actor> , El sistema} <acción realizada por el actor o sistema>, se realiza el caso de uso < caso de uso CU-x>
	2	Si <condición>, {el <actor> , el sistema} <acción realizada por el actor o sistema>, se realiza el caso de uso < caso de uso CU-x>

Postcondición	<postcondición del caso de uso>	
Excepciones	Paso	Acción
	1	Si <condición de excepción>, {el <actor> , el sistema} <acción realizada por el actor o sistema>, se realiza el caso de uso < caso de uso CU-x>, a continuación este caso de uso {continua, aborta}

Rendimiento	Paso	Cota de tiempo
	1	n segundos

Frecuencia esperada	<nº de veces> veces / <unidad de tiempo>	
Importancia	{sin importancia, importante, vital}	
Urgencia	{puede esperar, hay presión, inmediatamente}	
Comentarios	<comentarios adicionales>	

III. El Paradigma OO: Requisitos	
<h2>Comentarios</h2> <ul style="list-style-type: none"> En métodos OO que carecen de una técnica de captura de requisitos se comienza inmediatamente con la construcción del modelo de análisis/diseño <i>Los Casos de Uso son una idea maravillosa que ha sido generalmente complicada. El verdadero truco para los Casos de Uso es mantenerlos simples. Recordad, mañana van a cambiar.</i> Rober C. Martin Los requisitos NO funcionales también son importantes. Desempeño, cumplimiento de estándares o leyes, atributos de calidad (confiabilidad, disponibilidad, seguridad, mantenibilidad, portabilidad), etc. 	
<p>www.dsic.upv.es/~uml</p>	86

Interacción entre objetos

Interacción

- Los objetos interactúan para realizar colectivamente los servicios ofrecidos por las aplicaciones. Los diagramas de interacción muestran cómo se comunican los objetos en una interacción
- Existen dos tipos de diagramas de interacción: el Diagrama de Colaboración y el Diagrama de Secuencia

Mensajes

Sintaxis para mensajes:

predecesor / guarda secuencia: retorno := msg(args)

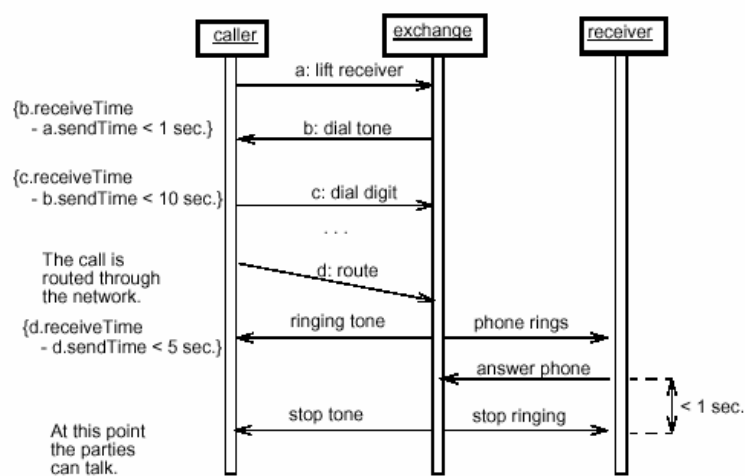
Diagramas de interacción

- El Diagrama de Secuencia es más adecuados para observar la perspectiva cronológica de las interacciones
- El Diagrama de Colaboración ofrece una mejor visión espacial mostrando los enlaces de comunicación entre objetos
- El D. de Colaboración puede obtenerse automáticamente a partir del correspondiente D. de Secuencia (o viceversa)

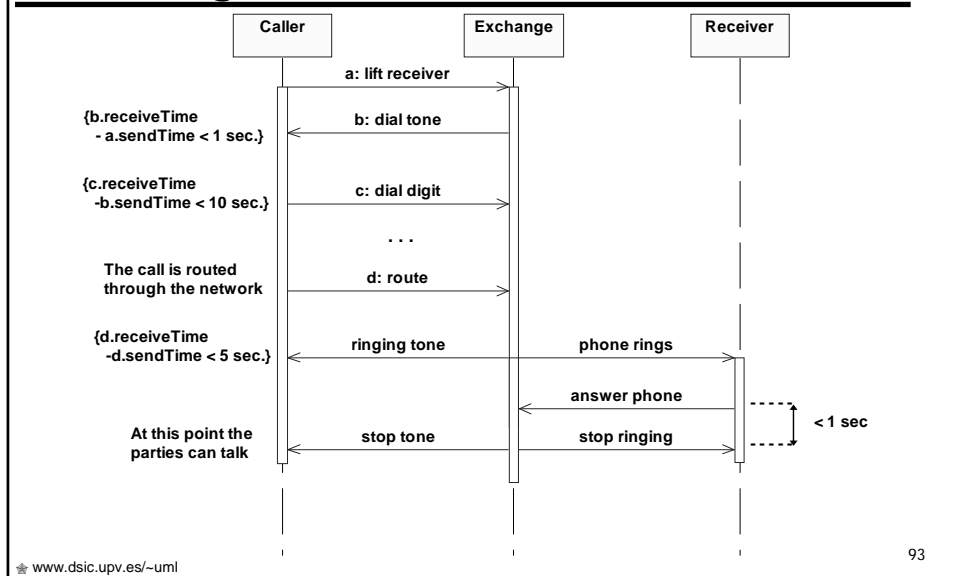
Diagrama de Secuencia

- Muestra la secuencia de mensajes entre objetos durante un escenario concreto
- Cada objeto viene dado por una barra vertical
- El tiempo transcurre de arriba abajo
- Cuando existe demora entre el envío y la atención se puede indicar usando una línea oblicua

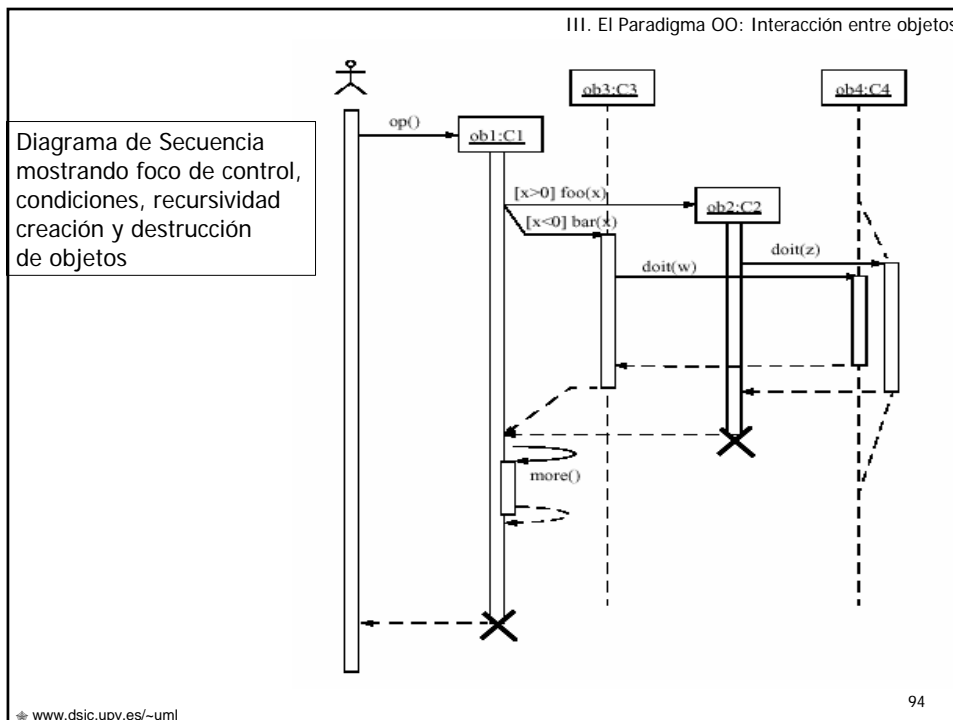
... Diagrama de Secuencia



... Diagrama de Secuencia



93



94

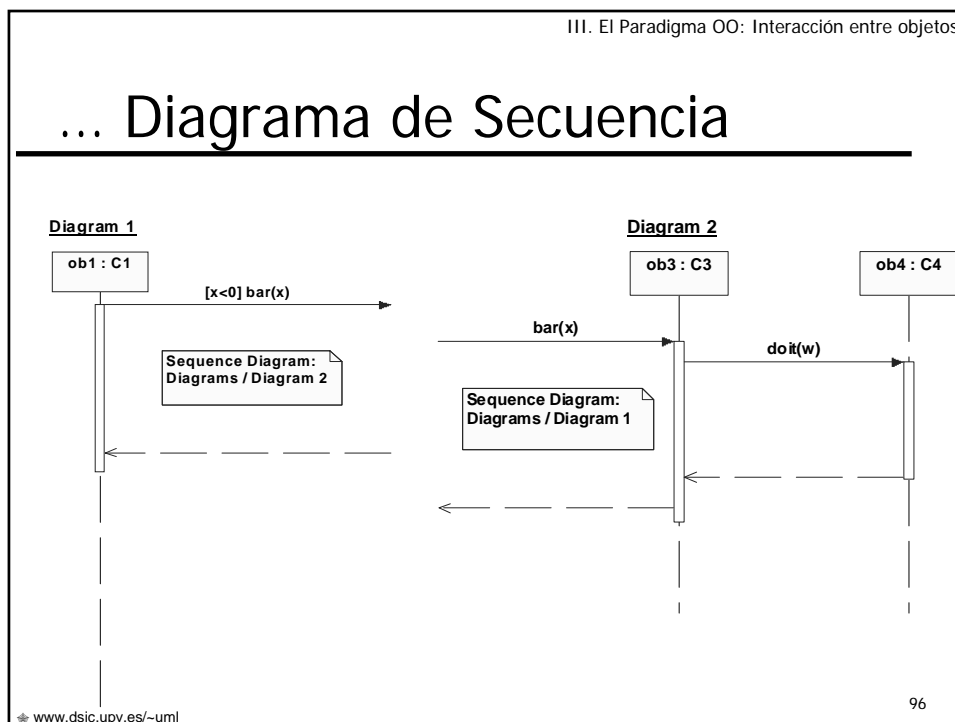
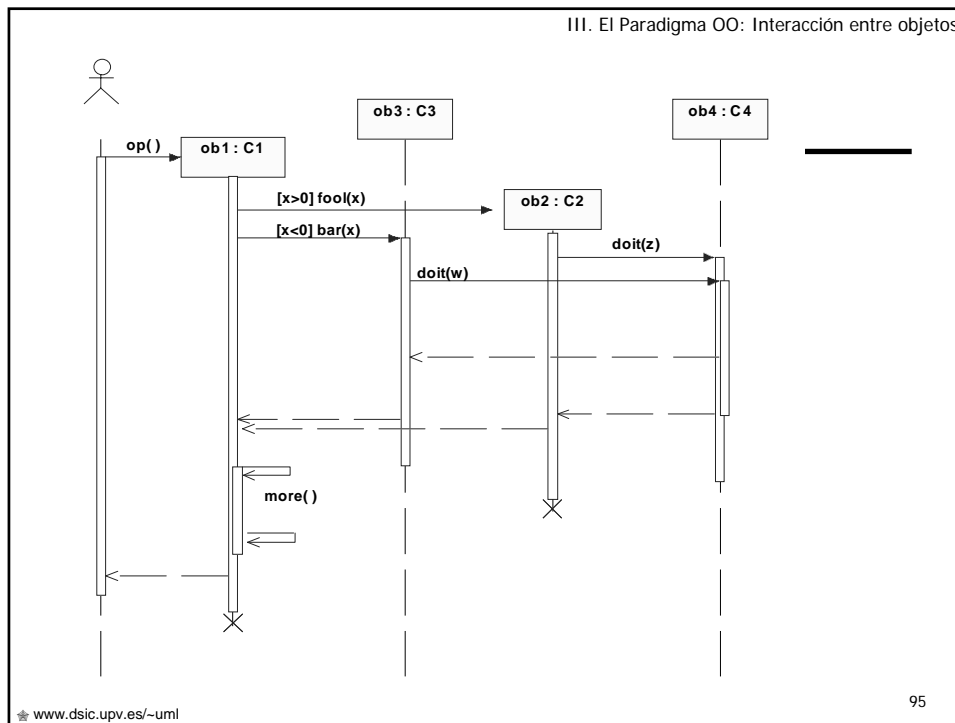
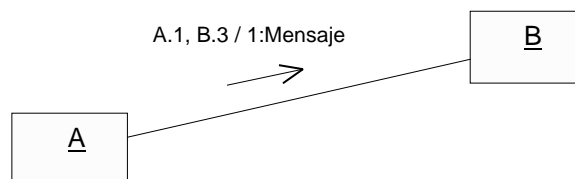


