Generalidades

0.1 Que es diseño de sistemas?

El diseño de sistemas es la especificación detallada de todos los componentes de un sistema de aquella solución seleccionada como mejor alternativa a desarrollar en la propuesta seleccionada durante el análisis de sistemas, aplicando técnicas y principios con el propósito de definir un dispositivo, proceso o sistema que tiene la característica de satisfacer la necesidad planteada desde análisis conteniendo suficientes detalles para permitir su interpretación y realización física.

El propósito del diseño es extender la arquitectura de análisis, el cual es un modelo conceptual y lógico del sistema, y se aplica el diseño para definir todo lo necesario para alcanzar el código final. Con el diseño se definen las decisiones estratégicas sobre cómo organizar la funcionalidad del sistema en torno al ambiente de implementación.

0.1.1Fases del diseño

Diseño Lógico: También conocido como diseño de arquitectura, diseño general o de alto nivel, se desarrolla una división modular del producto, se analizan con cuidado las especificaciones y se crea una estructura que tenga la funcionalidad deseada, la salida de esta actividad es una lista de módulos y una descripción de cómo se interconectaran, en esta fase se determinan la clases que son los módulos al igual que la división modular que se ha realizado durante el flujo de trabajo del análisis.

Diseño Físico: También conocido como diseño detallado, modular o de bajo nivel durante el cual se diseña cada modulo (clase) en detalle por ejemplo: algoritmos específicos y se eligen estructuras de datos, se ignora el hecho de que los módulos (clases) tengan que ser interconectados para formar un producto completo.

0.2 Que es programación Orientada a objetos?

La programación Orientada a objetos (POO) es una forma especial de programar, más cercana a como expresaríamos las cosas en la vida real. Define una estructura de más alto nivel llamada *objeto* el cual es la entidad básica del modelo de cualquier sistema.

0.2.1. Diferencias entre Programación Estructurada y Orientada a Objetos

1.- Permitir al programador organice su programa de acuerdo con la abstracción de más alto nivel, siendo estas más cercanas a la manera de pensar de la gente. Ejemplo: Cuentas de banco. Reservaciones de vuelos, Punto de Venta, Control de inventarios, etc. DIFERENCIA CON PROGRAMACION TRADICIONAL: El programador organiza el programa de acuerdo con la arquitectura de la computadora debe pensar como la maquina.

2.- Los datos globales desaparecen, siendo estos junto con las funciones parte interna de los objetos. Por lo tanto cualquier cambio en la estructura de algo de los datos solo debe afectar las funciones definidas en este mismo objeto y no el de los demás. DIFERENCIA CON PROGRAMACION TRADICIONAL: Las funciones programadas están

separadas esto produce que el cambio en la estructura de alguno de los datos pudieran llegar a requerir la modificación de todas las funciones del programa.

0.2.2. Características de los Lenguajes Orientados a Objetos:

1.- Encapsulamiento: Mecanismo Básico para ocultar los detalles internos del objeto, permite distinguir entre la interface del objeto y su implementación es decir los aspectos del objeto conocidos *externamente*. La interface de un objeto corresponde a las interfaces de sus funciones, mientras que la implementación corresponde a los datos y código o implementación de funciones.

2.- Clasificación: Permite organizar objetos de acuerdo a estructuras comunes. Los objetos con datos y funciones similares se clasifican como si fueran de una misma clase.

3.- Generalización: Permite organizar clases con estructura y comportamiento similar para ser reutilizables en la definición de nuevas clases, siendo estas mas especializadas que las anteriores. Las clases anteriores mas especializadas son subclases y las generales son superclases.

4.- Polimorfismo: Es definir múltiples funciones con nombres e interfaces similares solo que en distintas clases. El polimorfismo es útil para extender la funcionalidad del sistema al definir nuevas clases aun desconocidas al momento de especificarlo. Aquí es donde se define un estándar de interfaces que deben seguir todas las clases ya sean las existentes o nuevas.

0.3 Características que mejoran la calidad en el diseño de sistemas:

0.3.1 Abstracción: Consiste en elevar el nivel de las representaciones necesarias para un sistema de software de manera que se reduzcan los detalles. Cuanto más alto sea el nivel de la representación menor será el número de elementos necesarios para representar un sistema completo y más fácil será el manejo de la complejidad.

0.3.2 Modularidad: La cual permite dividir un sistema en componentes separados. Facilitando su operación y mantenimiento la modularidad se basa en objetos, un nivel más alto que los datos y funciones tradicionales.

0.3.3 Extensibilidad: Es la facilidad de modificar un sistema durante el transcurso de su vida. Los sistemas compuestos por múltiples módulos facilitan esta extensibilidad dado que los cambios en el sistema se pueden reducir a cambios en módulos particulares y no en todo el sistema. Los cambios se dan en dos niveles: modificación interna afectan al directamente al objeto, modificación externa afecta a los objetos externos en el resto del sistema.

0.3.4 Reutilización: Es aprovechar componentes o bibliotecas ya desarrolladas, logrando una mayor estandarización y simplificaciones en las aplicaciones, principalmente aplicaciones numéricas y estadísticas mediante procedimientos y funciones reutilizables. Se tienen que construir componentes genéricos, sencillos con interfaces bien definidas y que puedan utilizarse en varias áreas de aplicación. Al agrupar objetos similares se puede lograr la reutilización de componentes de más alto nivel.

También se pueden aprovechar objetos con estructuras de datos y funciones similares. Definiendo una sola vez los aspectos comunes y especializándolos en objetos adicionales. A este procedimiento se le conoce como *Herencia* o también como framework donde una aplicación genérica de dominio particular se especializa según las necesidades de diferentes empresas.

0.4 Que es un buen sistema?

Es aquel que cumple las necesidades del usuario tiene que ser

1.- Útil y aprovechable: Un buen software hace la vida de los usuarios más fácil o mejor.

2.- Fiable: un buen software tiene pocos errores.

3.- Flexible: Se tiene que adaptar a las necesidades de los usuarios, políticas y procedimientos cambiantes a lo largo del tiempo, incluso mientras el software se está desarrollando, de manera que es importante poder realizar cambios inmediatamente. Debe ser tan flexible que sea fácil usar hasta contener controles para no cometer errores.

4.- Disponible: Tiene que poder ejecutarse en el hardware disponible, con sistema operativo disponible debe ser suficientemente portable, también debe estar disponible para el usuario que tomara decisiones útiles

5.- Seguro: Debe haber seguridad en todo el entorno del sistema tanto de acceso, como de obtención de datos y realizar controles internos para verificación de información.

1. UML (Lenguaje Unificado de Modelado)

1.1. Introducción a UML

1.1.1. Conceptos Básicos

El ***modelado*** describe los conceptos principales de la orientación de a objetos, así como las estructuras estáticas y sus relaciones.

Estructuras Estáticas: Son los Objetos y Clases que están compuestos de atributos y operaciones.

Relaciones: Corresponde a las ligas y asociaciones de los objetos y clases.

UML (Unified Modeling Languaje) es el lenguaje de ***modelado*** de sistemas de software más conocido y utilizado en la actualidad; aún cuando todavía no es un estándar oficial, está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

UML ofrece ***un estándar*** para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

UML es un ***lenguaje*** para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional), pero no especifica en sí mismo qué metodología o proceso usar.

1.1.2. UML como lenguaje de Especificación

Para este contexto especifica significa construir modelos que sean precisos que no sean ambiguos y deben ser completos. Las directrices de UML especifican todos los aspectos importantes del análisis, diseño e implementación las cuales tienen que estar desarrolladas y especificadas en un lenguaje de modelado que faciliten el desarrollo de sistemas.

1.1.3. UML como lenguaje de Construcción

UML no es un lenguaje de programación visual, pero es un lenguaje de modelado que puede estar directamente conectado a una gran variedad de lenguajes de programación. Esto significa que puedes desarrollar un esquema de un modelo en UML para un lenguaje de programación como puede ser Java, C++ o Visual Basic o también tablas en una base de datos relacional o una base de datos orientada a objetos. La mejor manera de expresar estos aspectos es gráficamente en UML. Este esquema permite el desarrollo de ingeniería de software. La generación de esquemas UML hacen posible la generación de código en un lenguaje de programación, también se pude reconstruir un modelo a partir de un programa ya implementado. Esto quiere decir que puede ser reversible. Este lenguaje de modelado es suficientemente expresivo y carece de ambigüedades permitiendo una ejecución directa de modelo de datos, la simulación de sistema y la implementación de sistemas.

1.1.4. UML como lenguaje de Documentación

UML tiene el propósito de generar directrices para la documentación de sistemas así como su arquitectura y todos sus detalles. El UML también provee un lenguaje de expresión de requerimientos o como modelo de pruebas de un sistema. UML provee un lenguaje de modelado de actividades planeación y administración de proyectos. Se pueden documentar requerimientos, arquitectura, diseño, código fuente, Planeación de proyectos, pruebas y prototipos.

1.2. Modelo Conceptual

1.2.1. Tipos de Elementos

ELEMENTOS BASICOS:

OBJETO:

Son las entidades básicas de un modelo, son simples cosas que pueden ser conceptos como un sustantivo, también es cualquier cosa que incorpore una estructura y un comportamiento o acción. Debe tener una identidad coherente para que se le pueda asignar un nombre lógico y conciso. La existencia del objeto depende del contexto del problema. Lo que puede ser un objeto apropiado a una aplicación. Los objetos se definen según el contexto de la aplicación. Los objetos son entidades que existen en forma independiente. Es necesario distinguir entre los objetos, los cuales contienen características o propiedades. Deben tener nombre en singular y no en plural, parte de una cosa puede considerarse objeto.

El objeto integra una estructura de datos (atributos) y un comportamiento (operaciones)

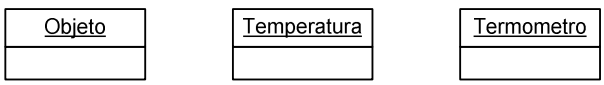
Ejemplo:

Temperatura es un objeto abstracto con propiedades como el valor de la temperatura y el tipo de escala en que se mide (Celsius o Fahrenheit).

El termómetro es el objeto concreto que mide la temperatura esta sería una propiedad del termómetro.

Diagrama:

Es una caja rectangular que contiene el nombre del objeto subrayado, el cual sirve para identificar al objeto



Identidad de los Objetos:

Los objetos se distinguen por su propia existencia. Su identidad aunque internamente los valores para todos sus datos sean iguales, debido a que todos los objetos se consideran diferentes. Los objetos tienen un nombre que puede no ser único.

Los objetos necesitan un identificador interno único cuando se instrumentan o implementan en un sistema computacional con el fin de permitir el acceso y distinguir entre los objetos. Estos identificadores no deben incluirse como una propiedad del objeto, ya que solo son importantes en el momento de la implantación.

Ejemplo:

Todas las personas se distinguen internamente dentro de una computadora mediante identificadores Persona1, Persona2, Persona3. Por otro lado el número de seguro social de la persona es un identificador externo valido, ya que existe fuera de la implementación en una computadora.

CLASE:

Una Clase describe un grupo de objetos con estructura y comportamiento común.

Un Tipo se define por las manipulaciones que se le pueden dar a un objeto dentro de un lenguaje y clase involucra una estructura que puede corresponder a una implementación particular de un tipo.

Atributos: Son las estructuras o propiedades de la clase

Comportamiento: Son las operaciones de esa clase de objetos.

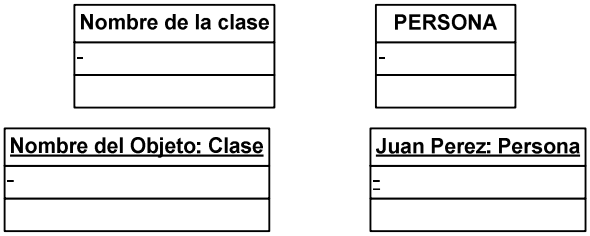
Una clase se considera un molde a partir del cual se crean múltiples objetos. Al definir múltiples objetos en clases se logra una abstracción del problema a partir de casos específicos se generalizan definiciones comunes como nombres de la clase, atributos y operaciones.

Ejemplo:

Juan Pérez y María López se consideran miembros de la clase PERSONA donde todas estas personas tienen una edad y un nombre.

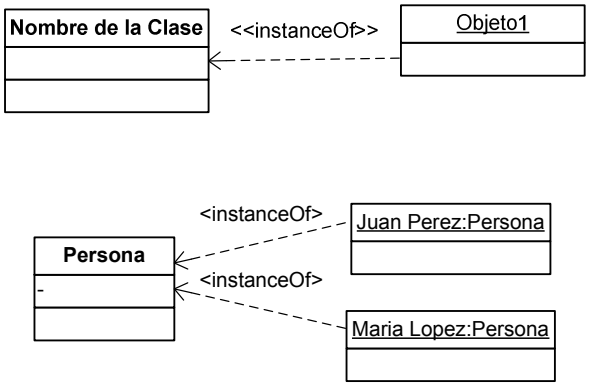
Diagrama:

Es una Caja Rectangular que contiene el nombre de la clase, la notación general para el objeto se extiende mediante el nombre de la clase subrayado seguido del nombre del objeto



Instanciación:

El proceso de crear objetos pertenecientes a una clase se conoce como instanciación donde los objetos son las instancias de las clases. El objeto es la instancia de la clase a la que pertenece. Se utiliza una flecha punteada para mostrar los objetos como instancias de las clases

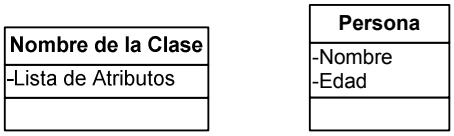


ATRIBUTOS:

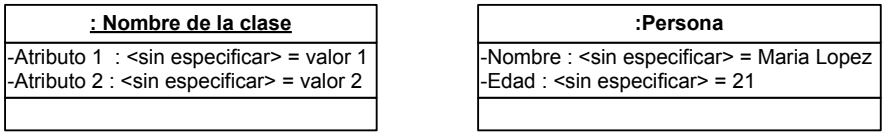
Definen la estructura de una clase y sus correspondientes objetos, define el valor de un dato para todos los objetos pertenecientes de una clase.

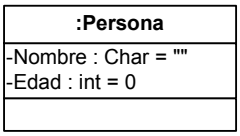
Se debe definir un valor para cada atributo de una clase. Los valores pueden ser iguales o distintos en los diferentes objetos. No se puede dar un valor en un objeto si no existe un atributo correspondiente en la clase.

Dentro de una clase, los nombres de los atributos deben ser únicos, aunque puede aparecer el mismo nombre de atributo en diferentes clases.

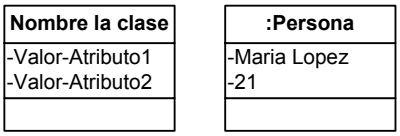


Notación General





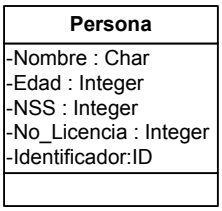
Notación extendida

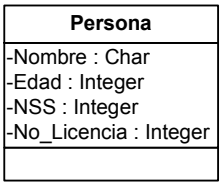


Notación Compacta

IDENTIFICADORES:

En el momento de incluir atributos en la descripción de una clase se deben distinguir los atributos los cuales reflejan las características de los objetos en la realidad y los identificadores que se utilizan exclusivamente por razones de implementación. Estos identificadores internos del sistema no deben ser incluidos como atributos

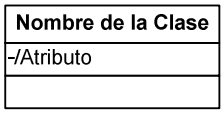


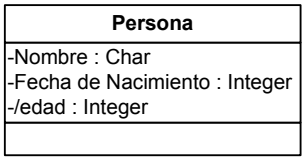


Incorrecto Correcto

ATRIBUTOS DERIVADOS:

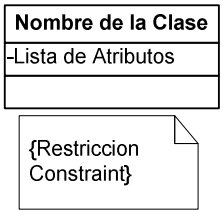
Son atributos independientes dentro del objeto. Son atributos que dependen de otros atributos, que estos a su vez pueden ser básicos o derivados.

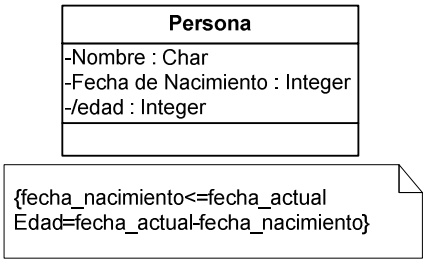




RESTRICCIONES DE LOS ATRIBUTOS:

Los valores de los atributos de una clase pueden restringirse. La notación para una restricción (constraint) es incluir por debajo de la clase y entre corchetes.



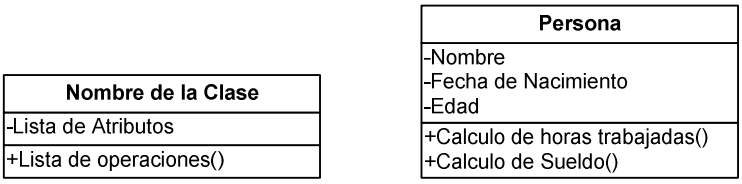


OPERACIONES:

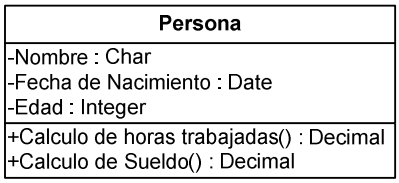
Las operaciones son funciones o transformaciones que se aplican a todos los objetos de una clase particular. La operación puede ser una acción ejecutada por el objeto o sobre el objeto

Las operaciones deben ser únicas dentro de una misma clase, aunque no necesariamente para diferentes clases.

Las operaciones pueden tener argumentos, es decir una lista de parámetros cada uno con un tipo y pueden también devolver resultados, Cada uno con un tipo. Las operaciones se incorporan en la tercera sección de la clase



Notación General



Notación extendida

CONSULTAS

Son las operaciones que no tienen efectos secundarios, las cuales no cambian los valores de los atributos. Es una operación que no modifica el objeto. Devuelven valores de atributos básicos o derivados y pueden o no tener argumentos.

La notación para las operaciones de consulta es la misma que para las operaciones, solo difieren en su comportamiento. Por lo general estas consultas no se incluyen en forma explicita en el modelo de objetos, sino que se agregan durante la etapa de diseño, ya que son operaciones bastante básicas.

ACCESOS

Son utilizadas para leer o escribir los atributos del acceso esta hecho para leer solamente, sin afectar los valores del atributo, entonces se considera también una operación de consulta. Se utiliza la notación de punto para indicar, dentro de la operación de acceso, el acceso a un atributo.

Ejemplo: Para ingresa el nombre de una persona se utiliza persona.nombre

METODOS

Se utiliza para distinguir la operación como un concepto de alto nivel de la propia implementación de la operación, algo correspondiente a un concepto de más bajo nivel.

Ejemplo: La operación imprimir de la clase archivo se implementa mediante el método imprimir, que contiene argumentos llamado dispositivo que a su vez tiene el código para la implementación de la operación imprimir.

POLIMORFISMO

Es una misma operación que toma diferentes formas y esta implementada en diferentes clases en forma distinta:

Ejemplo: Operación Imprimir para la clase archivos pueden existir varios métodos para implementar la operación imprimir. Estos métodos corresponden a la misma operación imprimir pero se implementara en diferentes formas.

PARAMETRIZACION

Consiste en que una operación esa definida por el número y tipo de su argumento de tipo dispositivo.

Ejemplo: La parametrización de la operación imprimir de la clase archivo esta definida por su argumento de tipo dispositivo.

Elementos estructurales o conceptuales

Clase: Es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Una clase implementa una o más interfaces.

