Materia: Análisis de Sistemas II

Año: 2020

Profesores: Dra. Lic. María José Reina
Mg. Lic. Daniel E. Lillo
Ing. Alberto Cortez

Tema: Análisis Orientado a Objetos

El Modelado y Diseño Orientado a Objetos es una nueva vía para encarar los problemas con conceptos de modelos que están en el mundo real.
La estructura fundamental es el *objeto*, el cual combina estructura de datos y el comportamiento en una sola entidad. Los modelos orientados a objetos son usados para entender problemas, comunicarse con expertos, diseños de programas y bases de datos.
El primer modelo de análisis es construido para abstraer los aspectos esenciales del dominio de la aplicación sin tener en cuenta los detalles eventuales.
Este modelo contiene objetos conocidos en el dominio de la aplicación, incluyendo una descripción de las propiedades de los objetos y su comportamiento. Las decisiones de diseño están hechas y detalladas y son sumadas al modelo que describe y optimiza la implementación.

Que es Orientado a Objetos?.

Superficialmente el término Orientado a Objetos, nos dice que organiza el software como una combinación discreta de objetos que incorpora estructura de datos y su comportamiento. Es esto en contraste con la programación convencional en la cual la estructura de datos y comportamiento están rara vez conectadas. Hay algunas disputas acerca de cuales son exactamente las características que se requiere para ser orientado a objetos, pero generalmente incluye cinco aspectos: **Identidad, Clasificación, Polimorfismo, Herencia y Encapsulación**.

Características de los Objetos.

Identidad se refiere a como el dato es cuantificado en entidades discretas y distinguibles llamadas *objeto*s. Ejemplo de esto es un párrafo en un documento, una ventana en mi workstation, etc.
Un *objeto* debe ser algo concreto tal como una política de seguridad en un sistema operativo con multiprocesamiento. Cada *objeto* tiene su propia identidad inherente. En otras palabras, dos *objeto*s son distintos aunque todos sus atributos sean idénticos.
En el mundo real un *objeto* simplemente existe, pero sin un lenguaje de programación adecuado cada *objeto* tiene un único índice por el cual es siempre referenciado. Esto puede ser implementado de varias formas, tal como direcciones o un arreglo de índices, o un único valor como un atributo.
Las referencias a objetos son uniformes e independientes del contenido del objeto, permitiendo combinar colecciones de objetos para ser creados, tal como un File System que contiene archivos directorios y archivos ordinarios.

La *Clasificación* trata a los objetos con la misma estructura de datos (atributos) y comportamiento (operaciones), como objetos agrupados en ***clase*s**. Párrafos, ventanas son ejemplos de *clase*s. Una *clase* es una abstracción que describe propiedades importantes para una aplicación y el resto es ignorada. Cualquier elección de *clases* es arbitraria y depende de la aplicación.
Cada *clase* describe infinitas posibilidades de objetos individuales. Cada objeto es también llamado como una *instancia* de la *clase*. Cada *instancia* de una *clase* tiene un valor propio para cada atributo pero comparte los mismos nombres de atributos y el mismo comportamiento que otras *instancia*s de la *clase*.

El *Polimorfismo* trata que una misma operación se pueda comportar de forma diferente en clases distintas. Por ejemplo la operación mover puede ser diferente en la clase ventana, que en la clase bicicleta. Una operación es una acción o una transformación que un objeto experimenta o de la cual forma parte. Una implementación específica de una operación en una cierta clase es llamada *método*. A causa de que un operador de objetos es polimórfico, este puede tener mas de un *método* que lo implemente.
En el mundo real una operación es simplemente una abstracción de comportamiento analógico que experimentan diferentes tipos de objetos. Cada objeto muestra como se comporta con sus propias operaciones. En cambio en los lenguajes de programación orientados a objetos, el lenguaje selecciona automáticamente el método correcto que implementa una operación basada en el nombre de la operación y la clase a la cual el objeto pertenece. Pueden agregarse nuevas clases sin la necesidad de cambiar el código.

La *herencia* es poder compartir atributos y operaciones con otras clases basadas en relaciones jerárquicas. Una clase es definida globalmente y es depurada en las sucesivas clases llamadas *subclase*s. Cada *subclase* incorpora, o hereda, todas las propiedades de la *superclase* y además agrega sus propias propiedades. Las propiedades de una *superclase* no necesitan ser repetidas en cada *subclase*. Una de las ventajas de sistema orientado a objetos, es que la posibilidad de tener como factor común las propiedades las *superclase*s y las *subclase*s, por las propiedades de herencia, lo que reduce la repetición en diseños y programas.

Encapsulación indica la capacidad de los objetos de construir una cápsula a su alrededor que lo convierte en una caja negra, **ocultando** la información interna que manipulan.
La encapsulación aparece ya en los lenguajes basados en procedimientos como una forma de proteger los datos. Con esta no nos preocupa, por ejemplo, si un nivel superior está usando una misma variable para una determinada iteración.
La encapsulación nos permite el uso de librerías de propósito general. Además, al verse como una caja negra, nosotros podemos, respetando sus entradas y salidas, modificar el código interno optimizándolo constantemente sin efectos laterales o secundarios.
La POO permite además encapsular las estructuras de datos que sirven como base a las funciones. Aportan, por tanto, un nivel superior en cuanto a protección de información.
La encapsulación está en el núcleo de los dos grandes pilares de la construcción de sistemas: *reusabilidad y mantenibilidad.*

Que es el desarrollo Orientado a Objetos?.

Es una nueva vía de desarrollo de software basado en una abstracción del mundo real. Desde este punto de vista el desarrollo se refiere a una porción del ciclo de vida del software: análisis, diseño, e implementación.

La esencia del desarrollo orientado a objetos es la identificación y organización del concepto de dominio de la aplicación, antes de llegar a representarlo en un lenguaje de programación orientado a objetos o no.

Concepto de Modelado, no Implementación.

Los lenguajes de programación orientados a objetos son usados para remover las restricciones que están dadas por la inflexibilidad que tienen los lenguajes tradicionales.
Conceptualmente el Modelado es un proceso independiente del lenguaje de programación. El desarrollo Orientado a Objetos es fundamentalmente una nueva vía para pensar y no una nueva técnica de programación. Esto trae grandes beneficios para la comunicación entre desarrolladores y usuarios ya que porque se pueden expresar conceptos abstractos claramente.
Esto puede servir como un medio para la especificación, análisis, documentación y las interfaces, así como para programar. Ya sea en lenguajes convencionales, como así también en bases de datos, y obviamente para lenguajes Orientados a Objetos.

Metodología Orientada a Objetos.

Hay diferentes metodologías para representar el desarrollo orientado a Objetos y notaciones gráficas para representar los conceptos Orientados a Objetos.
La metodología consiste en construir un modelo del dominio de una aplicación y a el adicionarle los detalles que durante el diseño del sistema se vayan agregando. La metodología tiene los siguientes estados:

1- *Análisis*: comienza con la definición del problema, el análisis construye un modelo de la situación en el mundo real mostrando las propiedades importantes. En el análisis se debe trabajar con una persona que entienda el problema, porque los estados del problema son raramente completos o correctos.
El modelo de análisis es conciso, una abstracción precisa de que hace el sistema, no como debería hacerse. Los objetos en el modelo deben ser conceptos del dominio de la aplicación, y no conceptos de implementar para la computadora, tales como estructuras de datos. El modelo de análisis no debe contener ninguna decisión de implementación, por ejemplo una clase Ventana en una workstation del sistema de pantalla, debe ser descripta desde el punto de vista de atributos y operaciones desde el punto de vista del usuario.

2- *Diseño del sistema*: el diseñador de sistemas toma decisiones de un nivel superior acerca de la arquitectura del sistema. Durante en diseño del sistema, el sistema destino es organizado en subsistemas basados en el análisis estructurado y en la arquitectura propuesta. El diseñador de sistemas debe decidir que características de performance a optimizar, elige una estrategia para atacar el problema y hace ubicaciones tentativas de los recursos. Por ejemplo, el diseñador del sistema puede elegir un apropiado protocolo de comunicaciones y una estrategia de manejo de buffers de memoria.

3- *Diseño de Objetos*: el diseñador de objetos construye un modelo basado en el modelo de análisis, pero contiene detalles de la implementación. El diseñador adiciona los detalles al modelo de diseño en concordancia con una estrategia establecida durante el diseño. El foco del diseñador de objetos es la estructura de datos y los algoritmos que necesita para implementar cada clase. Las clases de objetos que vienen del análisis son ahora argumentadas en el dominio de la computadora, con algoritmos y estructuras de datos elegidas para optimizar las medidas de

performance. Por ejemplo, las operaciones de la clase ventanas, son especificadas en términos que involucren al hardware y al sistema operativo.

4- *Implementación*: las clases de objetos y sus relaciones, desarrolladas durante el diseño de objetos son finalmente trasladadas a algún lenguaje de programación, base de datos o una implementación de hardware. La programación debe ser la parte menor y relativamente mecánica del ciclo de desarrollo, porque las decisiones de hard ya fueron hechas en la etapa del diseño.

Actividades de la Metodología Orientada a Objetos

Análisis

- Escribir u obtener una descripción del problema
- Construir el modelo de objetos
- Construir el modelo dinámico
- Construir el modelo funcional
- Verificar, iterar y refinar los tres modelos

Diseño del sistema

- Organizar el sistema en subsistemas
- Identificar concurrencia inherente al problema
- Asignar subsistemas a procesadores y tareas
- Escoger una estrategia para la implementación de almacenamiento de datos
- Determinar los mecanismos para controlar el acceso a recursos globales
- Escoger la implementación del control del software
- Manejar condiciones de frontera
- Establecer prioridades

Diseño de objetos

- Obtener operaciones de los modelos funcional y dinámico
- Diseñar algoritmos para realizar las operaciones
- Optimizar los caminos de acceso a los datos
- Implementar el control del software
- Ajustar la estructura de clases para incrementar herencia
- Diseñar implementación de asociaciones
- Determinar la representación de los atributos de las clases
- Agrupar clases y asociaciones en módulos

Implementación

- Diseñar bases de datos
- Codificar

Conceptos de modelado

Un modelo es una abstracción de algo, con el propósitos de entenderlo antes de construirlo. Porque un modelo que omite los detalles no esenciales, es muy fácil de manejar como una entidad original.
La abstracción es una fundamental capacidad humana.
Los ingenieros, artistas, arquitectos, hacen modelos para ver sus diseños antes de ejecutarlos. Los desarrolladores de hardware y sistemas de software no deberían ser la excepción.
Para construir sistemas complejos, el desarrollador debe abstraer diferentes vistas del sistema, construyendo modelos utilizando notaciones precisas, verifica que el modelo satisface los requerimientos del sistema, y gradualmente adiciona los detalles para transformarlo en una implementación.

Modelado

Los diseñadores construyen muchos tipos de modelos para varios propósitos antes de construir el definitivo. Ya di el ejemplo de las maquetas de arquitectura, aviones a escala, etc.. Los modelos sirven para los siguientes propósitos:

* testear una entidad física antes de construirla.
* comunicación con los usuarios.
* visualización.
* reducción de la complejidad.

Abstracción

Es la examinación selectiva de ciertos aspectos de un problema. El efecto de la abstracción es mostrar los aspectos más relevantes del sistema para algunos propósitos y descartar otros que no son tan importantes.
La abstracción siempre debe ser para algunos propósitos, pues el propósito determina que es lo importante y que no. Es posible hacer muchas abstracciones diferentes del mismo problema, depende del propósito para el cual fueron hechas.
Todas las abstracciones son incompletas. Todas las palabras de los humanos y los lenguajes son abstracciones - descripciones incompletas del mundo real -.
El propósito de una abstracción es limitar el universo sobre el cual me muevo.
Un buen modelo captura los aspectos cruciales de un problema y omite los otros.

Técnica para el modelado de objetos

Nosotros vamos a ver el modelado desde tres puntos de vistas diferentes relacionados entre sí, donde cada uno captura aspectos importantes del sistema, pero se requiere de los tres para una completa descripción. La técnica de modelado de objetos es el nombre de la metodología que combina estos tres puntos de vista del modelado de sistemas.
El *Modelo objeto* representa un aspecto estático, estructurado, "datos" del sistema.
El *Modelo dinámico* representa el temporal, comportamiento, aspectos de "control" del sistema.

El *modelo funcional* representa la transformación, aspectos de "función" del sistema.
Un procedimiento de software típico incorpora los tres aspectos: usa estructura de datos (*Modelo objeto*), tiene operaciones secuenciadas en el tiempo (*Modelo dinámico*), y transforma los valores (*Modelo funcional*).
Cada modelo contiene referencia a entidades de otros modelos.
Los tres modelos no son completamente independientes - un sistema es mas que una colección de partes -, pero cada modelo puede ser examinado y entendido por si mismo completamente. La interconexión entre los diferentes modelos está limitada y expresamente acotada.

Modelo objeto

El *Modelo objeto* describe la estructura de los objetos en el sistema, - su identidad, sus relaciones, con otros objetos, sus atributos, y sus operaciones. El *Modelo objeto* provee la vista esencial sobre la cual vamos a trabajar con los modelos dinámico y funcional. Los cambios y transformaciones son minimizados a menos que algunos deban ser cambiados o transformados.
Los objetos son las unidades en la cual nosotros dividimos el mundo real, las moléculas de nuestro modelo.
Algo importante en la construcción del *Modelo objeto*, es la captura de los conceptos del mundo real que son importantes para la aplicación.
El modelo de análisis no debe contener términos computacionales a menos que se trate de un problema inherente a la computación, tal como un compilador o un sistema operativo. El modelo de diseño describe como se resuelve el problema y puede contener construcciones computacionales.
El *Modelo objeto* se representa gráficamente con diagramas de objeto, conteniendo clases de objetos. Las clases son ordenadas jerárquicamente compartiendo las estructuras y comportamientos comunes y son asociadas con otras clases. Las clases definen los valores que llevan los atributos para cada instancia y las operaciones que cada objeto realiza.

Modelo dinámico

El *Modelo dinámico* describe los aspectos de un sistema que conciernen con el tiempo y la secuencia en la que se realizan las operaciones - eventos que marcan cambios, secuencias de eventos, estados que definen el contexto de los eventos, y la organización de eventos y estados. El *Modelo dinámico* captura el control, aquellos aspectos del sistema que describen la secuencia de las operaciones que ocurren, sin tener en cuenta por que las operaciones se hacen, porque esas operaciones están o porque son implementadas.
El *Modelo dinámico* es representado gráficamente con el diagrama de estados. Cada diagrama de estado muestra el estado y la secuencia de eventos permitida en un sistema para una *clase de objeto*s. El diagrama de estado se refiere también a otros modelos. Una acción en el diagrama de estado corresponde a una función en el *Modelo funcional*. Los eventos en el diagrama de estado realizan operaciones sobre los objetos en el *Modelo objeto*.

Modelo funcional

El *Modelo funcional* describe aquellos aspectos de un sistema que conciernen con las transformaciones de los valores, - funciones, mapeos, dependencias funcionales. El modelo funcional captura que hace el sistema, sin importarle como y quien lo hace.
El *Modelo funcional* esta representado con el diagrama de flujo de datos. El diagrama de flujo de datos muestra la dependencia entre los valores y la computación de los valores de salidas desde los valores de entrada y funciones, sin importarle cuando o si esas funciones son ejecutadas. Las

funciones son invocadas como acción en el *Modelo dinámico* y son mostradas como operaciones sobre los objetos en el *Modelo objeto*.

Relación entre los modelos

Cada modelo describe un aspecto de un sistema pero contiene referencia a los otros modelos.
El *Modelo objeto*, describe la estructura de datos sobre los cuales los modelos dinámicos y funcional operan. Las operaciones en el modelo objeto se corresponden con eventos en el modelo dinámico y las funciones en el modelo funcional.
El modelo dinámico describe las estructuras de control de los objetos. Este muestra decisiones, las cuales dependen sobre los valores de los objetos y cual causa acción que cambie valores de objetos e invoque funciones.
El modelo funcional describe que funciones son invocadas por las operaciones en el modelo objeto y que acciones en el modelo dinámico. Las funciones sobre los valores de los datos especificados por el modelo objeto.
Hay ambigüedades ocasionales acerca de cual modelo debe contener una parte de la información. Esto es natural porque ninguna abstracción es únicamente un corte de la realidad.

MODELANDO OBJETOS

Un modelo objeto captura las estructuras estáticas de un sistema mediante los objetos del sistema, las relaciones entre ellos y los atributos y operaciones que caracterizan cada *clase de objeto*s. El modelo objeto es el más importante de los tres modelos antes mencionados. Nosotros enfatizamos la construcción de un sistema alrededor de objetos, que alrededor de la funcionalidad ya que un modelo orientado a objetos se corresponde mas con el mundo real y esto consecuentemente más flexible con respecto a los cambios.
El modelo objeto provee una representación gráfica intuitiva de un sistema y son excelentes para la comunicación con los usuarios y documentar la estructura del sistema.

OBJETOS Y CLASES

Objetos

Un *objeto* es simplemente algo que toma valor en el contexto de una aplicación. Ejemplos proceso numero 15460, Paulo Aguiar, etc.
Nosotros definimos un *objeto* como un concepto, como una abstracción. Los *objeto*s sirven para dos propósitos:

- para entender el mundo real.
- proveen una idea básica para la implementación en la computadora.

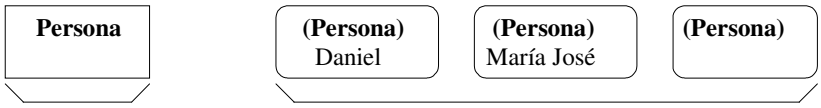
La descomposición del problema en objetos depende de la naturaleza del problema. No hay una sola representación correcta.
Todos los *objeto*s tienen una identidad y son diferenciables. Dos manzanas con el mismo color, tamaño y textura son consideradas en forma individual, una persona puede comer una y otra la restante. El termino identidad nos dice que el *objeto* es distinguible por su existencia y no por la descripción de sus propiedades que el pudiere tener.
Algunos *objeto*s son únicos, pero otras veces la palabra objeto literalmente usada se refiere a un grupo de objetos del mismo tipo. Cuando nosotros queramos precisar y referirnos a un objeto en particular vamos a usar la frase *instancia de un objeto*. También vamos a usar la palabra *clase de objeto*s para referirnos a un grupo de objetos similares.

Clases

Una *clase de objetos* describe un grupo de objetos con propiedades similares (atributos), comportamientos similares (operaciones), relaciones comunes con otros objetos, y semántica común. Ejemplos son persona, compañía, animal, procesos, etc. Cada persona tiene una edad, DNI, y pueden tener un trabajo. Cada proceso tiene un nro., un dueño, una prioridad, etc.
Los objetos de una misma clase tienen el mismo patrón de comportamiento y los mismos atributos. La mayoría de los objetos deriva su individualidad en los valores de sus atributos y sus relaciones con los otros objetos. Sin embargo, es posible que haya objetos con idénticos valores de atributos y relaciones.
Los objetos de una clase comparten una semántica común, antes y después de requerimientos de atributos y comportamientos comunes.
La pertenencia a una clase es una propiedad implícita de un objeto.
Si los objetos son el foco del modelado de objetos, por que agruparlos en clases?. Agrupando los objetos en clases, nosotros abstraemos el problema.

Diagrama de objetos

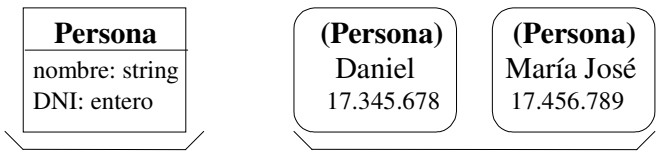
Los diagramas de objetos proveen una notación gráfica formal para modelar objetos, clases y sus relaciones con otros. Los diagramas de objetos son usados para abstraer modelos y para diseñar los programas actuales. Los diagramas de objetos son concisos, fáciles de entender, y bien manejados en la práctica. Hay dos tipos de diagramas de objetos: diagrama de clase y diagrama de instancia.
Un *diagrama de clase* es un esquema para describir muchas posibles instancia de datos. Un diagrama de clase describe clases de objetos.
Un *diagrama de instancia* describe como un set particular de objetos se relaciona con otros. Un diagrama de instancia describe objetos. Los diagramas de instancia son usados para documentar casos de test (escenarios especiales) y discutir ejemplos. Un diagrama de clase dado corresponde a infinitos diagramas de instancia.



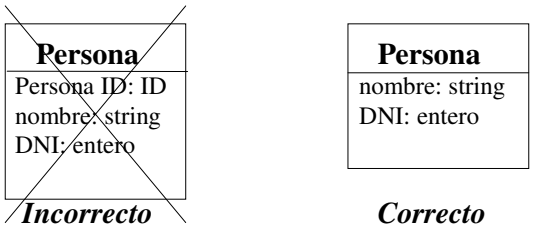
El gráfico muestra el diagrama de clase y una posible instancia descripta por él. Los objetos Daniel, María José y la persona anónima son instancias de la clase **Persona**.
Los diagramas de clase describen un caso general en el sistema modelado. Los diagramas de instancia son usados para mostrar ejemplos para clarificar un diagrama de clase complejo. La distinción entre diagrama de clase y diagrama de instancia en muy sutil; las clases y las instancias pueden aparecer en el mismo diagrama de objetos, pero en general no en muy usado mezclar clases e instancias (hay excepciones).

Atributos

Un atributo es el valor del dato que tiene el objeto de una clase. Nombre, edad, altura, peso, son atributos del objeto **Persona**. Cada atributo tiene un valor para cada instancia de objeto. Cada nombre de atributo en único en la clase a que pertenece (pero puede aparecer en otras clases).
Un atributo debe ser un valor de dato puro, no un objeto. A diferencia con los objetos, un dato no tiene identidad. Por ejemplo todas las ocurrencias de número 13 son indistinguibles, pero el string "Argentina", es un objeto con atributo nombre cuyo valor es Argentina (string).
Los atributos son mostrados en la segunda parte de la caja de clase. Cada nombre de atributo puede estar puesto con detalles opcionales, tales como el tipo y el valor por defecto. El tipo es precedido por dos puntos. El valor por defecto es precedido por el signo igual. Depende del nivel de detalle que se le quiera dar al modelo objeto se pueden omitir o no algunos detalles.
En la caja de clase hay una línea para dividir entre el nombre de la clase y sus atributos.



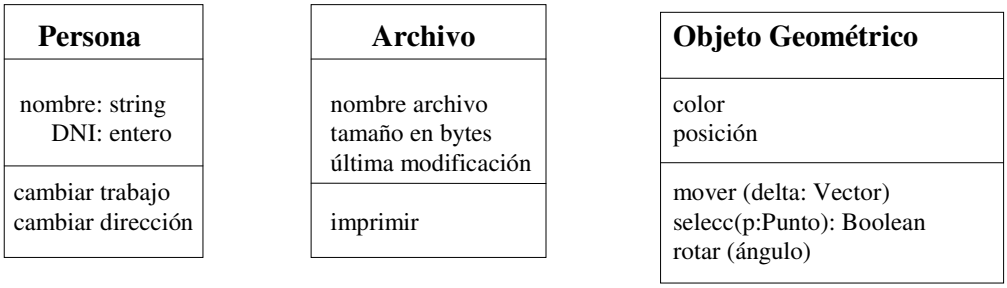
Algunas implementaciones, tales como las bases de datos, requieren que un objeto tenga un único identificador (índice), que identifique a cada objeto. Explicitar objetos no requiere un objeto modelo. Cada objeto tiene una única identidad. La mayoría de los lenguajes orientados a objeto general automáticamente en forma implícita identificadores con el cual reemplazan la referencia a los objetos (índice). **No se debe y no se necesita explicitar la lista de identificadores.**



No hay que confundir identificadores internos con atributos del mundo real. Los identificadores internos son puramente una conveniencia de implementación y no tienen cabida en el dominio del problema. Por ejemplo DNI no es un identificador interno y debe ser tratado como un atributo legítimo.

Operaciones y métodos

Una operación es una función o transformación que puede ser aplicada por o hacia los objetos en una clase.
Cada operación tiene un objeto destino como un argumento implícito. El comportamiento de una operación depende de la clase.
La misma operación puede ser aplicada a diferentes clases. Como una operación es polimórfica, ya que toma diferentes formas en diferentes clases. Un método es la implementación de una operación en una clase. Por ejemplo en la clase *Archivo*, puede haber una operación *imprimir*. Diferentes métodos son implementados para imprimir un archivo ASCII, uno binario, o una foto digitalizada. Todos los métodos hacen lógicamente la misma tarea - imprimir un archivo -, sin embargo, cada método puede implementarlo con un código diferente.
Cuando una operación tiene métodos en diferentes clase, es muy importante que todos los métodos tengan la misma *firma* - el número y tipos de argumentos y el tipo del valor del resultado. Por ejemplo, imprimir no debe tener como argumento un archivo para un método y un puntero a archivo para otro. El comportamiento de todos los métodos para una operación debe tener consistencia. Esto es mejor cuando uso el mismo número de para dos operaciones que son semánticamente diferentes, siempre y cuando ellas se apliquen a distintos tipos de clases. Por ejemplo, usar el mismo nombre para invertir a una matriz e invertir una figura geométrica. En proyectos grandes es necesario de vez en cuando acomodar los nombres, para evitar cualquier posibilidad de confusión.
Las operaciones son listadas en la parte inferior del cuadro de clases. Cada nombre de operación debe estar seguido por detalles opcionales, tales como las listas de argumentos y el tipo del resultado. Una lista de argumentos se escribe entre paréntesis, seguido por el nombre, los argumentos están separados por comas. El nombre y tipo de cada argumento puede figurar. El tipo del resultado es precedido por dos puntos y no debe ser omitido porque es muy importante distinguir las operaciones que retornan valores de las que no. Una lista de argumentos vacía muestra explícitamente que no hay argumentos. Las operaciones pueden ser omitidas en diagramas de alto nivel.



