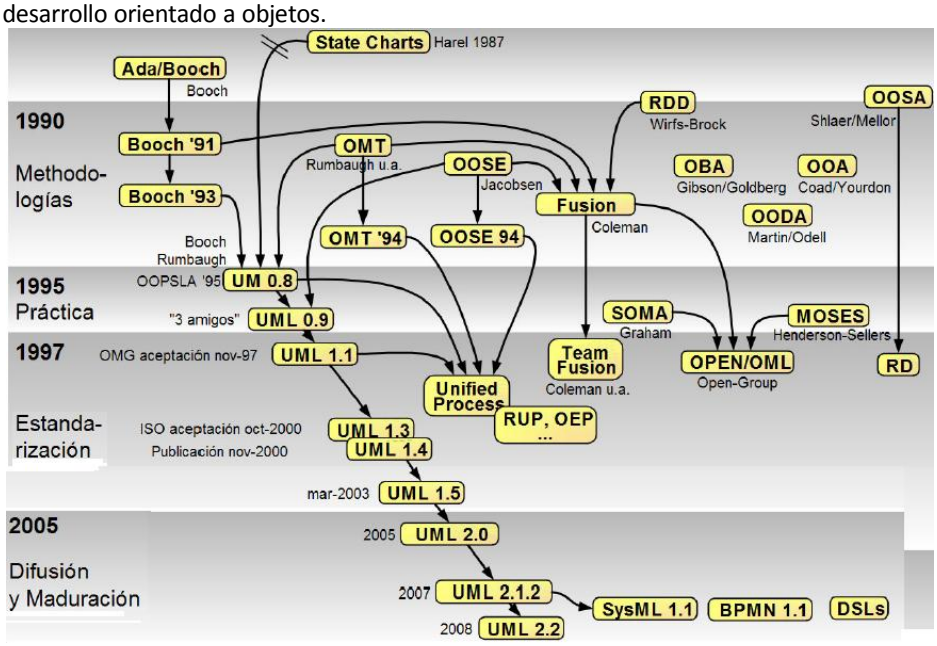
UML

I.1 Introducción

UML (*Unified Modeling Language*) es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos. Se ha convertido en el estándar de facto de la industria, debido a que ha sido concebido por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh. Estos autores fueron contratados por la empresa Rational Software Co. para crear una notación unificada en la que basar la construcción de sus herramientas CASE. En el proceso de creación de UML han participado, no obstante, otras empresas de gran peso en la industria como Microsoft, Hewlett-Packard, Oracle o IBM, así como grupos de analistas y desarrolladores.

Figura 1 Logo de UML

Esta notación ha sido ampliamente aceptada debido al prestigio de sus creadores y debido a que incorpora las principales ventajas de cada uno de los métodos particulares en los que se basa: Booch, OMT y OOSE. UML ha puesto fin a las llamadas "guerras de métodos" que se han mantenido a lo largo de los 90, en las que los principales métodos sacaban nuevas versiones que incorporaban las técnicas de los demás. Con UML se fusiona la notación de estas técnicas para formar una herramienta compartida entre todos los ingenieros software que trabajan en el desarrollo orientado a objetos.



El objetivo principal cuando se empezó a gestar UML era posibilitar el intercambio de modelos entre las distintas herramientas CASE orientadas a objetos del mercado. Para ello era necesario definir una notación y semántica común. En la Figura 2 se puede ver cuál ha sido la evolución de UML hasta la creación de UML 1.1.

Hay que tener en cuenta que el estándar UML no define un proceso de desarrollo específico, tan solo se trata de una notación. En este curso se sigue el método propuesto por Craig Larman
[Larman99] que se ajusta a un ciclo de vida iterativo e incremental dirigido por casos de uso.

En la parte II de este texto se expone la notación y semántica básica de UML, en la parte III se presentan conceptos avanzados de la notación UML, mientras que en la parte IV se presenta el método de desarrollo orientado a objetos de Larman, que se sirve de los modelos de UML que se han visto anteriormente.

II Notación básica UML

En esta parte se verá cómo se representan gráficamente en UML los conceptos principales de la orientación a objetos.

II.1 Modelos

Un modelo representa a un sistema software desde una perspectiva específica. Al igual que la planta y el alzado de una figura en dibujo técnico nos muestran la misma figura vista desde distintos ángulos, cada modelo nos permite

fijarnos en un aspecto distinto del sistema.
Los modelos de UML que se tratan en esta parte son los siguientes:
- Diagrama de Estructura Estática.
- Diagrama de Casos de Uso.
- Diagrama de Secuencia.
- Diagrama de Colaboración.
- Diagrama de Estados.

II.2 Elementos Comunes a Todos los Diagramas

II.2.1 Notas

Una nota sirve para añadir cualquier tipo de comentario a un diagrama o a un elemento de un diagrama. Es un modo de indicar información en un formato libre, cuando la notación del diagrama en cuestión no nos permite expresar dicha información de manera adecuada.
Una nota se representa como un rectángulo con una esquina doblada con texto en su interior. Puede aparecer en un diagrama tanto sola como unida a un elemento por medio de una línea discontinua. Puede contener restricciones, comentarios, el cuerpo de un procedimiento, etc.

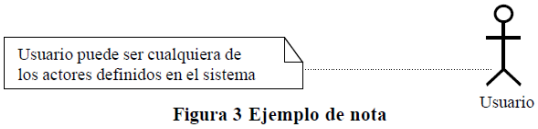


Figura 3 Ejemplo de nota

II.2.2 Dependencias

La relación de dependencia entre dos elementos de un diagrama significa que un cambio en el elemento destino puede implicar un cambio en el elemento origen (por tanto, si cambia el elemento destino habría que revisar el elemento origen).
Una dependencia se representa por medio de una línea de trazo discontinuo entre los dos elementos con una flecha en su extremo. El elemento dependiente es el origen de la flecha y el elemento del que depende es el destino (junto a él está la flecha).

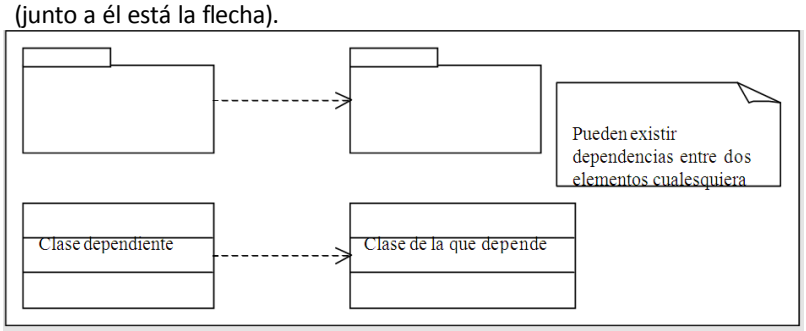


Figura 4 Dependencias

II.3 Diagramas de Estructura Estática

Con el nombre de Diagramas de Estructura Estática se engloba tanto al Modelo Conceptual de la fase de Análisis como al Diagrama de Clases de la fase de Diseño. Ambos son distintos conceptualmente, mientras el primero modela elementos del dominio el segundo presenta los elementos de la solución software. Sin embargo, ambos comparten la misma notación para los elementos que los forman (clases y objetos) y las relaciones que existen entre los mismos (asociaciones).

II.3.1 Clases

Una clase se representa mediante una caja subdividida en tres partes: En la superior se muestra el nombre de la clase, en la media los atributos y en la inferior las operaciones. Una clase puede representarse de forma esquemática (plegada), con los detalles como atributos y operaciones suprimidos, siendo entonces tan solo un rectángulo con el nombre de la clase. En la Figura 5 se ve cómo una misma clase puede representarse a distinto nivel de detalle según interese, y según la fase en la que se esté.

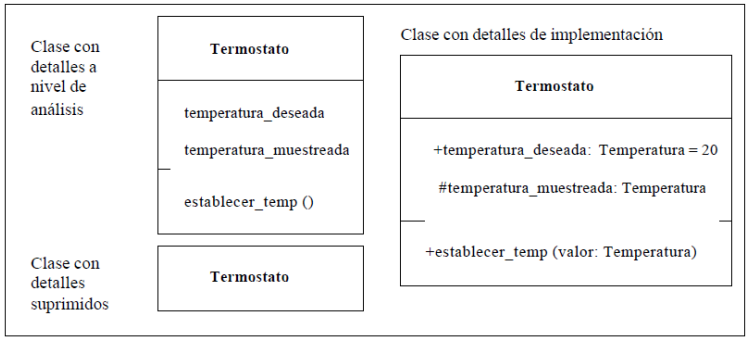


Figura 5 Notación para clases a distintos niveles de detalle

II.3.2 Objetos

Un objeto se representa de la misma forma que una clase. En el compartimento superior aparecen el nombre del objeto junto con el nombre de la clase subrayados, según la siguiente sintaxis:

nombre_del_objeto: nombre_de_la_clase

Puede representarse un objeto sin un nombre específico, entonces sólo aparece el nombre de la clase.

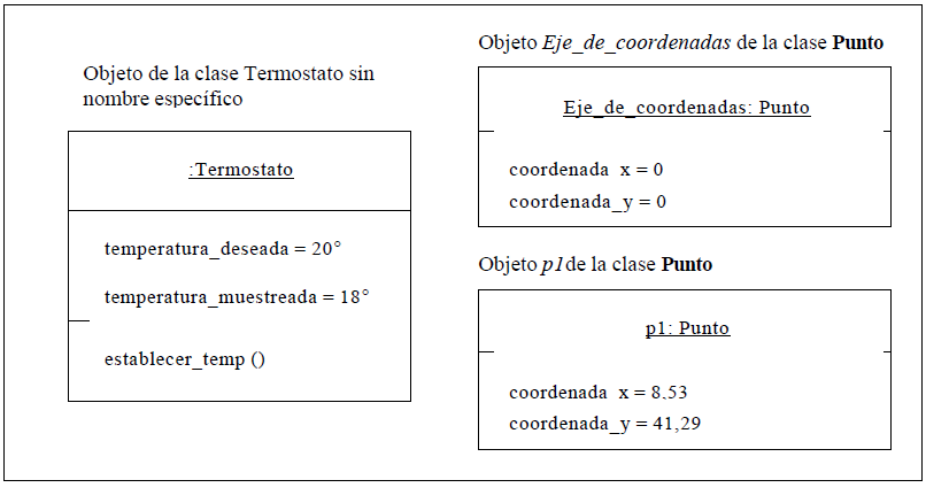


Figura 6 Ejemplos de objetos

II.3.3 Asociaciones

Las asociaciones entre dos clases se representan mediante una línea que las une. La línea puede tener una serie de elementos gráficos que expresan características particulares de la asociación. A continuación se verán los más importantes de entre dichos elementos gráficos.

II.3.3.1 Nombre de la Asociación y Dirección

El nombre de la asociación es opcional y se muestra como un texto que está próximo a la línea. Se puede añadir un pequeño triángulo negro sólido que indique la dirección en la cual leer el nombre de la asociación. En el ejemplo de la Figura 7 se puede leer la asociación como "Director manda sobre Empleado".

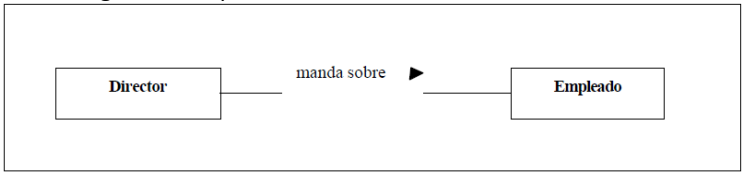


Figura 7 Ejemplo de asociación con nombre y dirección

Los nombres de las asociaciones normalmente se incluyen en los modelos para aumentar la legibilidad. Sin embargo, en ocasiones pueden hacer demasiado abundante la información que se presenta, con el consiguiente riesgo de saturación. En ese caso se puede suprimir el nombre de las asociaciones consideradas como suficientemente conocidas. En las asociaciones de tipo agregación y de herencia no se suele poner el nombre.

II.3.3.2 Multiplicidad

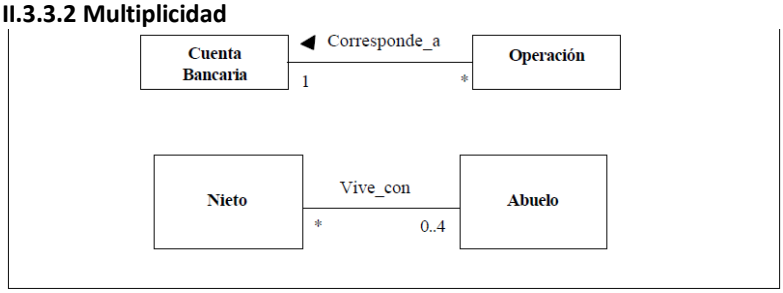


Figura 8 Ejemplos de multiplicidad en asociaciones

La multiplicidad es una restricción que se pone a una asociación, que limita el número de instancias de una clase que pueden tener esa asociación con una instancia de la otra clase. Puede expresarse de las siguientes formas:

- Con un número fijo: *1*.
- Con un intervalo de valores: *2..5*.
- Con un rango en el cual uno de los extremos es un asterisco. Significa que es un intervalo abierto. Por ejemplo, *2..** significa 2 o más.
- Con una combinación de elementos como los anteriores separados por comas: *1, 3..5, 7, 15..**.
- Con un asterisco: *** . En este caso indica que puede tomar cualquier valor (cero o más).

II.3.3.3 Roles

Para indicar el papel que juega una clase en una asociación se puede especificar un nombre de rol. Se representa en el extremo de la asociación junto a la clase que desempeña dicho rol.

