El Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado más conocido como UML es una notación gráfica que permite realizar modelos sobre cualquier aspecto durante la construcción de un software. A continuación realizaremos una descripción de los elementos que componen esta notación, sus aplicaciones y ejemplos de su uso.

Análisis de Sistemas - 2013

El Lenguaje Unificado de Modelado

1. Definiciones

El lenguaje Unificado de Modelado, de ahora en adelante UML, es un lenguaje gráfico estándar que permite generar modelos en el desarrollo de software durante todas las etapas de su construcción. UML permitirá generar modelos durante las etapas de Análisis y Diseño principalmente, pero también ayudará a trabajar en las etapas de Construcción, Pruebas e Implementación.

Hablamos de generar modelos, pero no los definimos. ¿Qué es un modelo? Un modelo es una simplificación de la realidad, de una realidad que nos interesa analizar. A su vez, esa realidad puede ser representada de distintas formas y con distintos niveles de detalle.

Veamos un ejemplo: si encargamos la construcción de una casa, los arquitectos nos mostraran algunas imágenes de la casa a nivel general, del frente de la misma, del interior, del exterior, etc. Posiblemente nos podrán realizar una maqueta. Más adelante nos confeccionarán los planos de la construcción para detallar las habitaciones con sus correspondientes medidas y aberturas. Tendrán que generarse planos para las cañerías, conexiones eléctricas, etc. Todos estos documentos mencionados, dibujos, planos, maquetas son modelos que nos mostraran la futura casa desde distintos puntos de vistas y cada uno con un nivel de detalle superior.

Durante el desarrollo de un sistema se generan modelos que nos mostrarán al software desde distintos puntos de vista y con menor o mayor detalle. Se generarán modelos para: ver cuál será la funcionalidad del sistema, con quienes o cuales sistemas interactuará; para detallar las clases que se utilizarán, atributos y operaciones; modelar aspectos dinámicos o estáticos del sistema; o bien detallar cómo se implementará físicamente. Todos estos y otros modelos podrán ser confeccionados a partir de los elementos que brinda UML.

UML es un lenguaje y como tal provee un vocabulario y un conjunto de reglas para combinar palabras de ese vocabulario. Puede utilizarse para Visualizar, Especificar, Construir y Documentar artefactos de sistemas orientados a objetos. (Booch, Rumbaugh, & Jacobson, 2006).

Visualizar

UML ayuda a representar lo que se desea construir.

Proporciona un soporte gráfico para la construcción de sus modelos, lo que permite que algunos modelos sean más fáciles de comprender.

Mejora la comunicación entre desarrolladores ya que al utilizar una notación bien definida, estandarizada y conocida, cualquier desarrollador puede interpretar el significado de un modelo confeccionado en UML sin ambigüedades.

Mejora la comunicación entre cliente y desarrollador, ya que algunos modelos pueden ser entendidos fácilmente por un cliente aunque no tengan el conocimiento previo.

Especificar

UML permite detallar distintos aspectos de un sistema tales como requisitos, funcionalidades, arquitectura, comportamiento, entre otros.

A partir de las reglas que define el lenguaje para confeccionar los distintos modelos estos se construyen en forma precisa y no presentan ambigüedades.

Construir

UML no es un lenguaje de programación y por lo tanto no puede construir un software, pero si permite generar diagramas que ayuden a la construcción de un software, como por ejemplo un diagrama de clases.

Existen algunas herramientas que permite transformar rápidamente un diagrama de UML en código escrito en lenguaje de programación.

Documentar

UML proporciona modelos que se utilizan durante todas las etapas del desarrollo de software. (Análisis, Diseño, Codificación, Pruebas, Implantación). Estos mismos modelos pueden utilizarse como documentación en cada etapa y para dejar registrado cuales fueron las decisiones que se fueron adoptado en el desarrollo.

Modelo - Proceso

UML no es un proceso, permite crear modelos, define como se realizan, pero no indica cuales o cuántos modelos deben elaborarse durante el desarrollo de un sistema ni en qué momento. Estas son propiedades que define el proceso de desarrollo que se haya seleccionado para la construcción del sistema. Es entonces el proceso de desarrollo quien debe indicar: quién es el responsable de la elaboración de un modelo, qué artefacto se debe construir, cómo construirlo y en qué momento.

2. Un poco de Historia

UML fue creado por tres personas Grady Booch, Ivar Jacobson y James Rumbaugh. Cada uno de ellos tenía su propia metodología de trabajo:

-El método de Booch de Grady Booch

-El método OOSE (Object-Oriented Software Engineering) de Ivar Jacobson

-El método OMT (Object Modeling Technique) de James Rumbaugh

A mitad de la década del 90 cada uno de ellos comenzó a adoptar ideas de los otros métodos, fue entonces cuando se presentó la idea de formar un lenguaje en común en donde se pudieran aportar ideas de los tres métodos.

En octubre de 1994 Rumbaug se une a Booch en Rational Sofware Corporation, con el objetivo de unificar sus dos métodos (El método de Booch y OMT) La primera versión se llamó UM (Método Unificado) y fue publicada un año más tarde en octubre de 1995. En esa misma época Jacobson se incorpora a Rational y ahora se incorpora el método OOSE. Durante 1996 se estuvo trabajando en la documentación y se pidió colaboración a distintas comunidades de la ingeniería de software. Se presento a distintas organizaciones y empresas las cuales mostraron un fuerte en obtener un fuerte lenguaje de modelado estándar. Se presentó entonces la primera versión: UML 1.0. En julio de 1997 se presentó a la OMG (Object Management Group) para su estandarización una versión un poco más refinada UML 1.1. A fines de ese mismo año ya había sido adoptada por la OMG.

Durante varios años la OMG Revision Task Force fue la encargada del mantenimiento de UML. Se crearon las versiones 1.3, 1.4, 1.5.

Desde el año 2000 al 2003 un nuevo grupo de colaboradores estuvieron realizando actualizaciones sobre UML, y así surgió la versión 2.0. Fue presentada a la OMG y a principios de 2005 fue adoptada.

3. Aplicación de UML

Está pensado para utilizarse en el desarrollo de sistemas de gran envergadura, sin embargo se puede utilizar en sistemas medianos o chicos adaptándose a cada caso en particular.

Se ha utilizado UML para el desarrollo de sistemas de distintos tipos, algunos ejemplos son:

- Sistemas Administrativos
- Sistemas Financieros
- Sistemas WEB
- Telecomunicaciones
- Sistemas médicos

El uso de este lenguaje no está limitado al desarrollo de software. Se puede aplicar en el desarrollo de otro tipo de sistemas.

4. Regulación y Estándar

OMG (Object Management Group) es una asociación sin fines de lucro que se encarga de realizar estándares en la industria de la computación internacional. Fue fundada en 1989 y desde entonces ha desarrollado varios estándares entre ellos el estándar de UML. La última especificación definida por la OMG es la versión 2.4.1 que fue publicada en agosto de 2011 y se encuentran trabajando en la versión 2.5. Las últimas actualizaciones se pueden consultar desde la página oficial de la OMG www.omg.org/UML.

5. Bloques de UML

El vocabulario de UML está formado por **Elementos, Relaciones** y **Diagramas**.

Los *elementos* son las abstracciones que se utilizarán para construir un modelo. Las *relaciones* definen la forma en que esos elementos se comunicarán entre sí. Los *diagramas* muestran las distintas formas de representar y/o agrupar los elementos y sus relaciones.

5.1. Elementos

Los elementos que componen UML se dividen en cuatro tipos: Elementos Estructurales, Elementos de Comportamiento, Elementos de Agrupación y Elementos de Anotación. A continuación detallaremos cada tipo.