En la clase **Persona**, sus atributos son nombre y DNI, y las operaciones cambiar trabajo y cambiar dirección. Nombre, DNI, cambiar trabajo y cambiar dirección son características de **Persona**. Característica es una palabra genérica para un atributo o una operación. Similarmente, **Archivo** tiene a *imprimir* como operación. **Objeto Geométrico** tiene *mover, seleccionar* y *rotar* como operaciones. *Mover* tiene argumento delta, el cual es un vector. *Seleccionar* tiene un argumento p el cual es del tipo Punto y retorna un Booleano. *Rotar* tiene como argumento ángulo.

Links (*vínculos*) y Asociaciones

Los links y las asociaciones son los recursos para establecer las relaciones entre objetos y clases.
Un link es una comunicación física o conceptual entre dos instancias de objeto. Por ejemplo Daniel *trabaja para* Universidad de Mendoza. Matemáticamente un link es una lista ordenada de instancias de objeto. Un link es una instancia de una asociación.
Una *asociación* describe un grupo de links con estructura y semántica común. Ejemplo Persona **trabaja para** Empresa. Todos los links en una asociación conectan objetos desde la misma clase. Asociaciones y links a menudo aparecen como verbos en la definición de un problema. Una asociación describe un set de link potenciales en el mismo camino, igual que una clase describe un set potencial de objetos.
Las asociaciones son inherentemente bidireccionales. El nombre de una asociación binaria usualmente se lee en una dirección en particular, pero la asociación binaria puede moverse en cualquier dirección.
Las asociaciones son a menudo implementadas en lenguajes de programación como punteros desde un objeto a otro. Un puntero es un atributo de un objeto que contiene una referencia explícita a otro objeto. Por ejemplo, la estructura de datos para **Persona** puede contener un atributo *empleador* que puntea a un objeto **Empresa**, y el objeto **Empresa** puede contener un atributo *empleados*, que puntea a un objeto **Empleados**. Implementando las asociaciones como punteros es perfectamente aceptable, pero las asociaciones no deben ser modeladas de esta forma.
Un link muestra una relación entre dos o más objetos. Modelando un link como un puntero, disfraza el hecho que el link no es parte de cada objeto en sí mismo, pero depende de ambos de los dos. Una Empresa no es parte de una Persona, y una Persona no es parte de una Empresa. Todas las conexiones entre clases deben de hecho ser modeladas como asociaciones, aún en diseños para programas. Debemos recalcar que las asociaciones no son construcciones de bases de datos, sin embargo, las bases de datos relacionales están construidas sobre este concepto de asociación.

Sin embargo, las asociaciones son modeladas como bidireccionales pero ellas no pueden ser implementadas en ambas direcciones.

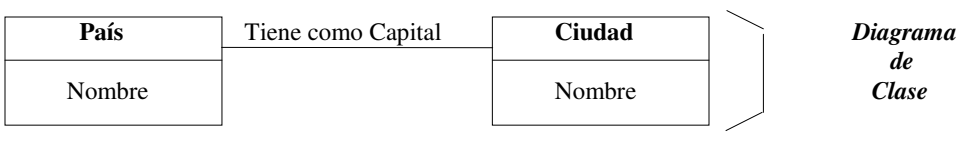
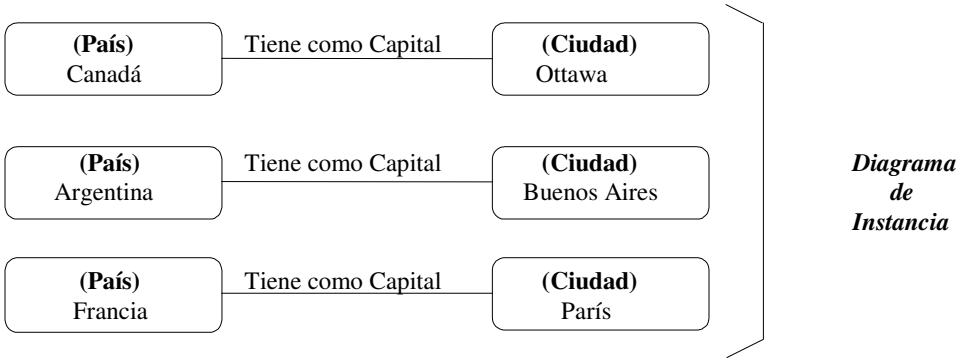
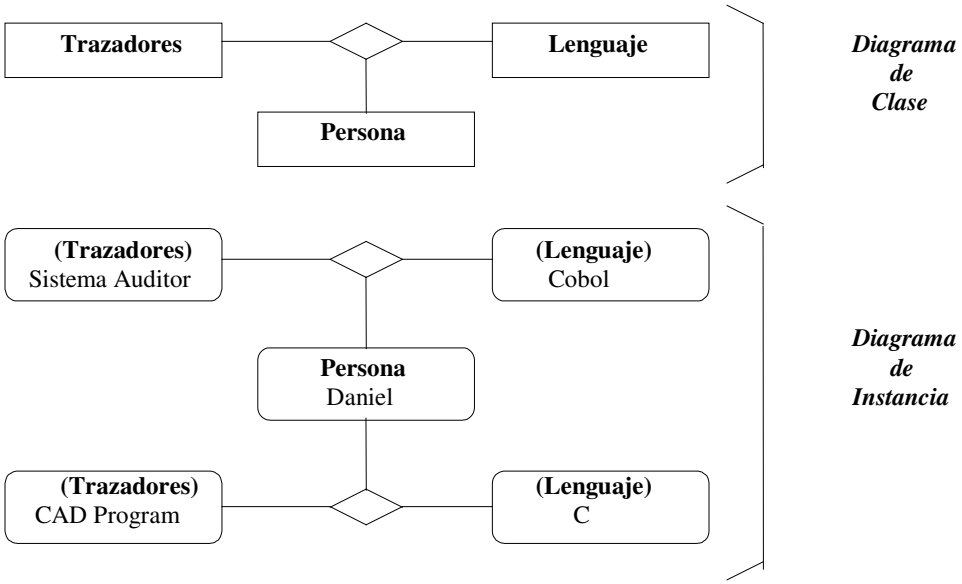


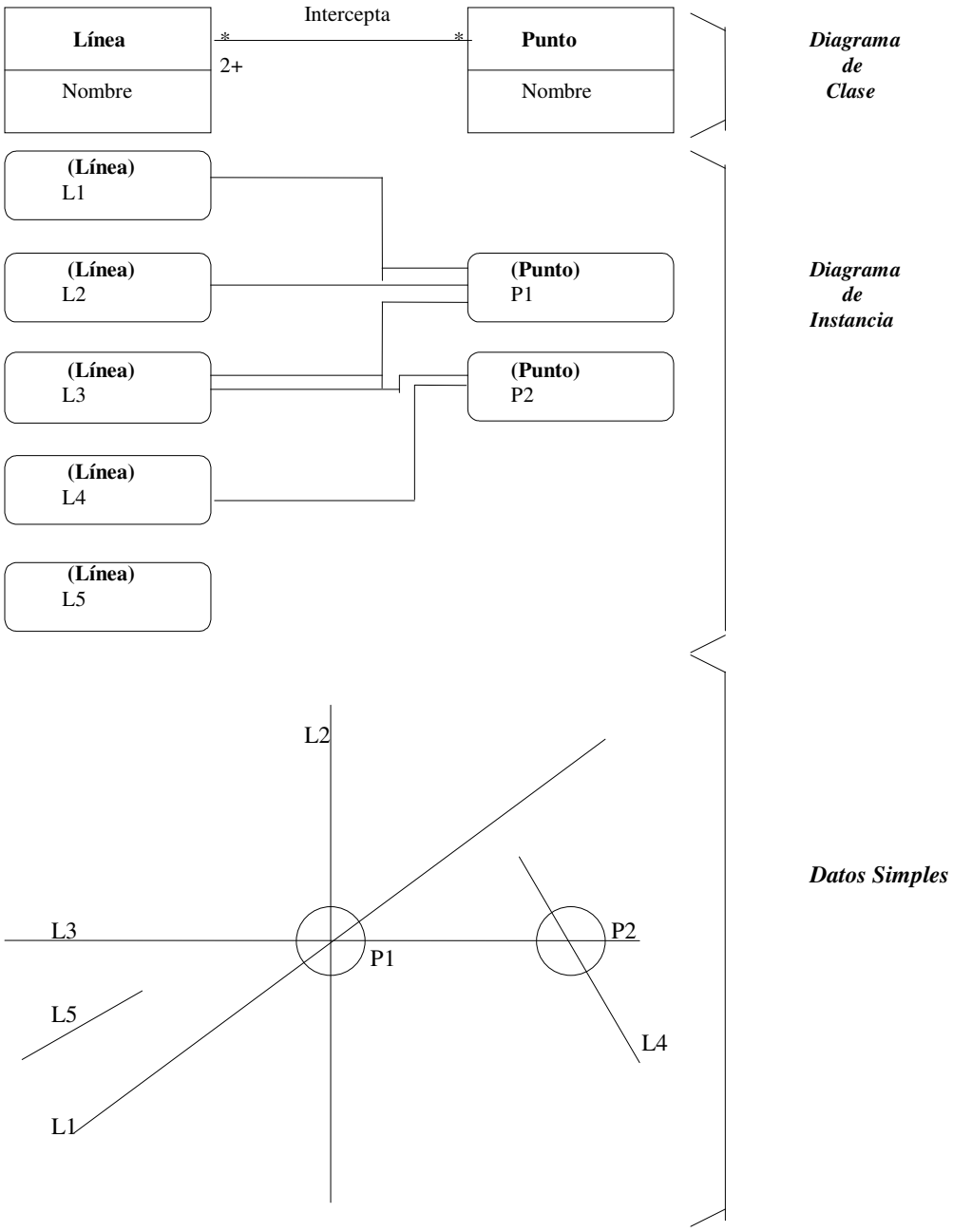
Diagrama de Clase



Como muestra el ejemplo, la notación para una asociación es una línea entre clases. Un link es una línea entre objetos. El nombre de la asociación puede omitirse si es muy obvio debido al nombre de las clases.



La figura anterior muestra una asociación ternaria: Personas, que son programadores usan lenguajes o aplicaciones de dibujo.
Esta asociación ternaria es una unidad atómica y no puede ser subdividida en asociaciones binarias sin perder información. Un programador puede conocer un lenguaje de programación y trabajar en un trazador (CAD), pero puede no usar el lenguaje cuando dibuje.
En general las relaciones ternarias y n-arias es un diamante con las líneas conectando a las clases relacionadas. El nombre de la asociación es escrito siguiendo al diamante.



La figura de la página anterior es un fragmento de un modelo objeto para un programa. Una tarea común que surge en diseño asistido por computadora (CAD), es buscar mallas de conectividad: dada una línea buscar todas las líneas que las interceptan; dado un punto de intersección, buscar todas las líneas que pasan por el; dado un área en la pantalla, buscar todos los puntos de intersección.

En el diagrama de clases, cada punto denota la intersección de dos o más líneas; cada línea tiene cero o más puntos de intersección. El diagrama de instancia muestra un posible set de líneas. L1, L2, y L3 se interceptan en el punto P1. Las líneas L3 y L4 se interceptan en el punto P2, la línea L5 no tiene ningún punto de intersección con ninguna línea y no tiene link. Las circunferencias llenas de los extremos de los links indican multiplicidad "+2". La multiplicidad especifica cuantas instancias de una clase pueden relacionar con cada instancia de la otra clase.

Multiplicidad

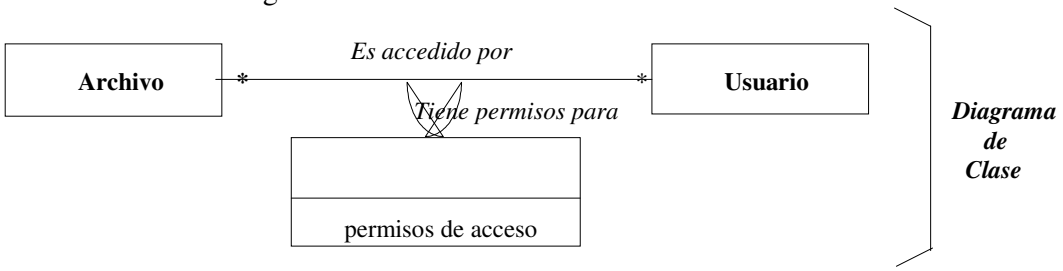
La multiplicidad especifica cuantas instancias de una clase se pueden relacionar con otra instancia de una clase asociada. La multiplicidad especifica el número de objetos relacionados. La multiplicidad es a menudo descripta como para "uno" o "muchos", pero generalmente es un subset de enteros positivos (posiblemente infinito). Generalmente el valor de multiplicidad es un intervalo continuo, pero puede ser también un intervalo discontinuo.
Hay muchos símbolos que indican multiplicidad:

- Un símbolo de multiplicar indica muchos, pudiendo ser cero o más.
- Una línea sin indicador de multiplicidad indica una relación uno a uno.
- Una relación entre objetos discontinua se indica con números en los extremos de la misma, pudiendo ser cero o uno.

Conceptos avanzados de Links y Asociaciones

Atributos de un Link

Un atributo es una propiedad de un objeto en una clase. Similarmente, un atributo de un link es una propiedad de el link en la asociación. Cada atributo de link tiene un valor para cada link, como se muestra en la figura.

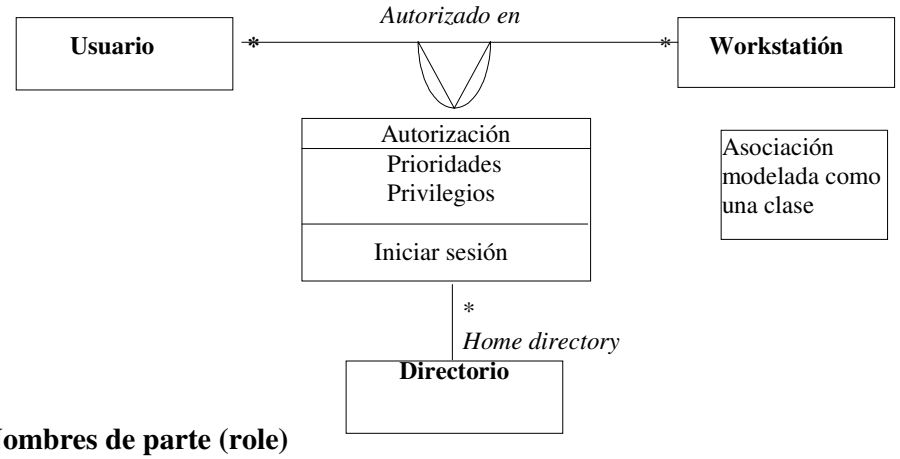


/etc/termcap	(read)	Contaduría
/etc/termcap	(read-write)	Administrador
/urs/contaduria/.profile	(read-write)	Contaduría

En la figura anterior se muestra en los datos de abajo como son los atributos de los links.

Modelando una Asociación como una clase

Algunas veces es muy usado modelar una Asociación como una clase. Cada link se convierte en una instancia de una clase. Se agrega una caja de atributos de links que no es otra cosa que una clase y puede tener un nombre, operaciones agregado a sus atributos.

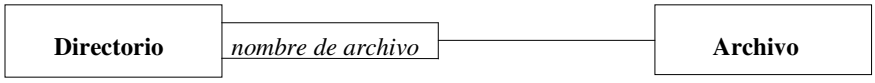


Nombres de parte (role)

Una parte (role) es uno de los extremos de una asociación. Una asociación binaria tiene dos roles, cada una de las cuales puede tener un nombre de parte (role). Este es un nombre que únicamente identifica a uno de los extremos de la asociación. Este nombre es un atributo derivado de cuyo valor es un set de objetos relacionados.
Es necesario para asociaciones entre dos objetos de la misma clase.

Calificación

El cualificador es un atributo especial que reduce el efecto de la multiplicidad de una asociación. Las asociaciones uno a muchos y muchos a muchos pueden ser calificadas.
El calificador distingue cual de un set de objetos es el final de una asociación.
Ejemplo un directorio tiene muchos archivos. Un archivo únicamente puede pertenecer a un directorio.

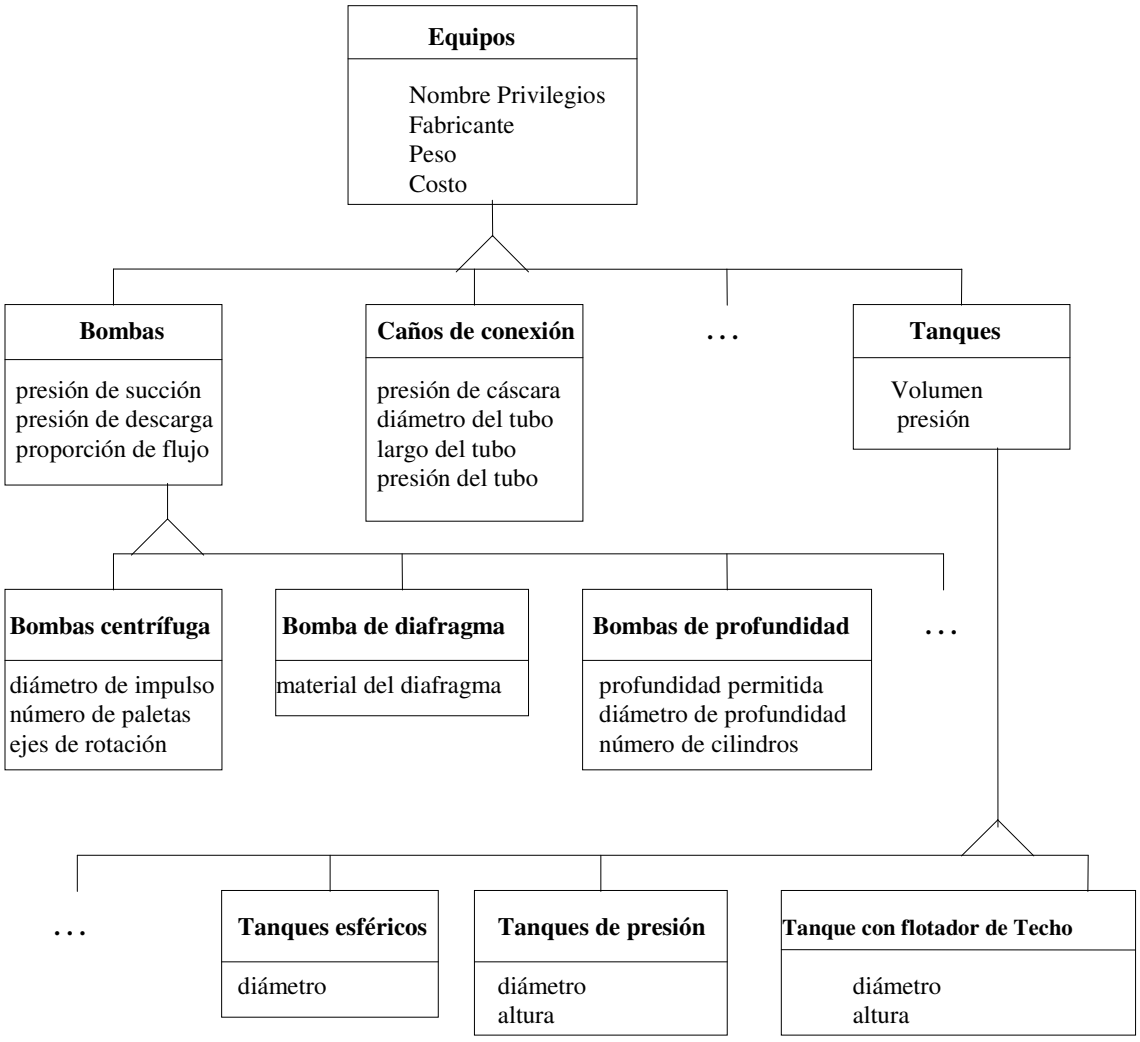


Directorio y **Archivo** son clases de objetos y *nombre de archivo* es el calificador.
La calificación reduce la multiplicidad efectiva. En el ejemplo de uno a muchos y de uno a uno. Un directorio tiene muchos archivos cada cual con un único nombre.

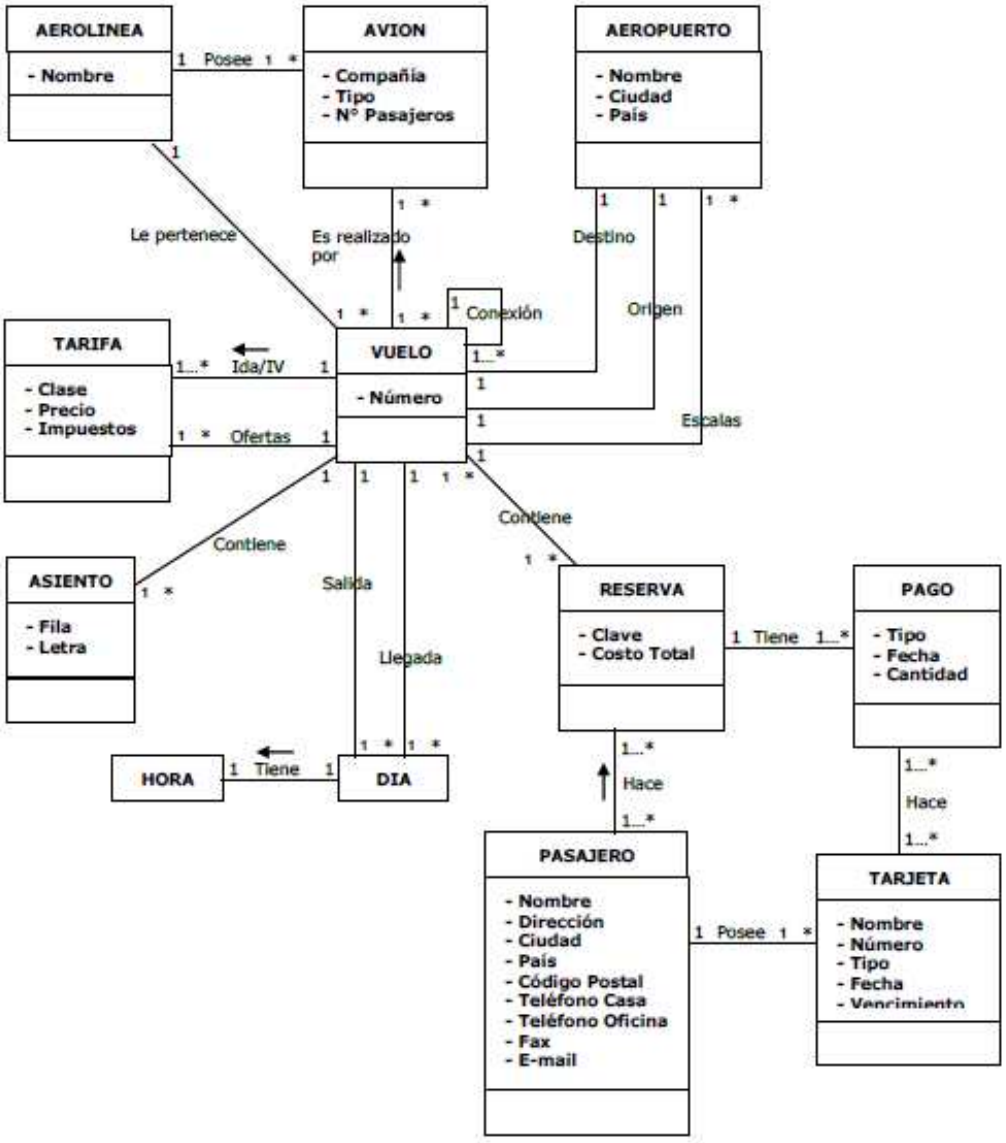
Generalización y Herencia

Conceptos generales

La generalización y la herencia son dos abstracciones poderosas para compartir similitudes entre clases mientras se preservan sus diferencias. Por ejemplo, nosotros tenemos el siguiente modelo: cada pieza de un equipo tiene un fabricante, peso y costo. Las bombas son equipos y también tienen presión de succión y un flujo de transferencia. Los tanques además de los argumentos de equipo, también pueden tener volumen y presión.
Veamos esto en un ejemplo.



Ejemplo de Modelo de Clases



Ver Ejemplo Completo

TOPICOS AVANZADOS DEL MODELADO DE OBJETOS

Agregación

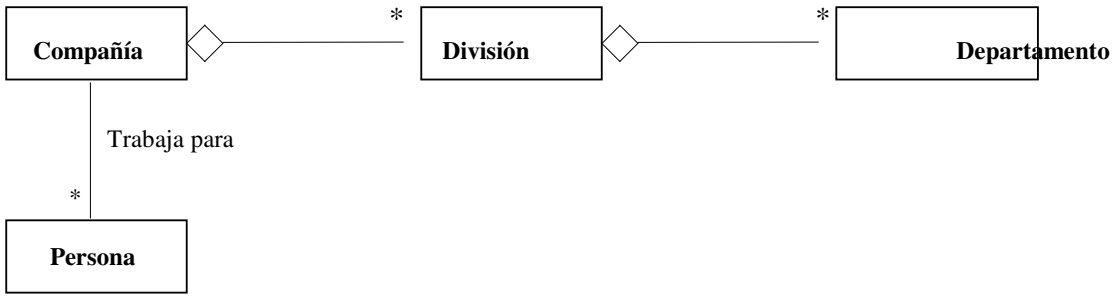
Agregación es una forma fuerte de asociación en la cual un objeto agregado esta hecho de componentes. Los componentes son partes de los agregados. Un agregado es semánticamente un objeto extendido, que es tratado como una unidad en muchas operaciones, sin embargo, físicamente este esta hecho de muchos objetos menores.
Un simple objeto agregado puede tener varias partes; cada parte-agregada relacionada es tratada como una agregación separada en orden para enfatizar la similaridad de asociación.
Las partes pueden existir o no, aparte del agregado o aparecer múltiples agregados. La agregación es transitiva en forma inherente.

Agregación versus Asociación

La agregación es una forma especial de asociación, no un concepto independiente. La agregación suma connotaciones semánticas en ciertos casos. Si dos objetos son considerados usualmente como independientes, y existe entre ellos un link, esto es una asociación.
Si dos objetos están relacionados por una parte de ellos esto es una agregación.
Algunos test para esto incluyen:

- usa la frase parte de?
- algunas operaciones son aplicadas automáticamente a estas partes?
- son propagados desde todo o algunas partes algunos valores de atributos?
- hay una asimetría intrínseca en la asociación, donde una clase de objeto esta subordinada a otra?.