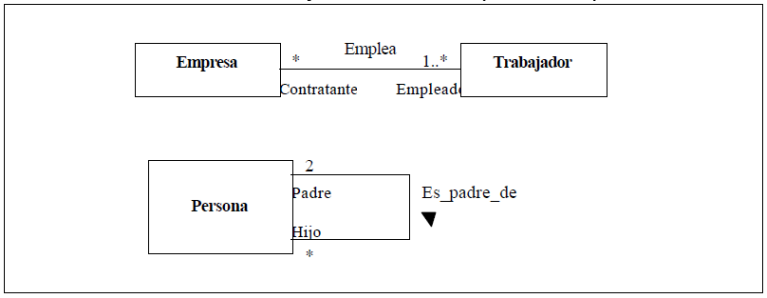


Figura 9 Ejemplo de roles en una asociación

II.3.3.4 Agregación

El símbolo de agregación es un diamante colocado en el extremo en el que está la clase que representa el "todo".

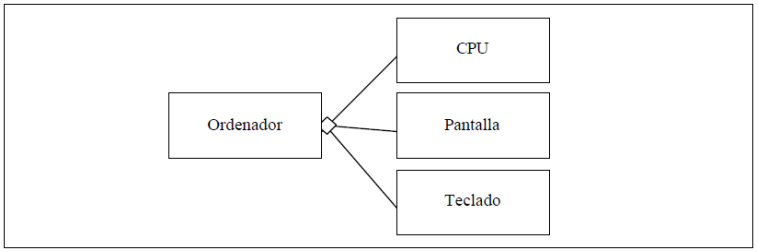


Figura 10 Ejemplo de agregación

II.3.3.5 Clases Asociación

Cuando una asociación tiene propiedades propias se representa como una clase unida a la línea de la asociación por medio de una línea a trazos. Tanto la línea como el rectángulo de clase representan el mismo elemento conceptual: la asociación. Por tanto ambos tienen el mismo nombre, el de la asociación. Cuando la clase asociación sólo tiene atributos el nombre suele ponerse sobre la línea (como ocurre en el ejemplo de la Figura 11). Por el contrario, cuando la clase asociación tiene alguna operación o asociación propia, entonces se pone el nombre en la clase asociación y se puede quitar de la línea.

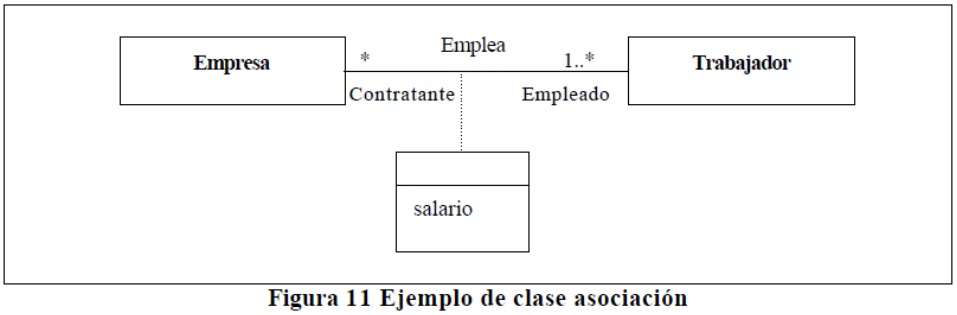


Figura 11 Ejemplo de clase asociación

II.3.3.6 Asociaciones N-Arias

En el caso de una asociación en la que participan más de dos clases, las clases se unen con una línea a un diamante central. Si se muestra multiplicidad en un rol, representa el número potencial de tuplas de instancias en la asociación cuando el resto de los N-1 valores están fijos. En la Figura 12 se ha impuesto la restricción de que un jugador no puede jugar en dos equipos distintos a lo largo de una temporada, porque la multiplicidad de "Equipo" es 1 en la asociación ternaria.

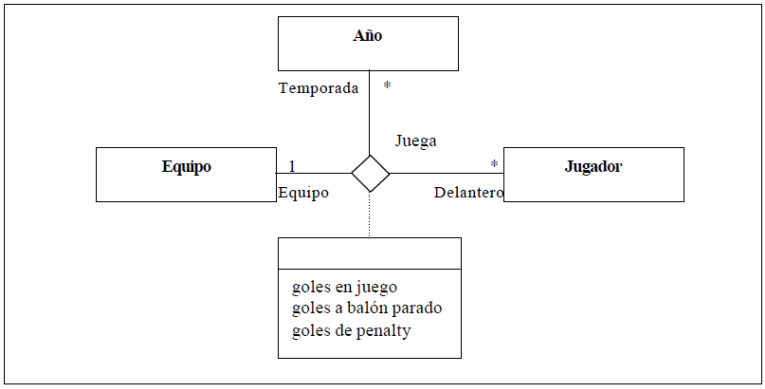


Figura 12 Ejemplo de asociación ternaria

II.3.3.7 Navegabilidad

En un extremo de una asociación se puede indicar la navegabilidad mediante una flecha. Significa que es posible "navegar" desde el objeto de la clase origen hasta el objeto de la clase destino. Se trata de un concepto de diseño, que indica que un objeto de la clase origen conoce al (los) objeto(s) de la clase destino, y por tanto puede llamar a alguna de sus operaciones.

II.3.4 Herencia

La relación de herencia se representa mediante un triángulo en el extremo de la relación que corresponde a la clase más general o clase "padre".

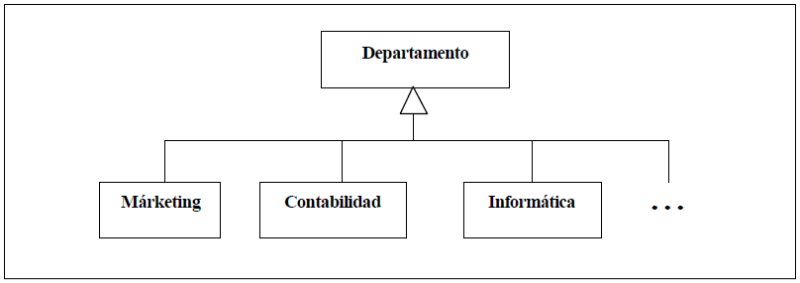


Figura 13 Ejemplo de herencia

Si se tiene una relación de herencia con varias clases subordinadas, pero en un diagrama concreto no se quieren poner todas, esto se representa mediante puntos suspensivos. En el ejemplo de la Figura 13, sólo aparecen en el diagrama 3 tipos de departamentos, pero con los puntos suspensivos se indica que en el modelo completo (el formado por todos los diagramas) la clase "Departamento" tiene subclases adicionales, como podrían ser "Recursos Humanos" y "Producción".

II.3.5 Elementos Derivados

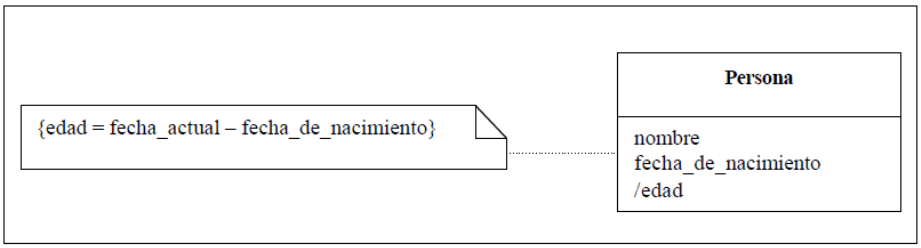


Figura 14 Ejemplo de atributo derivado

Un elemento derivado es aquel cuyo valor se puede calcular a partir de otros elementos presentes en el modelo, pero que se incluye en el modelo por motivos de claridad o como decisión de diseño. Se representa con una barra "/" precediendo al nombre del elemento derivado.

II.4 Diagrama de Casos de Uso

Un Diagrama de Casos de Uso muestra la relación entre los actores y los casos de uso del sistema. Representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa.

II.4.1 Elementos

Los elementos que pueden aparecer en un Diagrama de Casos de Uso son: actores, casos de uso y relaciones entre casos de uso.

II.4.1.1 Actores

Un actor es una entidad externa al sistema que realiza algún tipo de interacción con el mismo. Se representa mediante una figura humana dibujada con palotes. Esta representación sirve tanto para actores que son personas como para otro tipo de actores (otros sistemas, sensores, etc.).

II.4.1.2 Casos de Uso

Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Expresa una unidad coherente de funcionalidad, y se representa en el Diagrama de Casos de Uso mediante una elipse con el nombre del caso de uso en su interior. El nombre del caso de uso debe reflejar la tarea específica que el actor desea llevar a cabo usando el sistema.

II.4.1.3 Relaciones entre Casos de Uso

Entre dos casos de uso puede haber las siguientes relaciones:

- **Extiende**: Cuando un caso de uso especializa a otro extendiendo su funcionalidad.
- **Usa**: Cuando un caso de uso utiliza a otro.

Se representan como una línea que une a los dos casos de uso relacionados, con una flecha en forma de triángulo y con una etiqueta <<extiende>> o <<usa>> según sea el tipo de relación.

En el diagrama de casos de uso se representa también el sistema como una caja rectangular con el nombre en su interior. Los casos de uso están en el interior de la caja del sistema, y los actores fuera, y cada actor está unido a los casos de uso en los que participa mediante una línea. En la Figura 15 se muestra un ejemplo de Diagrama de Casos de Uso para un cajero automático.

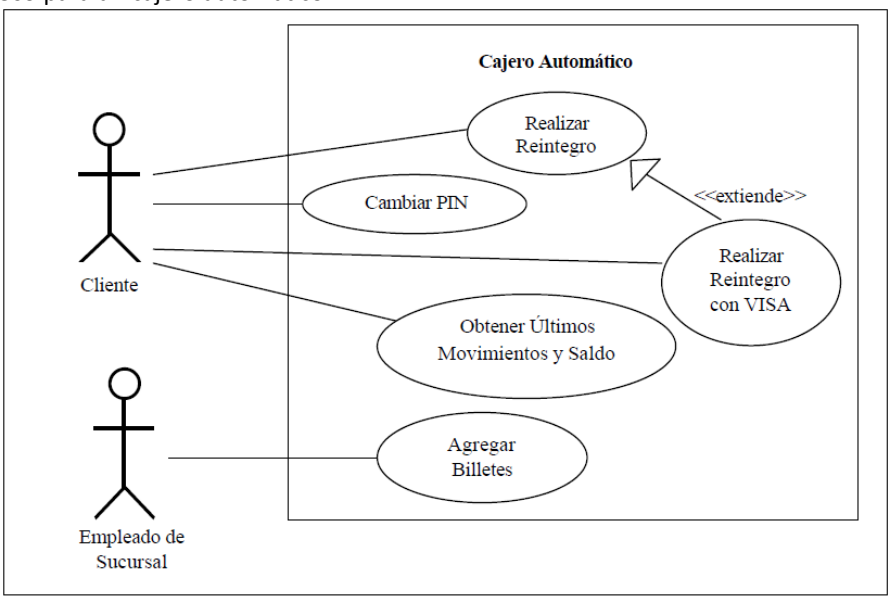


Figura 15 Diagrama de Casos de Uso

II.5 Diagramas de Interacción

En los diagramas de interacción se muestra un patrón de interacción entre objetos. Hay dos tipos de diagrama de interacción, ambos basados en la misma información, pero cada uno enfatizando un aspecto particular: Diagramas de Secuencia y Diagramas de Colaboración.

II.5.1 Diagrama de Secuencia

Un diagrama de Secuencia muestra una interacción ordenada según la secuencia temporal de eventos. En particular, muestra los objetos participantes en la interacción y los mensajes que intercambian ordenados según su secuencia en el tiempo.

El eje vertical representa el tiempo, y en el eje horizontal se colocan los objetos y actores participantes en la interacción, sin un orden prefijado. Cada objeto o actor tiene una línea vertical, y los mensajes se representan mediante flechas entre los distintos objetos. El tiempo fluye de arriba abajo.

Se pueden colocar etiquetas (como restricciones de tiempo, descripciones de acciones, etc.) bien en el margen izquierdo o bien junto a las transiciones o activaciones a las que se refieren.

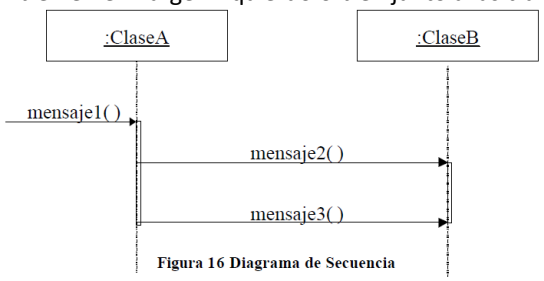


Figura 16 Diagrama de Secuencia

II.5.2 Diagrama de Colaboración

Un Diagrama de Colaboración muestra una interacción organizada basándose en los objetos que toman parte en la interacción y los enlaces entre los mismos (en cuanto a la interacción se refiere). A diferencia de los Diagramas de Secuencia, los Diagramas de Colaboración muestran las relaciones entre los roles de los objetos. La secuencia de los mensajes y los flujos de ejecución concurrentes deben determinarse explícitamente mediante números de secuencia.