5.1.1. Elementos Estructurales:

Estos elementos representan cosas físicas o abstractas y en su mayoría son aspectos estáticos de un modelo. Hay ocho tipos de elementos estructurales.

- Clase

Una clase es la definición de un conjunto de objetos[1] similares. *En una clase se definen los atributos y responsabilidades que se encuentran en cada objeto de la clase* (Kendall & Kendall, 2011).

Se representan con un rectángulo dividido en tres partes. En la primera se encuentra el nombre de la clase, en la segunda los atributos y en la tercera las responsabilidades.

Figura 1: Clase.

[1] Un objeto es una entidad única que tiene un nombre, atributos, responsabilidades y un estado interno (éste último formado por los valores que toman los atributos en un momento determinado).

➢ Interfaz

Una interfaz es un conjunto de operaciones que muestra el comportamiento visible externo de una clase o componente. Puede representar el comportamiento completo de una clase o solamente de una parte.

Se representa como una clase utilizando el estereotipo <<interface>> mostrando sus operaciones, generalmente sin atributos. Para mostrar la relación de la interfaz con:

- La clase que brinda la interface (Interfaz proporcionada) se utiliza una circunferencia unida a la clase.
- Una clase que utilizará la interfaz (Interfaz requerida) se utiliza una semicircunferencia unida a la clase.

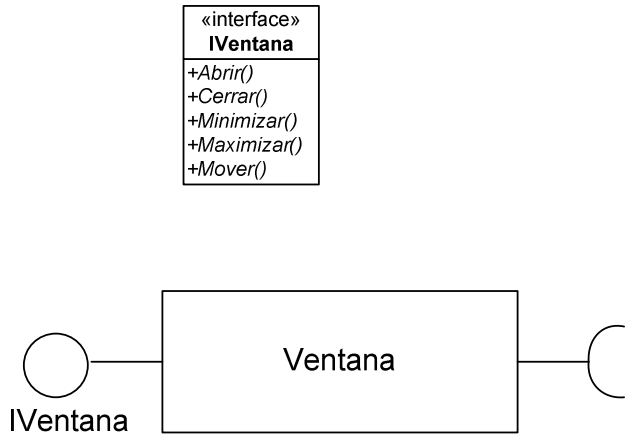


Figura 2: Interfaz. Interfaz proporcionada. Interfaz requerida. (Booch, Rumbaugh, & Jacobson, 2006)

➢ Caso de Uso

Es una funcionalidad completa del sistema que se describe enunciando una secuencia de interacciones entre un actor[2] y un sistema. Se representa con un óvalo en cuyo interior se encuentra el nombre del Caso de Uso.

[2] Actor: es aquello que interactúa con el sistema, puede ser una persona, otro sistema o un componente de hardware.

Figura 3: Casos de Uso.

➢ Clase Activa

Un objeto activo es un objeto que tiene un proceso o hilo y puede iniciar actividades de control. (Booch, Rumbaugh, & Jacobson, 2006).

Una clase activa es una descripción de un conjunto de objetos activos similares, es decir, es una clase cuyas instancias son objetos activos.

Se representa en forma similar a las clases con una línea doble a la derecha e izquierda.

Figura 4: Clase Activa. (Booch, Rumbaugh, & Jacobson, 2006)

➢ Colaboración

Una colaboración es un conjunto de clases, interfaces y otros elementos que trabajan en conjunto para cumplir con alguna funcionalidad en particular. Las colaboraciones se utilizan para definir las realizaciones de Casos de uso y de operaciones.

Figura 5: Colaboración

La Figura 5 muestra la representación de una colaboración, pero lo más importante es lo que se encuentra dentro ella. En el interior de una colaboración se encuentran otros diagramas tales como diagrama de clases, de objetos o de comunicación.

➢ Componente

Es una parte lógica del diseño de un sistema. Proporciona la implementación de un conjunto de interfaces. Se representa con un rectángulo con su nombre en el interior y un icono particular en la parte derecha superior.

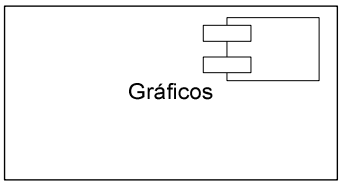


Figura 6: Componente.

Los componentes existen en tiempo de ejecución.

➢ Artefacto

Un artefacto es una parte física del sistema que contienen información como por ejemplo código fuente, archivos ejecutables, scripts, librerías, etc. Se representan con un rectángulo con su nombre en el interior y el estereotipo <<artifact>>.

Figura 7: Artefacto.

Los artefactos existen en tiempo de ejecución.

➢ Nodo

Un nodo es un elemento computacional que tiene procesamiento y memoria. Un conjunto de artefactos pueden ubicarse dentro de un nodo. Se representan con un cubo con su nombre en el interior.

Figura 8: Nodo.

5.1.2. Elementos de Comportamiento

Estos elementos representan la parte dinámica del modelado. Hay tres tipos de elementos de comportamiento:

➢ Interacciones

Una interacción es la comunicación que existe entre un conjunto de objetos para cumplir con un objetivo en común. Las interacciones incluyen los mensajes que son enviados. *La mayoría de las veces, un mensaje*

implica la invocación de una operación o el envío de una seña; un mensaje también puede incluir la creación o la destrucción de otros objetos. (Booch, Rumbaugh, & Jacobson, 2006)

Un mensaje se representa como una línea dirigida acompañada con su nombre.

Figura 9: Mensaje.

- Máquinas de Estado

Una máquina de estados especifica la secuencia de estados por los que pasa un objeto durante su vida útil. El comportamiento de una clase o de un conjunto de clases puede modelarse de esta manera.

Un estado se representa con un rectángulo de puntas redondeadas con su nombre en el interior.

Figura 10: Estado.

- Actividad

Una actividad es un comportamiento que determina un conjunto de pasos o acciones para cumplir con un objetivo.

Se representa gráficamente de igual manera que un estado, con un rectángulo con puntas redondeadas. Los estados y las Actividades se distinguen de acuerdo al contexto en el que se encuentren.

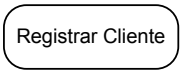


Figura 11: Actividad.

5.1.3. Elementos de Agrupación

Son elementos que se utilizan en la organización de los modelos. Hay solo un tipo de estos elementos.

- Paquete

Un paquete se utiliza para agrupar distintos elementos con el propósito de organización. En un paquete se pueden incluir elementos estructurales, de comportamiento e incluso otros paquetes. Es solo utilizado a nivel conceptual y por este motivo, existe en tiempo de desarrollo. Se representa gráficamente como una carpeta con su nombre en el interior.

Figura 12: Paquete.

5.1.4. Elementos de Anotación

Son elementos que se utilizan para agregar información adicional a los modelos. Hay solo un tipo de estos elementos.

- Nota

Una nota es un comentario que puede agregarse a cualquier elemento o diagrama de UML. No modifican la semántica del modelo simplemente agregan un valor adicional, puede utilizarse para agregar un requisito, una observación, una explicación, un restricción, etc. Puede incluir imágenes.

Se representa con un rectángulo con una esquina doblada y el comentario en el interior.

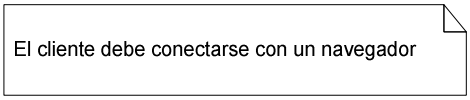


Figura 13: Nota.

5.2. Relaciones

Las relaciones definen las distintas formas de comunicación posibles entre los distintos elementos que hemos enumerado hasta el momento. Existen cuatro tipos de relaciones.