Diagrama de Colaboración

- Son útiles en la fase exploratoria para identificar objetos
- La distribución de los objetos en el diagrama permite observar adecuadamente la interacción de un objeto con respecto de los demás
- La estructura estática viene dada por los enlaces; la dinámica por el envío de mensajes por los enlaces

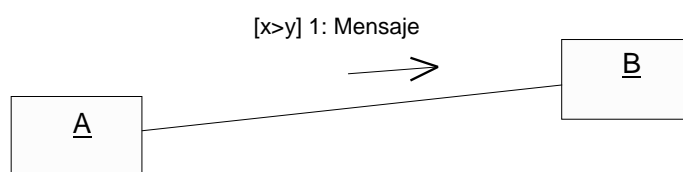
Mensajes

- Un mensaje desencadena una acción en el objeto destinatario
- Un mensaje se envía si han sido enviados los mensajes de una lista (sincronización):



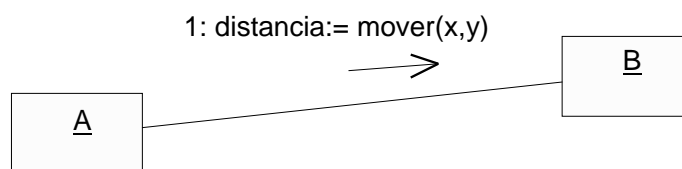
... Mensajes

- Un mensaje se envía de manera condicionada:



... Mensajes

- Un mensaje que devuelve un resultado:



Clases y relaciones entre clases

★ www.dsic.upv.es/~uml

101

III. El Paradigma OO: Clases y relaciones entre clases

Clasificación

- El mundo real puede ser visto desde abstracciones diferentes (subjetividad)
- Mecanismos de abstracción:
 - Clasificación / Instanciación
 - Composición / Descomposición
 - Agrupación / Individualización
 - Especialización / Generalización
- La clasificación es uno de los mecanismos de abstracción más utilizados

★ www.dsic.upv.es/~uml

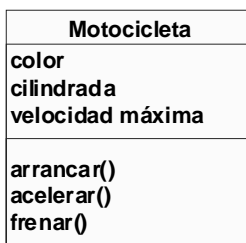
102

Clases

- La clase define el ámbito de definición de un conjunto de objetos
- Cada objeto pertenece a una clase
- Los objetos se crean por instanciación de las clases

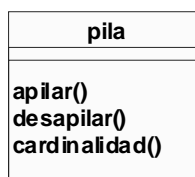
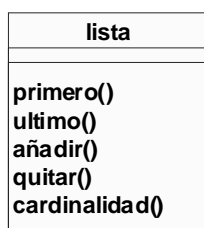
Clases: Notación Gráfica

- Cada clase se representa en un rectángulo con tres compartimentos:
 - nombre de la clase
 - atributos de la clase
 - operaciones de la clase



Clases: Notación Gráfica

- Otros ejemplos:



Clases: Encapsulación







- La encapsulación presenta dos ventajas básicas:
 - Se protegen los datos de accesos indebidos
 - El acoplamiento entre las clases se disminuye
 - Favorece la modularidad y el mantenimiento
- Los atributos de una clase no deberían ser manipulables directamente por el resto de objetos

... Clases: Encapsulación

- Los niveles de encapsulación están heredados de los niveles de C++:
 - **(-) Privado** : es el más fuerte. Esta parte es totalmente invisible (excepto para clases *friends* en terminología C++)
 - **(#) Los atributos/operaciones protegidos** están visibles para las clases *friends* y para las clases derivadas de la original
 - **(+) Los atributos/operaciones públicos** son visibles a otras clases (cuando se trata de atributos se está transgrediendo el principio de encapsulación)

... Clases: Encapsulación

- Ejemplo:

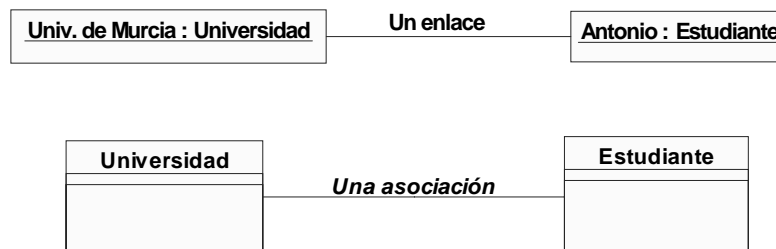
Reglas de visibilidad	
	Atributo público : Integer
	Atributo protegido : Integer
	Atributo privado : Integer
<hr/>	
	"Operación pública"()
	"Operación protegida"()
	"Operación privada"()

Relaciones entre Clases

- Los enlaces entre de objetos pueden representarse entre las respectivas clases
- Formas de relación entre clases:
 - Asociación y Agregación (vista como un caso particular de asociación)
 - Generalización/Especialización
- Las relaciones de Agregación y Generalización forman jerarquías de clases

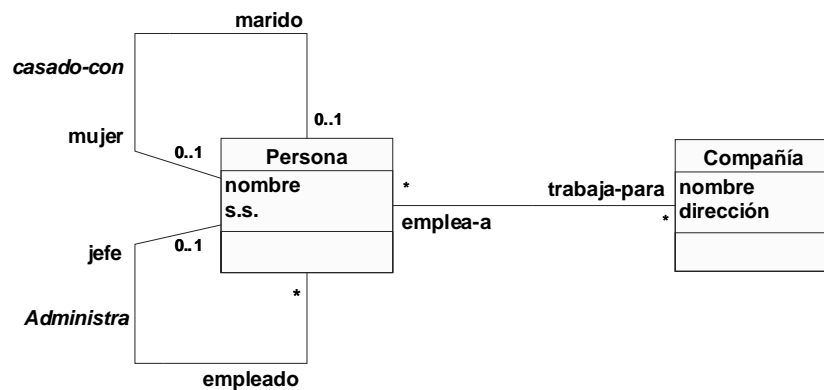
Asociación

- La asociación expresa una conexión bidireccional entre objetos
- Una asociación es una abstracción de la relación existente en los enlaces entre los objetos



... Asociación

■ Ejemplo:



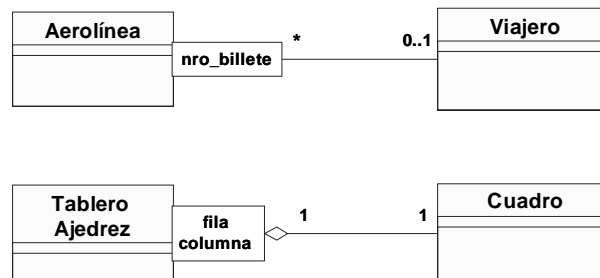
... Asociación

■ Especificación de multiplicidad (mínima...máxima)

1	Uno y sólo uno
0..1	Cero o uno
M..N	Desde M hasta N (enteros naturales)
*	Cero o muchos
0..*	Cero o muchos
1..*	Uno o muchos (al menos uno)

■ La multiplicidad mínima ≥ 1 establece una restricción de existencia

Asociación Cualificada

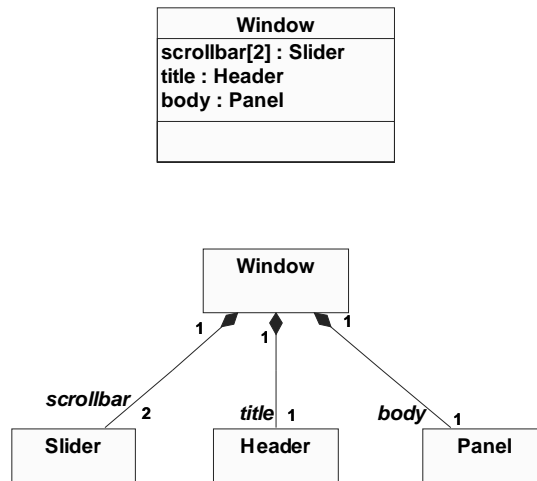


Reduce la multiplicidad del rol opuesto al considerar el valor del cualificador

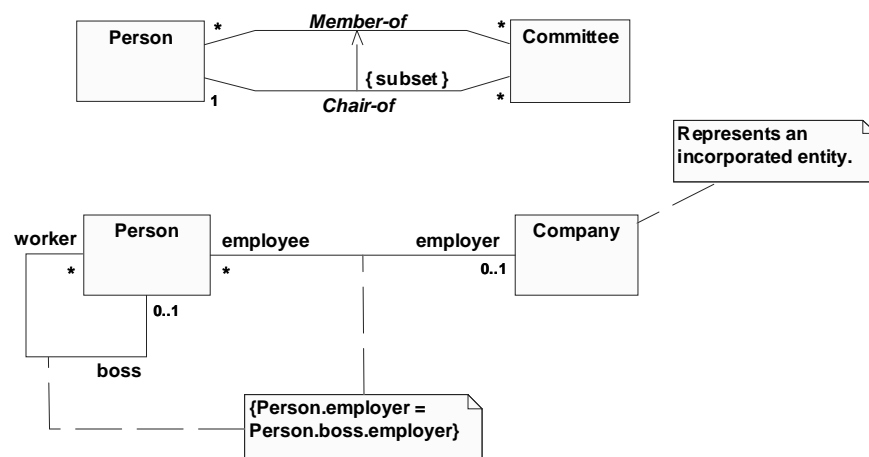
Agregación

- La agregación representa una relación *parte_de* entre objetos
- En UML se proporciona una escasa caracterización de la agregación
- Puede ser caracterizada con precisión determinando las relaciones de comportamiento y estructura que existen entre el objeto agregado y cada uno de sus objetos componentes