La agregación incluye explosiones y expansiones de un objeto en sus partes constituyentes.



En la figura, compañía es una agregación de sus divisiones, la cual es una agregación de sus departamentos; entonces compañía es indirectamente una agregación de departamento. Compañía no es una agregación de sus empleados, ya que compañía y personas son objetos independientes de igual estatura.
El uso de agregación es una decisión que es materia de discusión y es totalmente arbitrario. Pero no es obvio que una asociación deba ser modelada como una agregación.

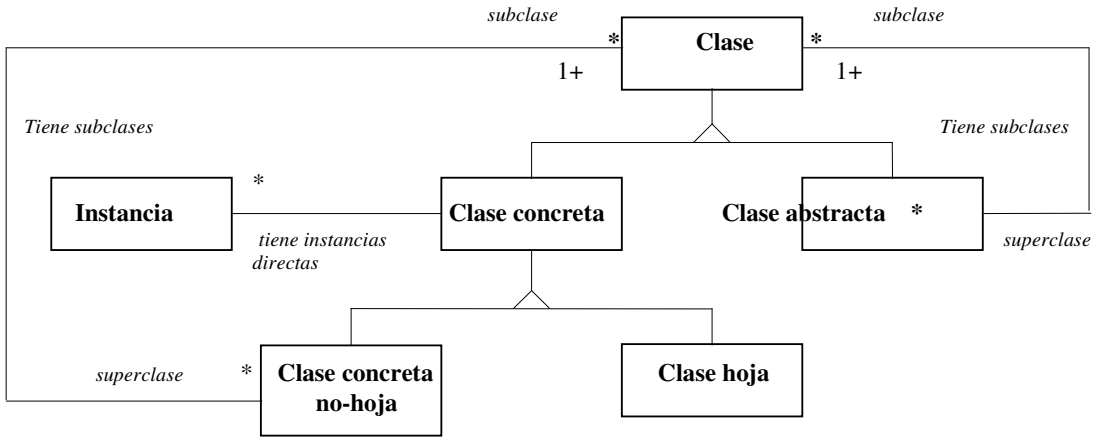
Agregación versus Generalización

Agregación no es lo mismo que generalización. Agregación relaciona instancias. Dos objetos distintos son relacionados. Uno de ellos es una parte del otro. La generalización relaciona clases y es una manera de estructurar la descripción de un objeto simple. Ambos, superclase y subclase

hacen referencia a propiedades de un objeto simple. Con la generalización, un objeto es simultáneamente una instancia de una superclase y de una subclase.
Un árbol de agregación está compuesto por instancias de objetos que son todas partes de un objeto compuesto. El árbol de generalización esta compuesto por clases que describen un objeto.

Clases abstractas

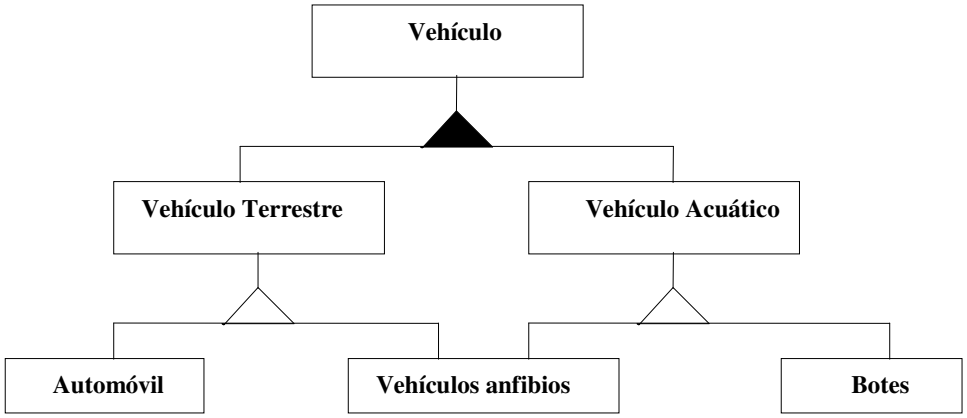
Una clase abstracta es una clase que no tiene instancias directas, pero sus clases descendentes tiene instancias directas. Una clase concreta es una clase que es instanciable; esto es que tiene instancias directas. Una clase concreta puede tener subclases abstractas, pero ella en cambio debe tener descendencias concretas. Una clase concreta puede ser una clase hoja en el árbol de herencia. La figura muestra la definición de clase abstracta y concreta.



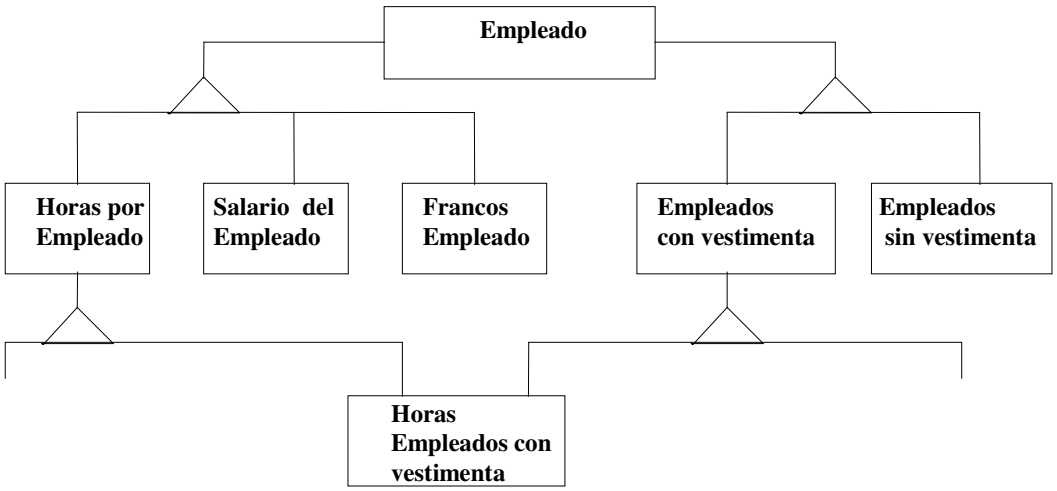
Las clases abstractas da características comunes a muchas clases. A menudo es muy usado crear una superclase abstracta para encapsular clases que participan en la misma asociación o agregación. Algunas clases abstractas aparecen naturalmente en el dominio de la aplicación, otras son introducidas artificialmente como un mecanismo de promoción de reuso del código.
Las clases abstractas son usadas frecuentemente para definir métodos que serán heredados por subclases.

Herencia múltiple
La herencia múltiple permite a una clase tener mas de una superclase y heredar características de todos los parientes. La ventaja de la herencia múltiple es el poder que da en la especificación de clases e incrementar las posibilidades del reuso. La desventaja es la perdida de simplicidad conceptual y de implementación.

Definición
Una clase puede heredar características de mas de una superclase. Una clase con mas de una superclase es llamada clase *acoplada (join class).* Una característica común de las superclases ancestras es solo heredada y fundida en una sola si: es exactamente la misma. Sino se presta luego para ambigüedades.



Cada generalización debe cubrir una sola propiedad, por ejemplo un vehículo puede viajar.
Si una clase puede ser refinada en varias dimensiones distintas e independientes, se puede usar múltiple generalización. Hay que recordar que el contenido de un modelo objeto es conducir a la solución de la aplicación, por eso si no es posible listar todas las generalizaciones posibles, se debe mostrar solamente las más importantes.



La generalización de subclases puede ser o no disjunta. Esto es por ejemplo, vehículo terrestre y acuático se superponen porque hay vehículos que viajan en el agua y en la tierra. Horas por empleado, salario del empleado y francos del empleado son disjuntos. Cada empleado debe tener exactamente uno de ellos.
Un triángulo hueco indica subclases disjuntas. Un triángulo lleno indica subclases superpuestas.

Modelado de Clases

Dado un sistema de la vida real, ¿cómo decide que clases usar?

- Los términos usados por usuarios y desarrolladores para describir el sistema son clases candidatas.
- Para cada clase ¿cuáles son sus responsabilidades? ¿están balanceadas entre las clases?
- ¿Qué atributos y operaciones necesita cada clase para llevar a cabo sus responsabilidades?

Identificación de Sustantivos: Un ejemplo de una biblioteca

Una biblioteca contiene libros y revistas.
Puede haber varias copias de un libro.
Algunos de los libros son reservados sólo para préstamos a corto plazo.
Todos los otros pueden ser prestados a cualquier miembro de la biblioteca por tres semanas. Los miembros de la biblioteca pueden normalmente solicitar hasta seis items de una vez, pero miembros del staff pueden solicitar hasta doce items a la vez.
Solamente miembros del staff pueden obtener prestado revistas.
El sistema debe conservar el registro de cuando son prestados y devueltos los libros y revistas, forzando las reglas de la biblioteca.

Clases Candidatas

Biblioteca	Nombre del Sistema
Libro	
Revista	
Copia	
Préstamos A Corto Plazo	evento
MiembroDeBiblioteca	
Semana	medida
Item	libro o revista
Tiempo	término abstracto
MiembroDelStaff	
Sistema	término general
Regla	término general

Relaciones entre Clases

Libro	es un	Item
Revista	es un	Item
Copia	es	una copia de Libro
MiembroDeBiblioteca		
Item		
MiembroDeStaff	es	un MiembroDeBiblioteca

¿Es el Item necesario?

Operaciones

MiembroDeBiblioteca	pide prestado	Copia
MiembroDeBiblioteca	devuelve	Copia
MiembroDeStaff	pide prestado	Revista
MiembroDeStaff	devuelve	Revista

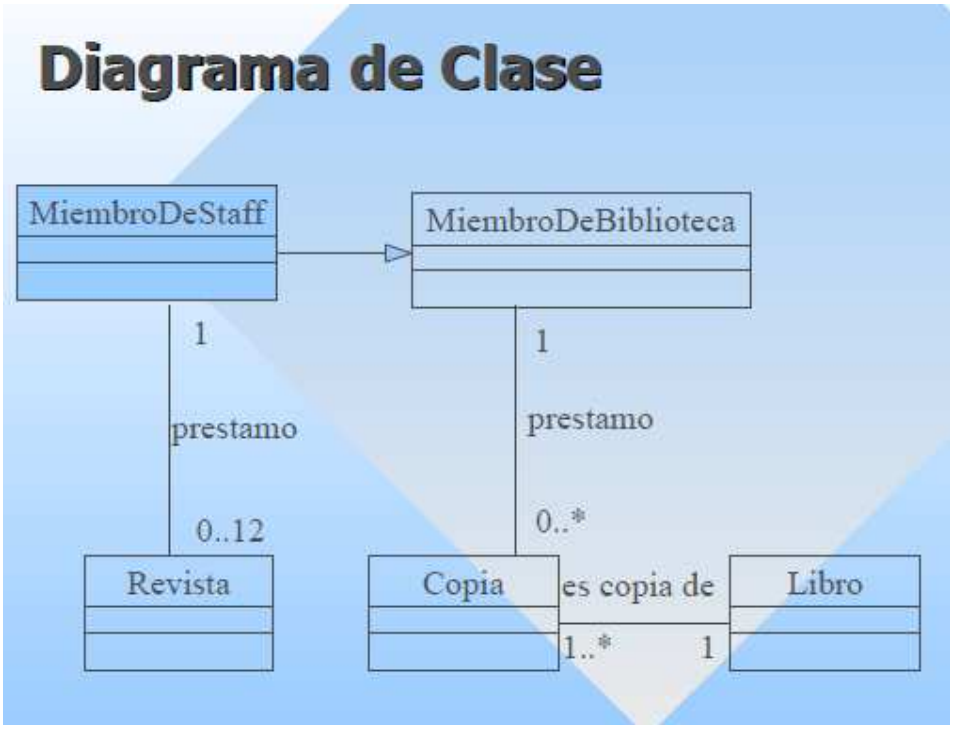


Diagrama de Clase
MiembroDeStaff
MiembroDeBiblioteca
1
prestamo
0..12
1
prestamo
0..*
Revista
Copia
es copia de
Libro
1..*
1

Test de Clases

- Uniformidad: cada objeto debe poseer iguales características y comportamiento.
- Atributos: toda clase debe poseer atributos.
- Comportamiento: toda clase debe poseer comportamiento.
- "O exclusiva": la definición de clase no debe poseer "o" exclusiva.
- No debe ser una lista de objetos de entre los que cumplen con la definición.
- Toda clase debe pertenecer al dominio del problema.

Fuentes de clases

Abott (83): extracción de objetos y clases de la descripción textual del problema (nombres propios o impropios adjetivos.)

Ross (1987): personas, lugares, cosas, organizaciones, conceptos y eventos.

Shaleer y Mellor (1988): cosas tangibles, roles, incidentes, interacciones y especificaciones.

Yourdon (1990): jerarquías (clase de) y (parte de), otros sistemas, dispositivos, eventos que deben ser recordados, roles, lugares y unidades organizativas.

Martin (1993): cosas tangibles e intangibles, roles, juicios, interacciones eventos, etc.

Consejos Prácticos

- No comience construyendo el modelo de objetos por las clases, asociaciones y herencia. Primero de debe entender el problema a resolver. El contenido del diagrama de objetos es el que conduce a la solución.
- Mantener en lo posible el modelo muy simple. Sin complicaciones innecesarias.
- Tener mucho cuidado en la elección de los nombres. Los nombres son muy importantes y tienen connotaciones muy poderosas. Los nombres deben ser descriptivos, cortos y no deben ser ambiguos. Los nombres no deben estar basados en algún aspecto de un objeto. Elegir buenos nombres es uno de los aspectos más difíciles del modelado de objetos.
- Use en lo posible asociaciones calificadas.
- Siempre documente su modelo objeto.
- Procure evitar las relaciones ternarias o n-arias, en la mayoría de los casos, estas pueden descomponerse en relaciones binarias, con posibles calificaciones y atributos de link.
- No suprima la revisión del modelo objeto. Siempre se requieren múltiples iteraciones para clarificar nombres, reparar errores y detalles, corregir estructuras, etc.
- Tratar de que otros revisen el modelo, para enfocar el problema de otra forma.

MODELO DINAMICO

Las relaciones temporales son difíciles de entender. Un sistema puede ser mejor entendido por una primera examinación de la estructura estática.
Esto es, la estructura de los objetos y sus relaciones con otras en un momento. Luego examinamos los cambios de los objetos y sus relaciones sobre el dominio del tiempo y los cambios en el modelo dinámico, en contraste con el modelo objeto.

Casos de uso

Una forma de describir los requisitos iniciales del usuario, durante la fase de conceptualización, es construir casos de uso del sistema, descritos inicialmente por Jacobson en 1987 y actualmente incorporados a la mayor parte de las metodologías de AOO.

Los Casos de Uso sirven justamente para capturar el comportamiento del sistema. Se define como "*una descripción de un conjunto de secuencias de acciones, incluyendo variantes, que ejecuta un sistema para producir un resultado observable de valor para un actor*." Cada secuencia representa la interacción de los elementos externos al sistema (los actores) con el propio sistema.

Un caso de uso especifica **qué** hace una parte del sistema, pero no **cómo** lo hace. Un caso de uso comienza cuando un actor ingresa un estímulo al sistema, y termina cuando se completa el proceso que dicho caso de uso definía.

La descripción de un caso de uso puede hacerse en forma textual informal, o formal estructurada, o pseudocódigo. Basta que su descripción sea suficientemente clara y unívoca, tanto para el usuario como para el desarrollador. Es importante que en esta descripción incluya cómo y cuándo empieza y acaba el caso de uso, cuándo interactúa con los actores y qué objetos se intercambian.

Actores

Los actores son fundamentalmente **roles** que juegan los agentes externos al sistema -sean usuarios, dispositivos u otros sistemas- y que interactúan con éste. Es importante no confundir un usuario (una persona real que interactúa con el sistema) con un actor, que es un rol determinado que puede ejercer un usuario. Por ejemplo, una misma persona puede utilizar el sistema para controlar lo que otros usuarios hacen (actor Controlador) o para hacer uso del sistema (actor Usuario).

El único tipo de relación que puede darse entre un actor y un caso de uso es una *asociación*, que va a representar la comunicación entre ambos.

Relaciones entre los Casos de Uso

Las relaciones que se pueden dar entre casos de uso son:

- Generalización y Especificación (o simplemente Herencia)

Implica que un caso de uso hijo hereda el comportamiento y el significado del caso de uso padre, y el hijo puede añadir o redefinir el comportamiento del padre.

- Inclusión

Se da cuando un caso de uso base incorpora explícitamente el comportamiento de otro caso de uso en el lugar especificado en el caso base. Esto nos evita describir una secuencia de acciones, poniendo este comportamiento común en un caso de uso aparte.

- Extensión

Sería imposible expresar todos los detalles de un caso de uso en una única secuencia, por tanto toda variante a esta secuencia se incorpora en un *punto de extensión*, a partir del cual comienza la misma. Esto nos relaciona dos o más casos de uso a través de relaciones de extensión.

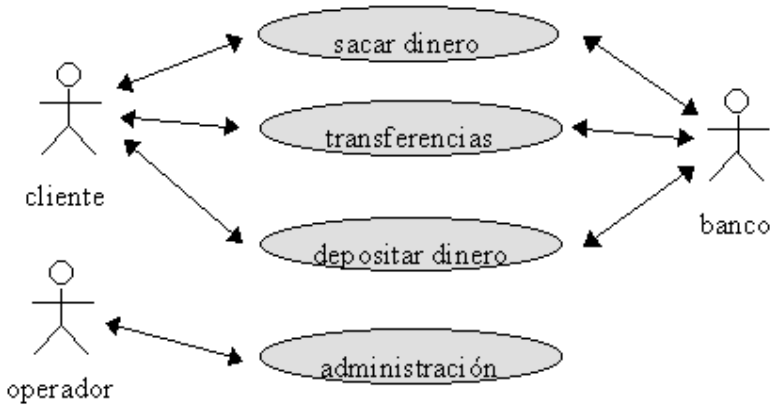
Si bien las relaciones de extensión e inclusión pueden parecerse, hay que tener claro que la diferencia radica en que la inclusión, el caso de uso base ejecutará *necesariamente* al caso de uso que es incluido, mientras que en la extensión, el caso de uso base ejecutará al extendido *opcionalmente*.

Por que usar casos de uso.

1. Son un foro de intercambio de opiniones entre expertos, desarrolladores y usuarios.
2. Permiten abordar y comprender el elemento que se modela, de acuerdo a cómo el usuario lo utilizará
3. Permiten validar su implementación a lo largo del desarrollo.

Los casos de uso proporcionan un medio para que los desarrolladores, los usuarios finales y los expertos del dominio lleguen a una comprensión común del sistema. Permitirán validar y verificar el sistema mientras evoluciona a lo largo del desarrollo. Por tanto se elaboran principalmente durante la captura de requisitos, pero se acude a ellos durante todo el desarrollo.

Un caso de uso está formado por una serie de interacciones entre el sistema y un *actor* (una entidad externa, ejerciendo un rol determinado), que muestran una determinada forma de utilizar el sistema. Cada interacción comienza con un evento inicial que el actor envía al sistema y continua con una serie de eventos entre el actor, el sistema y posiblemente otros actores involucrados.



Un caso de uso puede ser descrito en lenguaje natural, mediante trazas de eventos o mediante diagramas de interacción de objetos.

Los casos de uso describen las distintas formas de utilizar el sistema, visto desde su exterior, es decir, desde el punto de vista del usuario. Para realizar los diagramas de casos de uso se puede proceder como se describe a continuación:

Establecer los límites del sistema. Hay que determinar qué objetos forman parte del sistema, qué objetos interactúan con él y cuáles no. Los casos de uso consideran el sistema como una caja negra, es decir, no entran a describir su estructura interna.

Determinar los actores que interactúan con el sistema. Un actor es un rol o papel que juega un objeto externo en su relación con el sistema. Para determinar los actores debemos observar los objetos físicos que interactúan con el sistema, que en muchos casos juegan múltiples roles. Por ejemplo, una misma persona puede ser Usuario, Operador y Administrador de un cierto sistema. Cada rol es un actor diferente.

Para cada actor, **determinar las diferentes formas en las que usa el** sistema. Por cada una de esas formas tendremos un caso de uso.

No debería haber demasiados casos de uso en un sistema. Entre 5 y 10 casos bastan normalmente para mostrar los usos principales del sistema. Si no es así, es necesario utilizar un menor nivel de detalle. En cada caso de uso se debe escoger un nivel de detalle similar.

Identificar el evento inicial que comienza el caso de uso.

Determinar la condición de terminación que finaliza el caso de uso. A menudo un caso de uso puede ser descrito con diferentes niveles de detalle.

Describir la situación típica en la que ocurre el caso de uso. Si hay variaciones o excepciones, es necesario describirlas también. Basta con utilizar lenguaje natural, un caso de uso no necesita ser demasiado formal.

Diagramas de Casos de Uso

Un diagrama de caso de uso muestra un conjunto de casos de uso, actores y relaciones. Nos permite hacerlo abarcable y comprensible.

Para definir un diagrama es necesario:

a. Identificar bien a los actores en torno al sistema
b. Organizarlos jerárquicamente
c. Ponerles nombre
d. Considerar el comportamiento que los actores esperan (reflejado en un caso de uso)
e. Considerar la jerarquía, reutilización y comportamiento variante de los casos de uso
f. Agregar a los casos de uso las notas necesarias para hacerlo comprensible

Los diagramas de casos de uso no permiten hacer ingeniería directa ni inversa, dado que no especifican el comportamiento del sistema, sino que sólo lo reflejan.

Diagrama de subsistemas.

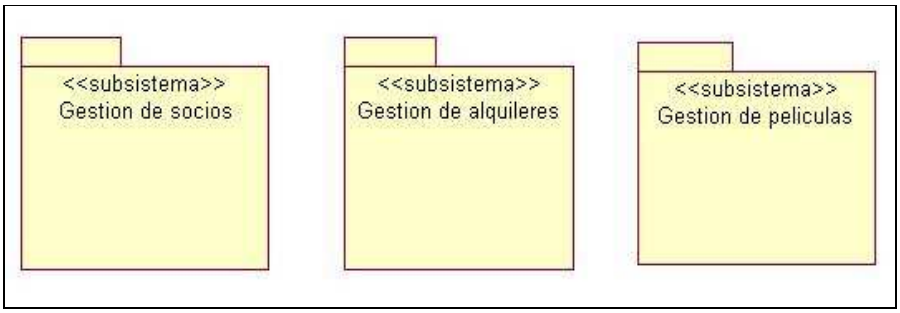


Diagrama de casos de uso del subsistema Gestión de socios

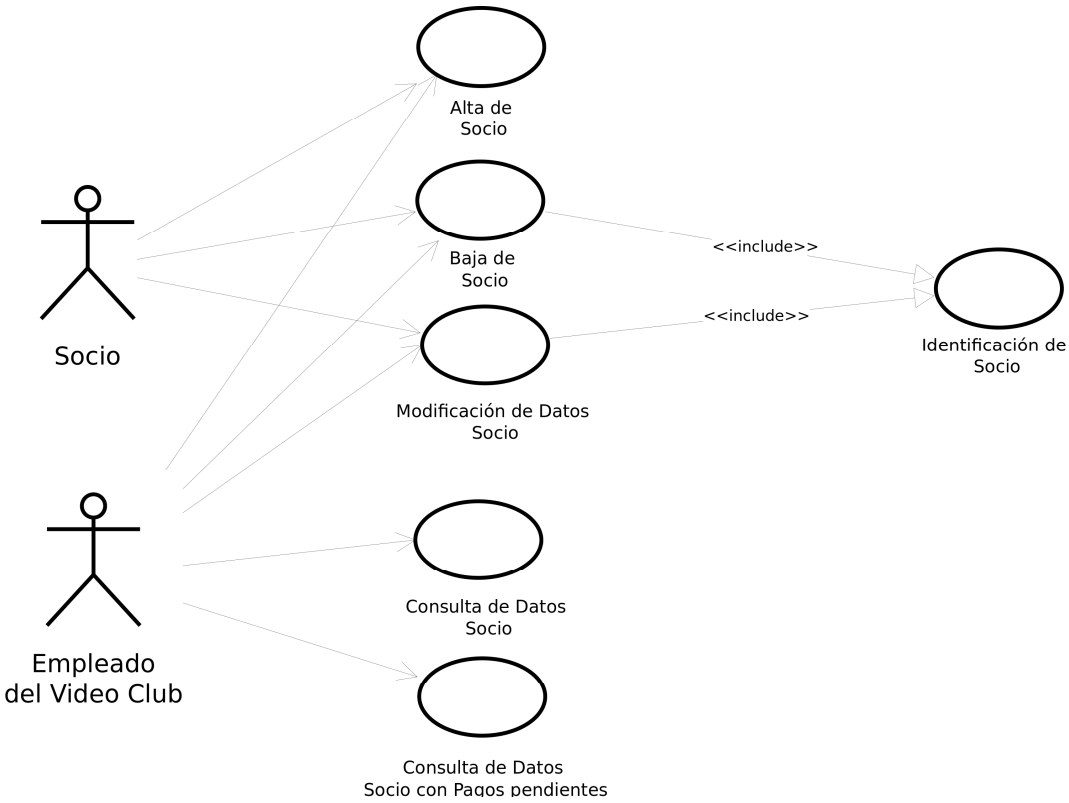


Diagrama de casos de uso del subsistema Gestión de películas

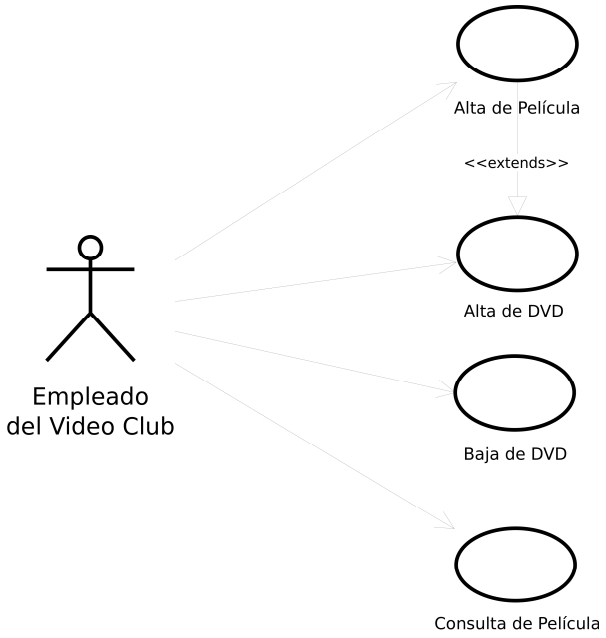
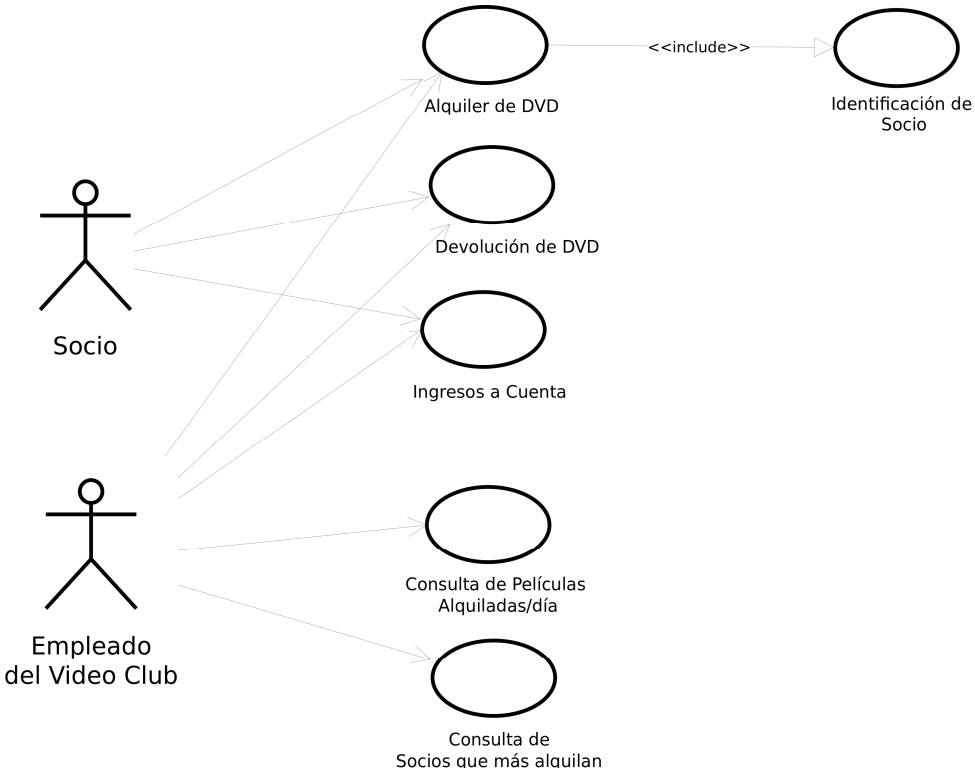