- Dependencia

Es una relación entre dos elementos, de los cuales un elemento (dependiente) utiliza la información y los servicios de otro elemento (independiente) pero no necesariamente a la inversa. (Booch, Rumbaugh, & Jacobson, 2006). El cambio en el elemento independiente afecta directamente sobre el elemento dependiente.

La dependencia se representa con una línea punteada dirigida hacia el elemento dependiente.

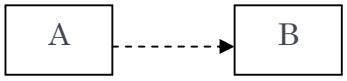


Figura 14: Dependencia.

Se puede ver en la Figura 14: Dependencia. que el elemento A depende del elemento B, en este caso un cambio en el elemento B afectará al elemento A.

- Asociación

Es una relación simple que indica que los objetos de un elemento están relacionados de alguna manera con los objetos de otro elemento. Se representa con una línea sin dirección que une los elementos que participan en la asociación. Normalmente esta relación se acompaña con adornos. UML define adornos que se pueden aplicar a las asociaciones, no es obligatorio el uso de los mismos pero ayudan a la comprensión del modelo:

- Nombre: indica el motivo de la relación. Se puede incluir también el sentido de lectura de la relación.

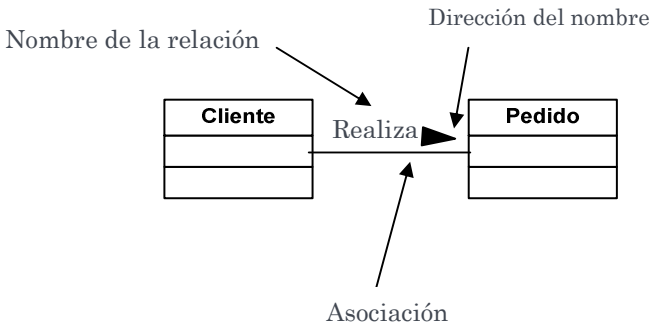


Figura 15: Nombre de una asociación.

- Rol: cuando un elemento se relaciona con otro está cumpliendo con un rol en particular, éste puede incorporarse en la relación, al nombre del rol se lo conoce como "nombre de extremo".

- Multiplicidad: define la cantidad de objetos de un elemento que se relacionan con los objetos del otro elemento. Se determina con un valor mínimo y uno máximo, separados por dos puntos por ejemplo: 1..3, 0..1. Se puede también indicar un solo número o utilizar el * (muchos) cuando no se conoce el número máximo. Ejemplos: 1..*, 0..*.

La Multiplicidad se determina de los dos lados de la relación.

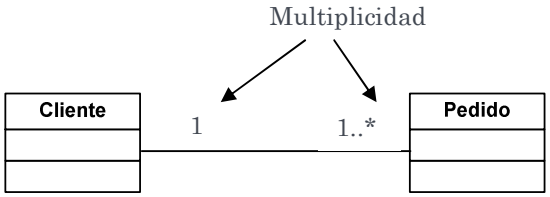


Figura 16: Multiplicidad.

Tipos especiales de Asociación

- Agregación simple: es una relación especial donde los elementos relacionados (generalmente clases) no están al mismo nivel. Uno de los elementos representa algo mayor ("el todo") y el o los otros elementos representan partes más pequeñas ("las partes"). La agregación se representa con un rombo pequeño ubicado sobre la relación de asociación del lado del elemento que representa "el todo".

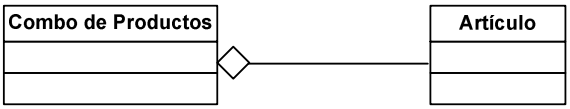


Figura 17: Agregación simple.

En el caso de la agregación simple si la clase que conforma "el todo" deja de existir, las clases que representan "las partes" pueden seguir existiendo en forma independiente. En el ejemplo de la Figura 17 si un combo se elimina, los artículos individualmente se pueden seguir utilizando.

- Composición: es una variación de la agregación en donde existe una fuerte conexión entre "el todo" y "las partes". En una composición si la

clase compuesta deja de existir, las partes también dejan de existir porque no tienen sentido por sí mismas.

La composición se representa con un rombo negro ubicado sobre la relación de asociación del lado de la clase principal.

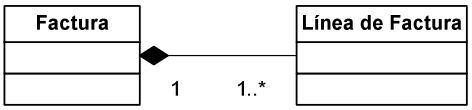


Figura 18: Composición.

- Generalización

En la generalización se encuentran dos tipos de elementos: Un elemento general llamado padre o superclase y un elemento más especifico llamado hijo o subclase. En esta relación las clases "padres" tienen la capacidad de transferir a las clases "hijos" sus atributos y responsabilidades. En general en la clase hijo se agregan atributos o responsabilidades propias.

A esta relación también se la conoce con el nombre de Herencia.

Gráficamente se representa con una línea punteada con una flecha vacía apuntando a la clase padre.

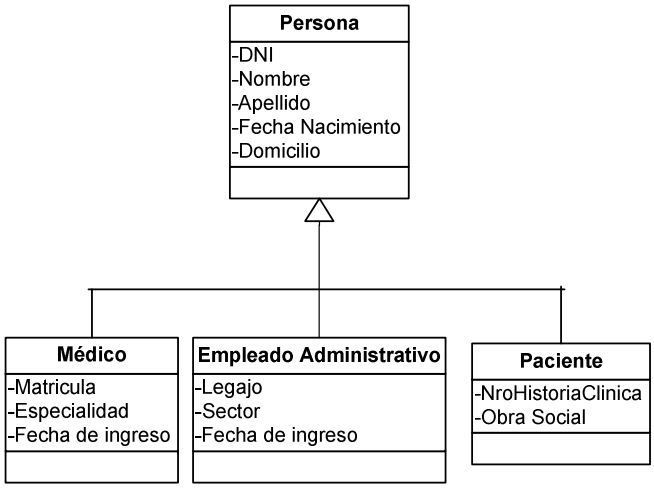


Figura 19: Generalización

Si una clase tiene un único padre, se dice que se utiliza una herencia simple. Si una clase tiene más de un padre utiliza entonces, una herencia múltiple.

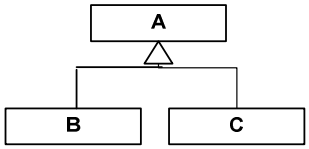


Figura 20: Herencia Simple.

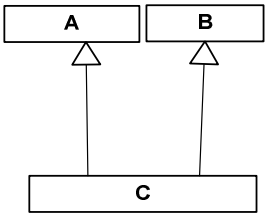


Figura 21: Herencia Múltiple.

- Realización

La realización se utiliza solamente en dos contextos: En interfaces y en el uso de colaboraciones.

Se utiliza entre clasificadores[3], en donde *un clasificador especifica un contrato que otro clasificador garantiza que cumplirá.* (Booch, Rumbaugh, & Jacobson, 2006)

Normalmente se utilizará para mostrar la relación entre una interfaz y una clase que proporciona los servicios que la interfaz necesita o para especificar la relación entre un caso de uso y la colaboración que realiza ese caso de uso.

Figura 22: Realización.