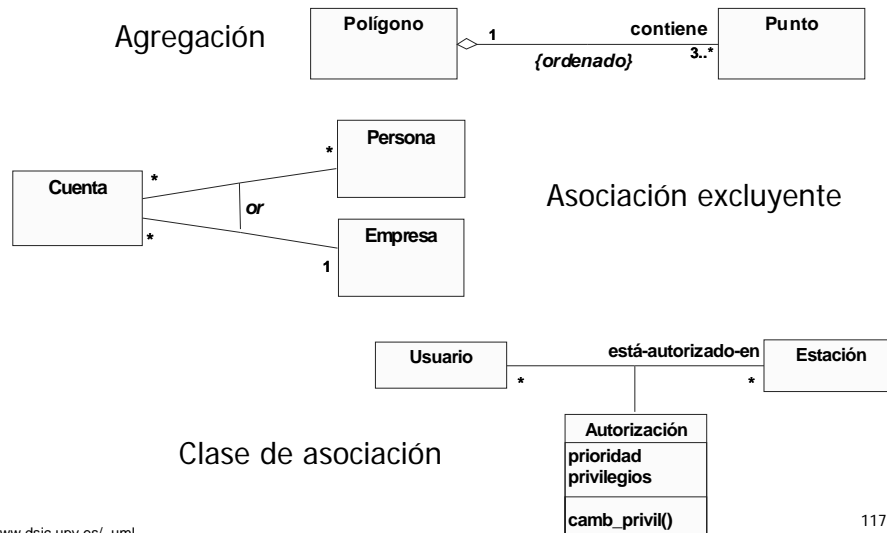
Ejemplos



... Ejemplos



... Ejemplos



☆ www.dsic.upv.es/~uml

117

Clases y Objetos

- Diagrama de Clases y Diagramas de Objetos pertenecen a dos vistas complementarias del modelo
- Un Diagrama de Clases muestra la abstracción de una parte del dominio
- Un Diagrama de Objetos representa una situación concreta del dominio
- Las clases abstractas no son instanciadas

☆ www.dsic.upv.es/~uml

118

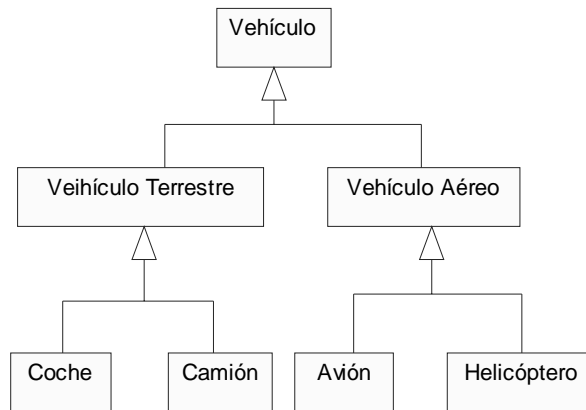
Generalización

- Permite gestionar la complejidad mediante un ordenamiento taxonómico de clases
- Se obtiene usando los mecanismos de abstracción de Generalización y/o Especialización
- La Generalización consiste en factorizar las propiedades comunes de un conjunto de clases en una clase más general

... Generalización

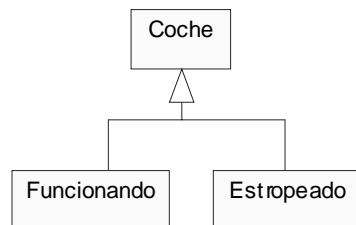
- Nombres usados: clase padre - clase hija.
Otros nombres: superclase - subclase, clase base - clase derivada
- Las subclases heredan propiedades de sus clases padre, es decir, atributos y operaciones (y asociaciones) de la clase padre están disponibles en sus clases hijas

... Generalización



... Generalización

- La especialización es una técnica muy eficaz para la extensión y reutilización



- Restricciones predefinidas en UML:
 - disjunta - no disjunta
 - total (completa) - parcial (incompleta)

... Generalización

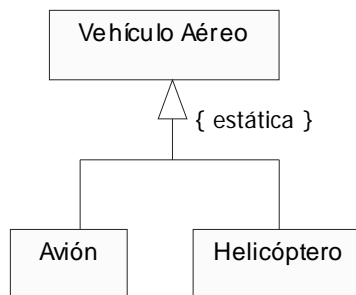
- La noción de clase está próxima a la de conjunto
- Dada una clase, podemos ver el conjunto relativo a las instancias que posee o bien relativo a las propiedades de la clase
- Generalización y especialización expresan relaciones de inclusión entre conjuntos

... Generalización

- Particionamiento del espacio de objetos => Clasificación Estática
- Particionamiento del espacio de estados de los objetos => Clasificación Dinámica
- En ambos casos se recomienda considerar generalizaciones/especializaciones disjuntas

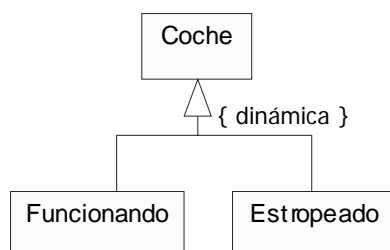
... Generalización

- Un ejemplo de Clasificación Estática:



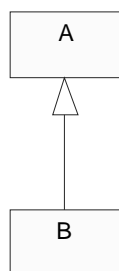
... Generalización

- Un ejemplo de Clasificación Dinámica:



... Generalización

- Extensión: Posibles instancias de una clase
- Intensión: Propiedades definidas en una clase

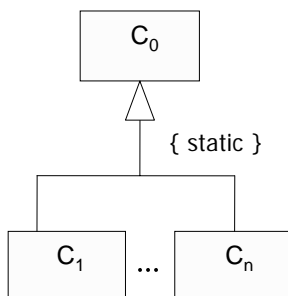


$$\text{int}(A) \subseteq \text{int}(B)$$

$$\text{ext}(B) \subseteq \text{ext}(A)$$

... Generalización

- Clasificación Estática

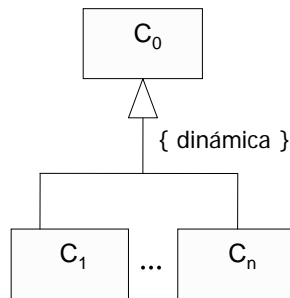


$$\text{ext}(C_0) = \cup \text{ext}(C_i) \Rightarrow \text{completa}$$

$$\text{ext}(C_i) \cap \text{ext}(C_j) = \emptyset \Rightarrow \text{disjunta}$$

... Generalización

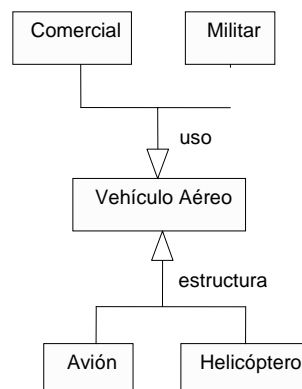
■ Clasificación Dinámica



$\text{ext}(C_0) = \cup \text{ext}(C_i) \Rightarrow \text{completa}$
 $\text{ext}_t(C_i) \cap \text{ext}_t(C_j) = \emptyset \Rightarrow \text{disjunta en } t$
 $\text{ext}_{t_1}(C_i) \cap \text{ext}_{t_2}(C_j) \neq \emptyset \Rightarrow \text{posiblemente no disjunta en diferentes instantes}$

... Generalización

■ Ejemplo: varias especializaciones a partir de la misma clase padre, usando discriminadores:

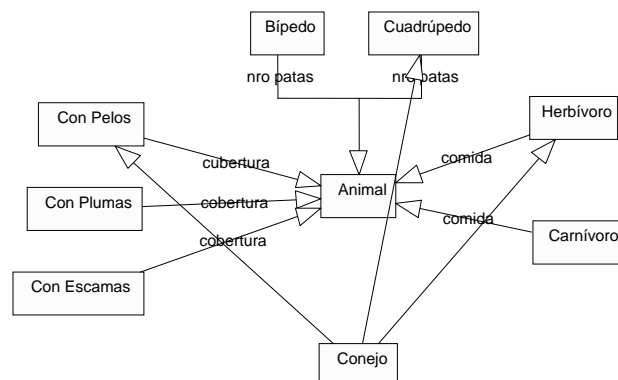


Clasificación Múltiple (herencia múltiple)

- Se presenta cuando una subclase tiene más de una superclase
- La herencia múltiple debe manejarse con precaución. Algunos problemas son el conflicto de nombre y el conflicto de precedencia
- Se recomienda un uso restringido y disciplinado de la herencia. Java y Ada 95 simplemente no ofrecen herencia múltiple

... Herencia Múltiple

- Uso disciplinado de la herencia múltiple: clasificaciones disjuntas con clases padre en hojas de jerarquías alternativas



Principio de Sustitución

- El Principio de Sustitución de Liskow afirma que:

“Debe ser posible utilizar cualquier objeto instancia de una subclase en el lugar de cualquier objeto instancia de su superclase sin que la semántica del programa escrito en los términos de la superclase se vea afectado.”