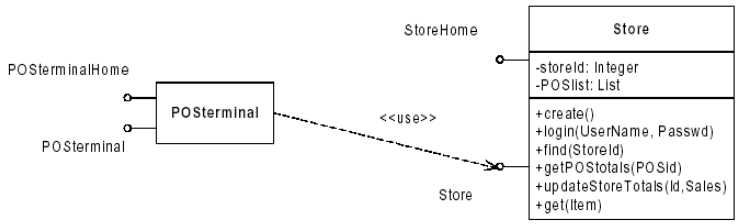
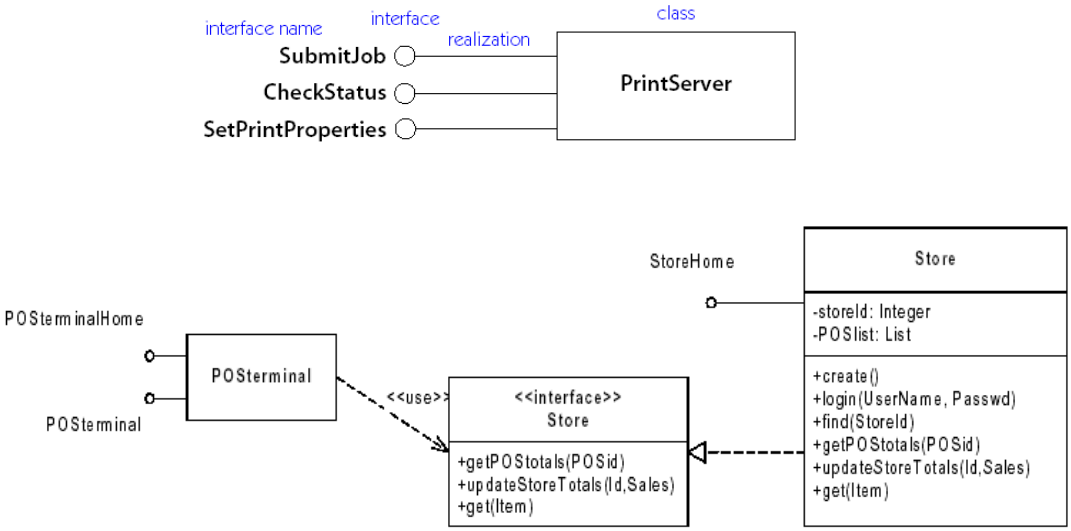


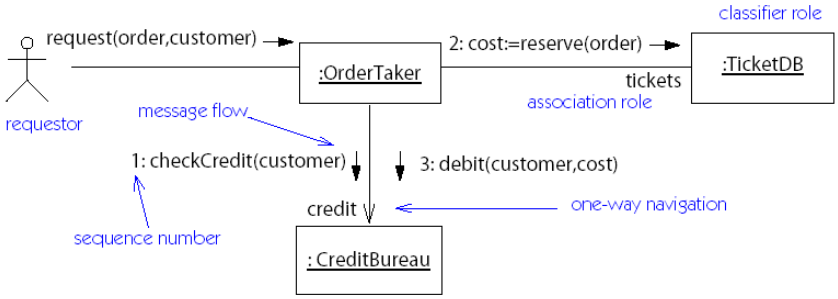
Figura: Representación de una clase

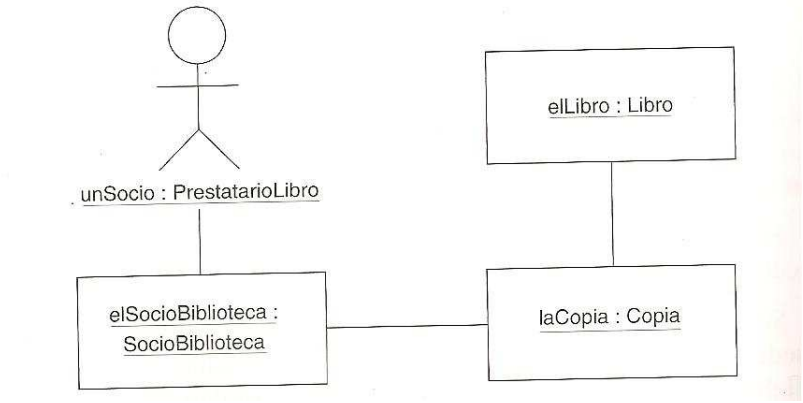
Interfaz: Es una colección de operaciones que especifican un servicio de una clase o componente. Por lo tanto, una interfaz describe el comportamiento visible externamente de ese elemento. Una interfaz puede representar el comportamiento completo de una clase o componente o sólo una parte de este comportamiento. Una interfaz define un conjunto de especificaciones de operaciones, pero nunca sus implementaciones. Una interfaz raramente se encuentra asilada, más bien, suele estar conectada a la clase o componente que la realiza.



Representación de una interfaz

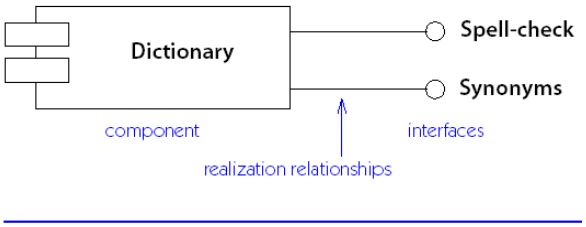
Colaboración: Define una interacción y es una sociedad de roles y otros elementos que colaboran para proporcionar un comportamiento cooperativo mayor que la suma de los comportamientos de sus elementos. Por lo tanto las colaboraciones tienen dimensión tanto estructural como de comportamiento. Una clase dada puede participar en varias colaboraciones.

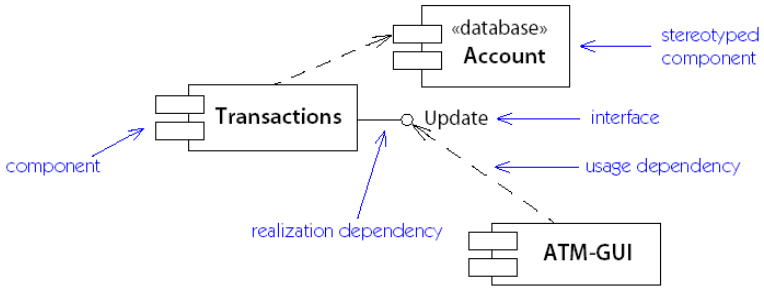




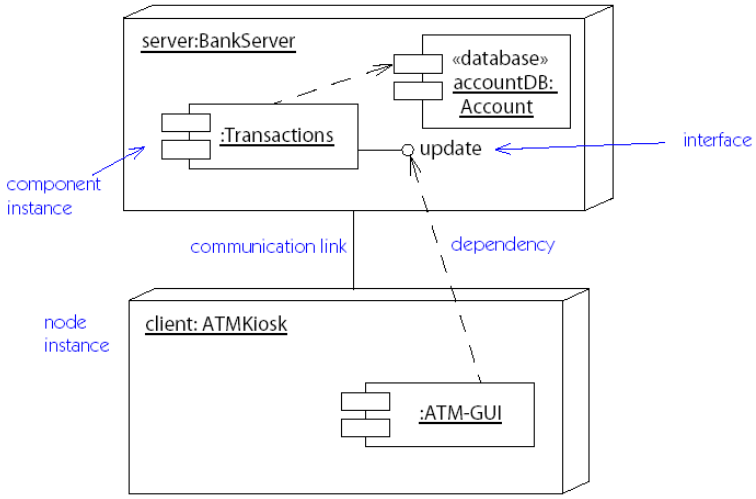
Elementos Físicos:

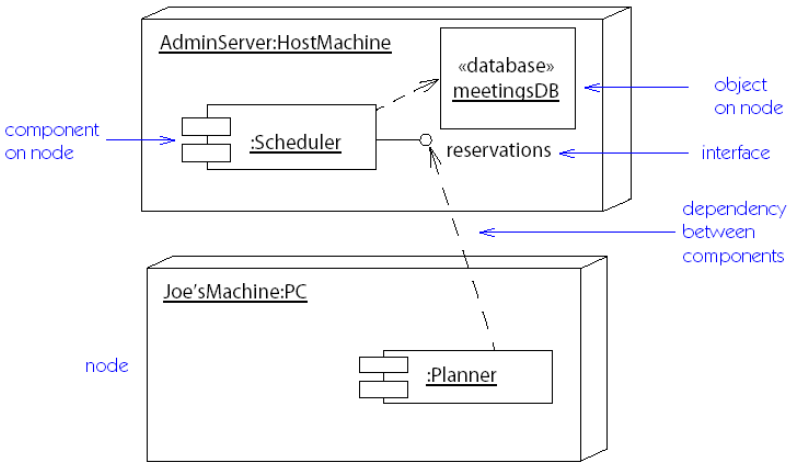
Componente: Es una parte física de un sistema que ofrece un conjunto de interfaces y proporciona la implementación de dicho conjunto. En un sistema, se pueden encontrar diferentes tipos de componentes de despliegue, tales como componentes Java Beans, así como componentes que sean artefactos del proceso de desarrollo, tales como archivos de código fuente. Un componente representa típicamente el empaquetamiento físico de diferentes elementos lógico, como clases interfaces y colaboraciones.





Nodo: Es un elemento físico que existe en tiempo de ejecución, representando un recurso computacional que, por lo general, dispone de algo de memoria y con frecuencia capacidad de procesamiento. Un conjunto de componentes puede residir en un nodo y puede también migrar de un nodo a otro.





Caso de uso: Es una descripción de un conjunto de secuencias de acciones que un sistema ejecuta y que produce un resultado observable de interés para un actor particular. Un caso de uso se utiliza para estructurar los aspectos de comportamiento en un modelo. Un caso de uso es realizado por una colaboración.

Especifica la funcionalidad que el sistema ha de ofrecer desde la perspectiva de los usuarios y lo que el sistema ha de realizar para satisfacer las peticiones de estos usuarios.

Este modelo utiliza 3 elemento básicos:

Actores: Modela los diferentes papeles que los usuarios del sistema pueden representar. Es un tipo o categoría de usuarios; cuando un usuario realiza una tarea con el sistema, actúa como una ocurrencia de ese tipo de usuarios. También puede desempeñar los papeles de diferentes actores definiendo diferentes roles que un usuario puede desempeñar.
El conjunto de actores representa todo aquello que necesita intercambiar información con el sistema que se esta desarrollando.
La diferencia entre un actor y un usuario es que actor es como una clase, que se define por la descripción de su comportamiento. Un usuario puede desempeñar varios papeles, es decir, puede actuar como actores diferentes.

Casos de usos: Representa todo aquello que el usuario ha de poder realizar en el sistema. Secuencia de transacciones que se realizan en un dialogo con el sistema y que se encuentran relacionadas por su comportamiento. Cada caso de uso constituye una secuencia completa de mensajes especificando la secuencia de interacción que tienen lugar entre un actor y el sistema.

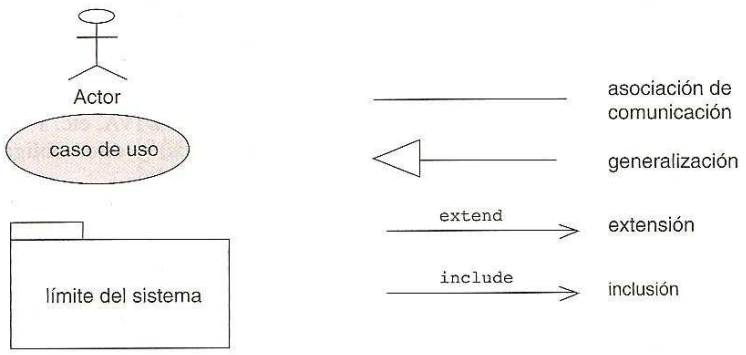
Las especificaciones de los casos de usos incluyen:

La descripción de la secuencia básica o comportamiento normal del caso de uso, la cual muestra la secuencia de mensajes más importantes facilitando la comprensión del caso uso que esta siendo descrito.

Las variantes sobre la secuencia básica del mensaje de un caso de uso, los posibles errores que pueden surgir durante su ejecución pueden ser descritos en diferentes secuencias alternativas.

Normalmente un caso de uso tiene una única secuencia básica, pero puede tener varias secuencias alternativas.

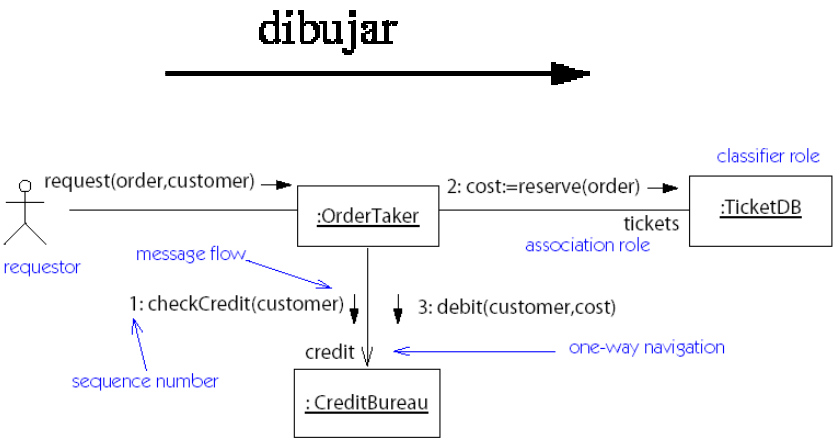
Relaciones: Para asociar los anteriores elementos. Estas se utilizan para identificar la comunicación existente entre los actores, los casos de uso y actores-casos de uso.



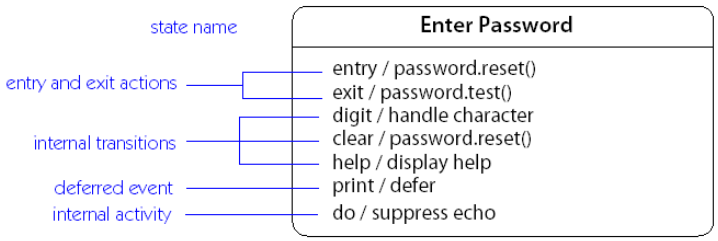
Elementos de comportamiento

Los elementos de comportamiento son las partes dinámicas de los modelos UML:

- *Interacción:* es un comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos, dentro de un contexto particular, para alcanzar un propósito específico. Una interacción involucra muchos otros elementos, incluyendo mensajes, secuencias de acción y enlaces.

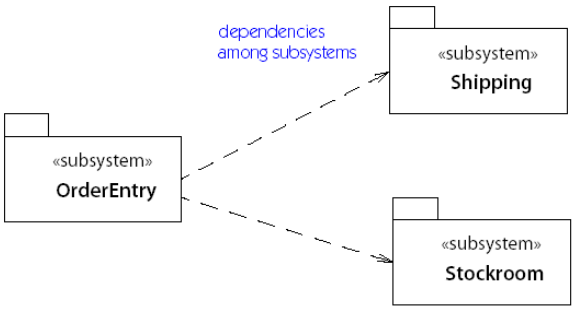


- *Máquina de estados*: es un comportamiento que especifica la secuencia de estados por las que pasa un objeto o una interacción durante su vida en respuesta a eventos, junto con sus reacciones a estos eventos. Una máquina de estados involucra a otros elementos, incluyendo estados, transiciones, eventos y actividades.



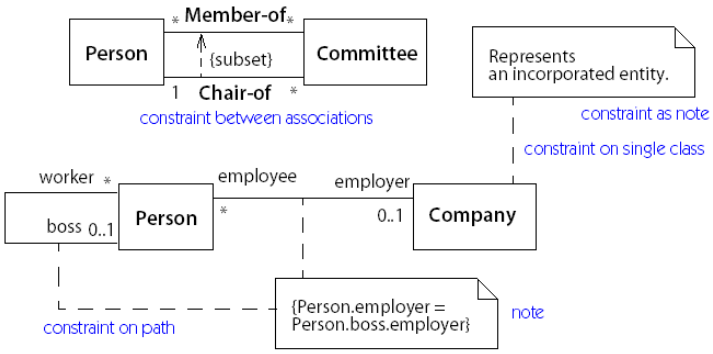
Elementos de agrupación

Son las partes organizativas de los modelos UML. Hay un elemento de agrupación principal, los paquetes. Un paquete es un mecanismo de propósito general para organizar elementos en grupos. Los elementos estructurales, los elementos de comportamiento, e incluso otros elementos de agrupación pueden incluirse en un paquete. Al contrario de los componentes (que existen en tiempo de ejecución), un paquete es puramente conceptual (sólo existe en tiempo de desarrollo).



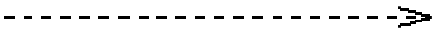
Elementos de anotación

Nota: Es simplemente un símbolo para mostrar restricciones y comentarios junto a un elemento o una colección de elementos. Las notas se utilizan para adornar los diagramas con restricciones o comentarios que se expresan mejor en texto informal o formal.

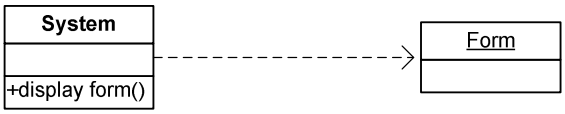


1.2.2. Tipos de Relaciones

- *Dependencia:* es una relación semántica entre dos elementos, en la cual un cambio a un elemento (el elemento independiente) puede afectar a la semántica del otro elemento (el dependiente). Las dependencias generalmente representan relaciones de uso que declara que un cambio en la especificación de un elemento puede afectar a otro elemento que la utiliza, pero no necesariamente a la inversa. Por ejemplo un refinamiento de un elemento de diseño a un elemento de implementación. La mayoría de las veces se utilizan en el contexto de las clases o paquetes, aunque también son habituales en la vinculación de notas.

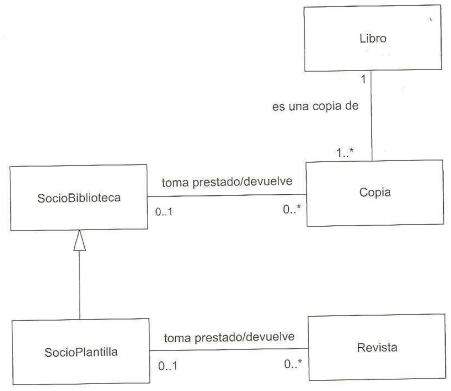


Representación de una Dependencia



- *Asociación:* es una relación estructural que describe un conjunto de enlaces, los cuales son conexiones entre objetos. La agregación es un tipo especial de asociación, que representa una relación estructural entre un todo y sus partes.

Representación de una Asociación



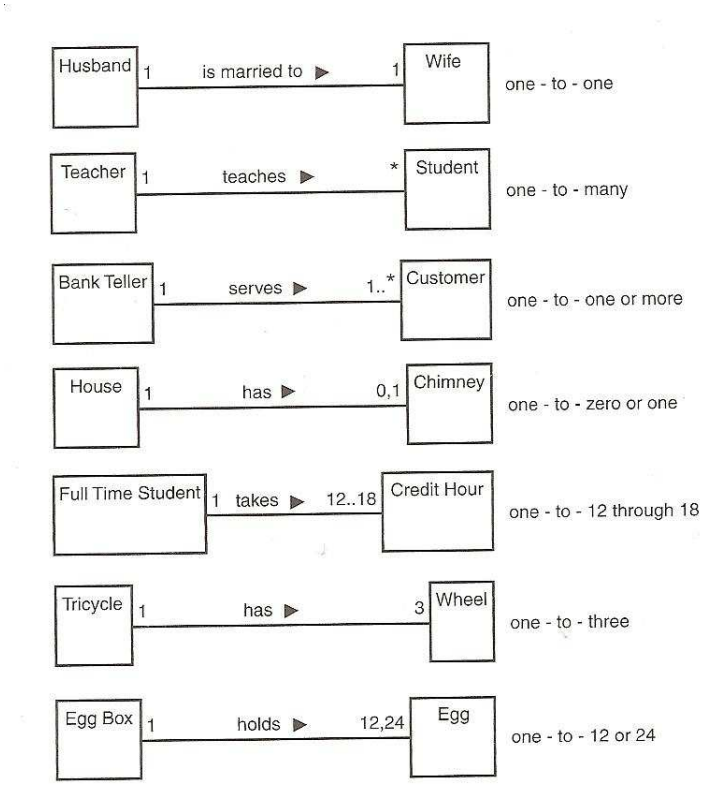
1.- Una copia es una copia de un libro

2.- Un socio de la biblioteca toma prestado/devuelve una copia.

3.- Un socio plantilla toma prestada/devuelve una copia.

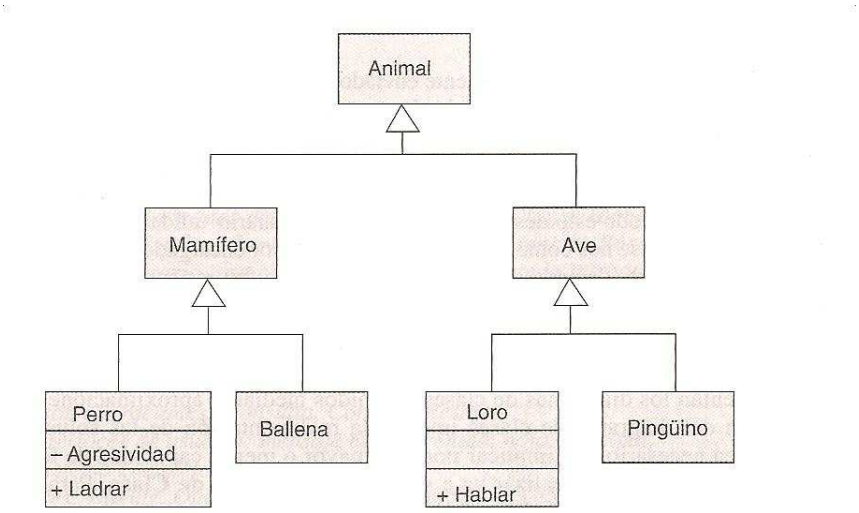
4.- Un socio plantilla toma prestada/devuelve una revista.