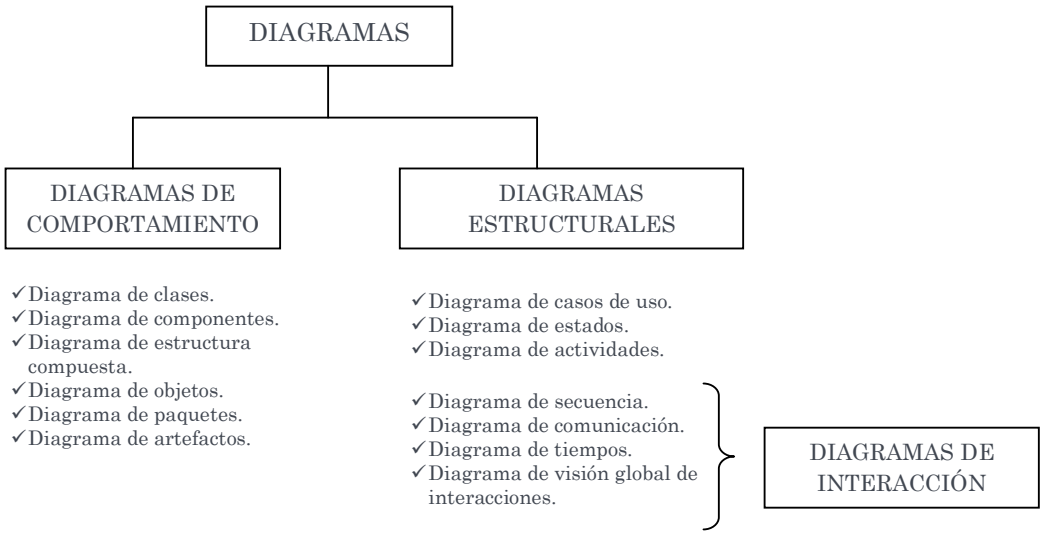
[3] Un clasificador es un bloque de construcción que define aspectos estructurales o de comportamientos.

5.3. DIAGRAMAS

Un diagrama es una representación gráfica de un conjunto de elementos, que la mayoría de las veces se dibuja como un grafo conexo de nodos (elementos) y arcos (relaciones). (Booch, Rumbaugh, & Jacobson, 2006)

Cada diagrama de UML indica de qué forma combinar los elementos y las relaciones que estuvimos analizando hasta el momento. Se utilizan para ver al sistema en construcción desde distintos puntos de vista.

UML presenta varios tipos de diagramas:



Diagramas Estructurales

Permiten modelar aspectos estáticos del sistema. Son diagramas que modelan propiedades que se mantienen estables, no muestran flujos de mensajes entre sus elementos ni secuencia. Dentro de esta clasificación se presentan:

Diagrama de clases.
Diagrama de componentes.
Diagrama de estructura compuesta.
Diagrama de objetos.
Diagrama de paquetes.
Diagrama de artefactos.

Diagramas de Comportamiento

Permiten modelar aspectos dinámicos del sistema. Estos diagramas permiten ver flujo de mensaje entre elementos, secuencia, actividades, y el movimiento físico de componentes. Dentro de esta clasificación se presentan:

Diagrama de casos de uso.
Diagrama de estados.
Diagrama de actividades.

Diagramas de Interacción

Son un tipo especial de diagramas de comportamiento. Se destacan por mostrar "interacciones". Una interacción es flujo de mensajes que existe entre dos o más objetos. Dentro de esta clasificación se presentan:

Diagrama de secuencia.
Diagrama de comunicación.
Diagrama de Tiempos.
Diagrama de visión global de interacciones.

Estos diagramas son los que UML presentan y a continuación describiremos. No son los únicos posibles ya que a veces se encuentran variaciones de algún diagrama que se aplica a algún caso en particular.

➢ Diagrama de clases

Este diagrama muestra las relaciones entre clases, interfaces y colaboraciones del sistema a desarrollar.

Elementos:

- Clase
- Interfaz
- Relaciones de Dependencia, Asociación (con todos sus adornos), Generalización.

Aplicaciones:

- Modelar las abstracciones y responsabilidades del sistema.
- Modelar el dominio. (Modelo de Dominio).
- Modelar la vista estática del sistema.
- Modelar colaboraciones.
- Modelar un diseño conceptual de base de datos.

Ejemplo:

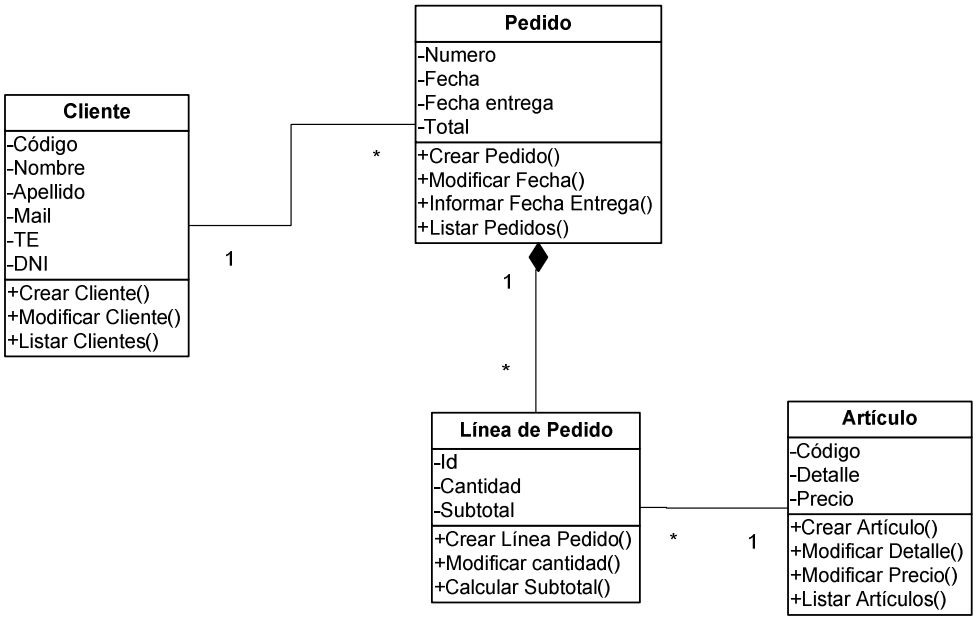


Figura 23: Diagrama de Clases

➢ Diagrama de objetos

Es un diagrama que permite ver las relaciones entre objetos en un determinado momento. Es similar al diagrama de clases, la diferencia radica en lo siguiente: el diagrama de clases define ciertas clases, el diagrama de objetos muestra las instancias de esas mismas clases, dándole valores reales en un momento específico.

Elementos:

- Objetos
- Enlaces

Un enlace es la conexión entre dos objetos. Representa la instancia de una "asociación".

Aplicación:

- Modelar la vista estática del sistema a partir de casos reales.

