



UNIVERSIDAD DEL BÍO-BÍO

Metodología de Desarrollo de Software

Tutorial de UML

Apuntes tomados de prof. Miguel Arregui, Depto. de lenguajes y Reingeniería de
Sistemas, Universitat Jaume I, España

Elizabeth Grandón Toledo
Departamento de Sistemas de Información
Universidad del Bío-Bío

Contenidos

1.- Introducción	4
2.- La Orientación a Objetos, OO	4
2.1.- Qué es un Objeto	5
2.2.- Qué es una Clase.....	6
2.3.- Qué es la Herencia.....	6
2.4.- Qué es una Interfaz	7
3.- El Lenguaje Unificado de Modelado, UML.....	8
3.1.- Bloques básicos de construcción de UML	8
3.1.1.- Elementos	9
3.1.2.- Relaciones	10
3.1.3.- Diagramas	10
3.1.3.1.- Diagrama de Clases y Diagrama de Objetos	12
3.1.3.2.- Diagrama de Componentes y Diagrama de Despliegue.....	14
3.1.3.3.- Diagrama de Casos de Uso.....	15
3.1.3.4.- Diagrama de Secuencia y Diagrama de Colaboración	17
3.1.3.5.- Diagrama de Estados y Diagrama de Actividades	19
3.2.- Cómo utilizar UML.....	21
4.- Referencias	24

Figuras

Figura 1: Objetos comunicándose	5
Figura 2: La Herencia	7
Figura 3: Diagrama de Clases.....	12
Figura 4: Relación de dependencia en Diagramas de Clase	13
Figura 5: Auto-agregación.....	14
Figura 6: Diagrama de Componentes	14
Figura 7: Diagrama de Despliegue	15
Figura 8: Diagrama de Casos de Uso nivel 1	15
Figura 9: Diagrama Casos de Uso nivel 2 A	16
Figura 10: Diagrama Casos de Uso nivel 2 B	16
Figura 11: Diagrama Casos de Uso nivel 1 detallado	17
Figura 12: Diagrama de Secuencia.....	17
Figura 13: Diagrama de Colaboración	19
Figura 14: Máquina de Estados, estados simples	19
Figura 15: Máquina de Estados, estados compuestos	20
Figura 16: Diagrama de Actividades	21

Tablas

Tabla 1: Elementos de construcción en UML	10
Tabla 2: Elementos de relación en UML.....	10
Tabla 3: Diagramas de UML	11
Tabla 4: Multiplicidad en Diagramas de Clases	13
Tabla 5: Tipos de mensaje en diagramas de interacción	18

1.- Introducción

En los tiempos que corren, el software tiene la tendencia de ser grande y complejo. Los usuarios demandan interfaces cada vez más completas y funcionalidades más elaboradas, todo ello influyendo en el tamaño y la complejidad del producto final. Por ello, los programas deben ser estructurados de manera que puedan ser revisados, corregidos y mantenidos, rápida y eficazmente, por gente que no necesariamente ha colaborado en su diseño y construcción, permitiendo acomodar nueva funcionalidad, mayor seguridad y robustez, funcionando en todas las situaciones que puedan surgir, de manera previsible y reproducible.

Ante problemas de gran complejidad, la mejor forma de abordar la solución es modelar. Modelar es diseñar y estructurar el software antes de programar y es la única forma de visualizar un diseño y comprobar que cumple todos los requisitos para él estipulados, antes de que los programadores comiencen a generar código. Modelando, los responsables del éxito del producto pueden estar seguros de que su funcionalidad es completa y correcta, que todas las expectativas de los usuarios finales se cumplen, que las posibles futuras ampliaciones pueden ser acomodadas, todo ello mucho antes de que la implementación haga que los cambios sean difíciles o imposibles de acomodar. Modelando, se abstraen los detalles esenciales de un problema complejo y se obtiene diseños estructurados que, además, permiten la reutilización de código, reduciendo los tiempos de producción y minimizando las posibilidades de introducir errores.

UML es un lenguaje gráfico que sirve para modelar, diseñar, estructurar, visualizar, especificar, construir y documentar software. UML proporciona un vocabulario común para toda la cadena de producción, desde quien recaba los requisitos de los usuarios, hasta el último programador responsable del mantenimiento. Es un lenguaje estándar para crear los planos de un sistema de forma completa y no ambigua. Fue creado por el Object Management Group, OMG, un consorcio internacional sin fines de lucro, que asienta estándares en el área de computación distribuida orientada a objetos, y actualmente revisa y actualiza periódicamente las especificaciones del lenguaje, para adaptarlo a las necesidades que surgen. El prestigio de este consorcio es un aval más para UML, considerando que cuenta con socios tan conocidos como la NASA, la Agencia Europea del Espacio ESA, el Instituto Europeo de Bioinformática EBI, Boeing, Borland, Motorola y el W3C, por mencionar algunos.

2.- La Orientación a Objetos, OO

Aunque UML puede emplearse en cualquier paradigma, como la programación estructurada o la lógica, está especialmente cerca del paradigma de la orientación a objetos. Por tanto, es precisa una familiarización con algunos detalles de este paradigma antes de continuar con UML.

2.1.- Qué es un Objeto

De manera intuitiva, la tendencia general es asociar el término objeto con todo aquello a lo que se puede atribuir la propiedad física de masa, como una tostadora de pan, aunque es posible encontrar objetos de índole no tangible, como por ejemplo una dirección postal. En el ámbito de la informática, **un objeto define una representación abstracta de las entidades del mundo, tangibles o no, con la intención de emularlas**. Existe pues, una relación directa entre los objetos del mundo y los objetos informáticos, de modo que puede emplearse el término objeto de manera indistinta.

Los objetos tienen dos características, que son su estado y su comportamiento. El estado es una situación en la que se encuentra el objeto, tal que cumple con alguna condición o condiciones particulares, realiza alguna actividad o espera que suceda un acontecimiento. Una tostadora puede estar encendida y cargada de pan y, en cuanto a su comportamiento, lo normal en este estado es tostar pan.

Los objetos mantienen su estado en uno o más atributos, que son simplemente datos identificados por un nombre, y exhiben su comportamiento a través de métodos, que son trozos de funcionalidad asociados al objeto. En este sentido, un objeto es realmente un conjunto de atributos y métodos. Pero un objeto sólo revela su verdadera utilidad cuando es integrado en un contexto de comunicación con otros objetos, a través del envío de mensajes, para componer un sistema mucho mayor y demostrar un comportamiento más complejo. Una tostadora en un armario resulta de poca utilidad, pero cuando interactúa con otro objeto, como un ser humano, se convierte en una herramienta útil para tostar pan. El humano intercambiaría con la tostadora el mensaje “tuesta el pan que tienes en la bandeja” a través de la pulsación del botón de tostar.

A partir del ejemplo anterior, es fácil deducir que el envío de mensajes es la forma en que se invocan los métodos de un objeto y que la invocación de métodos es el mecanismo a través del cual un objeto puede cambiar su estado o el de otro objeto. Los atributos y los métodos de un objeto pueden tener un menor o mayor grado de visibilidad, desde “privado” hasta “público”, lo que hace que aparezca un concepto nuevo, la **encapsulación**. La encapsulación oculta los detalles del funcionamiento interno del objeto, exponiendo sólo aquello que puedan ser de interés.

