

Modelado de Objetos con UML



Contenidos

1. Visión General de UML.....	3
Que es UML?	3
Vistas de UML	3
Relación entre Areas, Vistas, y Diagramas de UML.....	4
2. La Vista Estática.....	5
Clasificadores	5
Tipos de Clasificadores.....	5
Clase & Objeto	6
Objetos entidad.....	8
Objetos Interface.....	8
Objetos de control.....	8
Clases abstractas y concretas.....	9
Operaciones	9
Atributos.....	10
Visibilidad	10
Interfaz	11
Tipos de datos.....	11
Niveles de Significado	11
Relaciones	12
Asociación	12
Generalizacion	16
Realización	17
Dependencias.....	17
Restricción.....	18
Instancias	18
Diagrama de Objetos.....	18
3. La Vista de Casos de Uso.....	19
Actor	19
Caso de uso.....	20
Modelado de casos de uso	21
Herramientas de modelado de casos de uso	22
Ejemplo: ATM.....	22
Relaciones entre casos de uso.....	23
4. La Vista de la Máquina de Estados.....	26
Máquina de estados.....	26

Evento	26
Evento de señal	26
Evento de llamada.....	27
Evento de cambio.....	27
Evento de tiempo	27
Estado.....	28
Transición.....	28
Transición externa.....	28
Evento disparador	28
Condición de guarda	28
Transición de finalización	28
Acción	28
Cambio de estado.....	29
Estados anidados.....	29
Acciones de entrada y salida	29
Transición interna	29
Estados Compuestos	30
5. La Vista de Actividades	31
Diagrama de actividades	31
Actividades y otras vistas.....	31
6. La Vista de Interacción	33
Colaboración	33
Interacción.....	34
Diagrama de secuencia.....	34
Diagrama de colaboración.....	36
Mensaje	37
Flujos.....	38
Patrones	38
7. Las Vistas Físicas	40
Descripción.....	40
Componente.....	40
Nodo.....	41
8. La vista de Gestión del Modelo.....	42
Paquete	42
Dependencias en los paquetes	42
Dependencias de acceso e importación.....	43
Modelo y Subsistema.....	43
9. Mecanismos de Extensibilidad	44

1. Visión General de UML

Que es UML?

- UML quiere decir Unified Modeling Language
- UML es un lenguaje standard para **visualizar, especificar, construir y documentar** los artefactos de un sistema de software
- UML puede usarse en las diferentes etapas del ciclo de vida del desarrollo y en diferentes tecnologías de implementación
- UML es independiente del proceso de desarrollo de software

Vistas de UML

Una vista es un subconjunto de construcciones de modelado que se enfocan en un aspecto en particular del sistema.

Las vistas pueden dividirse en tres áreas: **clasificación estructural, comportamiento dinámico, y gestión del modelo.**

La **clasificación estructural** describe los elementos del sistema y sus relaciones con otros elementos.

Los clasificadores incluyen clases, casos de uso, componentes, y nodos. Los clasificadores proveen la base sobre la que se construye el comportamiento dinámico.

Las vistas de clasificación estructural incluyen:

- Vista Estática
 - Diagrama de Clases
- Vista de Casos de Uso
 - Diagrama de Casos de Uso
- Vista de Implementación.
 - Diagrama de Componentes
 - Diagrama de Despliegue

El **comportamiento dinámico** describe el comportamiento del sistema a través del tiempo. El comportamiento puede ser descrito como una serie de cambios a fotos del sistema tomadas desde la vista estática.

Las vistas de comportamiento dinámico incluyen:

- Vista de la Máquina de Estados
 - Diagrama de Estados (ciclo de vida de objeto)
- Vista de Actividades
 - Diagrama de Actividades
- Vista de Interacción
 - Diagrama de Secuencia
 - Diagrama de Colaboración

La **gestión del modelo** describe la organización de los modelos mismos en unidades jerárquicas. El paquete es la unidad de organización para los modelos. Tipos especiales de paquetes son los modelos y subsistemas.

- Vista de Gestión

- Diagrama de Clases (paquetes, subsistemas, modelos)

Relación entre Areas, Vistas, y Diagramas de UML

Area	Vista	Diagramas	Principales Conceptos
Estructural	Vista Estática	Diagrama de Clases	Clase, asociación, generalización, dependencia, realización, interface.
	Vista de Casos de Uso	Diagrama de Casos de Uso	Caso de uso, actor, asociación, extensión, inclusión, generalización.
	Vista de Implementación	Diagrama de Componentes	Componente, interface, dependencia, realización.
	Vista de Despliegue	Diagrama de Despliegue	Nodo, componente, dependencia, localización.
Dinámica	Vista de Máquina de Estados	Diagramas de Estados (ciclo de vida de objeto)	Estado, evento, transición, acción.
	Vista de Actividades	Diagrama de Actividades	Estado, actividad, transición, concurrencia (fork), reunión (join).
	Vista de Interacción	Diagrama de Secuencia	Interacción, objeto, mensaje, activación.
		Diagrama de Colaboración	Colaboración, interacción, rol de colaboración, mensaje.
Gestión del Modelo	Vista de Gestión del Modelo	Diagrama de clases	Paquete, subsistema, modelo.
Extensibilidad	Todos	Todos	Restricciones, estereotipos, valores etiquetados.

Major Area	View	Diagrams	Main Concepts
structural	static view	class diagram	class, association, generalization, dependency, realization, interface
	use case view	use case diagram	use case, actor, association, extend, include, use case generalization
	implementation view	component diagram	component, interface, dependency, realization
	deployment view	deployment diagram	node, component, dependency, location
dynamic	state machine view	statechart diagram	state, event, transition, action
	activity view	activity diagram	state, activity, completion transition, fork, join
	interaction view	sequence diagram	interaction, object, message, activation
		collaboration diagram	collaboration, interaction, collaboration role, message
model management	model management view	class diagram	package, subsystem, model
extensibility	all	all	constraint, stereotype, tagged values

Vistas y Diagramas de UML

2. La Vista Estática

La vista estática es la base de UML.

Los elementos de la vista estática de un modelo son los **conceptos significativos** en una aplicación, incluyendo conceptos del mundo real, conceptos abstractos, conceptos de implementación, conceptos de computación y todo tipo de conceptos encontrados en el sistema.

Propósitos de la Vista Estática

- Captura la estructura de objetos.
- Es la base sobre la que se construyen las otras vistas.
- Es un modelo incremental.

Los elementos claves en la vista estática son los **clasificadores** y sus **relaciones**.

Clasificadores

Clasificador: es un concepto discreto en el modelo que tiene identidad, estado, comportamiento, y relaciones.

Un clasificador es un elemento de modelado que describe cosas. Hay varias clases de clasificadores, como las clases, interfaces, y tipos de datos.

Los **casos de uso** son clasificadores que mapean los aspectos de comportamiento.

Los propósitos de implementación implican clasificadores como **subsistemas**, **componentes**, y **nodos**.

Un **paquete** es una unidad de organización de uso general para poseer y manejar el contenido de un modelo. Todo elemento del modelo está contenido dentro de un paquete.

Tipos de Clasificadores

Clasificadores de elementos del sistema:




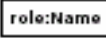
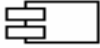




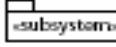

- Clase
- Interfaz
- Tipos de datos.

Clasificadores de conceptos de comportamiento:

- Caso de Uso
- Clasificadores de cosas del entorno:
- Actor

Clasificadores de estructuras de implementación:

- Componente
- Nodo
- Subsistema

<i>Classifier</i>	<i>Function</i>	<i>Notation</i>
actor	An outside user of a system	
class	A concept from the modeled system	
class-in-state	A class restricted to being in a given state	
classifier role	A classifier restricted to a particular usage in a collaboration	
component	A physical piece of a system	
data type	A descriptor of a set of primitive values that lack identity	
interface	A named set of operations that characterize behavior	
node	A computational resource	
signal	An asynchronous communication among objects	
subsystem	A package that is treated as a unit with a specification, implementation, and identity	
use case	A specification of the behavior of an entity in its interaction with outside agents	

Clase & Objeto

Objeto = estructura + operaciones + estado interno + identidad.

Un objeto es una *instancia* de una clase.

Clase:

- Conjunto de objetos con estructura, comportamiento, relaciones, y semántica común.
- Representa algo del Dominio del Problema o de la Solución.

Ejemplos

- algo físico → Avión
- algo del negocio → Pedido
- un concepto lógico → Horario
- algo de la aplicación → Window, Botón, Menú
- algo del comportamiento → Tarea, Proceso

Una clase representa un **concepto discreto** dentro de la aplicación que se está modelando: una cosa física (avión), una cosa de negocios (pedido), una cosa lógica (horario), una cosa de una aplicación (boton, ventana), una cosa del computador (tabla de hash), o una cosa del comportamiento (una tarea).

Una clase es el descriptor de un conjunto de objetos con una estructura, comportamiento y relaciones similares.

Un **objeto** es una entidad discreta con identidad, estado y un comportamiento invocable. El **estado** es descripto por atributos y asociaciones.

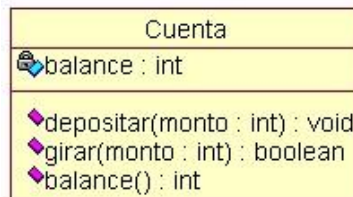
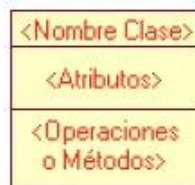
Los **atributos** son valores puros de datos sin identidad. Las asociaciones son conexiones entre objetos con identidad.

Las piezas individuales de comportamiento se describen mediante **operaciones**. Un **método** es la implementación de una operación.

Una clase tiene nombre único dentro de su contenedor, que es generalmente un paquete, pero algunas veces es otra clase.

La clase tiene visibilidad con respecto a su contenedor. La visibilidad especifica como puese ser utilizada por otras clases externas al contenedor.

Una clase tiene **multiplicidad** que especifica cuantas instancias de ella pueden existir. La mayoría de las veces puede ser mucho, pero existen clases unitarias de las que solo existe una instancia.



Un **estereotipo** es un mecanismo de extensibilidad para definir meta-clases. Los tres estereotipos más comunes son objetos entidad, objetos de interface o de frontera, objetos de control.

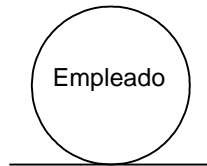
Esterotipo = mecanismo de extensibilidad para definir meta-clases.