Diagrama de casos de uso del subsistema Gestión de alquileres



Actor	*<< Nombre del Actor >>*			<< Identificador >>	
Descripción	*<< Una breve descripción del Actor >>*				
Características	*<< Características que describen al actor >>*				
Relaciones	*<< Relaciones que posee el actor con otros actores del sistema >>*				
Referencias	*<< Elementos del desarrollo en los que interviene el Actor (Caso de Uso, Diagrama de secuencia, ... >>*				
Autor	*<< Esta línea se podría repetir para mantener una historia de cambios en la descripción del actor >>*	**Fecha**		**Versión**	

Atributos		
Nombre	**Descripción**	**Tipo**
<< Listado de los atributos principales del actor, incluyendo su nombre, una pequeña descripción del atributo y su tipo >>		

Comentarios
<< Comentarios adicionales sobre el actor >>

Caso de Uso	Alquilar Artículo			CU2
Actores	Cliente (iniciador), Cajero			
Tipo	esencial			
Referencias	RFA1, RFA2, CU6			
Precondición				
Postcondición	El alquiler queda registrado, junto con su pago y el de los posibles recargos pendientes o la operación cancelada			
Autor	Larman	**Fecha**	**Versión**	2

Propósito
Registra un alquiler y su pago

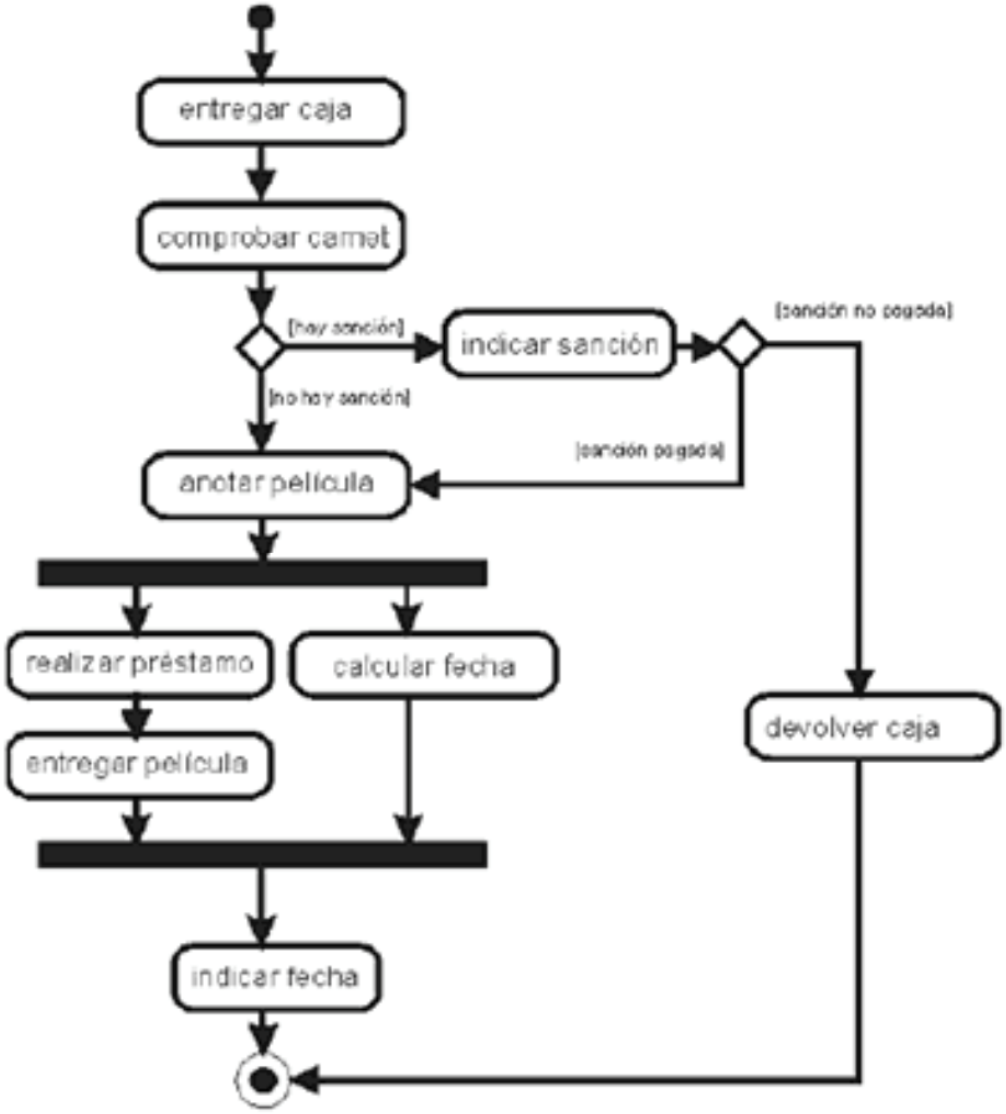
Resumen
Un Cliente llega a la caja con productos que desea alquilar. El Cajero registra los productos y recibe el pago. Al terminar la transacción, el cliente se marcha con los productos alquilados.

Curso Normal			
1	El Cliente llega al mostrador con videos para alquilar (excepcionalmente videojuegos).		
2	El Cliente presenta su tarjeta de socio y el empleado introduce su núimero de identificación en el sistema.	**3**	Presentar información sobre el Cliente y el estado de alquileres (normalmente nada en alquiler, sin penalizaciones pendientes)
4	Para cada video o juego, el Cajero graba en el sistema su número de identificación.	**5**	Presentar una lista de los títulos alquilados, fechas de devolución, precio total del alquiler y cargos por retraso en la devolución.
6	El Cajero informa al Cliente de la cantidad a abonar y le pide el pago.		
7	El Cliente paga en efectivo o a crédito.		
8	El Cajero graba el pago en el sistema.	**9**	Autorizar el pago a crédito.
		10	Generar un recibo.
11	El Cajero entrega el recibo al Cliente, que se va con los artículos alquilados.		

Cursos Alternos	
7a	El cliente no tiene suficiente dinero en efectivo. Sugerir pago a crédito, cancelación de la transacción o eliminar artículos hasta que la cantidad resultante pueda ser abonada. Volver a 7
7b	El cliente tiene recargos por retraso sin pagar y no quiere abonarlos. Antes de hacer nuevos alquileres deben pagarse los recargos: o paga todo y se vuelve a 7 o se cancela la transacción.
9a	Autorización de pago a crédito denegada, por crédito insuficiente o por fallo del servicio de autorización: borrar el pago, informar al Cajero y volver a 7.

Otros datos			
Frecuencia esperada	20 por hora	**Rendimiento**	
Importancia	alta	**Urgencia**	alta
Estado	Pendiente de revisión	**Estabilidad**	moderada

El siguiente paso es definir los diagramas de actividad. Estos diagramas se construyen a partir de la definición del curso normal y los distintos cursos alternos. De este modo se pueden visualizar de forma muy sencilla los caminos que puede tomar el caso de uso y que datos se verán en juego en esa funcionalidad.


entregar caja
comprobar carnet
[hay sanción]
indicar sanción
[sanción no pagada]
[no hay sanción]
[sanción pagada]
anotar película
realizar préstamo
calcular fecha
devolver caja
entregar película
indicar fecha

Las historias de usuario

Las historias de usuario son una forma de definir un requisito con los objetivos que cada una sea:

1. **Independiente:** Debe ser totalmente atómica en su definición.

2. **Negociable**: Ambiguas en su enunciado para poder ser debatidas. La concreción vendrá dada por los criterios de validación.

3. **Valorable**: Es importante conocer el valor que aportan al proyecto, poder valorar la cantidad de trabajo que suponen para poder planificar y gestionar el proyecto.

4. **Estimable**: Una buena historia de usuario debe de poder ser estimada con la precisión suficiente para ayudar al cliente, usuario o propietario del producto a priorizar y planificar su implementación. La estimación generalmente la realizará el equipo de trabajo y está directamente relacionada con el tamaño de la historia de usuario (una historia de usuario de gran tamaño es más difícil de estimar) y con el conocimiento del equipo de la necesidad expresada (en el caso de falta de conocimiento, serán necesarias mas fases de conversación acerca de la misma).

5. **Pequeña**: Es para poder hacer una idea rápida. Una historia de usuario debería poderse llevar a cabo durante un tiempo superior a dos días e inferior a una semana.

6. **Testeable**: Es obligatorio verificar que el software cumple con la funcionalidad acordada, se hace mediante los criterios de validación.

Para definir una historia de usuario debemos responder a la siguientes preguntas.

¿QUIÉN SE BENEFICIA?

¿QUÉ SE QUIERE?

¿CUÁL ES EL BENEFICIO?

Como **ROL** quiero **ALGO** para poder **BENEFICIO**

Aquí tenemos un ejemplo.

¿QUIEN SE BENEFICIA?

¿QUE SE QUIERE?

¿CUAL ES EL BENEFICIO?

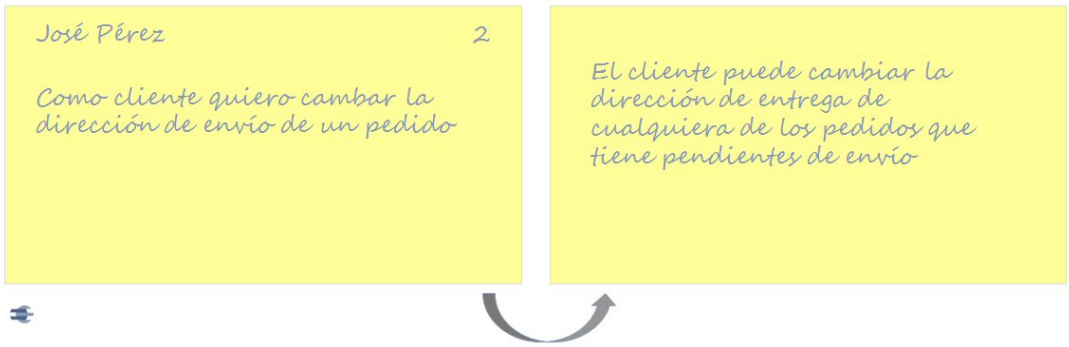
COMO CLIENTE

QUIERO PODER EDITAR EL PEDIDO

PARA CAMBIAR LA DIRECCION DEL PEDIDO

A esta frase debemos añadir los criterios de validación. Estos criterios son básicos para poder entender cada uno de los casos con los que debe tener presente en el diseño y desarrollo. En los criterios de validación deben aparecer descritos todo lo que quieres ver y poder hacer.
Siguiendo el caso anterior podría ser:

- Quiero poder ver los detalles de un pedido concreto.
- Quiero poder filtrar los pedidos anteriores por las fechas en las que se han despachado.
- En una fecha determinada, quiero poder separar los resultados anteriores por dirección.
- Quiero poder filtrar los resultados seleccionando una fecha de inicio y otra de final.
- No puedo seleccionar fechas anteriores a la creación del pedido.
- No puedo seleccionar fechas futuras.



Historia de Usuario	
Número: 1	**Usuario:** Cliente
Nombre historia: Cambiar dirección de envío	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Puntos estimados: 2	**Iteración asignada:** 1
Programador responsable: José Pérez	
Descripción: Quiero cambiar la dirección de envío de un pedido.	
Validación: El cliente puede cambiar la dirección de entrega de cualquiera de los pedidos que tiene pendientes de envío.	

Desde hace un tiempo, el Scrum apuesta para definir los **requisitos** mediante historias de usuario.

La idea principal en el punto partida de un proyecto es definir todas las historias de usuario que queremos que el software pueda ejecutar. Una vez escritas todos debemos valorar que historias tienen más importancia. Se trata de ordenar las historias de usuarios presentadas, en una lista (backlog) y definir un producto mínimo viable.

El objetivo del producto mínimo viable es poder hacer una entrega al cliente lo antes posible.
De este modo el usuario final podrá empezar a usar el software que desea lo antes posible, para poder dar feedback y así ajustar mejor que y como deseamos que evolucione nuestra aplicación.

Requisitos vs Casos de uso vs Historias de Usuario

¿Qué diferencias hay entre estos tres conceptos?
Los tres intentan representar las características que tiene que tener un sistema, la diferencia está en el enfoque dado.

Los **Requisitos del sistema** están escritos desde la perspectiva del sistema y no en la interacción del usuario, representan las características en estado puro.

Los **Casos de Uso** están escritos como una serie de interacciones entre el usuario y el sistema. Hacen hincapié en el contexto orientado al usuario. Las características que utiliza cada usuario identificado en el sistema. Son la forma de capturar los requisitos del sistema desde el punto de vista del usuario.

Las **Historias de Usuario** sirven para describir lo que el usuario desea ser capaz de hacer. Además, las historias de los usuarios se centran en el valor que viene de usar el sistema en lugar de una especificación detallada de lo que el sistema debe hacer. Están concebidos como un medio para fomentar la colaboración.

Los requisitos y los casos de uso se pueden utilizar en desarrollos ágiles de software, pero ambos se apoyan fuertemente en las especificaciones documentadas del sistema, en lugar de colaboración tradicional de los métodos ágiles. Si no existe un enfoque integrado en la colaboración, es posible que se ahonde en una especificación detallada en el caso de uso que se convierte en la fuente de registro en lugar de un mecanismo de conversaciones.
En resumen las diferencias son: los requisitos tradicionales se centran en las operaciones del sistema, con una tendencia a la especificación detallada del sistema, los casos de uso se centran en las interacciones entre el usuario y el sistema con una tendencia similar de las especificaciones detalladas, y las historias de los usuarios se centran en el valor del cliente con una fuerte intención de fomentar la comunicación.
¿Son las historias de usuario mejor que otros tipos de especificación de requerimientos? Depende de la situación, pero en un ambiente de colaboración, la experiencia indica que claramente sí. Las historias de usuarios no hará que el proyecto sea ágil, o la falta de ellas hará que sea difícil adquirir agilidad.
Si estamos en un contexto no-ágil, ¿ayudan las historias de usuario? Depende del equipo. Si un equipo está inmerso en un proceso de desarrollo en cascada o con procesos iterativos pesados y se utiliza el enfoque de «planificación pesada y requerimientos up-front", es muy probable que influya en las historias de los usuarios, convirtiéndolas en requisitos tradicionales. Las historias de usuarios son claramente un poco vagas en cuanto a descripción y se prestan a refinamientos sucesivos, la planificación y el diseño por adelantado no es su punto fuerte. La especificación detallada a menudo es la práctica en entornos de procesos controlados y pesados, antes de que comience la codificación de manera que los requisitos se puedan utilizar para planificar el presupuesto y el proyecto entero.
Las historias de usuarios (cuando se utiliza según lo previsto por algunos expertos, p.ej: Cockburn) son demasiado vagas en formato para prestarse a una documentación completa. Pero si, por el contrario, el equipo está abierto a la colaboración con los usuarios, el cliente, los analistas de negocios y los patrocinadores del proyecto y se puede o se desea tolerar el cambio, las historias de los usuarios pueden ser el método de requisitos más apropiado que además ayuda a la colaboración.
Habrá proyectos o contextos en los que unas veces será mejor utilizar historias de usuario, otras será mejor utilizar casos de uso como técnica de captura de requisitos, no por la moda del agilismo tenemos que pensar que las historias de usuario son la solución

Eventos y Estados

Un modelo objeto describe las posibles características de los objetos, atributos y vínculos que pueden existir en el sistema. Los valores de los atributos y vínculos que tiene un objeto son llamados estado.
En el tiempo, un objeto estimula a otros objetos, resultando en una serie de cambios en sus estados.
Un estímulo individual de un objeto hacia otro es un evento. La respuesta a un evento depende del estado del objeto que lo recibe y puede incluir cambios en el estado y envío de otros al que se lo envió, como así también a otros objetos.
El modelo de eventos, estados y estado de transición para una clase dada, puede ser abstraída y ser representada como un diagrama de estado. Un diagrama de estado es una red de eventos y estados, de la misma forma que un diagrama de objetos es una red de clases y relaciones.
El modelo dinámico consiste en un múltiple diagrama de estados. Un diagrama de estado por cada clase, con un importante comportamiento dinámico y mostrando el modelo de actividad para todo el sistema.

Eventos

Un evento es alguna cosa que sucede como un punto en el tiempo. Tal como una presión del botón del Mouse o la partida de un vuelo de avión. Un evento no tiene duración. Por supuesto, nada es realmente instantáneo, un evento es una ocurrencia.
Un evento puede preceder o seguir lógicamente a otro o dos eventos pueden no estar relacionados, dos eventos pueden estar casualmente relacionados, dos eventos pueden estar casualmente sin relación. Dos eventos que no se relacionan pueden ser concurrentes y no tienen efecto uno sobre el otro.
Si la ubicación física de dos eventos no es distante los podemos considerar concurrentes si no se afectan el uno con el otro.
Un evento es una vía de transmisión de información de un objeto a otro y no es un tipo de subrutina que retorna un valor.
En el mundo real, todos los objetos existen concurrentemente. Un objeto envía un evento a otro objeto y puede esperar una réplica, pero esa réplica es un evento separado bajo el control del segundo objeto, el cual puede o no elegirlo para mandárselo.

Clases de eventos

Siempre los eventos son ocurrencias únicas, pero dos o más eventos pueden ser agrupados en *clases de eventos* y se les da un nombre que indica que tiene estructura y comportamiento común.
Es una estructura jerárquica, igual que la estructura de objetos es una estructura jerárquica.
Algunos eventos son simples señales, pero las clases de eventos tienen atributos que indican la información que ellos conducen. El tiempo en que cada evento ocurre, es un atributo implícito para todos los eventos.
Un evento conduce información de un objeto a otro. Algunas clases de eventos pueden ser simplemente señales que algo ha ocurrido, mientras que otras clases conducen valores de datos.
Los valores de los datos enviados por un evento son sus atributos. Los atributos se muestran entre paréntesis en los diagramas de clases de eventos.

Partida de vuelos (línea aérea, nº de vuelo, ciudad)
Botón del Mouse presionado (botón, ubicación)
Dígito discado (dígito)

Escenarios y Señales de eventos

Un escenario es una secuencia de eventos que ocurren durante una ejecución particular de un sistema.
El alcance de un escenario puede variar y puede incluir todos los eventos de un sistema, como así también algunos eventos generados por ciertos objetos del sistema.
Un escenario es un registro histórico de ejecución de un sistema, o un experimento de ejecución de un sistema propuesto.

Se levanta el tubo
el tono comienza
se marca un dígito (2)
el tono finaliza
se marca otro dígito (0)
se marca otro dígito (1)
se marca otro dígito (8)
se marca otro dígito (7)
se marca otro dígito (2)
aparece la señal de que llama
el teléfono marcado suena
contesta el interlocutor
el teléfono deja de sonar
la señal de que llama desaparece
los teléfonos están conectados
el interlocutor corta
los teléfonos se desconectan

Cada evento transmite información desde un objeto a otro. La secuencia de eventos y los objetos que intercambian eventos, pueden ser representados en un escenario llamado evento trace (señales de eventos).

Este diagrama muestra en forma vertical los objetos y los eventos en forma horizontal.

Persona que llama	Línea telefónica	Interlocutor
Se levanta el tubo		
el tono comienza		
se marca un dígito (4)		
el tono finaliza		
se marca otro dígito (2)		
se marca otro dígito (0)		
se marca otro dígito (1)		
se marca otro dígito (8)		
se marca otro dígito (7)		
se marca otro dígito (2)		
aparece la señal de que llama	el teléfono marcado suena	
	contesta el interlocutor	
el teléfono deja de sonar	la señal de que llama desaparece	
los teléfonos están conectados	los teléfonos están conectados	
	el interlocutor corta	
los teléfonos se desconectan	los teléfonos se desconectan	
el que marcó corta		

Estados

Un estado es una abstracción de los valores de atributos y los links de un objeto. Sets de valores se agrupan en un estado de acuerdo a las propiedades que toman efecto con el comportamiento del objeto.
Un estado especifica la respuesta de un objeto a una entrada de un evento. La respuesta a un evento recibida por un objeto puede variar cuantitativamente dependiendo de los valores de los atributos.
Un estado se corresponde con el intervalo entre que dos eventos son recibidos por un objeto.
Los eventos representan puntos en el tiempo. Los estados representan intervalos de tiempo.
Un estado tiene duración, ocupa un intervalo de tiempo. Un estado a menudo es asociado con una actividad continua, tal como la señal de que nos están llamando, o, una actividad que toma tiempo en completarse, tal como volar desde Mendoza a Bs. As.
Los estados y los eventos son duales uno de otro, un evento separa dos estados y un estado separa dos eventos.
Un estado es a menudo asociado con el valor de un objeto satisfaciendo alguna condición.
Eventos y estados dependen del nivel de abstracción usado. Por ejemplo, un agente de viaje planea un itinerario que trata a cada segmento de una jornada como un evento simple.

Un ejemplo de un estado sería.

Estado: *alarma sonando*
Descripción: la alarma del reloj suena indicando la hora prefijada.
Secuencia de eventos que producen el estado:

setear la alarma (hora indicada)
cualquier secuencia no incluyendo *borrar alarma*
hora actual = hora indicada

Condición que caracteriza al estado:
alarma = on, y hora indicada <= hora actual <= hora indicada + 20 segundos, y ningún botón debe ser presionado mientras dure la alarma

Eventos aceptados en el estado:

evento	**acción**	**próximo estado**
hora actual = hora indicada +20 seg.	Reset alarma	*normal*
botón presionado (cualquiera)	Reset alarma	*normal*

Diagrama de Estados

Un diagrama de estado relata eventos y estados. Cuando un evento es recibido, el próximo estado depende del estado actual, como así también del evento. Un cambio de estado causado por un evento se llama una *transición*. Un diagrama de estados es un gráfico en el cual los nodos son estados y los arcos son las transiciones rotuladas con el nombre del evento. Un estado se representa con un rectángulo redondeado con el nombre. Una transición es una flecha desde el estado que fuente al estado destino, el nombre sobre la flecha es el nombre del evento causante de la transición.
El diagrama de estado especifica la secuencia de estados causado por una secuencia de eventos.
Un diagrama de estados describe el comportamiento de una sola clase de objetos. Ya que todas las instancias de una clase tienen el mismo comportamiento (por definición), todas comparten el mismo diagrama de estado, así como también comparten las características de la clase. Pero como cada objeto tiene sus propios valores de atributos, así también cada objeto tiene su propio estado.
Un diagrama de estado puede representar un solo ciclo o un loop continuo de ciclos. Un diagrama de un sólo ciclo representa objetos con vidas finitas, tiene un estado inicial y un estado final. El estado inicial se representa con un circulo lleno, el cual puede ser rotulado con las diferentes condiciones iniciales.
El estado final se representa con un "ojo" (círculos concéntricos, el del interior lleno), el cual también puede ser rotulado con las condiciones finales.
El modelo dinámico es una colección de diagramas de estados que interactúan uno con otro a través de eventos compartidos. Representa la estructura de control de un sistema.

Un escenario es al modelo dinámico, como un diagrama de instancia es al modelo objeto.