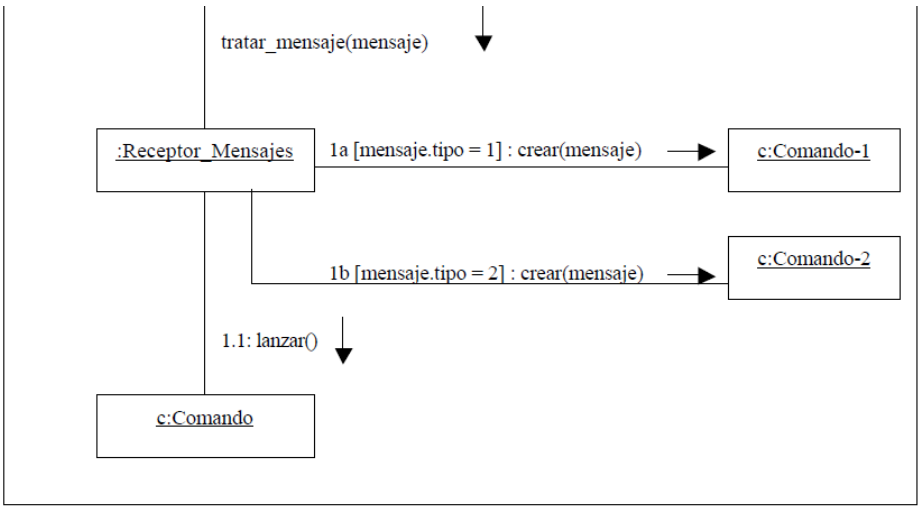


Figura 17 Diagrama de Colaboración

En cuanto a la representación, un Diagrama de Colaboración muestra a una serie de objetos con los enlaces entre los mismos, y con los mensajes que se intercambian dichos objetos. Los mensajes son flechas que van junto al enlace por el que "circulan", y con el nombre del mensaje y los parámetros (si los tiene) entre paréntesis.

Cada mensaje lleva un número de secuencia que denota cuál es el mensaje que le precede, excepto el mensaje que inicia el diagrama, que no lleva número de secuencia. Se pueden indicar alternativas con condiciones entre corchetes (por ejemplo *3 [condición_de_test] : nombre_de_método()*), tal y como aparece en el ejemplo de la Figura 17. También se puede mostrar el anidamiento de mensajes con números de secuencia como *2.1*, que significa que el mensaje con número de secuencia 2 no acaba de ejecutarse hasta que no se han ejecutado todos los *2. x* .

II.6 Diagrama de Estados

Un Diagrama de Estados muestra la secuencia de estados por los que pasa un caso de uso o un objeto a lo largo de su vida, indicando qué eventos hacen que se pase de un estado a otro y cuáles son las respuestas y acciones que genera.

En cuanto a la representación, un diagrama de estados es un grafo cuyos nodos son estados y cuyos arcos dirigidos son transiciones etiquetadas con los nombres de los eventos.

Un estado se representa como una caja redondeada con el nombre del estado en su interior. Una transición se representa como una flecha desde el estado origen al estado destino.

La caja de un estado puede tener 1 o 2 compartimentos. En el primer compartimento aparece el nombre del estado. El segundo compartimento es opcional, y en él pueden aparecer acciones de entrada, de salida y acciones internas.

Una acción de entrada aparece en la forma *entrada/acción_asociada* donde *acción_asociada* es el nombre de la acción que se realiza al entrar en ese estado. Cada vez que se entra al estado por medio de una transición la acción de entrada se ejecuta.

Una acción de salida aparece en la forma *salida/acción_asociada*. Cada vez que se sale del estado por una transición de salida la acción de salida se ejecuta.

Una acción interna es una acción que se ejecuta cuando se recibe un determinado evento en ese estado, pero que no causa una transición a otro estado. Se indica en la forma *nombre_de_evento/acción_asociada.*

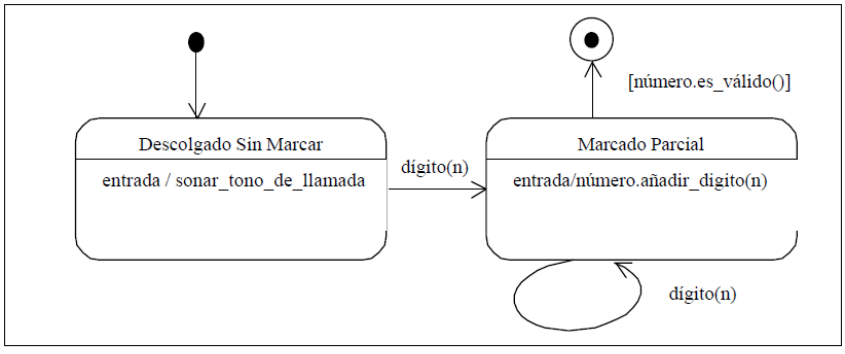


Figura 18 Diagrama de Estados

Un diagrama de estados puede representar ciclos continuos o bien una vida finita, en la que hay un estado inicial de creación y un estado final de destrucción (del caso de uso o del objeto). El estado inicial se muestra como un círculo sólido y el estado final como un círculo sólido rodeado de otro círculo. En realidad, los estados inicial y final son pseudoestados, pues un objeto no puede "estar" en esos estados, pero nos sirven para saber cuáles son las transiciones inicial y final(es).

III Notación Avanzada UML

III.1 Modelado Dinámico

III.1.1 Diagramas De Actividades

Vamos a recordar los diferentes modelos que sirven para representar el aspecto dinámico de un sistema:

- Diagramas de secuencia
- Diagramas de colaboración
- Diagramas de estados
- Diagramas de casos de uso
- ***Diagramas de actividades***

En este capítulo nos centraremos en los *diagramas de actividades* que sirven fundamentalmente para modelar el flujo de control entre actividades.

La idea es generar una especie de diagrama Pert, en el que se puede ver el flujo de actividades que tienen lugar a lo largo del tiempo, así como las tareas concurrentes que pueden realizarse a la vez. El diagrama de actividades sirve para representar el sistema desde otra perspectiva, y de este modo complementa a los anteriores diagramas vistos.

Gráficamente un diagrama de actividades será un conjunto de arcos y nodos.

Desde un punto de vista conceptual, el diagrama de actividades muestra cómo fluye el control de unas clases a otras con la finalidad de culminar con un flujo de control total que se corresponde con la consecución de un proceso más complejo.

Por este motivo, en un diagrama de actividades aparecerán acciones y actividades correspondientes a distintas clases. Colaborando todas ellas para conseguir un mismo fin. Ejemplo: Hacer un pedido.

III.1.2 Contenido del diagrama de actividades

Básicamente un diagrama de actividades contiene:

- Estados de actividad
- Estados de acción
- Transiciones
- Objetos

III.1.2.1 Estados de actividad y estados de acción

La representación de ambos es un rectángulo con las puntas redondeadas, en cuyo interior se representa bien una actividad o bien una acción. La forma de expresar tanto una actividad como una acción, no queda impuesta por UML, se podría utilizar lenguaje natural, una especificación formal de expresiones, un metalenguaje, etc.

La idea central es la siguiente:

> "Un estado que represente una acción es atómico, lo que significa que su ejecución se puede considerar instantánea y no puede ser interrumpida"

En la Figura 19, podemos ver ejemplos de estados de acción.

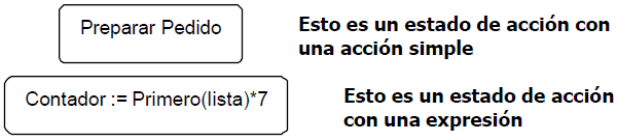


Figura 19 Estados de Acción

En cambio un *estado de actividad*, sí puede descomponerse en más sub-actividades representadas a través de otros diagramas de actividades. Además estos estados sí pueden ser interrumpidos y tardan un cierto tiempo en completarse. En los estados de actividad podemos encontrar otros elementos adicionales como son: acciones de entrada (entry) y de salida (exit) del estado en cuestión, así como definición de submáquinas, como podemos ver en la Figura 20.

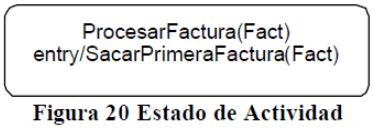


Figura 20 Estado de Actividad

III.1.2.2 Transiciones

Las transiciones reflejan el paso de un estado a otro, bien sea de actividad o de acción. Esta transición se produce como resultado de la finalización del estado del que parte el arco dirigido que marca la transición. Como todo flujo de control debe empezar y terminar en algún momento, podemos indicar esto utilizando dos disparadores de inicio y fin tal y como queda reflejado en el ejemplo de la Figura 21.

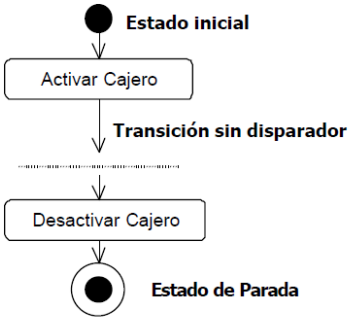


Figura 21 Transiciones sin disparadores

III.1.2.3 Bifurcaciones

Un flujo de control no tiene porqué ser siempre secuencial, puede presentar caminos alternativos. Para poder representar dichos caminos alternativos o bifurcación se utilizará como símbolo el rombo. Dicha bifurcación tendrá una transición de entrada y dos o más de salida. En cada transición de salida se colocará una expresión booleana que será evaluada una vez al llegar a la bifurcación, las guardas de la bifurcación han de ser excluyentes y contemplar todos los casos ya que de otro modo la ejecución del flujo de control quedaría interrumpida.

Para poder cubrir todas las posibilidades se puede utilizar la palabra ELSE, para indicar una transición obligada a un determinado estado cuando el resto de guardas han fallado.

En la Figura 22 podemos ver un ejemplo de bifurcación.

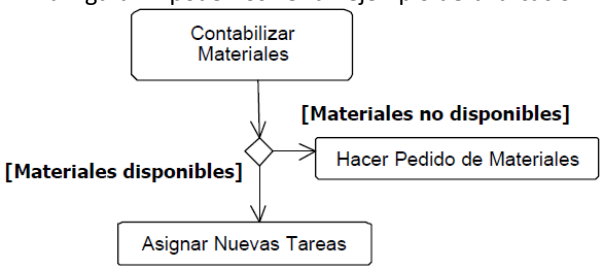


Figura 22 Bifurcación

III.1.2.4 División y unión

No sólo existe el flujo secuencial y la bifurcación, también hay algunos casos en los que se requieren tareas concurrentes. UML representa gráficamente el proceso de división, que representa la concurrencia, y el momento de la unión de nuevo al flujo de control secuencial, por una línea horizontal ancha. En la Figura 23 podemos ver cómo se representa gráficamente.

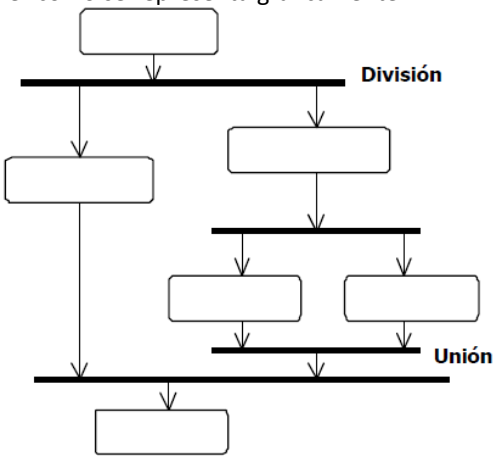


Figura 23 División y unión

III.1.2.5 Calles

Cuando se modelan flujos de trabajo de organizaciones, es especialmente útil dividir los estados de actividades en grupos, cada grupo tiene un nombre concreto y se denominan calles. Cada
calle representa a la parte de la organización responsable de las actividades que aparecen en esa calle. Gráficamente quedaría como se muestra en la Figura 24.

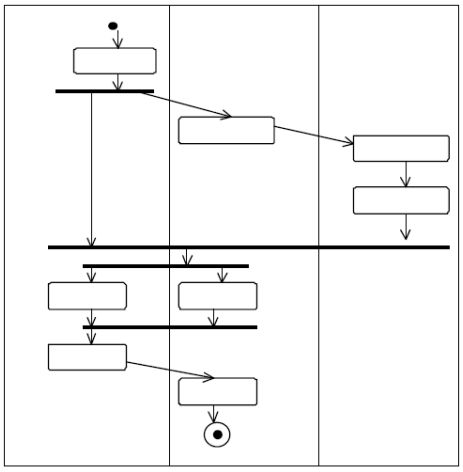


Figura 24 Calles

III.2 Modelado Físico De Un Sistema OO

III.2.1 Componentes

Los componentes pertenecen al mundo físico, es decir, representan un bloque de construcción al modelar aspectos físicos de un sistema.

Una *característica básica* de un componente es que:

> "debe definir una abstracción precisa con una interfaz bien definida, y permitiendo reemplazar fácilmente los componentes más viejos con otros más nuevos y compatibles."

En UML todos los elementos físicos se modelan como componentes. UML proporciona una representación gráfica para estos como se puede ver en la Figura 25, en la que XXXX.dll, es el nombre del componente.

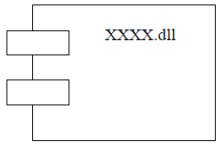


Figura 25 Representación de un componente

Cada componente debe tener un nombre que lo distinga de los demás. Al igual que las clases los componentes pueden enriquecerse con compartimentos adicionales que muestran sus detalles, como se puede ver en la Figura 26.

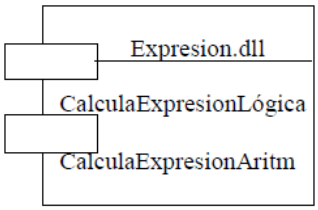


Figura 26 Representación extendida de un componente

Vamos a ver con más detalle cuáles son los parecidos y las diferencias entre los componentes y las clases.

PARECIDOS
Ambos tienen nombre
Ambos pueden realizar un conjunto de interfaces.
Pueden participar en relaciones de dependencia, generalización y asociación.
Ambos pueden anidarse
Ambos pueden tener instancias
Ambos pueden participar en interacciones

DIFERENCIAS	
Las Clases	Los Componentes
Representan abstracciones lógicas	Representan elementos físicos
Es decir los componentes pueden estar en nodos y las clases no	
Pueden tener atributos y operacionesdirectamente accesibles.	Sólo tienen operaciones y estas son alcanzables a través de la interfaz del componente.

