Desarrollo Orientado a Objetos con UML

Programación
C.E.C.yT. “Juan de Dios Bátíz Paredes” – IPN

Índice

I 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.

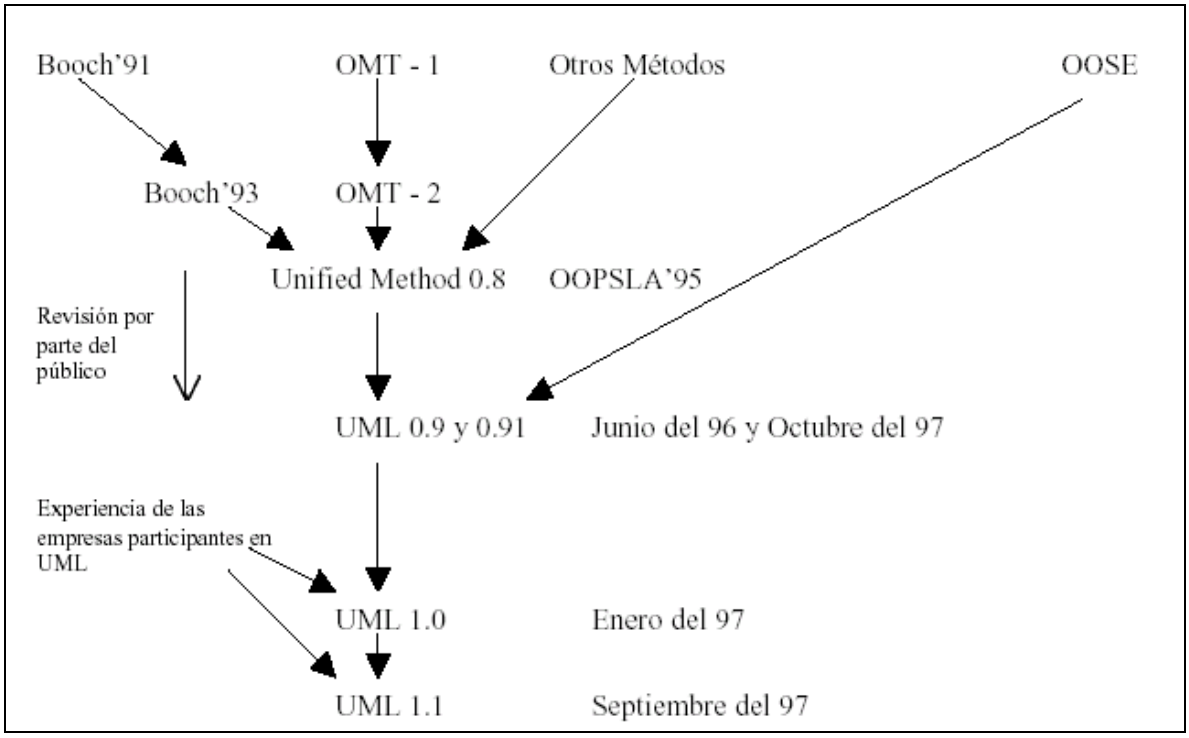

Figura 2 - Historia de UML.

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 proceso propuesto por Craig Larman[Larman99] que se ajusta a un ciclo de vida evolutivo e incremental dirigido por casos de uso.

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

II Notación 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 con los que vamos a trabajar son los siguientes:

- Diagrama de Estructura Estática.
- Diagrama de Casos de Uso.
- Diagrama de Secuencia.
- Diagrama de Colaboración.
- Diagrama de Estados.
- Diagrama de Paquetes.

II.2 Elementos Comunes a Todos los Diagramas

II.2.1 Notas

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 o un valor rotulado (*tagged value)*.

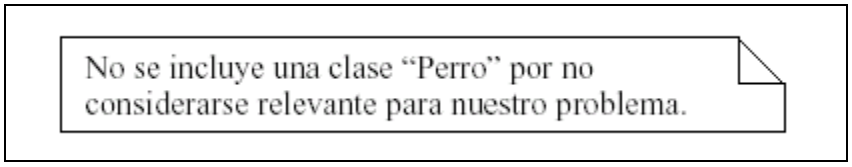

Figura 3 - Ejemplo de nota.

II.2.2 Agrupación de Elementos Mediante Paquetes

Un paquete es un mecanismo de propósito general para organizar elementos en grupos. Cualquier grupo de elementos, sean estructurales o de comportamiento, puede incluirse en un paquete. Incluso pueden agruparse paquetes dentro de otro paquete.

Un paquete se representa como un rectángulo grande con un pequeño rectángulo sobre la esquina superior izquierda a modo de lengüeta. Si no se muestra el contenido del paquete entonces el nombre del paquete se coloca dentro del rectángulo grande. Si, por el contrario, se quiere mostrar el contenido del paquete, entonces el contenido va dentro del rectángulo grande y el nombre del paquete va en la lengüeta.

Se pueden indicar relaciones de dependencia entre paquetes mediante una flecha con la línea a trazos. Si el paquete A depende del paquete B, entonces irá una flecha de A a B, tal y como se muestra en la figura 4.

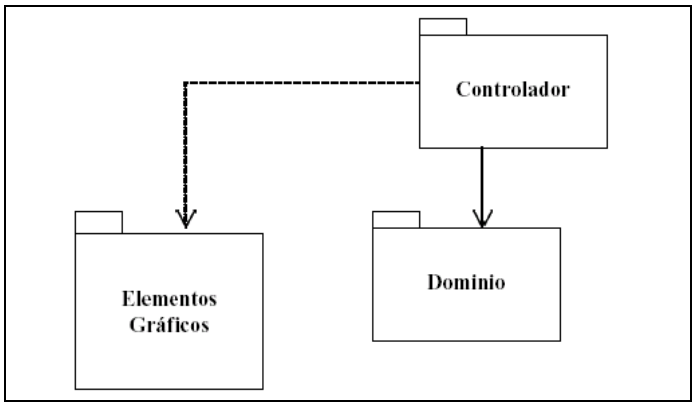

Figura 4 – Agrupación por Paquetes.

II.3 Diagramas de Estructura Estática

Con el nombre de Diagramas de Estructura Estática se engloba tanto al Modelo Conceptual de la fase de Diseño de Alto Nivel como al Diagrama de Clases 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.