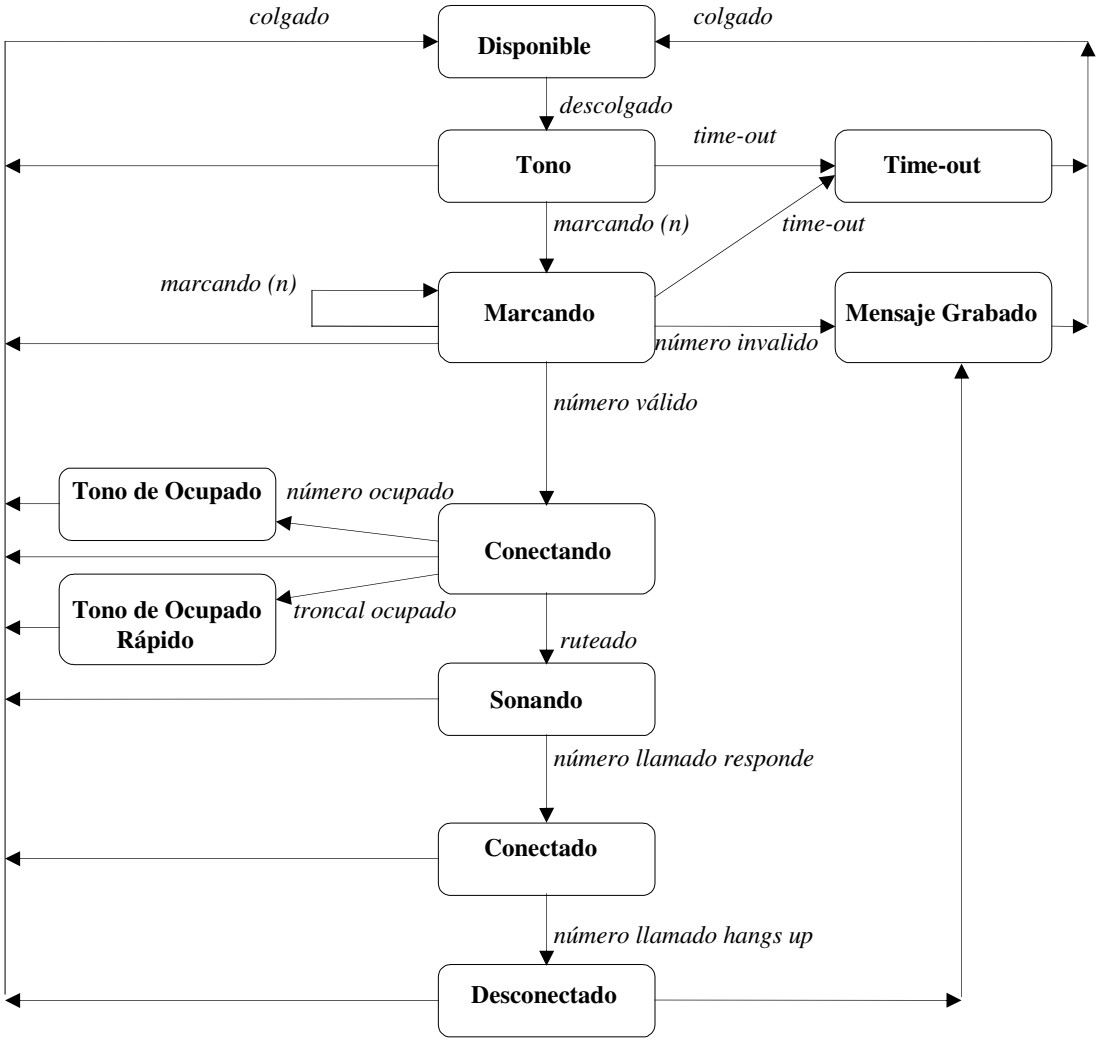
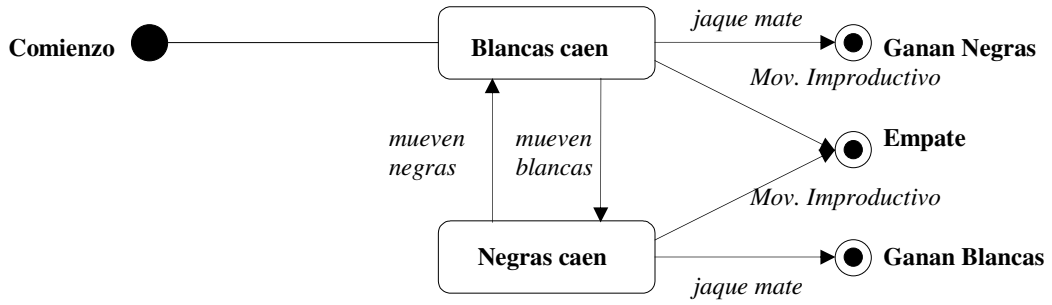


Diagrama de un sólo ciclo para un juego de Ajedrez



Condiciones

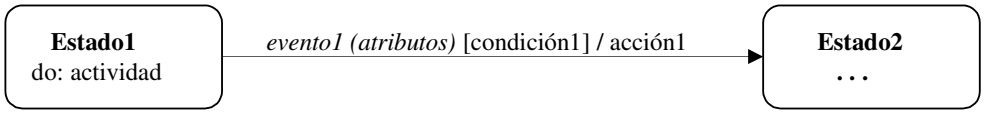
Una condición es una función Boolena de valores de Objetos. Una condición es válida sobre un intervalo de tiempo. Es muy importante distinguir *condiciones* de *eventos,* el cual no tiene duración. Un estado puede ser definido en términos de una condición.
Una condición puede ser usada como *guarda* en una transición. Una guarda de transición se chequea cuando ocurre el evento, pero únicamente si la guarda de condición es verdadera. Una condición *guarda* sobre una transición se muestran como una expresión Booleana entre corchetes siguiendo al nombre del evento.

Operaciones

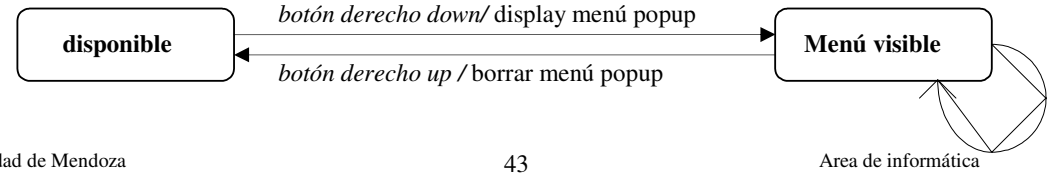
El diagrama de estado fue descripto hasta ahora como para mostrar las características de eventos y estados de un tipo de clases.

Controlando operaciones

Los diagramas de estados serían de uso limitado si sólo se usaran para describir las características de los eventos.
Una descripción de comportamiento de un objeto puede ser específicamente que hace el objeto en respuesta a un evento. Las operaciones asociadas a evento o transiciones se transforman en respuestas al estado correspondiente o evento.
Una ***actividad*** es una operación que toma un tiempo completo. Una actividad está asociada con un estado. Las actividades incluyen operaciones continuas, tal como, mostrar una imagen en la televisión, así como una operación secuencial que terminan por ellas mismas luego de un intervalo de tiempo, tal como, cerrar la válvula o ejecutar un cálculo.
Un estado puede controlar una actividad continua, tal como, un ring de teléfono que suena hasta que termine y este provoca una transición de estado.
Una ***acción*** es una operación instantánea. Una acción es asociada con un evento, representa una operación cuya duración es insignificante comparada con la resolución del diagrama de estado. Por ejemplo, desconectar una línea telefónica, puede ser una acción en respuesta al evento colgar.
En el mundo real una operación no es realmente instantánea. Si tengo cuidado una acción puede ser modelada como una actividad, con un evento de comienzo, un evento final y la posibilidad de algunos eventos intermedios.
Una operación también representa operaciones de control interno, tal como, setear atributos o generar otros eventos, por ejemplo: un contador interno.
La notación para una acción es "/" y el nombre (o descripción) de la acción, seguido al nombre del evento que lo provoca.



Ejemplo:



cursor movido/highlight ítem del menú

Ejemplo del semáforo inteligente.

Ejemplo completo de la notación para diagramas de estado con Operaciones

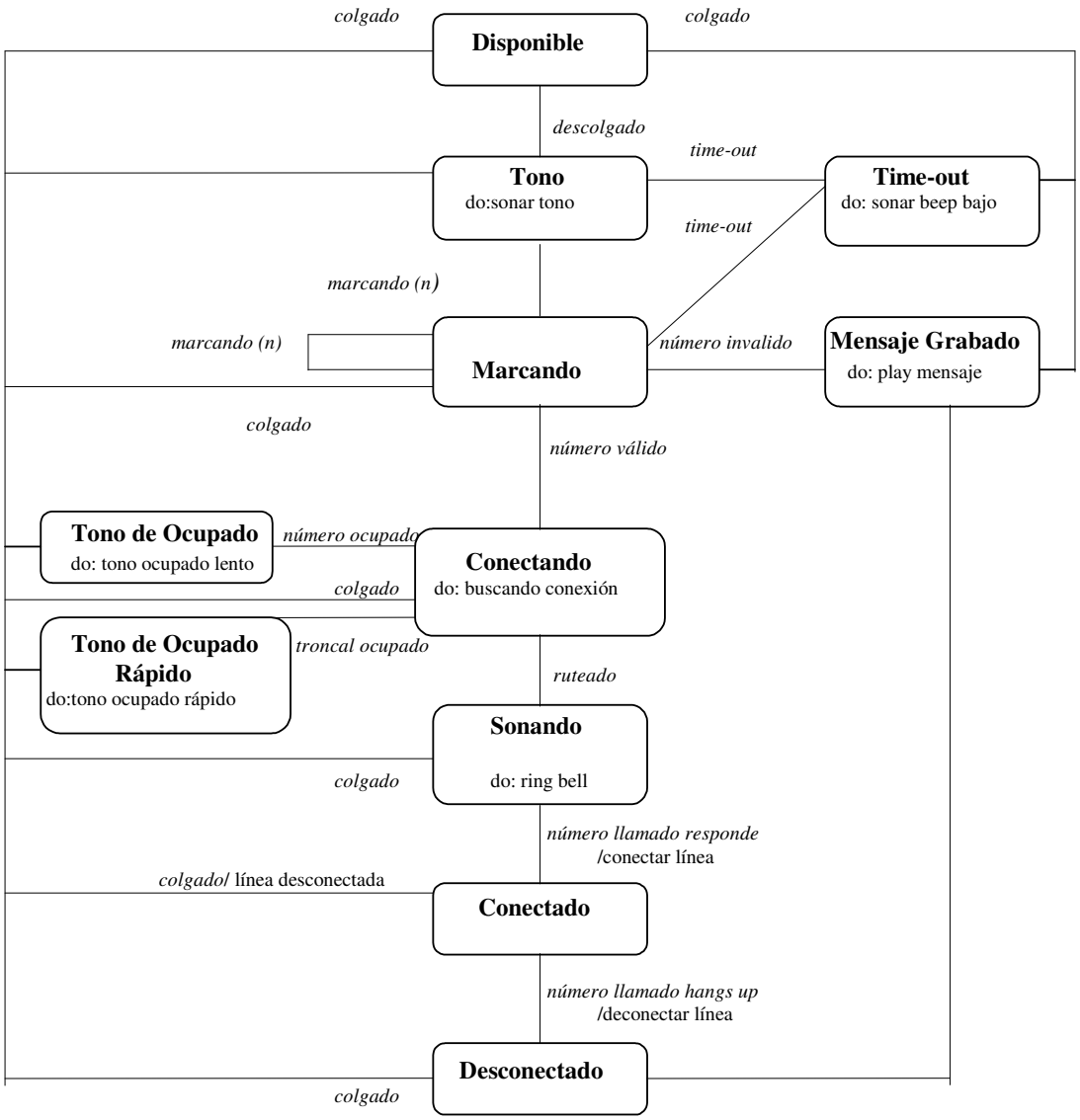


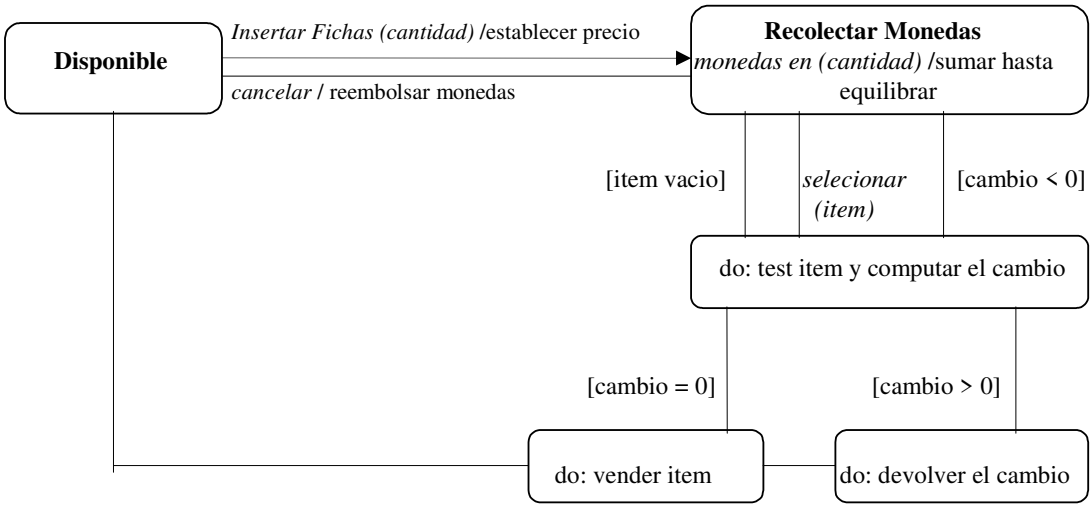
Diagrama De Estados Anidados

Los diagramas de estados pueden ser estructurados para permitir una concisa descripción de sistemas complejos. Las vías para estructurar máquinas de estado es similar a las vías para estructurar objetos: generalización y agregación.
Generalización es el equivalente a expandir actividades anidadas. Esto permite a una actividad descripta en un nivel superior expandirla a niveles menores adicionando detalles similar a una subrutina o llamado a función.
La generalización permite a los estados y eventos ser arreglados en una generalización jerárquica con herencia de estructuras y comportamiento común, similar a los objetos.
La agregación permite a un estado ser partido en componentes, interacción limitada entre ellos.

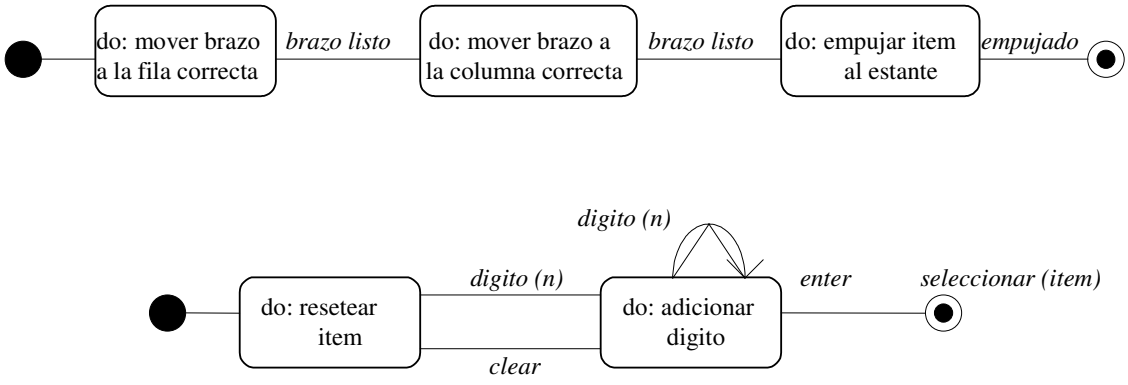
Anidando diagramas de estado

Una actividad en un estado puede ser explotada en un diagrama de estado de menor nivel, cada estado es representando un paso de la actividad.

Las actividades anidadas son diagramas de estado de un solo paso (one-shot), con una entrada y una salida, similar a una subrutina.



La figura del ejemplo muestra el diagrama de estado de nivel superior para una máquina expendedora con monedas (cafe, gaseosas, etc).
Ahora una de las actividades (*vender item*) y un evento (*selecionar item*), del diagrama pueden ser explotadas para obterner mas detalles en un diagrama de estado anidado.



Generalización de estados

Un diagrama anidado de estado es actualmente una forma de generalización de estado. La generalización es la relación "or". Un objeto en un estado en un diagrama de nivel superior debe estar exactamente en un estado en un diagrama anidado.
Este debe estar en el primer estado o en el segundo o en uno de los dos.
Los estados en el diagrama anidado son todas purificaciones del estado en el nivel superior. En el diagrama anidado se puede interactuar con otros estados. Los estados pueden tener sub-estados y heredar las transiciones de los super-estados. Cualquier transición que se aplica aun estado se aplica a todos los sub-estados.

Ejemplo de un Diagrama de Estados de una Clase en Particular.

Diagrama de Clase Socio

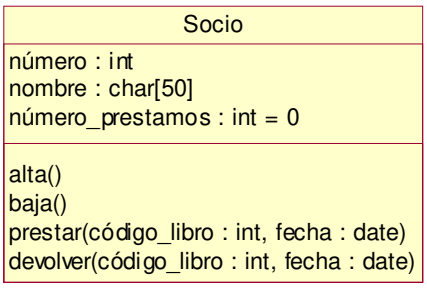
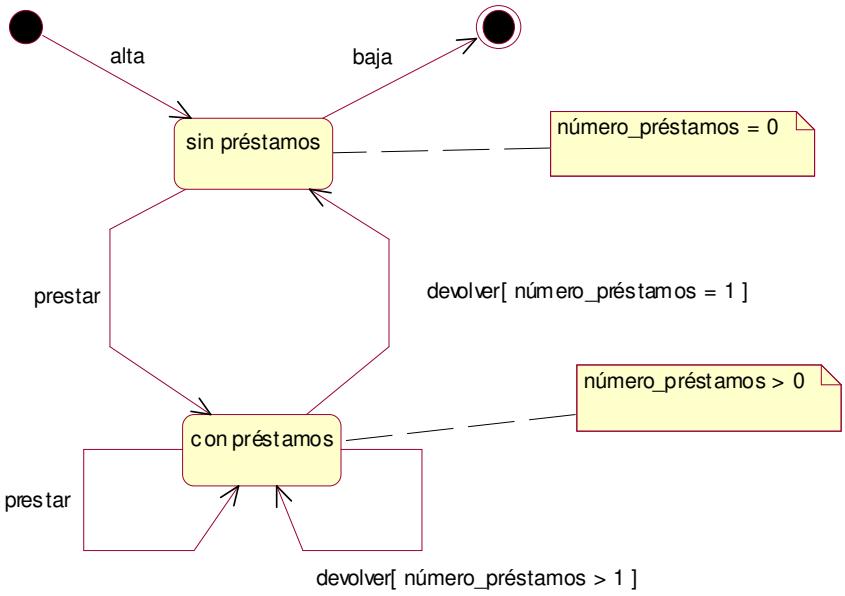


Diagrama de Estados de la Clase Socio



Diagramas de Interacción

Una parte central del modelado del comportamiento de un sistema son los diagramas de interacción.
Los diagramas de interacción son modelos que describen la manera en que colaboran grupos de objetos para cierto comportamiento.
Habitualmente, un diagrama de interacción capta el comportamiento de un solo caso de uso.
Una diferenciación conceptual importante con los Casos de uso, es que aquellos se limitaban al **qué hace** el sistema, mientras que los diagramas de interacción detallan el **cómo lo hace**, de cada Casos de uso. Modelan los aspectos dinámicos del sistema, es decir, los objetos y sus relaciones.

Esto presenta un problema, y es que se pretende representar algo que de por sí tiene movimiento, en un esquema gráfico. A las limitaciones que puede tener, hay que añadirle la complejidad de un sistema, por tanto hay que limitarse a ciertos objetos que nos interesen, y los mensajes enviados entre ellos, es decir, acotarnos a un escenario.

A cada Caso de uso le corresponde un diagrama de interacción. Las variaciones al curso básico de un caso de uso se representan con nuevos diagramas de interacción.

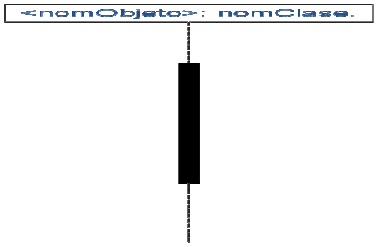
Pertenecen a los diagramas de interacción los Diagramas de secuencia y los Diagramas de colaboración. Un cuadro comparativo facilitará su comprensión.

Diagramas de secuencia	**Diagramas de colaboración** **Diagramas de colaboración**
Destaca la ordenación temporal de los mensajes. Su representación gráfica puede ser demasiado extensa. Los objetos se ubican en el eje horizontal y los mensajes se ordenan temporalmente en sentido vertical.	Destaca la organización estructural de los objetos que envían y reciben mensajes. Su representación gráfica es acotada. Los objetos se grafican como nodos distribuidos aisladamente y los mensajes como arcos entre ellos.

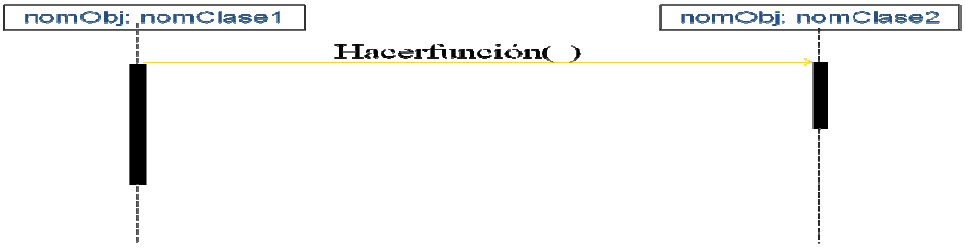
Un diagrama de secuencia se corresponde unívocamente con un diagrama de colaboración. Los programas de Diseño Asistido generan automáticamente el diagrama de colaboración a partir de un diagrama de secuencia.

Diagrama de Secuencia

En un diagrama de secuencia, un objeto está representado por línea vertical punteada y se grafica dentro de un rectángulo en la parte superior del diagrama.



Esta línea vertical se la puede llamar *línea de vida del objeto*. La línea de vida representa la vida del objeto durante la interacción. Esta forma fue popularizada por Ivar Jacobson.
Cada mensaje se representa mediante un segmento dirigido entre las líneas de vida de dos objetos. El orden en el que se dan estos mensajes transcurre de arriba hacia abajo. Cada mensaje es etiquetado por lo menos con el nombre del mensaje; pueden incluirse también, los argumentos y alguna información de control.



También se puede mostrar la autodelegación, que es un mensaje que un objeto se envía a sí mismo, regresando la flecha de mensaje de vuelta a la misma línea de vida.
La información de control tiene dos partes valiosas.

- La primera es la *condición*, que se utiliza para verificar cuándo se envía un mensaje. El mensaje se envía sólo si la condición es verdadera.
- La segunda, es el marcador de iteración, que muestra que un mismo mensaje se envía muchas veces a varios objetos receptores, como sucedería cuando se itera sobre una colección. La base de la iteración se puede mostrar entre corchetes. (como en *[para cada línea de pedido]).

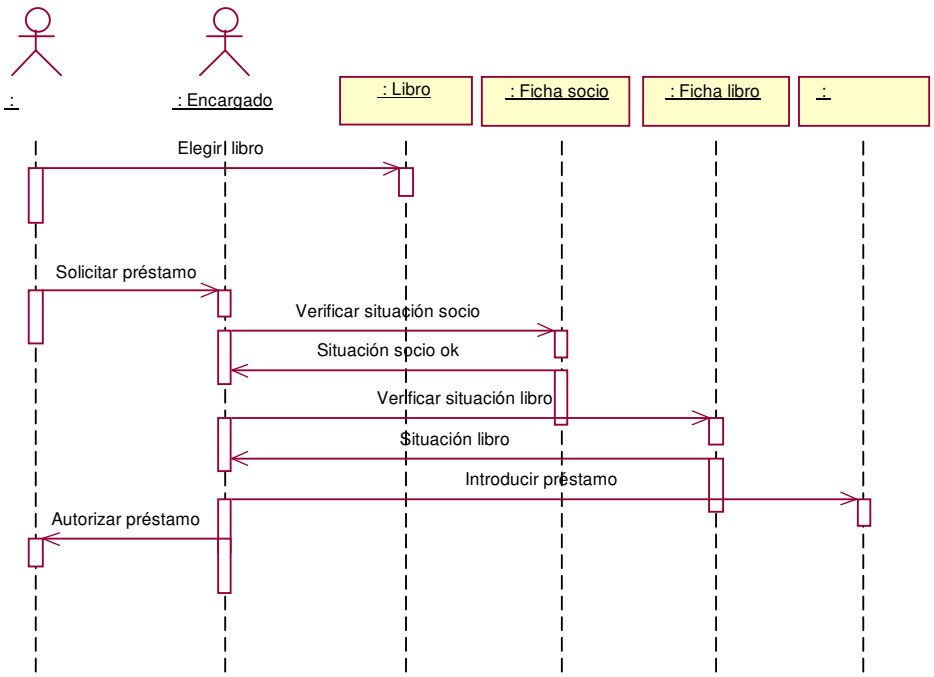
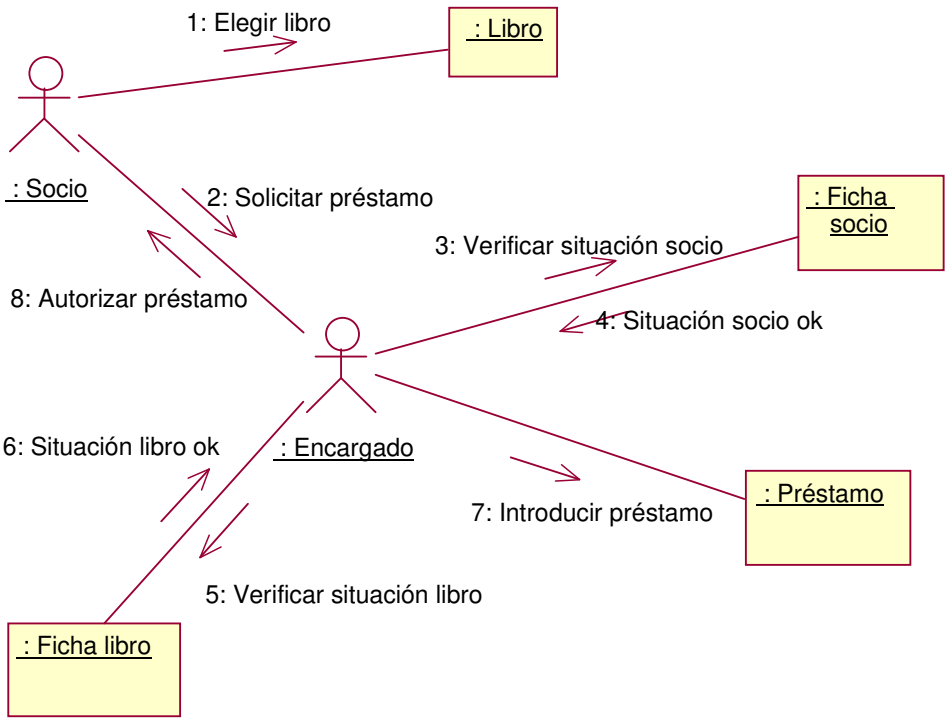


Diagrama de Colaboración

En los diagramas de colaboración, los objetos se muestran como iconos. Las fechas indican, como en los diagramas de secuencia, son enviados los mensajes dentro del caso de uso dado. Sin embargo, en esta ocasión la secuencia se indica numerando los mensajes.