- *Multiplicidad:* Denota el numero de objetos de una clase que pueden estar relacionados un objeto de la clase asociada



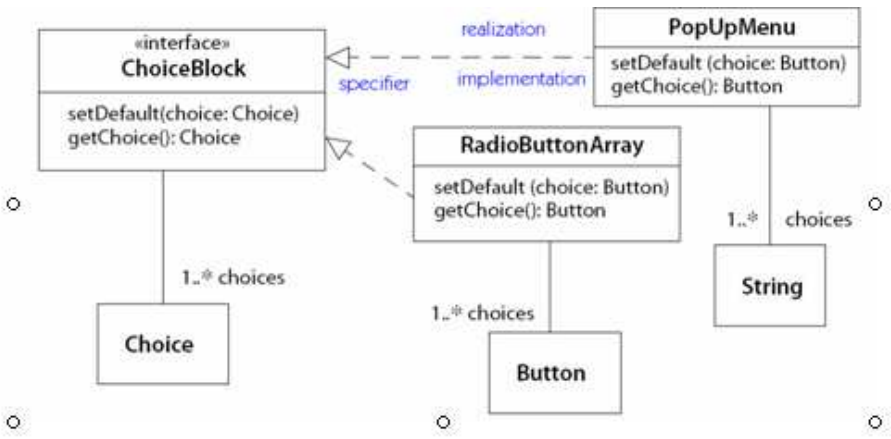
- *Generalización:* es una relación de especialización-generalización en la cual los objetos del elemento especializado (el hijo) pueden sustituir a los objetos del elemento general (el padre). De esta forma el hijo comparte la estructura y el comportamiento del padre.

Representación de una generalización

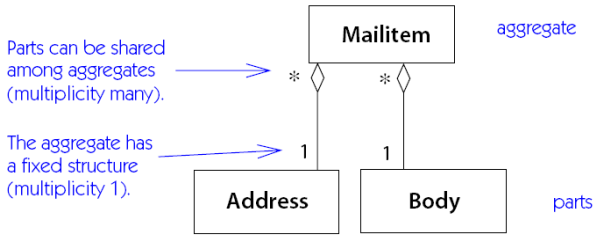


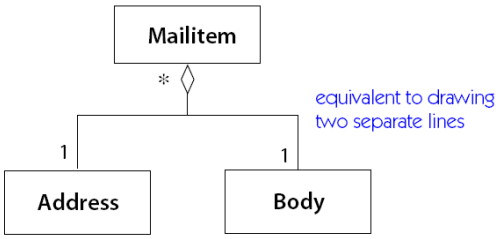
- *Realización:* es una relación semántica entre clasificadores, en donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá. Se pueden encontrar relaciones de realización: entre interfaces y las clases o componentes que las realizan, y entre los casos de uso y las colaboraciones que los realizan.

Representación de una realización



- *Agregación:* Es una forma especial de asociación que especifica una relación todo-parte entre el agregado y una parte que lo compone. Una agregación se representa mediante un rombo en el extremo ``todo'' de la relación.

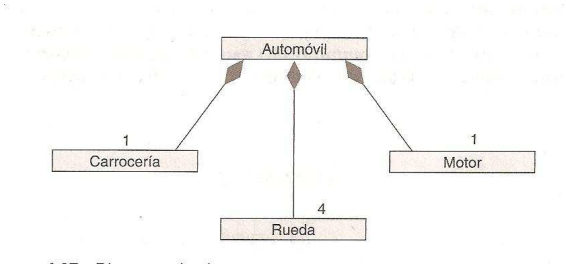




Características importantes:

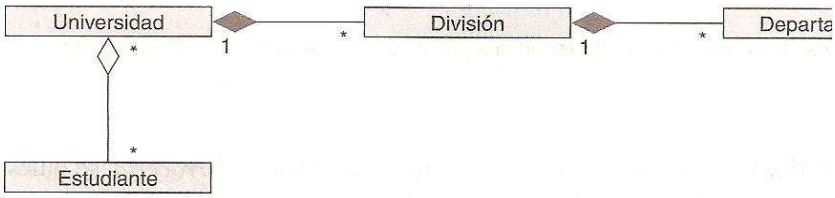
Dependencia existencial: El elemento dependiente desaparece al destruirse el que lo contiene y, si es de cardinalidad 1, es creado al mismo tiempo.
Pertenencia fuerte: Se puede decir que el objeto contenido es parte constitutiva y vital del que lo contiene.
No compartición: Los objetos contenidos no son compartidos, esto es, no forman parte del estado de otro objeto.

- *Composición:* Es un tipo fuerte de agregación, ya que cada componente constituye una composición que solo puede pertenecer al todo del que forma parte. En una composición cada componente pertenece exactamente a un solo “todo”.



Ejemplo:

Composición y agregación



Una universidad esta compuesta por sus Divisiones, que están compuestas de sus Departamentos. Una universidad es indirectamente una composición de sus departamentos; pero la universidad no es una composición de sus Estudiantes, sino mas bien una agregación, ya que sus estudiantes son objetos independientes, incluso pudiendo pertenecer a múltiples universidades.

Relationship	*Function*	*Notation*
association	A description of a connection among instances of classes	———
dependency	A relationship between two model elements	- - - - →
flow	A relationship between two versions of an object at successive times	- - - - →
generalization	A relationship between a more general description and a more specific variety of the general thing, used for inheritance	——▷
realization	Relationship between a specification and its implementation	- - - -▷
usage	A situation in which one element requires another for its correct functioning	- - - - →

1.2.3. Reglas de construcción:

Los bloques de construcción de UML no pueden combinarse de cualquier manera. Como cualquier lenguaje UML tiene unas reglas que especifican a qué debe parecerse un modelo bien formado. Un modelo bien formado es aquel que es semánticamente autoconsistente y está en armonía con todos sus modelos relacionados.

UML tiene reglas semánticas para:

Nombres: Cómo llamar a los elementos, relaciones y diagramas.
Alcance: El contexto que da significado específico a un nombre.
Visibilidad: Cómo se pueden ver y utilizar esos nombres por otros.
Integridad: Cómo se relacionan apropiada y consistentemente unos elementos con otros.
Ejecución: Qué significa ejecutar o simular un modelo dinámico.

Los modelos que construidos durante el proceso software de un sistema con gran cantidad de software tienden a evolucionar y pueden ser vistos por diferentes usuarios de formas diferentes y en momentos diferentes. Por esta razón, es común en el equipo de desarrollo no sólo construir modelos bien formados, sino también construir modelos que sean:

Abreviados: Ciertos elementos se ocultan para simplificar la vista.
Incompletos: Pueden estar ausentes ciertos elementos.
Inconsistentes: No se garantiza la integridad del modelo.

Estos modelos que no llegan a ser bien formados son inevitables conforme los detalles de un sistema van apareciendo y mezclándose durante el proceso software. Las reglas de UML estimulan (pero no obligan) a considerar las cuestiones más importantes de análisis, diseño e implementación que llevan a tales sistemas a convertirse en bien formados con el paso del tiempo.

1.2.4 Mecanismos de construcción

Existen cuatro mecanismos comunes que se aplican de forma consistente a través de todo el lenguaje:

Especificaciones: UML no es sólo un lenguaje gráfico, sino que detrás de cada notación gráfica hay una especificación que proporciona una explicación textual de la sintaxis y semántica del bloque de construcción. Por ejemplo, detrás del icono de una clase hay una especificación que proporciona información de sus atributos, operaciones, signaturas y comportamiento de la clase (visualmente el icono de la clase puede mostrar sólo parte de la especificación). La notación gráfica de UML se utiliza para visualizar el sistema, la especificación se utiliza para enunciar los detalles de dicho sistema.

Adornos: La mayoría de los elementos UML tienen una única y clara notación gráfica que proporciona una representación visual de los aspectos más importantes del elemento. Pero la especificación puede incluir otros detalles algunos de los cuales se pueden incluir como adornos gráficos en la notación base. Por ejemplo se pueden incluir adornos en el icono de una clase indicando la visibilidad de la operaciones.

Divisiones comunes: Al modelar sistemas orientados a objetos, el mundo puede dividirse por lo menos en un par de formas: clase-objeto e interfaz-implementación. Una clase es una abstracción, un objeto es una manifestación concreta de dicha abstracción. Todos los bloques de construcción de UML (y no solo las clases) presentan esta dicotomía. Por ejemplo se puede tener casos de uso e instancias de casos de uso, componentes e instancias de componentes. Una interfaz declara un contrato, una implementación representa la realización concreta de ese contrato responsable de hacer efectiva de forma fidedigna la semántica completa de la interfaz. Casi todos los bloques de construcción presentan esta otra dicotomía. Por ejemplo, se pueden tener casos de uso y las colaboraciones que los realizan, así como operaciones y los métodos que las implementan.

Mecanismos de extensibilidad: UML proporciona un lenguaje estándar para escribir "planos" software, pero no es posible que un lenguaje cerrado sea siempre suficiente para expresar todos los matices posibles de todos los modelos en todos los dominios y en todos los momentos. Por esta razón UML es abierto-cerrado, siendo posible extender el lenguaje de manera controlada.

Los mecanismos de extensión incluyen: Estereotipos, Valores etiquetados y Restricciones.

Estereotipos: El concepto inicial de estereotipo (stereotype) se refería a una clasificación de alto nivel de un objeto que proporciona una idea del tipo de objeto del que se trata. Dicho concepto ha sido recogido en UML para proporcionar un mecanismo de extensión para el propio lenguaje. Este mecanismo hace posible definir UML como un conjunto mínimo de símbolos que podrían ser extendidos, cuando fuese necesario, para proporcionar artefactos de modelado que tienen su significado dentro de un proceso software específico.

Mediante la utilización de estereotipos se pueden crear nuevos tipos de elementos de modelado basados en los elementos que forman el *metamodelo* UML, por tanto, un estereotipo será un nuevo tipo de elemento de modelado que extiende la semántica del metamodelo pero no la estructura de los tipos o clases preexistentes. Algunos estereotipos están predefinidos en el UML, otros pueden ser definidos por el usuario.

Gráficamente un estereotipo se representa como un nombre entre comillas <<nombre-estereotipo>>. Opcionalmente el elemento estereotipado se puede dibujar con un nuevo icono asociado al estereotipo.

Estereotipo de una clase

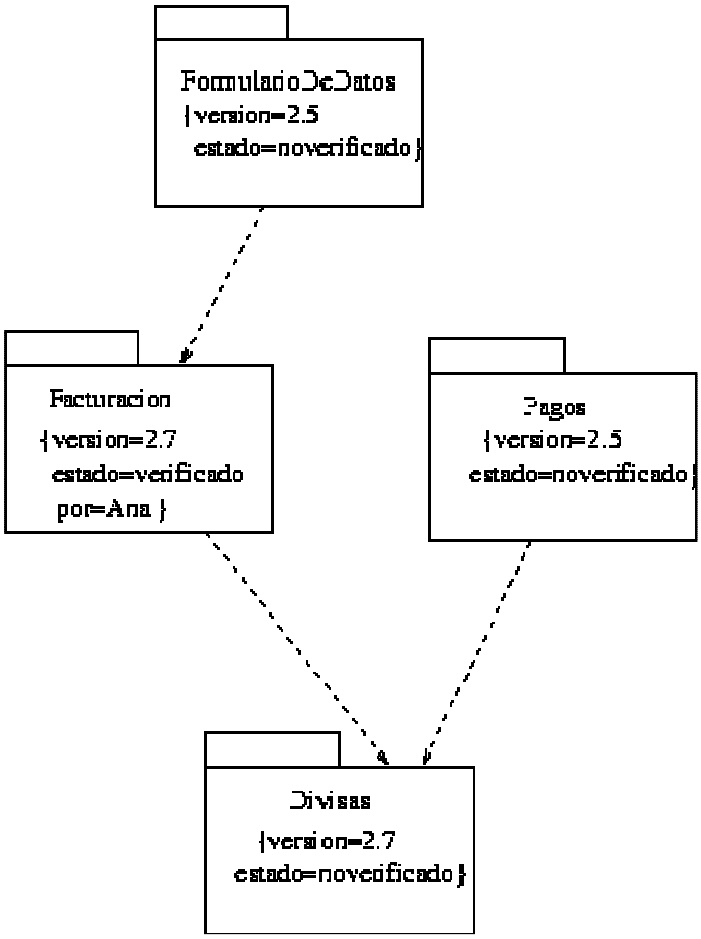
Un estereotipo: Es un metatipo ya que crea una nueva clase en el metamodelo UML. Mediante la creación de un estereotipo de crea un nuevo bloque de construcción con sus propias características (puede proporcionar su propio conjunto de valores etiquetados), semántica (puede proporcionar sus propias restricciones), y notación (puede proporcionar su propio icono).

Todas las extensiones realizadas sobre UML se pueden describir como una colección de estereotipos que dentro del diagrama de clases serán estereotipos de clases, asociaciones y generalizaciones. Se pueden pensar en los estereotipos como subtipos de los tipos Clase, Asociación y Generalización del metamodelo.

Valores etiquetados: Un valor etiquetado es una extensión de las propiedades de un elemento de UML, permitiendo a nadir nueva información en la especificación del elemento. Gráficamente un valor etiquetado se representa como una cadena de caracteres entre llaves asociada al nombre del elemento. La cadena incluye un nombre (etiqueta), un separador (=), y un valor (de la etiqueta).

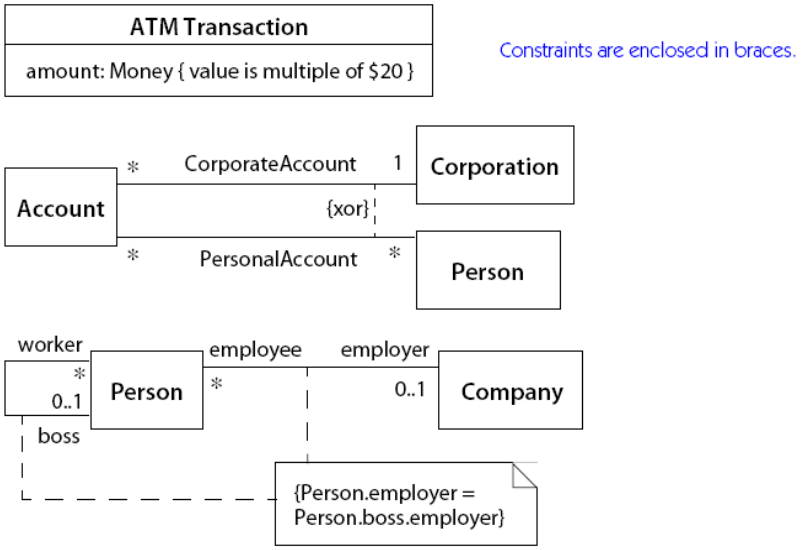
Todo elemento UML tiene su propio conjunto de propiedades: las clases tienen nombres, atributos y operaciones; las asociaciones tienen nombres y dos o más extremos, etc. Si con estereotipos podemos añadir nuevos elementos a UML, con los valores etiquetados podemos a nadir nuevas propiedades. Un valor etiquetado no es lo mismo que un atributo de una clase, sino que más bien es un metadato ya que su valor se aplica al propio elemento no a sus instancias.

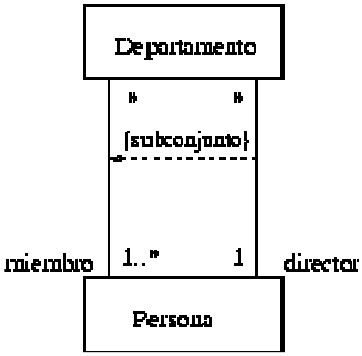
Los valores etiquetados se utilizan frecuentemente para modelar la gestión de configuraciones del proyecto. Entre otras cosas esto conlleva llevar el control de los números de versión, del estado actual y verificación e incluso de la fecha de creación.



Restricciones:

Una restricción es una extensión de la semántica UML, que permite a nadir nuevas reglas o modificar las existentes. Gráficamente, una restricción se representa como una cadena de caracteres entre llaves colocada junto al elemento al que está asociada o conectada a él por una relación de dependencia. Una restricción especifica condiciones que deben cumplirse para que el modelo esté bien formado. Las restricciones se pueden escribir como texto libre,





1.3 TIPOS DE DIAGRAMAS

Un diagrama es la representación gráfica de un conjunto de elementos, visualizándolo la mayoría de las veces como un grafico con nodos (elementos) y flechas (relaciones). Los diagramas se dibujan para visualizar el sistema desde diferentes perspectivas, de forma que un diagrama es una proyección de un sistema. En teoría un diagrama puede contener cualquier combinación de elementos y relaciones, sin embargo en la práctica solo surge un pequeño número de combinaciones:

1. Diagrama de clases.
2. Diagrama de objetos.
3. Diagrama de casos de uso.
4. Diagrama de secuencia.
5. Diagrama de colaboración.
6. Diagrama de estados.
7. Diagrama de actividades.
8. Diagrama de componentes.
9. Diagrama de despliegue.

Las vistas existentes en UML son:

- *Vista casos de uso:* Se forma con los diagramas de casos de uso, colaboración, estados y actividades.

- *Vista de diseño:* Se forma con los diagramas de clases, objetos, colaboración, estados y actividades.

- *Vista de procesos:* Se forma con los diagramas de la vista de diseño. Recalcando las clases y objetos referentes a procesos.

- *Vista de implementación:* Se forma con los diagramas de componentes, colaboración, estados y actividades.

- *Vista de despliegue:* Se forma con los diagramas de despliegue, interacción, estados y actividades.

Se Dispone de dos tipos diferentes de diagramas los que dan una vista estática del sistema y los que dan una visión dinámica.

Los diagramas estáticos son:

Diagrama de clases: Muestra las clases, interfaces, colaboraciones y sus relaciones. Son los más comunes y dan una vista estática del proyecto.

Diagrama de objetos: Es un diagrama de instancias de las clases mostradas en el diagrama de clases. Muestra las instancias y como se relacionan entre ellas. Se da una visión de casos reales.

Diagrama de componentes: Muestran la organización de los componentes del sistema. Un componente se corresponde con una o varias clases, interfaces o colaboraciones.

Diagrama de despliegue.: Muestra los nodos y sus relaciones. Un nodo es un conjunto de componentes. Se utiliza para reducir la complejidad de los diagramas de clases y componentes de un gran sistema. Sirve como resumen e índice.

Diagrama de casos de uso: Muestran los casos de uso, actores y sus relaciones. Muestra quien puede hacer que y relaciones existen entre acciones (casos de uso). Son muy importantes para modelar y organizar el comportamiento del sistema.

Lo diagramas dinámicos son:

Diagrama de secuencia y Diagrama de colaboración: Muestran a los diferentes objetos y las relaciones que pueden tener entre ellos, los mensajes que se envían entre ellos. Son dos diagramas diferentes, que se puede pasar de uno a otro sin pérdida de información, pero que nos dan puntos de vista diferentes del sistema. En resumen, cualquiera de los dos es un Diagrama de Interacción.

Diagrama de estados: muestra los estados, eventos, transiciones y actividades de los diferentes objetos. Son útiles en sistemas que reaccionen a eventos.

Diagrama de actividades: Es un caso especial del diagrama de estados. Muestra el flujo entre los objetos. Se utilizan para modelar el funcionamiento del sistema y el flujo de control entre objetos.

UML permite definir solo los necesarios, ya que no todos son necesarios en todos los proyectos.

Los diagramas a representar dependerán del sistema a desarrollar, para ello se efectúan las siguientes recomendaciones dependiendo del sistema. Estas recomendaciones se deberán adaptar a las características de cada desarrollo, y seguramente será la práctica lo que nos indicara si debemos utilizar algún diagrama mas para especificar a detalle cualquier fase de la aplicación.

Aplicación monopuesto:

- Diagrama de casos de uso.
- Diagrama de clases.
- Diagrama de colaboración.

Aplicación monopuesto, con entrada de eventos:

- Añadir: Diagrama de estados.

Aplicación cliente servidor:

- Añadir: Diagrama de despliegue y diagrama de componentes, dependiendo de la complejidad.

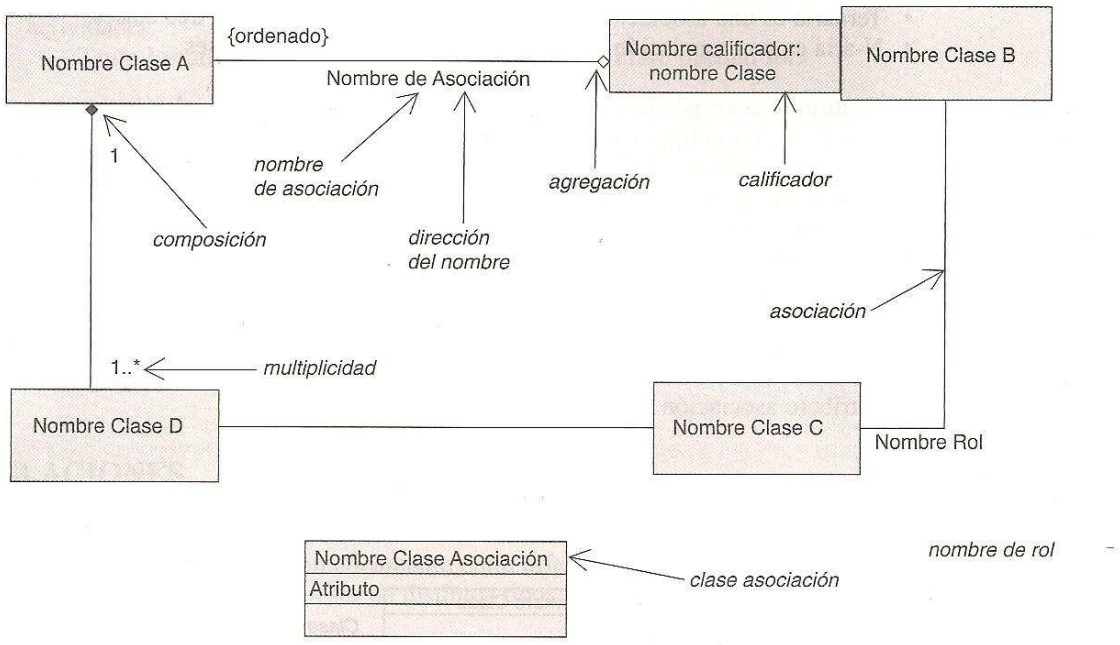
Aplicación compleja distribuida:

- Todos.