- Entidad
- Frontera
- Control

Objetos entidad

Representan algo real o abstracto sobre el cual el sistema necesita almacenar datos. Representan la *memoria esencial* del negocio. Los objetos del negocio (business objects) son normalmente objetos entidad. Ejemplos de objetos entidad pueden ser *empleado*, *alumno*, etc.

Se representan con el estereotipo <<entity>> o se puede utilizar el siguiente ícono:

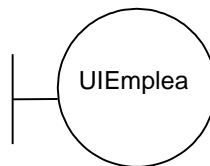


Objetos Interface

Representan los objetos técnicos requeridos para vincular la aplicación con el entorno.

Representan el vínculo a través del cual el sistema recibe o suministra datos e información al entorno. Típicamente incluyen interfaces con el usuario (pantallas, reportes, etc.) e interfaces con otras aplicaciones.

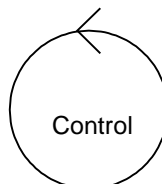
Se representan con el estereotipo <<boundary>> o se puede utilizar el siguiente ícono:



Objetos de control

Contienen comportamiento que no pertenece naturalmente ni a objetos entidad ni de interfaz. Son normalmente objetos transitorios, como ser un controlador de reportes.

Se representan con el estereotipo <<control>> o se puede utilizar el siguiente ícono:



Clases abstractas y concretas

Una clase de la cual pueden generarse instancias particulares (objetos), se denomina clase *concreta*.

Una clase *abstracta* es aquella que no instancias (objetos) propias. Las clases abstractas son un poderoso mecanismo conceptual para definir estructuras comunes de atributos, operaciones, y restricciones, reutilizadas a través de la herencia por múltiples clases concretas.

En el diagrama una clase abstracta se representa con el esterotipo <<abstracta>>.

Operaciones

Operación = servicio ofrecido por un objeto.

Método = implementación de una operación.

Una operación define un servicio ofrecido por un objeto junto con la información que debe suministrarse cuando es invocado (nombre, argumentos de entrada/salida).

También puede contener una especificación de *método*, el cual especifica una implementación para la operación.

Notación: Durante la etapa de Análisis o Diseño lógico pueden utilizarse típicamente texto libre o pseudocódigo. Durante el Diseño físico dichas especificaciones son trasladadas en un lenguaje de programación específico como ser Smalltalk, C++, Java, Visual Basic, etc.

Ejemplos:

```
unProceso comenzar.  
unPedido agregar: unProducto.  
unaVentana abrir.  
unaCuenta extraer: unMonto.
```

Algunas operaciones pueden asumirse que existen para cada clase de objetos sin identificarlas formalmente. Estas son llamadas *operaciones implícitas*. Las operaciones implícitas más importantes son Crear, Destruir, Leer, y Actualizar. Una operación implícita debe ser formalmente definida cuando sus pre y post condiciones no sean triviales.

Un **método** es la implementación de una operación. Se distinguen tres tipos de métodos:

- *Métodos concretos*: tienen una implementación.
- *Métodos abstractos*: deben ser redefinidos en subclases (métodos virtuales).
- *Métodos templates*: son métodos concretos que se implementan en base a métodos abstractos.

Ejemplo de método template:

```
Movil >> desplazarse x: valx y: valy  
    self posicionX: valx.  
    self posicionY: valy.
```

Movil >> posicionX: x.
"abstracto"

Movil >> posicionY: y.
"abstracto"

El uso de métodos abstractos y templates junto con clases abstractas permiten definir estructuras conceptuales, que luego deben especializarse en una estructura de herencia.

Atributos

Atributo = valor que describe un objeto. NO tienen identidad.

Son valores de datos asociados a los objetos de una clase al cual describen.

Están fuertemente asociados con la clase de objetos que los contienen, de tal forma que no tienen existencia independiente o identidad de objeto.

La decisión de cuando un ítem debe considerarse como atributo o como clase varía de sistema en sistema, dependiendo de su semántica en el dominio de problema. Lo que en un sistema se modela como atributo en otro puede modelarse como una clase.

Cada atributo identificado debe ser *atómico* en el sentido de que debe ser tratado como una unidad. En tal caso puede ser un valor simple o grupo de valores tratados siempre como unidad (ej. dirección).

Un atributo puede ser **un tipos de datos** o una **referencia** a otro objeto. En este último caso es la implementación de una relación de conocimiento.

Atributos identificadores: son atributos o grupos de atributos que identifican unívocamente un objeto de una clase. Se corresponde con el concepto de claves del modelo E-R.

Atributos derivados: son atributos cuyo valor puede obtenerse a partir de los valores de otros atributos.

Visibilidad

Define que objetos puede acceder y utilizar los atributos y operaciones de un objeto dado.

- (+) Público : todos
- (#) Protegido : solo desde el objeto mismo y operaciones definidas en subclases
- (-) Privado : solo desde el objeto mismo

Interfaz

Interfaz = descripción de un protocolo de comportamiento que no especifica su implementación.

Una interfaz es una descripción del comportamiento de objetos sin dar su implementación o estado. Una interfaz contiene operaciones pero no atributos y no tiene asociaciones salientes que muestren la visibilidad desde la propia interfaz.

Una o mas clases o componentes pueden *realizar* una interfaz, y cada clase *implementa* las operaciones de la interfaz.

Tipos de datos

Un tipo de dato es la descripción de valores primitivos que *carecen de identidad*.

Un tipo de datos no tiene atributos pero puede tener operaciones. Las operaciones no modifican valores de los datos, sino que pueden devolver valores de datos como resultados.

Niveles de Significado

Las clases pueden existir en varios niveles de significación en un modelo incluyendo los niveles de análisis, diseño, e implementación. (ver libro)

Relaciones

Las relaciones entre clasificadores son:

- Asociación (conocimiento)
- Agregación / Composición
- Generalización
- Dependencia
- Realización
- Uso
- Flujo

<i>Relationship</i>	<i>Function</i>	<i>Notation</i>
association	A description of a connection among instances of classes	_____
dependency	A relationship between two model elements	----->
flow	A relationship between two versions of an object at successive times	----->
generalization	A relationship between a more general description and a more specific variety of the general thing, used for inheritance	----->
realization	Relationship between a specification and its implementation	----->
usage	A situation in which one element requires another for its correct functioning	----->

Asociación

Asociación

- conexión semántica entre instancias de clases.
- proporciona una “conexión” entre los objetos para el envío de mensajes.

Describe **conexiones semánticas** entre los **objetos** individuales de clases dadas.

Las asociaciones proporcionan las conexiones, con las cuales los objetos de diversas clases pueden interactuar (relación de conocimiento).

Las restantes relaciones relacionan clasificadores, no instancias.

Una asociación relaciona una lista ordenada (tupla) de dos o más clasificadores, con las repeticiones permitidas. El tipo mas común de asociación es la binaria entre un par de clasificadores.

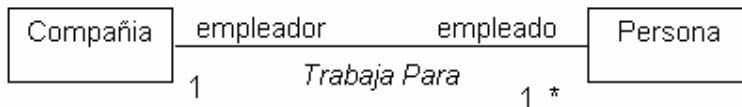
Enlace (link)

- instancia de una asociación.
- es una lista ordenada de referencias a objetos.

Una instancia de una asociación es un **enlace**. Un enlace abarca una **tupla** (lista ordenada) de objetos. Un enlace binario abarca un par de objetos.

Cada conexión de una asociación a una clase se llama **extremo** de la asociación, y allí reside la mayor parte de la información de una asociación.

Los extremos de la asociación pueden tener **nombres** (nombre de rol) y visibilidad. La propiedad mas importante que tienen es **cardinalidad**. La multiplicidad es más útil para las asociaciones binarias porque para relaciones n-arias se torna muy compleja.



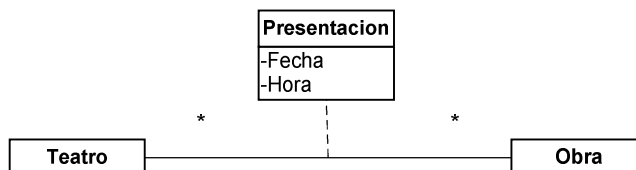
Cardinalidades:

- 0 : opcional
- 1 : uno
- * : muchos
- 0..* : cero a muchos
- 1..* : uno a muchos

Asociación como clase: una asociación puede tener atributos por si mismas, en cuyo caso es una asociación y una clase a la vez.

Ejemplo:

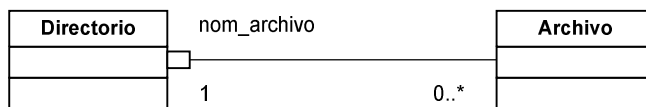
Una Obra de teatro puede presentarse en distintos Teatros en diferentes fechas y horarios.



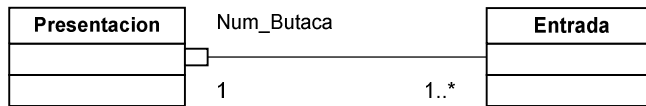
Asociación calificada: Si un atributo de la asociación es único dentro de un conjunto de objetos relacionados, entonces es un *calificador*. Un calificador es un valor que selecciona un objeto único del conjunto de objetos relacionados a través de la asociación. Las tablas de valores y los vectores se pueden modelar como asociaciones calificadas.

Ejemplo:

En un directorio un nombre de archivo es un identificador único ya que no pueden existir dos archivos con el mismo nombre en el mismo directorio (aunque pudieran haber archivos con el mismo nombre pero en distintos directorios). El atributo nombre de archivo de la clase Archivo es un *calificador* de la relación.



Otro ejemplo: una presentación tiene muchas entradas las que se pueden diferenciar por su nro.de butaca cuando la presentación sea con entradas numeradas.

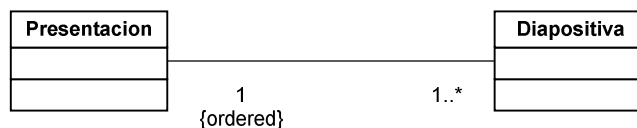


Durante el análisis las asociaciones representan relaciones lógicas entre objetos, no direccionales.

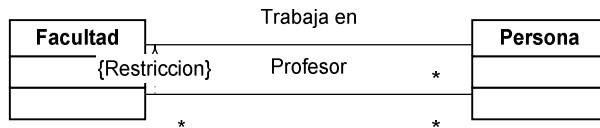
Durante el diseño las asociaciones capturan las decisiones de diseño sobre la estructura de datos así como la separación de responsabilidades entre clases.

Asociación Ordenada: En una asociación puede requerirse que uno de los extremos presente algún ordenamiento particular.

Ejemplo: una presentación requiere que las diapositivas que la componen se presente en forma ordenada.