III.2.1.1 Interfaces

Tanto los servicios propio de una clase como los de un componente, se especifican a través de una *Interfaz.* Por ejemplo, todas las facilidades más conocidas de los sistemas operativos, basados en componentes (COM+, CORBA, etc.), utilizan las interfaces como lazo de unión entre unos componentes y otros.

La relación entre un componente y sus interfaces se puede representar de dos maneras diferentes, de forma icónica como se indica en la Figura 27, y de forma expandida como se muestra en la Figura 28.

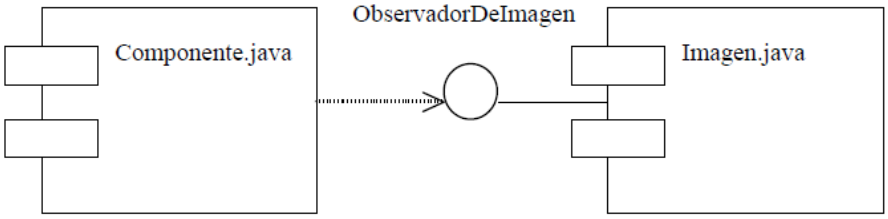


Figura 27 Componentes e interfaces, formato icónico

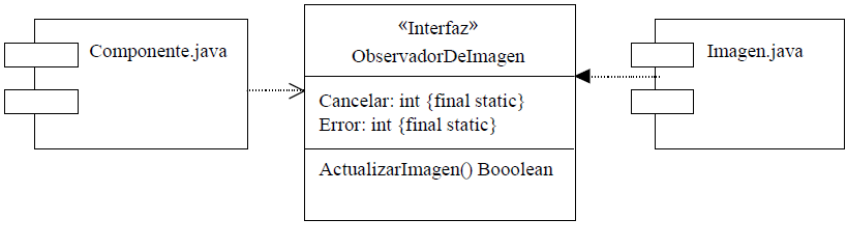


Figura 28 Componentes e interfaces, formato extendido

III.2.1.2 Tipos de componentes

Existen básicamente tres tipos de componentes:

- **Componentes de despliegue:** componentes necesarios y suficientes para formar un sistema ejecutable, como pueden ser las bibliotecas dinámicas (DLLs) y ejecutables (EXEs).
- **Componentes producto del trabajo:** estos componentes son básicamente productos que quedan al final del proceso de desarrollo. Consisten en cosas como archivos de código fuente y de datos a partir de los cuales se crean los componentes de despliegue.
- **Componentes de ejecución:** son componentes que se crean como consecuencia de un sistema en ejecución. Es el caso de un objeto COM+ que se instancia a partir de una DLL.

III.2.1.3 Organización de componentes

Los componentes se pueden agrupar en paquetes de la misma forma que se organizan las clases. Además se pueden especificar entre ellos relaciones de dependencia, generalización, asociación (incluyendo agregación), y realización.

III.2.1.4 Estereotipos de componentes

UML define cinco estereotipos estándar que se aplican a los componentes:

executable Componente que se puede ejecutar en un nodo.

library Biblioteca de objetos estática o dinámica.

table Componentes que representa una tabla de una base de datos.

file Componente que representa un documento que contiene código fuente o datos.

document Componente que representa un documento. UML no especifica iconos predefinidos para estos estereotipos.

III.2.2 Despliegue

III.2.2.1 Nodos

Al igual que los componentes los nodos pertenecen al mundo material. Vamos a definir un nodo como un elemento físico, que existe en tiempo de ejecución y representa un recurso computacional que generalmente tiene alguna memoria y, a menudo, capacidad de procesamiento.

Los nodos sirven para modelar la topología del hardware sobre el que se ejecuta el sistema. Un nodo representa normalmente un procesador o un dispositivo sobre el que se pueden desplegar los componentes.

Un nodo debe tener un nombre asignado que lo distinga del resto de nodos. Además los nodos se representan gráficamente como se indica en la Figura 29.

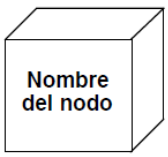


Figura 29 Nodos

III.2.2.2 Nodos y componentes

En muchos aspectos los nodos y los componentes tienen características parecidas. Vamos a ver con más detalle cuales son los parecidos y las diferencias entre los componentes y los nodos.

PARECIDOS
Ambos tienen nombre
Pueden participar en relaciones de dependencia, generalización y asociación.
Ambos pueden anidarse
Ambos pueden tener instancias
Ambos pueden participar en interacciones

DIFERENCIAS	
Los Nodos	Los Componentes
Son los elementos donde se ejecutan los componentes.	Son los elementos que participan en la ejecución de un sistema.
Representan el despliegue físico de los componentes.	Representan el empaquetamiento físico de los elementos lógicos.

La relación entre un nodo y los componentes que despliega se pueden representar mediante una relación de dependencia como se indica en la Figura 30.

Los nodos se pueden agrupar en paquetes igual que los las clases y los componentes.

Los tipos de relación más común entre nodos es la asociación. Una asociación entre nodos viene a representar una conexión física entre nodos como se puede ver en la Figura 31.

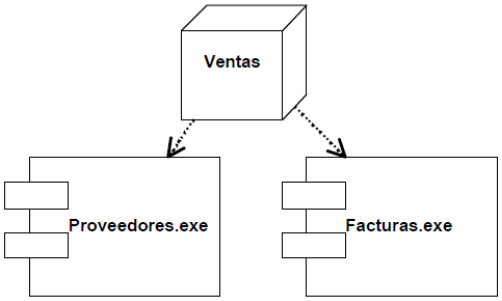


Figura 30 Relación entre nodos y componentes

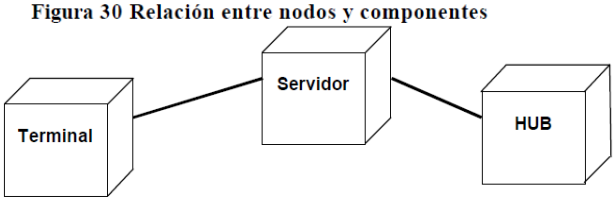


Figura 31 Conexiones entre nodos

Existen dos tipos de diagramas que sirven para modelar los aspectos físicos de un sistema orientado a objetos:

- Diagramas de Componentes
- Diagramas de Despliegue

Seguidamente veremos para qué sirve cada uno de ellos y cuál es su representación gráfica.

III.2.3 Diagramas de Componentes

Un diagrama de componentes muestra la organización y las dependencias entre un conjunto de componentes.

Para todo sistema OO se han de construir una serie de diagramas que modelan tanto la parte estática (diagrama de clases), como dinámica (diagramas de secuencia, colaboración, estados y de actividades), pero llegado el momento todo esto se debe materializar en un sistema implementado que utilizará partes ya implementadas de otros sistemas, todo esto es lo que pretendemos modelar con los diagramas de componentes.

III.2.3.1 Algunos conceptos

Un diagrama de componentes muestra un conjunto de componentes y sus relaciones de manera gráfica a través del uso de nodos y arcos entre estos.

Normalmente los diagramas de componentes contienen:

- Componentes
- Interfaces
- Relaciones de dependencia, generalización, asociaciones y realización.
- Paquetes o subsistemas
- Instancias de algunas clases

Visto de otro modo un diagrama de componentes puede ser un tipo especial de diagrama de clases que se centra en los componentes físicos del sistema.

III.2.3.2 Usos más comunes

a) Modelado de Código Fuente

Los diagramas de componentes se pueden utilizar para modelar la gestión de la configuración de los archivos de código fuente, tomando como productos de trabajo precisamente estos ficheros. Esto resulta bastante útil por ejemplo cuando se han implementado unas partes con Java otras con C, etc. El resultado de esta implementación pueden ser multitud de ficheros ejecutables con características particulares, de manera que la mejor forma de controlarlos es estableciendo gestión de configuración. Para poder llevar a cabo esta gestión con éxito será necesario definir los estereotipos de ficheros que se quieren tener bajo control así como las relaciones entre dichos tipos de ficheros.

Para modelar el código fuente de un sistema:

- Hay que identificar el conjunto de archivos de código fuente de interés y modelarlos como componentes estereotipados como archivos.
- Si el sistema es muy grande es necesario utilizar los paquetes para agrupar los archivos de código fuente.
- Es necesario identificar la versión del componente.

b) Modelado de una versión ejecutable y bibliotecas.

La utilización de los componentes para modelar versiones ejecutables se centra en la definición de todos los elementos que componen lo que se conoce como versión ejecutable, es decir la documentación, los ficheros que se entregan etc.

Para modelar una versión ejecutable es preciso:

- Identificar el conjunto de componentes que se pretende modelar.
- Identificar el estereotipo de cada componente del conjunto seleccionado.
- Para cada componente de este conjunto hay que considerar las relaciones con los vecinos. Esto implica definir las interfaces importadas por ciertos componentes y las exportadas por otros.

c) Modelado de una base de datos física

Para modelar una base de datos física es necesario:

- Identificar las clases del modelo que representan el esquema lógico de la base de datos.
- Seleccionar una estrategia para hacer corresponder las clases con tablas. Así como la distribución física de la/s base/s de datos.
- Para poder visualizar, especificar, construir y documentar dicha correspondencia es necesario crear un diagrama de componentes que tenga componentes estereotipados como tablas.
- Donde sea posible es aconsejable utilizar herramientas que ayuden a transformar diseño lógico en físico.

III.2.4 Diagramas de Despliegue

III.2.4.1 Técnicas más comunes de modelado

a) Modelado de un sistema empotrado

El desarrollo de un sistema empotrado es más que el desarrollo de un sistema software. Hay que manejar el mundo físico. Los diagramas de despliegue son útiles para facilitar la comunicación entre los ingenieros de hardware y los de software.

Para modelar un sistema empotrado es necesario:

- Identificar los dispositivos y nodos propios del sistema.
- Proporcionar señales visuales, sobre todo para los dispositivos poco usuales.
- Modelar las relaciones entre esos procesadores y dispositivos en un diagrama de despliegue.
- Si es necesario hay que detallar cualquier dispositivo inteligente, modelando su estructura en un diagrama de despliegue más pormenorizado.

b) Modelado de un sistema cliente servidor

La división entre cliente y servidor en un sistema es complicada ya que implica tomar algunas decisiones sobre dónde colocar físicamente sus componentes software, qué cantidad de software debe residir en el cliente, etc.

Para modelar un sistema cliente/servidor hay que hace lo siguiente:

- Identificar los nodos que representan los procesadores cliente y servidor del sistema.
- Destacar los dispositivos relacionados con el comportamiento del sistema.
- Proporcionar señales visuales para esos procesadores y dispositivos a través de estereotipos.
- Modelar la tipología de esos nodos mediante un diagrama de despliegue.

III.2.5 Arquitectura del Sistema

III.2.5.1 Arquitectura de tres niveles

La llamada "Arquitectura en Tres Niveles", es la más común en sistemas de información que además de tener una interfaz de usuario contemplan la persistencia de los datos.

Una descripción de los tres niveles sería la siguiente:

Nivel 1: Presentación – ventanas, informes, etc.

Nivel 2: Lógica de la Aplicación – tareas y reglas que gobiernan el proceso.

Nivel 3: Almacenamiento – mecanismo de almacenamiento.

III.2.5.2 Arquitectura de tres niveles orientadas a objetos

a) Descomposición del nivel de lógica de la aplicación

En el diseño orientado a objetos, el nivel de lógica de la aplicación se descompone en sub- niveles que son los siguientes:

Objetos del Dominio: son clases que representan objetos del dominio. Por ejemplo en un problema de ventas, una "Venta" sería un objeto del dominio.

Servicios: se hace referencia a funciones de interacción con la base de datos, informes, comunicaciones, seguridad, etc.

III.2.5.3 Arquitectura MULTI-nivel

La arquitectura de tres niveles puede pasar a llamarse de Múltiples Niveles si tenemos en cuenta el hecho de que todos los niveles de la arquitectura de tres niveles se pueden descomponer cada uno de ellos cada vez más.

Por ejemplo el nivel de Servicios, se puede descomponer en servicios de alto y de bajo nivel, identificando como de alto nivel los servicios de generación de informes y como de bajo nivel los de manejo de ficheros de entrada y salida.

El motivo que lleva a descomponer la arquitectura del sistema en diferentes niveles es múltiple:

- Separación de la lógica de la aplicación en componentes separados que sean más fácilmente reutilizables.
- Distribución de niveles en diferentes nodos físicos de computación.
- Reparto de recursos humanos en diferentes niveles de la arquitectura.

III.2.5.4 Paquetes

La forma que tiene UML de agrupar elementos en subsistemas es a través del uso de **Paquetes**, pudiéndose anidar los paquetes formando jerarquías de paquetes. De hecho un sistema que no tenga necesidad de ser descompuesto en subsistemas se puede considerar como con un único paquete que lo abarca todo.

Gráficamente un paquete viene representado como se indica en la Figura 32.

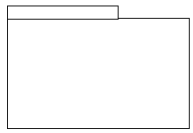


Figura 32 Representación de un paquete en UML

En la Figura 33, vemos cómo se representa la arquitectura del sistema con la notación de paquetes.

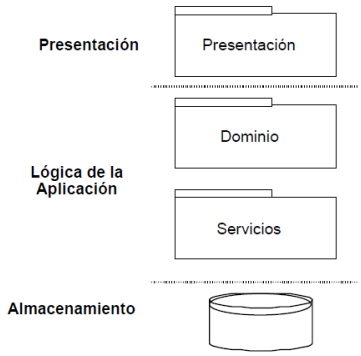


Figura 33 Arquitectura de un sistema utilizando paquetes

III.2.5.5 Identificación de Paquetes

Vamos a definir una serie de reglas que nos pueden ser de utilidad a la hora de agrupar los diferentes elementos en paquetes.