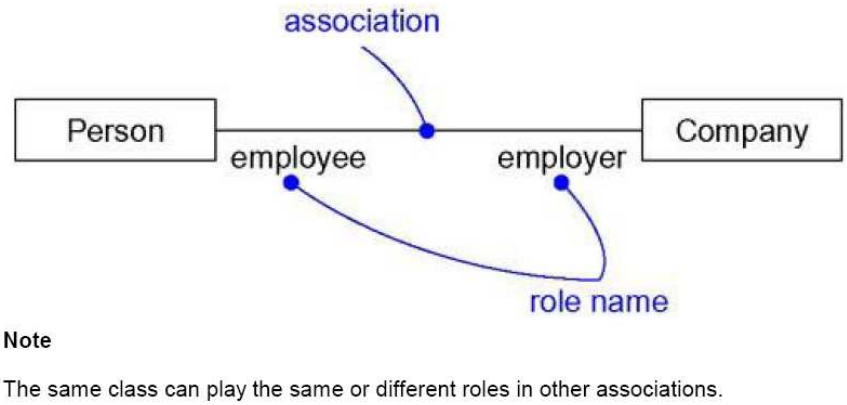
Así tenemos que para una aplicación sencilla debemos realizar entre tres y seis tipos de diagramas, y para una aplicación compleja unos nueve tipos. El tiempo dedicado a la realización de los diagramas es proporcional al tamaño del sistema a realizar. Para la mayoría de los casos tendremos suficiente con tres o cuatro diagramas. UML está pensado para el modelado tanto de pequeños sistemas como de sistemas complejos, y debemos tener en cuenta que los sistemas complejos pueden estar compuestos por millones de líneas de código y ser realizados por equipos de centenares de programadores y gracias a estos diagramas podemos diseñar cualquier tipo de sistema.

1.3.1 DIAGRAMAS UML

1.3.1 Diagramas de Clases.

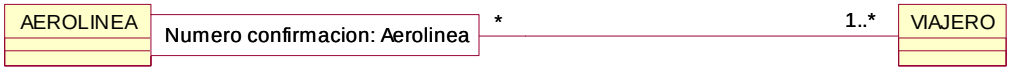


ROL: Es conocido como el papel de una asociación que identifica un extremo de una relación. Una asociación binaria tiene dos roles, cada uno de los cuales recibe su propio nombre. El rol es un nombre que identifica unívocamente cada una de las clases participantes en una asociación.

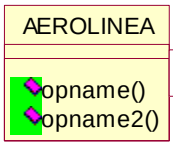


CALIFICADOR: Cuando una multiplicidad es de uno a muchos el mayor desafío es realizar la búsqueda. Cuando un objeto de una clase tiene que escoger un objeto de otra para llevar a cabo el rol en una asociación, la primera clase tiene que relacionarse con un atributo en específico para encontrar el objeto correcto. Ese atributo es típicamente un identificador, el cual puede ser un número de identificación. Ejemplo cuando realizas una reservación de un vuelo, la aerolínea te asigna un número de confirmación. Es decir si quieres realizar una consulta de esta reservación, tienes que proveer el número de confirmación para realizar cualquier cambio.

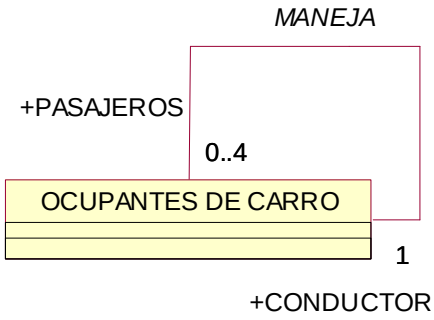
Una asociación calificada reduce la multiplicidad del rol opuesto al considerar el valor del calificador.



CLASE ACTIVA: Son similares a las clases excepto que sus objetos representan elementos cuyo comportamiento es concurrente con otros elementos.



ASOCIACION REFLEXIVA: Es una asociación que se da en una clase misma. Esto ocurre cuando un clase contiene objetos que pueden jugar una variedad de roles en la misma clase. Ejemplo: Un ocupante de un carro puede ser conductor o también pasajero. En el rol de conductor, el Ocupante del carro maneja con una relación de cero o más ocupantes de carro quien juega el rol de pasajeros.

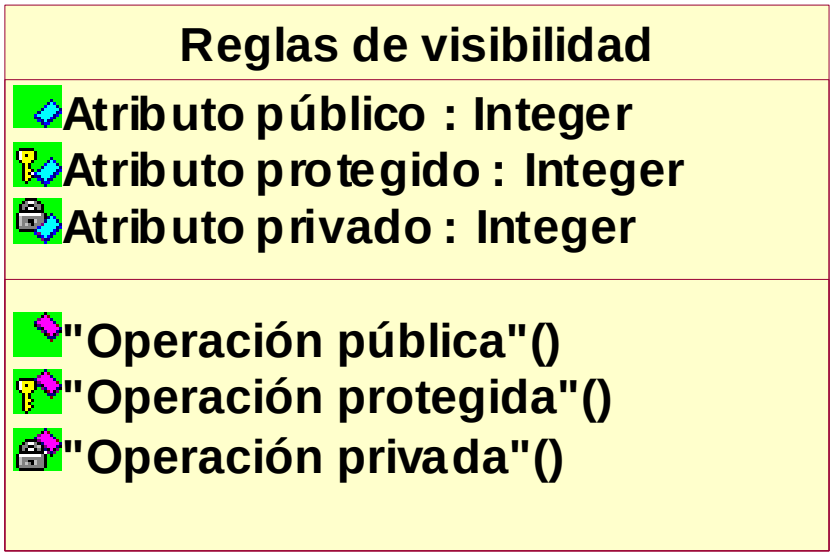


ENCAPSULAMIENTO DE CLASES:

(-) *Privado* : es el más fuerte. Esta parte es totalmente invisible para otras clases.

(#) Los atributos/operaciones *protegidos* están visibles para la clase y para las clases derivadas de la original

(+) Los atributos/operaciones *públicos* son visibles a otras clases



CASO PRACTICO: “ANIMALES DEL UNA CIERTA REGION”

Diseñar una aplicación orientada a objetos que describa la siguiente situación:

En una ~~región~~ viven cinco animales: Una ballena “Moby Dick”, que no dice nada; un perro fiero llamado “Cain” que dice “Grr”, un perro manso llamado “Abel” que dice “Guau” un pingüino llamado “Adela” que no dice nada y un loro que dice “Lorito bonito”, “Pretty Polly” y “Viva mi dueño”.

Especificar la jerarquía de herencia, las clases, los atributos y métodos de cada clase.

1.- El primero de los pasos que debemos realizar para obtener un diagrama de clases de este problema consiste en extraer todos los sustantivos que aparecen en el enunciado. La lista de sustantivos escritos deben ser en singular.

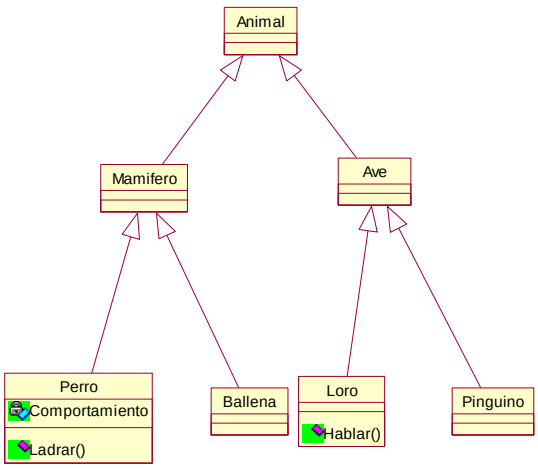
~~2.-~~ Después de identificar todas las posibles clases que se extraen de forma directa del enunciado del problema pasamos a eliminar aquellos que no aportan nada para modelar el problema. *Eliminamos la clase región ya que hace referencia al lugar en el que se describe la situación y no aporta nada.*

3.- Identificar los atributos de las clases. Para ello observamos los posibles valores que una propiedad de una clase puede tomar, el rango de los valores o una regla que enuncia a todos los posibles valores. *Solo tiene atributo la clase perro se define comportamiento que puede contener valores fiero o manso. El atributo será privado (-)*

4.- Identificar las operaciones de una clase. *El perro hace “guau” y el loro repite palabras varias veces con esto se deduce que el perro tendrá una operación Ladrar y el Loro una operación Hablar. Esta operación será publica ya que se presenta en dos clases.*

5.- Seleccionar el símbolo (–) privado (+) público para las operaciones y atributos en cada clase.

6.- Para realizar el diagrama jerárquico necesitamos agrupar lógica y semánticamente a cada una de las clases. *En este caso agrupamos como mamífero al perro y a la ballena y como Ave al Loro y al Pingüino.*



RESERVACION DE VUELOS

El sistema de reservación de vuelos es un sistema que permite al usuario hacer consultas y reservaciones de vuelos, además de poder comprar los boletos de forma remota, sin necesidad de recurrir a un agente de viajes humano. Se desea que el sistema de reservaciones sea accesible a través de World Wide Web.

El sistema actualmente tiene una Terminal de Servicio de Reservaciones (formado por un ratón Genius, teclado IBM y monitor Sony) en donde se presenta un mensaje de bienvenida describiendo los servicios ofrecidos junto con la opción para registrarse por primera vez, o si ya está registrado, poder utilizar el sistema de reservaciones de vuelos. Este acceso se da por medio de la inserción de un login previamente especificado (dirección de correo electrónico del usuario) y una contraseña previamente escogida y que debe validarse.

Una vez registrado el usuario, y después de haberse validado el registro y contraseña del usuario, se pueden seleccionar las siguientes actividades.

Consulta de vuelos
Reservación de vuelos
Compra de boletos de avión

La consulta de vuelos se puede hacer de 3 maneras diferentes:

Horario de vuelos
Tarifas de Vuelos
Información de Vuelo

La consulta según horario muestra los horarios de las diferentes aerolíneas que dan servicio entre dos ciudades. La consulta según tarifas muestra los diferentes vuelos entre dos ciudades ordenados por su costo. La información de vuelos se utiliza principalmente para consultar el estado de algún vuelo, incluyendo información de si existen asientos disponibles y, en el caso de un vuelo para el mismo día, si este está en hora. Se pueden incluir preferencias en las búsquedas como fecha y horario deseado, categoría de asientos, aerolínea deseada y si se desean solo vuelos directos.

La reservación de vuelo permite al cliente hacer una reservación para un vuelo en particular, especificando la fecha y horario, bajo una tarifa establecida. Es posible reservar un itinerario compuesto de múltiples vuelos, para uno o más pasajeros, además de poder reservar asientos.

La compra permite al cliente, dada una reserva de vuelo previa y una tarjeta de crédito valida, adquirir los boletos de avión.

Los boletos de avión serán posteriormente enviados al cliente, o estarán listos para ser recogidos en el mostrador del aeropuerto antes de la salida del primer vuelo.

Es necesario estar previamente registrado con un número de tarjeta de crédito válida para poder hacer compras de boletos de avión o bien proveerla al momento de la compra.

Además de los servicios de vuelo, el usuario podrá en cualquier momento leer, modificar o cancelar su propio registro, todo esto después de haber sido el usuario validado en el sistema.

Finalmente se nos comenta que existe un operador encargado del mantenimiento del sistema, pero no se describirá su labor, dejando dichas descripciones para futuras entrevistas.

PASOS PARA LA REALIZACION DE ESTE DIAGRAMA

1.- Identificación de clases, seleccionando todos los sustantivos en la descripción del problema.

2.- Selección de clases vamos a eliminar las clases innecesarias

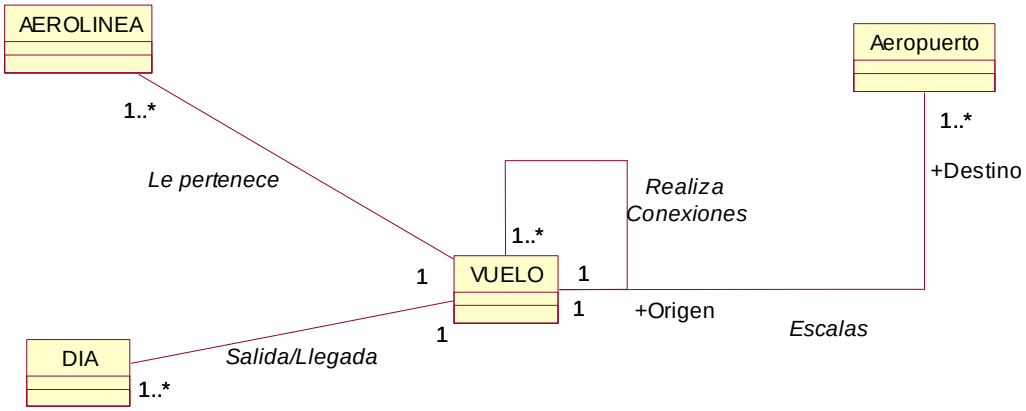
- ~~Clases redundantes:~~
  - Usuario puede ser mas descriptivo para una aplicación de análisis de sistemas En el caso del sistema de reservación de vuelos Cliente se mantiene
  - Consulta de vuelo (consulta)
  - Reserva de vuelo (reserva)
  - Compra de boleto (compra)
  - Sistema de reservación de vuelo (sistema de reservación)
  - Boleto (boleto aéreo)
  - Costo (Tarifa)
  - Tarifa de vuelo (tarifa)
  - Vuelo Directo (vuelo)
  - Login (email)
  - Horario (hora)
  - Fecha (día)
  - Dirección de correo electrónico
  -
- ~~Clases irrelevantes:~~
  - Mostrador del aeropuerto, Agente de Viajes humano
- ~~Clases imprecisas:~~ Estas clases pueden ser utilizadas como herencia y puede que sea necesario una clase para compartir aspectos comunes a ambas clases.
  - Sistema
  - Servicios
  - Actividad
  - Preferencia
  - Búsqueda
  - Información
  - Estado
  - Opción
  - Acceso
  - Itinerario
- Nombre de clases: Se asigna Aeropuerto en lugar de ciudad
- ~~Clases que son Atributos:~~
  - Número de Tarjeta de Crédito – Tarjeta de crédito
  - Categoría asiento – asiento
  - Información de vuelo – vuelo
  - Horario de vuelo – vuelo
- ~~Clases que son Operaciones:~~
  - Consulta
  - Compra
  - Reserva
- ~~Clases de interfaces de usuario:~~
  - Mensaje de bienvenida

- ~~Clase de sistema completo:~~
    - Sistema de reservaciones
- ~~Clases que son actores:~~
    - Cliente
    - Operador
- Clases que son agrupación de una clase
    - Teclado
    - Mouse
    - Monitor
- Clase que hay que tomar en cuenta pero que en este momento no tiene relevancia
    - Entrevista

Clases candidatas de nuestro sistema a analizar:

Reserva	Asiento	contraseña
Vuelo	Día	Registro
Aerolínea	hora	TSR (Terminal sistema Reserva)
Aeropuerto (que era ciudad)	Pasajero	
Tarifa	Tarjeta de Crédito	

3.-Despues de haber seleccionado las clases, se construye un primer diagrama



4.- El siguiente paso que realizaremos será la identificación de relaciones. Inicialmente se muestran las relaciones básicas existentes entre las diferentes clases del sistema. Para ello identificamos las siguientes

- Reserva de vuelos
- Asientos en un vuelo
- Fecha y horario del vuelo
- Aerolínea deseada
- Tarifa de vuelo
- Itinerario de vuelos.

Identificando lo siguiente:

- El vuelo contiene reservaciones
- El vuelo contiene asientos
- El vuelo tiene día y hora
- El vuelo pertenece a una aerolínea
- El vuelo tiene tarifas
- El vuelo se compone de un itinerario
- El pasajero realiza reservaciones
- El pasajero posee una tarjeta de crédito

Después de identificar y seleccionado las asociaciones, se construye un diagrama de clases con las asociaciones, los roles y la multiplicidad quedando el siguiente diagrama y se eliminan las clases que no utilizaremos (Terminal, contraseña y registro)

El vuelo se denomina por medio de un número, tiene como origen un aeropuerto en una ciudad y tiene como destino un aeropuerto de otra ciudad. Un vuelo puede tener múltiples escalas y múltiples vuelos, se relacionan por medio de conexiones. El vuelo pertenece a una aerolínea y puede operar viarios días a la semana teniendo un horario de salida y otro de llegada.

El aeropuerto sirve como origen, destino y escalas de un vuelo. El aeropuerto se encuentra en una ciudad de un país determinado.

Se identifica una clase adicional como avión y las relaciones básicas existentes entre las clases aerolínea, Avión, tarifa, asiento y vuelo.

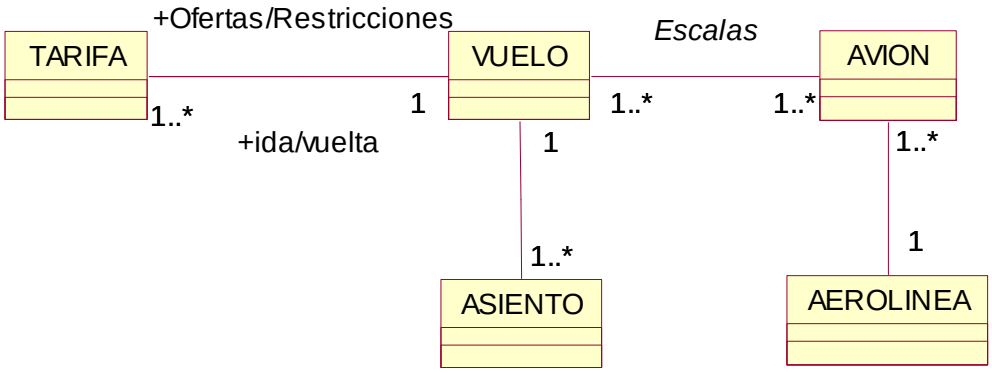
La aerolínea provee servicio de múltiples vuelos entre diferentes ciudades bajo diferentes ciudades bajo diferentes horarios. La aerolínea se identifica por un nombre.

Un vuelo en una fecha determinada se hace en un tipo de avión particular. El tipo de avión define la cantidad máxima de pasajeros que puede viajar en ese vuelo para esa fecha.

Los diferentes vuelos tienen múltiples tarifas para compra de billete, variando según la clase de boleto de avión, si son de ida o de idea y vuelta, y dependiendo de las diversas restricciones y ofertas existentes.

En la reservación de vuelos se puede incluir una solicitud de asignación de asiento, especificando preferencias como pasillo o ventana. El número de asientos disponibles en un vuelo particular depende del tipo de avión que opere ese día.

SEGUNDO DIAGRMA CON RELACIONES Y DETALLE DE CLASES

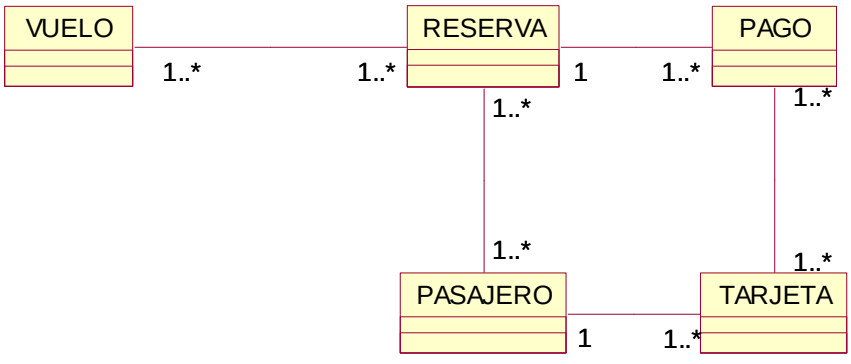


Otras de las relaciones que se encuentran son las siguientes:

- El horario de un vuelo se define según los días que opera
- El horario de un vuelo se determina por su hora de salida y la hora de llegada durante los días que opera.

- También identificamos una clase llamada pago que consta de información sobre la cantidad, fecha y tipo de transacción.

- Por razones de seguridad los pagos de los boletos se hacen con tarjeta de crédito



5.- Finalmente identificamos los atributos según la descripción del problema. Los atributos de las clases los cuales se han podido obtener antes de proceder a la determinación de las asociaciones.