... Principio de Sustitución

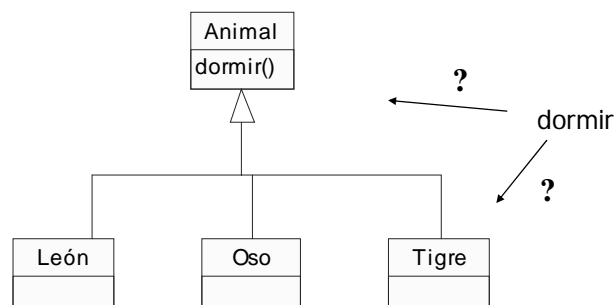
- Dado que los programadores pueden introducir código en las subclases redefiniendo las operaciones, es posible introducir involuntariamente incoherencias que violen el principio de sustitución
- El polimorfismo que veremos a continuación no debería implementarse sin este principio

Polimorfismo

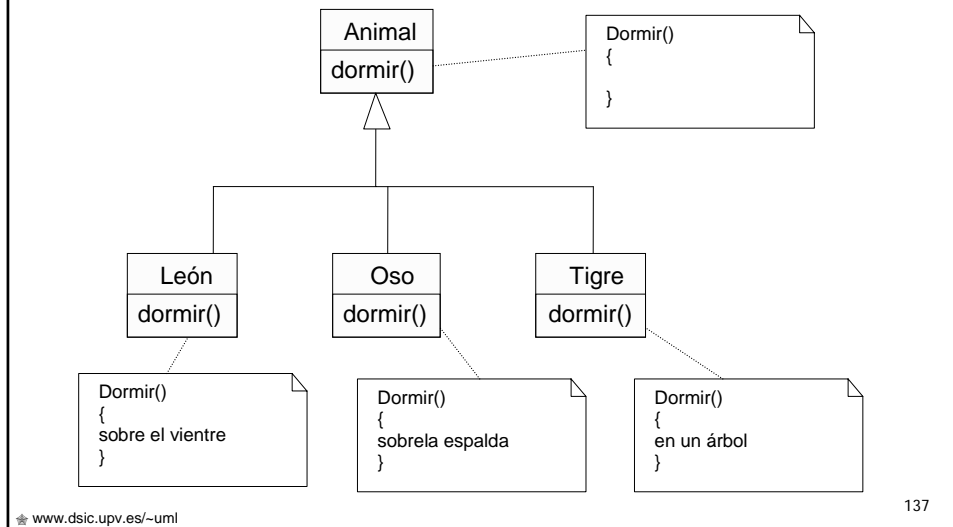
- El término polimorfismo se refiere a que una característica de una clase puede tomar varias formas
- El polimorfismo representa en nuestro caso la posibilidad de desencadenar operaciones distintas en respuesta a un mismo mensaje
- Cada subclase hereda las operaciones pero tiene la posibilidad de modificar localmente el comportamiento de estas operaciones

... Polimorfismo

- Ejemplo: todo animal duerme, pero cada clase lo hace de forma distinta

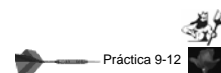


... Polimorfismo



... Polimorfismo

- La búsqueda automática del código que en cada momento se va a ejecutar es fruto del enlace dinámico
- El cumplimiento del Principio de Sustitución permite obtener un comportamiento y diseño coherente



Comportamiento de objetos

Diagrama de Estados

- Los Diagramas de Estados representan autómatas de estados finitos, desde el p.d.v. de los estados y las transiciones
- Son útiles sólo para los objetos con un comportamiento significativo
- El formalismo utilizado proviene de los *Statecharts* (Harel)

... Diagrama de Estados

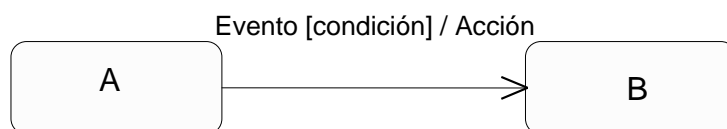
- Cada objeto está en un estado en cierto instante
- El estado está caracterizado parcialmente por los valores algunos de los atributos del objeto
- El estado en el que se encuentra un objeto determina su comportamiento
- Cada objeto sigue el comportamiento descrito en el D. de Estados asociado a su clase
- Los D. De Estados y escenarios son complementarios

... Diagrama de Estados

- Los D. de Estados son autómatas jerárquicos que permiten expresar concurrencia, sincronización y jerarquías de objetos
- Los D. de Estados son grafos dirigidos
- Los D. De Estados de UML son deterministas
- Los estados inicial y final están diferenciados del resto
- La transición entre estados es instantánea y se debe a la ocurrencia de un evento

... Diagrama de Estados

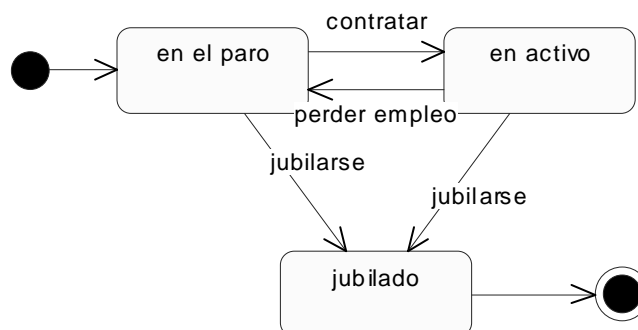
- Estados y Transiciones



Tanto el evento como la acción se consideran instantáneos

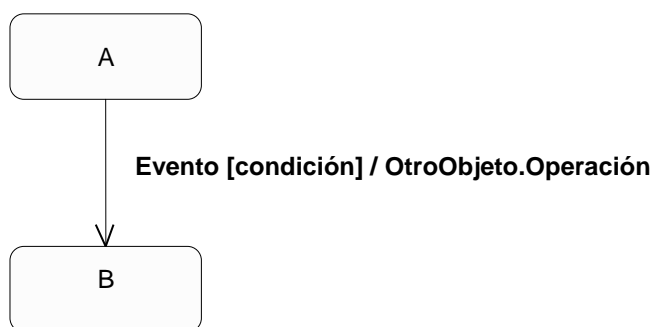
... Diagrama de Estados

- Ejemplo de un Diagrama de Estados para la clase persona:



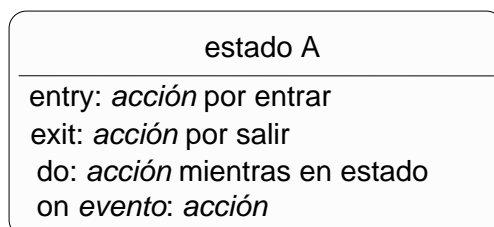
Acciones

- Podemos especificar la solicitud de un servicio a otro objeto como consecuencia de la transición:



... Acciones

- Se puede especificar el ejecutar una acción como consecuencia de entrar, salir, estar en un estado, o por la ocurrencia de un evento:

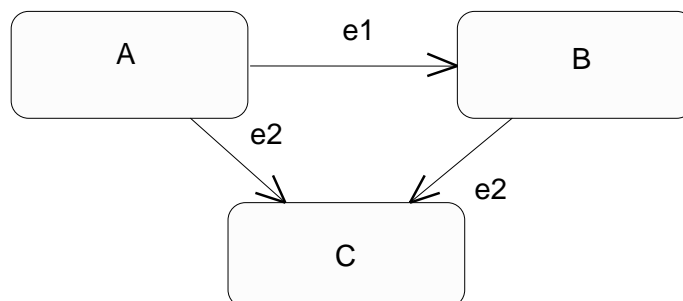


Generalización de Estados

- Podemos reducir la complejidad de estos diagramas usando la generalización de estados
- Distinguimos así entre superestado y subestados
- Un estado puede contener varios subestados disjuntos
- Los subestados heredan las variables de estado y las transiciones externas

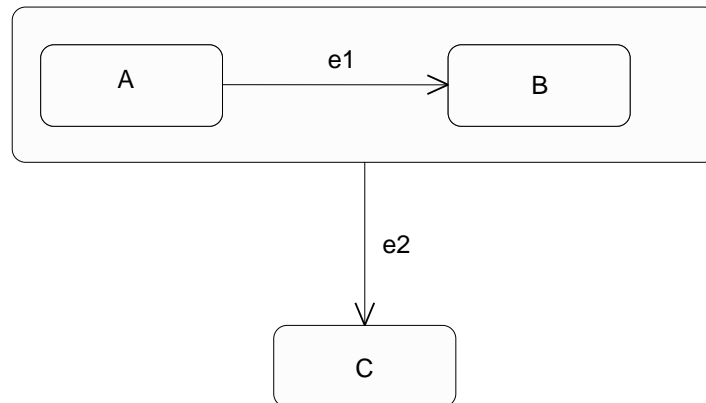
Generalización de Estados

- Ejemplo:



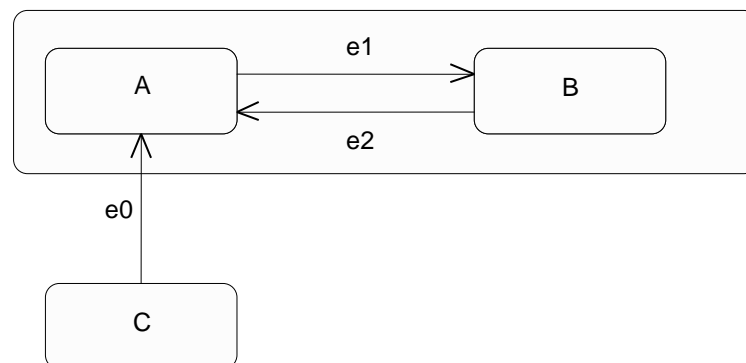
Generalización de Estados

- Quedaría como:



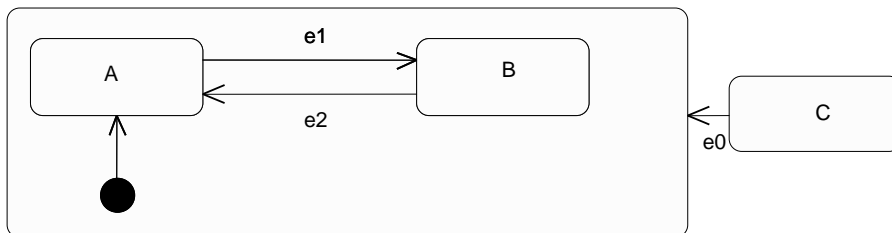
... Generalización de Estados

- Las transiciones de entrada deben ir hacia subestados específicos:



... Generalización de Estados

- Es preferible tener estados iniciales de entrada a un nivel de manera que desde los niveles superiores no se sepa a qué subestado se entra:

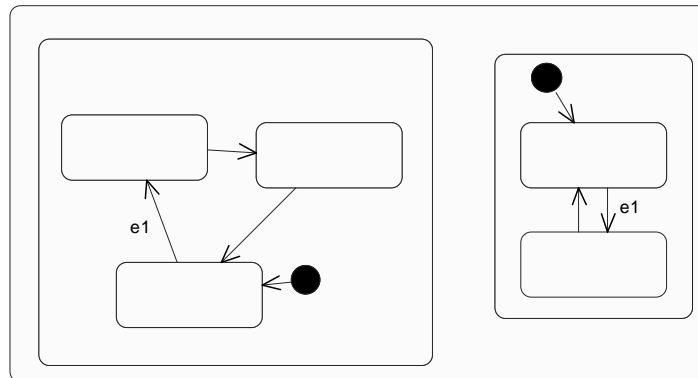


... Generalización de Estados

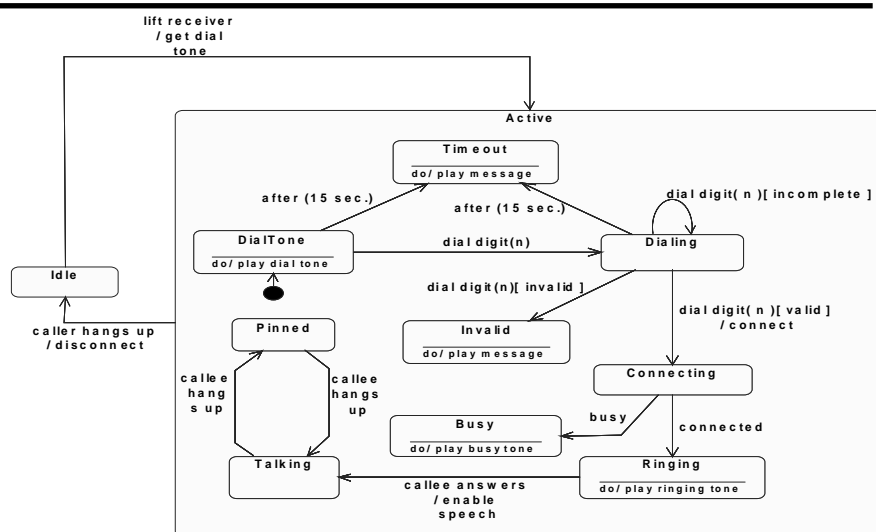
- La agregación de estados es la composición de un estado a partir de varios estados independientes
- La composición es concurrente por lo que el objeto estará en alguno de los estados de cada uno de los subestados concurrentes

... Generalización de Estados

■ Ejemplo:



... Generalización de Estados

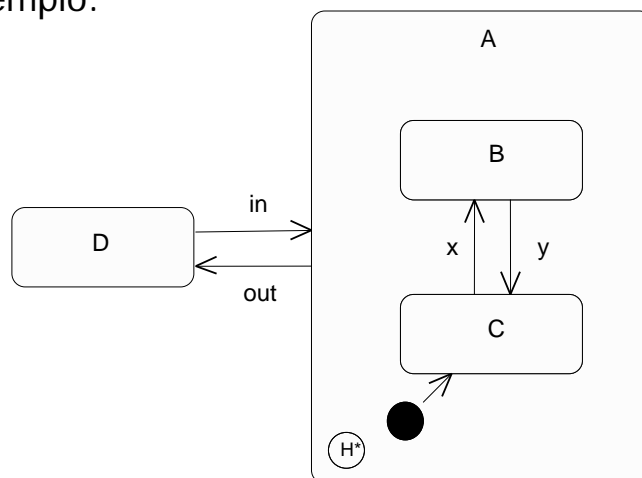


Historia

- Por defecto, los autómatas no tienen memoria
- Es posible memorizar el último subestado visitado para recuperarlo en una transición entrante en el superestado que lo engloba
- También es posible la memorización para cualquiera de los subestados anidados (aparece un * junto a la H)

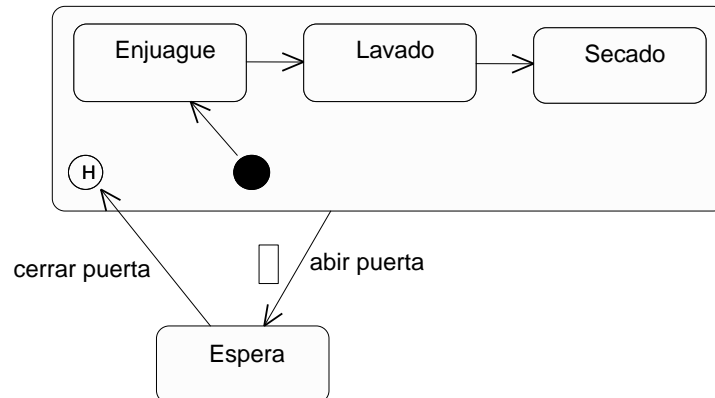
... Historia

- Ejemplo:



... Historia

- Ejemplo:

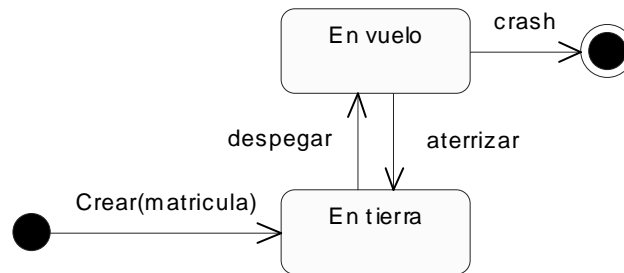


Destrucción del Objeto

- La destrucción de un objeto es efectiva cuando el flujo de control del autómata alcanza un estado final no anidado
- La llegada a un estado final anidado implica la "subida" al superestado asociado, no el fin del objeto

... Destrucción de Objeto

- Ejemplo:



Transiciones temporizadas

- Las esperas son actividades que tienen asociada cierta duración
- La actividad de espera se interrumpe cuando el evento esperado tiene lugar
- Este evento desencadena una transición que permite salir del estado que alberga la actividad de espera. El flujo de control se transmite entonces a otro estado

... Transiciones temporizadas

- Ejemplo:

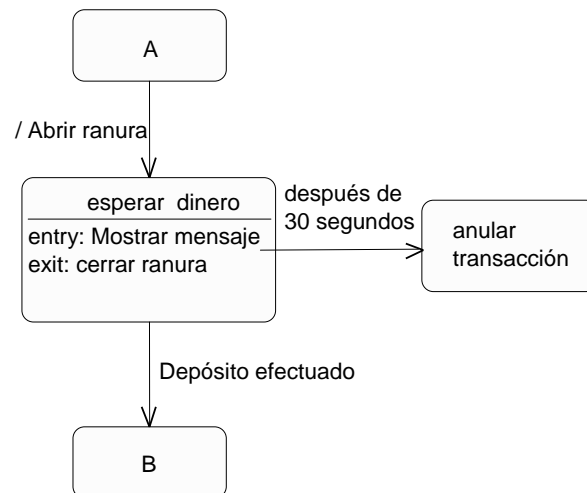
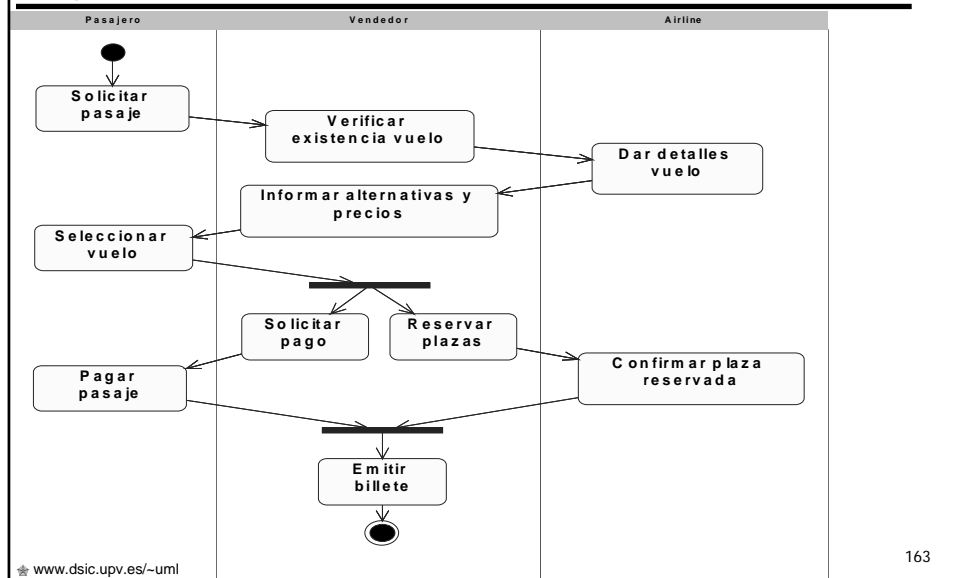


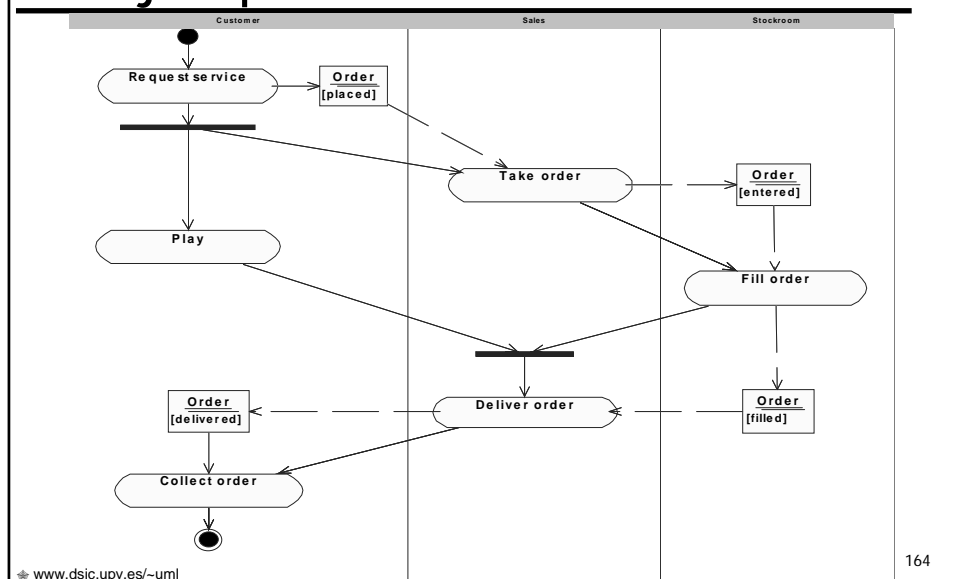
Diagrama de Actividad

- El Diagrama de Actividad es una especialización del Diagrama de Estado, organizado respecto de las acciones y usado para especificar:
 - Un método
 - Un caso de uso
 - Un proceso de negocio (Workflow)
- Las actividades se enlazan por transiciones automáticas. Cuando una actividad termina se desencadena el paso a la siguiente actividad

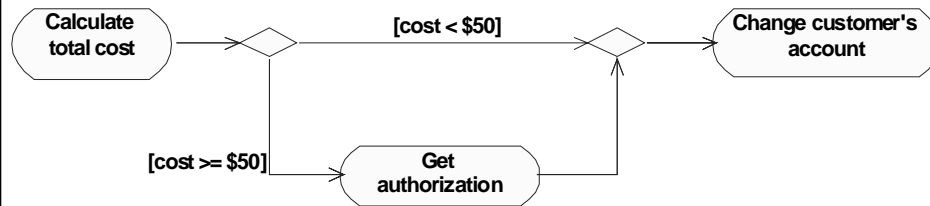
Ejemplo (con *swim lines*)