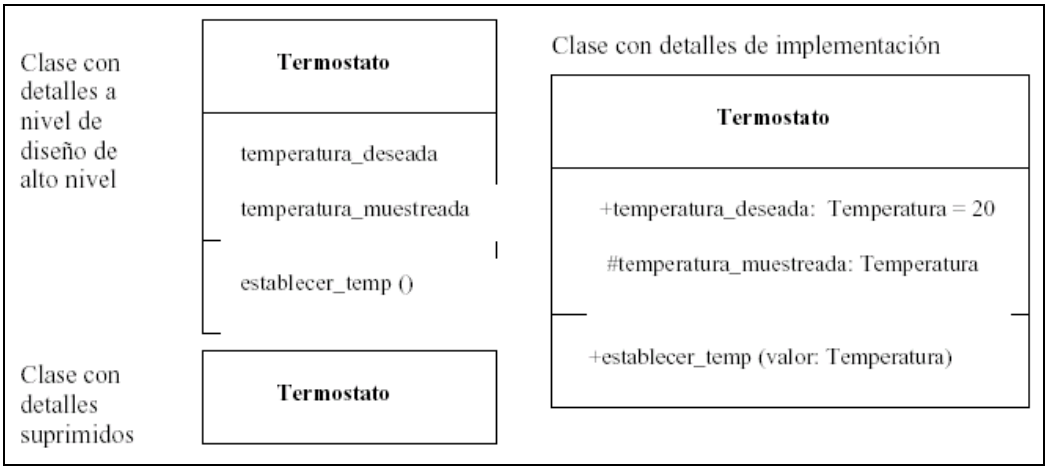

Figura 5 - Notación para clases a distintos niveles de detalle.

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.

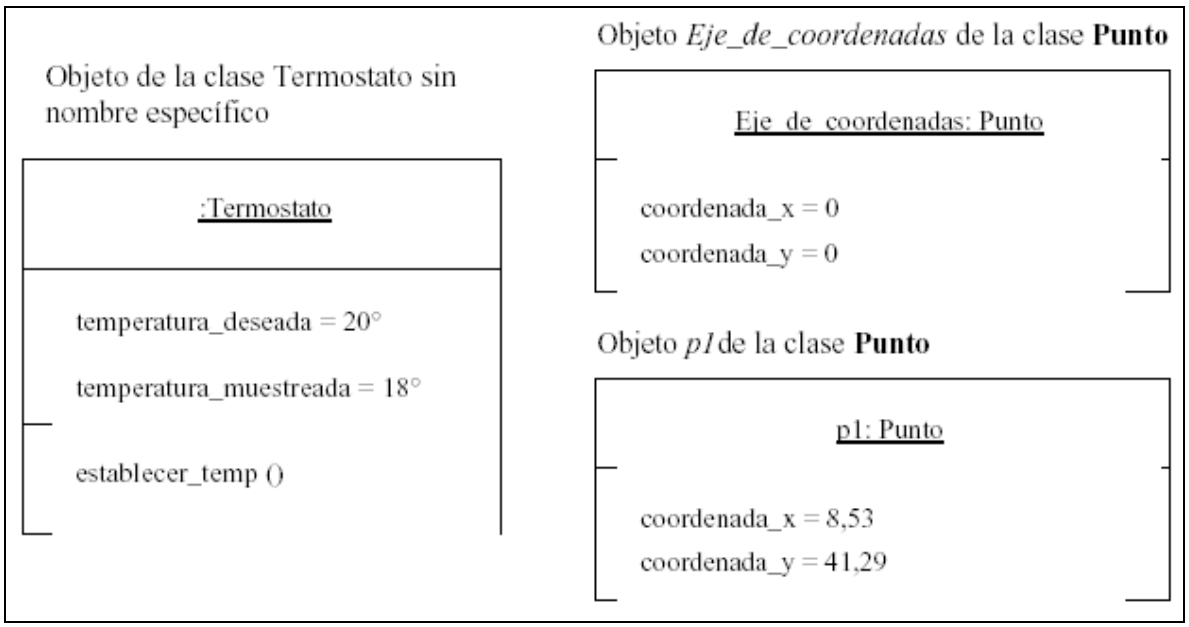

Figura 6 - Ejemplos de objetos.

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 "Un objeto de la clase Perro es mascota de un objeto de la clase Persona".

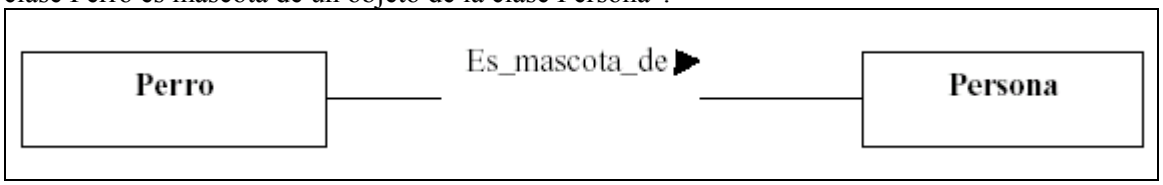

Figura 7 - Ejemplo de asociación con nombre y dirección.

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

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).

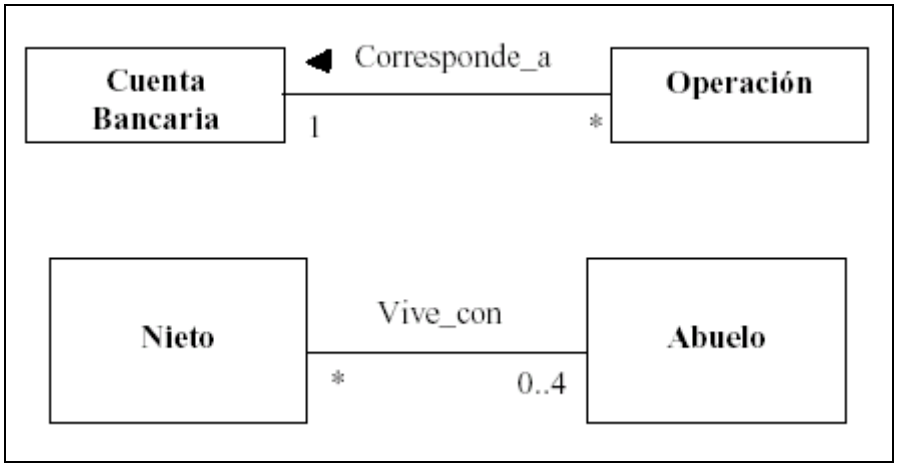

Figura 8 - Ejemplos de multiplicidad en asociaciones.

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.