- Conviene agrupar elementos que proporcionen un mismo servicio.
- Los elementos que se agrupen en un mismo paquete han de presentar un alto grado de cohesión, es decir deben estar muy relacionados.
- Los elementos que estén en diferentes paquetes deben tener poca relación, es decir deben colaborar lo menos posible.

IV Desarrollo Orientado a Objetos

IV.1 Proceso de Desarrollo

Cuando se va a construir un sistema software es necesario conocer un lenguaje de programación, pero con eso no basta. Si se quiere que el sistema sea robusto y mantenible es necesario que el problema sea analizado y la solución sea cuidadosamente diseñada. Se debe seguir un proceso robusto, que incluya las actividades principales. Si se sigue un proceso de desarrollo que se ocupa de plantear cómo se realiza el análisis y el diseño, y cómo se relacionan los productos de ambos, entonces la construcción de sistemas software va a poder ser planificable y repetible, y la probabilidad de obtener un sistema de mejor calidad al final del proceso aumenta considerablemente, especialmente cuando se trata de un equipo de desarrollo formado por varias personas.

Para este curso se va a seguir el método de desarrollo orientado a objetos que propone Craig Larman [Larman99]. Este proceso no fija una metodología estricta, sino que define una serie de actividades que pueden realizarse en cada fase, las cuales deben adaptarse según las condiciones del proyecto que se esté llevando a cabo. Se ha escogido seguir este proceso debido a que aplica los últimos avances en Ingeniería del Software, y a que adopta un enfoque eminentemente práctico, aportando soluciones a las principales dudas y/o problemas con los que se enfrenta el desarrollador. Su mayor aportación consiste en atar los cabos sueltos que anteriores métodos dejan.

La notación que se usa para los distintos modelos, tal y como se ha dicho anteriormente, es la proporcionada por UML, que se ha convertido en el estándar de facto en cuanto a notación orientada a objetos. El uso de UML permite integrar con mayor facilidad en el equipo de desarrollo a nuevos miembros y compartir con otros equipos la documentación, pues es de esperar que cualquier desarrollador versado en orientación a objetos conozca y use UML (o se esté planteando su uso).

Se va a abarcar todo el ciclo de vida, empezando por los requisitos y acabando en el sistema funcionando, proporcionando así una visión completa y coherente de la producción de sistemas software. El enfoque que toma es el de un ciclo de vida iterativo incremental, el cual permite una gran flexibilidad a la hora de adaptarlo a un proyecto y a un equipo de desarrollo específicos. El ciclo de vida está dirigido por casos de uso, es decir, por la funcionalidad que ofrece el sistema a los futuros usuarios del mismo. Así no se pierde de vista la motivación principal que debería estar en cualquier proceso de construcción de software: el resolver una necesidad del usuario/cliente.

IV.1.1 Visión General

El proceso a seguir para realizar desarrollo orientado a objetos es complejo, debido a la complejidad que nos vamos a encontrar al intentar desarrollar cualquier sistema software de tamaño medio-alto. El proceso está formado por una serie de actividades y subactividades, cuya realización se va repitiendo en el tiempo aplicadas a distintos elementos.

En este apartado se va a presentar una visión general para poder tener una idea del proceso a alto nivel, y más adelante se verán los pasos que componen cada fase.

Las tres fases al nivel más alto son las siguientes:

- **Planificación y Especificación de Requisitos**: Planificación, definición de requisitos, construcción de prototipos, etc.
- **Construcción**: La construcción del sistema. Las fases dentro de esta etapa son las siguientes:
  - **Análisis**: Se analiza el problema a resolver desde la perspectiva de los usuarios y de las entidades externas que van a solicitar servicios al sistema.
  - **Diseño**: El sistema se especifica en detalle, describiendo cómo va a funcionar internamente para satisfacer lo especificado en el análisis.
  - **Implementación**: Se lleva lo especificado en el diseño a un lenguaje de programación.
  - **Pruebas**: Se llevan a cabo una serie de pruebas para corroborar que el software funciona correctamente y que satisface lo especificado en la etapa de Planificación y Especificación de Requisitos.
- **Instalación**: La puesta en marcha del sistema en el entorno previsto de uso.

De ellas, la fase de Construir es la que va a consumir la mayor parte del esfuerzo y del tiempo en un proyecto de desarrollo. Para llevarla a cabo se va adoptar un enfoque iterativo, tomando en cada iteración un subconjunto de los requisitos (agrupados según casos de uso) y llevándolo a través del análisis y diseño hasta la implementación y pruebas, tal y como se muestra en la Figura 34. El sistema va creciendo incrementalmente en cada ciclo.

Con esta aproximación se consigue disminuir el grado de complejidad que se trata en cada ciclo, y se tiene pronto en el proceso una parte del sistema funcionando que se puede contrastar con el usuario/cliente.

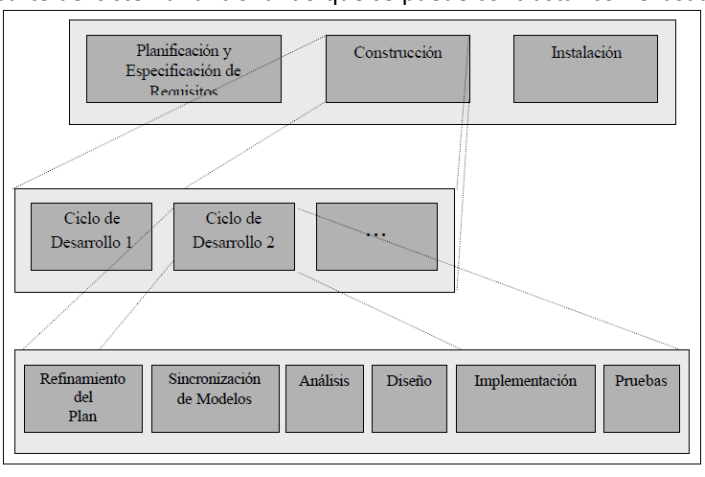


Figura 34 Desarrollo Iterativo en la Construcción

IV.2 Fase de Planificación y Especificación de Requisitos

Esta fase se corresponde con la Especificación de Requisitos tradicional ampliada con un Borrador de Modelo Conceptual y con una definición de Casos de Uso de alto nivel. En esta fase se decidiría si se aborda la construcción del sistema mediante desarrollo orientado a objetos o no, por lo que, en principio, es independiente del paradigma empleado posteriormente.

IV.2.1 Actividades

Las actividades de esta fase son las siguientes:

1. Definir el Plan-Borrador.
2. Crear el Informe de Investigación Preliminar.
3. Definir los Requisitos.
4. Registrar Términos en el Glosario. (*continuado en posteriores fases*)
5. Implementar un Prototipo. (*opcional*)
6. Definir Casos de Uso (de alto nivel y esenciales).
7. Definir el Modelo Conceptual-Borrador. (*puede retrasarse hasta una fase posterior*)
8. Definir la Arquitectura del Sistema-Borrador. (*puede retrasarse hasta una fase posterior*)
9. Refinar el Plan.

El orden propuesto es el que parece más lógico, y en él los pasos 5 y 7 pueden estar en posiciones distintas. De todos modos, el orden no es estricto, lo normal es que las distintas actividades se solapen en el tiempo. Esto sucede también en las actividades de las fases de Análisis y de Diseño, que se verán más adelante.

De estas actividades no se va a entrar en las que corresponden al campo de la planificación de proyectos software, como las correspondientes a creación de planes e informes preliminares.

Tan solo se va a ver por encima la actividad de Definición de Requisitos en cuanto está relacionada con los Casos de Uso, pues son éstos los que van a servir de punto de partida en el Análisis Orientado a Objetos.

IV.2.2 Requisitos

Un requisito es una descripción de necesidades o aspiraciones respecto a un producto. El objetivo principal de la actividad de definición de requisitos consiste en identificar qué es lo que realmente se necesita, separar el grano de la paja. Esto se hace en un modo que sirva de comunicación entre el cliente y el equipo de desarrollo.

Es aconsejable que un documento de Especificación de Requisitos tenga los siguientes puntos:

- Propósito.
- Ámbito del Sistema, Usuarios.
- Funciones del Sistema.
- Atributos del Sistema.

El formato del documento de Especificación de Requisitos no está definido en UML, pero se ha incluido este punto para resaltar que la actividad de definición de requisitos es un paso clave en la creación de cualquier producto software.

Para refinar los requisitos y mejorar la comprensión de los mismos la técnica de casos de uso constituye una valiosa ayuda.

IV.2.3 Casos de Uso

Un Caso de Uso es un documento narrativo que describe la secuencia de eventos de un actor (un agente externo) que usa un sistema para completar un proceso [Jacobson92]. Es una historia o una forma particular de usar un sistema. Los casos de uso no son exactamente requisitos ni especificaciones funcionales, pero ilustran e implican requisitos en las historias que cuentan.

En la página 9 se definió la notación de UML para los Diagramas de Casos de Uso. Nótese que UML no define un formato para describir un caso de uso. Tan sólo define la manera de representar la relación entre actores y casos de uso en un diagrama (Diagrama de Casos de Uso). El formato textual que se va a usar en este texto para definir los caso de uso se va a definir a continuación, mientras que la representación de los escenarios correspondientes a un caso de uso por medio de Diagramas de Secuencia se verá más adelatnte.

En un primer momento interesa abordar un caso de uso desde un nivel de abstracción alto, es lo que se denomina Caso de Uso de Alto Nivel.

IV.2.3.1 Casos de Uso de Alto Nivel

El siguiente Caso de Uso de Alto Nivel describe el proceso de sacar dinero cuando se está usando un cajero automático:

- Caso de Uso: **Realizar Reintegro**

- Actores: Cliente
- Tipo: primario
- Descripción: Un Cliente llega al cajero automático, introduce la tarjeta, se identifica y solicita realizar una operación de reintegro por una cantidad específica. El cajero le da el dinero solicitado tras comprobar que la operación puede realizarse. El Cliente coge el dinero y la tarjeta y se va.

En un caso de uso descrito a alto nivel la descripción es muy general, normalmente se condensa en dos o tres frases. Es útil para comprender el ámbito y el grado de complejidad del sistema.

IV.2.3.2 Casos de Uso Expandidos

Los casos de uso que se consideren los más importantes y que se considere que son los que más influencian al resto, se describen a un nivel más detallado: en el formato expandido.

La principal diferencia con un caso de uso de alto nivel está en que incluye un apartado de *Curso Típico de Eventos*, pero también incluye otros apartados como se ve en el siguiente ejemplo:

- Caso de Uso: **Realizar Reintegro**
- Actores: Cliente (iniciador)
- Propósito: Realizar una operación de reintegro de una cuenta del banco.
- Visión General: Un Cliente llega al cajero automático, introduce la tarjeta, se identifica y solicita realizar una operación de reintegro por una cantidad específica. El cajero le da el dinero solicitado tras comprobar que la operación puede realizarse. El Cliente coge el dinero y la tarjeta y se va.
- Tipo: primario y esencial
- Referencias: *Funciones:* R1.3, R1.7
- Curso Típico de Eventos:

Acción del Actor	Respuesta del Sistema
1. Este caso de uso empieza cuando un Cliente introduce una tarjeta en el cajero.	2. Pide la clave de identificación.
3. Introduce la clave.	4. Presenta las opciones de operaciones disponibles.
5. Selecciona la operación de Reintegro.	6. Pide la cantidad a retirar.
7. Introduce la cantidad requerida.	8. Procesa la petición y, eventualmente, da el dinero solicitado. Devuelve la tarjeta y genera un recibo.
9. Recoge la tarjeta. 10. Recoge el recibo. 11. Recoge el dinero y se va.	
Cursos Alternativos: • Línea 4: La clave es incorrecta. Se indica el error y se cancela la operación. • Línea 8: La cantidad solicitada supera el saldo. Se indica el error y se cancela la operación.	

El significado de cada apartado de este formato es como sigue:

- Caso de Uso: **Nombre del Caso de Uso**
- Actores: Lista de actores (agentes externos), indicando quién inicia el caso de uso. Los actores son

normalmente roles que un ser humano desempeña, pero puede ser cualquier tipo de sistema.

- Propósito: Intención del caso de uso.

- Visión General: Repetición del caso de uso de alto nivel, o un resumen similar.

- Tipo: 1. primario, secundario u opcional (descritos más adelante).
  2. esencial o real (descritos más adelante).

- Referencias: Casos de uso relacionados y funciones del sistema que aparecen en los requisitos.

- Curso Típico de Eventos: Descripción de la interacción entre los actores y el sistema mediante las acciones numeradas de cada uno. Describe la secuencia más común de eventos, cuando todo va bien y el proceso se completa satisfactoriamente. En caso de haber alternativas con grado similar de probabilidad se pueden añadir secciones adicionales a la sección principal, como se verá más adelante.

- Cursos Alternativos: Puntos en los que puede surgir una alternativa, junto con la descripción de la excepción.

IV.2.3.3 Identificación de Casos de Uso

La identificación de casos de uso requiere un conocimiento medio acerca de los requisitos, y se basa en la revisión de los documentos de requisitos existentes, y en el uso de la técnica de *brainstorming* entre los miembros del equipo de desarrollo.

Como guía para la identificación inicial de casos de uso hay dos métodos:

a) Basado en Actores

1. Identificar los actores relacionados con el sistema y/o la organización.
2. Para cada actor, identificar los procesos que inicia o en los que participa.

b) Basado en Eventos

1. Identificar los eventos externos a los que el sistema va a tener que responder.
2. Relacionar los eventos con actores y casos de uso.

Ejemplos de casos de uso:

- Pedir un producto.
- Matricularse en un curso de la facultad.
- Comprobar la ortografía de un documento en un procesador de textos.
- Realizar una llamada telefónica.