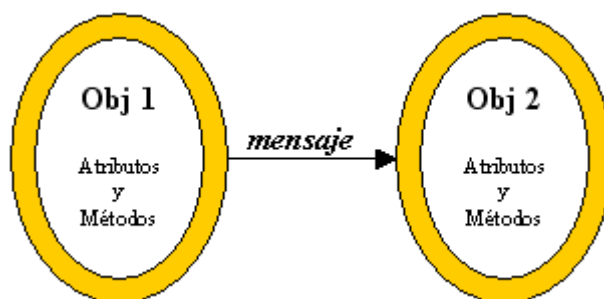


Figura 1: Objetos comunicándose

2.2.- Qué es una Clase

Los objetos no son entidades que existan de modo único. Hay muchos tipos de tostadoras e, igualmente, muchas tostadoras del mismo tipo. Se puede entender fácilmente el concepto de clase si nos permitimos emplear el término *tipo* como equivalente. Así, todos los objetos que son del mismo *tipo*, comparten el mismo juego de atributos y métodos (aunque cada objeto pueda tener un valor distinto asociado a cada atributo) y por tanto pertenecen a una misma clase. Las clases son como patrones que definen qué atributos y qué métodos son comunes a todos los objetos de un mismo tipo.

Cada objeto tiene sus atributos y su comportamiento, creados empleando una clase a modo de patrón. **Una vez creado el objeto, pasa a ser una instancia particular de la clase a la que pertenece** y sus atributos tienen unos valores concretos, que podrán variar de un objeto a otro (dos objetos distintos pertenecientes a la misma clase, pueden tener exactamente los mismos valores en todos sus atributos). A estos atributos, que pueden variar de un objeto a otro, se les conoce también como *variables de instancia*.

Hay atributos que, sin embargo, no varían de un objeto a otro, es decir todas las instancias de la clase a la que pertenecen, tienen el mismo valor para ese atributo. Todas las tostadoras del mismo tipo consumen los mismos Watts y sus resistencias son de los mismos Ohmios. A estos atributos se les conoce como *variables de clase* y son compartidos por todas y cada una de las instancias de la clase. De manera análoga al caso de los atributos, encontramos *métodos de instancia* y *métodos de clase*.

2.3.- Qué es la Herencia

Los objetos se definen en función de clases, es decir, tomando una clase como patrón. Se puede saber mucho acerca de un objeto sabiendo la clase a la que pertenece. Por ejemplo, con decir que la “Russell Hobbs 10243 Kudos” es un tipo de tostadora, inmediatamente se sabe que se trata de una máquina para tostar pan, probablemente eléctrica y con por lo menos una ranura en la que insertar una rebanada de pan y un botón para activar su funcionamiento.

Las clases llegan un paso más lejos, permitiendo su definición en función de otras clases, de modo que es posible establecer una jerarquía de especialización. Una clase que se define en función de otra, hereda todos los atributos y métodos de aquella y permite el añadido de nuevos o la sobre escritura de los heredados. La clase patrón se conoce con el nombre de *superclase* o *clase padre*, mientras que la que hereda se conoce como *clase hija*. La herencia no está limitada simplemente a padre-hija(s), la jerarquía puede ser todo lo profunda que sea necesario, hablando en términos de nietas, bisnietas, etc. De la misma manera, una clase puede heredar de varias clases a la vez.

En la siguiente figura se puede ver una jerarquía de especialización de dos niveles. La clase “Animal” es la raíz, la clase padre en la jerarquía. Especifica que los animales *comen*, como característica más significativa de éstos. En el primer nivel de especialización encontramos las clases “Carnívoro” y “Herbívoro”, ambas son tipos de animal y por tanto *comen*, sólo que en el caso de los carnívoros se ha especializado el

tipo de comida que comen para indicar que se trata de *carne*. Aparece una nueva característica de este tipo de animal, que es el hecho de que los carnívoros *cazan*. En el caso de los herbívoros, encontramos que *comen plantas* y *pacen* (*comen pasto*). En el segundo nivel de especialización, encontramos un animal que es a la vez “Herbívoro” y “Carnívoro” y, como cabe esperar, este nuevo tipo de animal puede hacer todo lo que pueden hacer sus ancestros, comer carne, comer plantas, cazar y pacer, no encontrando ninguna característica adicional en los “Omnívoros”.

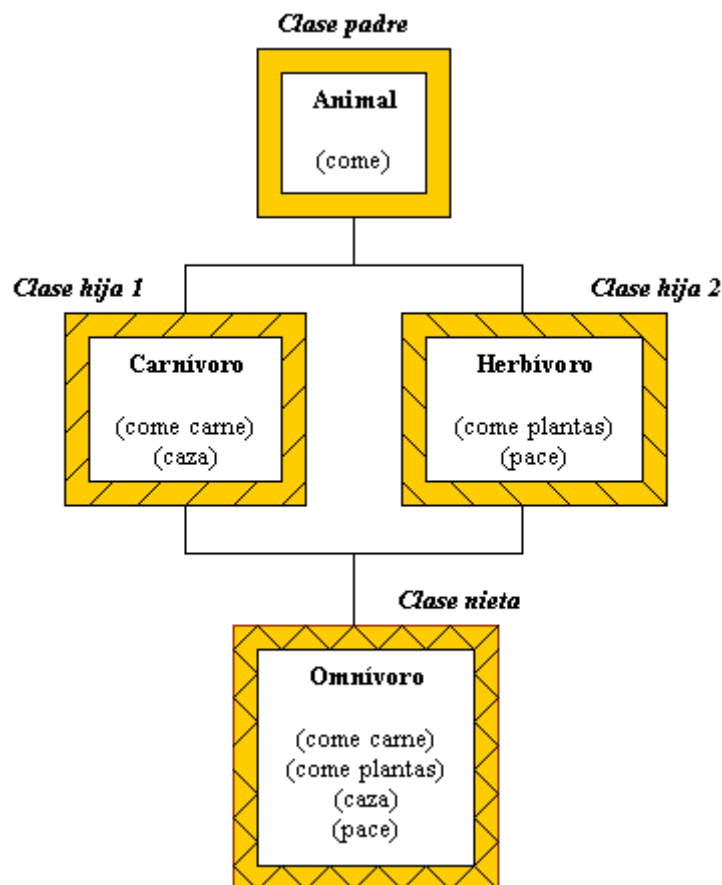


Figura 2: La Herencia

2.4.- Qué es una Interfaz

Una interfaz es un mecanismo que emplean dos objetos para interactuar. En nuestro ejemplo de la tostadora, el humano emplea el botón de tostar a modo de interfaz para pasar el mensaje “tuesta el pan que tienes en la bandeja”.

Las interfaces definen un conjunto de métodos para establecer el protocolo en base al cual interactúan dos objetos. En este sentido, existe una analogía entre interfaces y protocolos. Para que el humano pueda tostar, debe seguir el protocolo establecido por la interfaz botón de tostar, consistente en pulsar dicho botón.

Las interfaces capturan las similitudes entre clases no relacionadas, sin necesidad de forzar una interrelación y son a su vez clases.

3.- El Lenguaje Unificado de Modelado, UML


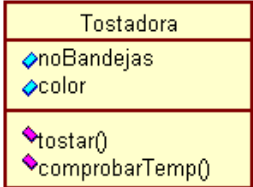



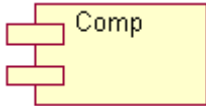

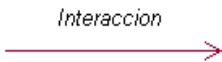
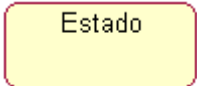
El UML es un lenguaje de modelado cuyo vocabulario y sintaxis están ideados para la representación conceptual y física de un sistema. Sus modelos son precisos, no ambiguos, completos y pueden ser trasladados directamente a una gran variedad de lenguajes de programación, como Java, C++ o Visual Basic, pero también a tablas de bases de datos relacionales y orientadas a objetos. Es posible generar código a partir de un modelo UML (ingeniería directa) y también puede construirse un modelo a partir de la implementación (ingeniería inversa), aunque en las dos situaciones debe intervenir un mayor o menor grado de supervisión por parte del programador, en función de lo buenas que sean las herramientas empleadas.

3.1.- Bloques básicos de construcción de UML

Los bloques básicos de construcción de UML son tres, los elementos, las relaciones y los diagramas.