Cuándo utilizar los diagramas de interacción

Se deberán usar los diagramas de interacción cuando se desee ver el comportamiento de varios objetos en un caso de uso. Son buenos para mostrar la colaboración entre los objetos; sin embargo, no sirven tan bien para la definición precisa del comportamiento. Si desea ver el comportamiento de un solo objeto a través de muchos casos de uso, use un diagrama de estado de transición

Ejemplo de Sistema: Compañía de Autobuses

Con este ejemplo vamos a aprender a:

- Crear diagramas de casos de uso, clases, secuencia y colaboración

Como ejemplo se trata de diseñar un modelo de representación de Compañías de Autobuses que sea capaz de gestionar las reservas que los clientes de la compañía deseen realizar sobre los diferentes trayectos que ésta realiza. El modelo debe cumplir los siguientes requisitos:

- Las compañías de autobuses realizan trayectos entre dos ciudades (origen y destino), con un número, una fecha y una hora de salida y otra de llegada previstas.
- Los trayectos se realizan en autobuses que tienen un modelo, una matrícula y la capacidad máxima que admiten.
- Las personas que participan en el modelo tienen todas: nombre, DNI, dirección, nº de teléfono y edad, y serán empleados o pasajeros.
- Los empleados tienen un número de la seguridad social y se conoce su antigüedad en la empresa.
- Los pasajeros por su parte tienen un código de cliente.
- Los autobuses sólo realizan un trayecto y un trayecto sólo lo realiza un autobús.
- Para que un pasajero pueda viajar en un autobús debe haber una reserva sobre un determinado trayecto. Las reservas tienen un precio y un número de asiento en el autobús.

A continuación se pueden observar diferentes diagramas UML representando el sistema.

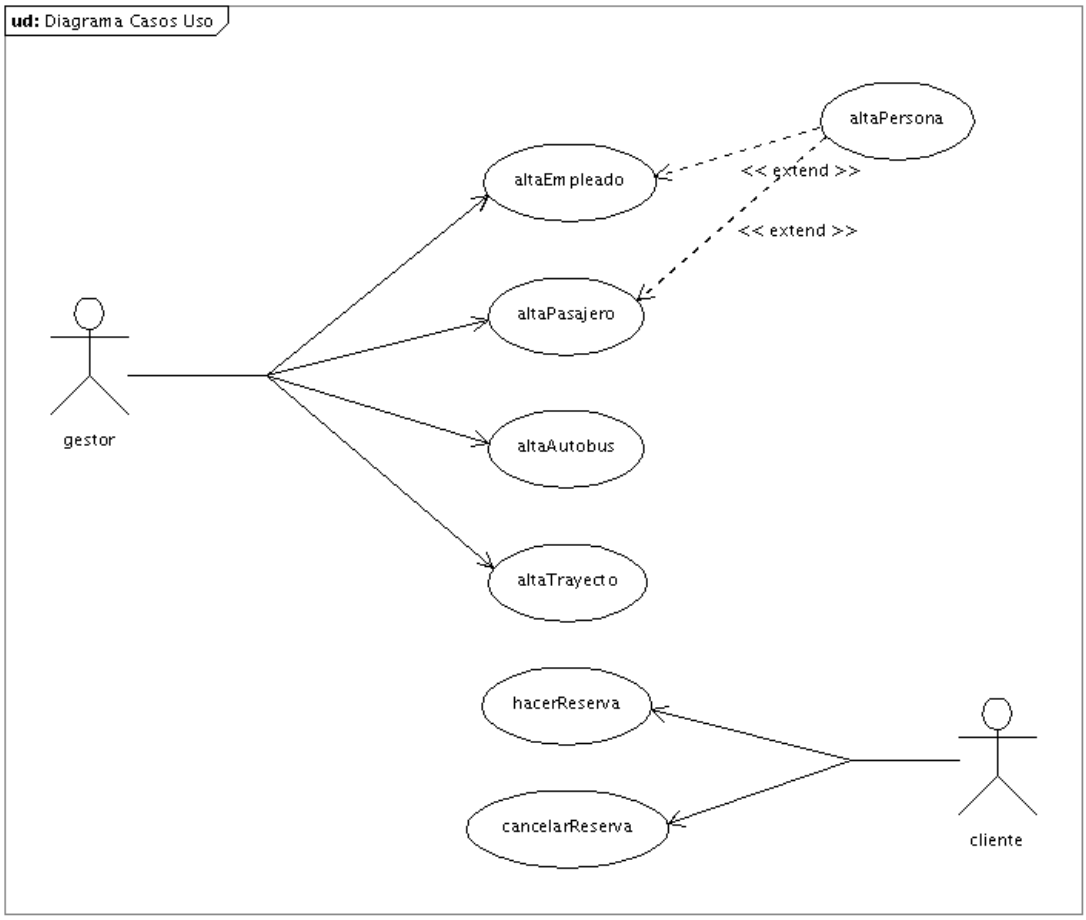


Diagrama de Casos de Uso

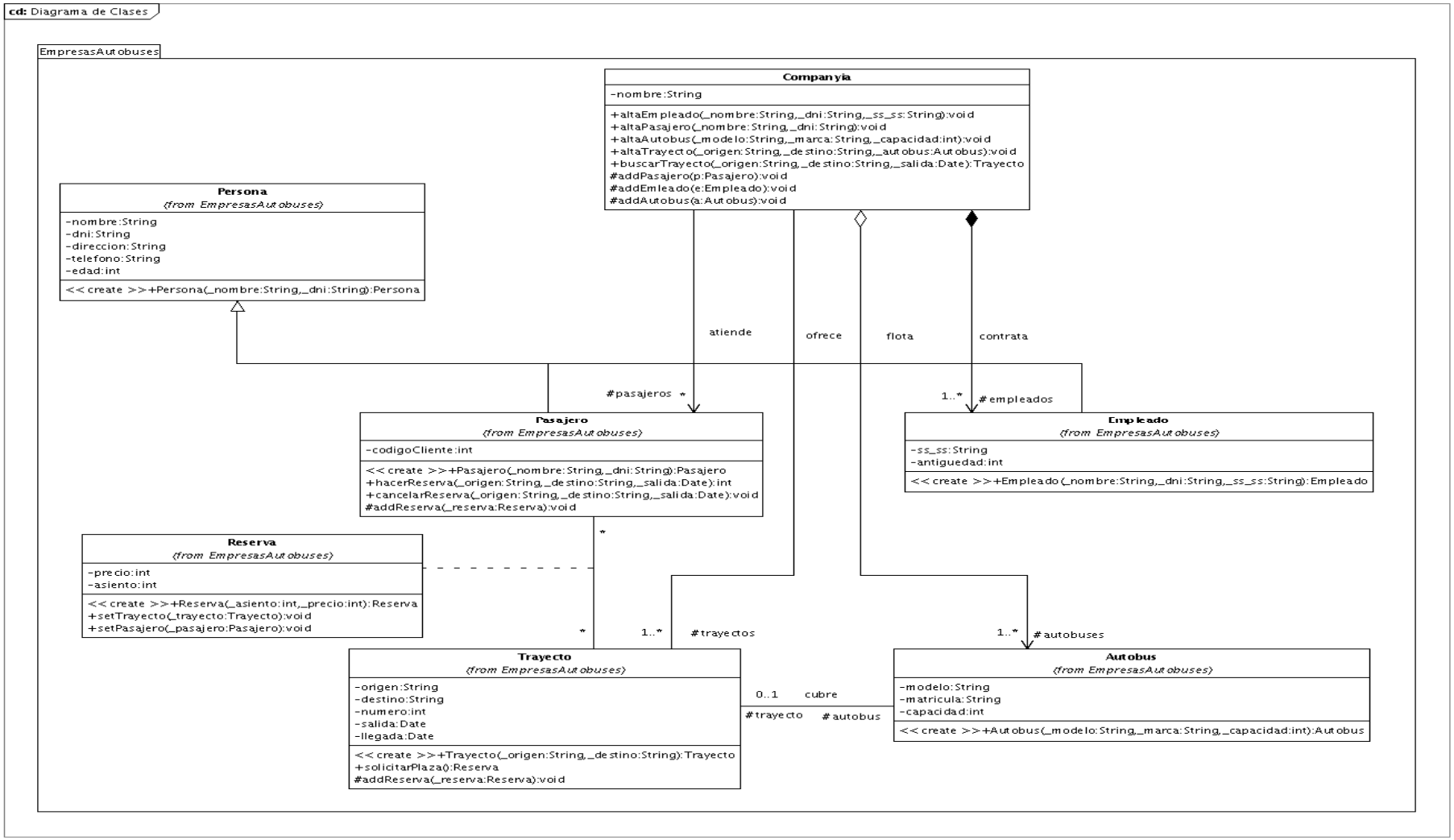


Diagrama de Clases

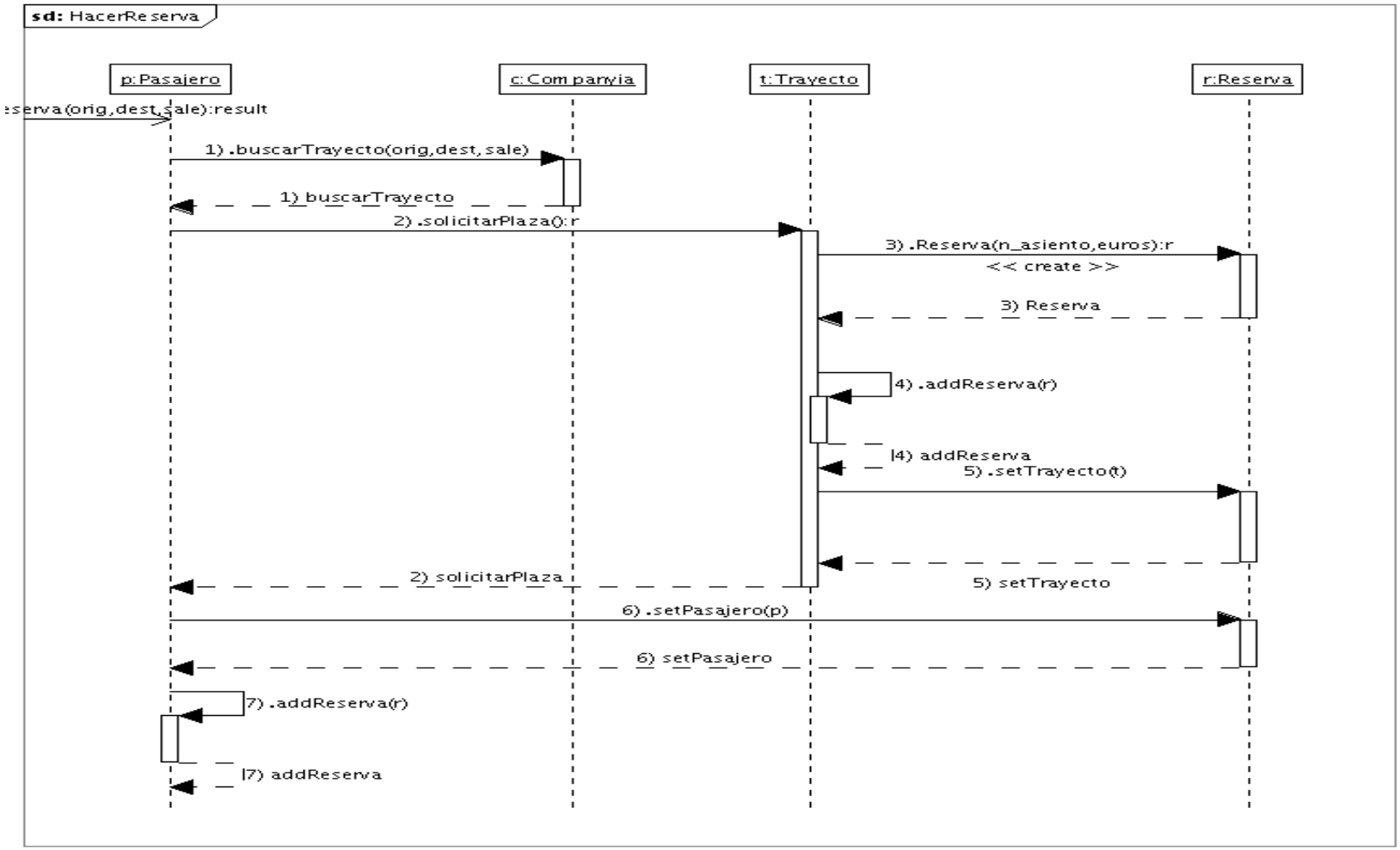


Diagrama de Secuencia

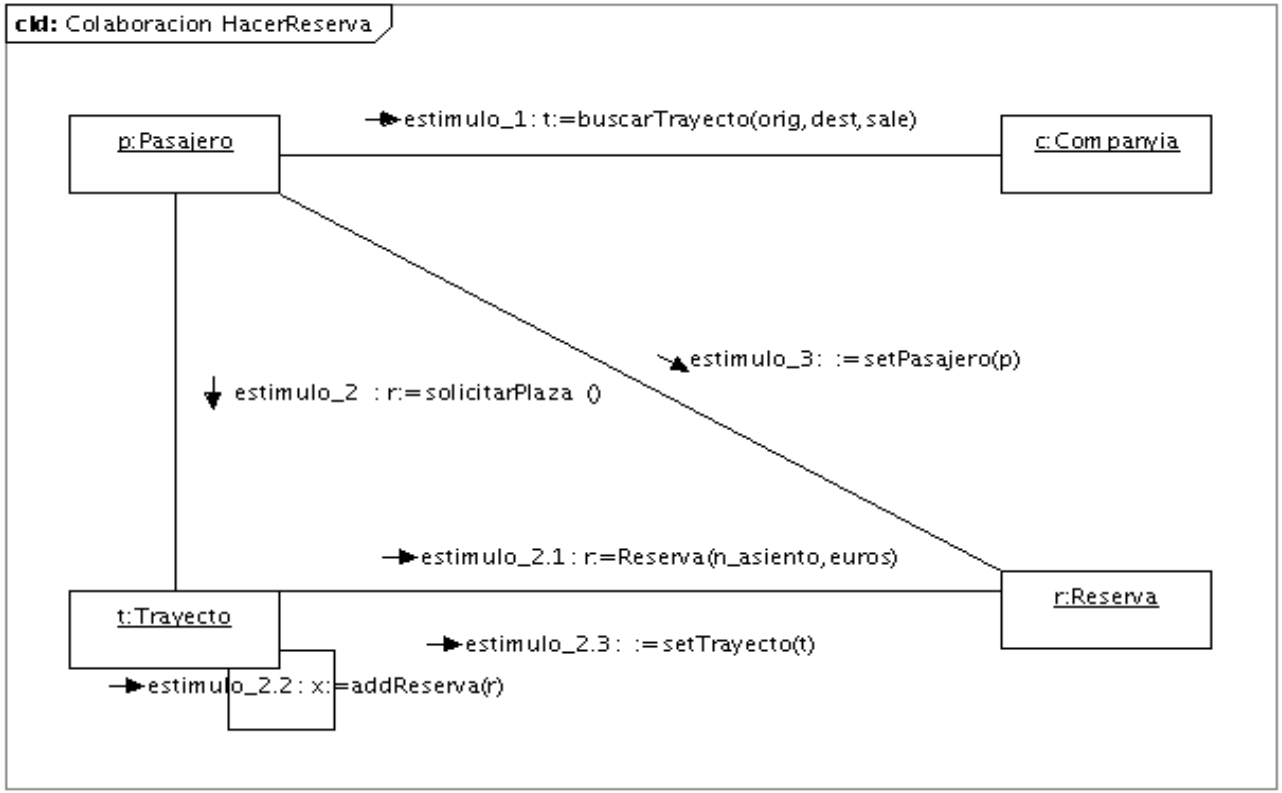


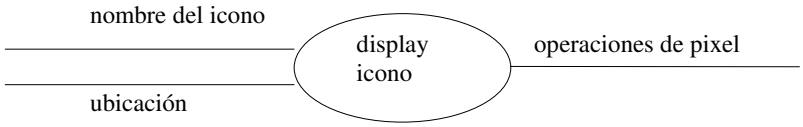
Diagrama de Colaboración

MODELO FUNCIONAL

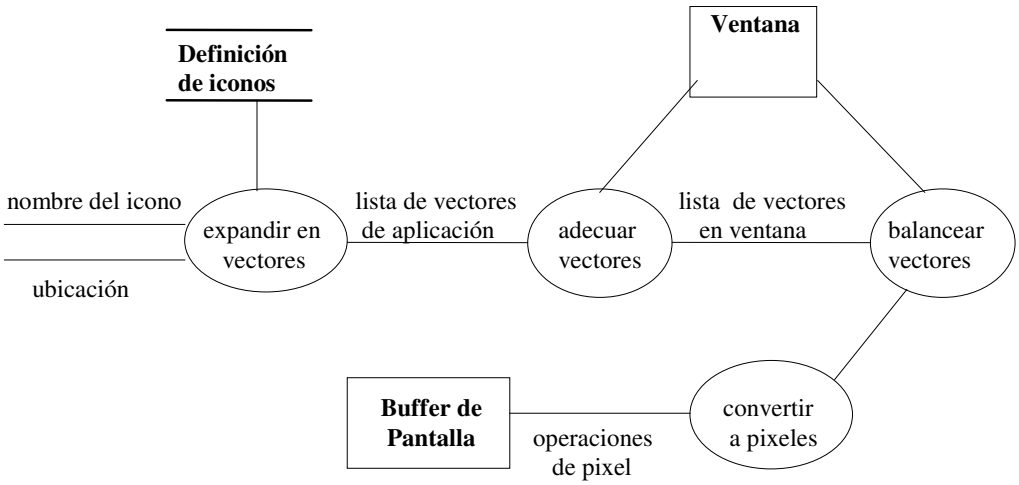
El modelo funcional describe los cálculos dentro de un sistema. El modelo funcional es el tercer pie del trípode del modelado, completando al modelo objeto y al dinámico. El modelo funcional especifica que sucede, el modelo dinámico cuando sucede y el modelo objeto que es lo que va a suceder.
El modelo funcional muestra como los valores de salida en un cálculo son derivados de valores de entrada, sin considerar el orden en los cuales los valores son calculados. El modelo funcional consiste en un múltiple diagrama de flujo de datos el cual se muestra el flujo de valores desde entradas externas a través de operaciones y almacenamientos internos de datos, hacia salidas externas.
Los diagramas de flujo de datos no muestran información de control o de la estructura de los objetos.
El modelo funcional especifica el significado de las operaciones en el modelo objeto y las acciones en el modelo dinámico, así como cualquier restricción en el modelo objeto.
Los programas no interactivos, tales como los compiladores, tienen un modelo dinámico trivial, su propósito es calcular una función. El modelo funcional es el modelo esencial para estos programas, si bien el modelo objeto es importante para todos los problemas con estructuras de datos no triviales. Muchos programas interactivos también tienen un modelo funcional significativo. Por contraste las bases de datos a menudo tienen un modelo funcional trivial dado que su propósito es almacenar y organizar datos, no transformarlos.

DIAGRAMA DE FLUJO DE DATOS

El modelo funcional consiste de múltiples diagramas de flujo de datos los cuales especifican el sentido de las operaciones y las restricciones. Un diagrama de flujo de datos (DFD) muestra la relación funcional de los valores calculados por el sistema, incluyendo los valores de entrada, valores de salida y almacenamiento interno de datos.
Un diagrama de flujo de datos es un gráfico mostrando el flujo de valores de datos desde sus fuentes en los objetos pasando por los *procesos* que los transforman hasta sus destinos en otros objetos. Un diagrama de flujo de datos no muestra información de control, tal como el tiempo en el cual un proceso es ejecutado o las decisiones en medio de caminos alternados de datos; esta información se muestra en el modelo dinámico.
Un DFD no muestra la organización de los valores en los objetos; esta información se ve en el diagrama de objetos.
Un DFD contiene *procesos* que transforman datos, *flujos de datos* que mueven datos, *objetos actores* que producen y consumen datos, y *almacenamientos de datos* que guardan datos en forma pasiva.



En el gráfico anterior se puede observar un diagrama de primer nivel para hacer un display de un icono en un sistema bajo Windows. Este sería el primer nivel de abstracción.



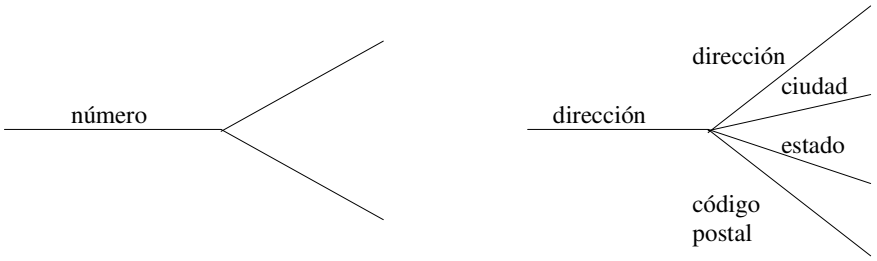
En la figura anterior, muestra el diagrama de flujo completo para hacer el display de un icono en sistema bajo Windows.
Nombre de icono y ubicación son las entradas al diagrama desde una fuente no especificada. El icono es expandido a vectores en el sistema de coordenadas de la aplicación usando una existente definición de icono. Los vectores son adecuados a la medida de la ventana, ellos son balanceados por la ubicación de la ventana en la pantalla, obteniendo vectores en el sistema de coordenadas de pantalla. Finalmente los vectores son convertidos en operaciones de pixel que serán enviadas al buffer de pantalla para ser mostrado. El diagrama de flujo de datos muestra la secuencia de transformaciones experimentadas, así como los valores externos y objetos que afectan al cálculo.

Procesos

Un proceso transforma valores de datos. Los procesos de nivel mas bajo son funciones puras sin efectos colaterales. Funciones típicas incluyen la suma de dos números, etc. Un proceso puede tener efectos colaterales si este contiene componentes no funcionales, tal como, almacenamientos u objetos externos.
El modelo funcional no indica únicamente el posible camino funcional. El resultado de un proceso depende de lo especificado en modelo dinámico como así también del comportamiento del sistema.
Un proceso se representa con una elipse que contiene una descripción de la transformación. Cada proceso tiene un número fijo de flujos de datos de entrada y salida.

Flujos de Datos

Un flujo de datos conecta la salida de un objeto o proceso con la entrada de otro objeto o proceso. Este representa un dato intermedio sin cálculo. El valor no puede ser cambiado por un flujo de datos.
Un flujo de datos se dibuja con una flecha entre el productor y el consumidor del dato. La flecha es rotulada con una descripción del dato, usualmente es el nombre o tipo. El mismo valor puede ser enviado a varios lugares, esto se indica por medio de una bifurcación con muchas flechas emergiendo de el. Las flechas pueden no estar rotuladas, ya que son el mismo valor de entrada.



algunas veces una agregación de datos es dividida en componentes, cada uno de los cuales va a diferentes procesos. Esto se muestra por medio de una bifurcación en el camino en la cual cada una de las flechas es rotulada con el nombre del componente. La combinación de muchos componentes es una agregación de valores, exactamente lo opuesto.
Cada flujo de dato representa un valor en algún punto en el cálculo. El flujo de datos interno de el diagrama que representa valores intermedios dentro de un cálculo no es necesario tenerlo si no tiene significado alguno.
Los flujos en el límite del diagrama son entradas y salidas. Estos flujos pueden estar desconectados, o pueden ser conectados a los objetos.

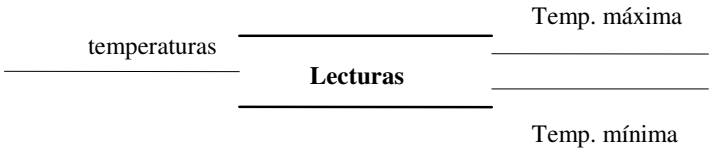
Actores

Un actor es un objeto activo que maneja el gráfico de flujo de datos consumiendo o produciendo valores. Un actor está asociado a las entradas y salidas de un gráfico de flujo de datos. La acción del actor esta fuera del alcance de el diagrama de flujo de datos, pero deben ser parte de el modelo dinámico.
Un actor se dibuja con como un rectángulo que muestran que este es un objeto. Las flechas entre el actor y el diagrama son entradas y salidas del diagrama.

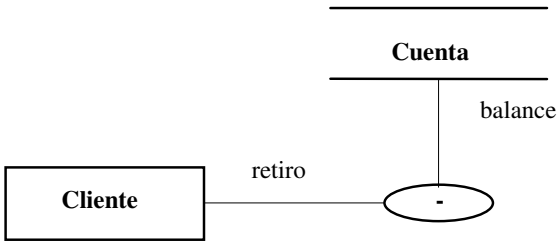
Almacenamiento de Datos

Un almacenamiento de datos es un objeto pasivo dentro de un diagrama de flujo de datos que almacena datos para su acceso posterior. A diferencia de un actor , un almacenamiento no genera ninguna operación en si mismo, sólo responde para almacenar y acceder datos.
Un almacenamiento permite que los valores sean accedidos en diferente orden de los que ellos son generados. Agregaciones de almacenamientos, tal como tablas y listas, proveen acceso a los datos por medio de inserción ordenada o claves de índices.
Un almacenamiento de datos se dibuja con un par de líneas paralelas conteniendo el nombre del almacenamiento. Las flechas de entrada indican información u operaciones que modifican los datos almacenados; esto incluye, agregar elementos, modificar valores o borrar elementos. Las flechas saliendo indican información que sale del almacenamiento. Pudiendo ser esto, sacar un valor entero o algún componente de el. La descripción de quien puede acceder a operaciones permitidas sobre los almacenamientos esta descripta en el modelo objeto.