IV.2.3.4 Identificación de los Límites del Sistema

En la descripción de un caso de uso se hace referencia en todo momento al "sistema". Para que los casos de uso tengan un significado completo es necesario que el sistema esté definido con precisión.

Al definir los límites del sistema se establece una diferenciación entre lo que es interno y lo que es externo al sistema. El entorno exterior se representa mediante los actores.

Ejemplos de sistemas son:

- El hardware y software de un sistema informático.
- Un departamento de una organización.
- Una organización entera.

Si no se está haciendo reingeniería del proceso de negocio lo más normal es escoger como sistema el primero de los ejemplos: el hardware y el software del sistema que se quiere construir.

IV.2.3.5 Tipos de Casos de Uso

a) Según Importancia

Para poder priorizar los casos de uso que identifiquemos los vamos a distinguir entre:

- **Primarios**: Representan los procesos principales, los más comunes, como *Realizar Reintegro* en el caso del cajero automático.
- **Secundarios**: Representan casos de uso menores, que van a necesitarse raramente, tales como *Añadir Nueva Operación*.
- **Opcionales**: Representan procesos que pueden no ser abordados en el presente proyecto.

b) Según el Grado de Compromiso con el Diseño

En las descripciones que se han visto anteriormente no se han hecho apenas compromisos con la solución, se han descrito los casos de uso a un nivel abstracto, independiente de la tecnología y de la implementación. Un caso de uso definido a nivel abstracto se denomina **esencial**. Los casos de uso definidos a alto nivel son siempre esenciales por naturaleza, debido a su brevedad y abstracción.

Por el contrario, un caso de uso **real** describe concretamente el proceso en términos del diseño real, de la solución específica que se va a llevar a cabo. Se ajusta a un tipo de interfaz específica, y se baja a detalles como pantallas y objetos en las mismas.
Como ejemplo de una parte de un Caso de Uso Real para el caso del reintegro en un cajero automático tenemos la siguiente descripción del Curso Típico de Eventos:

Acción del Actor	Respuesta del Sistema
1. Este caso de uso empieza cuando un Cliente introduce una tarjeta en la ranura para tarjetas.	2. Pide el PIN (*Personal Identification Number*).
3. Introduce el PIN a través del teclado numérico.	4. Presenta las opciones de operaciones disponibles.
5. etc.	6. etc.

En principio, los casos de uso reales deberían ser creados en la fase de Diseño y no antes, puesto que se trata de elementos de diseño. Sin embargo, en algunos proyectos se plantea la definición de interfaces en fases tempranas del ciclo de desarrollo, en base a que son parte del contrato. En este caso se pueden definir algunos o todos los casos de uso reales, a pesar de que suponen tomar decisiones de diseño muy pronto en el ciclo de vida.
No hay una diferencia estricta entre un Caso de Uso Esencial y uno Real, el grado de compromiso con el Diseño es un continuo, y una descripción específica de un caso de uso estará situada en algún punto de la línea entre Casos de Uso Esenciales y Reales, normalmente más cercano a un extremo que al otro, pero es raro encontrar Casos de Uso Esenciales o Reales puros.

IV.2.3.6 Consejos Relativos a Casos de Uso

a) Nombre

El nombre de un Caso de Uso debería ser un verbo, para enfatizar que se trata de un proceso, por ejemplo: *Comprar Artículos* o *Realizar Pedido*.

b) Alternativas equiprobables

Cuando se tiene una alternativa que ocurre de manera relativamente ocasional, se indica en el apartado *Cursos Alternativos*. Pero cuando se tienen distintas opciones, todas ellas consideradas normales se puede completar el *Curso Típico de Eventos* con secciones adicionales.
Así, si en un determinado número de línea hay una bifurcación se pueden poner opciones que dirigen el caso de uso a una sección que se detalla al final del Curso Típico de Eventos, en la siguiente forma:

- Curso Típico de Eventos:
  - Sección: Principal

Acción del Actor	Respuesta del Sistema
1. Este caso de uso empieza cuando *Actor* llega al sistema.	2. Pide la operación a realizar.
3. Escoge la operación *A*.	4. Presenta las opciones de pago.
5. Selecciona el tipo de pago: a. Si se paga al contado ver sección *Pago al Contado*. b. Si se paga con tarjeta ver sección *Pago con Tarjeta*	6. Genera recibo.
7. Recoge el recibo y se va.	
Cursos Alternativos: • Líneas 3 y 5: Selecciona *Cancelar*. Se cancela la operación.	

  - Sección: Pago al Contado

Acción del Actor	Respuesta del Sistema
1. Mete los billetes correspondientes	2. Coge los billetes y sigue pidiendo dinero hasta que la cantidad está satisfecha 3. Devuelve el cambio.
Cursos Alternativos: • Línea 3: No hay cambio suficiente. Se cancela la operación.	

  - Sección: Pago con Tarjeta

...

IV.2.4 Construcción del Modelo de Casos de Uso

Para construir el Modelo de Casos de Uso en la fase de Planificación y Especificación de Requisitos se siguen los siguientes pasos:

1. Después de listar las funciones del sistema, se definen los límites del sistema y se identifican los actores y los casos de uso.
2. Se escriben todos los casos de uso en el formato de *alto nivel*. Se categorizan como primarios, secundarios u opcionales.
3. Se dibuja el Diagrama de Casos de Uso.
4. Se relacionan los casos de uso y se ilustran las relaciones en el Diagrama de Casos de Uso (<<extiende>> y <<usa>>).
5. Los casos de uso más críticos, importantes y que conllevan un mayor riesgo, se describen en el formato expandido esencial. Se deja la definición en formato expandido esencial del resto de casos de uso para cuando sean tratados en posteriores ciclos de desarrollo, para no tratar toda la complejidad del problema de una sola vez.
6. Se crean casos de uso reales sólo cuando:
   - Descripciones más detalladas ayudan significativamente a incrementar la comprensión del problema.
   - El cliente pide que los procesos se describan de esta forma.
7. Ordenar según prioridad los casos de uso (este paso se va a ver a continuación).

IV.2.5 Planificación de Casos de Uso según Ciclos de Desarrollo

La decisión de qué partes del sistema abordar en cada ciclo de desarrollo se va a tomar basándose en los casos de uso. Esto es, a cada ciclo de desarrollo se le va a asignar la implementación de uno o más casos de uso, o versiones simplificadas de casos de uso. Se asigna una versión simplificada cuando el caso de uso completo es demasiado complejo para ser tratado en un solo ciclo (ver Figura 35).

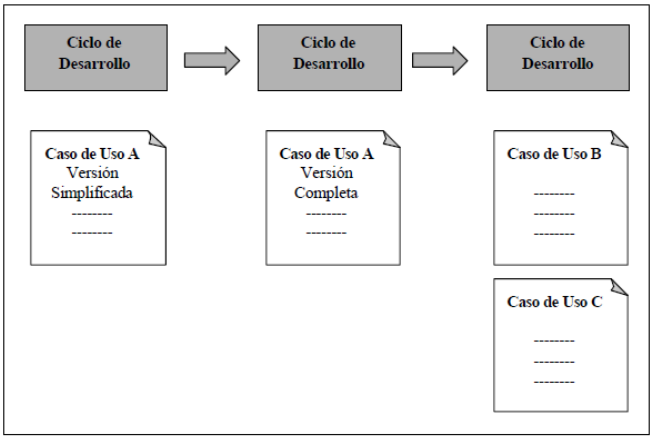


Figura 35 Planificación de Ciclos de Desarrollo según Casos de Uso

Para tomar la decisión de qué casos de uso se van a tratar primero es necesario ordenarlos según prioridad. Las características de un caso de uso específico que van a hacer que un caso de uso tenga una prioridad alta son las siguientes:

a. Impacto significativo en el diseño de la arquitectura. Por ejemplo, si aporta muchas clases al modelo del dominio o requiere persistencia en los datos.
b. Se obtiene una mejor comprensión del diseño con un nivel de esfuerzo relativamente bajo. c. Incluye funciones complejas, críticas en el tiempo o de nivel elevado de riesgo.
c. Incluye funciones complejas, críticas en el tiempo o de nivel elevado de riesgo.
d. Implica bien un trabajo de investigación significante, o bien el uso de una tecnología nueva o arriesgada.
e. Representa un proceso de gran importancia en la línea de negocio.
f. Supone directamente un aumento de beneficios o una disminución de costes.

Para realizar la clasificación se puede asignar a cada caso de uso una valoración numérica de cada uno de estos puntos, para conseguir una puntuación total aplicando pesos a cada apartado. En la siguiente tabla se muestra un ejemplo de tal tipo de clasificación:

Peso	3	2	4	1	3	4	Suma
Caso de Uso	**a**	**b**	**c**	**d**	**e**	**f**	
Reintegro	5	4	1	0	5	2	50
...							

IV.2.5.1 Caso de Uso *Inicialización*

Prácticamente todos los sistemas van a tener un caso de uso *Inicialización*. Aunque puede ser que no tenga una prioridad alta en la clasificación realizada según el punto anterior, normalmente va a interesar que sea desarrollado desde el principio. Inicialmente se desarrolla una versión simplificada, que se va completando en cada ciclo de desarrollo para satisfacer las necesidades de inicialización de los casos de uso que se tratan en dicho ciclo. Así se tiene un sistema en cada ciclo de desarrollo que puede funcionar.

IV.3 Fase de Construcción: Análisis

En la fase de Análisis de un ciclo de desarrollo se *investiga* sobre el problema, sobre los conceptos relacionados con el subconjunto de casos de uso que se esté tratando. Se intenta llegar a una buena comprensión del problema por parte del equipo de desarrollo, sin entrar en cómo va a ser la solución en cuanto a detalles de implementación.

Cuando el ciclo de desarrollo no es el primero, antes de la fase de Análisis hay una serie de actividades de planificación. Estas actividades consisten en actualizar los modelos que se tengan según lo que se haya implementado, pues siempre se producen desviaciones entre lo que se ha analizado y diseñado y lo que finalmente se construye. Una vez se tienen los modelos acordes con lo implementado se empieza el nuevo ciclo de desarrollo con la fase de Análisis.

En esta fase se trabaja con los modelos de Análisis construidos en la fase anterior, ampliándolos con los conceptos correspondientes a los casos de uso que se traten en el ciclo de desarrollo actual.

IV.3.1 Actividades

Las actividades de la fase de Análisis son las siguientes:

1. Definir Casos de Uso Esenciales en formato expandido. (*si no están definidos*)
2. Refinar los Diagramas de Casos de Uso.
3. Refinar el Modelo Conceptual.
4. Refinar el Glosario. (*continuado en posteriores fases*)
5. Definir los Diagramas de Secuencia del Sistema.
6. Definir Contratos de Operación.
7. Definir Diagramas de Estados. (*opcional*)

IV.3.2 Modelo Conceptual

Una parte de la investigación sobre el dominio del problema consiste en identificar los conceptos que lo conforman. Para representar estos conceptos se va a usar un Diagrama de Estructura Estática de UML, al que se va a llamar Modelo Conceptual.

En el Modelo Conceptual se tiene una representación de conceptos del mundo real, no de componentes software.

El objetivo de la creación de un Modelo Conceptual es aumentar la comprensión del problema. Por tanto, a la hora de incluir conceptos en el modelo, es mejor crear un modelo con muchos conceptos que quedarse corto y olvidar algún concepto importante.

IV.3.2.1 Identificación de Conceptos

Para identificar conceptos hay que basarse en el documento de Especificación de Requisitos y en el conocimiento general acerca del dominio del problema.

En la Tabla 1 se muestran algunas categorías típicas, junto con ejemplos pertenecientes al dominio de los supermercados y al de la reserva de billetes de avión:

Tabla 1 Lista de Conceptos Típicos

Tipo de Concepto	Ejemplos
Objetos físicos o tangibles	Avión Terminal_de_Caja
Especificaciones, diseños o descripciones de cosas	Especificación_de_Producto Descripción_de_Vuelo
Lugares	Supermercado Aeropuerto
Transacciones	Venta, Pago Reserva
Líneas de una transacción	Artículo_de_Venta
Roles de una persona	Cajero Piloto
Contenedores de otras cosas	Supermercado, Cesta Avión
Cosas en un contenedor	Artículo Pasajero
Otros ordenadores o sistemas electromecánicos externos a nuestro sistema	Sistema_de_Autorización_de_Tarjetas_de Crédito Sistema_Controlador_de_Tráfico_Aéreo
Conceptos abstractos	Hambre
Organizaciones	Departamento_de_Ventas Compañía_Aérea_Toto
Eventos	Venta, Robo, Reunión Vuelo, Accidente, Aterrizaje
Reglas y políticas	Política_de_Devoluciones Política_de_Cancelaciones
Catálogos	Catálogo_de_Productos Catálogo_de_Piezas

Otro consejo para identificar conceptos consiste en buscar sustantivos en los documentos de requisitos o, más concretamente, en la descripción de los casos de uso. No es un método infalible, pero puede servir de guía para empezar.

Para poner nombre a los conceptos se puede usar la analogía con el cartógrafo, resumida en los siguientes tres puntos:

- *Usar los nombres existentes en el territorio*: Hay que usar el vocabulario del dominio para nombrar conceptos y atributos.
- *Excluir características irrelevantes*: Al igual que el cartógrafo elimina características no relevantes según la finalidad del mapa (por ejemplo datos de población en un mapa de carreteras), un Modelo Conceptual puede excluir conceptos en el dominio que no son pertinentes en base a los requisitos.
- *No añadir cosas que no están ahí*: Si algo no pertenece al dominio del problema no se añade al modelo.