Ejemplo:

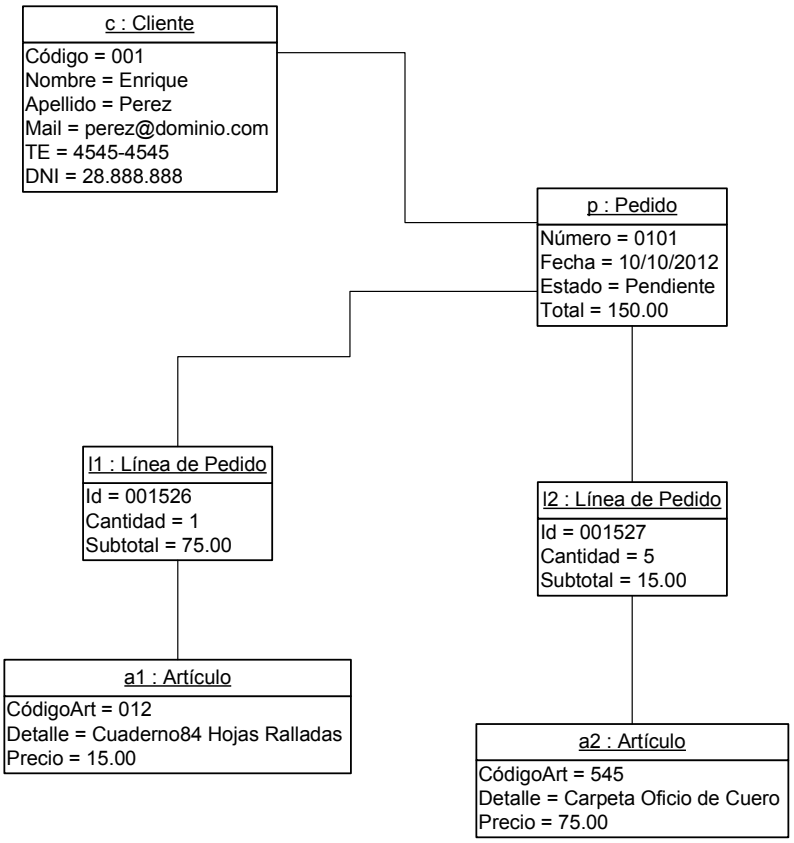


Figura 24: Diagrama de Objetos

➢ Diagrama de componentes

Este diagrama muestra las relaciones entre los distintos componentes del sistema. Representan la estructura del software.

Elementos:

- ➢ Componentes
- ➢ Interfaces
- ➢ Relaciones

Aplicación:

- ➢ Modelar la vista de implementación del sistema.

Ejemplos:

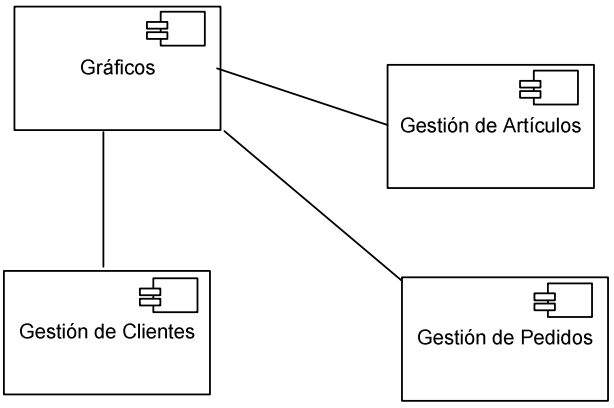


Figura 25: Diagrama de Componentes

➢ Diagrama de estructura compuesta

Este diagrama se encarga de mostrar el comportamiento interno de una clase o una colaboración. Son similares a los diagramas de componentes.

Ejemplo:

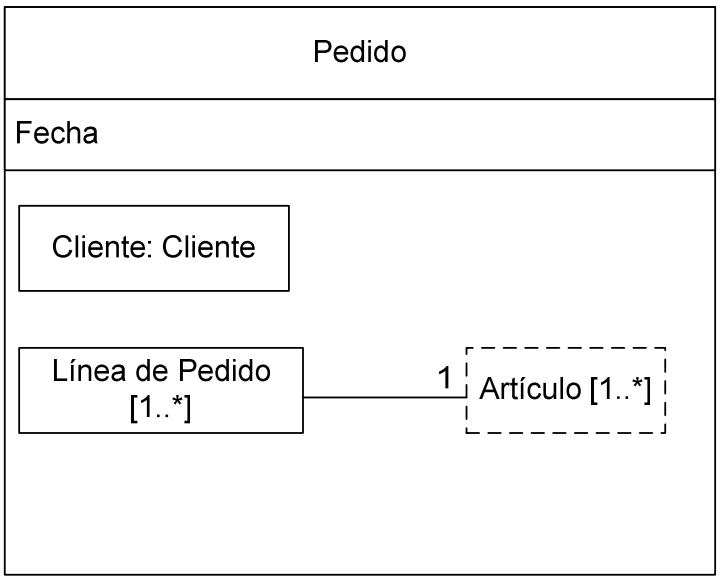


Figura 26: Clase Estructurada.

➢ Diagrama de casos de uso

Este diagrama muestra la relación entre casos de uso y actores. Ayuda a definir los límites y el comportamiento del sistema.

Elementos:

- Actor.
- Casos de uso.
- Relaciones entre actor y casos de uso: Asociación, Generalización.
- Relaciones entre casos de uso: include, extend

Aplicación:

- Modelar el comportamiento del sistema.
- Modelar requisitos.
- Identificar el límite del sistema.

Ejemplo:

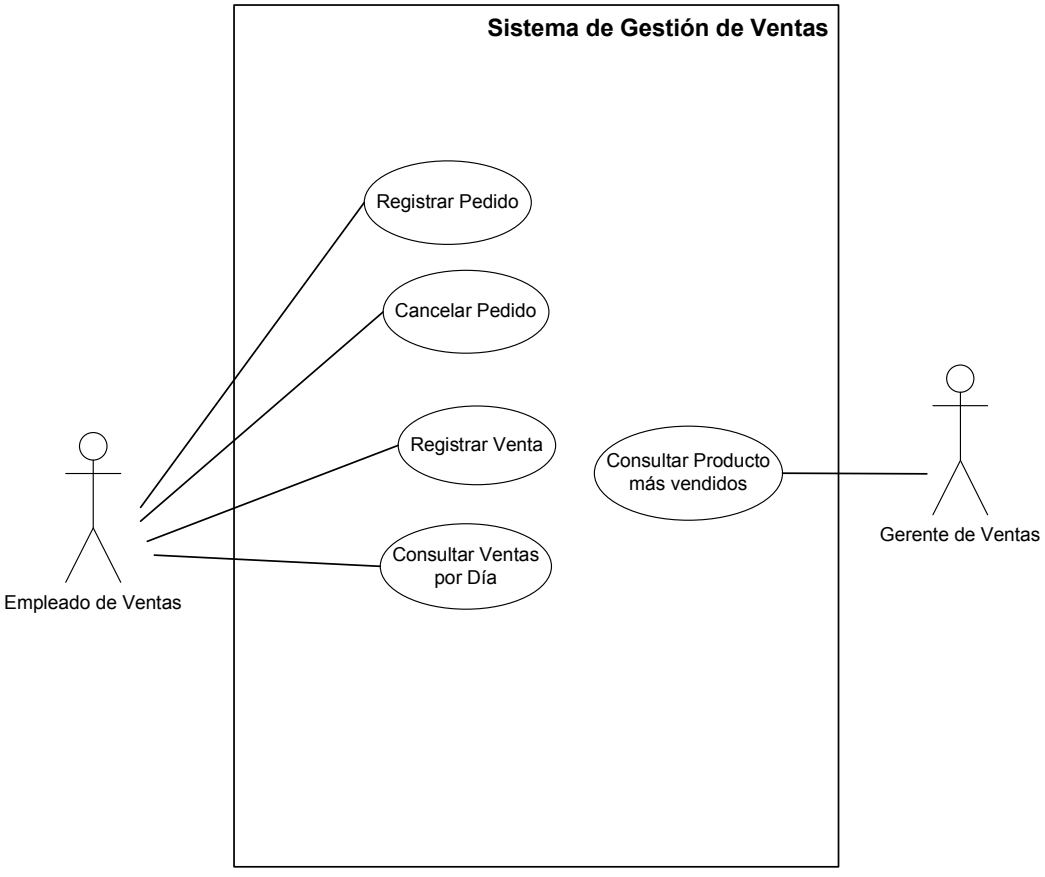


Figura 27: Diagrama de Casos de Uso.

