Introducción a UML

Qué es UML

UML es un lenguaje gráfico que permite modelar, visualizar y documentar sistemas. Esta compuesto por distintos diagramas que permiten ir representando las distintas vistas de un sistema, cada diagrama tiene un objetivo bien definido.

UML significa Unified Modeling Language o Lenguaje Unificado de Modelado, y esta basa en tres principios fundamentales:

- Es un Lenguaje: está formado por elementos y reglas bien definidas, que poseen su propia sintaxis y semántica
- Esta Unificado: unifica los distintos criterios utilizados antes de su creación, es decir que toma las mejores propuestas de herramientas previas para presentar una propuesta sumamente abarcativa e integradora
- Permite Modelar: está basado en la construcción de modelos que permite representar abstracciones de la realidad.

UML está estrechamente ligado con el paradigma de objetos, lo que permite construir sistemas de información de una forma mucho más intuitiva, integrada y sencilla con el proceso de desarrollo.

UML no es una metodología que presenta los pasos a seguir para realizar un desarrollo, sino que es un lenguaje gráfico de modelado.

Qué es un Modelo

Un modelo es una posibilidad de visualizar a escala o de una manera simulada algo que será construido en la realidad. Es posible decir que antes de construir un edificio se realizan maquetas a escala que representa modelos a seguir, así como también cuando se construye un auto se confeccionan distintos planos o modelos a escala para intentar simular o prever su comportamiento.

Académicamente, es posible definirla como una abstracción de la realidad, es una simplificación de la realidad, con el objetivo final de pasar del modelo a producto real.

Los modelos pueden ser expresados en distintos niveles de precisión, desde algo muy genérico presentando una visión, hasta algo mucho mas especifico que representa un gran compromiso con el producto a construir.

Cómo nace UML

La historia cuenta que UML da sus primeros pasos con la unión de los “tres amigos”: Booch, Rumbaugh y Jacobson.

En los años 80 cada uno utilizaba un lenguaje propietario aunque como denominador común tenían como objetivo el desarrollo de sistemas, con previa modelización. A partir de los años 90 comienzan a intercambiar ideas para intentar unificar criterios.

Booch es el fundador de Rational Software Corp, y recluta en el año 1995 a Rumbaugh y Jacobson para comenzar a determinar una especificación genérica, sencilla y abarcativa.

Así es como las grandes empresas de tecnologías de información – entre ellas Rational - deciden formar un consorcio para la construcción de un Lenguaje Unificado de Modelado: UML. La primer versión de UML, la 1.0, sale a la luz en el año 1997, y a partir de 1998 un organización llamada OMG (Object Management Group) se encargó de generar nuevas revisiones. En el año 1998 UML se establece como Standard de facto en la industria del Software.

Dónde se utiliza

UML se utiliza dentro del marco de IT, aunque puede utilizarse en proyectos que no son de tecnología de la información, como ser el modelado de un motor o de una turbina.

En el campo de IT, se utiliza tanto para sistemas monolíticos como para sistemas distribuidos, abarca desde proyectos pequeños hasta grandes proyectos. Permite realizar la integración del software, donde representa el correcto enlace de los roles para lograr el éxito de la construcción del sistema. En proyectos de software, es utilizado desde la gestación hasta la instalación y el testing.

Si bien para utilizar UML es posible realizar los distintos diagramas con papel y lápiz, es conveniente contar con alguna herramienta del tipo IDE que facilite su construcción, corrección e integración entre diagramas.

Introducción a los diagramas de UML

Introducción

UML está organizado en una serie de diagramas que tienen objetivos bien definidos, con una sintaxis y semántica determinada, que intentan representar / modelar distintas vistas de un sistema.

Los Diagrama de UML

Diagrama de Clases

El Diagrama de Clases tiene como objetivo describir las clases del dominio y sus relaciones. Permite modelar la estructura del sistema desde un punto de vista estático, modelando las clases desde distintos enfoques de acuerdo a la etapa del proyecto.

Esta compuesto por clases, relaciones entre clases y opcionalmente los paquetes que agrupan a las clases.