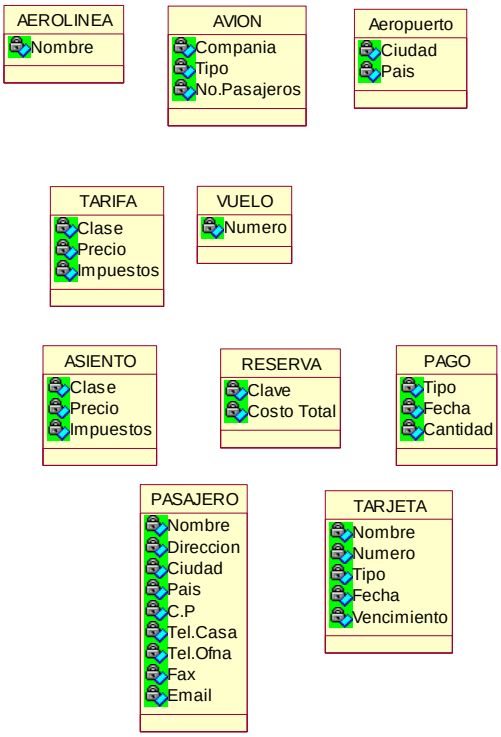
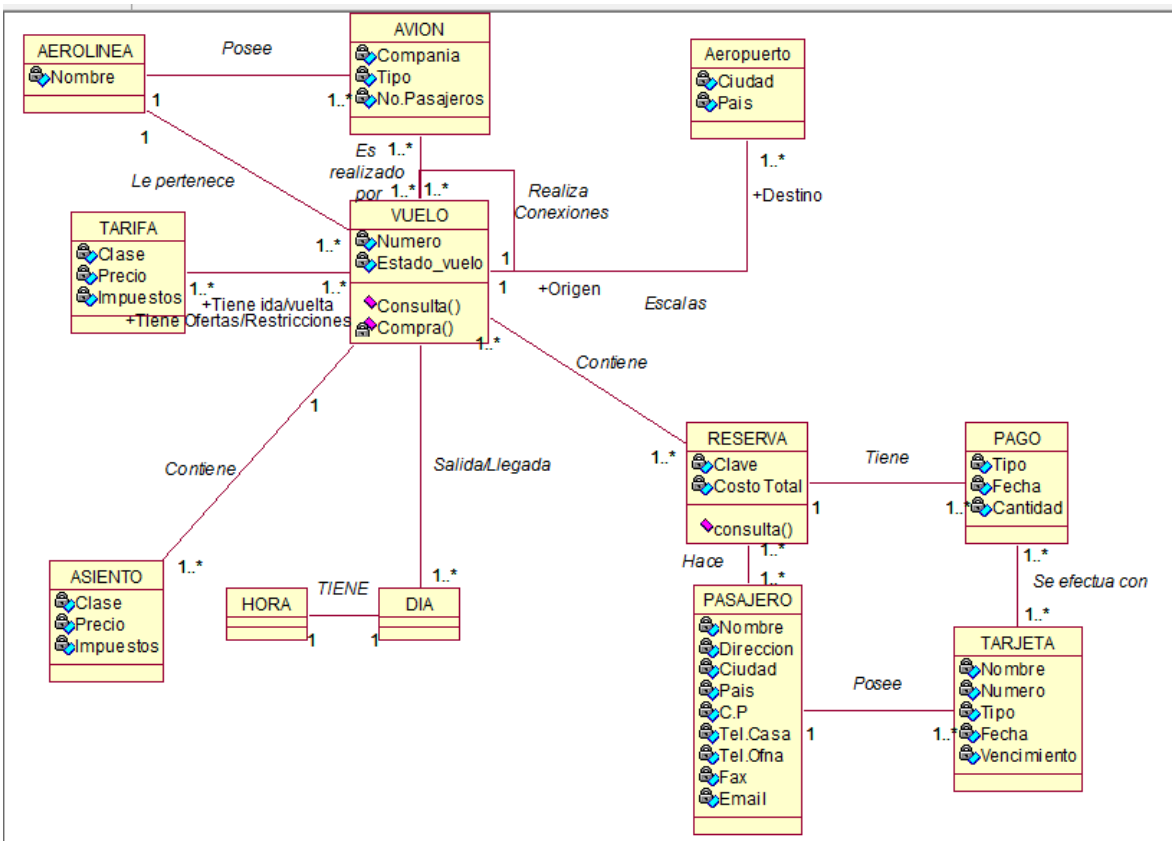


Diagrama Final



1.3.2 DIAGAMAS DE OB JETOS

Los diagramas de objetos modelan las instancias de elementos contenidos en los diagramas de clases. Un diagrama de objetos muestra un conjunto de objetos y sus relaciones en un momento concreto. En UML, los diagramas de clase se utilizan para visualizar los aspectos estáticos del sistema y los diagramas de interacción se utilizan para ver los aspectos dinámicos del sistema, y constan de instancias de los elementos del diagrama de clases y mensajes enviados entre ellos.

Los diagramas de objetos se emplean para modelar la vista de diseño no estática o la vista de procesos estática de un sistema al igual que se hace con los diagramas de clases, pero desde la perspectiva de instancias reales o prototípicas. Esta vista sustenta principalmente los requisitos funcionales de un sistema. Los diagramas de objetos permiten modelar estructuras de datos estática por medio de una colaboración entre los objetos.

En general los diagramas de objetos se utilizan para modelar estructuras de objetos, lo que implica tomar una vista instantánea de los objetos de un sistema en un cierto momento.

Los diagramas de objetos se utilizan para visualizar, especificar, construir y documentar la existencia de ciertas instancias en el sistema, junto a las relaciones entre ellas.

EJEMPLO:

CAMPAÑA DE MERCADOTECNIA

En una importante corporación se realizara una nueva campaña de un producto.

El gerente corporativo del departamento de mercadotecnia le comenta al gerente regional de ventas que está pensando crear una campaña para su nuevo producto de telefonía celular. El gerente regional de ventas crea la campaña y se la asigna al gerente de ventas de Tijuana para ver cómo funciona en este nicho de mercado considerado el mas difícil de penetrar por su cercanía a Estados Unidos. El gerente de ventas de Tijuana le indica a un vendedor que ese producto se tiene que ofrecer de acuerdo a la campaña corporativa que se indico el gerente corporativo del depto. de mercadotecnia en la cd. De México. El vendedor realizara llamadas a clientes potenciales, también se realizará por medio de la agencia de relaciones publicas corporativa una campaña por medio del periódico ya que este totalmente creada la campaña por el gerente regional de ventas y que el gerente de ventas Tijuana coordine a los vendedores para esta importante campaña.

PASOS PARA LA REALIZACION DE ESTE DIAGRAMA

1.- Identificar la secuencia del flujo de información al igual que sus objetos involucrados.
2. Encontrar de las clases, las instancias que puedan tener cada una de ellas.
3. Encontrar los atributos valorados de cada una de las instancias de las clases.
4. Modelar las instancias de las clases.
5. Identificar los objetos y sus relaciones

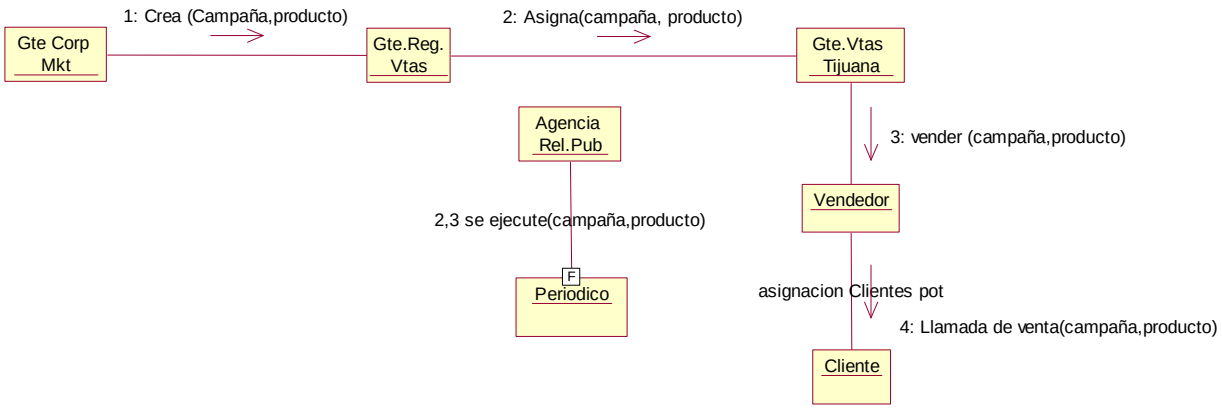
1.- El gerente corporativo de mercadotecnia informa al gerente regional de ventas que se realizara una campaña de un producto en particular.

2.- El gerente regional de ventas crea la campaña y se la asigna al gerente de ventas de Tijuana.

3.- El gerente de ventas Tijuana dirige a un vendedor para vender el producto de acuerdo a la campaña indicada.

4.- El vendedor realiza llamadas a clientes potenciales

5.- Posterior a que el gerente regional de ventas haga la asignación de la campaña y el que gerente de ventas Tijuana implemente su campaña (cuando el paso 2 y 3 este completo) la agencia de relaciones publicas corporativa lanzara la campaña en el periódico.



Como vemos los diagramas de objetos son especialmente útiles para modelar estructuras de datos complejas. Evidentemente puede existir una multitud de posibles instancias de una clase particular, y para un conjunto de clases con t relaciones entre ellas, pueden existir muchas más configuraciones posibles de estos objetos. Por lo tanto, al utilizar diagramas de objetos sólo se pueden mostrar significativamente conjuntos interesantes de objetos concretos o prototípicos.

1.3.3 DIAGRAMAS DE CASOS DE USO

Los casos de uso documentan el comportamiento del sistema desde el punto de vista del usuario. En este caso por <<usuario>> se entiende cualquier cosa que interactúa con el mismo. Un usuario podría ser una persona, otro sistema de información, un dispositivo hardware etc. El modelado de los casos de uso ayuda con tres de los aspectos más difíciles del desarrollo:

- La captura de requisitos
- La planificación de las interacciones del desarrollo
- La validación de los sistemas.

Los casos de uso los presento por primera vez Ivar Jacobson a principios de los 90, como un desarrollo a partir de la idea escenarios.

Los casos de uso son relativamente fácil de comprender de forma intuitiva, incluso sin conocer la notación. Esto es una ventaja importante ya que el modelo de casos de uso se puede tratar de forma coherente con el cliente que no necesita estar familiarizado con UML.

Normalmente un caso de uso tiene una única secuencia básica, pero puede tener varias secuencias alternativas.

Cuando se está analizando y describiendo los casos de uso en detalle, lo que en realidad se realiza es la identificación detallada de toda la funcionalidad que ha de tener el sistema. Es posible que aparezcan puntos que no están suficientemente claros en la especificación de requisitos que se ha elaborado hasta el momento. La razón es que las posibles opciones que puedan aparecer, parecen obvias en las primeras fases de la elaboración de la especificación de requisitos y siempre se tienen que especificar.

Puesto que los casos de uso se centran en una funcionalidad particular del sistema, es posible analizar toda la funcionalidad del sistema de un modo incremental. De esta manera se pueden desarrollar los casos de uso de diferentes aéreas de la aplicación de un modo independiente y posteriormente, juntarlos para que todos ellos formen el modelo de requisitos completo.

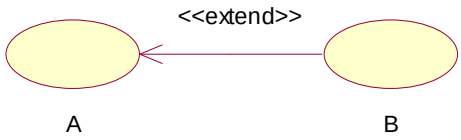
Para definición de casos de uso se pueden utilizar los diagramas de actividad, diagramas de interacción o descripciones informales de texto.

RELACIONES:

Para identificar la comunicación existente entre los actores, los casos de uso

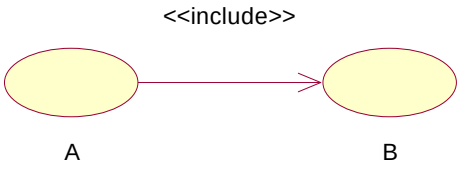
- La relación de generalización entre actores se utiliza para organizar los distintos actores, indicando que una descripción abstracta mayor es compartida y aumentada por una o más descripciones especificas del actor.
- La relación de generalización entre casos de uso identifica que un caso de uso específico hereda y añade propiedades a un caso de uso general.
- La relación de extensión <<extend>> entre casos de uso se utiliza para factorizar las variantes sobre la secuencia básica de un caso de uso en nuevos casos de uso que extiende los flujos principales. Asi pues, la extensión especifica cómo la descripción de un caso puede ser insertada en la descripción de otro caso de uso para ampliarla.

- EJEMPLO:



Esto quiere decir que A ***"opcionalmente"*** ejecuta B

- La relación de inclusión <<include>> entre casos de uso se utiliza para señalar que un caso de uso incorpora el comportamiento de otro caso de uso como parte de su propio comportamiento. Esta relación se utiliza para factorizar el comportamiento común en nuevos casos de uso que puedan ser utilizados por otros. La relación inclusión apunta al caso de uso a ser incluido.
  - EJEMPLO:



Esto quiere decir que A ***"siempre"*** ejecuta B

EJEMPLO:

Gestión de Fincas e inmuebles

Se desea desarrollar una aplicación de gestión de fincas e inmuebles. La aplicación deberá cubrir todos los aspectos relacionados con dicho tema. Teniendo en cuenta la siguiente dinámica de funcionamiento:

Una empresa gestiona un conjunto de inmuebles, que administra en calidad de propietaria. Cada inmueble puede ser bien un local (local comercial, oficina…) un piso o bien un edificio que a su vez tiene pisos y locales. Como el numero de inmuebles que la empresa gestiona no es un número fijo, la empresa propietaria exige que la aplicación permita tanto introducir nuevos inmuebles, con sus datos correspondientes (dirección, CP…) así como darlos de baja, modificarlos y consultarlos. Así mismo que una empresa administre un edificio determinado no implica que gestione todos sus pisos y locales, por lo que la aplicación también deberá permitir introducir nuevos piso o locales con sus datos correspondientes (Planta, letra…), darlos de baja, modificarlos y hacer consultas sobre ellos.

Cualquier persona que tenga una nomina, un aval bancario, un contrato de trabajo o venga avalado por otra persona puede alquilar el edificio completo o alguno de los pisos o locales que no estén ya alquilados, y posteriormente desalquilarlo. Por ello deberán poderse dar de alta, si son nuevos inquilinos, con sus datos correspondientes (Nombre, RFC, edad, sexo…) se pueden modificar, darlos de baja, consultarlos, etc. (para la realización de cualquiera de estas operaciones es necesaria la identificación por parte del inquilino).

Por otra parte, cada mes el secretario de la empresa pedirá la generación de un recibo por cada uno de los pisos y de los locales, el cual lleva asociado un numero de recibo que es único para cada piso y para cada local y que no variara a lo largo del tiempo, indicando el piso o local a que pertenece, la fecha de emisión, la renta, el agua , la luz la actualización del predial anual, mantenimiento, IVA,etc. Y otros conceptos, teniendo en cuenta que unos serán opcionales (solo para algunos recibos) y otros son obligatorios (para todos los recibos).Además para cada recibo se desea saber si está o no cobrado.

Con vistas a facilitar la emisión de recibos cada mes, la aplicación deberá permitir la generación de recibos idénticos a los del mes anterior, a excepción de la fecha. Además deberán existir utilidades par

inicializar los conceptos que se desee de los recibos a una determinada cantidad y también debe ser posible modificar recibos emitidos en meses anteriores al actual. La aplicación también deberá presentar los recibos en formato impreso, pero teniendo en cuenta que en un recibo nunca aparecerán aquellos conceptos cuyo importe seas igual a cero.

De igual forma, el secretario debe poder gestionar los movimientos bancarios que se producen asociados a cada edificio, piso o local. Un movimiento bancario siempre estará asociado a un banco y a una cuenta determinada de ese banco. En esa cuenta existirá un saldo. Acreedor o deudor, que aumentara o disminuirá con cada movimiento. Para cada movimiento se desea saber también la fecha en que se ha realizado. Un movimiento bancario puede ser de dos tipos: un gasto o un ingreso.

Si el movimiento bancario es un gasto, entonces estará asociado a un inmueble determinado, y se indicara el tipo de gasto al que pertenece entre los que se tienen estipulados. Ejemplos de gastos son el costo de la reparación de un elevador del inmueble que pertenece a gastos de reparación, el sueldo de intendencia, etc. Si el movimiento bancario es un ingreso entonces estará asociado a un piso del inmueble determinado o a un local y también se indicara el tipo de ingreso al que pertenece, como en el caso de los gastos. Ejemplos de ingresos son precisamente los recibos que se cobran cada mes a los inquilinos.

Basándose en los gastos e ingresos que se deducen de los movimientos bancarios, la aplicación deberá ser capaz de manejar la gestión económica generando los reportes que facilitan la realización de la declaración de la renta.

Por último, la aplicación deberá ser capaz de proporcionar el acceso, de forma estructurada, a toda la información almacenada en el sistema, generando para ellos los reportes necesarios que requiera el secretario.

Ejemplos de reportes son: El reporte de inquilinos ordenado por fecha, el reporte de inquilinos que han pagado o no en un determinado intervalo de tiempo, el reporte de todos los inmuebles, el reporte de todos los pisos y locales de cada edificio, el reporte de todos los recibos pendientes de cobro en un determinado intervalo de tiempo, etc.

PASOS PARA LA ELABORACION DE ESTE DIAGRAMA

1.- *Identificar actores*: Los actores son mejor dicho, los roles que un usuario o usuarios del sistema llevan a cabo en algún momento del tiempo. También pueden ser otros sistemas con los que el 'sistema' en proceso de modelado tiene interacción. Ejemplo: Para un sistema de ventas (directas y por catalogo), nuestros actores pueden ser: Vendedor, Cliente, Supervisor de Ventas.

2.- *Identificar Metas*: (Metas, objetivos generales o responsabilidades): Todos los actores en el entorno a modelar tienen metas u objetivos, o en su defecto responsabilidades, o en su defecto, acciones que desean realizar u obtener del sistema. Por ejemplo: Para el sistema de Ventas, el Vendedor tiene como meta, objetivo o responsabilidad (Ofrecer productos, Cerrar Venta, Ganar mucho dinero vía Cobrar comisiones).

3.- *Obtener o identificar los Casos de Uso a partir de las Metas:* Las metas son importantes porque a partir de su identificación pasamos a realizarlas y estas se convierten en los Casos de Uso, de esta forma tan sencilla obtenemos la información de funcionalidad que requiere nuestro sistema.

4.- *Especificar cada Caso de Uso:* Una vez identificados seguimos a especificarlos uno a uno, es probable que en el inter, de esos sencillos pasos algunos casos de uso desaparezcan o se fusionen con otros, por ambigüedades detectadas o por detalles que se hayan escapado durante el proceso. Es importante recordar que de eso se trata el modelado, no es indispensable que quede al cien por ciento el modelo desde la primera vez, por eso hay que hacerlo en iteraciones o ciclos. La especificación de los casos de uso contiene varias partes, las fundamentales son Nombre, Descripción, Actores, Flujo Principal y Flujos Alternos. Este grupo de elementos constituyen lo que se conoce como plantilla (Template)

1. Id. Clave o numero de control del Caso de Uso
2. Nombre. Es el Caso de Uso en si
3. Descripción. Aquí detallamos lo que el caso de uso resuelve con base a su objetivo primordial.
4. Actores. En esta sección especificamos el actor o actores principales y los actores secundarios o auxiliares en el caso de uso. Podemos detallar su nombre, una breve descripción y en que otros casos de uso intervienen si se desea. Aunque preferentemente esta especificación se puede hacer por separado en otro documento.
5. Pre-condiciones. Las reglas o condiciones que se deben cumplir antes de que sea iniciado el caso de uso. Por ejemplo, usuario firmado (logged), pago realizado, etc.
6. Post-condiciones. Condiciones que se deben cumplir cuando termine el caso de uso.
7. Flujo Principal En la secuencia de pasos del flujo principal, podemos usar texto solamente numerando cada paso, podemos usar un diagrama de flujo, un diagrama de secuencia, o una grafica de estados para efectos de dar claridad.
8. Inclusiones. <<include>>
9. Extensiones. <<extend>>
10. Requerimientos no funcionales. Cualquier elemento indispensable para la realización del caso de uso, que no tenga impacto en la funcionalidad.
11. Diagrama de Contexto. Nos ilustra el alcance del caso de uso, entradas y salidas generales.
12. Diagrama de Navegación. Este diagrama nos ayuda a ilustrar el flujo entre las pantallas (prototipo) que tendrá el sistema para el caso de uso.