➢ Diagrama de secuencia

Es un diagrama que muestra el flujo de mensajes entre objetos remarcando el orden cronológico.

Elementos:

- Objetos.
- Actores.
- Línea de Vida.
- Activación.
- Mensajes.

Aplicación:

- Definir el comportamiento de un subsistema, clase, caso de uso o escenario.
- Definir el comportamiento interno de una colaboración.

Ejemplo:

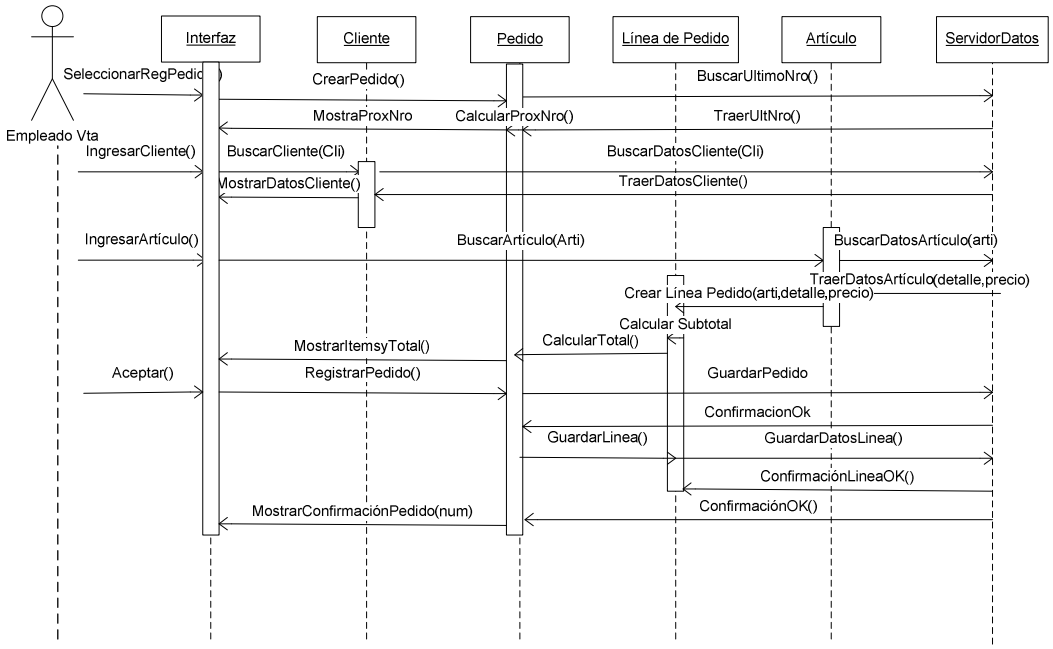


Figura 28: Diagrama de Secuencia.

Diagrama de comunicación

Este diagrama es similar al diagrama de secuencia, muestra el flujo de mensajes entre objetos, pero en este caso su visualización no destaca el orden cronológico sino la organización estructural.

El diagrama de secuencia y el diagrama de comunicación son **isomorfos**, partiendo del diagrama de secuencia se puede obtener el diagrama de comunicación y viceversa.

Elementos:

- Objetos
- Mensajes
- Relaciones

Aplicación:

- Ver la organización de objetos involucrados en el comportamiento de un subsistema, clase, caso de uso, escenario o colaboración.

Ejemplo:

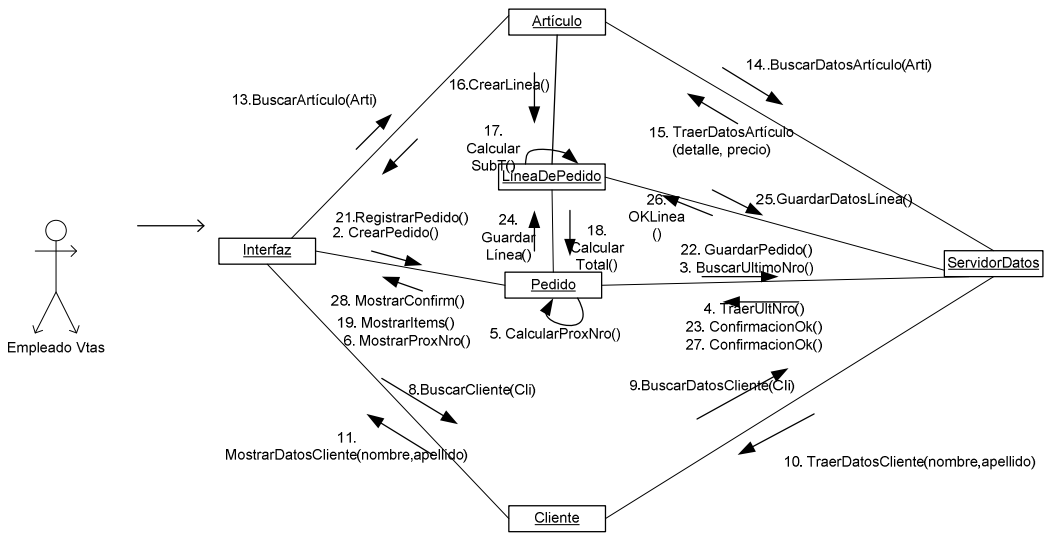


Figura 29: Diagrama de Comunicación.

➢ Diagrama de estados

El diagrama de estados es uno de los diagramas de UML que permite modelar los aspectos dinámicos del sistema.
Un diagrama de estados es una **máquina de estados** donde se muestra el flujo de control entre los estados por los que pasa un objeto durante su vida útil. **(Para mayor detalle ver apunte Maquinas de Estado en UML)**

➢ Diagrama de actividades

Un diagrama de actividades permite describir un comportamiento paso a paso de un proceso determinado, destacando el flujo de control entre las actividades.
Una actividad es una acción o bien un conjunto de operaciones que pueden ser ejecutadas en forma automatizada o manualmente.
En este diagrama se tiene en cuenta, el orden de ejecución de las actividades y las actividades que puede realizarse en forma paralela.
(Para mayor detalle ver apunte Maquinas de Estado en UML)

➢ Diagrama de despliegue

Este diagrama muestra la relación entre los nodos y componentes. Cada nodo en su interior puede tener uno o más componentes. Este diagrama muestra la arquitectura física del sistema.

Elementos:

- Nodos.
- Componentes.
- Interfaces.
- Relaciones.

Aplicación:

- Representar la estructura física final del sistema.

Ejemplo:

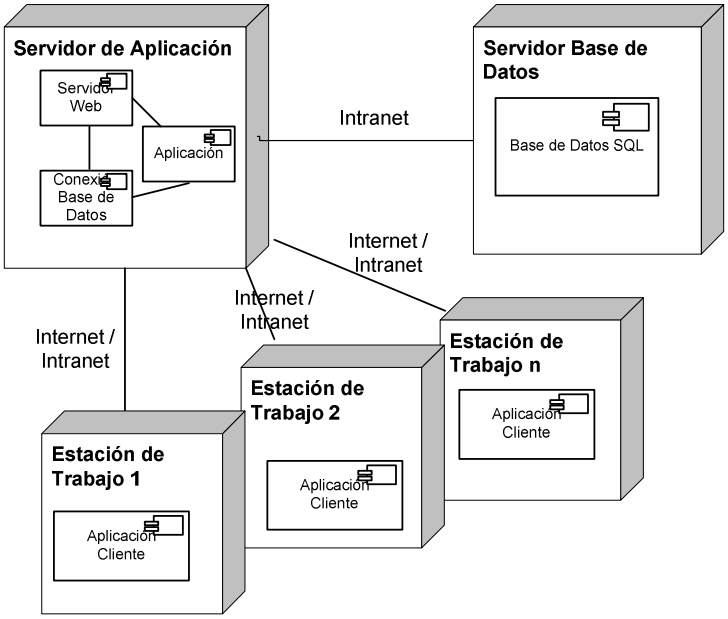


Figura 30: Diagrama de Despliegue.