Diagrama de Objetos

El Diagrama de Objetos tiene como objetivo describir los objetos del dominio y sus relaciones. Permite representar al sistema en un momento determinado del tiempo, es proporcional a obtener una fotografía o snapshot del sistema en un momento determinado.

Esta compuesto por objetos y relaciones de enlace. También es posible pensarlo como una instancia de un Diagrama de Clases.

Diagrama de Casos de Uso

El Diagrama de Casos de Uso tiene como objetivo describir las acciones del sistema desde el punto de vista del usuario. Representa las formas que tiene un usuario de utilizar un sistema, y se puede utilizar como un “contrato” entre cliente y proveedor de software para determinar la funcionalidad del sistema, es decir los requisitos funcionales.

Esta compuesto por actores (agentes externos al sistemas, pueden ser usuarios u otros sistemas), casos de uso y distintos tipos de relaciones. Es posible construir diagramas con diferentes niveles de detalle.

Diagrama de Estados

El Diagrama de Estados tiene como objetivo describir los estados por los cuales puede pasar un objeto durante su ciclo de vida. Permite modelar tanto estas simples como compuestos y concurrentes.

Esta compuesto por estados, pseudo-estados y transiciones entre estados.

Diagrama de Actividades

El Diagrama de Actividades tiene como objetivo describir las acciones que ocurren dentro de un proceso. Se utiliza principalmente para modelar flujo de trabajo o workflow, con lo cual visualiza las acciones de manera ordenada.

Esta compuesto por acciones simples y concurrentes, y transiciones entre las acciones.

Diagrama de Comunicación

El Diagrama de Comunicación tiene como objetivo describir cómo colaboran o se comunican los distintos objetos entre sí para conseguir un objetivo. Se lo suele llamar también Diagrama de Colaboración. Es posible verlo como una extensión del Diagrama de Objetos, ya que es muy parecido pero tiene como valor agregado los mensajes que se envían entre los objetos.

Esta compuesto por objetos, relaciones de enlace y relaciones del tipo llamadas, representando que objeto se comunica con que otro.

Es semánticamente equivalente al Diagrama de Secuencia.

Diagrama de Secuencia

El Diagrama de Secuencia tiene como objetivo describir cómo colaboran los distintos objetos entre sí para conseguir un objetivo a lo largo del tiempo. Esta directamente relacionado con el Diagrama de Comunicación ya que el objetivo es el mismo, pero tiene la particularidad de estar obligatoriamente ordenado en el tiempo.

Esta compuesto por objetos y relaciones del tipo llamadas, representando que objeto se comunica con que otro.

Es semánticamente equivalente al Diagrama de Comunicación.

Diagrama de Componentes

El Diagrama de Componentes tiene como objetivo describir la relación que existe entre los distintos componentes

del sistema.Está directamente vinculado con el diseño del sistema, permitiendo modelar las relaciones e interfaces que existen entre los componentes. Está orientado a la implementación del sistema.

Esta compuesto por componentes, interfaces y sus relaciones.

Diagrama de Despliegue

El Diagrama de Despliegue tiene como objetivo describir la arquitectura de un sistema. Es posible representar la arquitectura desde el punto de vista lógico, basándose en la organización del software, o desde una punto de vista físico, representando directamente cada unidad de hardware.

Esta compuesto por nodos, componentes y sus relaciones.

Clasificación

Es posible clasificar a los diagramas según si tienen alguna relación con el tiempo o no. Existen dos posibles categorías, los diagramas estáticos y los diagramas dinámicos.

Diagramas Estáticos

Los Diagramas Estáticos son aquellos que no tienen ninguna relación con el tiempo. Generalmente representan a estructuras o a posibles acciones pero sin una relación directa con el tiempo.

Estos diagramas son:

- Diagrama de Clases
- Diagrama de Objetos
- Diagrama de Casos de Uso
- Diagrama de Componentes
- Diagrama de Despliegue

Diagramas Dinámicos

Los Diagramas Dinámicos son aquellos que tienen relación con el tiempo. Están directamente vinculados con acciones que ocurren bajo cierta secuencia, es decir primero una acción, luego otra, y así sucesivamente.