- Los *elementos* son abstracciones que actúan como unidades básicas de construcción. Hay cuatro tipos, los *estructurales*, los de *comportamiento*, los de *agrupación* y los de *notación*. En cuanto a los elementos estructurales son las partes estáticas de los modelos y representan aspectos conceptuales o materiales. Los elementos de comportamiento son las partes dinámicas de los modelos y representan comportamientos en el tiempo y en el espacio. Los elementos de agrupación son las partes organizativas de UML, establecen las divisiones en que se puede fraccionar un modelo. Sólo hay un elemento de agrupación, el paquete, que se emplea para organizar otros elementos en grupos. Los elementos de notación son las partes explicativas de UML, comentarios que pueden describir textualmente cualquier aspecto de un modelo. Sólo hay un elemento de notación principal, la nota.
- Las *relaciones* son abstracciones que actúan como unión entre los distintos *elementos*. Hay cuatro tipos, la *dependencia*, la *asociación*, la *generalización* y la *realización*.
- Los *diagramas* son la disposición de un conjunto de elementos, que representan el sistema modelado desde diferentes perspectivas. UML tiene nueve diagramas fundamentales, agrupados en dos grandes grupos, uno para modelar la estructura estática del sistema y otro para modelar el comportamiento dinámico. Los **diagramas estáticos** son: el de *clases*, de *objetos*, de *componentes* y de *despliegue*. Los **diagramas de comportamiento** son: el de *casos de uso*, de *secuencia*, de *colaboración*, de *estados* y de *actividades*.

3.1.1.- Elementos

E L E M E N T O S E S T R U C T U R A L E S	Clase	 <pre> classDiagram class Tostadora { +noBandejas +color +tostar() } </pre>	Describe un conjunto de objetos que comparten los mismos atributos, métodos, relaciones y semántica. Las clases implementan una o más interfaces.
	Clase activa	 <pre> classDiagram class Tostadora { +noBandejas +color +tostar() +comprobarTemp() } </pre>	Se trata de una clase, en la que existe procesos o hilos de ejecución concurrentes con otros elementos. Las líneas del contorno son más gruesas que en la clase “normal”
	Interfaz	 <p>Interfaz</p>	Agrupación de métodos u operaciones que especifican un servicio de una clase o componente, describiendo su comportamiento, completo o parcial, externamente visible. UML permite emplear un círculo para representar las interfaces, aunque lo más normal es emplear la clase con el nombre en cursiva.
	Colaboración	 <p>Colaboracion</p>	Define una interacción entre elementos que cooperan para proporcionar un comportamiento mayor que la suma de los comportamientos de sus elementos.
	Caso de uso	 <p>Caso de Uso</p>	Describe un conjunto de secuencias de acciones que un sistema ejecuta, para producir un resultado observable de interés. Se emplea para estructurar los aspectos de comportamiento de un modelo.
	Componente	 <p>Comp</p>	Parte física y por tanto reemplazable de un modelo, que agrupa un conjunto de interfaces, archivos de código fuente, clases, colaboraciones y proporciona la implementación de dichos elementos.
	Nodo	 <p>Nodo</p>	Elemento físico que existe en tiempo de ejecución y representa un recurso computacional con capacidad de procesar.
Elementos de comportamiento	Interacción	 <p>Interaccion</p>	Comprende un conjunto de mensajes que se intercambian entre un conjunto de objetos, para cumplir un objetivo específico.
	Máquinas de estados	 <p>Estado</p>	Especifica la secuencia de estados por los que pasa un objeto o una interacción, en respuesta a eventos.

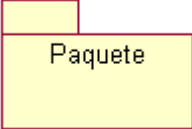
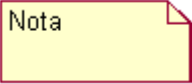
Elementos de agrupación	Paquete		Se emplea para organizar otros elementos en grupos.
Elementos de notación	Nota		Partes explicativa de UML, que puede describir textualmente cualquier aspecto del modelo

Tabla 1: Elementos de construcción en UML

3.1.2.- Relaciones





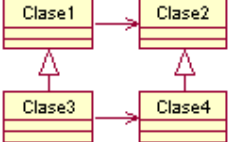
Dependencia		Es una relación entre dos elementos, tal que un cambio en uno puede afectar al otro.
Asociación		Es una relación estructural que resume un conjunto de enlaces que son conexiones entre objetos.
Generalización		Es una relación en la que el elemento generalizado puede ser substituido por cualquiera de los elementos hijos, ya que comparten su estructura y comportamiento.
Realización		Es una relación que implica que la parte realizante cumple con una serie de especificaciones propuestas por la clase realizada (interfaces).

Tabla 2: Elementos de relación en UML

3.1.3.- Diagramas

M O D E L A N	Clases		Muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones, cubriendo la vista de diseño estática del sistema.
	Objetos		Análogo al diagrama de clases, muestra un conjunto de objetos y sus relaciones, pero a modo de vista instantánea de instancias de una clase en el tiempo.

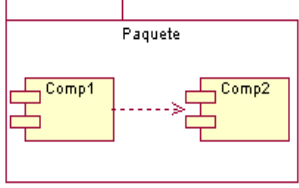
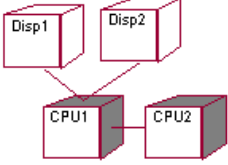
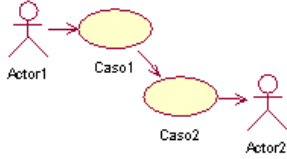
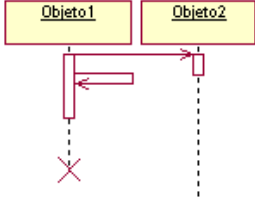
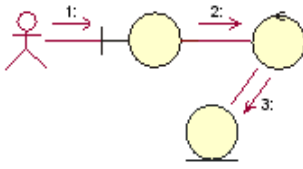
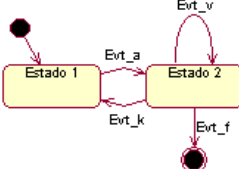
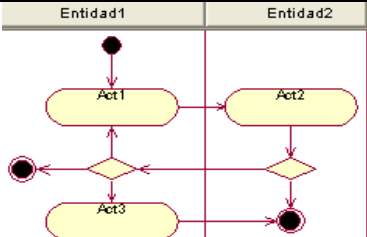
E S T R U C T U R A	Componentes		Muestra la organización y dependencias de un conjunto de componentes. Cubren la vista de implementación estática de un sistema. Un componente es un módulo de código, de modo que los diagramas de componentes son los análogos físicos a los diagramas de clases.
	Despliegue		Muestra la configuración del hardware del sistema, los nodos de proceso y los componentes empleados por éstos. Cubren la vista de despliegue estática de una arquitectura.
M O D E L A N C O M P O R T A M I E N T O	Casos de Uso		Muestra un conjunto de casos de uso, los actores implicados y sus relaciones. Son diagramas fundamentales en el modelado y organización del sistema.
	Secuencia		Son diagramas de interacción, muestran un conjunto de objetos y sus relaciones, así como los mensajes que se intercambian entre ellos. Cubren la vista dinámica del sistema. El diagrama de secuencia resalta la ordenación temporal de los mensajes, mientras que el de colaboración resalta la organización estructural de los objetos, ambos siendo equivalentes o isomorfos. En el diagrama de colaboración de la figura de la izquierda, se puede ver que los elementos gráficos no son cajas rectangulares, como cabría esperar, y en su lugar encontramos sus versiones adornadas. Estas versiones tienen como finalidad evidenciar un rol específico del objeto siendo modelado. En la figura encontramos de izquierda a derecha y de arriba abajo un Actor, una Interfaz, un Control (modela un comportamiento) y una Instancia (modela un objeto de dato).
	Colaboración		
	Estados		Muestra una máquina de estados, con sus estados, transiciones, eventos y actividades. Cubren la vista dinámica de un sistema. Modelan comportamientos reactivos en base a eventos.
	Actividades		Tipo especial de diagrama de estados que muestra el flujo de actividades dentro de un sistema.