Restricción: Entre dos clases puede existir más de una asociación. Una de ellas puede ser una restricción de la otra.



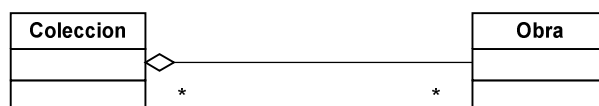
Agregación y Composición: Una agregación es una relación todo-partes donde los objetos de una clase son compuestos por objetos de otra. Por ejemplo un automovil está compuesto por carrocería, motor, ruedas, etc.

Una composición es una forma de asociación más fuerte en la cual el compuesto es responsable de gestionar sus partes, por ejemplo asignacion y desasignacion.

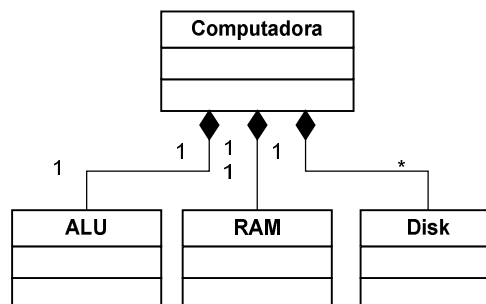
La composición implica tres cosas

- Dependencia existencial. El elemento dependiente desaparece al destruirse el que lo contiene y, si es de cardinalidad 1, es creado al mismo tiempo.
- Hay una pertenencia fuerte. Se puede decir que el objeto contenido es parte constitutiva y vital del que lo contiene
- Los objetos contenidos no son compartidos, esto es, no hacen parte del estado de otro objeto.

Ejemplo: una Colección de obras musicales puede contener diversas obras, pero la misma obra puede pertenecer a más de una colección.



Ejemplo: una Computadora es un ensamble (composición) de partes.



Los conceptos mas habituales que se pueden modelar como agregaciones son:

- Ensamble de partes (ej. una bicicleta). Generalmente un ensamble es una composición.
- Colecciones (ej. un club es una colección de miembros.)
- Contenedor contenidos (ej. un aula contiene bancos, pizarron, etc)

Generalizacion

Propósitos de la generalización:

1. *Principio de sustitución de Liskov*: Operaciones polimórficas. Una instancia de un descendiente se puede utilizar donde quiera que este declarado el antecesor.
2. *Herencia*: Permitir la descripción incremental de un elemento que comparte descripciones con sus antecesores. Esto se llama Herencia.

Herencia

La herencia es el mecanismo a través del cual los atributos, operaciones, y restricciones definidas para una clase, denominada *superclase*, pueden ser heredados (reutilizados) por otras clase denominadas *subclases*.

La herencia utiliza el principio “*es un tipo de ...*”.

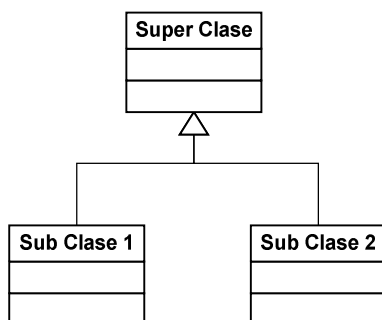
Una subclase puede redefinir las operaciones heredadas. Adicionalmente, una subclase puede definir nuevos atributos, operaciones, y restricciones.

Se distingue entre herencia simple y múltiple. La herencia simple se da cuando un subclase hereda de una única superclase. En el caso de la herencia múltiple, una subclase puede heredar de varias superclases. La decisión de soportar herencia simple o múltiple ha dado lugar a largos debates conceptuales y metodológicos.

Herencia de atributos: no permite redefinición

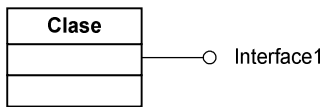
De operaciones: puede ser redefinida en los descendientes.

La representación gráfica en UML es la siguiente:



Realización

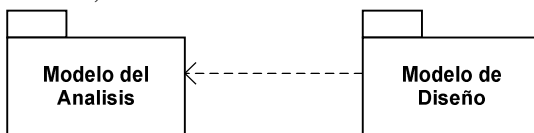
La relación de realización conecta un elemento del modelo, tal como una clase, con otro elemento, tal como una interfaz, que especifica su comportamiento pero no su estructura o implementación.



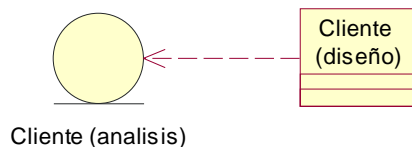
Dependencias

Una dependencia indica una relación semántica entre dos o más elementos del modelo. Indica una situación, en la cual un cambio al elemento proveedor puede requerir un cambio o indicar un cambio en el significado del elemento cliente en la dependencia.

Dependencia de traza: una traza es una conexión conceptual entre elementos de diversos modelos, a menudo modelados en diferentes etapas del desarrollo, que carece de semántica detallada. Se utiliza generalmente para seguir la pista de requisitos del sistema, a través de los modelos



Dependencia de refinamiento: un refinamiento es una relación entre dos versiones de un concepto en diversas etapas del desarrollo o en diversos niveles de abstracción.



Dependencia de derivación: una dependencia de derivación indica que un elemento se puede computar a partir de otro elemento.

La derivación, la realización, el refinamiento, y la traza, son dependencias de abstracción; relacionan dos versiones de la misma cosa subyacente.

Dependencia de Uso: una dependencia de uso es una declaración de que el comportamiento o la implementación de un elemento, afectan al comportamiento o a la implementación de otro elemento.

Dependencia de Acceso: Permiso para un paquete para acceder a los contenidos de otro paquete.

Dependencia de Importación: Permiso para un paquete para acceder a los contenidos de otro paquete y añadir alias de sus nombres en el espacio de nombres del importador.

Dependencia de Amistad: es una dependencia de acceso, que permite que el cliente vea incluso el contenido privado del proveedor.

Dependencia de Ligadura: una ligadura es una asignación de valores a los parámetros de una plantilla.

Restricción

Una restricción es una expresión booleana representada como una cadena interpretable en un determinado lenguaje. Para expresar restricciones se puede utilizar el lenguaje natural, notación de teoría de conjuntos, lenguajes de restricciones o varios lenguajes de programación.

UML incluye OCL para expresar restricciones.

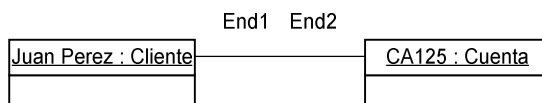
Instancias

Una instancia es una entidad de ejecución con identidad, es decir, algo que puede ser distinguido de las otras entidades en ejecución.

Un **objeto** es una instancia de una clase.

Un **enlace** es una instancia de una asociación.

Diagrama de Objetos



3. La Vista de Casos de Uso

Los modelos de objetos son apropiados para representar la estructura de los objetos, sus asociaciones, y como interactúan dinámicamente. Sin embargo no son modelos apropiados para capturar y comunicar requerimientos funcionales de la aplicación, ni para verificar si el modelo de objetos se corresponde con la realidad.

El primer modelo del sistema que se debe construir debe ser comprensible tanto para los usuarios como para los desarrolladores.

Los modelos de objetos son muy complejos para este propósito. Un sistema real, aunque pequeño, puede consistir de alrededor de 100 objetos. Sistemas más grandes pueden implicar varios cientos o miles de objetos.

Por lo tanto, el primer modelo del sistema no debe ser un modelo de objetos. Debe ser un modelo que describa el sistema, su entorno, y como se relaciona con el mismo.

En otras palabras debe describir el sistema tal como se lo ve desde el exterior, con una vista de 'caja negra'. Los *casos de uso* son una forma de especificar esta vista de caja negra.

Un *caso de uso* es una forma de usar el sistema. El usuario interactúa con el sistema interactuando con sus casos de uso.

La vista de casos de uso captura el comportamiento del sistema, de un subsistema, o de una clase, tal como se muestra a un usuario desde el exterior.

Particiona la funcionalidad del sistema en transacciones significativas para los actores-usuarios del sistema.

Un caso de uso describe una interacción como secuencia de mensajes entre el sistema y uno o más actores.

Actor

Un actor es una idealización de una persona externa, de un proceso, o de una cosa que interactúa con el sistema.

Los actores son objetos que residen *fuera* del sistema, en tanto que los casos de uso son objetos que residen *dentro* del sistema.

Un actor participa en uno o más casos de uso.

Un actor puede ser caracterizado por un conjunto de atributos que caracterizan su estado.

Los actores pueden ser definidos en jerarquías de generalización.

Un actor puede ser una persona, otro sistema informático, o un proceso.



Diferencia entre actor y usuario

Un usuario es una persona (humana) que usa el sistema de algún modo, en tanto que un actor representa un rol específico que un usuario puede ejercer.

Un usuario puede actuar como instancias de diferentes actores.

Los actores son instancias de una clase. Los usuarios son un tipo de “recurso” que implementan dichas instancias.

Ejemplos de actores

En un sistema ATM (Cajero automático) podemos identificar tres tipos de actores:

- Cliente del banco (actor humano)
- Operador del CA (actor humano)
- Sistema del banco (sistema externo)

Una misma persona (usuario) puede desempeñarse como instancia de actor Cliente y como instancia de actor Operador.

Caso de uso

Un caso de uso es una *secuencia de transacciones* realizadas por el sistema que brinda un resultado de valor a un actor en particular.

Un caso de uso es una unidad coherente de funcionalidad, externamente visible, proporcionada por una unidad del sistema y expresada por secuencias de mensajes intercambiados por el sistema y uno o más actores.

El propósito del caso de uso es definir una pieza de comportamiento coherente, sin revelar la estructura interna del sistema.

Los casos de uso cumplen dos funciones importantes:

1) Capturan requerimientos funcionales del sistema:

El modelo de casos de uso define el comportamiento del sistema a través de un conjunto de casos de uso. El *entorno* del sistema es descrito por un conjunto de actores que usan el sistema a través de los casos de uso.

El modelo de casos de uso es una vista *externa* del sistema, en tanto que los modelos de objetos son vistas *internas* del mismo.

2) Estructuran los modelos de objetos en vistas manejables:

En orden de manejar la complejidad de un sistema real, es práctico contruir modelos de objetos para cada caso de uso con los objetos que participan en dicho caso de uso.

Un objeto puede participar en varios casos de uso. Esto significa que el modelo de objetos completo se obtiene a partir de un conjunto de vistas de modelos de objetos, uno por cada caso de uso.