Estos diagramas son:

- Diagrama de Actividades
- Diagramas de Interacción (es una subcategoría, que incluye a los Diagramas de Secuencia y Comunicación)
- Diagrama de Estado

Diagramas Estructurales

Los Diagramas Estructurales son aquellos que reflejan relaciones estáticas de una estructura.

Estos diagramas son:

- Diagrama de Clases
- Diagrama de Objetos

- Diagrama de Componentes
- Diagrama de Despliegue

Diagramas de Comportamiento

Los Diagramas de Comportamiento son aquellos que reflejan características de comportamiento del sistema o modelan procesos de negocios.

Estos diagramas son:

- Diagrama de Casos de Uso
- Diagrama de Comunicación
- Diagrama de Secuencia
- Diagrama de Actividades
- Diagrama de Estados

El Diagrama de Clases (Class Diagram)

Definición

El diagrama de clases es un diagrama que permite modelar la estructura del sistema desde un punto de vista estático, modelando las clases desde distintos enfoques de acuerdo a la etapa del proyecto. Describe tipos de jerarquías, relaciones y abstracciones.

Objetivo

“..Describir las clases del dominio y sus relaciones..”

Elementos

Clase

Una clase representa a una agrupación de cosas, es una plantilla para armar un objeto. Las clases son detectadas – en la mayoría de los casos - como sustantivos en singular.

Ejemplos de una clase puede ser la clase Auto, que es una clase representante de todos los autos posibles (autos de carrera, autos urbanos, cualquier marca, patente, color, etc.) Otro ejemplo puede ser la clase Animal, representando a todos los animales posibles (mamíferos, herbívoros u otros, cualquier cantidad de patas, color, etc.)

Las clases están formadas por atributos y métodos, y por convención, la primera letra debe estar en mayúscula.

Qué son los atributos

Los atributos son características que posee una clase, y determinan el estado que posteriormente tendrá un objeto En el caso de la clase Auto, atributos pueden ser color, marca, modelo, cantidad de puertas y velocidad. Por convención, la primera letra debe estar en minúscula.

Qué son los métodos

Los métodos son las responsabilidades (o comportamiento) que realiza una clase. Generalmente los métodos son verbos. Por convención, la primera letra debe estar en minúscula.

Representación gráfica de una clase

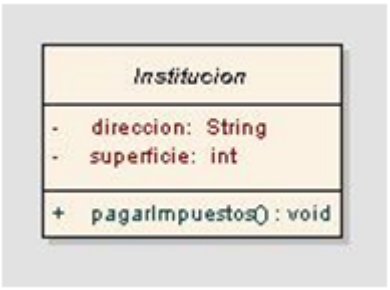
Las clases abstractas

Las clases abstractas son clases que representan un concepto abstracto, de carácter muy general. Por ejemplo una institución, que tiene los atributos dirección y superficie, pero no es posible determinar que tipo de institución es.

Es una clase que no se puede instanciar, es decir que no se puede crear un objeto a partir de ésta clase. Se utiliza como base para otras clases en la relación de generalización, pero no tiene sentido por sí sola.

Las clases que no son abstractas se denominan concretas. Por convención, la primera letra debe estar en mayúscula, y la palabra en negrita e itálica.

Representación gráfica de una clase abstracta



Interfaz

A diferencia de la clase, la interfaz define únicamente un comportamiento, es decir un conjunto de métodos que no poseen implementación. Estos métodos deberán ser implementados por las clases que decidan implementar la interfaz. Representa un “contrato” que una clase debe respetar en caso de implementar la interfaz.

Por ejemplo, se puede crear un interfaz denominada Volador, que tiene los métodos despegar,

aterrizar y volar.

Por convención, para definir el nombre de una interfaz se aplican las mismas reglas que para una clase.