Tabla 3: Diagramas de UML

3.1.3.1.- Diagrama de Clases y Diagrama de Objetos

Los diagramas de clases muestran un resumen del sistema en términos de sus clases y las relaciones entre ellas. Son diagramas **estáticos** que muestran **qué** es lo que interactúa, pero no cómo interactúa o qué pasa cuando ocurre la interacción.

El siguiente diagrama modela los pedidos de un cliente a una tienda de venta por catálogo. La clase principal es “Pedido”, asociada a un cliente, una forma de pago y un conjunto de artículos.

La clase “Pago” es abstracta, en UML los nombres de clases abstractas se representan en *Itálica*. Las clases abstractas actúan a modo de **interfaz**, proporcionando únicamente un listado de métodos a ser “realizados” por las clases que las implementan o realizan. “Pago” es una superclase especializada, y a la vez realizada, por sus formas más comunes “Credito” y “Efectivo”. Un “Pedido” tiene una única forma de pago, expresada por su multiplicidad, 1, mientras que una forma de pago puede estar presente en uno o más pedidos, como sugiere su multiplicidad, 1..*.

En cuanto a las asociaciones, observamos que algunas vienen representadas como una *flecha navegable*, cuya orientación expresa el sentido en que se consultan los datos. Las asociaciones sin flecha son bi-direccionales. Las agregaciones expresan “conjunto de”; la relación entre “Pedido” y “Articulo” es de conjunto. Un pedido es una agregación de una o más líneas de pedido, donde cada una hace alusión a un artículo concreto, así mismo una línea de pedido puede estar presente en varios pedidos y un artículo puede no haber sido solicitado nunca.

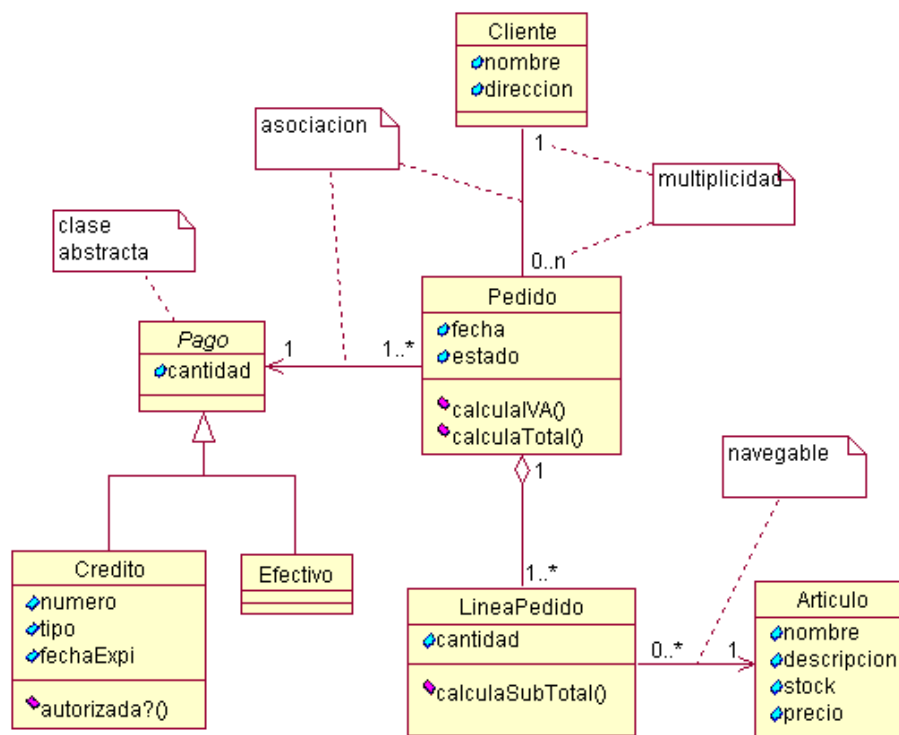


Figura 3: Diagrama de Clases

En cuanto a la multiplicidad, la siguiente tabla resume las más comunes. Hay que tener en cuenta que la multiplicidad se expresa “en el lado opuesto” de la relación y es el número de posibles instancias de una clase asociadas con una única instancia de la clase en el otro extremo.

Multiplicidad	Significado
1	Una única instancia
N / *	N instancias
0..N / 0..*	Entre ninguna y N instancias
1..N / 1..*	Entre una y N instancias
0..1	Ninguna o una instancia
N..M	Entre N y M instancias

Tabla 4: Multiplicidad en Diagramas de Clases

El siguiente diagrama muestra una dependencia existente entre las clases “Pedido” y “Fecha”. Cualquier cambio en la clase dependida, “Fecha”, afectará la clase dependiente, “Pedida”.

Así mismo se puede observar que las clases vienen representadas por cajas en las que hay tres separaciones, o compartimentos. El **primero** se emplea siempre para indicar el *nombre* de la clase, el **segundo** para mostrar los *atributos* y el **tercero** para los *métodos*. Tanto los atributos como los métodos vienen precedidos por un símbolo de acceso, que normalmente suele ser un “+” para el acceso *público*, un “-” para el acceso *privado*, (sólo por otros métodos de la clase) y un “#” para el acceso *protegido* (sólo por clases hija), aunque la herramienta empleada en la elaboración del tutorial traduce estos elementos en iconos.

Los atributos tienen un tipo que puede mostrarse a continuación de su nombre separado por “:”. De igual manera, los métodos pueden devolver un elemento de un tipo determinado y recibir parámetros, expresados entre paréntesis mediante el nombre del parámetro y el tipo, separados por “:”. Para el caso de múltiples parámetros, se separan por comas (*p1:t1, p2:t2 ... pn:tn*). Los parámetros que tienen un valor por defecto se expresan mediante un “=” y el valor, a continuación del tipo (*p1:t1=v1*) y si un parámetro en la posición “i” de la lista de parámetros tiene valor por defecto, todos los parámetros que le sigan, es decir que ocupen posiciones sucesivas a “i” en la lista, deberán tener también un valor por defecto.

Los atributos y métodos estáticos (de clase) se representan mediante un subrayado (en el caso de los métodos se puede emplear el estereotipo <<static>>, los estereotipos se ven más adelante).

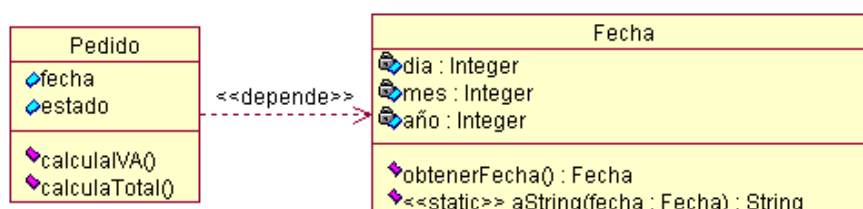


Figura 4: Relación de dependencia en Diagramas de Clase

El siguiente diagrama muestra una auto-relación de agregación. Un “Departamento” puede estar compuesto a su vez por más sub-departamentos, o ninguno, con la restricción de que el mínimo número de personas en los sub-departamentos debe ser dos. Las *restricciones* son condiciones que deben ser cumplidas siempre, se expresan *entre llaves* “{condición }”.

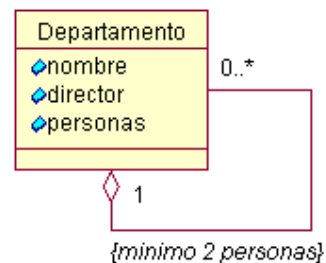


Figura 5: Auto-agregación

Los diagramas de objetos son análogos a los de clases, con la particularidad de que en lugar de encontrar clases, encontramos instancias de éstas. Son útiles para explicar partes pequeñas del modelo en las que hay relaciones complejas.

3.1.3.2.- Diagrama de Componentes y Diagrama de Despliegue

Los componentes son módulos de código, así que los diagramas de componentes vienen a ser los análogos físicos a los diagramas de clases. Muestran como está organizado un conjunto de componentes y las dependencias que existen entre ellos.