... Ejemplos



... Ejemplos



Componentes

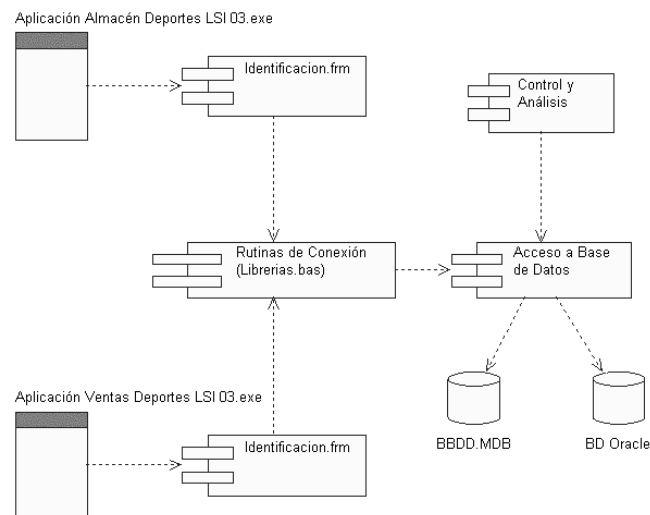
Diagrama de Componentes

- Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones
- Muestran las opciones de realización incluyendo código fuente, binario y ejecutable

...Diagrama de Componentes

- Los componentes representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes de Ada, bibliotecas cargadas dinámicamente, etc.
- Las relaciones de dependencia se utilizan en los diagramas de componentes para indicar que un componente utiliza los servicios ofrecidos por otro componente

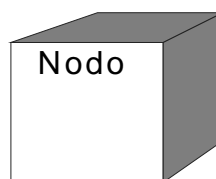
...Diagrama de Componentes



Distribución y despliegue de Componentes

Diagrama de Despliegue

- Los Diagramas de Despliegue muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos

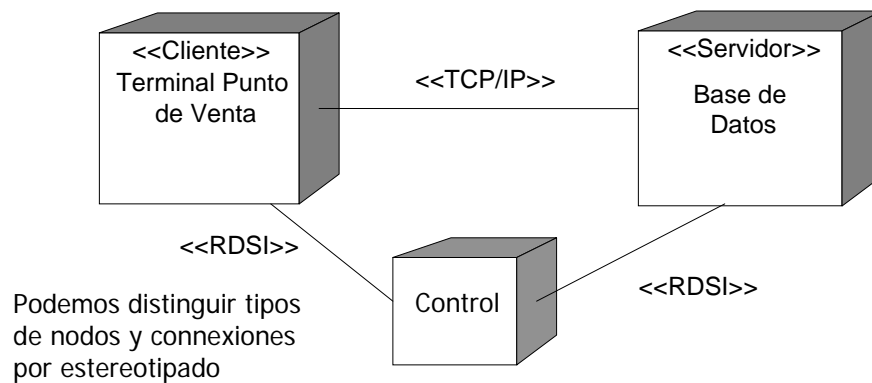


... Diagrama de Despliegue

- Los estereotipos permiten precisar la naturaleza del equipo:
 - Dispositivos
 - Procesadores
 - Memoria
- Los nodos se interconectan mediante soportes bidireccionales que pueden a su vez estereotiparse

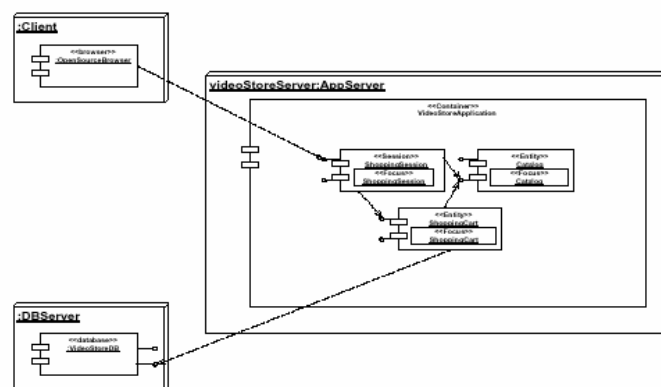
... Diagrama de Despliegue

- Ejemplo de conexión entre nodos:



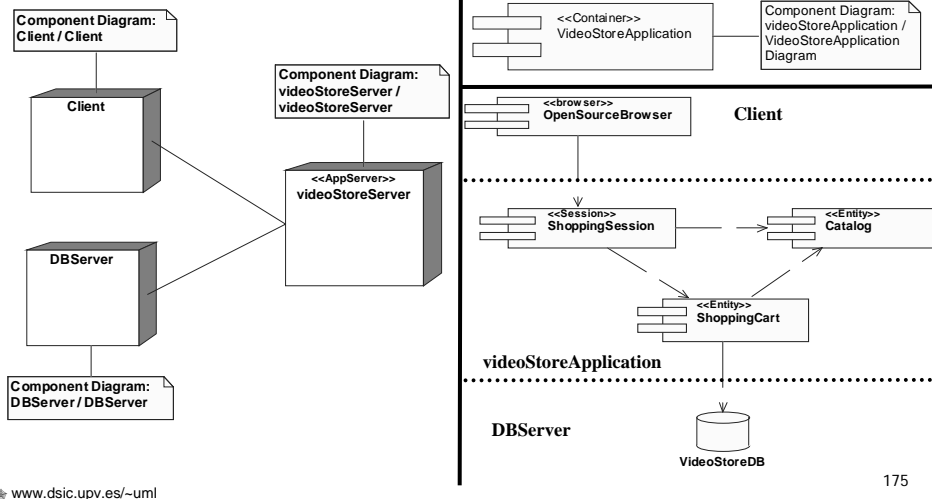
... Diagramas de Despliegue

- Ejemplo:



... Diagramas de Despliegue

■ Ejemplo:



175

Object Constraint Language OCL

176

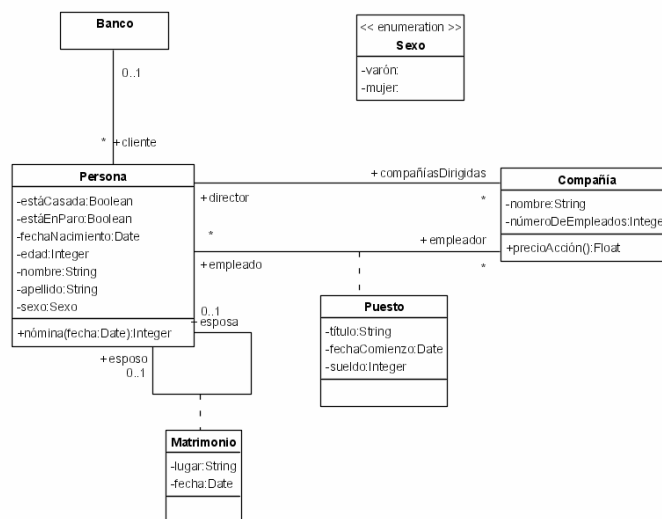
¿Qué es OCL?

- OCL es un lenguaje formal usado para describir expresiones acerca de modelos UML
- OCL es parte del estandar UML y fue desarrollado en IBM
- Las evaluación de expresiones OCL no afecta el estado del sistema en ejecución

Usos de OCL

- Lenguaje de consulta
- Especificación de invariantes en clases y tipos
- Especificación de invariantes de tipo para Estereotipos
- Describir pre- y post condiciones en Operaciones y Métodos
- Describir Guardas
- Especificar destinatarios para mensajes y acciones
- Especificar restricciones en Operaciones
- Especificar reglas de derivación para atributos

Ejemplo



★ www.dsic.upv.es/~umi

179

Invariantes

- "El número de empleados debe ser mayor que 50"

```
self.númeroDeEmpleados > 50 (siendo el contexto Company)
```

context Compañía

inv: self. númeroDeEmpleados > 50

context c:Compañía

inv: c.númeroDeEmpleados > 50

context c:Compañía

inv suficientesEmpleados: c.númeroDeEmpleados > 50

✱ www.dsic.upv.es/~uml

180

Pre- Post condiciones

- Sintaxis

context *NombreTipo::NombreOperación*(*Param₁ : Tipo₁, ...*):*TipoRetorno*
pre *parametroOk*: *param₁ > ...*
post *resultadoOk* : *result = ...*

- Ejemplo

context *Persona::nómina*(*fecha : Date*) : *Integer*
post: *result = 5000*

Valores iniciales y derivados

- Sintaxis

context *NombreTipo::NombreAtributo*: *Tipo*
init: -- alguna expresión representando el valor inicial

context *NombreTipo::NombreRolAsociación*: *Tipo*
init: -- alguna expresión representando la regla de derivación

- Ejemplo

context *Persona::nómina* : *Integer*
init: *padres.nómina->sum()* * 1%
derive: **if** *menorDeEdad* **then**
 padres.nómina->sum() * 1%
 else
 puesto.sueldo
 endif

Expresiones Let

- Ejemplo

```
context Persona inv:  
let ingresos : Integer = self.puesto.sueldo->sum() in  
if estáEnParo then  
    ingresos < 100  
else  
    ingresos >= 100  
endif
```

Definiciones

- Ejemplo

```
context Persona  
def: ingresos : Integer = self.puesto.sueldo->sum()  
def: apodo : String = 'Gallito rojo'  
def: tieneElTitulo(t : String) : Boolean = self.puesto->exists(titulo = t)
```

Navegación

- Ejemplos

context Compañía
inv: self.director.estáEnparo = false
inv: self.empleado->notEmpty()

context Compañía
inv: self.director.edad > 40

context Persona
inv: self.empleador->size() < 3

context Persona
inv: self.empleador->isEmpty()

context Persona
inv: self.esposa->notEmpty() **implies** self.esposa.sexo = Sexo::mujer

... Navegación

- Ejemplo

a) "Los casados tienen al menos 18 años de edad"

context Persona **inv:**
self.esposa->notEmpty() **implies** self.esposa.edad >= 18 **and**
self.esposo->notEmpty() **implies** self.esposo.edad >= 18

"Una compañía tiene a lo más 50 empleados"

context Compañía **inv:**
self.empleado->size() <= 50

IV Proceso de Desarrollo de SW basado en UML

★ www.dsic.upv.es/~uml

187

IV. Proceso de Desarrollo de SW basado en UML

¿Qué es un Proceso de Desarrollo de SW?