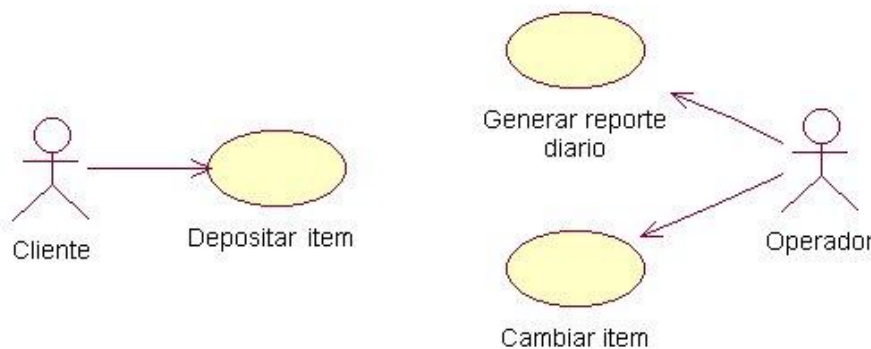
Pueden definirse todas las responsabilidades de un objeto viendo los roles en que participa en todos los casos de uso.

Un caso de uso es especificado por interacciones de UML, y representadas en diagramas de secuencia, diagramas de colaboración, descripciones de texto informales.

Cuando se implementan, los casos de uso son *realizados* mediante colaboraciones entre clases del sistema. Una clase puede participar en múltiples colaboraciones y por ende en múltiples casos de uso.

Una *colaboración* es una realización de un caso de uso.

Los casos de uso también se pueden aplicar internamente a unidades más pequeñas de un sistema, tales como subsistemas y clases individuales. Un caso interno de uso representa el comportamiento que una parte del sistema presenta al resto del sistema.



Modelado de casos de uso

El modelado de casos de uso es una actividad que se realiza en conjunto con el diseño de la interfaz de usuario, donde participa activamente el usuario quien es el centro de interés. Los usuarios son entrevistados para describir diferentes escenarios de uso (instancias de casos de uso).

A medida que se tiene una mejor comprensión de las necesidades del usuario, los bocetos de la interfaz avanzan. Una técnica útil para esto es la realización de prototipos.

Como identificar y definir secuencias de transacciones?

1) Identificación a través de actores

1.1 Identificar los actores que se comunicarán con el sistema.

1.2 Para cada actor considerar:

1.2.1 Cuales son las principales tareas del actor

1.2.2 Qué accesos (lectura o escritura) requiere el actor del sistema

1.2.3 Cuando el actor informará al sistema acerca de cambios fuera del sistema

1.2.4 Cuando el actor será informado de cambios a través del sistema

2) Identificación a través de eventos

Consiste en identificar a que eventos externos o temporales debe ser capaz de responder el sistema:

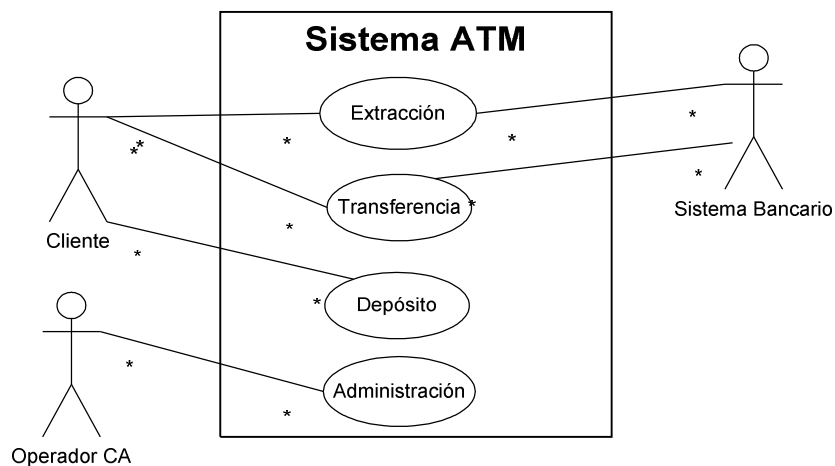
2.1 Confeccionar la lista de eventos.

2.2 Asociar una secuencia de transacciones para cada evento identificada.

Herramientas de modelado de casos de uso

- 1) Diagrama de casos de uso
- 2) Descripción textual
 - *Camino estándar*: es una descripción secuencial de todas las actividades que deben realizarse en forma normal. No se describe el proceso de excepciones.
 - *Caminos alternativos*: Describen casos inusuales de procesamiento y manejos de excepciones o errores.

Ejemplo: ATM



Caso de Uso: Extracción

Camino Estándar

- Un mensaje de bienvenida está en espera en la pantalla del CA.
- El cliente inserta su tarjeta en el CA.
- El CA lee el código de la banda magnética y verifica que sea aceptable.
- Si la tarjeta es aceptable, el CA solicita al cliente su código PIN.

Esperando código PIN:

- El cliente ingresa su código PIN.
- Si el código PIN es correcto, el CA solicita al cliente el tipo de transacción a realizar.

Esperando tipo de transacción:

- El cliente selecciona <extracción> y el CA envía el código PIN al Sistema bancario solicitando los datos de la cuenta del cliente.
- Los datos de la cuenta recibidos se despliegan en la pantalla.

Esperando decisión del cliente:

- El cliente selecciona una cuenta y el monto a extraer.
- El CA envía al sistema bancario el requerimiento de extracción.
- El CA preparan los billetes a ser dispensados.
- El CA imprime el comprobante del movimiento.
- Los billetes son dispensados al cliente.

Caminos Alternativos

La tarjeta no es aceptable:

- Si la tarjeta no es aceptable debido a un error en la banda magnética, la tarjeta es ejetada con un 'bip'.

Código PIN incorrecto:

- Si el código PIN es incorrecto, un mensaje de error se desplegará al cliente, dando la opción de corregir.

Extracción no permitida:

- Si el sistema bancario no acepta la extracción, un mensaje es desplegado al cliente y la tarjeta es ejetada.

Cancelación:

- El cliente puede cancelar la transacción en cualquier momento presionando el boton cancel. Si esto ocurre el CA ejeta la tarjeta y termina la transacción.

Relaciones entre casos de uso

Inclusión

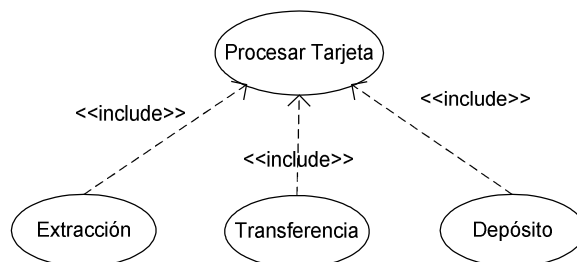
Un caso de uso incluir en su comportamiento el comportamiento de otro caso de uso base a traves de una relación de inclusión.

Cuando varios casos de uso comparten descripciones similares en orden de evitar redundancia y maximizar reutilización se puede extraer dichas *secuencias comunes*.

La secuencia común es un caso de uso *abstracto* y no puede instanciarse por si mismo.

Los casos de uso que *incluyen* la secuencia común son casos de uso *concretos* y pueden crearse instancias de ellos.

La relación de inclusión es una relación de dependencia entre casos de uso. La relación de inclusión reemplazó a la relación de uso que se usaba originalmente en estos casos.



Extensión

Un caso de uso se puede definir como una extensión incremental de un caso de uso base.

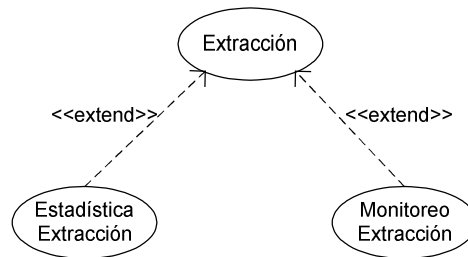
Un caso de uso se puede definir como una extensión incremental de un caso de uso base.

Siempre se instancia un caso de uso base, en tanto las extensiones solo son instanciados bajo determinadas condiciones. No se genera una instancia de una extensión.

Su usa la relación de extensión para:

- Partes opcionales de un caso de uso.
- Cursos complejos y alternativos.
- Subsecuencias que se ejecutan solo bajo ciertas condiciones.

La relación de extensión permite un desarrollo incremental, comenzando el desarrollo con casos de uso más simples e ir agregando comportamientos específicos (extensiones a los caso de uso base).



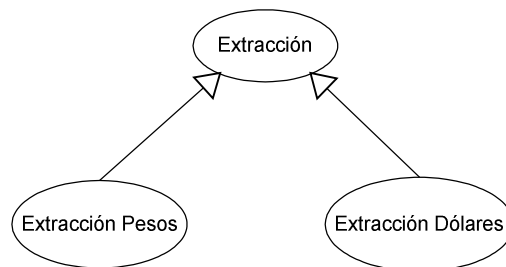
Inclusión vs. Extensión

En líneas generales debe aplicarse la siguiente regla:

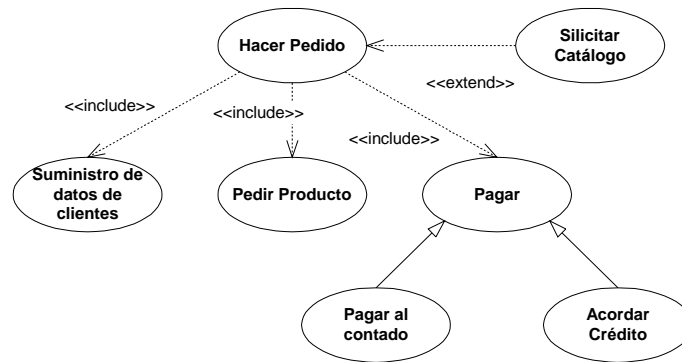
Secuencias comunes	→	<<inclusión>>
Nuevos cursos/opcionales	→	<<extensión>>

Generalización

Un caso de uso se puede especializar en uno o más casos de uso hijos, utilizando una relación de generalización.



Otro ejemplo:



4. La Vista de la Máquina de Estados

La vista de la máquina de estados describe el comportamiento dinámico de los objetos, en un cierto plazo, modelando los ciclos de vida de los objetos de cada clase.

Cada objeto se trata como una entidad aislada que se comunica con el resto del mundo detectando eventos y respondiendo a ellos.

Máquina de estados

Una máquina de estados se representa por un diagrama de estados y de transiciones. Una máquina de estados se une a una clase y describe la respuesta de una instancia de la clase a los eventos que recibe.

Una máquina de estados es una vista localizada de un objeto, una vista que lo separa del resto del mundo.

Es una buena manera de especificar un comportamiento exacto de los objetos de una clase, pero no es una buena manera de entender el funcionamiento total de un sistema.

Para una idea más amplia de los efectos del comportamiento de un sistema, son más útiles las vista de interacción.