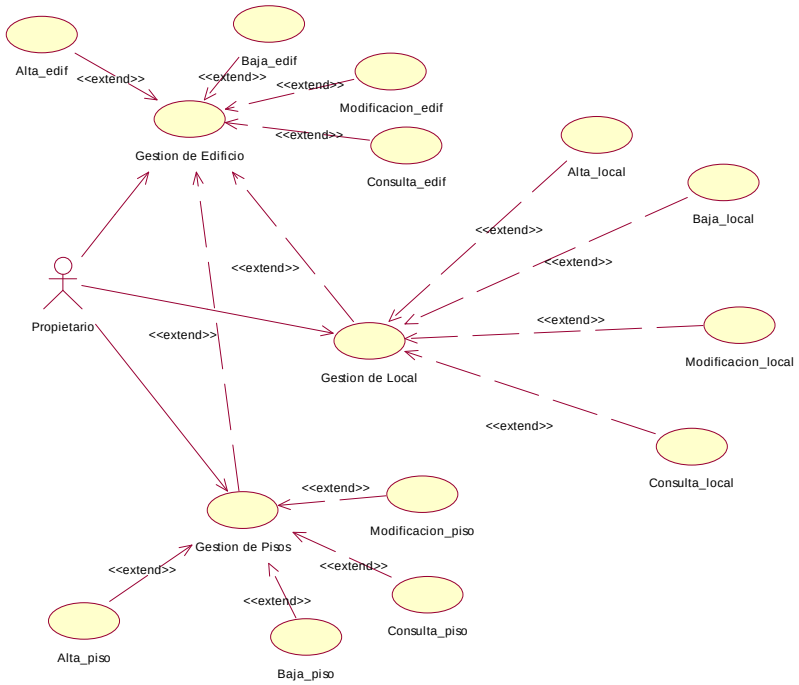
Actor Propietario:

El propietario será el responsable de la gestión del edificio, local o de los pisos mediante el alta, las modificaciones, consulta y baja de los mismos.

Casos de usos:

- Gestión de edificio
- Gestión de locales
- Gestión de piso

Cada uno de los casos de usos refleja las actividades comunes que se deben realizar en el alta, baja, modificación y consulta. Ya que en el problema se hace referencia a estas 4 funciones que deben permitirse en el sistema. Se refleja tal situación introduciendo un caso de uso específico si se hace referencia al edificio, al local o piso, ya que las operaciones que conllevan cada uno es distinta aunque podamos nombrarlas de la misma forma.



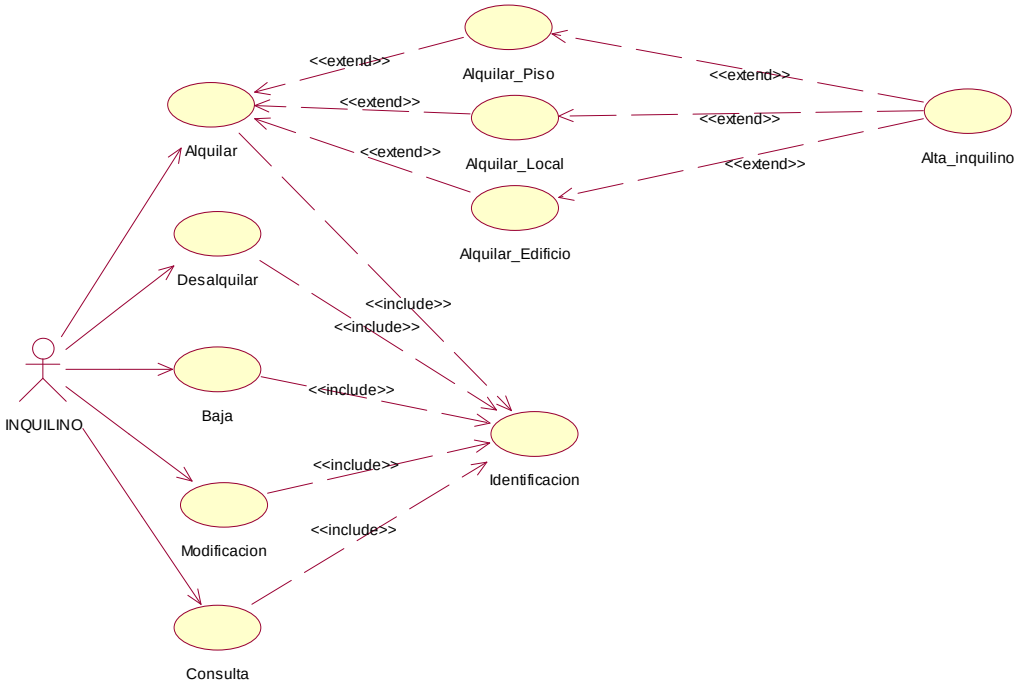
Nota: la opción extend se puede realizar individual o conjuntamente. Además, existe una relación <<extend>> el caso de uso de Gestión de locales y de piso con el caso de uso Gestión de edificio. Con esto reflejamos que la gestión de edificios puede conllevar la gestión de locales de pisos o de ambos.

Actor inquilino:

El inquilino va a ser aquella persona que tiene algún tipo de aval, y puede realizar algunas de las siguientes operaciones en el sistema:

- Alquilar
- Desalquilar
- Darse de baja
- Modificar sus datos
- Consultarlos

La realización de cualquiera de las operaciones es necesario presentar una identificación que se relaciona con todos los casos de usos anteriores mediante la relación de <<include>>



Cuando se produce el alquiler este puede ser el de un piso un local o el edificio. Para que se realice la operación alquiler se debe permitir el alta de los datos del inquilino por lo cual se crea la extensión de Alquiler de piso, local y edificio.

Actor Secretario:

Las tareas son las siguientes:

- Obtención de los distintos tipos de recibos
- Obtener los informes económicos
- Generación de los listados.

Aquí enumeramos las funcionalidades generales, a partir de estos casos de uso se desglosan otros utilizando la relación de extensión en algunos casos de uso

Generación de recibo

- Recibos idénticos mes anterior
- Inicializar concepto
- Modificar los del mes anterior

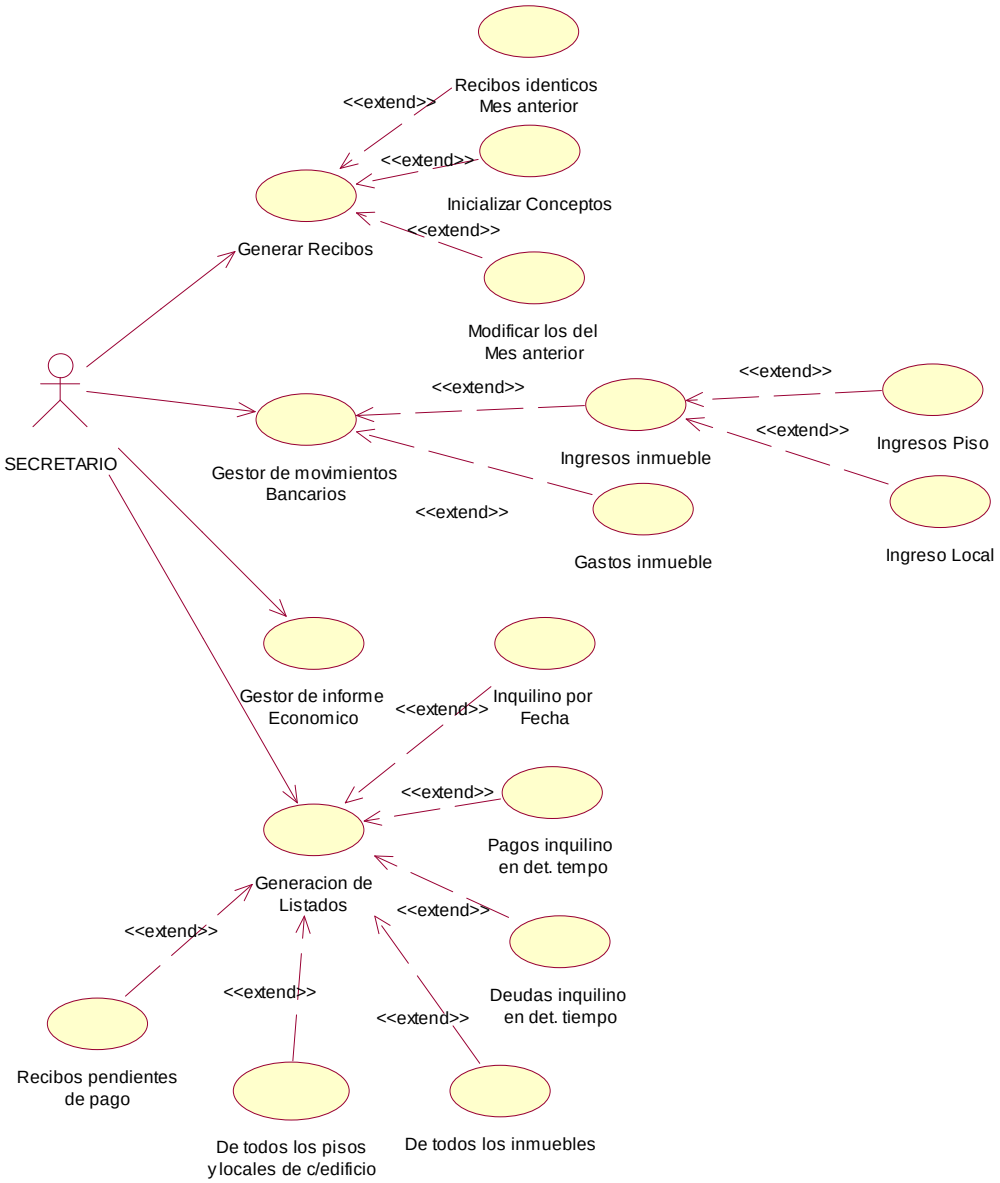
Este desglose se ha realizado para reflejar lo que el enunciado muestra con detalle y así poder tener una compresión mayo r de lo que el sistema debe de hacer.

Gestión de Movimientos bancarios: Tienen los siguientes casos de uso

- Ingresos
  - Ingresos de pisos
  - Ingresos de local
    Esto quiere decir que los ingresos pueden ser de pisos, de locales o ambos pero solo por esos conceptos.
- Gastos de inmuebles

Generación de listados: Se extiende en distintos casos de uso dependiendo del tipo de listado, la extensión refleja esos comportamiento opciones que puede haber en el sistema y que no tienen porque ser exclusivos, en el sentido de que si se realiza uno se pueden realizar los otros, cuando se generan los listados.

- Reporte inquilino por fecha
- Pagos de inquilino en un intervalo de tiempo
- Deudas de un inquilino en un intervalo de tiempo
- De todos los inmuebles
- De todos los pisos y locales de cada edificio
- Recibos pendientes de cobro



1.3.4. DIAGRAMAS DE SECUENCIA

Un diagrama de secuencia muestra las interacciones entre objetos ordenadas en secuencia temporal. Muestra los objetos que se encuentran en el escenario y la secuencia de mensajes intercambiados entre los objetos para llevar a cabo la funcionalidad descrita por el escenario. En aplicaciones grandes además de los objetos se muestran también los componentes y casos de uso. El mostrar los componentes tiene sentido ya que se trata de objetos reutilizables, en cuanto a los casos de uso hay que recordar que se implementan como objetos cuyo rol es encapsular lo definido en el caso de uso.

Para mostrar la interacción con el usuario o con otro sistema se introducen en los diagramas de secuencia las boundary classes. En las primeras fases de diseño el propósito de introducir estas clases es capturar y documentar los requisitos de interfaz, pero no el mostrar cómo se va a implementar dicha interfaz.

Los diagramas de secuencia o de interacción de objetos, se utilizan con frecuencia para validar los casos de uso. Documentan el diseño desde el punto de vista de los casos de uso. Observando qué mensajes se envían a los objetos, componentes o casos de uso y viendo a grosso modo cuanto tiempo consume el método invocado, los diagramas de secuencia nos ayudan a comprender los cuellos de botella potenciales, para así poder eliminarlos. A la hora de documentar un diagrama de secuencia resulta importante mantener los enlaces de los mensajes a los métodos apropiados del diagrama de clases

El diagrama de secuencia se lee de izquierda a derecha y de arriba abajo.

Normalmente cada caso de uso tiene asociados varios diagramas de secuencia. Uno que representa el curso típico de funcionamiento del caso de uso y uno o más por cada posible ejecución alternativa del caso de uso, es decir, si se producen errores.

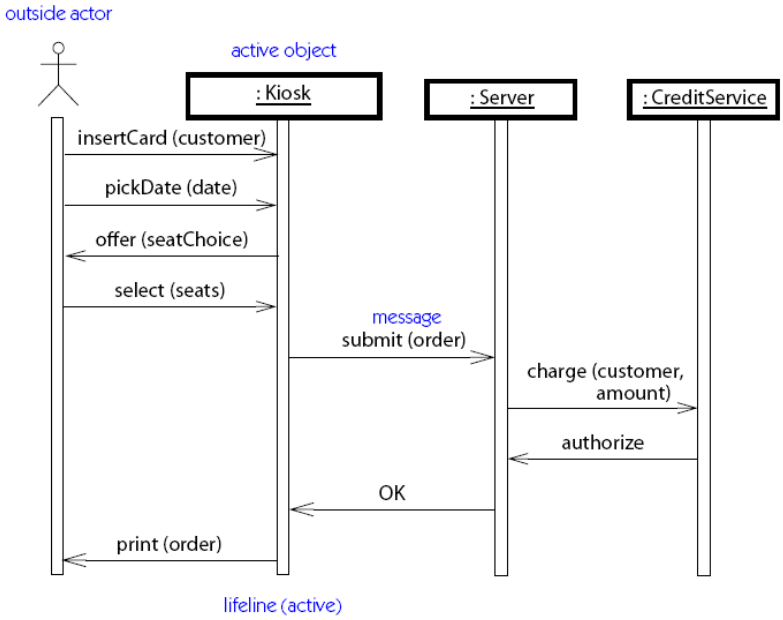
No es viable hacer todos los diagramas de secuencia posibles para cada caso de uso, pero si se recomienda hacer al menos el diagrama de secuencia que refleja el curso típico del caso de uso, ya que ayuda a comprender el funcionamiento del sistema.

Los diagramas de secuencia representan el comportamiento del sistema de una forma más amplia ya que en un caso de uso se pueden ver implicadas más de una clase.

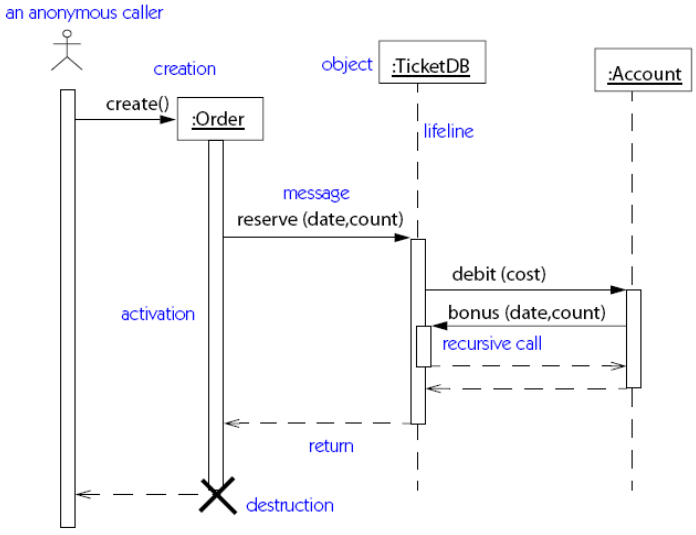
Los diagramas de secuencia son un tipo de diagramas de interacción. Se utilizan especialmente cuando se trata de sistemas en tiempo real. A diferencia de los diagramas de colaboración, que recordamos también son diagramas de secuencia pueden ser excesivamente grandes si intervienen muchas clases, lo cual hace que se pierda visibilidad; en estos casos se puede optar por los diagramas de colaboración que por su forma de representación son más reducidos.

Un objeto o una clase en el diagrama de secuencia se representa con un rectángulo en cuyo interior aparece el nombre de la nombre del objeto:clase.

Todas las clases involucradas en el diagrama de secuencia se construyen se colocan una al lado de otra. Debajo de cada clase se coloca una línea vertical. Entre las clases se pueden enviar mensajes, si se está en fase de análisis, o llamadas a métodos, si se está en fase de diseño.



Ejemplo de diagramas de secuencia



Ejemplo de diagramas de con activaciones

Elementos de los diagramas de secuencia

Línea de vida de un objeto (lifeline): La línea de vida de un objeto representa la vida del objeto durante la interacción. En un diagrama de secuencia un objeto se representa como una línea vertical punteada con un rectángulo de encabezado y con rectángulos a través de la línea principal que denotan la ejecución de métodos (activación). El rectángulo de encabezado contiene el nombre del objeto y el de su clase, en un formato nombreObjeto : nombreClase. Esta recta vertical es un lugar conveniente para sujetar mensajes entrantes y salientes, agregar múltiples líneas de vida en un solo diagrama y sujetar mensajes ordenados en el tiempo que permiten mostrar todas las clases las interacciones descritas

al igual que los mensajes necesarios para completar un escenario descrito por una caja de usos. Esto nos ayuda a la eliminación de situaciones ambiguas o evita la repetición de clases y mensajes, con este diagrama podemos obtener una solución completa, en un escenario a la vez.

Activación: Muestra el período de tiempo en el cual el objeto se encuentra desarrollando alguna operación, bien sea por sí mismo o por medio de delegación a alguno de sus atributos. Se denota como un rectángulo delgado sobre la línea de vida del objeto. Gracias a esta denotación podemos mostrar la duración que tienen los objetos, hay que poner atención de cuando empezamos a usar un objeto y cuando terminamos de usarlo, a menos que este represente un recurso finito. En ambos casos el símbolo de activación representa la amplitud de la duración de un objeto. También es importante saber que un objeto puede representar como fue creado y destruido con el uso de una sola línea de vida. El símbolo de activación es un rectángulo vertical que reemplaza la línea de vida en el transcurso de la duración de la existencia de ese caso.

Mensaje: El envío de mensajes entre objetos se denota mediante una línea sólida dirigida, desde el objeto que emite el mensaje hacia el objeto que lo ejecuta. La línea se inicia en una línea de vida y la flecha apunta hacia aquella línea de vida que contenga el mensaje invocado. El mensaje puede empezar y finalizar en la misma línea de vida; a esto se le conoce como *llamada anidada* .Se usa una línea punteada para los mensajes de retorno. Incluidos como mensajes posibles, se encuentran los mensajes hallados y los perdidos. Un mensaje hallado tiene un receptor conocido, pero el emisor se conoce; uno perdido tiene un emisor conocido pero no receptor especificado. El triangulo relleno (punta de flecha) representa un mensaje de sincronización que llama a otro procedimiento; un triangulo de palillos representa un mensaje de sincronización simple.

Tiempos de transición: En un entorno de objetos concurrentes o de demoras en la recepción de mensajes, es útil agregar nombres a los tiempos de salida y llegada de mensajes.

Caminos alternativos de ejecución y concurrencia: En algunos casos sencillos los caminos alternativos pueden expresarse en un diagrama de secuencias alternativas de ejecución. Estas alternativas pueden representar condiciones en la ejecución o diferentes hilos de ejecución (threads).

Destrucción de un objeto: Se representa como una X al final de la línea de ejecución del objeto.

Ejemplo:

Administración de Bibliotecas

Se requiere se realice el diseño de un sistema de bibliotecas en donde se encuentran las siguientes restricciones. La biblioteca tiene libros y revistas, donde puede haber varias copias de un libro determinado. Algunos libros son solo para préstamos a corto plazo. Todos los demás libros pueden ser prestados a cualquier socio de la biblioteca durante 3 semanas. Solo los socios del personal pueden tomar prestadas las revistas. Los socios de la biblioteca normalmente pueden tomar prestados hasta seis artículos de una sola vez mientras que los socios del personal pueden tomar prestados hasta doce artículos de una sola vez. Regularmente llegan nuevos libros y revista y a veces hay que deshacerse

de los antiguos. Las revistas del año actualmente se envían para encuadernarlas en volúmenes al finalizar el año.

En cuestión del sistema de préstamos es esencial que el sistema lleve un control de cuando se prestan y devuelven los libros y las revistas, ya que el sistema actual lo hace. El nuevo sistema debería avisar cuando un libro ha sido prestado, pero no devuelto. En un futuro se puede requerir que los usuarios puedan ampliar el préstamo de un libro si no está reservado. Un prestatario libro presenta un libro. El sistema comprueba que el prestatario es socio de la biblioteca y que él o ella no tenga ya el máximo numero de libros permitidos. El máximo es de 6 a menos que el socio sea un miembro del personal, cuyo caso es doce. Si ambas comprobaciones son correctas el sistema almacenara que este socio de la biblioteca tiene esta copia del libro en préstamo. En caso contrario se rechaza el préstamo.

El sistema deberá permitir a los usuarios buscar un libro por tema, por autor, etc. Para comprobar si hay una copia del libro disponible para ser prestado, y si no, lo reserva.