- ▣ Define Quién debe hacer Qué, Cuándo y Cómo debe hacerlo



- ▣ No existe un proceso de software universal. Las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigen que el proceso sea configurable

★ www.dsic.upv.es/~uml

188

Rational Unified Process (RUP)



Rational Unified Process
1998

Rational Objectory Process
1996-1997

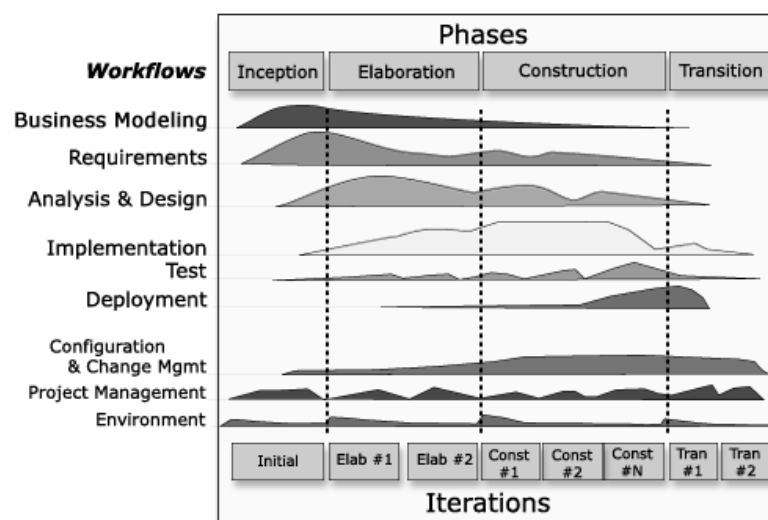
Objectory Process
1987-1995

Enfoque Ericsson

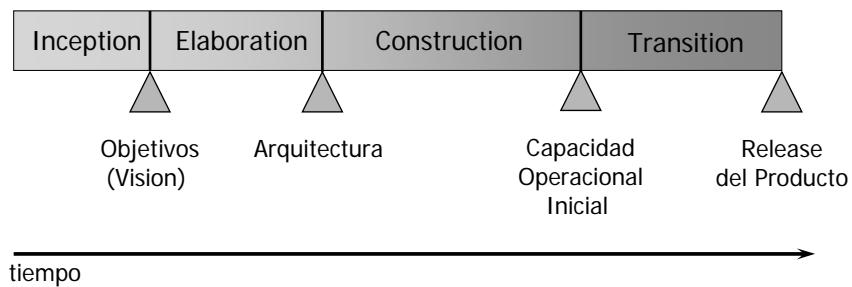
- Pruebas funcionales
- Pruebas de desempeño
- Gestión de requisitos
- Gestión de cambios y configuración
- Ingeniería de Negocio
- Ingeniería de datos
- Diseño de interfaces

UML

Dos Dimensiones



Fases e Hitos (Milestones)



Elementos en RUP

□ Workflows (Disciplinas)

Workflows Primarios

- Business Modeling (Modelado del Negocio)
- Requirements (Requisitos)
- Analysis & Design (Análisis y Diseño)
- Implementation (Implementación)
- Test (Pruebas)
- Deployment (Despliegue)

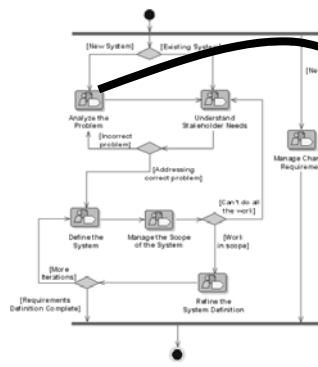
Workflows de Apoyo

- Environment (Entorno)
- Project Management (Gestión del Proyecto)
- Configuration & Change Management (Gestión de Configuración y Cambios)

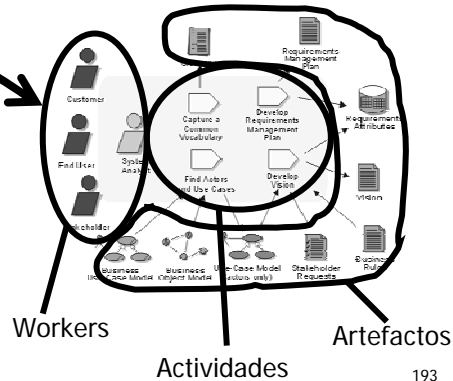
... Elementos en RUP

Workflow, Workflow Detail , Workers, Actividades y Artefactos
 Ejemplo

Workflow: Requirements



Workflow Detail: Analyse the Problem



☆ www.dsic.upv.es/~uml

193

... Elementos en RUP

Workers

Analyst workers

- Business-Process Analyst
- Business Designer
- Business-Model Reviewer
- Requirements Reviewer
- System Analyst
- Use-Case Specifier
- User-Interface Designer

Developer workers

- Architect
- Architecture Reviewer
- Capsule Designer
- Code Reviewer
- Database Designer
- Design Reviewer
- Designer
- Implementer
- Integrator

Testing professional workers

- Test Designer
- Tester

Manager workers

- Change Control Manager
- Configuration Manager
- Deployment Manager
- Process Engineer
- Project Manager
- Project Reviewer

Other workers

- Any Worker
- Course Developer
- Graphic Artist
- Stakeholder
- System Administrator
- Technical Writer
- Tool Specialist

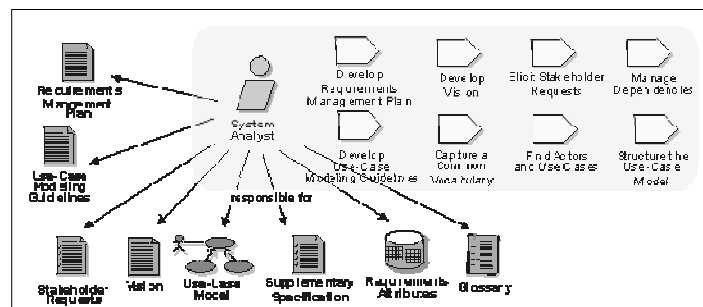
☆ www.dsic.upv.es/~uml

194

... Elementos en RUP

Workers, Actividades, Artefactos

Ejemplo: System Analyst Worker



... Elementos en RUP

Artefactos

- Resultado parcial o final que es producido y usado durante el proyecto. Son las entradas y salidas de las actividades
- Un artefacto puede ser un documento, un modelo o un elemento de modelo
- Conjuntos de Artefactos
 - Business Modeling Set
 - Requirements Set
 - Analysis & Design Set
 - Implementation Set
 - Test Set
 - Deployment Set
 - Project Management Set
 - Configuration & Change Management Set
 - Environment Set

... Elementos en RUP

Artefactos, Workers, Actividades

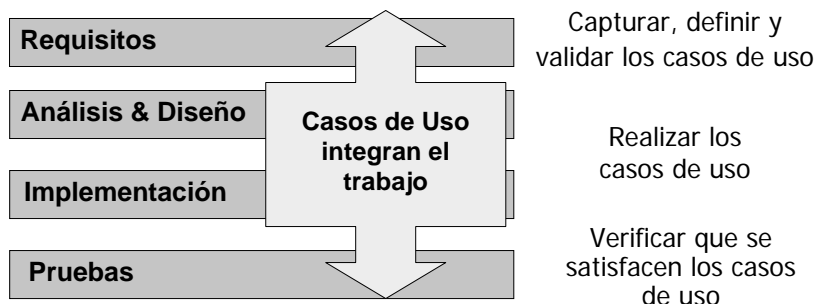
Ejemplo: Business Modeling Artifact Set



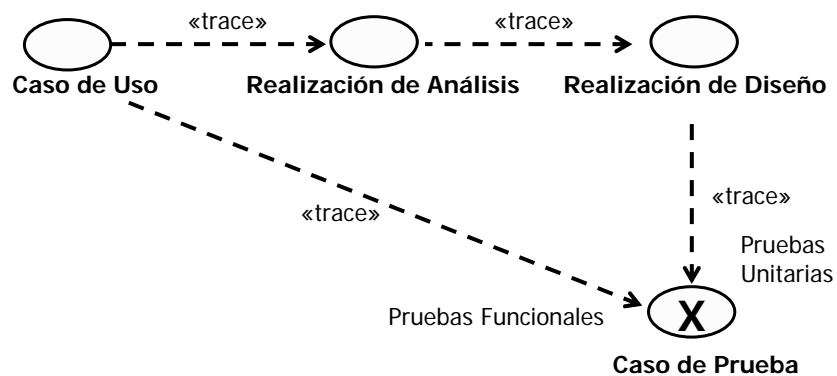
Características Esenciales de RUP

- ▣ Proceso Dirigido por los Casos de Uso
- ▣ Proceso Iterativo e Incremental
- ▣ Proceso Centrado en la Arquitectura

Proceso dirigido por los Casos de Uso



... Proceso dirigido por los Casos de Uso



... Proceso dirigido por los Casos de Uso

Estado de aspectos de los Casos de Uso al finalizar cada fase

	Modelo de Negocio Terminado	Casos de Uso Identificados	Casos de Uso Descritos	Casos de Uso Analizados	Casos de Uso Diseñados, Implementados y Probados
Fase de Concepción	50% - 70%	50%	10%	5%	Muy poco, puede que sólo algo relativo a un prototipo para probar conceptos
Fase de Elaboración	Casi el 100%	80% o más	40% - 80%	20% - 40%	Menos del 10%
Fase de Construcción	100%	100%	100%	100%	100%
Fase de Transición					

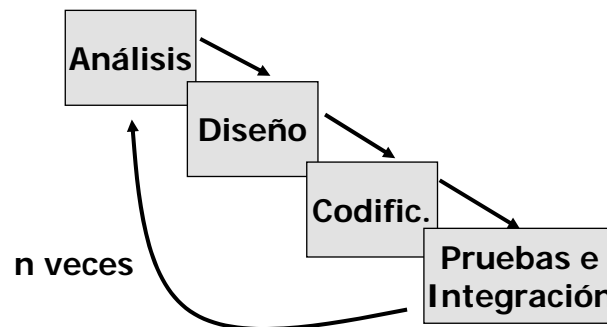
The Unified Software Development Process. I. Jacobson, G. Booch y J. Rumbaugh. página 358. Addison-Wesley, 1999.