Evento

Un evento es una ocurrencia significativa que tiene una localización en tiempo y espacio. Ocurre en un punto del tiempo y no tiene duración.

Cuando utilizamos la palabra evento por sí mismo, queremos decir generalmente que es una descripción de todas las ocurrencias individuales del evento que tienen la misma forma general, del mismo modo que la palabra clase expresa todos los objetos individuales, que tienen la misma estructura. Una ocurrencia específica de un evento se llama *instancia del evento*.

Los eventos pueden tener parámetros que caracterizan cada instancia del evento.

Los eventos se pueden organizar en jerarquías de generalización para compartir la estructura común.

Los eventos se clasifican en varios tipos explícitos e implícitos:

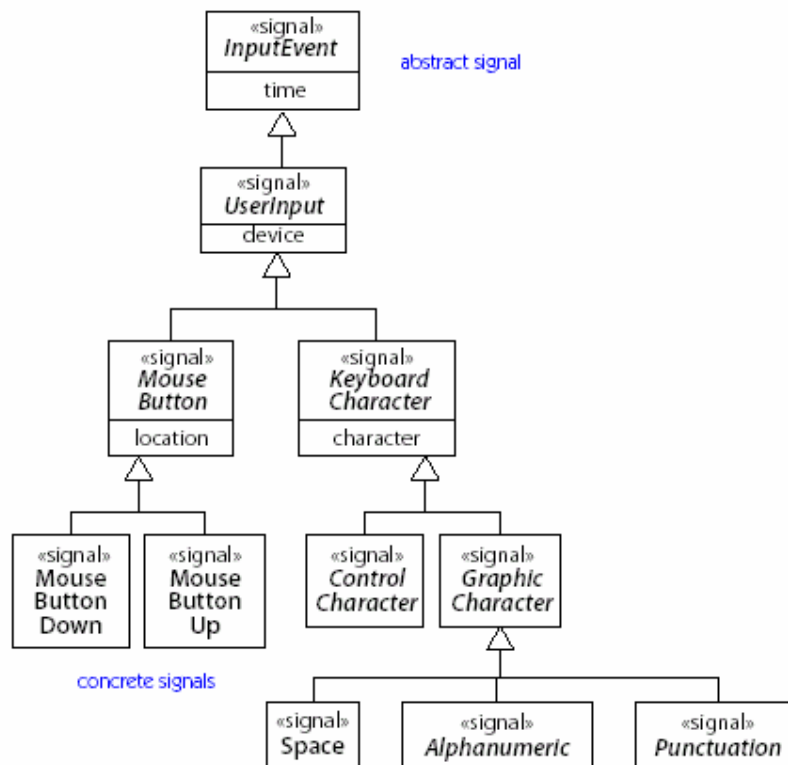
- eventos de señal
- eventos de llamada
- eventos de cambio
- eventos de tiempo

Evento de señal

Una señal es una entidad con nombre, que se piensa explícitamente como vehículo de *comunicación entre dos objetos*. La recepción de una señal es un evento para el objeto receptor.

Las señales incorporan la comunicación unidireccional asíncrona.

Las señales se pueden modelar en diagramas de clases como clasificadores utilizando el estereotipo <<señal/signal>>. Los parámetros de la señal se declaran como atributos.



Evento de llamada

Un evento de llamada es la recepción de una llamada por un objeto que elige poner una operación en ejecución, como transición de la máquina de estados.

Para el objeto llamador, un evento de llamada es indistinta de una llamada a método ordinaria.

El objeto receptor, elige si la operación será implementa como un método o como un disparador de un evento de llamada en la máquina de estados. Los parámetros de la operación son los parámetros del evento.

Evento de cambio

Un evento de cambio es la satisfacción de una expresión booleana que dependa de ciertos valores de un atributo.

Es una manera declarativa de esperar a que una condición esté satisfecha.

La diferencia entre una condición de guarda y un evento de cambio es que una condición de guarda se evalúa una vez cuando ocurre el evento disparador de la transición. Si es falsa la transición no se activa. Un evento de cambio se evalúa continuamente hasta que llega a ser verdad, en cuyo caso se dispara la transición.

Evento de tiempo

Los eventos de tiempo representan el paso del tiempo. Un evento de tiempo se puede especificar de modo absoluto (hora) o de modo relativo (tiempo transcurrido desde).

Estado

Un estado describe un período de tiempo durante la vida de un objeto de una clase. Puede ser caracterizado de tres formas:

- conjunto de valores de objeto cualitativamente similares (atributos, relaciones)
- período de tiempo durante el cual el objeto espera que ocurra un evento
- período de tiempo durante el cual el objeto realiza alguna actividad

Transición

Cambio de un estado a otro. Toda transición tiene un evento disparador, una condición de guarda, una acción, y un estado destino.

Existen dos tipos de transiciones: transición externa, y transición interna.

Transición externa

Una transición externa es una transición que cambia el estado activo.

Evento disparador

El disparador es un evento cuya ocurrencia permite la transición.

El evento puede tener parámetros que estarán disponibles para una acción en la transición.

Si una señal tiene descendientes, cualquier descendiente de la señal permite la transición.

Los eventos se manejan uno solo a la vez.

Condición de guarda

Es una condición booleana referida a los atributos del objeto o a los parámetros del evento disparador, que se evalúa cuando ocurre el evento disparador. Si la expresión se evalúa como cierta entonces se dispara la transición.

Transición de finalización

Es una transición que carece de evento disparador explícito. Es accionada por la terminación de la actividad del estado del que sale.

Puede tener una condición de guarda que se evalúa en el momento en que termina la actividad.

Acción

Cuando se dispara una transición, su acción (si la hay) es ejecutada. Una acción es un cómputo atómico y breve. A menudo es:

- una sentencia de asignación
- una operación aritmética
- el envío de una señal a otro objeto
- la invocación de una operación propia
- asignación de valores de retorno
- creación o destrucción de objetos
- una secuencia de acciones simples

Cambio de estado

Cuando se completa la ejecución de la acción, el estado destino de la transición pasa a ser el estado activo.

Estados anidados

Los estados se puede anidar dentro de otros estados compuestos.

Una transición que deja el estado mas externo es aplicable a todos los estados internos.

Acciones de entrada y salida

Un estado puede tener acciones que se realicen siempre que se entre o se salga del estado.

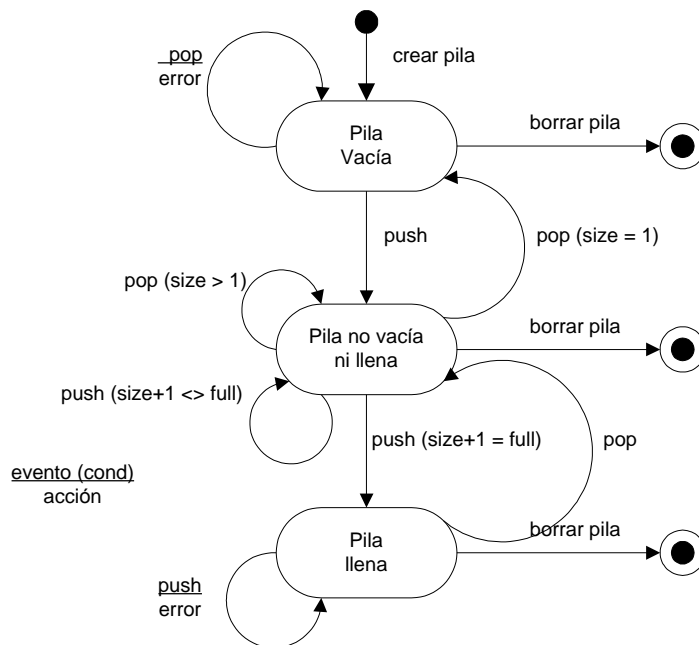
Si la transición sale del estado original, entonces su acción de salida se ejecuta antes de la acción de la transición y de la acción de entrada en el estado nuevo.

Transición interna

Una transición interna tiene un estado origen pero ningún estado destino.

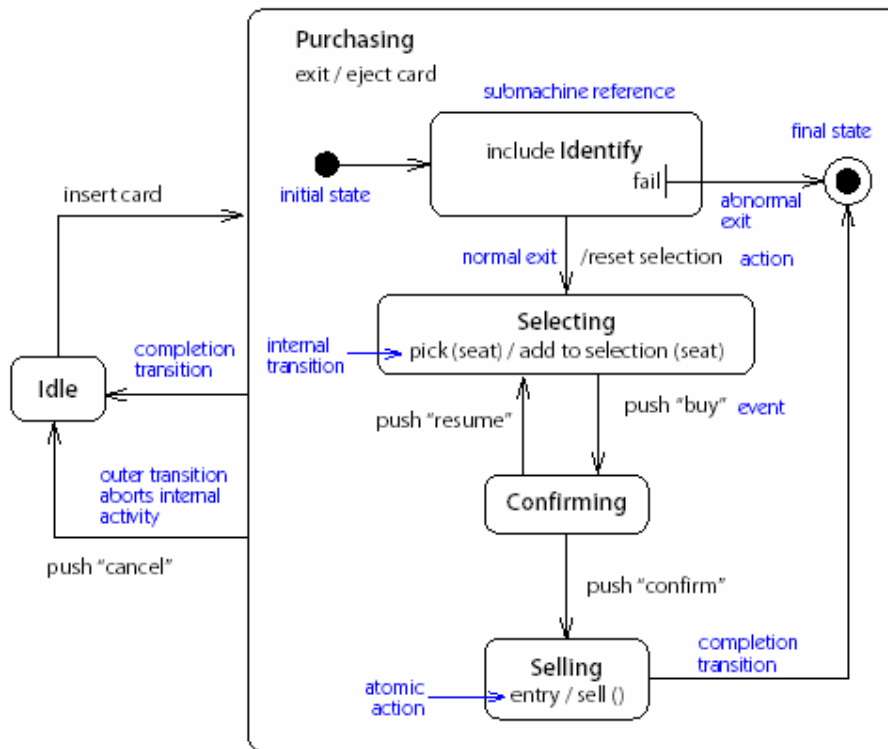
Si una transición interna tiene acción, se ejecuta pero no existe cambio de estado.

Ejemplo: Pila



Estados Compuestos

Un estado compuesto es un estado que se ha descompuesto en subestados secuenciales o concurrentes.



5. La Vista de Actividades

Un grafo de actividades es una forma especial de máquina de estados, prevista para modelar cálculos y *flujo de trabajos*. Los estados del grafo de actividades representan los estados de ejecución del cálculo, no los estados de un objeto.