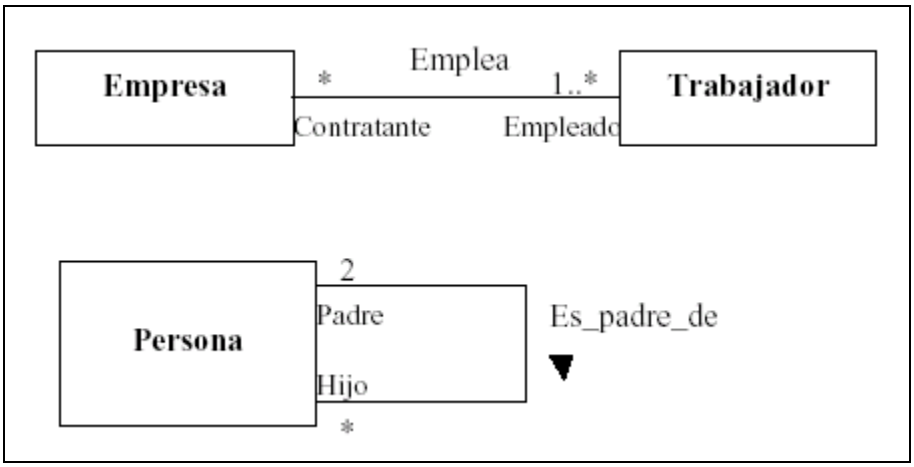

Figura 9 - Ejemplo de roles en una asociación.

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".

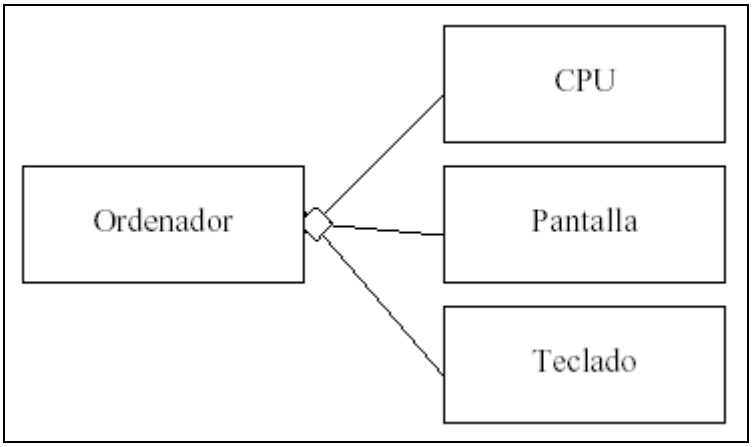

Figura 10 - Ejemplo de agregación.

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 12). 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.

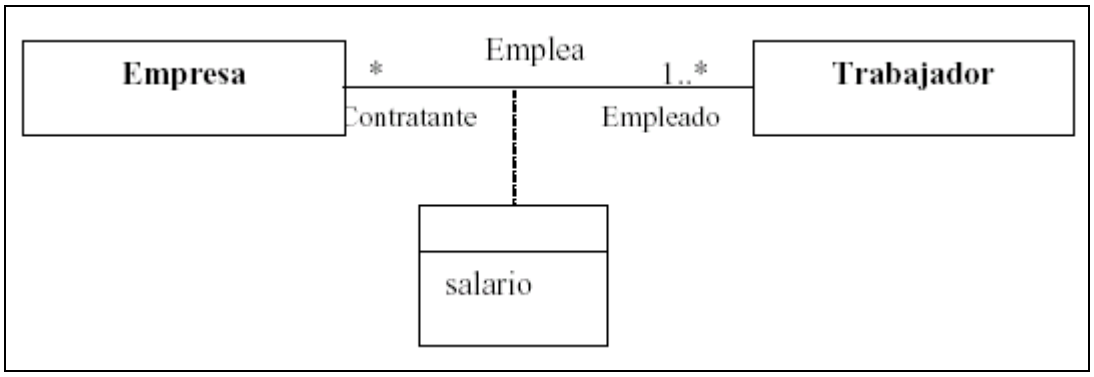

Figura 12 - Ejemplo de clase asociación.

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 11 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.

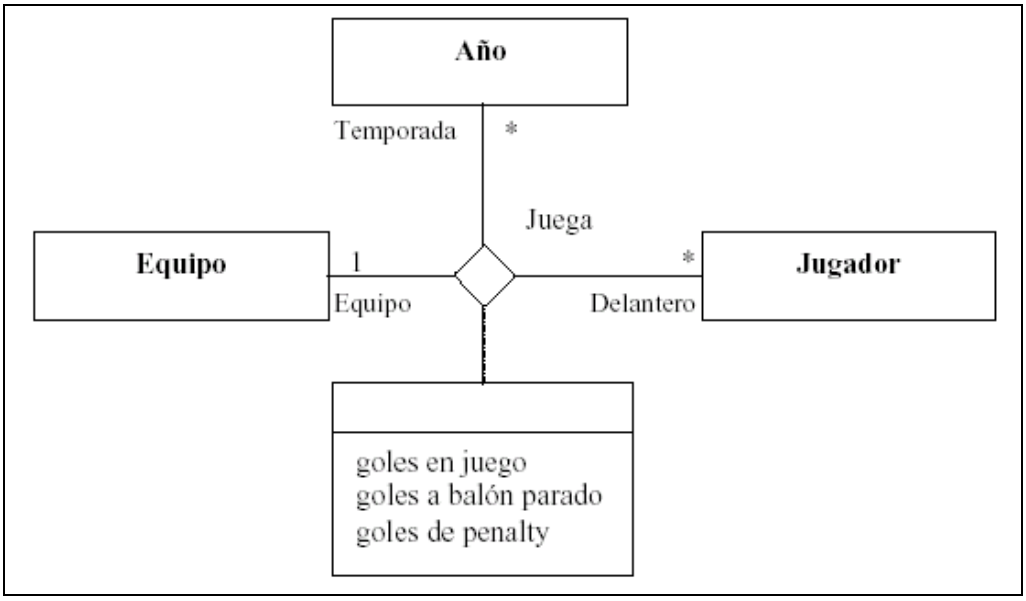

Figura 11 - Ejemplo de asociación ternaria.

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 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".