Proceso Iterativo e Incremental

- El ciclo de vida iterativo se basa en la evolución de prototipos ejecutables que se muestran a los usuarios y clientes
- En el ciclo de vida iterativo a cada iteración se reproduce el ciclo de vida en cascada a menor escala
- Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes

... Proceso Iterativo e Incremental

- Las actividades se encadenan en una minicascada con un alcance limitado por los objetivos de la iteración

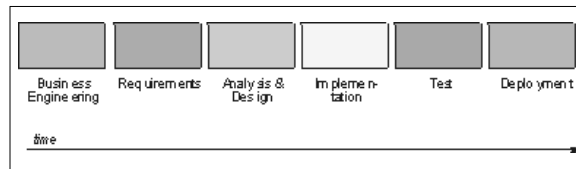


... Proceso Iterativo e Incremental

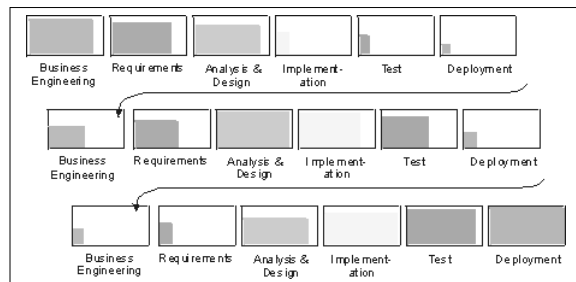
- Cada iteración comprende:
 - Planificar la iteración (estudio de riesgos)
 - Análisis de los Casos de Uso y escenarios
 - Diseño de opciones arquitectónicas
 - Codificación y pruebas. La integración del nuevo código con el existente de iteraciones anteriores se hace gradualmente durante la construcción
 - Evaluación de la entrega ejecutable (evaluación del prototipo en función de las pruebas y de los criterios definidos)
 - Preparación de la entrega (documentación e instalación del prototipo)

Proceso Iterativo e Incremental

Enfoque
Secuencial

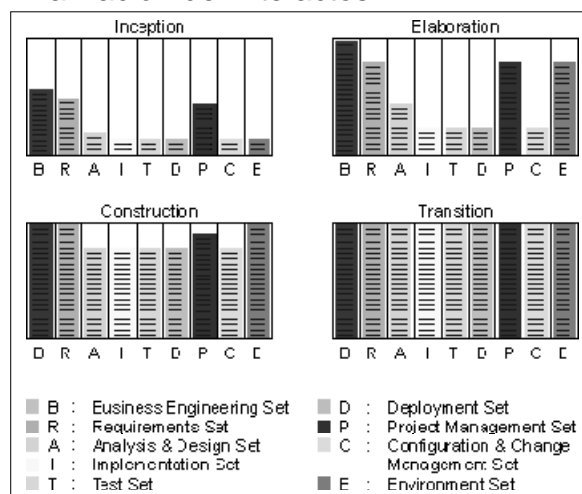


Enfoque
Iterativo e
Incremental



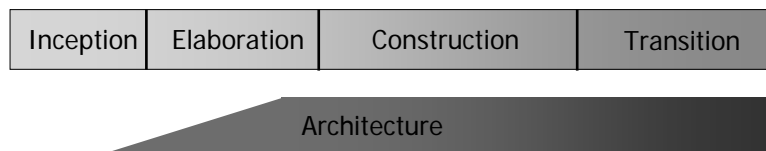
... Proceso Iterativo e Incremental

Grado de Finalización de Artefactos

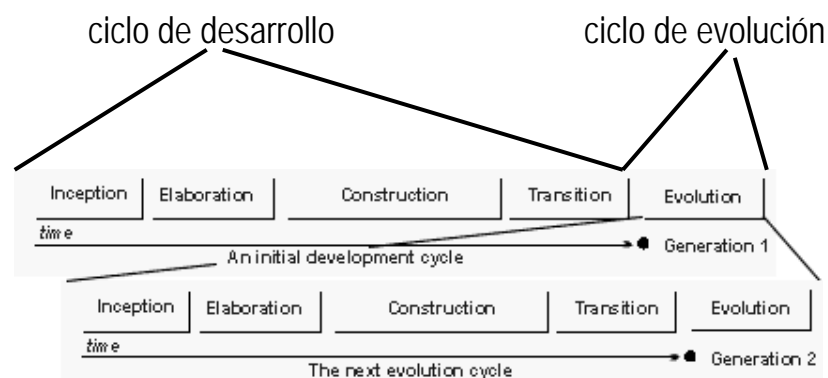


Proceso Centrado en la Arquitectura

- Arquitectura de un sistema es la organización o estructura de sus partes más relevantes
- Un arquitectura ejecutable es una implementación parcial del sistema, construida para demostrar algunas funciones y propiedades
- RUP establece refinamientos sucesivos de una arquitectura ejecutable, construida como un prototipo evolutivo



Fases, Release, Base Line, Generación



release
(producto al final de
una iteración)

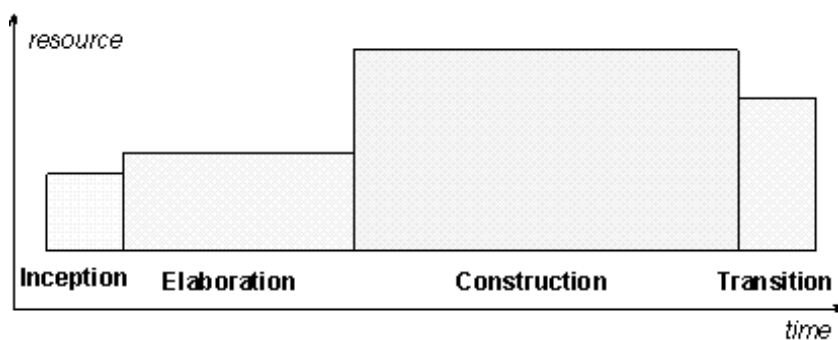
base line
(release asociada
a un hito)

generación
(release final de
un ciclo de desarrollo)

Esfuerzo y dedicación por Fases en RUP

	Inicio	Elaboración	Construcción	Transición
Esfuerzo	5 %	20 %	65 %	10%
Tiempo Dedicado	10 %	30 %	50 %	10%

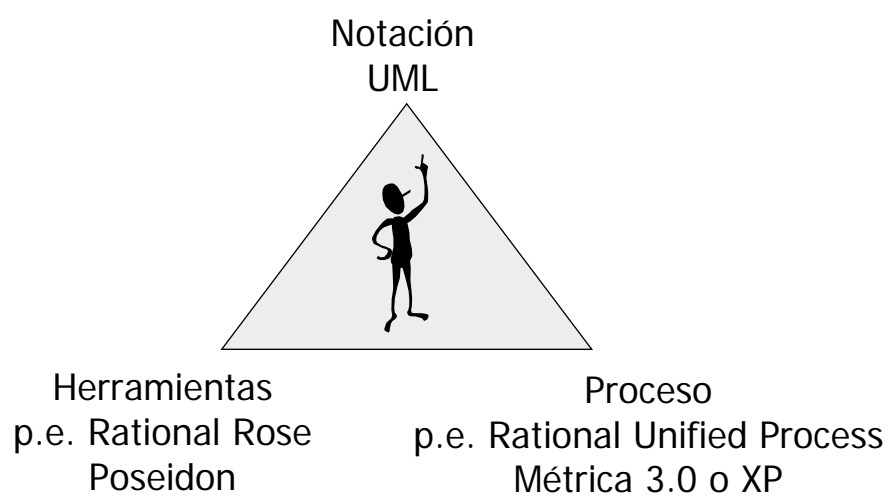
Distribución de Recursos por Fases en RUP



V

Conclusiones

Claves en el Desarrollo de SI



Modelado de SI: Algunas Reflexiones

- ¿Cuál es el propósito de nuestros modelos?
 - “Documentar” (a posteriori)
 - Comunicar ideas y estudiar alternativas
 - Tomar decisiones de análisis/diseño que dirijan la implementación
 - Generar parcial o totalmente una implementación a partir de los modelos
- Pragmatismo, los modelos deben ser útiles. Principio básico: “Sencillez y Elegancia”
- Gestión de modelos
 - Distintos nivel de abstracción, expresados en diferentes modelos
 - Seguimiento de transformaciones durante el proceso (*Traceability*)
 - Sincronización de modelos
- Dificultades para la introducción de notaciones y herramientas de modelado. La importancia del Proceso de Desarrollo

Adaptabilidad al contexto del proyecto



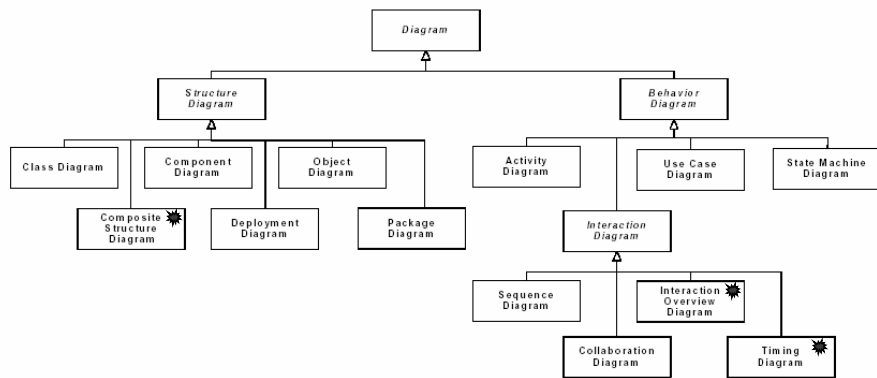
Tendencias

- UML: actualmente la notación más detallada, amplia y consensuada para modelar software orientado a objetos
- Dificultades actuales para derivar de forma directa una implementación a partir de los modelos UML
 - Entornos de programación visual y el paradigma OO subyacente
 - Utilización de bases de datos relacionales
 - Arquitectura de 3 capas
 - Frameworks de persistencia para materializar y desmaterializar objetos
- Metodologías de desarrollo de software y el papel que juega UML en ellas
 - Modelado Ágil
 - Modelado opcional y/o desechable (en Metodologías Ágiles)

... Tendencias

- Nuevas versiones de UML, uff!
- Extensiones de UML (SysML, www.sysml.org)
- Generación automática de código a partir de modelos
 - Model-Driven Development (MDD), Model-Driven Architecture (MDA), Compiladores de Modelos
 - Round-trip engineering. Convergencia entre herramientas CASE e IDEs
- Extendiendo UML mediante Profiles
(www.objecteering.com/products_uml_profile_builder.php)
- Modelado y generación de código en dominios específicos (más allá de UML)
 - Eclipse Modeling Framework (EMF, download.eclipse.org/tools/emf/scripts/home.php)
 - Microsoft Tools for Domain Specific Languages
 - Domain-Specific Modeling (DSM, www.dsmforum.org)
 - Meta CASE Tools (www.metacase.com)

Diagramas en UML 2.0



★ www.dsic.upv.es/~uml

217

V. Conclusiones

Bibliografía Adicional

UML

- www.omg.org/uml/
- Meta-links www.cetus-links.org/oo_uml.html
- Martin Fowler, autor de "UML Destilled" ("UML Gota a Gota")
<http://www.martinfowler.com/>

Herramientas CASE

- Herramientas basadas en UML
www.objectsbydesign.com/tools/umltools_byPrice.html
- International Council in SE (INCOSE) www.incose.org/tools/

Otras

- Revista IEEE Software, Conferencias: OOPSLA, ECOOP
- Patrones <http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html>,
- Foro UML en yahoo: <http://groups.yahoo.com/group/uml-forum/>

★ www.dsic.upv.es/~uml

218