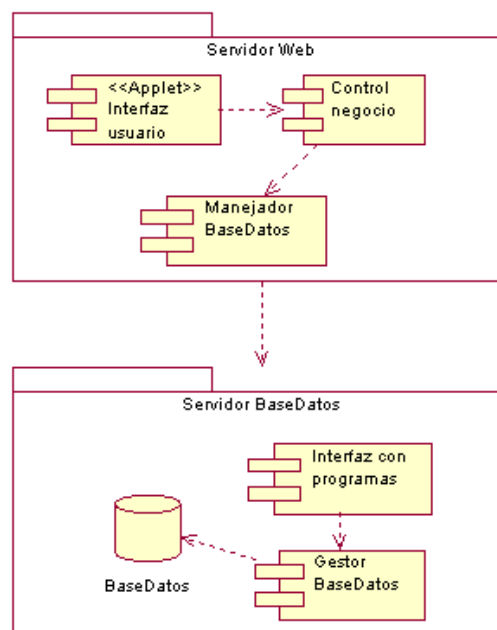


Figura 6: Diagrama de Componentes

Los diagramas de despliegue sirven para modelar la configuración hardware del sistema, mostrando qué nodos lo componen.

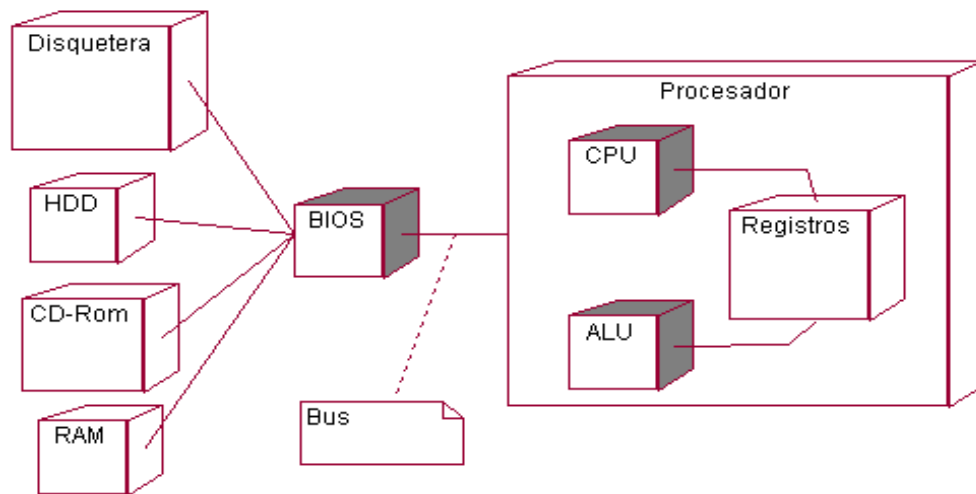


Figura 7: Diagrama de Despliegue

3.1.3.3.- Diagrama de Casos de Uso

Los diagramas de Casos de Uso describen lo que hace un sistema desde el punto de vista de un observador externo, enfatizando el **qué** más que el **cómo**. Plantean escenarios, es decir, lo que pasa cuando alguien interactúa con el sistema, proporcionando un resumen para una tarea u objetivo. El siguiente Caso de Uso describe como Carlos va a desayunar (este es su objetivo), para lo que se plantea el escenario de preparar su café y el pan tostado

.

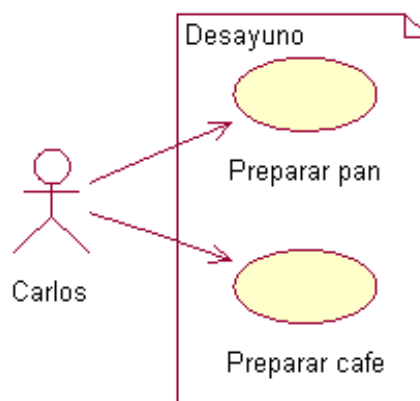


Figura 8: Diagrama de Casos de Uso nivel 1

En los Casos de Uso, los *Actores* son papeles que determinadas personas u objetos desempeñan. Se representan mediante un “hombre de palitos”, de modo que en el ejemplo, Carlos es un Actor. Los Casos de Uso se representan por medio de *óvalos* y las líneas que unen Actores con Casos de Uso representan una asociación de comunicación.

Por su puesto, un Caso de Uso puede ser descrito en mayor profundidad. Por ejemplo si tomamos por separado “Preparar pan” y “Preparar cafe”, podemos bajar un nivel de descripción y llegar a los siguientes Casos de Uso.

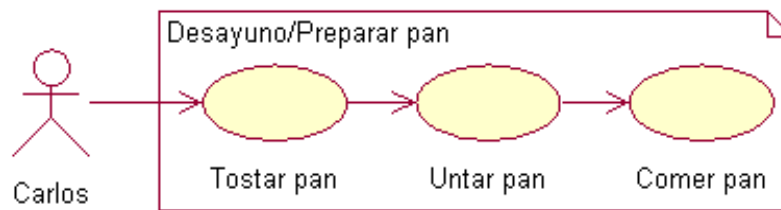


Figura 9: Diagrama Casos de Uso nivel 2 A

“Carlos tuesta el pan en la tostadora, después lo unta con mantequilla y mermelada y se lo come, posiblemente mojándolo en un café.”

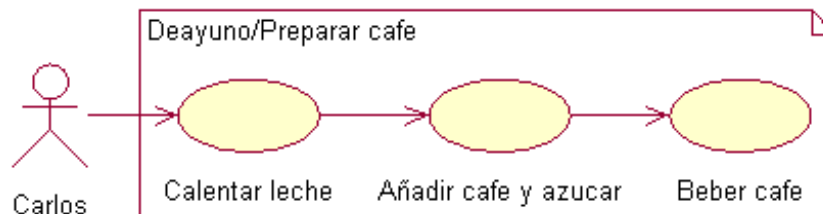


Figura 10: Diagrama Casos de Uso nivel 2 B

“Carlos calienta leche, añade café y azúcar al gusto y se lo bebe.”

Los Casos de Uso suelen venir delimitados por fronteras o límites, que definen una separación entre lo que es realmente la funcionalidad del sistema y los actores que la usan o colaboran en su desempeño. En las figuras, esta separación viene representada por medio de la caja que encapsula los óvalos.

Los Casos de Uso son acompañados por una explicación textual que clarifica las posibles carencias del lenguaje meramente gráfico. De esta manera, combinando Casos de Uso y explicación textual, se puede obtener escenarios no ambiguos, que resultan ideales en la captura de requisitos de usuario, dada su sencillez de comprensión incluso por quien no está familiarizado con UML. Los Casos de Uso se emplean también en la preparación de escenarios de pruebas con que verificar el software una vez ha sido construido.

El siguiente Caso de Uso es equivalente al primero, “Desayuno”, sólo que en él se ha condensado la máxima cantidad posible de información. En él se muestra un nuevo elemento que hasta ahora no se había mostrado, el “**estereotipo**”, que viene entre símbolos angulados “<<” y “>>” y concreta un paso más allá el tipo de relación existente entre dos Casos de Uso. Encontramos dos estereotipos <<**include**>> y <<**extend**>>. El primero indica que el Caso de Uso “Tostar pan” requiere de “Usar tostadora” para poder ser llevado a cabo. Esta es una forma muy adecuada de sacar factor común entre Casos de Uso, o incluso de fraccionar Casos de Uso muy grandes. El segundo indica que el Caso de Uso “Untar pan” es una variación de “Untar”. Observamos también que “Comer pan” y “Beber cafe” son una generalización de “Alimentarse”.