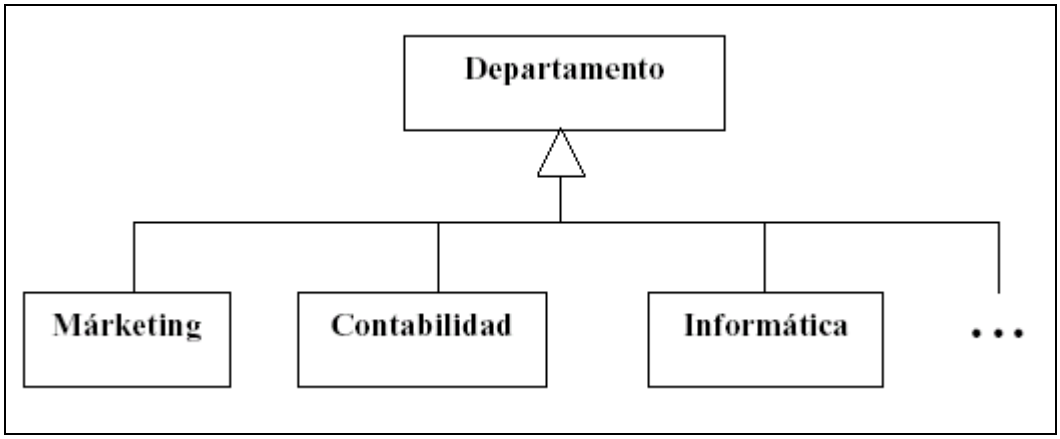

Figura 13 - Ejemplo de herencia.

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

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.

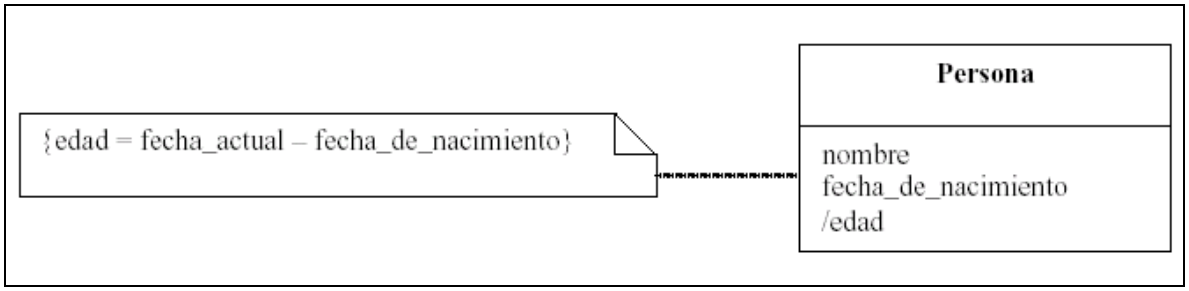

Figura 14 - Ejemplo de atributo derivado.

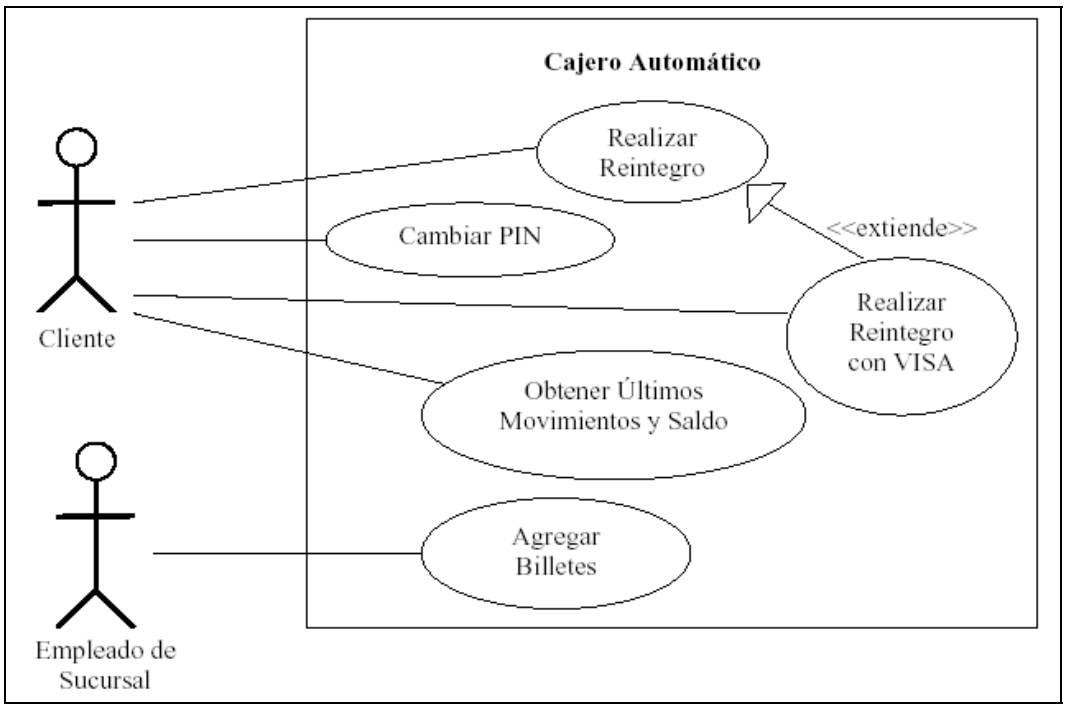

Figura 15 - Diagrama de Casos de Uso.

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.

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.