Relaciones

Generalización

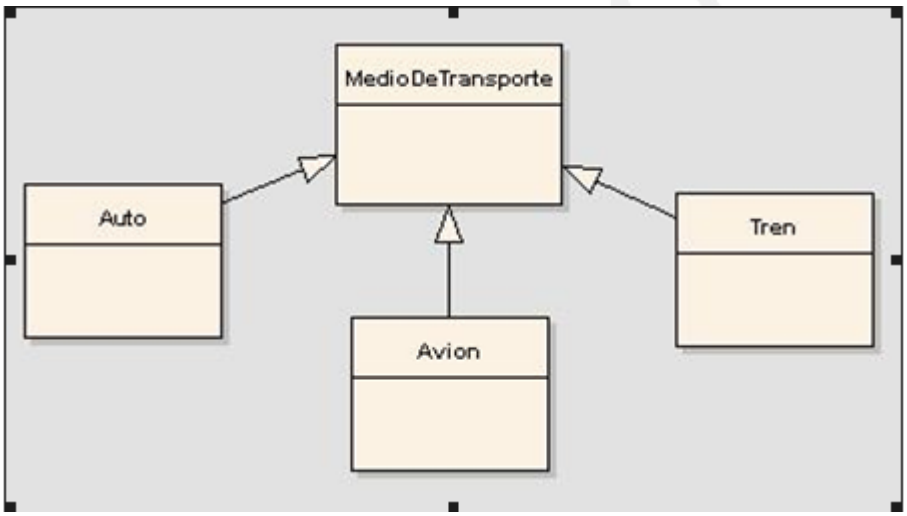
La relación de generalización se produce entre dos clases. La clase superior es una generalización de la clase inferior, como así también la clase inferior es una particularización de la clase superior.

A la clase superior se la denomina super clase, y a la clase inferior se la denomina subclase. La subclase deberá respetar la relación “es un” o “is a”, que representa a la relación de generalización.

Al construir varios relaciones de este tipo entre clases, se genera la jerarquía de clases (o árbol de clases). En términos de un lenguaje de programación, es posible entenderla como herencia.

Por ejemplo, la clase MedioDeTransporte puede ser una superclase, y algunas de sus subclases pueden ser

Auto, Avión y Tren.

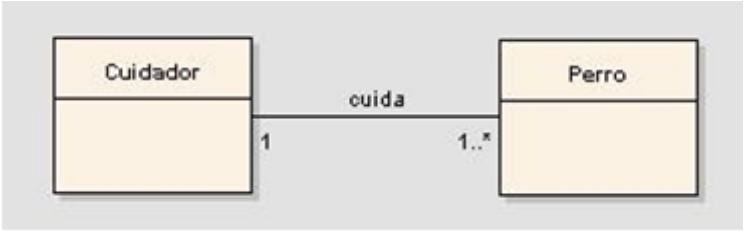


Asociación

La relación de asociación representa una asociación entre dos clases, donde una clase “es socia” de otra o tienen algún tipo de relación. Es común describir con un nombre definido por el usuario la relación entre ambas.

Por ejemplo, la clase Cuidador tiene una relación con la clase Perro, donde Cuidador “cuida” al Perro. Es posible determinar la multiplicidad de los extremos de la relación, en el caso anterior un cuidador cuida de uno a muchos

perros.



Existen dos relaciones denominadas Composición y Agregación que son casos particulares de la relación de Asociación.

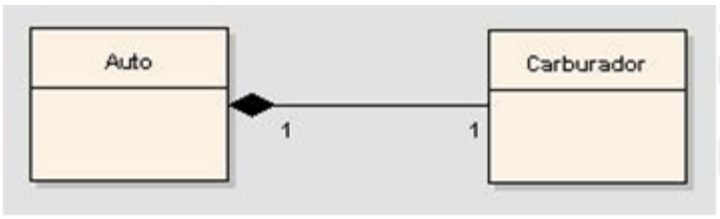
Composición

La relación de composición se puede describir como “está compuesta por”, ya que una clase determina la existencia de la otra. Si Clase1 está compuesta por Clase2 entonces Clase1 determina la existencia de Clase2. Clase2 no podrá existir por si sola si no existe Clase1, es decir que Clase1 controla el tiempo de vida de Clase2. Si la Clase1 es eliminada, también será eliminada Clase 2, es decir que si se elimina el “todo” (Clase1), también serán eliminadas sus partes (Clase2).

UML denomina a la relación como “strong has-a relationship”, representando un fuerte sentido de pertenencia del “todo” hacia la “parte”.