➢ Diagrama de paquetes

Muestra la organización del software y sus dependencias. Permite simplificar los diagramas agrupando sus elementos en unidades menores.

Elementos:

- Paquetes:
- Relaciones: Asociación –Dependencia.

Aplicación:

- Mostrar la organización de los modelos.
- Mostrar la organización del software.

Ejemplo:

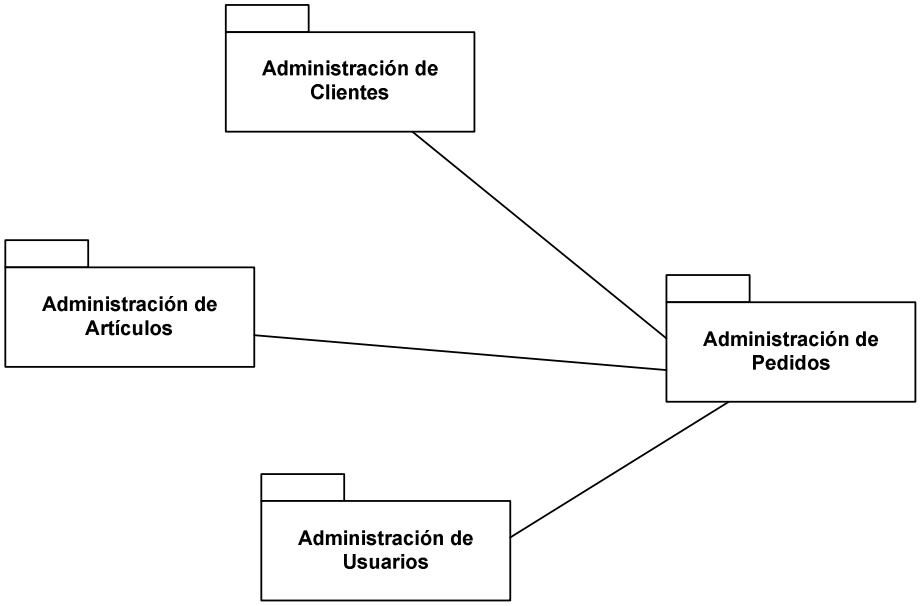


Figura 31: Diagrama de Paquetes.

En ocasiones se puede necesitar detallar el contenido del paquete. En estos casos se agrega la descripción en el interior del mismo y el nombre se coloca en la pestaña.

Ejemplo:

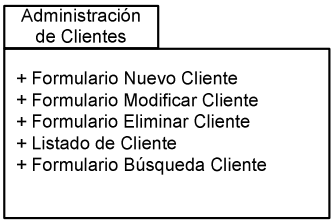


Figura 32: Paquete con detalle de su contenido.

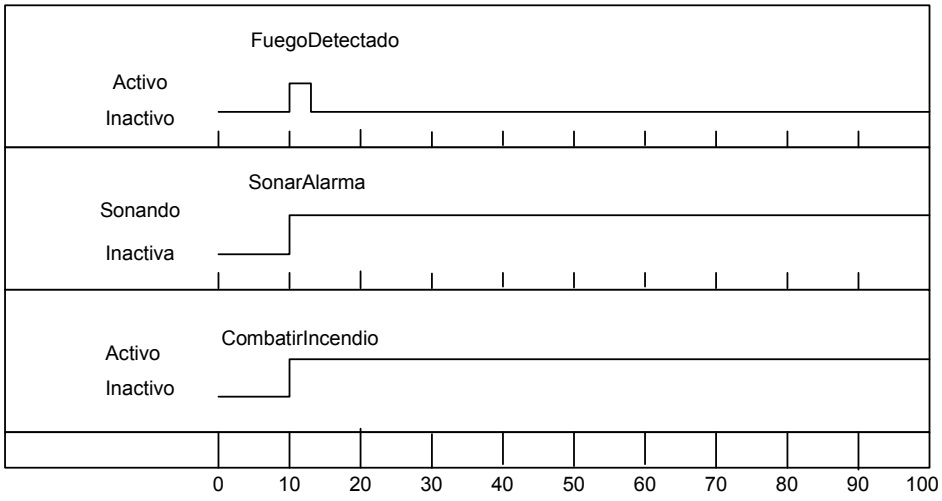
➢ Diagrama de tiempos

Muestra los tiempos reales en interacciones entre objetos o partes del sistema.

Aplicación:

- ➢ Analizar el comportamiento del sistema en un tiempo dado

Ejemplo:



➢ Diagrama de visión global de interacciones

Muestra el flujo de control entre interacciones. Es un diagrama que tiene parte del diagrama de secuencia y parte de diagrama de actividades.

Ejemplo:

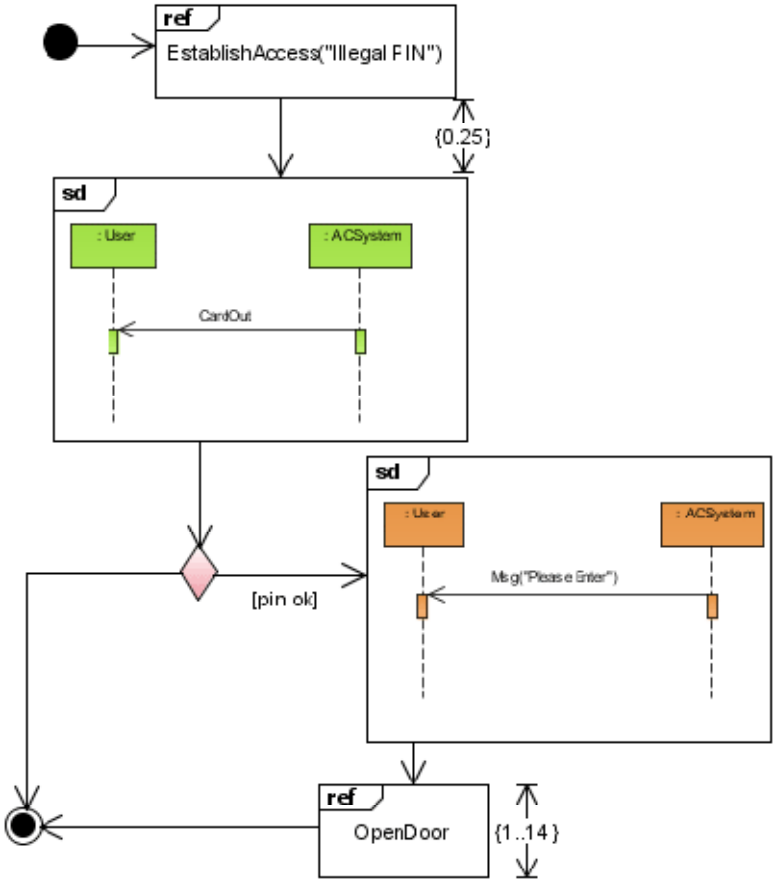


Figura 33: Diagrama Visión Global de Interacciones. (López & Ruiz, 2013)

6. Arquitectura – Vistas

La arquitectura es la forma de organizar las distintas partes de software, generalmente se define a principio del diseño y se utiliza como guía durante toda la construcción del sistema.