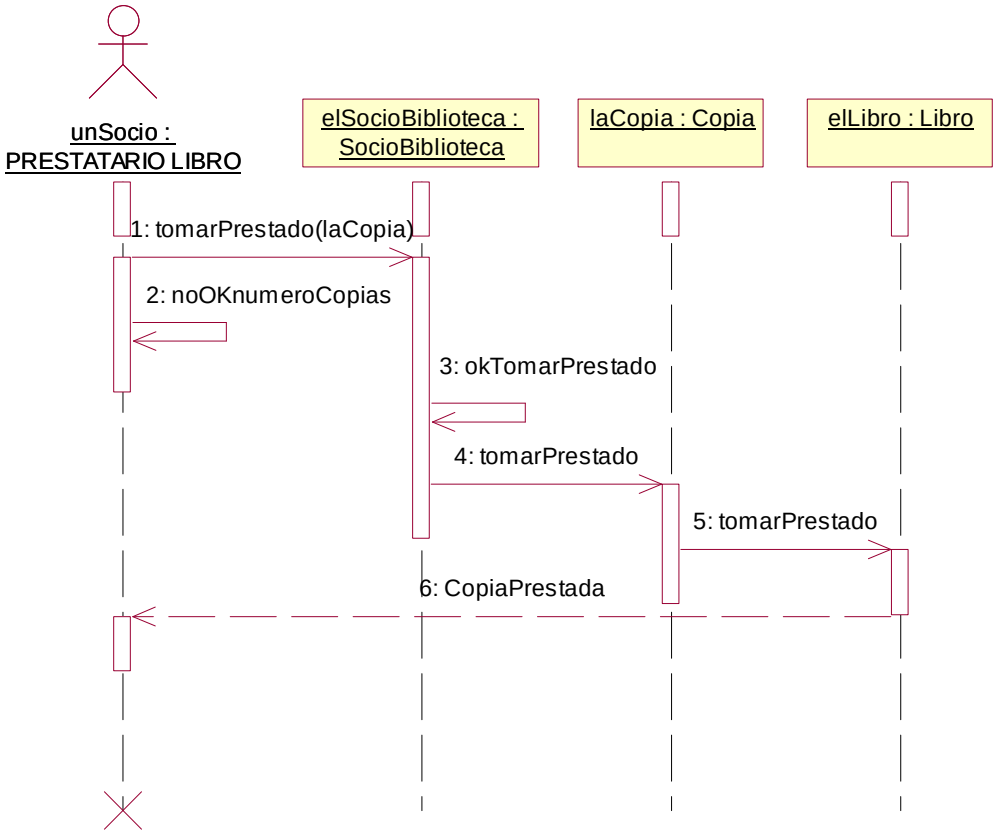
Cualquier persona puede hojear los libros en la biblioteca.

Realizar Diagrama de Secuencia para resolver este problema.

PASOS PARA LA ELABORACION DE ESTE DIAGRAMA

1. Colocar los objetos que participan en la interacción en la parte de arriba del diagrama, a través del eje de las X.
2. Colocar los objetos que inician la interacción a la izquierda y los objetos más subordinados a la derecha.
3. Colocar los mensajes que estos objetos envían y reciben junto al eje de las Y, en orden de incremento de tiempo de arriba hacia abajo.
4. Ubicar la línea de vida del objeto, que representa la existencia de un objeto en un período de tiempo.
5. Ubicar el foco de control, que muestra el período de tiempo en el que el objeto se encuentra representando una acción.

DIAGRAMA DE SECUENCIA



Este diagrama surge a partir de los siguientes diagramas

Diagrama de Casos de Uso

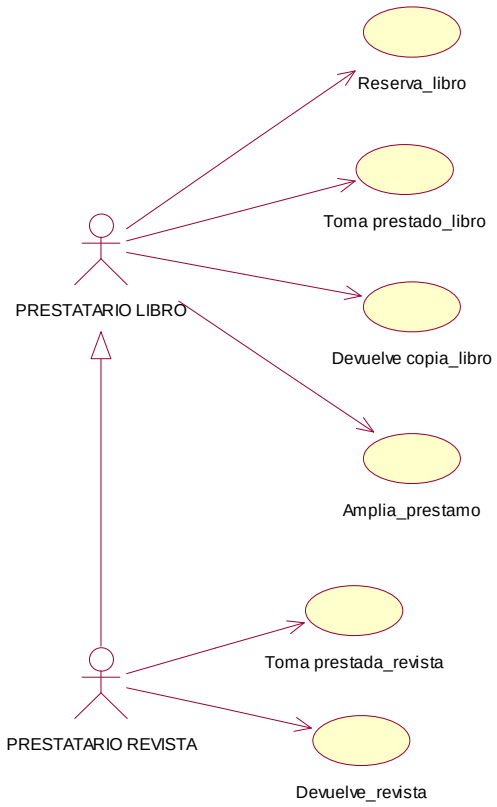


Diagrama de Clases

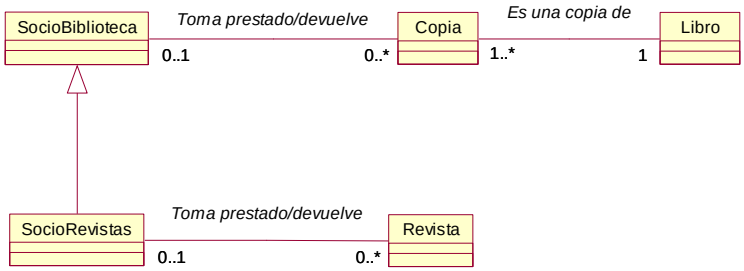
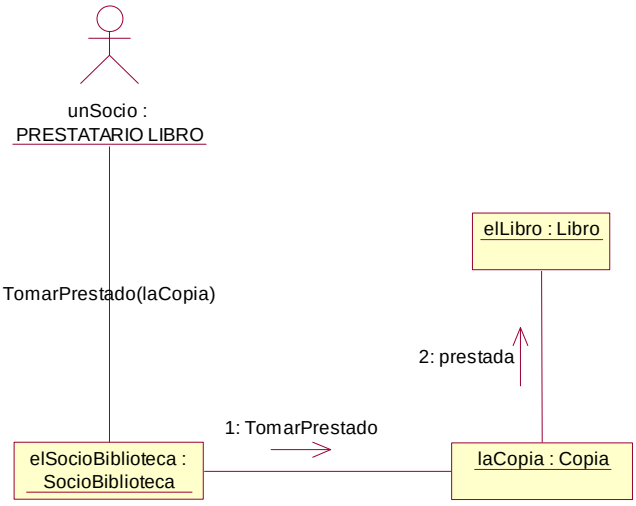


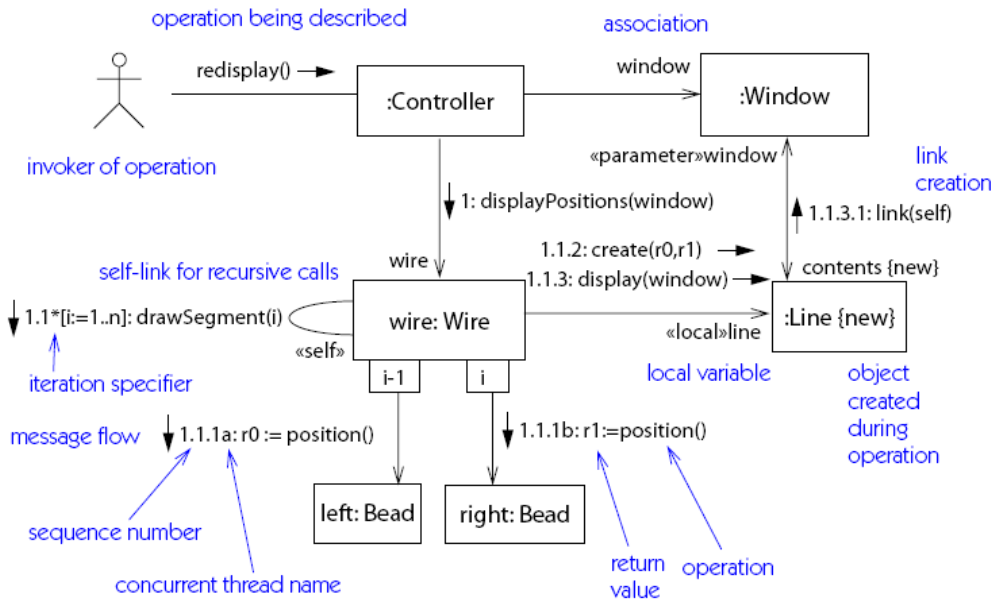
Diagrama de Objeto



1.3.5. DIAGRAMA DE COLABORACION

Un diagrama de colaboración es una forma alternativa al diagrama de secuencia de mostrar un escenario. Este tipo de diagrama muestra las interacciones entre objetos organizadas entorno a los objetos y los enlaces entre ellos.

Los diagramas de secuencia proporcionan una forma de ver el escenario en un orden temporal - qué pasa primero, qué pasa después -. Los clientes entienden fácilmente este tipo de diagramas, por lo que resultan útiles en las primeras fases de análisis. Por lo contrario los diagramas de colaboración proporcionan la representación principal de un escenario, ya que las colaboraciones se organizan entorno a los enlaces de unos objetos con otros. Este tipo de diagramas se utilizan más frecuentemente en la fase de diseño, es decir, cuando estamos diseñando la implementación de las relaciones.

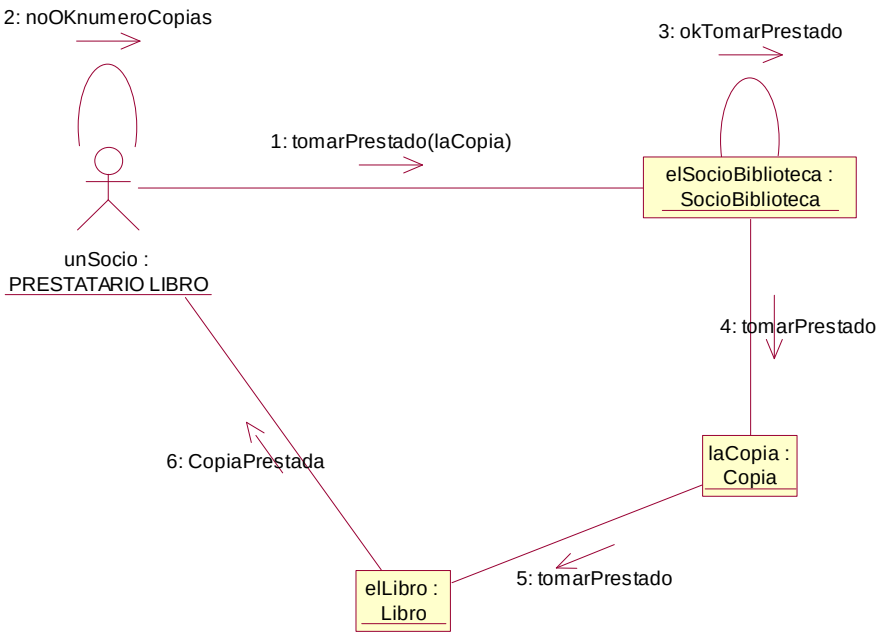


EJEMPLO:

CONTINUAREMOS CON EL CASO DE LA BIBLIOTECA

PASOS PARA LA ELABORACION DE ESTE DIAGRAMA

1. Colocar los objetos que participan en la interacción como los vértices en una gráfica.
2. Interpretar las ligas que conectan a estos objetos.
3. Adornar estas ligas con los mensajes que los objetos envían y reciben.
4. Establecer una ruta, para indicar como un objeto es ligado a otro. Podemos unirle un estereotipo al final o al inicio de una liga.
5. Establecer un número de secuencia, para indicar el orden de tiempo de un mensaje. Éste debe ser único.



1.3.6 DIAGRAMA DE ESTADOS

El diagrama de transición de estados es un grafico compuesto básicamente de estados y transiciones entre estos.

Cada diagrama de transición de estados:

Si se asocia a una clase:

Describe la forma en que una instancia de una clase reaccionara ante los eventos que recibe.

Si se asocia a un caso de uso:

Describe la forma en que funcionara ese caso de uso cuando se ejecute en el sistema

Este diagrama es especialmente útil para reflejar el ciclo de vida de clases que tienen una complejidad media, lo que significa que no es relevante para clases que tengan un ciclo de vida muy simple, ni para clases con ciclos de vida excesivamente complejos.

El diagrama de estados y transiciones engloba todos los mensajes que un objeto puede enviar o recibir.

ELEMENTOS DE UN DIAGRAMA DE TRANSICION DE ESTADOS

EVENTO: Es una ocurrencia que tiene lugar en un momento preciso del tiempo, pero que no tiene duración. Suele representarse como un mensaje que una clase envía a otra del sistema.

ESTADO: Es el periodo de tiempo que tiene lugar entre la recepción de dos eventos consecutivos por parte de una clase del sistema. Durante el tiempo que dura un estado pueden ocurrir varias cosas:

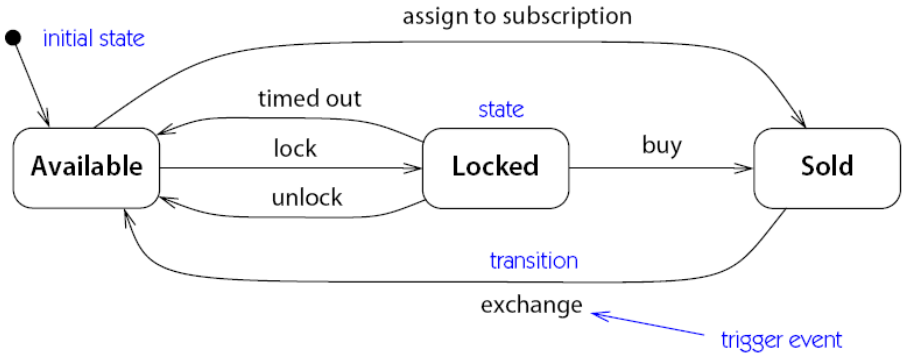
Una acción a la entrada del estado
Una acción durante el tiempo que dura el estado
Una acción al salir de dicho estado.

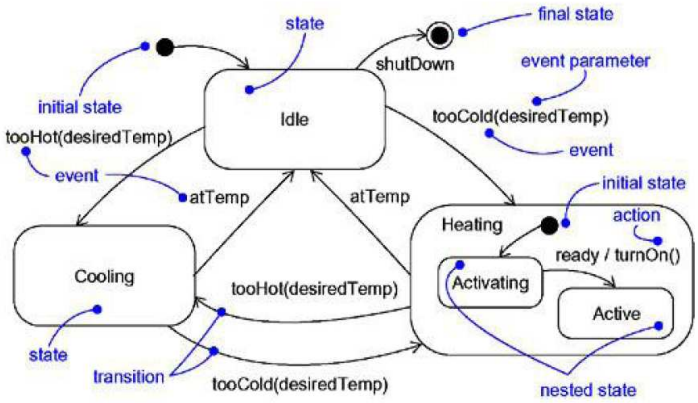
Es una condición durante la vida de un objeto, de forma que cuando dicha condición se satisface se lleva a cabo alguna acción o se espera por un evento. El estado de un objeto se puede caracterizar por el valor de uno o varios de los atributos de su clase, además, el estado de un objeto también se puede caracterizar por la existencia de un enlace con otro objeto.

TRANSICION: Una transición tiene lugar entre dos estados en los que se puede encontrar una clase. Puede tener una acción y/o una condición asociada a la transición. Además puede disparar un Evento.

ACCION: Es un comportamiento que ocurre, cuando tiene lugar una transición entre estados.

CONDICION DE GUARDA: Es una expresión boolena de valores de atributos que permiten que se produzca una transición de estados solo si se cumple la condición.





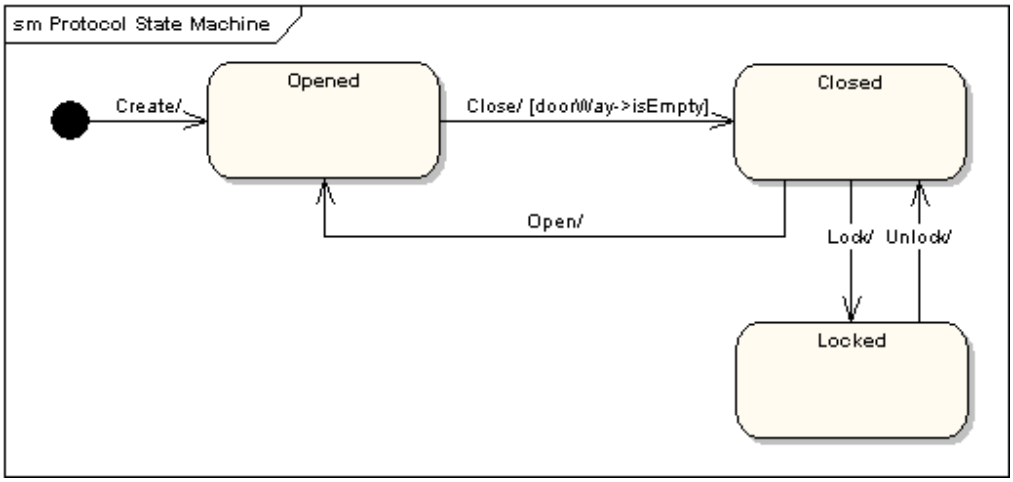
ESCENARIO: Representa un camino dentro del diagrama. Dado que generalmente el intervalo entre dos envíos de mensajes representa un estado, se pueden utilizar los diagramas de secuencia para buscar los diferentes estados de un objeto.

En todo diagrama de estados existen por lo menos dos estados especiales inicial y final: start y stop. Cada diagrama debe tener uno y sólo un estado start para que el objeto se encuentre en estado consistente. Por contrario, un diagrama puede tener varios estados stop.

Diagrama de Máquina de Estados

Un diagrama de máquina de estado modela el comportamiento de un solo objeto, especificando la secuencia de eventos que un objeto atraviesa durante su tiempo de vida en respuesta a los eventos.

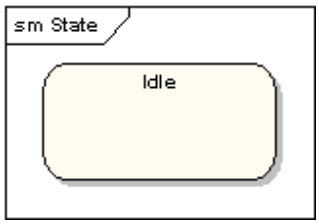
Como ejemplo, el siguiente diagrama de máquina de estado muestra los estados que una puerta atraviesa durante su tiempo de vida.



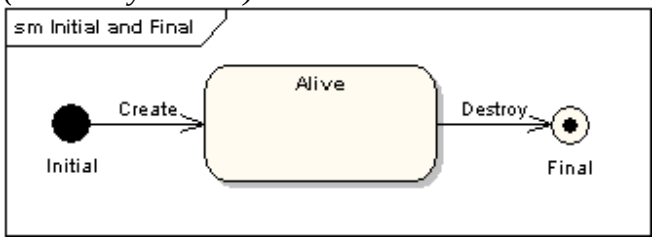
La puerta puede estar en uno de tres estados: “Opened”(Abierta), “Closed” (Cerrada) o “Locked”(Bloqueada). Puede responder a tres estados Abrir, Cerrar, Bloquear y No bloqueado. Tener en cuenta que no todos los eventos son válidos en todos los estados: por ejemplo, si una puerta está abierta, no lo puede bloquear hasta que lo cierre. También tener en cuenta de que como una transición de estado puede tener una condición de guarda adjunta. Si la puerta está abierta, esta solo puede responder al Evento cerrar si la condición doorWay->isEmpty esta completa.

Estados

Un estado se denota por un rectángulo con las esquinas redondeadas y con el nombre del estado escrito dentro del mismo.



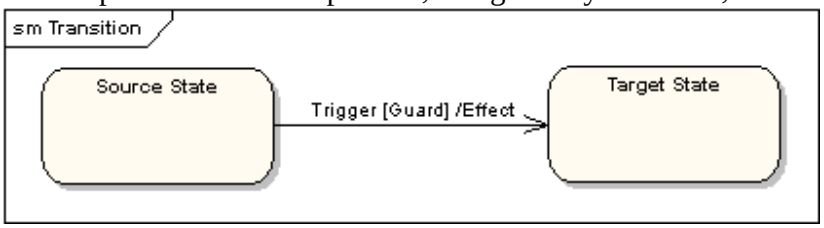
Estados Initial y Final (Iniciales y Finales)



El estado inicial se denota con un círculo negro y se le puede proporcionar un nombre. El estado final se denota con un círculo con un punto negro en el medio y también se lo puede nombrar.

Transiciones

Las transiciones desde un estado al siguiente se denotan por líneas con flechas. Una transición puede tener un disparador, una guarda y un efecto, como a continuación.



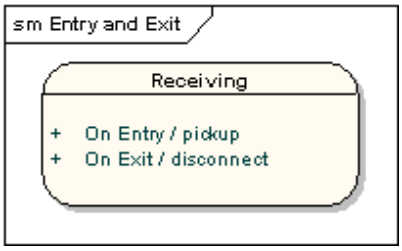
“Trigger” (Disparador) es la causa de la transición, la cual podría ser una señal, un evento, un cambio en alguna condición, o el pasaje de tiempo.

"Guard" (guarda) es una condición que debe ser verdadera para que el disparador cause la transición.

"Effect" (efecto) es una acción que se llamará directamente en el objeto que tiene la maquina de estado como resultado de la transición.