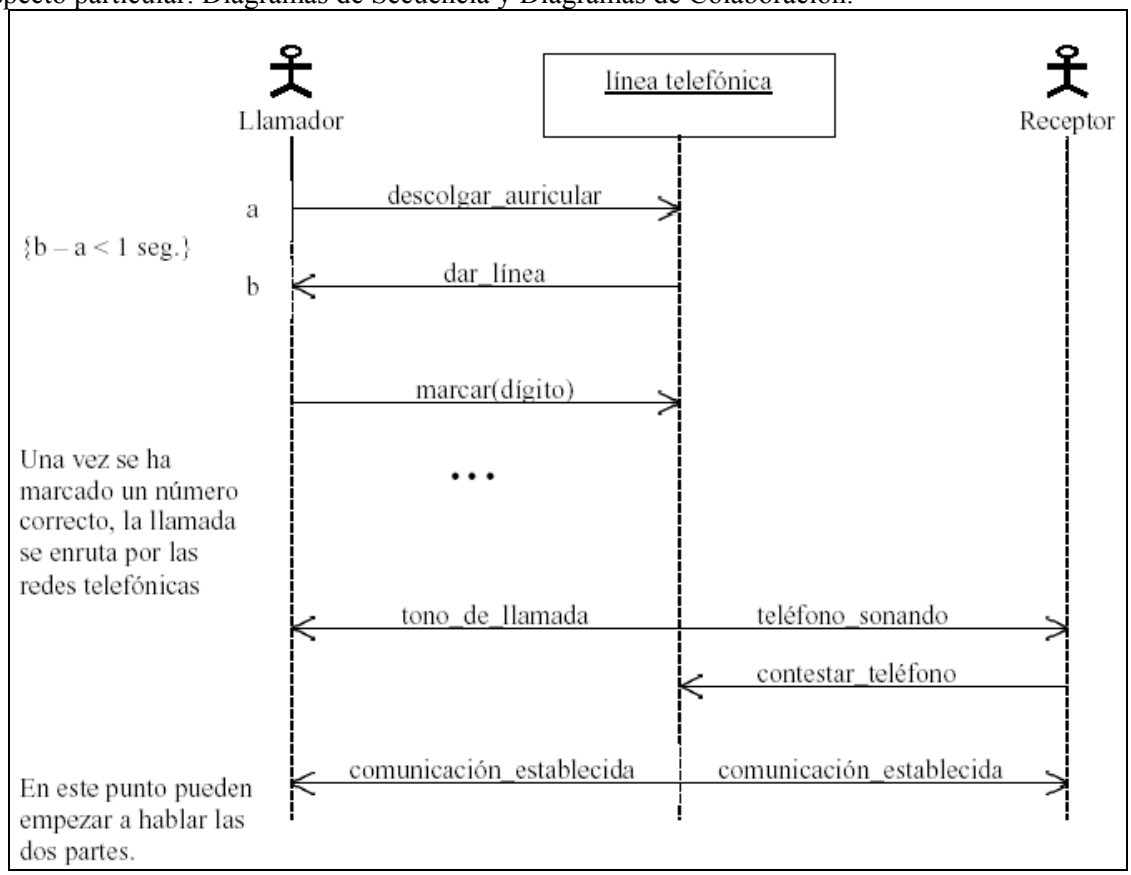

Figura 16 - Diagrama de Secuencia.

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. En la figura 16 se representa el Diagrama de Secuencia para la realización de una llamada telefónica.

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.

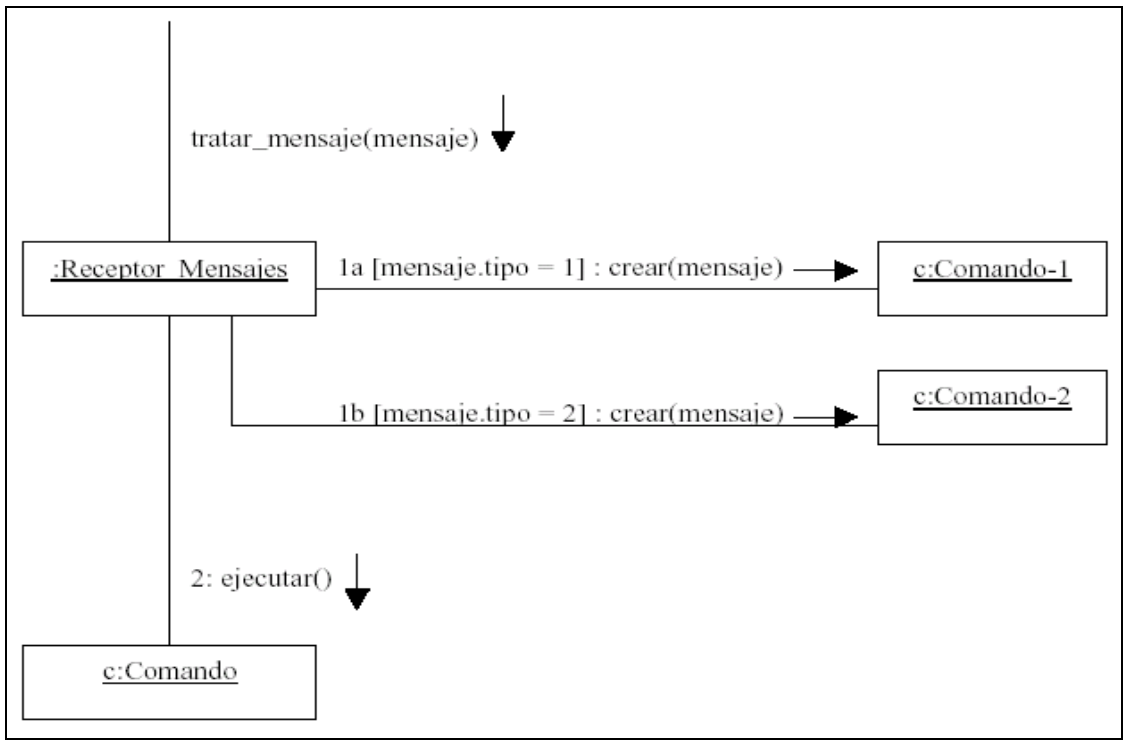

Figura 17 - Diagrama de Colaboración.

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.*

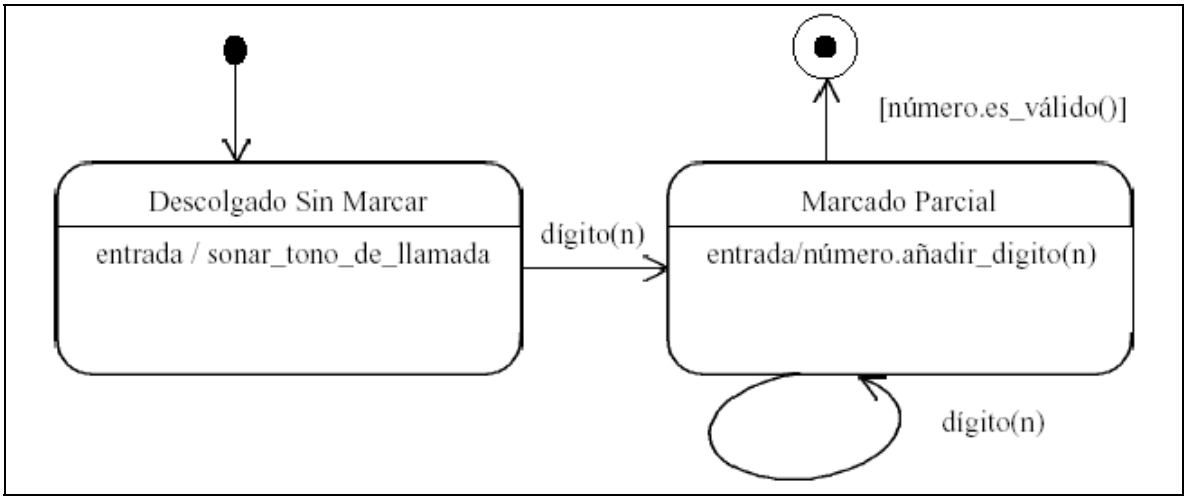

Figura 18 - Diagrama de Estados.