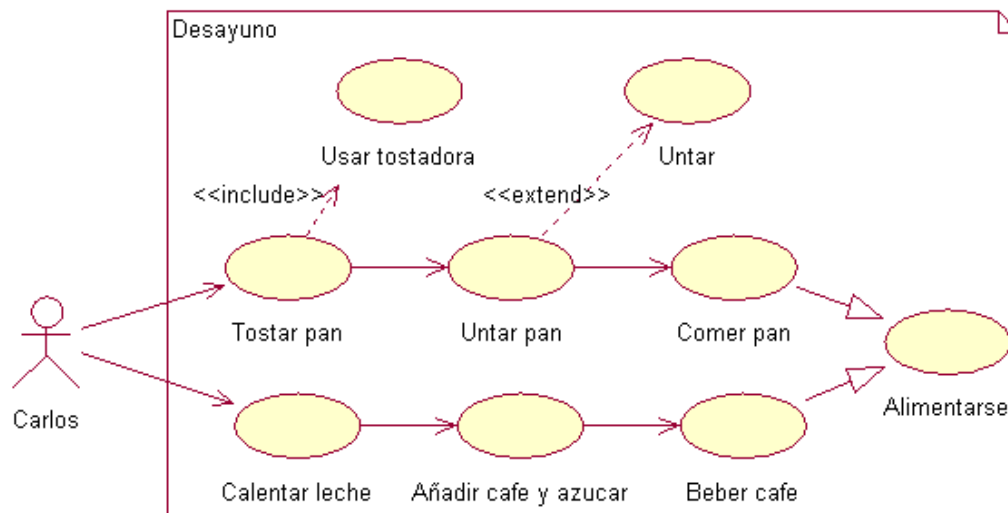


Figura 11: Diagrama Casos de Uso nivel 1 detallado

“Carlos va a desayunar. Para ello debe hacer dos actividades distintas, pero relacionadas. La primera consiste en tostar pan, para lo cual necesita emplear una tostadora. Una vez tostado el pan, lo unta de mantequilla y mermelada (untar pan no es muy distinto de untar otro tipo de alimentos). La segunda consiste en preparar el café, para lo cual necesita calentar leche y añadir café y azúcar. Terminadas ambas actividades, Carlos puede proceder a alimentarse, comiendo el pan tostado y bebiendo el café. El orden en que realice las actividades da igual y también da igual si se realizan a la vez.”

3.1.3.4.- Diagrama de Secuencia y Diagrama de Colaboración

Los diagramas de secuencia describen como los objetos del sistema colaboran. Se trata de un diagrama de interacción que detalla como las operaciones se llevan a cabo, qué mensajes son enviados y cuando, organizado todo en torno al tiempo. El tiempo avanza “hacia abajo” en el diagrama. Los objetos involucrados en la operación se listan de izquierda a derecha de acuerdo a su orden de participación dentro de la secuencia de mensajes.

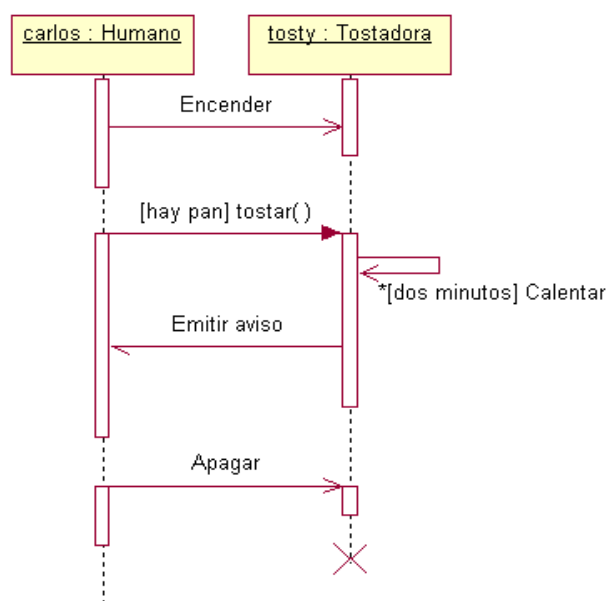


Figura 12: Diagrama de Secuencia

Las líneas verticales o “líneas de la vida” representan el tiempo de vida del objeto. La vida del objeto “carlos” no termina en este diagrama, sin embargo la del objeto “tosty” sí y esto viene representado mediante el aspa al final de su línea de la vida.

Los rectángulos verticales son barras de activación y representan la duración de la ejecución del mensaje. El mensaje “Encender”, posiblemente implementado mediante la introducción del enchufe en un toma corriente, tiene una duración escasa y similar a la de “Apagar”. No ocurre lo mismo con la llamada al método “tostar()”, que dura desde la pulsación del botón de tostar hasta que el pan es retirado de la bandeja y además interviene la emisión de un aviso cuando el pan está lo suficientemente caliente, a fin de evitar que se queme.

Como se puede observar, la acción tostar viene condicionada por la presencia de pan en la bandeja de la tostadora. En UML los corchetes “[]” expresan condición y si están precedidos de un asterisco indican interacción mientras se cumpla la condición.

Los mensajes que son intercambiados entre los objetos de un diagrama de secuencia pueden ser *síncronos* o *asíncronos*. Los mensajes asíncronos son aquellos tal que el emisor puede enviar nuevos mensajes mientras el original está siendo procesado. El mensaje asíncrono ocurre en el tiempo de manera independiente a otros mensajes. Los mensajes síncronos son todo lo contrario, el emisor debe esperar a que termine el tiempo de proceso del mensaje antes de que pueda emitir nuevos mensajes. UML emplea los siguientes convenios para representar el tipo de mensaje.




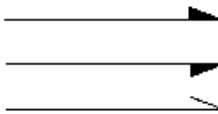
Símbolo	Significado
	Mensaje simple que puede ser síncrono o asíncrono.
	Mensaje simple de vuelta (opcional).
	Mensaje síncrono.
	Mensaje asíncrono.

Tabla 5: Tipos de mensaje en diagramas de interacción

Los diagramas de colaboración son otro tipo de diagramas de interacción, que contiene la misma información que los de secuencia, sólo que se centran en las responsabilidades de cada objeto, en lugar de en el tiempo en que los mensajes son enviados. Cada mensaje de un diagrama de colaboración tiene un número de secuencia. El primer nivel de la secuencia es 1, y los mensajes que son enviados durante la misma llamada a un método se numeran 1.1, 1.2 y así sucesivamente para tantos niveles como sea necesario.

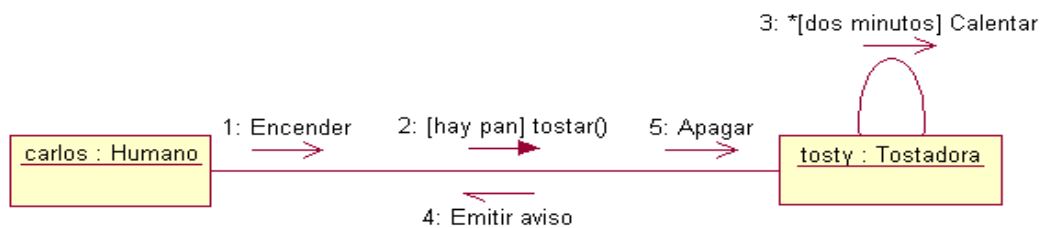


Figura 13: Diagrama de Colaboración

3.1.3.5.- Diagrama de Estados y Diagrama de Actividades

Los diagramas de estados muestran los posibles estados en que puede encontrarse un objeto y las transiciones que pueden causar un cambio de estado. El estado de un objeto depende de la actividad que esté llevando a cabo o de alguna condición.

Las transiciones son las líneas que unen los diferentes estados. En ellas se representa la condición que provoca el cambio, seguida de la acción oportuna separada por “/”. En un estado en que el objeto está pendiente de algún tipo de validación que dependa de un proceso en curso, no es necesario evento externo alguno para que se produzca la transición, ya que ésta ocurrirá cuando termine el proceso, en función del resultado de éste. En estos casos es conveniente, por claridad, incluir la condición que de la que depende la transición (entre corchetes).