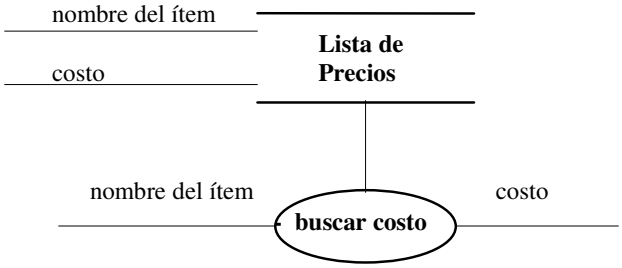
La figura siguiente muestra un almacenamiento de datos para temperaturas leídas. Cada hora se lee una nueva temperatura que se almacena. Al final del día, la temperatura máxima y mínima se leen del almacenamiento. En definitiva un almacenamiento permite ir acumulando datos para ser leídos cuando sea necesario.



esta figura muestra un almacenamiento para cuentas de banco. La doble punta de flecha indica que *balance* es entrada como salida de una operación de substracción. Esto podría haberse dibujado con flechas separadas, pero puede ser así ya que es una operación común.



la siguiente figura muestra una lista de precios para diferentes artículos. La entrada es el nombre del ítem y el costo. La flecha sin rotulo desde el almacenamiento indica que la lista entera de precios es la entrada a la selección de datos.



Ambos, actores y almacenamiento de datos son objetos. Nosotros los distinguimos porque sus comportamientos y uso son generalmente diferentes, sin embargo en un lenguaje orientado a objetos ambos pueden ser implementados como objetos. Un almacenamiento puede ser implementado como un archivo y un actor como un dispositivo externo. Algunos flujos de datos son también objetos, sin embargo en muchos casos son valores puros, tal como, enteros.

Hay una diferencia entre ver un objeto como un simple valor y un almacenamiento que contiene mucho valores. Esto es, veamos en el ejemplo que el nombre de usuario selecciona una cuenta de un banco. El resultado de esta operación es la cuenta en si misma, la cual es usada como un dato a almacenar en la operación update. Un flujo de datos que genera un objeto usado como el destino de otra operación se indica con un triángulo hueco como extremo del flujo. En contraste. La operación update modifica el balance en el objeto cuenta, y se indica como un flujo común. Esto es un concepto nuevo ya que el los flujos de datos tradicionales no era adecuadamente representado el dato cuando es tratado como un objeto.

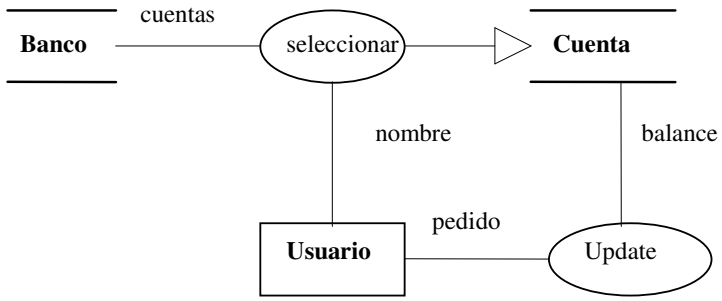
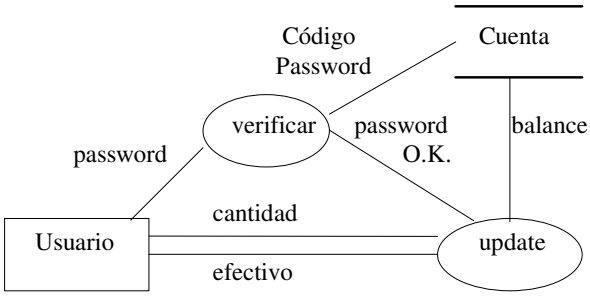


Diagrama de Flujo Anidados

El diagrama de flujo de datos es usado particularmente para mostrar la funcionalidad en el primer nivel de un sistema y luego esto es mostrado en pequeñas unidades funcionales para que sea mas explícito. Cada proceso puede ser explotado en otros diagramas de flujo. Cada entrada y salida de un proceso es una entrada o una salida de un nuevo diagrama. El nuevo diagrama pude tener almacenamientos que no se muestran en el diagrama de nivel superior.

Control de Flujo

un DFD muestra todas los posibles caminos de operación para los valores; pero no muestra en que orden son ejecutados. Las decisiones y secuencias son eventos de control que salen del modelo dinámico.
Un control de flujo es un valor Booleano que afecta si un proceso se evalúa. El control de flujo no es una entrada al proceso. El control de flujo se dibuja con una línea de puntos desde el proceso que produce el valor Booleano al proceso que lo controla.
En la figura vemos un diagrama para retirar dinero de la cuenta de un banco. El retiro es sólo posible si la password es correcta.



ESPECIFICANDO OPERACIONES

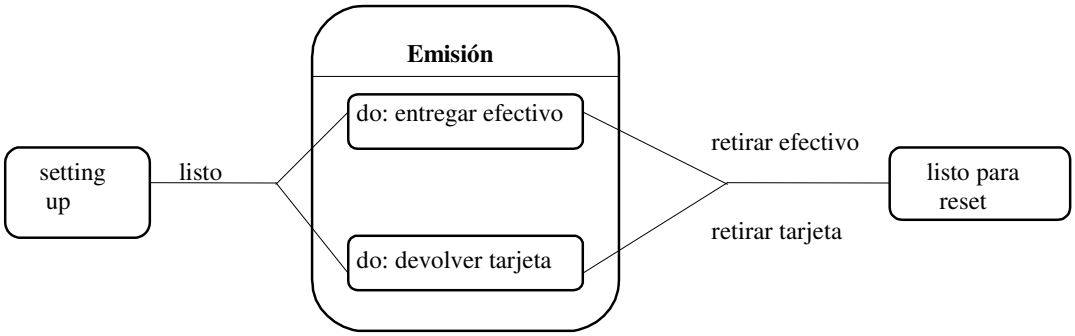
Los procesos en los diagramas de flujo de datos deben ser implementados eventualmente como operaciones sobre objetos.
Cada nivel-inferior, un proceso átomo es una operación. Los proceso en los niveles altos pueden ser considerados operaciones, sin embargo una operación puede ser organizada en forma diferente desde el diagrama de flujo de datos.
Cada operación puede ser especificada en varias formas, incluyendo las siguientes:

- funciones matemáticas, tal como funciones trigonométricas;
- tablas de entrada y salidas de valores (enumeración) para set finitos;
- ecuaciones;
- pre y post condiciones (definiciones axiomáticas);
- tablas de decisión;
- pseuodocódigo;
- lenguaje natural.

La especificación de una operación incluye una firma y una transformación. La firma define la interfase a la operación.
La operación se lista usualmente en el modelo objeto para mostrar el patrón de herencia; la firma de todos los métodos implementados debe coincidir. La transformación define el efecto de la operación.

RELACION DEL MODELO OBJETO Y DINAMICO

El modelo dinámico especifica las secuencias permisibles de cambios para los objetos desde el modelo objeto.
Un diagrama de estado describe todo o parte del comportamiento de un objeto de una clase dada.
Los estados tienen equivalencia con los atributos y los valores de link para el objeto. Los eventos pueden ser representados como las operaciones del modelo objeto.
La estructura del modelo dinámico está relacionada y restringida por la estructura del modelo objeto.
Un subestado refina los atributos y los valores de los vínculos que un objeto puede tener. Cada subestado restringe los valores que cada objeto puede tener. Pero este refinamiento de los valores de un objeto es exactamente generalización por restricción.
Una jerarquía de estados de un objeto es equivalente a una restricción jerárquica de la clase de objetos. Los modelos y lenguajes orientados a objeto, no soportan usualmente, restricciones en la generalización jerárquica, así, el modelo dinámico es el lugar adecuado para representarlo.
Ambas, generalización de estados y clases particionan el set posible de valores de un objeto. Un objeto simple puede tener diferentes estados en el tiempo - el objeto preserva su identidad - pero no puede pertenecer a clases diferentes.
Las diferencias inherentes en medio de los objetos son, por lo tanto, modeladas propiamente como estados diferentes de la misma clase.
Un estado compuesto es la suma de mas de un subestado concurrente. Hay tres fuentes de concurrencia dentro del modelo objeto:
La primera es la agregación de objetos: cada componente de una agregación tiene su estado independiente, también la reunión puede ser considerada para tener un estado que sea la composición de los estados de todas las partes.
La segunda fuente es la agregación dentro de un objeto.
La tercera es el comportamiento concurrente de un objeto.



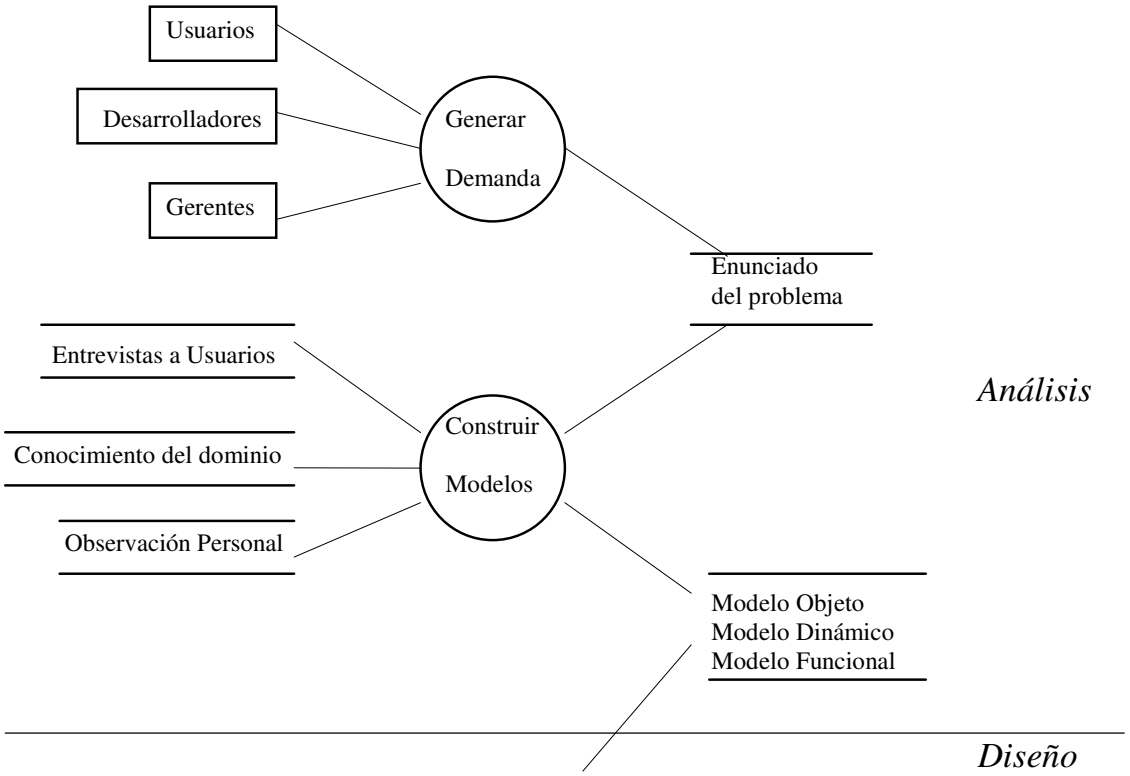
El modelo dinámico de una clase es heredado por las subclases. Las subclases heredan los estados y las transiciones de sus ancestros. Las subclases pueden tener sus propios diagramas de estado.
Pero, ¿como hace el diagrama de estados de la superclase y de la subclase para interactuar?.
Debemos notar que los estados son equivalentes a las restricciones en las clases. Si un diagrama de estados de una superclase y el de una subclase tratan con atributos disjuntos no hay problema.
La subclase tiene un estado compuesto, de diagrama de estados concurrentes. Sin embargo existe un conflicto potencial cuando la subclase y la superclase involucran los mismos atributos.
Un diagrama de estados de una subclase debe ser un refinador del diagrama de una superclase.
La jerarquía de eventos es independiente de la jerarquía de la clase.
Los eventos pueden ser definidos a lo largo de diferentes clases de objetos. Los eventos son mas importantes que los estados y mas parecidos a las clases. Los estados están definidos por medio de la interacción de objetos y eventos. Las transiciones pueden, frecuentemente ser implementadas como operaciones sobre objetos.
El nombre de una operación se corresponde con el nombre del evento.
Los eventos son mas expresivos que las operaciones, porque el efecto de un evento depende no solo de la clase sino también del estado.

ANALISIS

Es el primer paso de la metodología OMT (Object Modeling Technique), está comprometido con divisar un preciso, conciso, entendible, correcto modelo del mundo real. Antes de construir cualquier modelo complejo, el diseñador debe entender los requerimientos y ambiente del mundo real en el cual el sistema existe.
El propósito del análisis orientado a objetos es para modelar el mundo real del sistema de manera que este pueda ser entendido. Para hacer esto, se debe examinar los requerimientos, analizar sus implicancias rigurosamente.
Se deben abstraer las características mas importantes del mundo real y diferir los detalles para después. Se debe hacer un modelo satisfactorio de análisis, sin restringir como se hace. El resultado del análisis debe ser entendiendo el problema como una preparación para el diseño.

OVERVIEW DEL ANALISIS

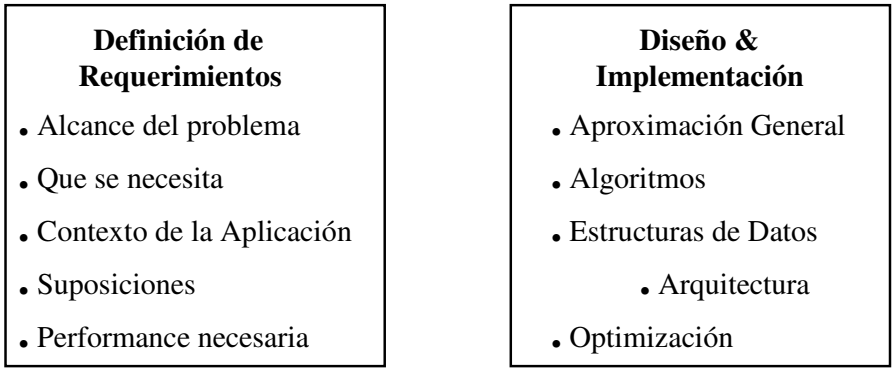
Como se ve en la figura, el análisis comienza con el enunciado de un problema generado por clientes y posiblemente involucre a desarrolladores.
El enunciado puede ser incompleto o informal; el análisis hace que este sea mas preciso y exponga las ambigüedades e inconsistencias. El enunciado de un problema no debe tomarse como inmutable pero debe servir como una base para ir refinando los requerimientos reales



Luego, debe ser entendido el sistema en el mundo real descripto por el enunciado del problema, y abstraer las características esenciales en un modelo.
El enunciado en lenguaje natural esta a menudo con ambigüedades, incompleto e inconsistente.
El modelo de análisis es una precisa, concisa representación del problema que permite responder las preguntas y construir una solución.
Tal vez sea mas importante el proceso de construcción de un riguroso modelo del dominio del problema que fuerce al diseñador a confrontar los mal entendidos lo mas pronto posible en el proceso de desarrollo mientras hay facilidad para llegar a lo correcto.
El análisis no debe ser siempre llevado como una secuencia rígida. Los modelos grandes son construidos iterativamente. Primero se construye un subset, luego se lo extiende hasta completar el problema.
El análisis no es un proceso mecánico, la mayoría de los enunciados de un problema carecen de información esencial, la cual se debe obtener del entrevistado o del conocimiento personal del domino del problema (observación personal).

ENUNCIADO DEL PROBLEMA

El primer paso en cualquier desarrollo es enunciar los requerimientos. Esto es aplicable a la investigación de programas que pueden ser resueltos en forma personal, como así también a aquellos que requieren un gran trabajo en equipo.
Como se ve resume en la figura, la definición del problema debe enunciar que es lo que se quiere hacer y no como se va a hacer. Debe ser un enunciado de necesidades y no u propósito de solución.
El entrevistado debe indicar cuales cosas son esenciales y cuales son opcionales, debe evitar describir sistemas internos pues estos lo que hacen es restar flexibilidad a la implementación.
Las especificaciones de performance y protocolos para la interacción con sistemas externos es una parte legítima de los requerimientos. Los standard de ingeniería de software, tal como, construcción modular, diseño de prototipos de testeo, y previsión para expansiones futuras también son apropiados.



Muchas enunciados de problemas, para particulares, compañías, departamentos del gobierno, se mezclan verdaderos requerimientos con decisiones de diseño. Puede haber algunas veces alguna razón comprensible para requerir algún lenguaje en particular ; pero raramente se justifica el uso en particular de determinado algoritmo.
El analista debe separar los requerimientos verdaderos de las decisiones de diseño o implementación. El analista debe desafiar tales pseudorequerimientos, ya que ellos restringen la flexibilidad.
Hay razones políticas o institucionales para los pseudorequerimientos, pero al menos el analista debe reconocer que esas imposiciones externas no son partes esenciales del dominio del problema.
Una definición de un problema puede tener mas o menos detalles. Un requerimiento para un producto convencional tal como un sistema de facturación, puede tener detalles considerables. Otros sistemas pueden tener baja cantidad de detalles. Pero deben quedar establecidos los objetivos.
Además de que algunos enunciados de problemas son incompletos, etc, hay algunos que sencillamente están mal.
Entonces el propósito de un análisis es obtener un total entendimiento de la problemática. No hay razón para especular que se puede preparar una definición del problema, sin un análisis, será correcto.
El analista debe trabajar con el entrevistado para refinar los requerimientos de manera de representar lo que el desea. Esto involucra seleccionar los requerimientos y sondeando para buscar información que puede perderse.

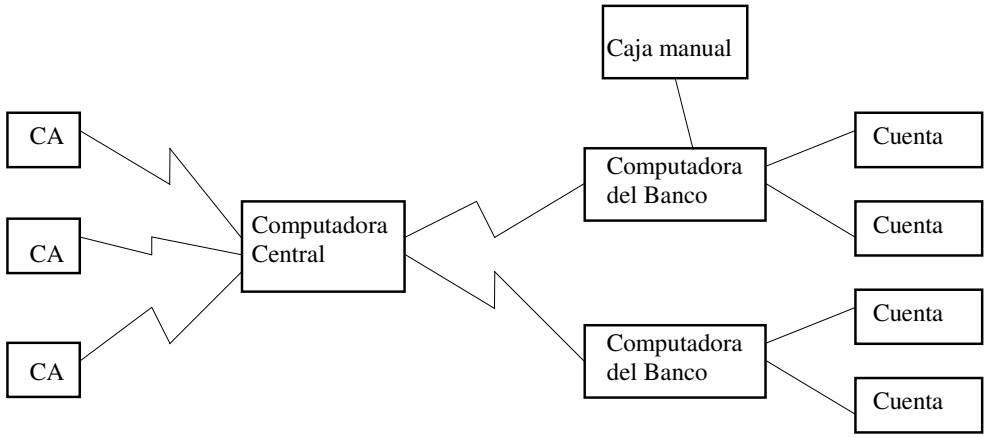
Las consideraciones psicológicas, organizacionales y políticas acá no se mencionan, excepto los siguiente:

" ... si el analista hace exactamente lo que el entrevistado le pide, pero los resultados no son los que el usuario final necesita, probablemente no lo llamen nunca más ...".

EJEMPLO

Cajero Automático

Vamos a desarrollar un ejemplo completo para una red de cajeros automáticos.



Red de Cajeros Automáticos

Enunciado del problema

Diseñar el software para soportar una red de bancos computarizados, que incluye cajas comunes (atendidas por personas) y cajeros Automáticos (CA), la cual va a ser compartida por un consorcio de bancos.
Cada banco provee sus propias computadoras para mantener las cuentas de sus usuarios y procesar las transacciones. Las cajas normales están comunicadas directamente con la computadora propia del banco. Las personas encargadas de la caja (cajeros), ingresan la cuenta y el dato de la transacción. Los cajeros automáticos se comunican con una computadora central la cual liquida la transacción con el banco apropiado. Un CA acepta una tarjeta de crédito, interactúa con el usuario, se comunica con la computadora central para llevar la transacción, entrega el efectivo e imprime un recibo. El sistema requiere un apropiado mantenimiento de registros y provisión de seguridad. El sistema también debe poder manejar accesos concurrentes a la misma cuenta correctamente. Los bancos proveerán su propio software para sus computadoras, acá se debe diseñar el software para la red de CA. El costo del sistema que se comparte, será aportado en forma proporcional por los bancos de acuerdo al número de usuarios con tarjeta que posea cada uno.

Modelando Objetos

El primer paso en el análisis de requerimientos es para construir un modelo objeto. El modelo objeto muestra la estructura estática de datos del mundo real y lo organiza en piezas trabajables.

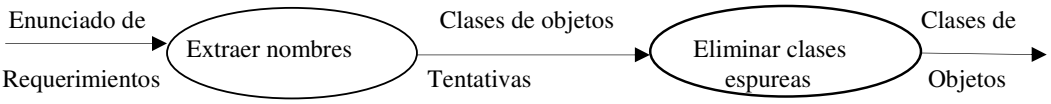
El modelo objeto describe clases de objetos del mundo real y sus relaciones con cada uno de los otros.
Lo mas crucial es la organización del nivel superior del sistema en clases conectadas por medio de asociaciones, las particiones de menor nivel dentro de las clases (generalizaciones) son menos críticas. El modelo objeto precede al modelo dinámico y al funcional porque la estructura estática es usualmente mejor definida, menos dependencia de sobre los detalles de la aplicación, mas estable como la solución, y mas fácil para entender por los humanos.
La información sobre los objetos comienza con el enunciado del problema, los expertos en el conocimiento del domino de la aplicación y los que en general conocen el mundo real del problema. Si el diseñador no es un experto del dominio, la información debe ser obtenida de un experto y chequeada con el modelo repetidamente. El diagrama del modelo objeto promueve la comunicación entre los profesionales del la computación y los expertos en el dominio del sistema.
Identificar primeros las clases y las asociaciones, y como ellas afectan la estructura en total y aproximan al problema. Luego hay que adicionar los atributos además de describir la red básica de clases y asociaciones. La combinación y organización de clases usando herencia. Intentar especificar la herencia directamente sin describir primero las clases de los niveles bajos y sus atributos que a menudo distorsionan la estructura de clases ya que preconciben notaciones.
Luego adicionar las operaciones a las clases como una manera de producir la construcción del modelo dinámico y funcional. Las operaciones modifican objetos y por lo tanto no deben ser explicadas a full si luego la dinámica y funcionalidad se entiende.
Lo mejor es tomar ideas y bajarlas al papel antes de organizar todo completamente, siempre van a haber inconsistencias y redundancias, el modelo correcto se encuentra luego de algunas iteraciones. El análisis y el diseño es raramente lineal.

Los siguientes pasos se ejecutan en la construcción del modelo objeto:

- Identificar objetos y clases.
- Preparar un diccionario de datos.
- Identificar las asociaciones (incluyendo agregaciones) entre objetos.
- Identificar los atributos de los objetos y links.
- Organizar y simplificar clases de objetos usando herencia.
- Verificar que caminos de acceso existe para interrogantes similares.
- Hacer iteraciones para refinar el modelo.
- Agrupar clases en módulos.

Identificando las Clases de Objetos

El primer paso en la construcción de un modelo objeto es la identificación de las clases de objetos relevantes desde el dominio de la aplicación.
Los objetos incluyen entidades físicas, tal como casas, empleados, y máquinas así como, conceptos, tal como, trayectorias, horarios de pago, etc. Todas las clases deben tener sentido en el dominio de la aplicación; evitar construir implementaciones por computadora, tal como subrutinas, etc.
No todas las clases son explícitas en el enunciado del problema, algunas son implícitas en el dominio de la aplicación o en el conocimiento general.



A este nivel no hay que preocuparse por la herencia o clases de nivel superior; primero hay que tomar las clases que aparecen en forma natural aunque subconcientemente se las adapte a una estructura preconcebida.
Por ejemplo, si se está construyendo un sistema de catálogo y control de libros prestados para una librería, primero hay que identificar los diferentes tipos de materiales, tal como, libros, revistas, publicaciones, videos, etc., luego estos se pueden organizar en categorías comparando similaridades y diferencias acerca de las clases básicas.

Enunciado del problema

Diseñar el software para soportar una red de bancos computarizados, que incluye cajas comunes (atendidas por persona) y cajeros Automáticos (CA), la cual va a ser compartida por un consorcio de bancos.
Cada banco provee sus propias computadoras para mantener las cuentas de sus usuarios y procesar las transacciones. Las cajas normales están comunicadas directamente con la computadora propia del banco. Las personas encargadas de la caja (cajeros), ingresan la cuenta y el dato de la transacción. Los cajeros automáticos se comunican con una computadora central la cual liquida la transacción con el banco apropiado. Un CA acepta una tarjeta de crédito, interactúa con el usuario, se comunica con la computadora central para llevar la transacción, entrega el efectivo e imprime un recibo. El sistema requiere un apropiado mantenimiento de registros y provisión de seguridad. El sistema también debe poder manejar accesos concurrentes a la misma cuenta correctamente. Los bancos proveerán su propio software para sus computadoras, acá se debe diseñar el software para la red de CA. El costo del sistema que se comparte será aportado en forma proporcional por los bancos de acuerdo al número de clientes con tarjeta que posea cada uno.