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 Desarrollo Orientado a Objetos

III.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. Tal proceso de desarrollo se ocupa de plantear cómo se realizan las distintas actividades, y cómo se relacionan los productos de las mismas. Con el uso de un proceso de desarrollo adecuado 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 proceso 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 evolutivo 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.

III.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:

  - **Diseño de Alto Nivel**: Se aborda el problema viendo al sistema a construir como una caja negra, centrándonos en la visión que desde el exterior tienen los actores, esto es, en los casos de uso. Se analiza el problema construyendo un modelo conceptual.
  - **Diseño de Bajo Nivel**: El sistema definido en la fase anterior se especifica en detalle, describiendo todas las operaciones que el sistema va a tener que realizar internamente para satisfacer lo especificado en el diseño de alto nivel.
  - **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 evolutivo, tomando en cada iteración un subconjunto de los requisitos (agrupados según casos de uso) y llevándolo a través del diseño de alto y bajo nivel hasta la implementación y pruebas, tal y como se muestra en la figura 19. 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.

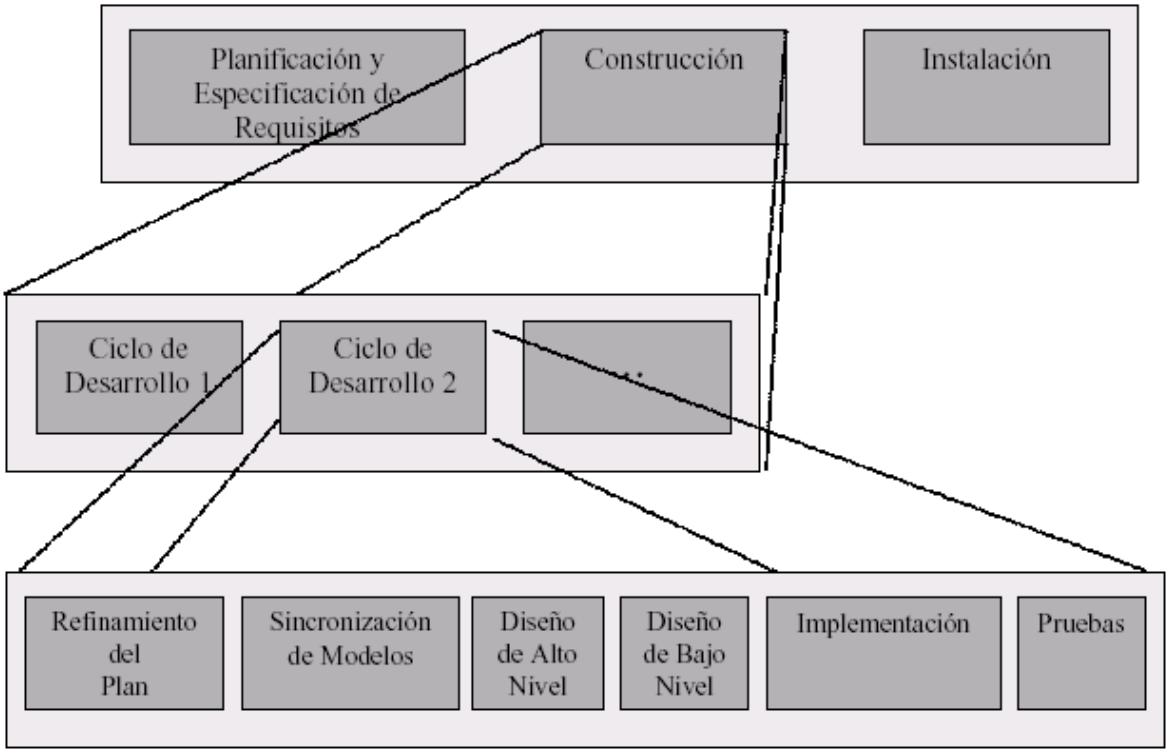

Figura 19 - Desarrollo Evolutivo en la Construcción.

III.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.

III.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 posteriores. 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 la fase de Diseño de Alto nivel.

III.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.

III.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 10 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 adelante. 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.

III.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.

III.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 el dinero, el recibo y la tarjeta, 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.

III.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.

III.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.

III.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.

III.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. 3. Escoge la operación *A*. 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*. 7. Recoge el recibo y se va.	2. Pide la operación a realizar. 4. Presenta las opciones de pago. 6. Genera recibo.

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

...

III.2.4 Construcción del Modelo de Casos de Uso

Para construir el Modelo de Casos de Uso en la fase de Planificación y Especificación 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).

III.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 20).

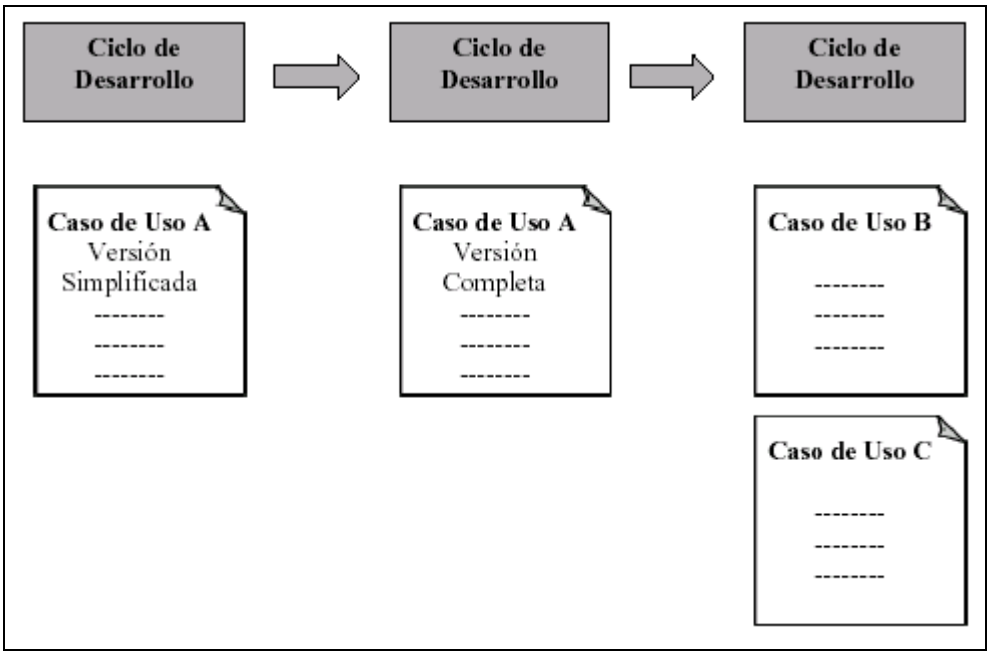

Figura 20 - Planificación de Ciclos de Desarrollo según Casos de Uso.