Los estados inicial, a partir del que se “entra” en la máquina de estados, y final, que indica que la máquina de estados termina, no tienen otro significado adicional, son elementos ornamentales y se representan mediante un círculo negro y un círculo negro resaltado respectivamente.

Los estados de un diagrama de estados pueden anidarse, de forma que los estados relacionados pueden ser agrupados en un estado compuesto. Esto puede ser necesario cuando una actividad involucra sub-actividades asíncronas o concurrentes.

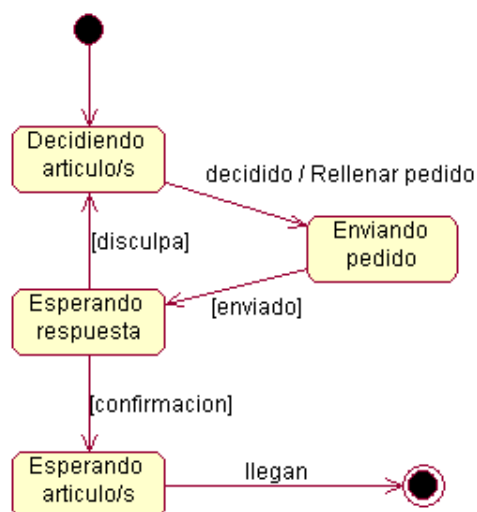


Figura 14: Máquina de Estados, estados simples

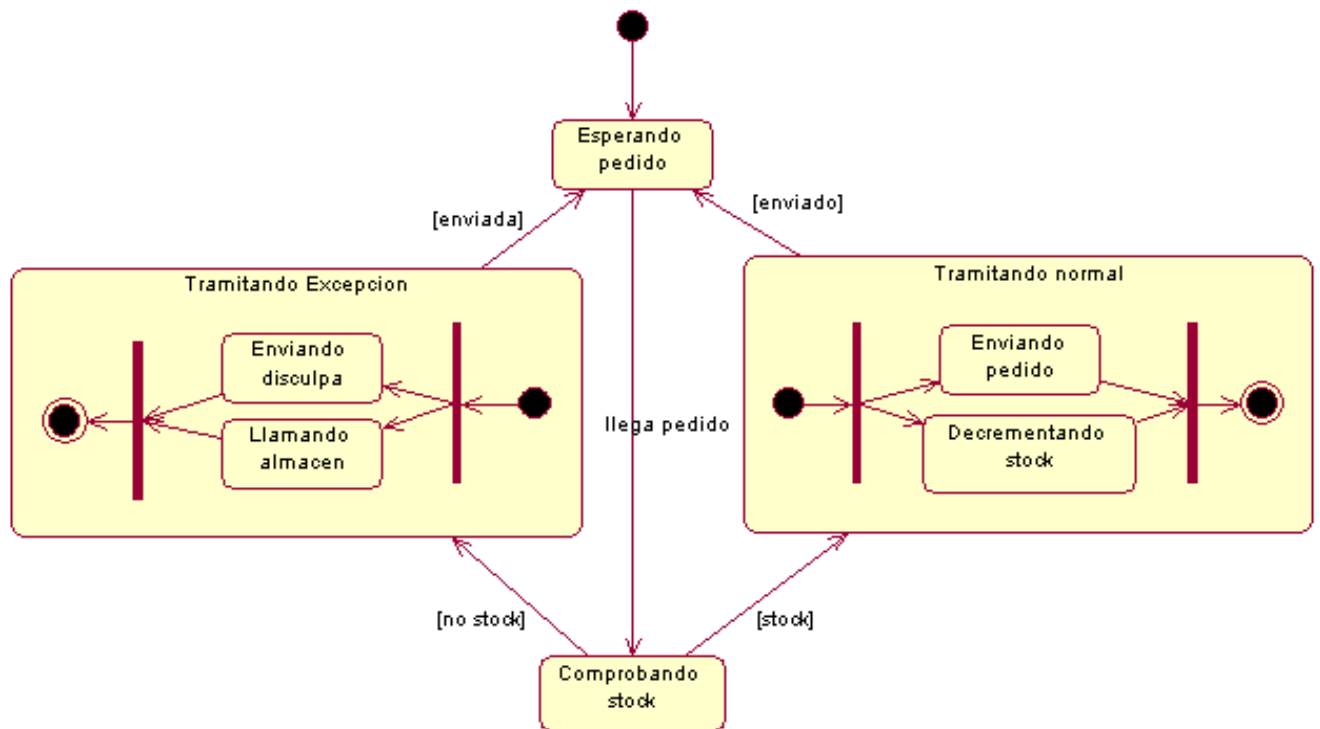


Figura 15: Máquina de Estados, estados compuestos

Los diagramas de actividades son básicamente diagramas de flujo adornados, que guardan mucha similitud con los diagramas de estados. Mientras que los diagramas de estados centran su atención en el proceso que está llevando a cabo un objeto, los diagramas de actividades muestran como las actividades fluyen y las dependencias entre ellas.

Los diagramas de actividades pueden dividirse en “calles” que determinan qué objeto es responsable de qué actividad. Las actividades vienen unidas por transiciones, que pueden separarse en ramas en función del resultado de una condición expresada entre corchetes. Cada rama muestra la condición que debe ser satisfecha para que el flujo opte por ese camino. Igualmente, las transiciones se pueden bifurcarse en dos o más actividades paralelas.