Diagrama de actividades

Estado de actividad: Se representa como una caja con los extremos redondeados y que contiene una descripción de la actividad.

Transiciones simples: De cada estado de actividad se sale con una transición simple de terminación, representada como una flecha. Las ramas se muestran como condiciones de guarda en transiciones o como diamantes con múltiples flechas de salida etiquetadas.

División/Unión del control: se representan con barras gruesas a donde llegan o salen flechas.

Eventos externos: se pueden incluir eventos externos que disparen transiciones. Sin embargo si hay muchas transiciones dirigidas por eventos externos puede ser preferible utilizar una máquina de estados convencional.

Calles: se pueden agrupar todas las actividades manejadas por una organización del negocio en una pista o calle del grafo.

Flujo de objetos: un diagrama de actividades puede mostrar el flujo de objetos como entrada o salida de una actividad. En tal caso se dibuja el objeto, pudiéndose indicar el estado para representar su evolución. Para un valor de salida se dibuja una flecha con línea discontinua desde la actividad al objeto. Para un valor de entrada se dibuja una flecha continua desde el objeto a la actividad.

Actividades y otras vistas

Un grafo de actividades no es completo. Muestra una vista parcial de un cálculo o proceso. Debe ser ampliado en una colaboración donde se muestren todas las operaciones atómicas y las clases que las implementan.

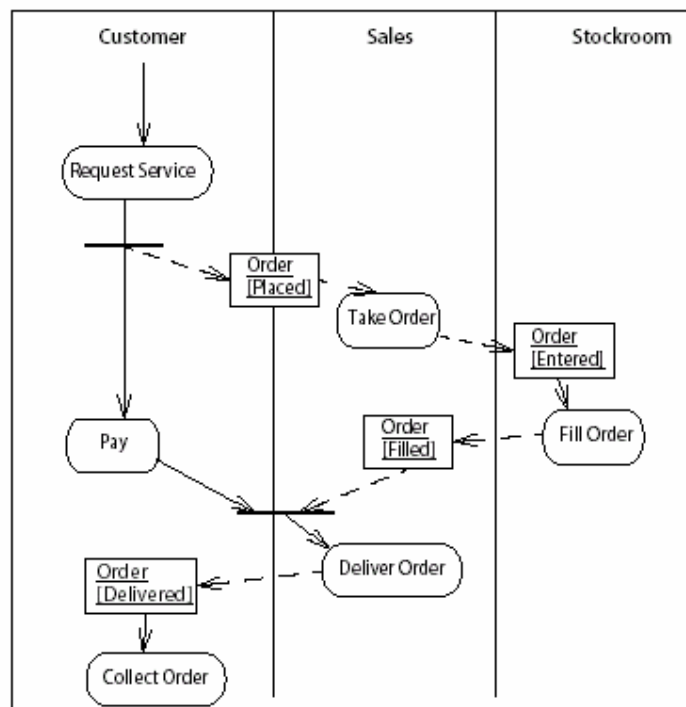
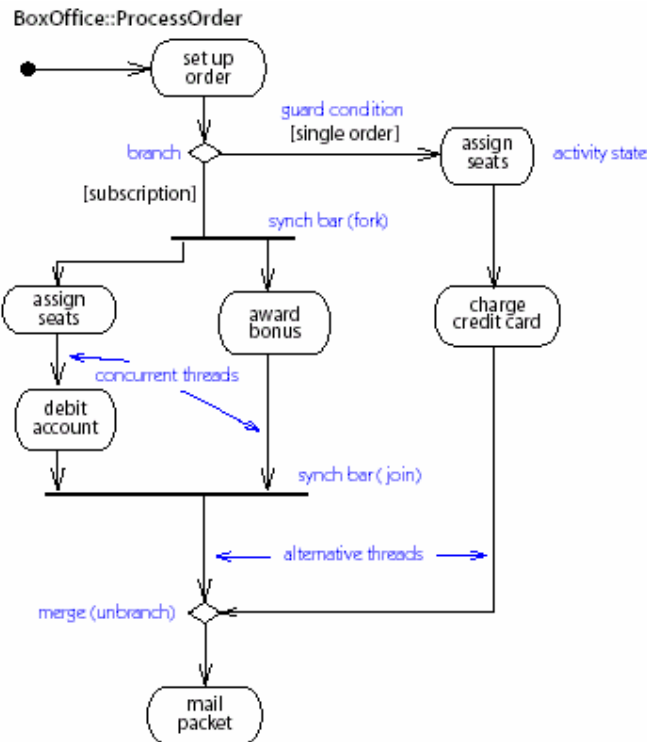


Figure 7-2. Swimlanes and object flows

6. La Vista de Interacción

Los objetos interactúan para implementar el comportamiento del sistema. Esta interacción puede ser descrita en dos formas complementarias, una centrada en objetos individuales (máquina de estados) y otra en una colección de objetos cooperantes.

La máquina de estados es una visión estrecha y profunda que se enfoca en cada objeto individualmente. Es difícil comprender el funcionamiento del

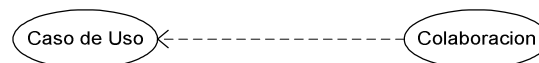
Colaboración

Una colaboración es una descripción de una colección de objetos que interactúan para implementar un comportamiento en un contexto dado. Describe una sociedad de objetos cooperantes ensamblados para llevar adelante algún propósito.

Cada colaboración contiene ranuras que son ocupadas por objetos y links en tiempo de ejecución. Cada objeto y/o link cumple un *rol* dentro de una colaboración.

Un objeto puede participar en más de una colaboración.

Una colaboración implementa la funcionalidad de un caso de uso dado a través de una dependencia de realización.



El flujo de mensajes dentro de una colaboración puede opcionalmente ser especificado por una *máquina de estados*, la cual especifica las secuencias de eventos legales. Los eventos en la máquina de estados representan los mensajes intercambiados entre los roles dentro de la colaboración.

Una colaboración tiene un aspecto *estático* y un aspecto *dinámico*. El aspecto estático es similar a la vista estática, contiene un conjunto de roles y sus relaciones que definen el contexto para el aspecto dinámico.

El aspecto dinámico es el conjunto de mensajes intercambiados entre los objetos que ocupan los roles. Tal intercambio de mensajes en una colaboración es llamada una *interacción*. Una colaboración puede incluir una o más interacciones.

Las interacciones son representadas por diagramas de *secuencia* o diagramas de *colaboración*.

Interacción

Una interacción es un conjunto de mensajes que se intercambian dentro del contexto de una colaboración por roles de clasificadores (objetos) a través de enlaces (instancias de asociación).

Cuando una colaboración existe en tiempo de ejecución, objetos ocupan los roles de clasificadores e intercambian instancias de mensajes a través de enlaces (links).

Un mensaje es una comunicación unidireccional entre objetos, un flujo de control con información desde un emisor a un receptor. Un mensaje puede tener parámetros. Puede ser una señal (asíncrono) o una llamada (síncrono).

Los mensajes pueden agruparse en *hilos de control* secuenciales. Hilos separados representan conjuntos de mensajes concurrentes.

La sincronización entre hilos es modelada por restricciones entre mensajes en diferentes hilos. Una construcción de sincronización puede modelar una bifurcación (fork) del control, una reunión (join) del control, una ramificación del control.

La secuencia de mensajes puede representarse en dos tipos de diagramas: *diagramas de secuencia* (se enfocan en la relación de tiempo) y *diagramas de colaboración* (se enfocan en las relaciones entre objetos).

Diagrama de secuencia

Es un gráfico bidimensional. La dimensión vertical es el eje de tiempo. La dimensión horizontal muestra los roles de clasificadores que representan objetos individuales en la colaboración. Cada rol de clasificador es representada por una línea vertical que representa su línea de vida. Una línea punteada representa el período de ‘existencia’ del objeto. Una línea doble representa una activación del objeto.

Un mensaje es representado por una flecha desde la línea de vida de un objeto hacia la línea de vida de otro objeto. La secuencia de mensajes está ordenada en forma descendente en el diagrama.

Activación: una activación es la ejecución de una operación. Es representada por una línea de doble trazo sobre la línea de vida del objeto que recibe el mensaje.

Un *objeto activo* es un objeto que mantiene la pila de activaciones. Cada objeto activo tiene su propio hilo de control que se ejecuta en paralelo con otros objetos. Los objetos que son llamados por los objetos activos se denominan *objetos pasivos*. Estos reciben el control solo cuando reciben un mensaje solicitando una operación.

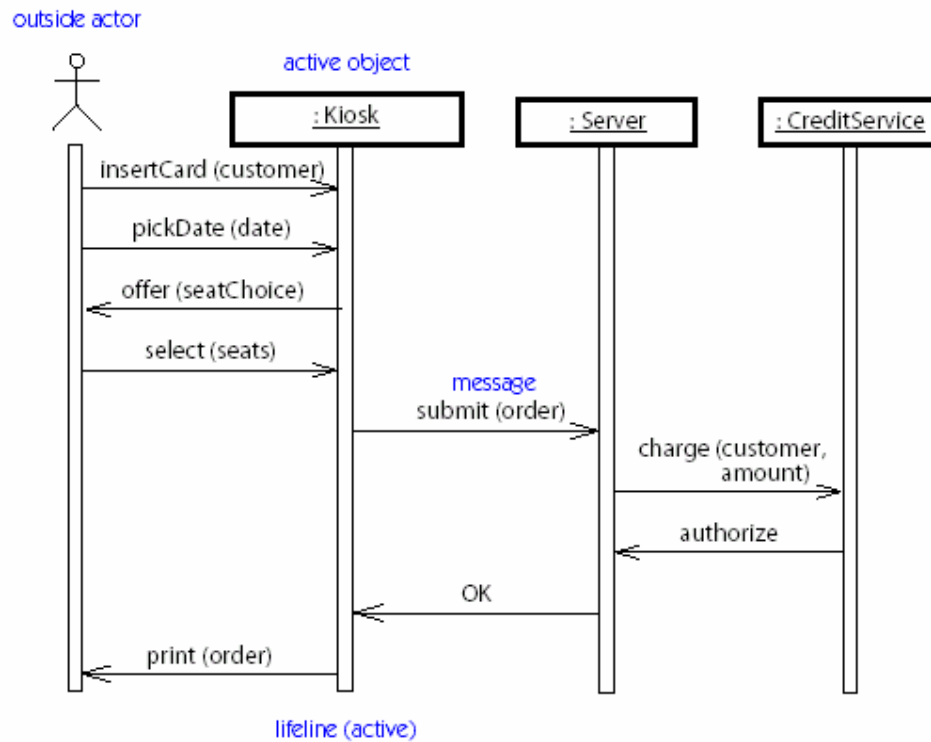


Figure 8-1. *Sequence diagram*

Ejemplo para diagrama de secuencia para el caso de uso “Extracción” del Cajero Automático:

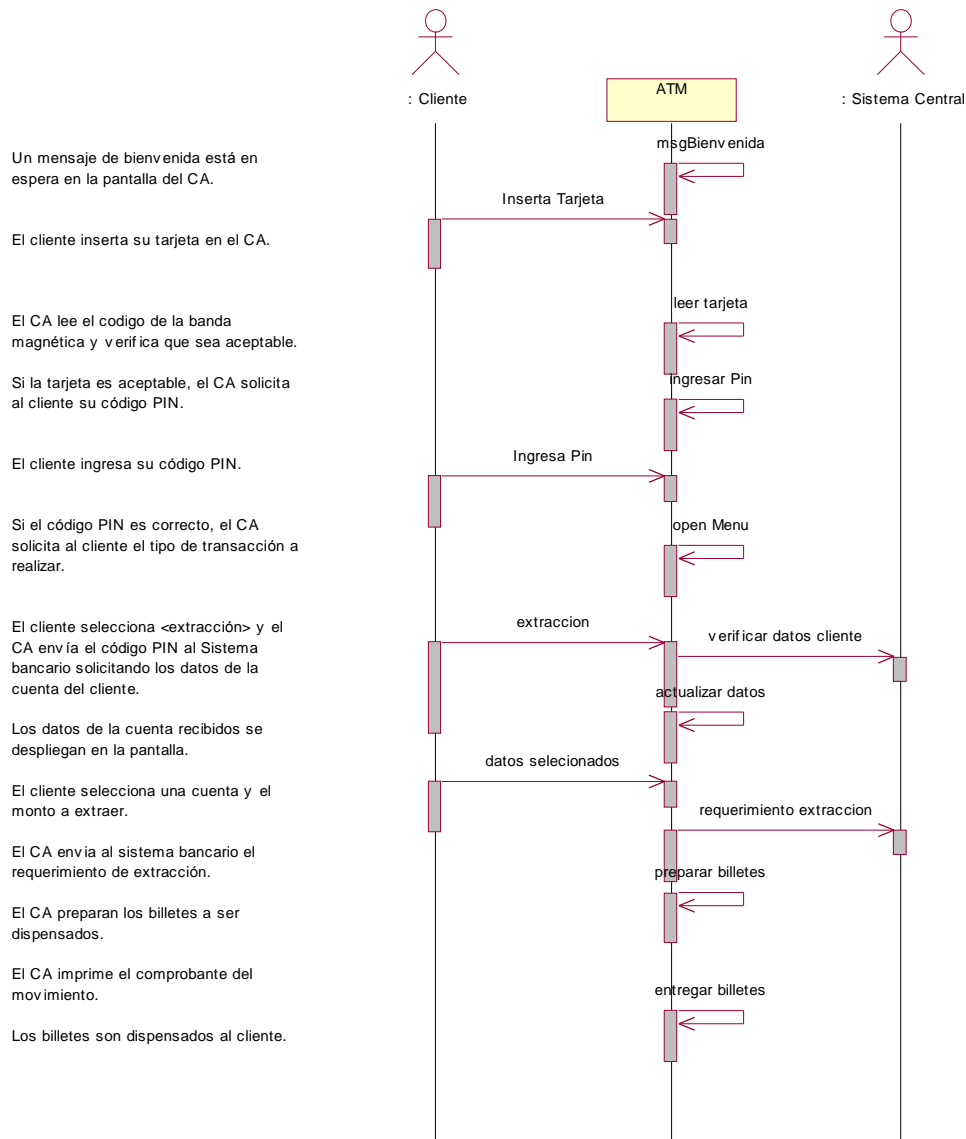


Diagrama de colaboración

Un diagrama de colaboración es un diagrama de clases que contiene *roles de clasificador* y *roles de asociación* (instancias de clases). Los roles de clasificación y los roles de asociación describen la configuración de objetos y links que ocurren cuando

una instancia de colaboración es ejecutada. Cuando una colaboración es instanciada, los objetos son ligados a roles de clasificador y los links son ligados a roles de asociación.

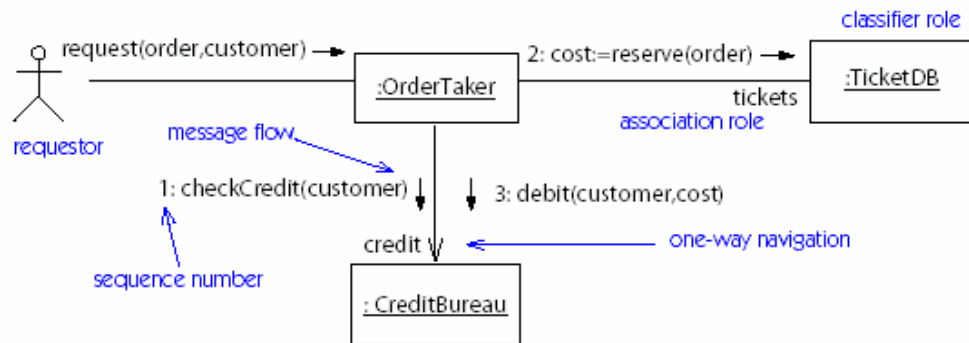
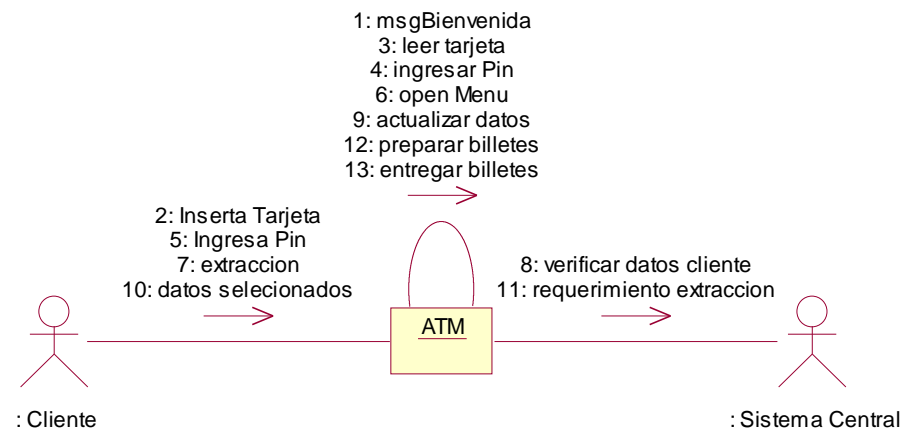


Figure 8-3. Collaboration diagram

Ejemplo de diagrama de colaboración para el caso de uso extraccion



Es útil agrupar los objetos en una de cuatro categorías:

- objetos que existe durante toda la interacción
- objetos creados durante la interacción (constrain{new})
- objetos destruidos durante la interacción (constrain{destroyed})
- objetos creados y destruidos durante la interacción (constrain{transient})

Durante el diseño esto ayuda a determinar como debe fluir el flujo de control dentro de la interacción.

Mensaje

Los mensajes son representados con flechas etiquetadas vinculadas a los links. Cada mensaje tiene:

- un nro.de secuencia

- un nombre y una lista de argumentos
- una lista de mensajes predecesores (opcional)
- una condición de guarda (opcional)
- un nombre de valor de retorno (opcional)

El nro. de secuencia tiene opcionalmente el nro. de hilo de ejecución. Todos los mensajes dentro del mismo hilo de ejecución están numerados secuencialmente.

Pueden agregarse detalles como ser tipo de mensajes (sincrónicos o asincrónicos).

Flujos

Generalmente en un diagrama de colaboración un objeto se representa una sola vez. Sin embargo, si un objeto tiene distintos estados que se deban hacer explicitos (cambio de localización, o cambio de asociaciones) el objeto se puede representar mas de una vez vinculando los diferentes símbolos con un flujo etiquetado <<become>> o <<conversión>>. Un flujo <<become>> es una transición de un estado a otro de un objeto. Se dibuja como una flecha con línea discontinua con el estereotipo <<become>>.

También se puede utilizar el estereotipo <<copy>> o <<copia>> que representa una copia de un objeto que a partir de dicho momento es independiente.

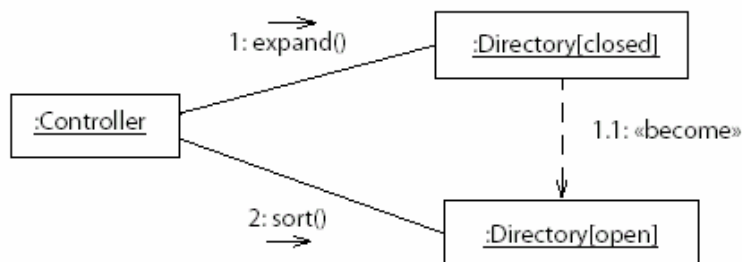


Figure 8-4. Become flow

Diagramas de Secuencia Vs. Diagramas de Colaboración

Los diagramas de secuencia y de colaboración muestran las mismas interacciones pero enfatizan distintos aspectos. Los diagramas de secuencia enfatizan la secuencia en el tiempo de los mensajes intercambiados pero no ponen de relieve las relaciones entre los objetos. Los diagramas de colaboración muestran claramente las relaciones, pero la secuencia de tiempo debe tomarse de los diagramas de secuencia.

Patrones

Un patrón es una colaboración parametrizada junto con las pautas sobre cuándo utilizarlo.

Un parámetro se puede sustituir por diversos valores para producir distintas colaboraciones.

El uso de un patrón se representa como una elipse de línea discontinua conectada con cada una de sus clases por una línea discontinua, que se etiqueta con el nombre de rol.

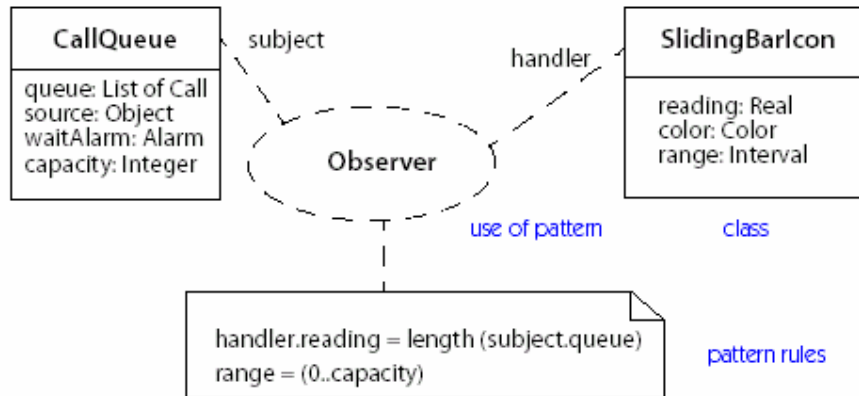


Figure 8-5. Pattern usage

7. Las Vistas Físicas

Descripción

Importancia -> propósitos de reutilización y de rendimiento

UML incluye dos tipos de vistas para representar *unidades de implementación*:

- la vista de implementación
- la vista de despliegue

La vista de **implementación** muestra el *empaquetado físico* de las partes reutilizables del sistema en unidades sustituibles llamadas **componentes**.