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.
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
...							

III.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.

III.3 Fase de Construcción: Diseño de Alto Nivel

En la fase de Diseño de Alto Nivel de un ciclo de desarrollo se *investiga* sobre el problema, intentando diseñar lo que va a ser la interacción del sistema con el exterior. Se identifican los conceptos relacionados con el subconjunto de casos de uso que se esté tratando. Los detalles de implementación se dejan para la fase de Diseño de Bajo Nivel. Cuando el ciclo de desarrollo no es el primero, antes de la fase de Diseño de Alto Nivel 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 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 Diseño de Bajo Nivel. En esta fase se trabaja con los modelos de Diseño de Alto Nivel construidos en la fase anterior, ampliándolos con los conceptos correspondientes a los casos de uso que se traten en el ciclo de desarrollo actual.

III.3.1 Actividades

Las actividades de la fase de Diseño de Alto Nivel 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. Definir los Diagramas de Secuencia del Sistema.
4. Refinar el Modelo Conceptual.
5. Refinar el Glosario. (*continuado en posteriores fases*)
6. Definir Contratos de Operación.
7. Definir Diagramas de Estado. (*opcional*)

III.3.2 Diagramas de Secuencia del Sistema

Para comenzar a modelar el comportamiento del sistema tal y como los actores lo necesitan vamos a partir de los casos de uso. Para cada caso de uso se va a construir una serie de Diagramas de Secuencia que muestren los eventos que llegan al sistema para cada escenario del caso de uso.

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 11). 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. Tan solo se representan los eventos que *entran* al sistema, no los que *salen* del mismo. Tales eventos entrantes constituyen las **operaciones del sistema**, las operaciones que son activadas por alguna acción de un actor.

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.

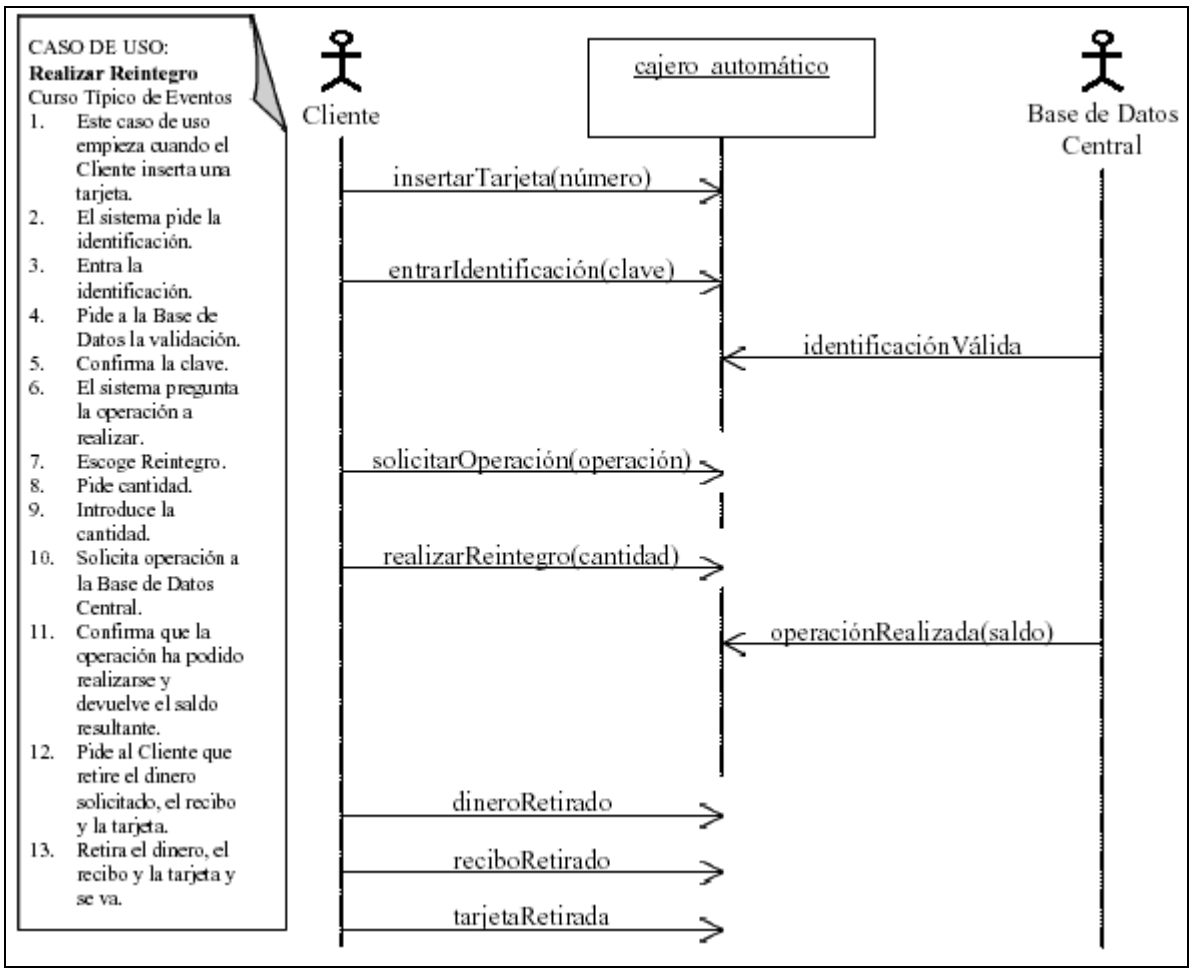

Figura 21 - Ejemplo de Diagrama de Secuencia del Sistema.

En la figura 21 se muestra el Diagrama de Secuencia del Sistema para el caso de uso Realizar Reintegro de un cajero automático.

III.3.2.1 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.

Con el Diagrama de Secuencia identificamos las Operaciones del Sistema. Cada Operación del Sistema va a definirse en cuanto a qué se espera de ella mediante un contrato. Tal definición se va a basar en los cambios que sufren los elementos del Modelo Conceptual. Los Contratos de las Operaciones del Sistema y el Modelo Conceptual van a ir desarrollándose en paralelo: en cada definición de un contrato se va hacer referencia a cambios en el modelo conceptual. Vamos a tratar en primer lugar el Modelo Conceptual y a continuación los Contratos de Operaciones del Sistema.

III.3.3 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.