Por ejemplo, un Auto está compuesta por un Carburador, con lo cual el Carburador no tiene sentido por sí solo si no existe el Auto. Dicho Carburador puede utilizarse únicamente para ese Auto, y no para otros, es decir que el mismo Carburador no puede utilizarse al mismo tiempo en

más de un Auto.



Otro ejemplo muy ilustrativo puede ser la relación entre una Tabla y una Columna, donde la Columna es construida si la Tabla es construida, y la Columna es eliminada si la Tabla es

eliminada.

Agregación

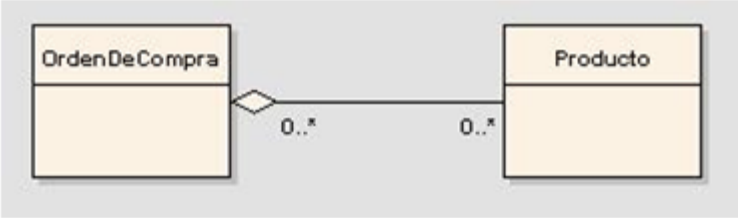
La relación de agregación se puede describir cómo “tiene como partes a”. A diferencia de la composición, si Clase1 tiene como partes a Clase2, Clase1 no determina la existencia de Clase2. Clase2 puede existir aún cuando Clase1

no exista, es decir tiene sentido por sí sola. Clase1 no controla el tiempo de vida de Clase2. Si la Clase1 es eliminada, puede no ser eliminada Clase2, es decir que si se elimina el “todo” (Clase1), no necesariamente serán eliminadas sus partes (Clase2).

UML denomina a la relación como “weak has-a relationship”, representando un débil sentido de pertenencia del “todo” hacia la “parte”.

Por ejemplo, una OrdenDeCompra tiene como partes a uno o muchos Producto(s), pero si el Producto no forma parte de ninguna OrdenDeCompra, sigue teniendo sentido por si mismo. Si la OrdenDeCompra es eliminada, el

Producto no será eliminado.



Implementación o Realización

La relación de implementación se produce entre una clase y una interfaz. La clase que implementa la interfaz tiene la obligación de implementar todos los métodos que forman parte de esa interfaz.

Por ejemplo, un Avión para “saber volar” deberá implementar un interfaz denominada Volador, que incluye los métodos despegar, aterrizar y volar.

Clases Estereotipadas

Que es un estereotipo de clase

El estereotipo representa la construcción de un nuevo elemento de UML que extiende a partir de uno ya existente, en el caso de las clases representa a una categoría o un tipo nuevo de clases. Representa una funcionalidad determinada, identificada por su nombre.

El Proceso Unificado de Desarrollo (presentado más adelante) utiliza tres estereotipos de clases que se han estandarizado en el mercado, estos son Boundary, Control y Entity.

El estereotipo Boundary

El estereotipo Boundary se utiliza para representar clases que se encuentran en el límite (bound) del sistema. Estas clases representan a la interfaz de usuario dentro de un sistema, generalmente implementadas como ventanas. Se utilizan para capturar la interacción entre el usuario y el sistema a nivel de pantalla.

Estas clases dentro de una arquitectura multicapa generalmente pertenecen a la Capa de Presentación (PL o Presentation Layer).

En el patrón de diseño M-V-C representa a la vista.

El estereotipo Control

El estereotipo Control se utiliza para representar clases que se encargan de controlar los procesos de negocios, son clases que llevan a cabo las reglas de negocios, realizando la coordinación entre las clases del tipo Boundary y las clases del tipo Entity. Se encargan de la organización y planificación de actividades.

Estas clases dentro de una arquitectura multicapa generalmente pertenecen a la Capa de Negocios (BL o Business Layer).

En el patrón de diseño M-V-C representa al controlador.