El desarrollo de un sistema se puede ver desde distintos puntos de vistas. Con UML se pueden expresar cada una de estas vistas:

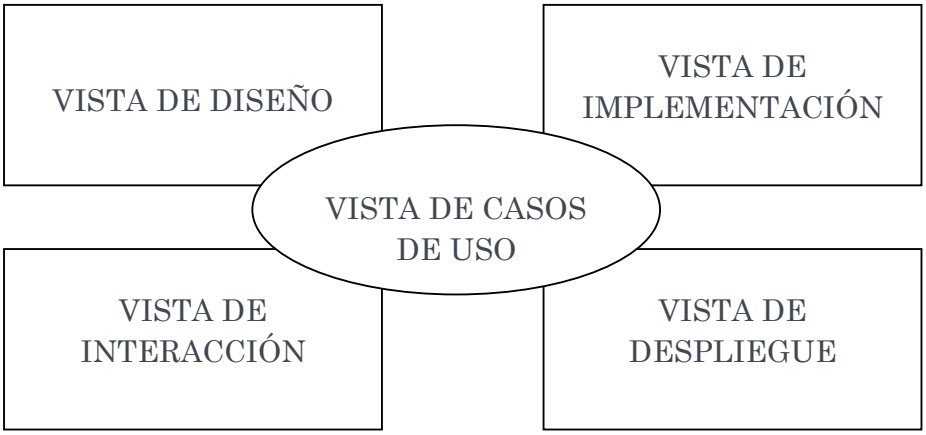


Figura 34: Modelo de la arquitectura de un sistema. (Booch, Rumbaugh, & Jacobson, 2006)

Vista de Casos de Uso

En esta vista se incluyen los casos de uso que describen la funcionalidad y el comportamiento del sistema tal como lo ven los usuarios, analistas y los encargados de las pruebas del sistema. Se utilizaran los diagramas de casos de uso, de estados, de actividades y diagramas de interacción.

Vista de Diseño

En esta vista se incluyen las clases, interfaces y colaboraciones que intervienen en el desarrollo. Para su definición se utilizaran los diagramas de clases, objetos, estructura interna, estados, actividades y diagramas de interacción.

Vista de Interacción

En esta vista se incluye todo lo referido al flujo de control entre elementos, teniendo en cuenta sincronización y concurrencia. Se utilizaran los diagramas de clases, objetos, estructura interna, estados, actividades y diagramas de interacción. En este caso se deberá hacer hincapié en las clases activas.

Vista de Implementación

En esta vista se incluyen los artefactos necesarios para poner en funcionamiento el sistema software. Se muestra la relación entre las clases, los componentes y los archivos físicos. Se utilizarán los diagramas de secuencia, actividades y estados.

Vista de Despliegue

En esta vista se incluyen los nodos, componentes de red y hardware donde se instalará el sistema físico y su distribución. Se utilizarán diagramas de despliegue, estado, actividades y diagramas de interacción.

7. MECANISMOS

Puede ser necesario agregar información adicional a los modelos previstos por UML o bien modificar alguna regla o diagrama en particular. Para estas necesidades UML presenta cuatro mecanismos comunes:

- **Especificaciones:** Permite agregar de forma textual un detalle adicional a los bloques de construcción confeccionados donde se defina el significado.

- **Adornos:** Agregan información adicional a los elementos básicos de UML. Los adornos pueden ser gráficos o textuales.

  Ejemplos:

  Nota: cualquier comentario que puede agregarse en cualquier modelo o diagrama. Puede definir un requisito, una restricción, o simplemente una observación.

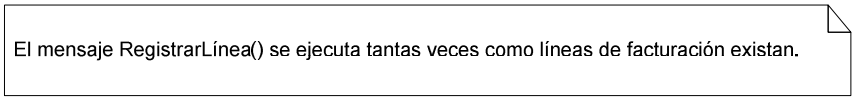


Figura 35: Adorno 1: Nota

Una asociación se puede adornar mostrando la multiplicidad o el nombre de la relación.

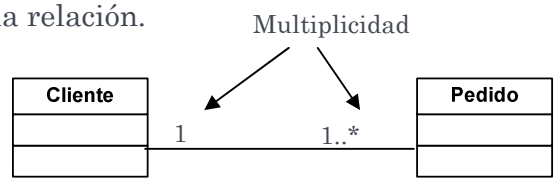


Figura 36: Adorno2 :multiplicidad

- **Divisiones comunes.**
  Algunos de los elementos que UML define pueden separarse o "dividirse" en dos elementos similares. Por ejemplo: CLASE / OBJETO. Se utiliza la misma nomenclatura pero se distingue al objeto (instancia de una clase) subrayando su nombre. Algo similar ocurre con la INTERFAZ / IMPLEMENTACIÓN y con CASOS DE USO / COLABORACIÓN.

- **Extensibilidad:** Permite agrandar el vocabulario de UML generando nuevos bloques de construcción. Existen tres tipos básicos:

  Estereotipos: define un nuevo bloque de construcción necesario para definir el problema.

  Se pueden identificar con un nombre entre "<< >>" o simplemente con un nuevo ícono.

  Ejemplos:

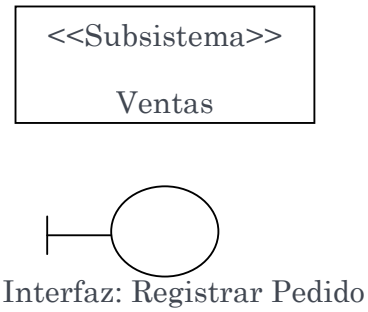


Figura 37: Estereotipos.

Valores etiquetados: definen nuevas propiedades a los estereotipos creados. Los valores etiquetados se agregan en notas.

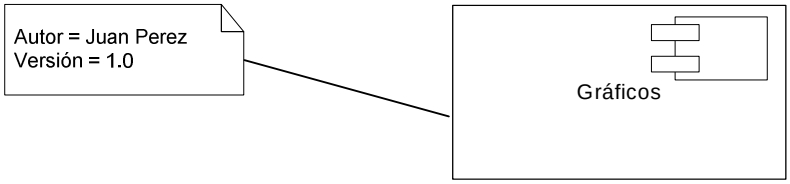


Figura 38: Valores etiquetados.

Restricciones: Permiten agregar o modificar las reglas existentes de UML. Para definir una restricción se puede utilizar el lenguaje natural o bien si se desea confeccionar un restricción utilizando un lenguaje más formal se puede utilizar OCL (Lenguaje de Restricciones de Objetos).

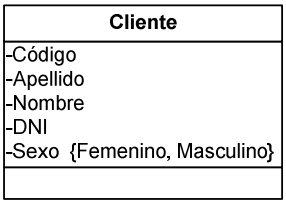


Figura 39: Restricciones.

Las restricciones se incorporan en el modelo entre {}.

BIBLIOGRAFÍA

Booch, G., Rumbaugh, J., & Jacobson, I. (2006). *El Lenguaje Unificado de Modelado Guía del Usuario.* Addison - Wesley.

Kendall, K. E., & Kendall, J. E. (2011). *Análisis y Diseño de Sistemas.* México: Pearson.

López, P., & Ruiz, F. (2013). *Universidad Cantabria.* Obtenido de http://ocw.unican.es/ensenanzas-tecnicas/ingenieria-del-software-i/materiales-de-clase-1/is1-t02-trans.pdf

Orozco, S., & Macías, C. (s.f.). *Revista SG Software Guru.* Obtenido de http://sg.com.mx/content/view/568

Pfleeger, S. L. (2002). *Ingeniería de Software. Teoría y Práctica.* Buenos aires: Pertince Hall.