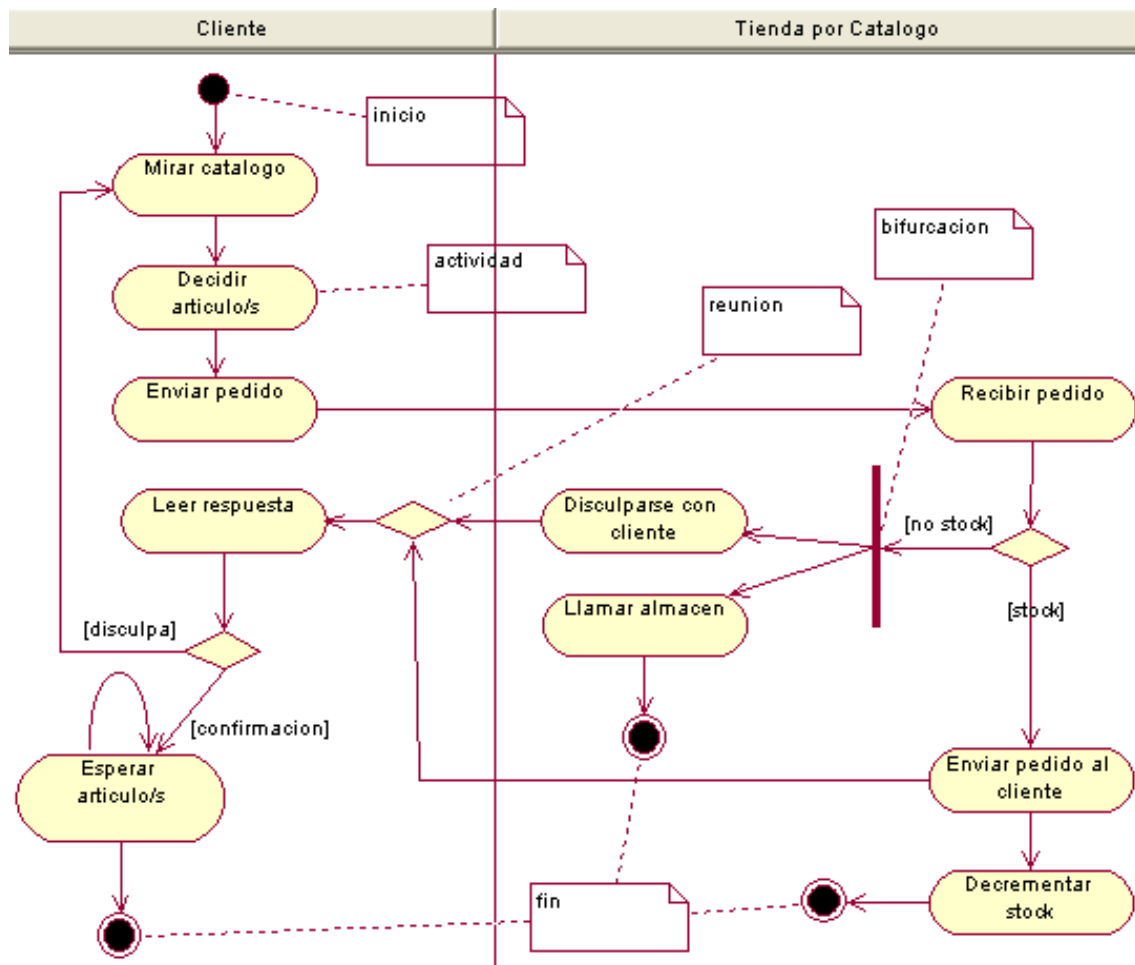


Figura 16: Diagrama de Actividades

3.2.- Cómo utilizar UML

UML es simplemente un lenguaje de modelado. Define un conjunto de elementos y relaciones entre ellos, que se emplean en la definición de modelos. UML es típicamente usado como parte de un proceso de desarrollo, con la ayuda de una herramienta CASE (Computer Aided Software Engineering), para definir requerimientos, interacciones y elementos del software que se está desarrollando. UML es independiente de cualquier proceso particular, no está ligado a ningún ciclo de vida de desarrollo del software concreto, no obstante se obtienen mayores beneficios si se selecciona un proceso que esté dirigido por Casos de U so, se centre en la arquitectura y sea incremental.

La **arquitectura de un sistema** es el conjunto de decisiones significativas que se toma en torno a su organización, la selección de elementos estructurales, la definición de las interfaces entre estos elementos, su comportamiento, su división en subsistemas, qué elementos son estáticos y cuales dinámicos. La arquitectura también incluye el uso que se le va a dar al sistema, la funcionalidad, el rendimiento, la capacidad de adaptación, la reutilización, la capacidad de ser comprendido, las restricciones económicas, las temporales, los compromisos entre alternativas y los aspectos estéticos.

Un proceso incremental es aquél que consiste en sucesivas ampliaciones y mejoras de la arquitectura, a partir de una línea básica. Cada incremento resuelve los problemas encontrados en la versión anterior minimizando incrementalmente los riesgos más significativos para el éxito del proyecto.

Lo primero que se debe hacer para comenzar a desarrollar un proyecto con UML, es seleccionar una metodología de desarrollo que defina la naturaleza concreta del proceso a seguir. El modelo a definir en base al proceso elegido, se divide en realidad en varios tipos de modelo o vistas, cada una centrada en un aspecto o punto de vista del sistema. En general, independientemente del proceso que se emplee, se puede encontrar las siguientes vistas:

- **Vista de Casos de Uso:** Engloba los Casos de Uso que describen el comportamiento del sistema como lo verían los usuarios finales, los analistas y demás componentes del equipo de desarrollo. No especifica la organización del sistema. Con UML los **aspectos estáticos** de esta vista se pueden concretar con los diagramas de *Casos de Uso*; los **aspectos dinámicos** con los diagramas de iteración (*secuencia* y *colaboración*), diagramas de *estados* y de *actividades*.
- **Vista de Diseño:** Engloba las clases e interfaces que conforman el vocabulario del problema y su solución. Da soporte a los requisitos funcionales del sistema, es decir los servicios que proporciona a los usuarios finales. Con UML los **aspectos estáticos** de esta vista se pueden concretar con los diagramas de *clases* y de *objetos*; los **aspectos dinámicos** con los diagramas de iteración (*secuencia* y *colaboración*), diagramas de *estados* y de *actividades*.
- **Vista de Procesos:** Engloba los hilos y procesos que forman los mecanismos de sincronización y concurrencia del sistema. Da soporte al funcionamiento, capacidad de crecimiento y rendimiento del sistema. Con UML los **aspectos estáticos** de esta vista se pueden concretar con los diagramas de *clases*, de *clases activas* y de *objetos*; los **aspectos dinámicos** con los diagramas de iteración (*secuencia* y *colaboración*), diagramas de *estados* y de *actividades*.
- **Vista de Despliegue:** Engloba los nodos que forman la topología hardware sobre el que se ejecuta el sistema. Da soporte a la distribución, entrega e instalación de las partes que conforman el sistema físico. Con UML los **aspectos estáticos** de esta vista se pueden concretar con los diagramas *despliegue*; los **aspectos dinámicos** con los diagramas de iteración (*secuencia* y *colaboración*), diagramas de *estados* y de *actividades*.
- **Vista de Implementación:** Engloba los componentes y archivos empleados para hacer posible el sistema físico. Da soporte a la gestión de configuraciones de las distintas versiones del sistema, a partir de componentes y archivos. Con UML los **aspectos estáticos** de esta vista se pueden concretar con los diagramas de

componentes; los aspectos dinámicos con los diagramas de iteración (*secuencia y colaboración*), diagramas de *estados* y de *actividades*.

Un ejemplo de proceso para la construcción de un programa, podría ser similar al siguiente, teniendo en cuenta que el proceso descrito deja muchas cosas por ampliar y puede no adaptarse a las necesidades particulares de un grupo de trabajo determinado. Se proporciona meramente como un ejemplo de cómo se puede encajar UML como soporte para el desarrollo de un proyecto:

1. Iniciar y mantener *reuniones* con los usuarios finales del programa, para comprender sus necesidades, el contexto en que lo usarán y todos los detalles necesarios para comprender el ámbito del problema a resolver. Esta información será empleada para capturar las actividades y procesos involucrados y susceptibles de ser incorporados en el programa, a un nivel alto, y proporcionará la base para construir la vista de Casos de Uso.
2. Construir la *vista de Casos de Uso* definiendo exactamente la funcionalidad que se va a incorporar en el programa, desde el punto de vista de sus usuarios. El modelo resultante es realmente un mapeo de la información obtenida en el paso anterior, en el que cada nuevo Caso de Uso realiza un aspecto de la funcionalidad planteada. Refinar, en conjunto con los usuarios finales, todos los diagramas de Casos de Uso, incluyendo requisitos y restricciones, para llegar a un acuerdo común en lo que el programa hará y no hará. En este punto puede ser conveniente diseñar escenarios de prueba que ayuden a verificar si el programa finalizado cumple con las expectativas del contrato.
3. Partiendo del modelo de Casos de Uso se comienza a estructurar los requisitos en una arquitectura llamada “línea base”. Se definen clases y relaciones entre ellas, los primeros diagramas de secuencia y colaboración, definiendo los comportamientos de cada clase, también las interfaces entre los diferentes elementos de la arquitectura. Se construye aquí la *vista de diseño* y la *vista de procesos*. Construir diagramas de clases más elaborados y refinar los comportamientos del sistema.
4. A medida que crece el modelo se puede fraccionar en componentes software y paquetes. Aparecen nuevos requisitos que deben ser integrados. Se define la *vista de despliegue*, que define la arquitectura física del sistema, y la *vista de implementación*.
5. Construir el sistema, repartiendo las tareas entre el equipo de programación.
6. Buscar errores de programación, o incluso de diseño, corregirlos e ir sacando sucesivas versiones del programa hasta llegar a una versión que cumpla con todos los requisitos especificados en el contrato con los usuarios.
7. Documentar y entregar el programa a los usuarios finales.

4.- Referencias

Grady Booch, James Rumbaugh, Ivar Jacobson, (1996) *El Lenguaje Unificado de Modelado*, Addison Wesley.

Schneider G., Winters J.P., (2001) *Applying Use Cases: A Practical Guide*, Addison Wesley.