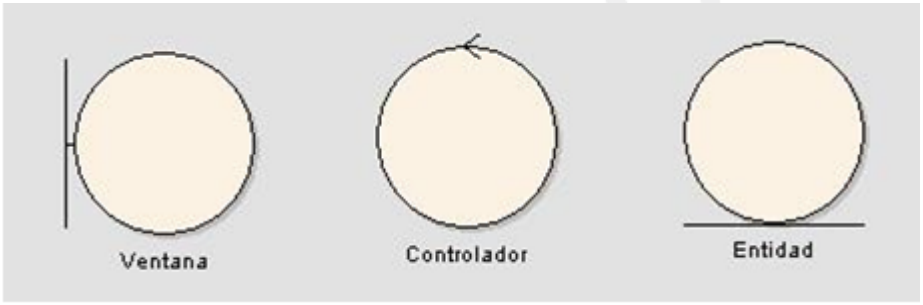
El estereotipo Entity

El estereotipo Entity se utiliza para almacenar o persistir información propia del sistema.

Estas clases dentro de una arquitectura multicapa generalmente pertenecen a la Capa de Acceso a Datos (DAL o Data Access Layer).

En el patrón de diseño M-V-C representa al modelo.

Representación gráfica



Aplicación

Modelo de Análisis o Modelo Conceptual

Uno de los posibles usos del diagrama de clases es la construcción del denominado Modelo Conceptual. El modelo conceptual es uno o varios diagramas de clase construidos por un Analista Funcional que está basado en la detección de clases (junto con sus atributos y posibles métodos) y sus relaciones pero desde un punto de vista funcional, es decir dentro de la de la etapa de Análisis.

No se definen características propias de un lenguaje de programación, sino que se intenta reflejar la realidad. En algunos casos se categorizar las clases como entity / controller / boundary, que son estereotipos de RUP (Racional Unified Process) presentados mas adelante.

Adicionalmente, es posible utilizar una CRC Card (Class Responsability and Collaboration) para detallar el nombre de la clase, descripción, atributos y responsabilidades.

Modelo de Diseño

El modelo de diseño es el conjunto de diagramas de clases (puede ser uno solo) que se utiliza como base para realizar la codificación de la aplicación. El encargado de construirlo es el Diseñador, y debe definir todo lo que sea necesario para que el desarrollador pueda codificar sin problemas. Toma como uno de los documentos de entrada el correspondiente al Modelo de Análisis, y se definen nuevas clases (en general las detectadas en Análisis se mantienen) que tiene un perfil netamente de diseño y resuelven cuestiones técnicas y ya no de negocios.

A partir del Modelo de Diseño se genera el código fuente de manera automática que será tomado como base por los desarrolladores. De esta manera el programador no toma decisiones ni de Análisis ni de Diseño, lo cual queda restringido a programar en el código recibido.

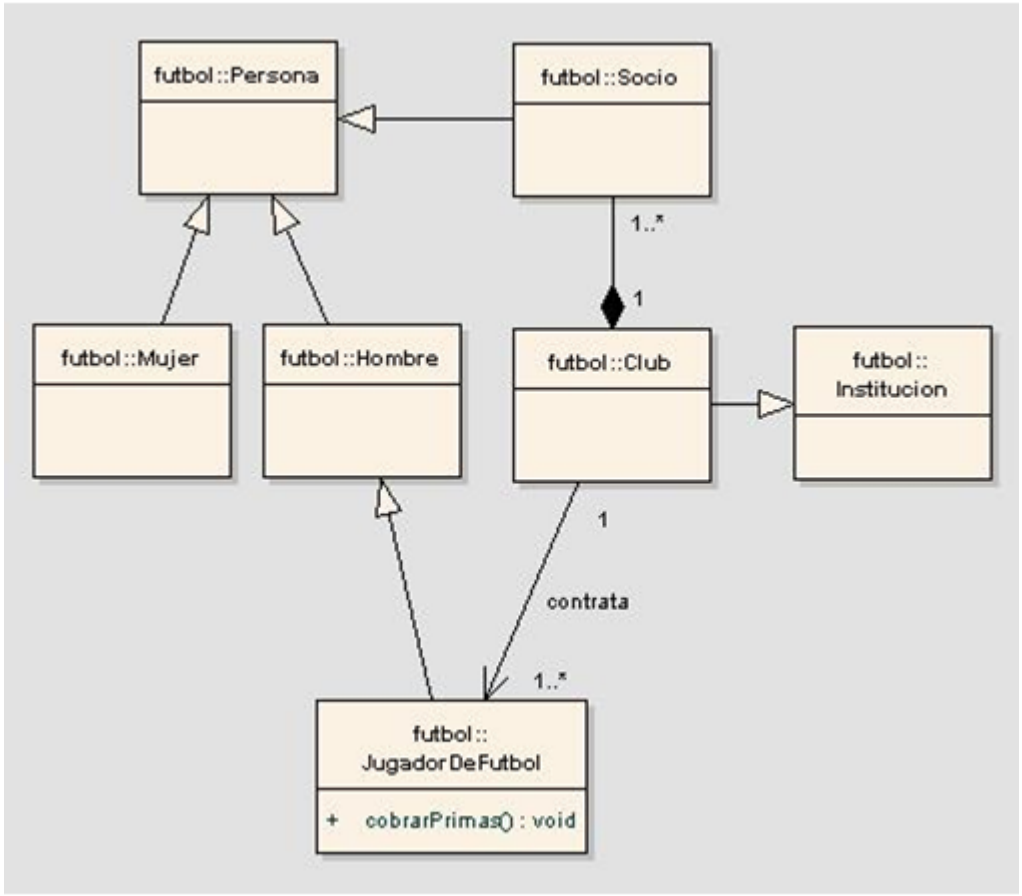
El modelo tiende a evolucionar en la fase de construcción a través de feed-backs que realizan los desarrolladores.