La vista de **despliegue** muestra la *disposición física* de los recursos de ejecución computacional, tales como computadores y sus interconexiones. Se llaman **nodos**. Durante la ejecución los nodos pueden contener componentes y objetos.

Componente

Def.: Un componente es una unidad física de implementación con interfaces bien definidas pensada para ser utilizada como parte reemplazable de un sistema.

Cada componente incorpora la implementación de ciertas clases del diseño del sistema. Los componentes bien diseñados no dependen directamente de otros componentes sino de sus interfaces (bajo acoplamiento).

El uso de interfaces permite evitar la dependencia directa entre componentes.

La vista de componentes muestra la red de dependencias entre componentes.

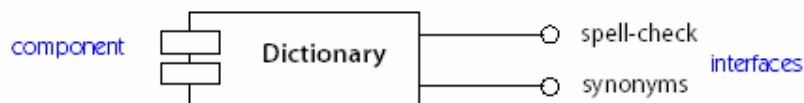


Figure 9-1. Component with interfaces

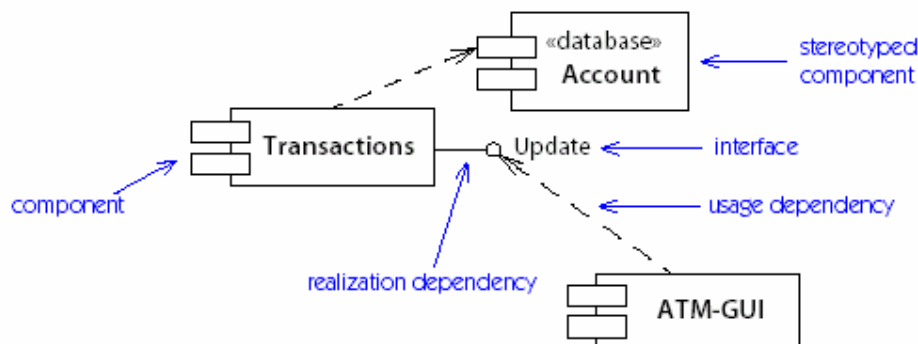


Figure 9-2. Component diagram

Nodo

Un nodo es un objeto físico de ejecución que representa un recurso computacional (computador). Los nodos pueden tener estereotipos como UCP, dispositivos, y memorias.

Las asociaciones entre nodos representan líneas de comunicación. Las asociaciones pueden tener estereotipos para distinguir distintos tipos de enlaces.

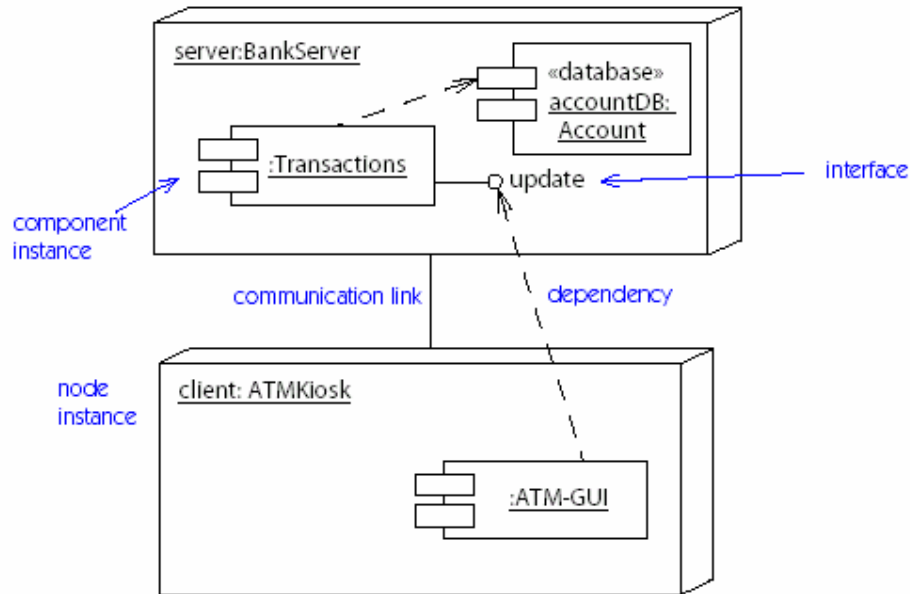


Figure 9-3. *Deployment diagram*

8. La vista de Gestión del Modelo

La gestión de modelo consiste en paquetes y relaciones de dependencias entre paquetes.

Paquete

Un paquete es una parte del modelo. Cada parte del modelo debe pertenecer a un paquete.

Los paquetes tienen elementos del modelo tales como clases, máquinas de estado, diagramas de casos de uso, interacciones y colaboraciones, etc.

Los paquetes también pueden contener otros paquetes.

Hay varias maneras posibles de organizar los paquetes en un sistema: por la vista, por funcionalidad, o por cualquier otra base que elija el modelador.

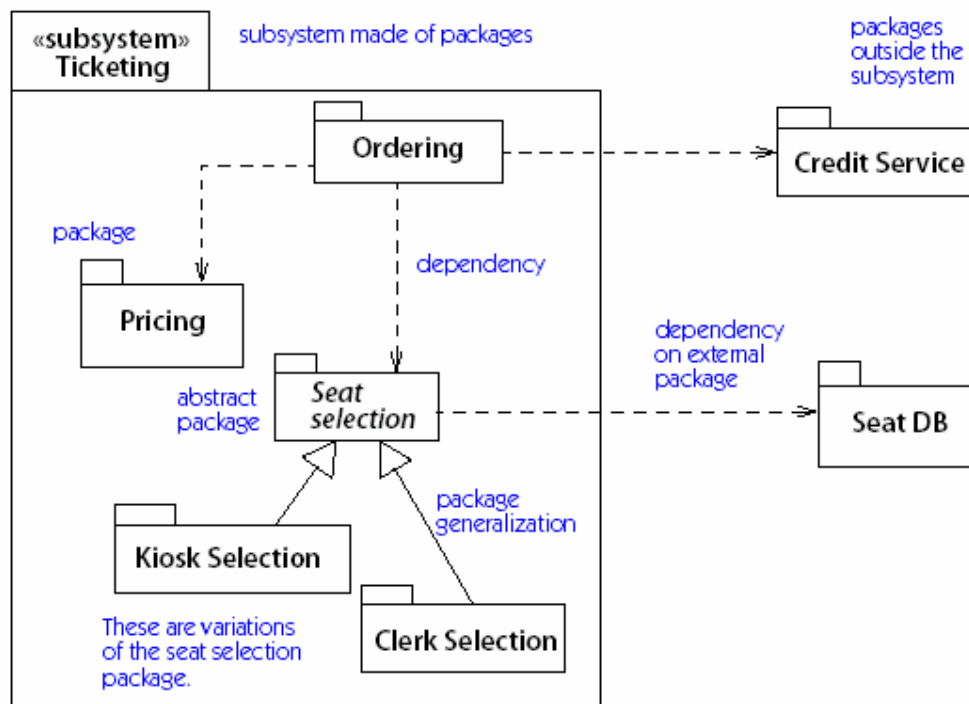
Los paquetes son unidades de organización jerárquica.

Si se eligen bien, los paquetes reflejan la arquitectura de alto nivel de un sistema: su descomposición en subsistemas y sus dependencias.

Dependencias en los paquetes

Las dependencias entre los paquetes resumen dependencias entre elementos internos de ellos.

Los paquetes se dibujan como carpetas. Las dependencias como trazos discontinuos



Dependencias de acceso e importación

En general, un paquete no puede tener acceso al contenido de otro paquete. Los paquetes son opacos, a menos que sean abiertos por una dependencia de *acceso* o de *importación*.

Dependencia de acceso => el contenido del paquete proveedor puede aparecer en referencias efectuadas por elementos del paquete cliente o paquetes incluidos dentro del paquete cliente.

Visibilidad entre paquetes

- Pública: un elemento definido con visibilidad pública dentro de su paquete, puede ser visto desde cualquier otro paquete.
- Protegida: un elemento definido con visibilidad protegida dentro de su paquete, puede ser visto solamente por los paquetes que son descendientes del paquete que contiene los elementos..
- Privada: un elemento definido con visibilidad privada dentro de su paquete, puede ser visto solamente por el paquete que los contiene, y por cualquier paquete anidado en el interior de es paquete.

El permiso de acceso y visibilidad apropiada son necesarios para referenciar a un elemento. Así para que un elemento en un paquete pueda ver un elemento de otro paquete, el primer paquete debe tener acceso o importar al segundo paquete, y el elemento destino debe tener visibilidad pública dentro del segundo paquete.

Un paquete anidado dentro de otro es parte de su contenedor y tiene acceso completo a su contenido, sin necesidad de accesos. El contenedor sin embargo puede no ver el interior de sus paquetes anidados si no tiene acceso.

La dependencia de **acceso no modifica el espacio de nombres** del cliente ni crea referencias. Solo concede permiso para establecer referencias.

La dependencia de **importación** se utiliza para agregar nombres al espacio de nombres del paquete del cliente como sinónimos de los caminos completos.

Modelo y Subsistema

Un modelo es un paquete que abarca una descripción completa de una vista particular de un sistema. Proporciona una descripción cerrada de un sistema a partir de un punto de vista.

La relación de traza es una forma débil de dependencia entre elementos en distintos modelos, que observan la presencia de una cierta conexión sin implicación semánticas específicas.

Generalmente un modelo se estructura con forma de árbol. El paquete raíz contiene anidados en sí mismo paquetes que constituyen el detalle completo del sistema desde el punto de vista dado.

Un subsistema es un paquete que tiene piezas separadas de especificación y de realización. Representa generalmente la partición del sistema en un límite funcional o de implementación.

Los modelos y subsistemas se dibujan como paquetes con las palabras clave de estereotipo.

9. Mecanismos de Extensibilidad