IV.3.2.2 Creación del Modelo Conceptual

Para crear el Modelo Conceptual se siguen los siguientes pasos:

1. Hacer una lista de conceptos candidato usando la Lista de Categorías de Conceptos de la Tabla 1 y la búsqueda de sustantivos relacionados con los requisitos en consideración en este ciclo.
2. Representarlos en un diagrama.
3. Añadir las asociaciones necesarias para ilustrar las relaciones entre conceptos que es necesario conocer.
4. Añadir los atributos necesarios para contener toda la información que se necesite conocer de cada concepto.

IV.3.2.3 Identificación de Asociaciones

Una asociación es una relación entre conceptos que indica una conexión con sentido y que es de interés en el conjunto de casos de uso que se está tratando.

Se incluyen en el modelo las asociaciones siguientes:

- Asociaciones para las que el conocimiento de la relación necesita mantenerse por un cierto período de tiempo (asociaciones "necesita-conocer").
- Asociaciones derivadas de la Lista de Asociaciones Típicas que se muestra en la Tabla 2.

Tabla 2 Lista de Asociaciones Típicas

Tabla 2 Lista de Asociaciones Típicas

Categoría	Ejemplos
A es una parte física de B	Ala – Avión
A es una parte lógica de B	Artículo_en_Venta –Venta
A está físicamente contenido en B	Artículo – Estantería
	Pasajero – Avión
A está lógicamente contenido en B	Descripción_de_Artículo – Catálogo
A es una descripción de B	Descripción_de_Artículo – Artículo
A es un elemento en una transacción o un informe B	Trabajo_de_Reparación – Registro_de_Reparaciones
A es registrado/archivado/capturado en B	Venta – Terminal_de_Caja
A es un miembro de B	Cajero – Supermercado Piloto – Compañía_Aerea
A es una subunidad organizativa de B	Sección – Supermercado Mantenimiento – Compañía_Aerea
A usa o gestiona B	Cajero – Terminal_de_Caja Piloto – Avión
A comunica con B	Cliente – Cajero Empleado_de_Agencia_de_Viajes – Pasajero
A está relacionado con una transacción B	Cliente – Pago Pasajero – Billete
A es una transacción relacionada con otra transacción B	Pago – Venta Reserva – Cancelación
A está junto a B	Ciudad – Ciudad
A posee B	Supermercado – Terminal_de_Caja Compañía_Aérea – Avión

IV.3.3 Glosario

En el glosario debe aparecer una descripción textual de cualquier elemento de cualquier modelo, para eliminar toda posible ambigüedad. Un formato tipo para el glosario es el que se muestra en la Tabla 3.

Tabla 3 Formato tipo de Glosario

Término	Categoría	Descripción
Realizar Reintegro	caso de uso	Descripción del proceso por el que un Cliente realiza un reintegro en un cajero automático.
Banco	concepto	Entidad que ofrece servicios financieros a sus clientes.
...	...	...

IV.3.4 Diagramas de Secuencia del Sistema

Además de investigar sobre los conceptos del sistema y su estructura, también es preciso investigar en el Análisis sobre el comportamiento del sistema, visto éste como una caja negra. Una parte de la descripción del comportamiento del sistema se realiza mediante los Diagramas de Secuencia del Sistema.

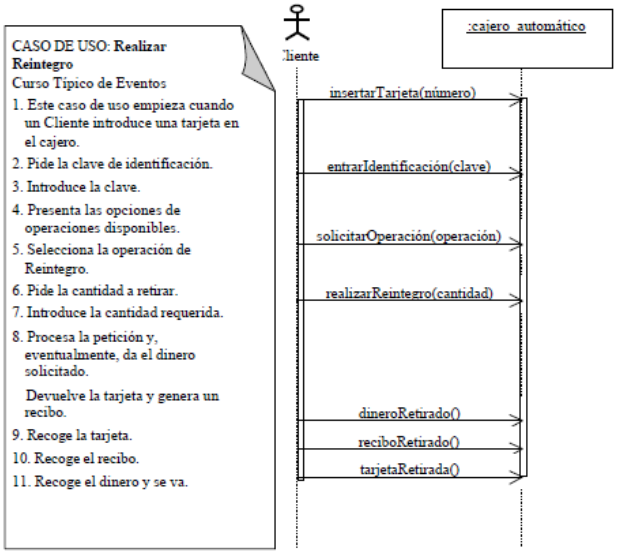


Figura 36 Ejemplo de Diagrama de Secuencia del Sistema

En cada caso de uso se muestra una interacción de actores con el sistema. En esta interacción los actores generan eventos, solicitando al sistema operaciones. Por ejemplo, en el caso de una reserva de un billete de avión, el empleado de la agencia de viajes solicita al sistema de reservas que realice una reserva. El evento que supone esa solicitud inicia una operación en el sistema de reservas.

Los casos de uso representan una interacción genérica. Una instancia de un caso de uso se denomina **escenario**, y muestra una ejecución real del caso de uso, con las posibles bifurcaciones y alternativas resueltas de forma particular.

Un Diagrama de Secuencia de Sistema se representa usando la notación para diagramas de secuencia de UML (ver página 10). En él se muestra para un escenario particular de un caso de uso los eventos que los actores generan, su orden, y los eventos que se intercambian entre sistemas.

Para cada caso de uso que se esté tratando se realiza un diagrama para el curso típico de eventos, y además se realiza un diagrama para los cursos alternativos de mayor interés.

En la Figura 36 se muestra el Diagrama de Secuencia del Sistema para el caso de uso Realizar Reintegro de un cajero automático.

IV.3.4.1 Construcción de un Diagrama de Secuencia del Sistema

Para construir un Diagrama de Secuencia del Sistema para el curso típico de eventos de un caso de uso, se siguen los siguientes pasos:

1. Representar el sistema como un objeto con una línea debajo.
2. Identificar los actores que directamente operan con el sistema, y dibujar una línea para cada uno de ellos.
3. Partiendo del texto del curso típico de eventos del caso de uso, identificar los eventos (externos) del sistema que cada actor genera y representarlos en el diagrama.
4. Opcionalmente, incluir el texto del caso de uso en el margen del diagrama.

Los eventos del sistema deberían expresarse en base a la noción de operación que representan, en vez de en base a la interfaz particular. Por ejemplo, se prefiere "finOperación" a "presionadaTeclaEnter", porque captura la finalidad de la operación sin realizar compromisos en cuanto a la interfaz usada.

IV.3.5 Contratos de Operaciones

Una vez se tienen las Operaciones del Sistema identificadas en los Diagramas de Secuencia, se describe mediante contratos el comportamiento esperado del sistema en cada operación.

Un Contrato es un documento que describe qué es lo que se espera de una operación. Tiene una redacción en estilo declarativo, enfatizando en el *qué* más que en el *cómo*. Lo más común es expresar los contratos en forma de pre- y post-condiciones en torno a cambios de estado.

Se puede escribir un contrato para un método individual de una clase software, o para una operación del sistema completa. En este punto se verá únicamente éste último caso.

Un Contrato de Operación del Sistema describe cambios en el estado del sistema cuando una operación del sistema es invocada.

A continuación se ve un ejemplo de Contrato:

Contrato

Nombre:	insertarTarjeta (número_tarjeta: número)
Responsabilidades:	Comenzar una sesión con el sistema para realizar una operación. Presentar las opciones disponibles.
Referencias Cruzadas:	Funciones del Sistema: R1.2, R1.6, R1.7 Casos de Uso: Reintegro
Notas:	
Excepciones:	Si la tarjeta es ilegible, indicar que ha habido un error.
Salida:	
Pre-condiciones:	No hay una sesión activa.
Post-condiciones:	• Una nueva *Sesión* se ha creado. (*creación de instancia*). • La *Sesión* se ha asociado con *Cajero*. (*asociación formada*).

La descripción de cada apartado de un contrato es como sigue:

Nombre:	Nombre de la operación y parámetros.
Responsabilidades:	Una descripción informal de las responsabilidades que la operación debe desempeñar.
Referencias Cruzadas:	Números de referencia en los requisitos de funciones del sistema, casos de uso, etc.
Notas:	Comentarios de diseño, algoritmos, etc.
Excepciones:	Casos excepcionales. Situaciones que debemos tener en cuenta que pueden pasar. Se indica también qué se hace cuando ocurre la excepción.
Salida:	Salidas que no corresponden a la interfaz de usuario, como mensajes o registros que se envían fuera del sistema. (En la mayor parte de las operaciones del sistema este apartado queda vacio)
Pre-condiciones:	Asunciones acerca del estado del sistema antes de ejecutar la operación. Algo que no tenemos en cuenta que pueda ocurrir cuando se llama a esta operación del sistema.
Post-condiciones:	El estado del sistema después de completar la operación.

IV.3.5.1 Construcción de un Contrato

Los pasos a seguir para construir un contrato son los siguientes:

1. Identificar las operaciones del sistema a partir de los Diagramas de Secuencia del Sistema.
2. Para cada operación del sistema construir un contrato.
3. Empezar escribiendo el apartado de *Responsabilidades*, describiendo informalmente el propósito de la operación. Este es el apartado más importante del contrato.
4. A continuación rellenar el apartado de *Post-condiciones*, describiendo declarativamente los cambios de estado que sufren los objetos en el Modelo Conceptual. Puede ser que este apartado quede vacío si no cambia el valor de ningún dato de los maneja el sistema (por ejemplo en una operación del sistema que tan solo se encarga de sacar por pantalla algo al usuario).
5. Para describir las post-condiciones, usar las siguientes categorías:
   - Creación y borrado de instancias.
   - Modificación de atributos.
   - Asociaciones formadas y retiradas.
6. Completar el resto de apartados en su caso.

IV.3.5.2 Post-condiciones

Las post-condiciones se basan en el Modelo Conceptual, en los cambios que sufren los elementos del mismo una vez se ha realizado la operación.

Es mejor usar el tiempo pasado o el pretérito perfecto al redactar una post-condición, para enfatizar que se trata de declaraciones sobre un cambio en el estado que ya ha pasado. Por ejemplo es mejor decir "se ha creado una *Sesión*" que decir "crear una *Sesión*".

Cuando se ha creado un objeto, lo normal es que se haya asociado a algún otro objeto ya existente, porque si no queda aislado del resto del sistema. Por tanto, al escribir las post- condiciones hay que acordarse de añadir asociaciones a los objetos creados. Olvidar incluir estas asociaciones es el fallo más común cometido al escribir las post-condiciones de un contrato.

IV.3.6 Diagramas de Estados

Para modelar el comportamiento del sistema pueden usarse los Diagramas de Estados que define UML (ver página 12).

Se puede aplicar un Diagrama de Estados al comportamiento de los siguientes elementos:
- Una clase software.
- Un concepto.
- Un caso de uso.

En la fase de Análisis sólo se haría para los dos últimos tipos de elemento, pues una clase software pertenece al Diagrama de Clases de Diseño.
Puesto que el sistema entero puede ser representado por un concepto, también se puede modelar el comportamiento del sistema completo mediante un Diagrama de Estados.
La utilidad de un Diagrama de Estados en el análisis reside en mostrar la secuencia permitida de eventos externos que pueden ser reconocidos y tratados por el sistema. Por ejemplo, no se puede insertar una tarjeta en un cajero automático si se está en el transcurso de una operación.
Para los casos de uso complejos se puede construir un Diagrama de Estados. El Diagrama de Estados del sistema sería una combinación de los diagramas de todos los casos de uso.
El uso de Diagramas de Estados es opcional. Tan solo los usaremos cuando consideremos que nos ayudan a expresar mejor el comportamiento del elemento descrito.

IV.4 Fase de Construcción: Diseño

En la fase de Diseño se crea una solución a nivel lógico para satisfacer los requisitos, basándose en el conocimiento reunido en la fase de Análisis.

IV.4.1 Actividades

Las actividades que se realizan en la etapa de Diseño son las siguientes:
1. Definir los Casos de Uso Reales.
2. Definir Informes e Interfaz de Usuario.
3. Refinar la Arquitectura del Sistema.
4. Definir los Diagramas de Interacción.
5. Definir el Diagrama de Clases de Diseño. (*en paralelo con los Diagramas de Interacción*)
6. Definir el Esquema de Base de Datos.

El paso de Refinar la Arquitectura del Sistema no tiene por qué realizarse en la posición 3, puede realizarse antes o después.

IV.4.2 Casos de Uso Reales

Un Caso de Uso Real describe el diseño real del caso de uso según una tecnología concreta de entrada y de salida y su implementación. Si el caso de uso implica una interfaz de usuario, el caso de uso real incluirá bocetos de las ventanas y detalles de la interacción a bajo nivel con los *widgets* (botón, lista seleccionable, campo editable, etc.) de la ventana.
Como alternativa a la creación de los Casos de Uso Reales, el desarrollador puede crear bocetos de la interfaz en papel, y dejar los detalles para la fase de implementación.

IV.4.3 Diagramas de Colaboración

Los Diagramas de Interacción muestran el intercambio de mensajes entre instancias del modelo de clases para cumplir las post-condiciones establecidas en un contrato.
Hay dos clases de Diagramas de Interacción:
1. Diagramas de Colaboración.
2. Diagramas de Secuencia.

La notación en UML para ambos es la definida en la página 10.
De entre ambos tipos se prefieren los Diagramas de Colaboración por su expresividad y por su economía espacial (una interacción compleja puede ser muy larga en un Diagrama de Secuencia).
La creación de los Diagramas de Colaboración de un sistema es una de las actividades más importantes en el desarrollo orientado a objetos, pues al construirlos se toman unas decisiones clave acerca del funcionamiento del futuro sistema. La creación de estos diagramas, por tanto, debería ocupar un porcentaje significativo en el esfuerzo dedicado al proyecto entero.