Diseño de Base de Datos

El modelo de datos o diseño de una base de datos también es posible realizarlo a través de un diagrama de clases. En este caso las clases representan las tablas y los atributos representan los campos.

Para esto es necesario utilizar estereotipos (ver Sección Conceptos Generales), determinando el estereotipo <<table>> para las tablas, el estereotipo <<column>> para los campos, y otros estereotipos posibles como <<PK>> para las claves primarias y <<FK>> para las claves foráneas, entre otros.

Ejemplo



Conceptos Generales

Estereotipos

Un estereotipo es un nuevo elemento de UML que extiende a partir de uno existente, quedando definido con una semántica extendida y un nuevo icono gráfico, aunque este último es opcional.

En caso de no establecer un icono para el estereotipo, se visualiza de la forma <<nombreEstereotipo>>.

Los estereotipos pueden utilizarse para cualquier elemento de UML, como ser clases y/o relaciones. Las clases estereotipadas más comunes son las clases del tipo <<Boundary>>, <<Control>> y <<Entity>>, y entre las relaciones la más conocida es <<use>>.

Valor etiquetado (Tagged Values)

El valor etiquetado se utiliza para guardar información adicional dentro un elemento de UML. Está formado por una etiqueta (tag) y un valor (value), donde por ejemplo, se puede guardar el nombre de la persona que construyó una clase o la versión de la misma.

Ingeniería Directa

La ingeniería directa es el proceso que permite generar código fuente en cualquier lenguaje a partir de los distintos diagramas de UML. Por ejemplo, a partir del diagrama de clases es posible generar el “esqueleto del sistema”, es decir el código fuente formado por clases y métodos, pero vacíos.

El Enterprise Architect permite generar código en diversos lenguajes, entre ellos Java, VB.Net, C#, C++, Perl, Delphi y PHP.

Ingeniería Inversa

La ingeniería inversa es el proceso que permite generar diagramas a partir de código fuente. Se utiliza generalmente para construir el diagrama de clases a partir de las clases de código fuente de cualquier lenguaje.

El Enterprise Architect permite generar el diagrama de clases a partir del código fuente de diversos lenguajes tales como Java, VB.Net, C#, C++, Perl, Delphi y PHP

El Lenguaje XMI

XMI significa XML Metadata Interchange, y es una forma de almacenar los diagramas de UML en un formato de texto basado en XML. El objetivo de XMI es permitir la portabilidad de un proyecto armado en UML desde cualquier tipo de aplicación.

Por ejemplo, es posible utilizar el Enterprise Architect y guardar el proyecto en formato .xml, para luego utilizar otra aplicación como Rational o PoseidonUML y poder abrir el proyecto sin inconvenientes.