Enunciado del problema

Frases verbales

Diseñar el software para soportar una red de bancos computarizados, que incluye cajas comunes (atendidas por persona) y cajeros Automáticos (CA), la cual va a ser compartida por un consorcio de bancos.
Cada banco provee sus propias computadoras para mantener las cuentas de sus usuarios y procesar las transacciones. Las cajas normales están comunicadas directamente con la computadora propia del banco. Las personas encargadas de la caja (cajeros), ingresan la cuenta y el dato de la transacción. Los cajeros automáticos se comunican con una computadora central la cual liquida la transacción con el banco apropiado. Un CA acepta una tarjeta de crédito, interactúa con el usuario, se comunica con la computadora central para llevar la transacción, entrega el efectivo e imprime un recibo. El sistema requiere un apropiado mantenimiento de registros y provisión de seguridad. El sistema también debe poder manejar accesos concurrentes a la misma cuenta correctamente. Los bancos proveerán su propio software para sus computadoras, acá se debe diseñar el software para la red de CA. El costo del sistema que se comparte será aportado en forma proporcional por los bancos de acuerdo al número de usuarios con tarjeta que posea cada uno.

Ejemplo de Análisis Orientado a Objetos
Requisitos

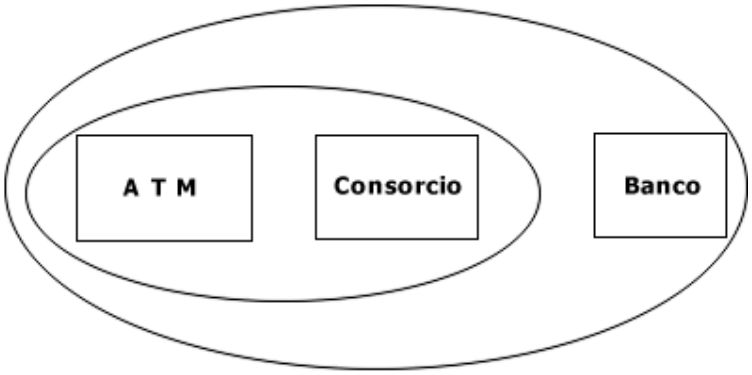
Explican lo que se desea que haga el sistema, ya sea en lenguaje natural, o en forma de casos de uso a la Jacobson.

Ejemplo:

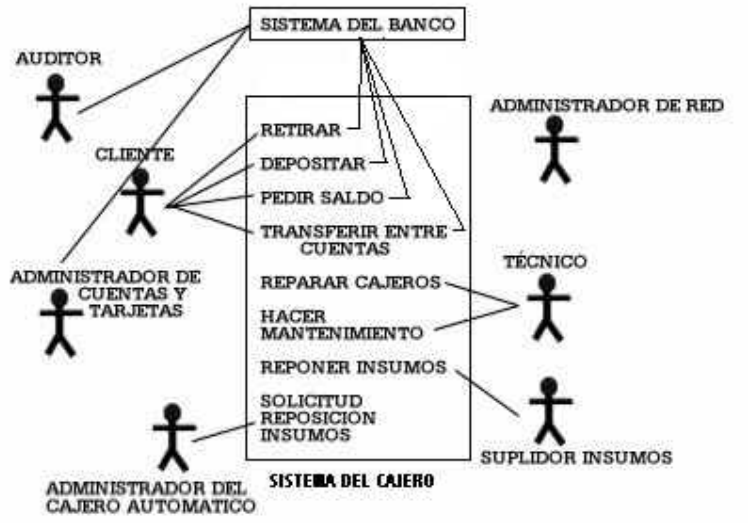
Se desea diseñar el software necesario para una red bancaria provista de cajeros automáticos (ATM, *automatic teller machines*), que serán compartidos por un consorcio de bancos. Cada banco dispone de su propia computadora, provisto de software propio, que lleva la información sobre sus cuentas y procesa las transacciones que actúan sobre dichas cuentas. A esta computadora están conectadas las estaciones de cajero, que son propiedad del banco y en las que operan cajeros humanos, que pueden crear cuentas e introducir transacciones sobre ellas.

Los cajeros automáticos aceptan tarjetas de crédito, interaccionan con el usuario, se comunican con un computadora central para llevar a cabo las transacciones, entregan dinero en efectivo al usuario e imprimen recibos. El sistema llevará correctamente el registro de las transacciones efectuadas, cumplirá características aceptables de seguridad y manejará correctamente accesos concurrentes a la misma cuenta.

El costo de desarrollo de la parte compartida del sistema se dividirá entre los bancos que forman parte del consorcio en función del número de clientes provistos de tarjetas de crédito.



Expresar los requisitos como Casos de Uso



Preparar escenarios detallados. Primero los normales. Después se añaden los problemas que pueden surgir. En el ejemplo de los cajeros automáticos:

- Escenario normal:
  - El cajero automático pide al cliente que inserte la tarjeta de crédito.
  - El cliente inserta la tarjeta de crédito.
  - El cajero automático acepta la tarjeta de crédito y lee el número de tarjeta y el código del banco.
  - El cajero automático pide la contraseña al cliente.
  - El cliente teclea "1234".
  - El cajero automático envía el número de tarjeta, el código del banco y la contraseña al consorcio.
  - El consorcio envía el número de tarjeta y la contraseña al banco.
  - El banco notifica la aceptación al consorcio.
  - El consorcio notifica la aceptación al cajero automático.
  - El cajero automático pide al cliente que elija el tipo de transacción: retirada de fondos, depósito, transferencia, información.
  - El cliente selecciona retirada de fondos.
  - El cajero automático pide al cliente que teclee la cantidad.
  - El cliente teclea 250,00.
  - El cajero automático comprueba que la cantidad está dentro de los límites generales.
  - El cajero automático genera una transacción y la envía al consorcio.
  - El consorcio pasa la transacción al banco.
  - El banco aprueba la transacción.
  - El banco actualiza la cuenta.
  - El banco envía al consorcio la notificación de aceptación y el nuevo saldo de la cuenta.
  - El consorcio envía al cajero automático la notificación de aceptación y el nuevo saldo de la cuenta.
  - El cajero automático entrega el dinero al cliente.
  - El cliente toma el dinero.
  - El cajero automático pregunta al cliente si quiere un recibo.
  - El cliente contesta SI.
  - El cajero automático imprime un recibo y pide al cliente que lo tome.
  - El cliente toma el recibo.
  - El cajero automático pregunta al cliente si quiere hacer otra operación.
  - El cliente contesta NO.

- El cajero automático expulsa la tarjeta de crédito e indica al cliente que la tome.
- El cliente toma la tarjeta de crédito.
- El cajero automático vuelve a la situación inicial.

- Escenario con problemas:
  - El cajero automático pide al cliente que inserte la tarjeta de crédito.
  - El cliente inserta la tarjeta de crédito.
  - El cajero automático acepta la tarjeta de crédito y lee el número de tarjeta y el código del banco.
  - El cajero automático pide la contraseña al cliente.
  - El cliente teclea "9999".
  - El cajero automático envía el número de tarjeta, el código del banco y la contraseña al consorcio.
  - El consorcio envía el número de tarjeta y la contraseña al banco.
  - El banco notifica el rechazo al consorcio.
  - El consorcio notifica el rechazo al cajero automático.
  - El cajero automático notifica el rechazo al cliente y pide que teclee de nuevo la contraseña.
  - El cliente teclea "1234".
  - El cajero automático envía el número de tarjeta, el código del banco y la contraseña al consorcio.
  - El consorcio envía el número de tarjeta y la contraseña al banco.
  - El banco notifica la aceptación al consorcio.
  - El consorcio notifica la aceptación al cajero automático.
  - El cajero automático pide al cliente que elija el tipo de transacción: retirada de fondos, depósito, transferencia, información.
  - El cliente selecciona retirada de fondos.
  - El cajero automático pide al cliente que teclee la cantidad.
  - El cliente teclea CANCELAR.
  - El cajero automático expulsa la tarjeta de crédito e indica al cliente que la tome.
  - El cliente toma la tarjeta de crédito.
  - El cajero automático vuelve a la situación inicial.

Modelo de objetos

Consta de los siguientes pasos:

- Identificar objetos y clases
- Identificar y depurar relaciones
- Identificar atributos de objetos y relaciones
- Añadir herencia

- Comprobar los casos de uso (iterar)
- Modularizar
- Añadir y simplificar métodos

Identificar objetos y clases

Consta de los siguientes pasos:

- Seleccionar nombres en los requisitos
- Añadir clases adicionales procedentes de nuestro conocimiento del tema
- Eliminar redundancias
- Eliminar clases irrelevantes
- Eliminar clases vagas
- Separar atributos
- Separar métodos
- Eliminar objetos de diseño

Resultado: Preparar diccionario de clases

En el ejemplo de los cajeros automáticos:

- Seleccionar nombres en los requisitos

  Los nombres extraídos de los requisitos en lenguaje natural del sistema de cajeros automáticos son los siguientes (23):

  Software, Red bancaria, Cajero automático, Consorcio de bancos, Banco, Computadora del banco, Cuenta bancaria, Información sobre la cuenta, Transacción, Estaciones de cajero, Cajero humano, Tarjeta de crédito, Usuario, Computadora central, Dinero en efectivo, Recibo, Sistema, Registro de transacciones, Características de seguridad, Acceso a la cuenta, Costo de desarrollo, Parte compartida, Cliente.

- Añadir clases adicionales procedentes de nuestro conocimiento del tema

  Podemos añadir la clase *Línea de comunicaciones*.

- Eliminar redundancias

  Cliente y *Usuario* son la misma clase. Nos quedamos con *Cliente* por adaptarse mejor al concepto.

- Eliminar clases irrelevantes

  Costo de desarrollo no tiene nada que ver con el problema, queda fuera del sistema.

- Eliminar clases vagas

  Sistema, *Características de seguridad*, *Red bancaria* y *Parte compartida* pueden considerarse vagas.

- Separar atributos

Los atributos definen datos asociados a un objeto, en lugar de objetos. Aunque la separación no es clara (los atributos pueden ser objetos embebidos) en algunos casos se pueden distinguir. En el ejemplo, pueden considerarse atributos *Información sobre la cuenta*, (atributo de *Cuenta bancaria*), *Dinero en efectivo* y *Recibo* (atributos de *Cajero automático*).

- Separar métodos

Algunos nombres (por ejemplo, *Llamada telefónica*) definen realmente operaciones o eventos.

- Eliminar objetos de diseño

Todas las clases que corresponden más a la solución del problema que a la situación real, deben considerarse objetos de diseño y eliminarse en la fase del análisis. En el ejemplo, eliminaremos *Registro de transacciones*, *Línea de comunicaciones*, *Acceso a la cuenta* y *Software*.

Resultado. Del análisis anterior, resultan seleccionadas las siguientes clases (11):

Cajero automático, Consorcio de bancos, Banco, Computadora del banco, Cuenta bancaria, Transacción, Estaciones de cajero, Cajero humano, Tarjeta de crédito, Computadora central, Cliente.

El diccionario de clases contiene la definición detallada de todas estas clases en lenguaje natural. Ejemplo:

- *Cajero automático*: Terminal remoto que permite a los clientes realizar transacciones utilizando tarjetas de crédito para identificarse. El cajero automático interacciona con el cliente para identificar la transacción deseada y sus datos asociados, envía esta información a la computadora central para su validación y proceso, y entrega al usuario dinero en efectivo y un recibo. Suponemos que el cajero automático no opera cuando está desconectado de la red.

- *Consorcio de bancos*: Conjunto organizado de bancos que lleva la gestión de los cajeros automáticos. Suponemos que sólo se gestionan transacciones para los bancos que pertenecen al consorcio.

- *Banco*: Institución financiera que maneja las cuentas bancarias de sus clientes y emite tarjetas de crédito que facilitan el acceso a dichas cuentas a través de la red de cajeros automáticos.

Identificar y depurar relaciones

Consta de los siguientes pasos:

- Seleccionar verbos relacionales en los requisitos
- Añadir relaciones adicionales procedentes de nuestro conocimiento del tema
- Eliminar relaciones de diseño o entre clases eliminadas
- Eliminar eventos transitorios
- Reducir relaciones ternarias
- Eliminar relaciones redundantes o derivadas
- Añadir relaciones olvidadas
- Definir la multiplicidad de cada relación

En el ejemplo de los cajeros automáticos:

- Seleccionar verbos relacionales en los requisitos

1. Una *Red bancaria* está provista de *Cajeros automáticos*.
2. El *Consorcio de bancos* comparte los *Cajeros automáticos*.
3. Cada *Banco* dispone de un *Computadora del banco*.
4. La *Computadora del banco* dispone de *Software*.
5. La *Computadora del banco* lleva la información sobre las *Cuentas bancarias*.
6. El *Computadora del banco* procesa *Transacciones*.
7. Una *Transacción* actúa sobre una *Cuenta bancaria*.
8. Las *Estaciones de cajero* están conectadas a la *Computadora del banco*.
9. Las *Estaciones de cajero* son propiedad del *Banco*.
10. El *Cajero humano* opera en la *Estación de cajero*.
11. El *Cajero humano* crea *Cuentas bancarias*.
12. El *Cajero humano* introduce *Transacciones* sobre las *Cuentas bancarias*.
13. Los *Cajeros automáticos* aceptan *Tarjetas de crédito*.
14. Los *Cajeros automáticos* interaccionan con el *Usuario*.
15. Los *Cajeros automáticos* comunican con la *Computadora central*.
16. La *Computadora central* lleva a cabo las *Transacciones*.
17. Los *Cajeros automáticos* entregan *Dinero en efectivo* al *Usuario*.
18. Los *Cajeros automáticos* imprimen *Recibos*.
19. El *Sistema* lleva el *Registro de las transacciones*.
20. El *Sistema* cumple *Características de seguridad*.
21. El *Sistema* maneja *Accesos* concurrentes a la *Cuenta bancaria*.
22. El *Costo de desarrollo* se divide entre los *Bancos*.
23. Los *Bancos* forman parte del *Consorcio*.
24. Los *Clientes* están provistos de *Tarjetas de crédito*.
25. Relaciones adicionales implícitas en el texto:
26. Las *Cuentas bancarias* están en los *Bancos*.
27. La *Computadora central* pertenece al *Consorcio*.
28. Los *Bancos* tienen *Clientes*.

- Añadir relaciones adicionales procedentes de nuestro conocimiento del tema

29. Las *Tarjetas de crédito* están asociadas a las *Cuentas bancarias* .

30. Los *Cajeros humanos* son empleados de los *Bancos*.

•Eliminar relaciones de diseño o entre clases eliminadas

Eliminamos las relaciones números 1, 4, 17, 18, 19, 20, 21, 22.

•Eliminar eventos transitorios

Son sucesos que pertenecen al modelo dinámico y no constituyen relaciones estructurales (estáticas) entre los objetos.

Eliminamos las relaciones números 13 y 14. Otras veces conviene reformularlas, como en el caso de la número 16, la *Computadora central* lleva a cabo las *Transacciones*, que debería sustituirse por:

16 a. La *Computadora central* se comunica con el *Banco*.

•Reducir relaciones ternarias

Son relaciones entre tres o más clases. Muchas veces es posible descomponerlas en varias relaciones binarias (entre dos clases). Por ejemplo, la relación número 12 (El *Cajero humano* introduce *Transacciones* sobre las *Cuentas bancarias*) puede descomponerse en:

- 12a. El *Cajero humano* introduce *Transacciones*
- 12b. Las *Transacciones* actúan sobre las *Cuentas bancarias*.

De igual modo, la número 17 puede descomponerse así:

- 17a. Los *Cajeros automáticos* entregan *Dinero en efectivo*.
- 17b. El *Usuario* recoge el *Dinero en efectivo*.

•Eliminar relaciones redundantes o derivadas

Por ejemplo, la relación número 2 es una combinación de las relaciones número 15 y 26. Hay que tener cuidado, sin embargo, de no eliminar relaciones aparentemente redundantes, pero que en realidad son necesarias (por ejemplo, si la multiplicidad es distinta).

•Añadir relaciones olvidadas

Por ejemplo:

- 31. Los *Clientes* tienen *Cuentas*.
- 32. Las *Transacciones* son autorizadas por la *Tarjeta de crédito*.
- 33. Las *Transacciones* pueden introducirse en una *Estación de cajero*.

•Definir la multiplicidad de cada asociación

- Un *Banco* puede contener muchas *Cuentas*.
- Un *Cliente* puede tener muchas *Cuentas*.
- Un *Cliente* puede tener muchas *Tarjetas de crédito*.
- Un *Banco* emplea muchos *Cajeros*.

- Un *Banco* tiene una sola *Computadora del banco*.
- La *Computadora central* se comunica con muchos *Computadoraes del banco*.
- Etc.

El resultado de estas operaciones es un esqueleto del modelo de clases sin herencia.

Identificar atributos de objetos y relaciones

Consta de los siguientes pasos:

- Distinguir los objetos de los atributos
- Distinguir entre los atributos de objetos y de relaciones
- El identificador del objeto es siempre un atributo implícito
- Eliminar atributos privados (de diseño)
- Eliminar atributos de detalle fino
- Localizar atributos discordantes (dividir la clase)

En el ejemplo de los cajeros automáticos:

- Atributos de los objetos
  - Del *Banco*: Nombre.
  - De la *Cuenta*: Saldo, Límite de crédito, Tipo de cuenta.
  - Del *Cliente*: Nombre, Dirección.
  - Del *Cajero*: Nombre.
  - De una *Transacción del cajero*: Tipo, Fecha y hora, Cantidad.
  - Del *Cajero automático*: Efectivo disponible, Cantidad entregada.
  - De una *Transacción remota*: Tipo, Fecha y hora, Cantidad.
  - De la *Tarjeta de crédito*: Clave, Código del banco, Código de la tarjeta.
- Atributos de las relaciones
  - 8 y 9: Código de la estación de cajero.
  - 15: Código del cajero automático.
  - 16a: Código del banco.
  - 23: Código del banco.
  - 25: Código de la cuenta.
  - 29: Código de empleado.

Añadir herencia

Introducimos clases nuevas (virtuales) que contienen información común a dos o más clases preexistentes. Procurar evitar la herencia múltiple, a menos que sea estrictamente necesaria.

Resultado: Primer diagrama de clases

En el ejemplo de los cajeros automáticos:

- La clase *Estación de entrada* será superclase de *Cajero automático* y de *Estación de cajero*.
- La clase *Transacción* será superclase de *Transacción de cajero* y de *Transacción remota*.
- Podrían refinarse los tipos de cuentas.

Comprobar los casos de uso (iterar)

Para localizar fallos que deben corregirse fijarse en:

- Asimetrías en las relaciones: añadir clases nuevas para equilibrarlas.
- Atributos muy dispares: descomponer una clase en dos.
- Dificultades en la formación de superclases: descomponer una clase en dos. Una de sus partes puede ajustar mejor.
- Operaciones sin objetivo: añadir clase.
- Relaciones duplicadas: crear superclase.
- Conversión de relaciones en clases: por ejemplo, clase *Empleado*.
- Operaciones que no encuentran camino para realizarse: añadir relaciones.
- Relaciones redundantes: eliminarlas.
- Relaciones demasiado detalladas o demasiado vagas: subirlas a una superclase o bajarlas a una subclase.
- Clases sin atributos, sin métodos o sin relaciones: eliminarlas.
- Relaciones que nadie atraviesa: eliminarlas.
- Atributos de clase necesarios en un acceso: pasarlos a atributos de relación.

En el ejemplo de los cajeros automáticos:

- *Tarjeta de crédito* desempeña dos roles: la tarjeta física, que se introduce y que permite al cajero automático conectarse con el banco, con información sobre el mundo real (banco, número de la tarjeta) y las autorizaciones concedidas por éste, que sólo son números en la memoria de un computadora y se pueden cambiar con facilidad (contraseña, límite de crédito). Se puede descomponer en *Tarjeta de crédito* y Autorización de la tarjeta. Una sola autorización puede afectar a más de una tarjeta física. Una misma autorización puede permitir acceder a más de una cuenta (y viceversa).
- Introducimos la clase *Actualización de cuenta* para refinar el concepto de *Transacción*. Una misma transacción puede estar compuesta de varias actualizaciones de cuenta (por ejemplo, transferencia entre cuentas son dos actualizaciones).
- No hay distinción significativa entre *Banco* y *Computadora del banco*, por una parte, y entre *Consorcio* y *Computadora central*, por otra. Fusionamos esas clases.

Modularizar

Agrupar clases en módulos.

En el ejemplo de los cajeros automáticos. Posibles módulos:

- Cajeros en general: *Cajero*, *Estación de cajero*, *Cajero automático*, *Estación de entrada*.
- Cuentas en general: *Cuenta*, *Tarjeta de crédito*, *Autorización*, *Cliente*, *Transacción*, *Transacción de cajero*, *Transacción remota*.

- Bancos: *Banco*, *Consorcio*.

Añadir y simplificar métodos

- Todos los atributos se suponen accesibles.
- Añadir métodos que permitan navegar de un objeto a otro.

Modelo dinámico

Consta de los siguientes pasos:

- Identificar sucesos
- Construir diagramas de estados
- Comprobar consistencia (iterar)
- Añadir métodos

Identificar sucesos

Los sucesos se extraen de los casos de uso (escenarios). Pueden ser de los siguientes tipos:

- Señales
- Entradas
- Decisiones
- Interrupciones
- Transiciones
- Acciones externas
- Condiciones de error

Resultados: Diagramas de secuencia (trazas de eventos) y diagramas de colaboración (diagramas de flujo de eventos).

Los casos de uso (escenarios) se convierten en diagramas de secuencia. Estas se compactan en diagramas de colaboración.

En el ejemplo de los cajeros automáticos:

El cliente introduce la contraseña define un evento de entrada que el objeto *Cliente* envía al objeto *Cajero automático*. *El cajero automático entrega el dinero al cliente* es un evento que el objeto *Cajero automático* envía al objeto *Cliente*.

Agrupar los eventos equivalentes: *El cliente introduce la contraseña* es el mismo evento independientemente de la contraseña introducida. *El cajero automático entrega el dinero al cliente* es el mismo evento independientemente de la cantidad entregada.

No agrupar los eventos no equivalentes: *El banco autoriza la transacción* es distinto evento que El banco rechaza la transacción.

Construir diagramas de estados

Uno por clase.

En el ejemplo de los cajeros automáticos centrarse en las clases dinámicas, que cambian de estado:

- Cajero automático
- Banco
- Consorcio

- Estación de cajero

No hace falta construir diagramas de estado de las clases pasivas, que no cambian de estado de modo significativo:

- Tarjeta de crédito
- Transacción
- Cuenta

Tampoco hace falta considerar a fondo los objetos externos, que no forman parte del sistema informático:

- Cliente
- Cajero humano

Añadir métodos

Los eventos son métodos. Es preciso decidir de qué clase de objetos.

Las acciones y actividades realizadas en los estados son métodos.

Modelo funcional

Consta de los siguientes pasos:

- Identificar valores de entrada/salida
- Construir diagramas de flujo de actividad
- Describir funciones
- Identificar restricciones y dependencias funcionales entre objetos
- Definir criterios de optimización (iterar)
- Añadir métodos

Identificar valores de entrada/salida

Son los que pasan información desde los objetos externos al sistema de software propiamente dicho.

En el ejemplo de los cajeros automáticos son objetos externos:

- Cliente
- Tarjeta de crédito
- Cajero humano

Los valores de entrada/salida serán:

- Del cliente al cajero automático: contraseña, tipo de transacción, tipo de cuenta, cantidad solicitada.
- De la tarjeta de crédito al cajero automático: código del banco, código de la tarjeta.
- Del cajero automático al cliente: dinero en efectivo, recibo, mensajes.

Construir diagramas de flujo de actividad

Relacionan los valores de entrada con los de salida. Suele dividirse en varias capas o niveles.

En el ejemplo de los cajeros automáticos:

- Nivel superior: relaciona el cliente, la tarjeta de crédito y la cuenta.
- Nivel intermedio: expande la operación "realizar transacción", incluida en el nivel superior.

Describir funciones

Descripción de cada una de las funciones de nivel mínimo que aparecen en los diagramas de flujo de actividad. La descripción puede ser:

- En lenguaje natural
- Un modelo matemático
- Pseudocódigo
- Tablas de decisión
- Etc.

En el ejemplo de los cajeros automáticos:

Descripción de la función "actualizar cuenta":

```
actualizar cuenta (cuenta, cantidad, tipo de transacción)
-> efectivo, recibo, mensaje
  Si es una retirada de efectivo:
    Si la cantidad a retirar excede el saldo,
      rechazar la transacción y no entregar dinero
    En caso contrario:
      Restar la cantidad del saldo y entregar dinero
  Si es un depósito:
    Aumentar el saldo de la cuenta y no entragar dinero
  Si es una petición de información:
    Escribir saldo y no entregar dinero
  En cualquier caso:
    El recibo incluye:
      - número del cajero automático
      - fecha y hora
      - número de la cuenta
      - tipo de transacción
      - cantidad movida
      - nuevo saldo
```

Identificar restricciones y dependencias funcionales entre objetos

En el ejemplo de los cajeros automáticos:

- El saldo de una cuenta no puede ser negativo

o bien:

- El saldo de una cuenta, si es negativo, no puede rebasar el límite de crédito.

Definir criterios de optimización (iterar)

En el ejemplo de los cajeros automáticos:

- Minimizar el número de mensajes enviados entre localidades diferentes.
- Minimizar el tiempo de bloqueo de una cuenta.
- Extremadamente urgente: minimizar el tiempo de bloqueo de un banco entero.

Añadir métodos

Las funciones del modelo funcional pueden ser simples transferencias de información, o corresponder a un método de algún objeto (operaciones interesantes). En este caso hay que asignarlos y añadirlos al modelo de objetos.

En el ejemplo de los cajeros automáticos, son interesantes:

- Comprobar contraseña (método de *Autorización de la tarjeta*).
- Actualizar cuenta (método de la clase *Cuenta*).

Se añadirán otros métodos, no relacionados con ninguna de las cuestiones anteriores, procedentes de nuestro conocimiento del tema:

- Cerrar una cuenta (sobre *Cuenta*).
- Autorizar una tarjeta de crédito (sobre *Cuenta*, parámetro la *Autorización de la tarjeta*).
- Crear una cuenta (sobre *Banco*, parámetro el *Cliente*).
- Crear una tarjeta de crédito (sobre *Banco*, parámetro el *Cliente*).

- Cerrar una autorización (sobre *Autorización de la tarjeta*).