III.3.3.1 Identificación de Conceptos

Para identificar conceptos hay que basarse en los requisitos, en la descripción de los casos de uso 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
Archivos financieros, de trabajo, de contratos, de asuntos legales.	Recibo, Contrato_de_Empleo Registro_de_Revisiones
Instrumentos y servicios financieros	Línea_de_Crédito Stock
Manuales, libros	Manual_del_Empleado Manual_de_Reparaciones

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 del cartógrafo, resumida en los siguientes tres puntos:

- *Usar lo s 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.

III.3.3.2 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.

III.3.3.3 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.

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

Una vez identificadas las asociaciones se representan en el Modelo Conceptual con la multiplicidad adecuada.

III.3.3.4 Identificación de Atributos

Es necesario incorporar al Modelo Conceptual los atributos necesarios para satisfacer las necesidades de información de los casos de uso que se estén desarrollando en ese momento. Los atributos deben tomar valor en tipos simples (número, texto, etc.), pues los tipos complejos deberían ser modelados como conceptos y ser relacionados mediante asociaciones.

Incluso cuando un valor es de un tipo simple es más conveniente representarlo como concepto en las siguientes ocasiones:

- Se compone de distintas secciones. Por ejemplo: un número de teléfono, el nombre de una persona, etc.
- Tiene operaciones asociadas, tales como validación. Ejemplo: NIF.
- Tiene otros atributos. Por ejemplo un precio de oferta puede tener fecha de fin.
- Es una cantidad con una unidad. Ejemplo: El precio, que puede estar en pesetas o en euros.

Una vez definidos los atributos se tiene ya un Modelo Conceptual. Este modelo no es un modelo definitivo, pues a lo largo del Diseño de Alto y Bajo Nivel se va refinando según se le añaden conceptos que se habían pasado por alto.

III.3.4 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.
...	...	...

III.3.5 Contratos de Operaciones

Una vez se tienen las Operaciones del Sistema identificadas en los Diagramas de Secuencia, sedescribe 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.
Tipo:	Sistema
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.
Tipo:	Nombre del tipo (clase software, sistema).
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.
Salida:	Salidas que no corresponden a la interfaz de usuario, como mensajes o registros que se envían fuera del sistema.
Pre-condiciones:	Asunciones acerca del estado del sistema antes de ejecutar la operación.
Post-condiciones:	El estado del sistema después de completar la operación.

III.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.
4. A continuación rellenar el apartado de *Post-condiciones*, describiendo declarativamente los cambios de estado que sufren los objetos en el Modelo Conceptual.
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.

III.3.5.2 Post-condiciones

La parte más importante de un contrato es aquella en la que se describen las post-condiciones. Éstas 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 postcondiciones 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.

III.3.6 Diagramas de Estados

Para modelar el comportamiento del sistema pueden usarse los Diagramas de Estados que define UML (ver página 13). 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 Diseño de Alto Nivel 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 Diseño de Alto Nivel 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.

III.4 Fase de Construcción: Diseño de Bajo Nivel

En la fase de Diseño de Bajo Nivel se crea una solución a nivel lógico para satisfacer los requisitos, basándose en lo diseñado en la fase de Diseño de Alto Nivel.

III.4.1 Actividades (Realizadas en la etapa de Diseño de Bajo Nivel)

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.

III.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* 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.

III.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.

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.

III.4.3.1 Creación de Diagramas de Colaboración

- 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.

III.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.

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.

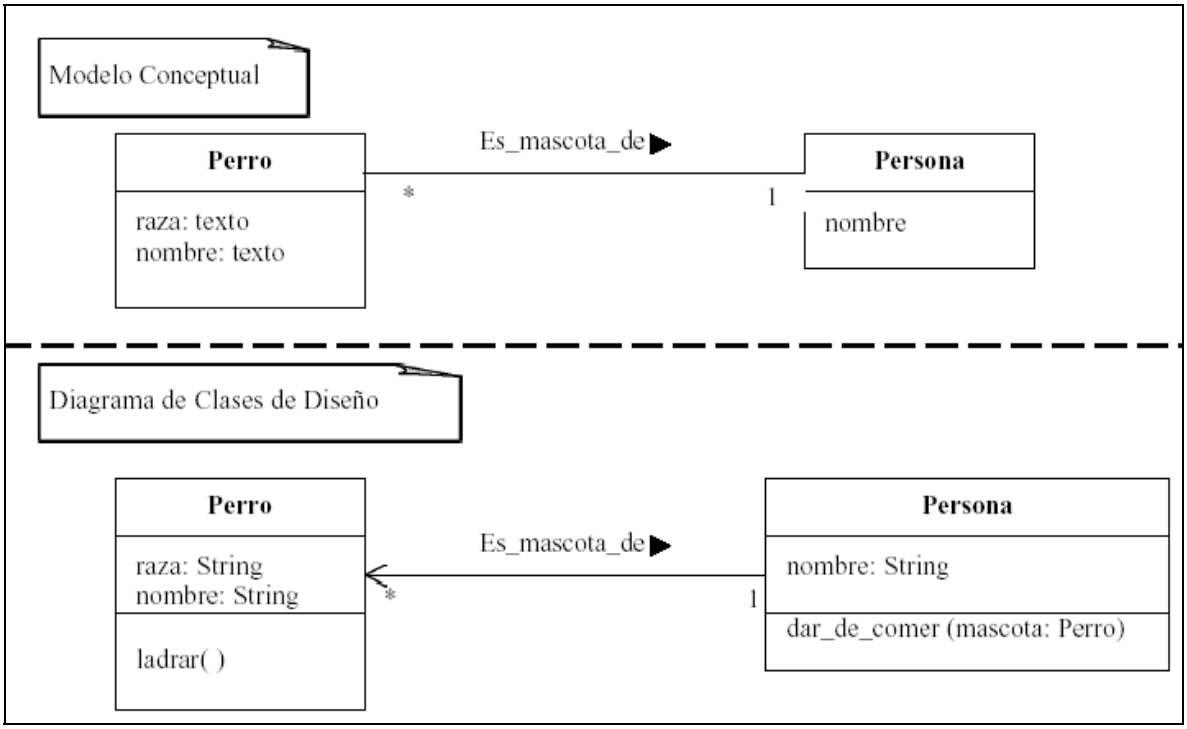

Figura 22 - Comparación entre el Modelo Conceptual y un Diagrama de Clases de Diseño.

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 22 se muestra una comparación entre clases del Modelo Conceptual y clases del Diagrama de Clases de Diseño.

III.4.4.1 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.

III.4.4.2 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.

III.4.4.3 Visibilidad

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.

III.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. 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.

Para realizar una división en módulos o paquetes del sistema, se empleará la notación para paquetes que define UML (ver página 3).

III.5 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 planeado 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.

IV 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/