IV.4.3.1 Creación de Diagramas de Colaboración

Para crear los Diagramas de Colaboración se pueden seguir los siguientes consejos:
- Crear un diagrama separado para cada operación del sistema en desarrollo en el ciclo de desarrollo actual.
  - Para cada evento del sistema, hacer un diagrama con él como mensaje inicial.
- Si el diagrama se complica, dividirlo en diagramas más pequeños.
- Usando los apartados de responsabilidades y de post-condiciones del contrato de operación, y la descripción del caso de uso como punto de partida, diseñar un sistema de objetos que interaccionan para llevar a cabo las tareas requeridas.

La capacidad de realizar una buena asignación de responsabilidades a los distintos objetos es una habilidad clave, y se va adquiriendo según aumenta la experiencia en el desarrollo orientado a objetos.

Booch, Rumbaugh y Jacobson definen **responsabilidad** como "un contrato u obligación de una clase o tipo"[BJR97]. Las responsabilidades están ligadas a las obligaciones de un objeto en cuanto a su comportamiento. Básicamente, estas responsabilidades son de los dos siguientes tipos:

- Conocer:
  - Conocer datos privados encapsulados.
  - Conocer los objetos relacionados.
  - Conocer las cosas que puede calcular o derivar.
- Hacer:
  - Hacer algo él mismo.
  - Iniciar una acción en otros objetos.
  - Controlar y coordinar actividades en otros objetos.

Por ejemplo, puedo decir que "un *Recibo* es responsable de imprimirse" (tipo hacer), o que "una *Transacción* es responsable de saber su fecha" (tipo conocer). Las responsabilidades de tipo "conocer" se pueden inferir normalmente del Modelo Conceptual.

Una responsabilidad no es lo mismo que un método, pero los métodos se implementan para satisfacer responsabilidades.

IV.4.4 Diagrama de Clases de Diseño

Al construir los Diagramas de Colaboración se van usando clases procedentes del Modelo Conceptual, junto con otras creadas para encargarse de responsabilidades específicas. El conjunto de todas las clases usadas, junto con sus relaciones, forma el Diagrama de Clases de Diseño.

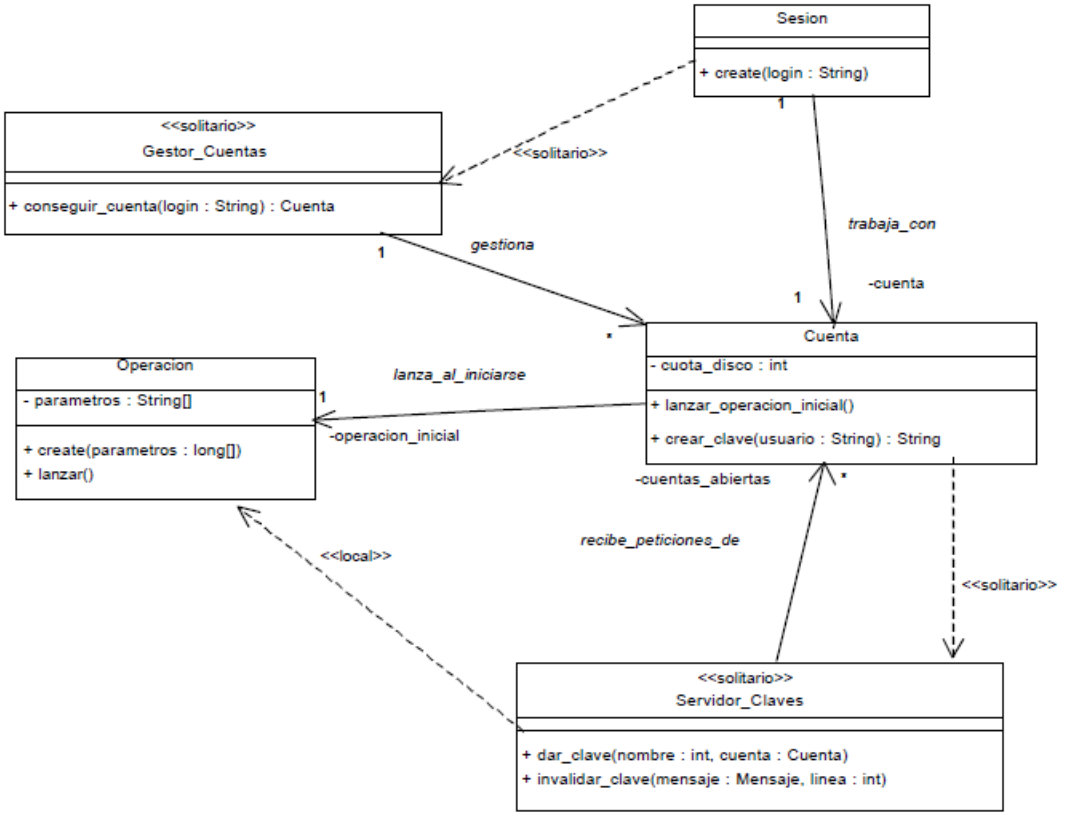


Figura 37 Ejemplo de Diagrama de Clases de Diseño

Un Diagrama de Clases de Diseño muestra la especificación para las clases software de una aplicación. Incluye la siguiente información:

- Clases, asociaciones y atributos.
- Interfaces, con sus operaciones y constantes.
- Métodos.
- Navegabilidad.
- Dependencias.

A diferencia del Modelo Conceptual, un Diagrama de Clases de Diseño muestra definiciones de entidades software más que conceptos del mundo real. En la Figura 37 se muestra un ejemplo de Diagrama de Clases de Diseño sencillo.

IV.4.4.1 Relaciones de Dependencia para Representar Visibilidad entre Clases Cuando una clase conoce a otra por un medio que no es a través de un atributo (una asociación con la navegabilidad adecuada), entonces es preciso indicar esta situación por medio de una dependencia. Un objeto debe conocer a otro para poder llamar a uno de sus métodos, se dice entonces que el primer objeto tiene "visibilidad" sobre el segundo. La visibilidad más directa es por medio de atributo, cuando hay una asociación entre ambas clases y se puede navegar de la primera a la segunda (un atributo de la primera es un puntero a un objeto de la segunda). Hay otros tres tipos de visibilidad que hay que representar en el diagrama de clases mediante relaciones de dependencia:

- **Parámetro**: Cuando a un método de una clase se le pasa como parámetro un objeto de otra clase, se dice que la primera tiene visibilidad de parámetro sobre la segunda. La relación de dependencia entre ambas clases se etiqueta con el estereotipo <<parámetro>> (<<*parameter*>> en inglés).
- **Local**: Cuando en un método de una clase se define una variable local que es un objeto de otra clase, se dice que la primera tiene visibilidad local sobre la segunda. La relación de dependencia entre ambas clases se etiqueta con el estereotipo <<local>>.
- **Global**: Cuando hay una variable global en el sistema, y un método de una clase llama a un método de esa variable global, se dice que la clase tiene visibilidad global sobre la clase a la que pertenece la instancia que es una variable global. La relación de dependencia entre ambas clases se etiqueta con el estereotipo <<global>>.

No es necesario representar la relación de dependencia entre clases que ya están relacionadas por medio de una asociación, que se trata de una "dependencia" más fuerte. Las relaciones de dependencia se incluyen tan solo para conocer qué elementos hay que revisar cuando se realiza un cambio en el diseño de un elemento del sistema.

a) Solitario: Caso Particular de Visibilidad global

El uso de variables globales no se aconseja por los efectos laterales que se pueden presentar, pero hay un caso en el que sí hay cierta globalidad: las clases que sólo van a tener una instancia. Suele tratarse de clases que se han creado en el diseño, que no aparecían en el Modelo Conceptual.

Varias clases de nuestro sistema pueden querer llamar a los métodos de la única instancia de una clase de ese tipo, entonces sí se considera que es beneficioso que se pueda acceder a esa instancia como un valor accesible de forma global. Una solución elegante para este caso se implementa mediante un método de clase para obtener esa única instancia (los métodos de clase los permiten algunos lenguajes orientados a objetos, por ejemplo Java), en vez de definirla directamente como una variable global. Para indicar que una clase sólo va a tener una instancia, se etiqueta la clase con el estereotipo <<solitario>> (<<*singleton*>> en inglés), y las relaciones de dependencia entre las clases que la usan y se etiquetan también <<solitario>> en vez de <<global>>.

A continuación se muestra un ejemplo del código en Java de una clase solitario:

```
public class Solitario {
// se define la instancia como atributo de clase (static) Solitario static instancia := null;
// método de clase que devuelve la instancia public static Solitario
dar_instancia() {
        if (instancia = = null) {
      // si no está creada la instancia la crea instancia := new Solitario();
        }
        return instancia;
} ... // otros métodos de la clase
}
```

Cuando otra clase quiere llamar a un método de la instancia incluye el siguiente código:

```
variable Solitario;
variable = Solitario.dar_instancia();
variable.método(); // llamada al método que necesitamos
```

IV.4.4.2 Construcción de un Diagrama de Clases de Diseño

Para crear un Diagrama de Clases de Diseño se puede seguir la siguiente estrategia:

1. Identificar todas las clases participantes en la solución software. Esto se lleva a cabo analizando los Diagramas de Interacción.
2. Representarlas en un diagrama de clases.
3. Duplicar los atributos que aparezcan en los conceptos asociados del Modelo Conceptual.
4. Añadir los métodos, según aparecen en los Diagramas de Interacción.
5. Añadir información de tipo a los atributos y métodos.
6. Añadir las asociaciones necesarias para soportar la visibilidad de atributos requerida.
7. Añadir flechas de navegabilidad a las asociaciones para indicar la dirección de visibilidad de los atributos.
8. Añadir relaciones de dependencia para indicar visibilidad no correspondiente a atributos.

No todas las clases que aparecían en el Modelo Conceptual tienen por qué aparecer en el Diagrama de Clases de Diseño. De hecho, tan solo se incluirán aquellas clases que tengan interés en cuanto a que se les ha asignado algún tipo de responsabilidad en el diseño de la interacción del sistema. No hay, por tanto, un transición directa entre el Modelo Conceptual y el Diagrama de Clases de Diseño, debido a que ambos se basan en enfoques completamente distintos: el primero en comprensión de un dominio, y el segundo en una solución software.
En la fase de Diseño se añaden los detalles referentes al lenguaje de programación que se vaya a usar. Por ejemplo, los tipos de los atributos y parámetros se expresarán según la sintaxis del lenguaje de implementación escogido.

IV.4.4.3 Navegabilidad

La navegabilidad es una propiedad de un rol (un extremo de una asociación) que indica que es posible "navegar" unidireccionalmente a través de la asociación, desde objetos de la clase origen a objetos de la clase destino. Como se vio en la parte II, se representa en UML mediante una flecha.
La navegabilidad implica visibilidad, normalmente visibilidad por medio de un atributo en la clase origen. En la implementación se traducirá en la clase origen como un atributo que sea una referencia a la clase destino.
Las asociaciones que aparezcan en el Diagrama de Clases deben cumplir una función, deben ser necesarias, si no es así deben eliminarse.
Las situaciones más comunes en las que parece que se necesita definir una asociación con navegabilidad de A a B son:

- A envía un mensaje a B.
- A crea una instancia B.
- A necesita mantener una conexión con B.

IV.4.4.4 Visibilidad de Atributos y Métodos

Los atributos y los métodos deben tener una visibilidad asignada, que puede ser:
+ Visibilidad pública.
# Visibilidad protegida.
- Visibilidad privada.
También puede ser necesario incluir valores por defecto, y todos los detalles ya cercanos a la implementación que sean necesarios para completar el Diagrama de Clases.

IV.4.5 Otros Aspectos en el Diseño del Sistema

En los puntos anteriores se ha hablado de decisiones de diseño a un nivel de granularidad fina, con las clases y objetos como unidades de decisión. En el diseño de un sistema es necesario también tomar decisiones a un nivel más alto sobre la descomposición de un sistema en subsistemas y sobre la arquitectura del sistema (ver sección III.2.5). Esta parte del Diseño es lo que se denomina Diseño del Sistema. Estas decisiones no se toman de forma distinta en un desarrollo orientado a objetos a como se llevan a cabo en un desarrollo tradicional. Por tanto, no se va a entrar en este documento en cómo se realiza esta actividad.
Sí hay que tener en cuenta que las posibles divisiones en subsistemas tienen que hacerse en base a las clases definidas en el Diagrama de Clases del Diseño.

IV.5 Fases de Implementación y Pruebas

Una vez se tiene completo el Diagrama de Clases de Diseño, se pasa a la implementación en el lenguaje de programación elegido.
El programa obtenido se depura y prueba, y ya se tiene una parte del sistema funcionando que se puede probar con los futuros usuarios, e incluso poner en producción si se ha planificado una instalación gradual.
Una vez se tiene una versión estable se pasa al siguiente ciclo de desarrollo para incrementar el sistema con los casos de uso asignados a tal ciclo.

V Bibliografía

[Booch99] **El Lenguaje Unificado de Modelado**. G. Booch, J. Rumbaugh, I. Jacobson. Addison Wesley Iberoamericana, 1999.

[Booch94] **Object-Oriented Analysis and Design**. G. Booch. Benjamin/Cummings, 1994.

[BJR97] **The UML Specification Document**. G. Booch, I. Jacobson and J. Rumbaugh. Rational Software Corp., 1997.

[Jacobson92] **Object-Oriented Software Engineering: A Use Case Driven Approach**. I. Jacobson. Addison-Wesley, 1992.

[Larman99] **UML y Patrones**. C. Larman. Prentice Hall, 1999.

[Rumbaugh91] **Object-Oriented Modeling and Design**. J. Rumbaugh et al.Prentice- Hall,1991.

[UML00] **UML Resource Center**. Rational Software. http://www.rational.com/uml/