Acciones de Estado

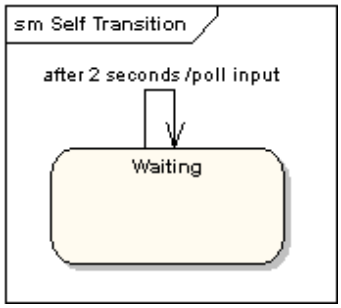
En el ejemplo de transición, un efecto se asoció con la transición. Si el estado de destino tenía muchas transiciones llegando al mismo, y cada transición tenía el mismo efecto asociado con este, sería mejor asociar el efecto con el estado de destino en lugar de con las transiciones. Esto se puede realizar para definir una acción de entrada para el estado. El siguiente diagrama muestra un estado con una acción de entrada y una acción de salida.



También es posible definir las acciones que ocurren en los eventos, o acciones que siempre ocurren. Es posible definir cualquier número de acciones de cada tipo.

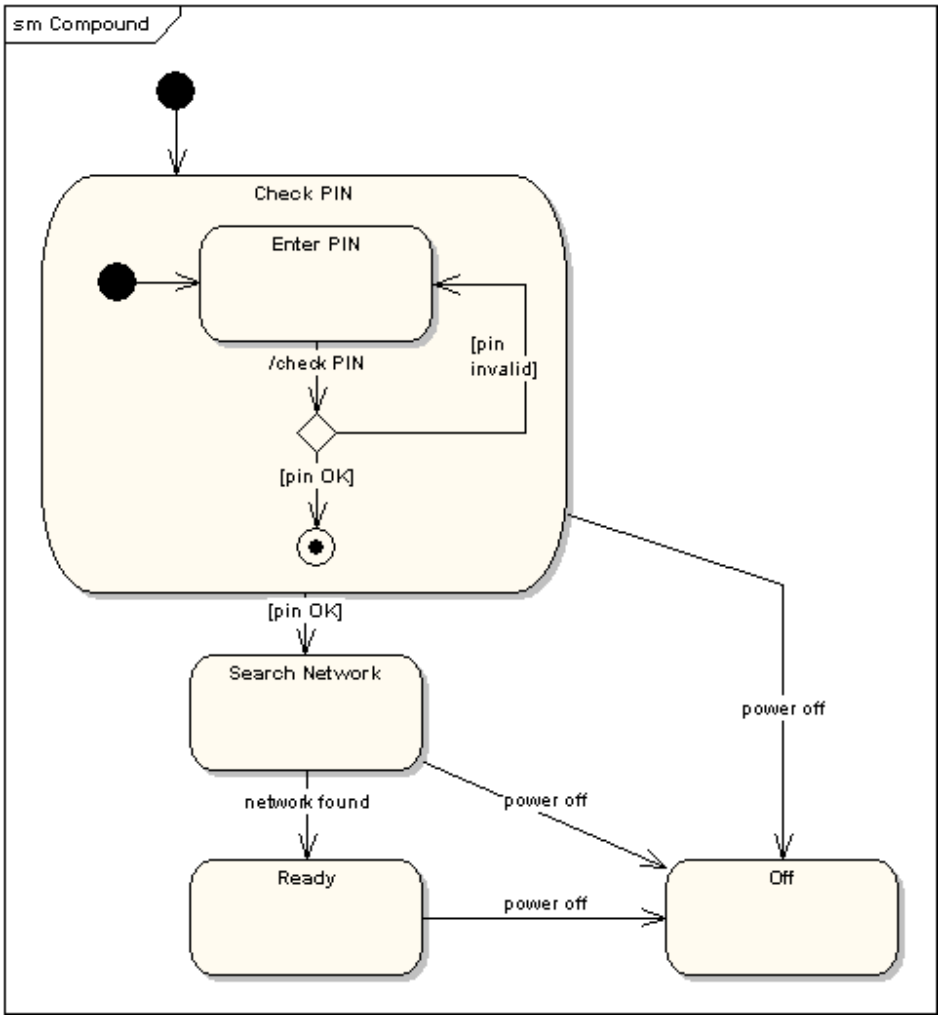
Transiciones recursivas

Un estado puede tener una transición que retorna a sí misma, como en el siguiente diagrama. Esto es más útil cuando un efecto se asocia con la transición.

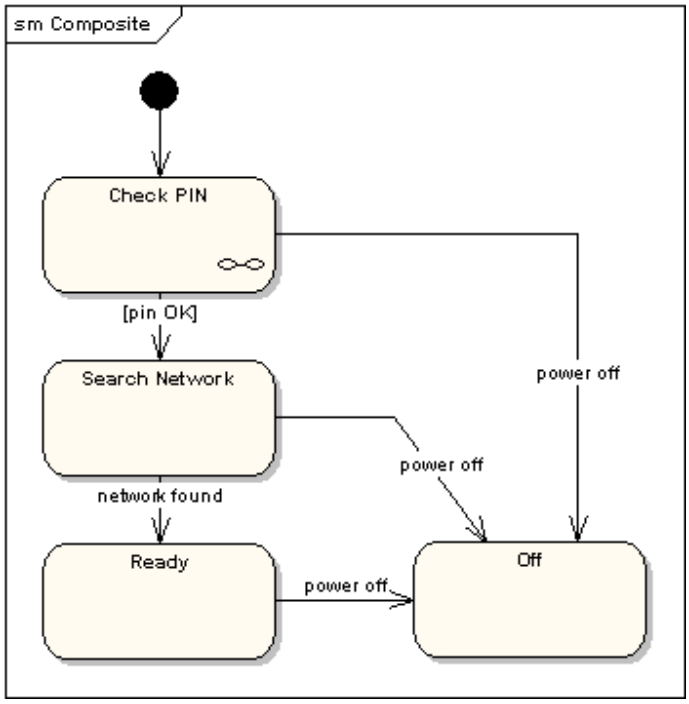


Estados Compuestos

Un diagrama de máquina de estado puede incluir diagramas de sub maquinas, como en el siguiente ejemplo.



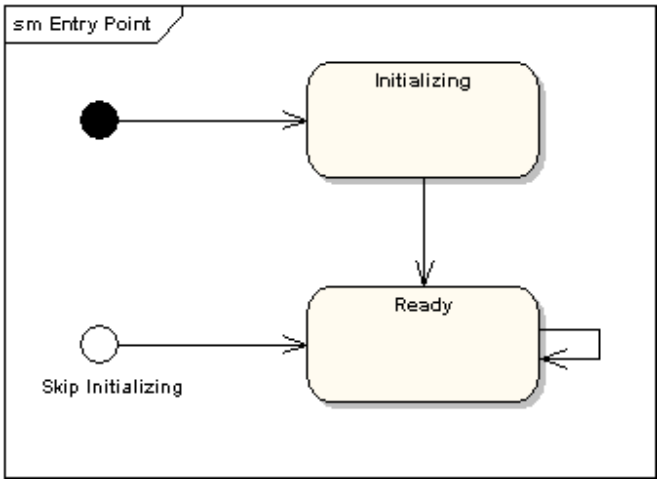
La forma alternativa de mostrar la misma información es como el siguiente ejemplo.



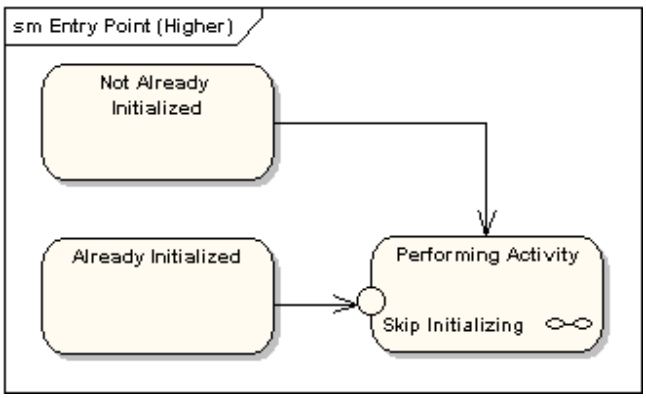
La notación en la versión anterior indica que los detalles de la sub maquina Check Pin se muestran en un diagrama separado.

Punto de Entrada

Algunas veces no deseará ingresar una sub maquina en un Estado Inicial normal. Por ejemplo, en la siguiente sub maquina sería normal comenzar en el estado inicial, pero si por alguna razón no fuera necesario realizar la inicialización, sería posible comenzar en el estado Ready realizando una transición al punto de entrada nombrado.

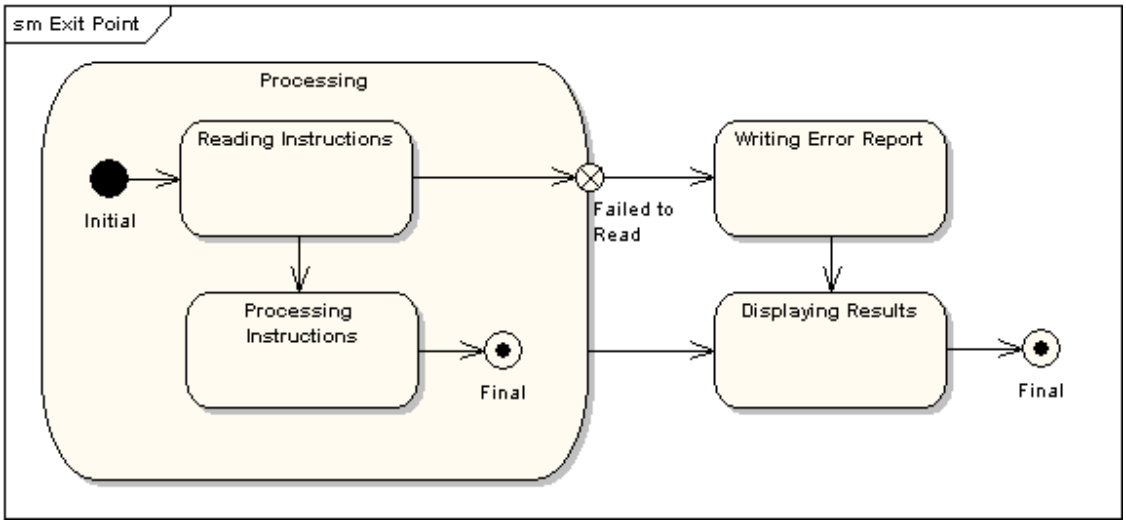


El siguiente diagrama muestra la máquina de estado un nivel hacia arriba:



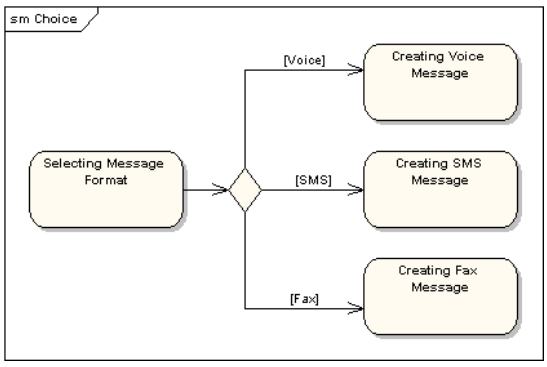
Punto de Salida

Similar al Punto de Entada, es posible nombrar Puntos de Salida nombrados. El siguiente diagrama provee un ejemplo donde el estado ejecutado después del estado de procesos principal depende de que ruta se use para realizar la transición del estado.



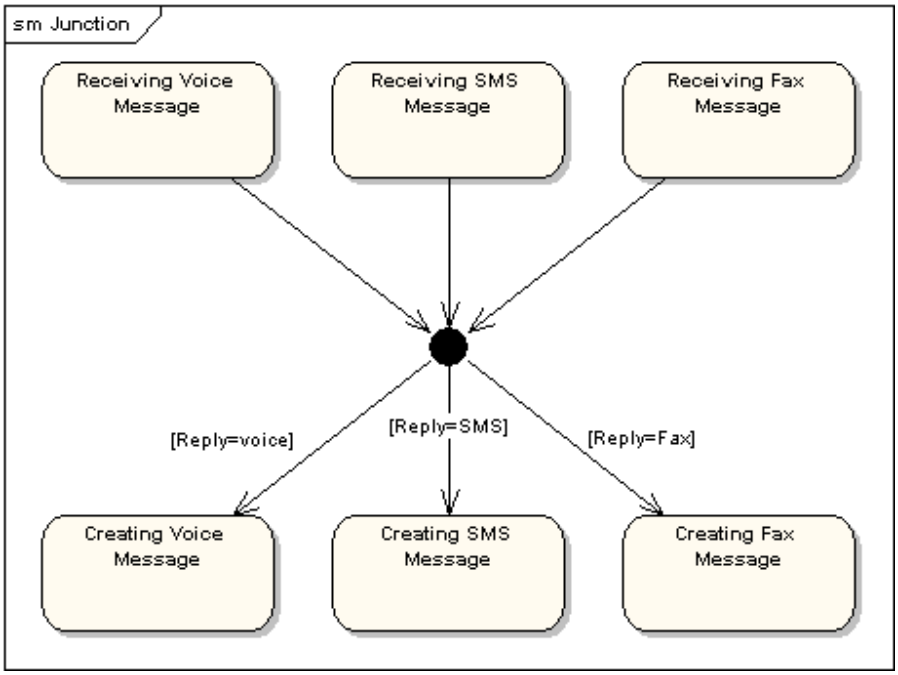
Pseudo estado "Choice" (Elección)

Un pseudo estado se muestra como un diamante con una transición llegando y dos o más transiciones saliendo. El siguiente diagrama muestra que cualquier estado al que se llega después del pseudo estado elección depende del formato del mensaje seleccionado durante la ejecución del estado anterior.



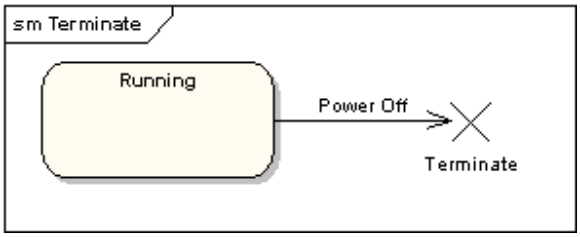
Pseudo estado "Junction" (unión)

Los pseudo estados unión se usan para unir transiciones múltiples. Una sola unión puede tener una o más transiciones de entradas y una o más de salida, y se puede aplicar una guarda a cada transición. Las uniones son libres de semántica; una unión que divide una transición de entrada en transiciones de salida múltiples realiza una rama condicional estática, opuesto a un pseudo estado elección que realiza una rama condicional dinámica.



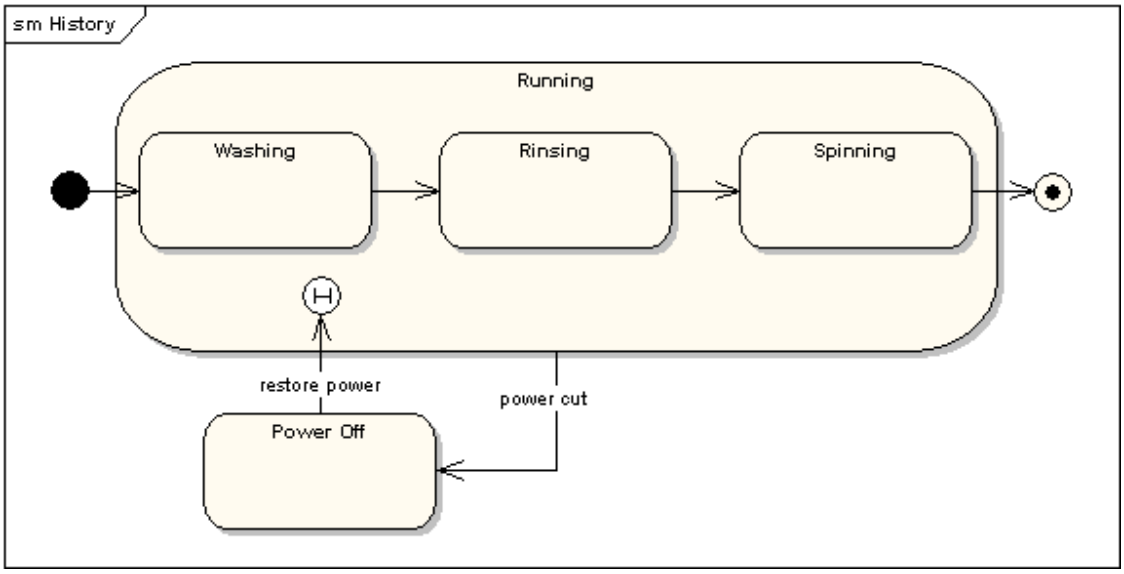
Pseudo estado "Terminate" (terminar)

Ingresar un pseudo terminar indica que la línea de vida de la maquina de estado ha terminado. Un pseudo estado indica que una línea de vida de la maquina de estado ha terminado. Un pseudo estado terminar se denota como una cruz.



Estado "History" (Historial)

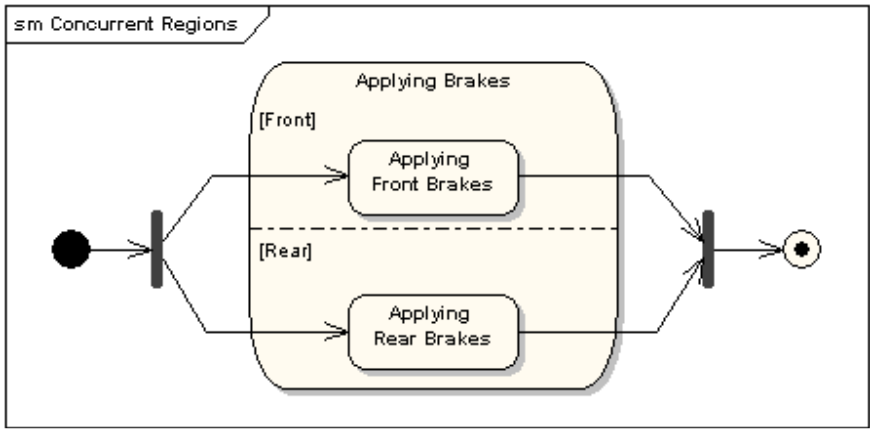
Un estado historial se usa para recordar el estado anterior de una maquina de estado cuando fue interrumpida. El siguiente diagrama ilustra el uso de estados del historial. El ejemplo es una maquina de estado que pertenece a un lavarropas.



En esta maquina de estado, cuando un lavarropas comience, su proceso será desde el lavado y enjuague hasta el secado. Si hay un corte de luz, el lavarropas se detendrá por lo que pasará al estado Power off (apagado). Luego, cuando la energía retorne, el estado Running (ejecutar) ingresa al símbolo de estado Historial, lo que significa que debería seguir con el proceso donde quedó cuando se corto la energía.

Regiones recientes

Un estado se puede dividir en regiones conteniendo sub estados que existen y se ejecutan concurrentemente. El siguiente ejemplo muestra que dentro del estado “Applying Brakes” (Aplicar frenos), los frenos de adelante y atrás estarán operando simultáneamente e independientemente. Tener en cuenta el uso de los pseudo estados en lugar de los pseudo estados elección y combinación. Estos símbolos se usan para sincronizar los hilos concurrentes.



PASOS PARA LA ELABORACION DE ESTE DIAGRAMA

1. Identificar el inicio del problema.
2. Identificar las actividades que se van a diagramar.
3. Encontrar las diferentes situaciones en las que se encuentran las actividades y una vez encontradas ir numerándolas a un lado dependiendo la actividad realizada.
4. Definir la finalización.
5. Ir diagramando a partir del inicio en forma vertical y dependiendo el estado en que se encuentre se diagramara en forma horizontal.

EJEMPLO CONTINUAMOS CON EL DE LA BIBLIOTECA:

PROBLEMA 1: Si se activa el disparador solicitar_libro, si la guarda dice que está disponible, entonces el estado cambia a reservado.

PROBLEMA 2: En el sistema de la biblioteca un objeto de la clase Copia puede tener un atributo lógico en los Estantes, que se pretende que almacene si el objeto de una copia de un libro está actualmente en la biblioteca, o uno que está en préstamo. La interfaz de la clase copia especifica que el objeto puede aceptar el mensaje tomarPrestado(). Se pretende que informe al objeto Copia, que la copia real acaba de ser tomada prestada de la biblioteca. Este mensaje solo debería llegar cuando el atributo Estante es cierto; si la copia real ha sido tomada prestada, no debe estar en la biblioteca. La reacción del objeto Copia debería ser poner el atributo en el estante a falso aquí se tiene que marcar el camino con el aspecto real que se supone que tiene que describir y enviar un mensaje al objeto Libro asociado con él para informarle de que esa copia acaba de ser tomada prestada.

Solución problema 1

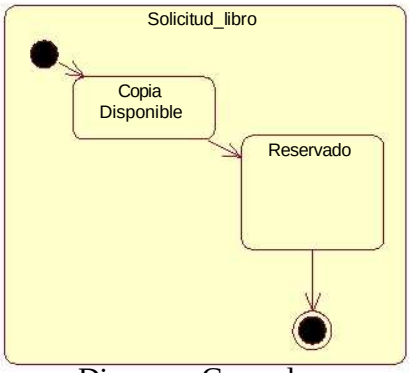


Diagrama General

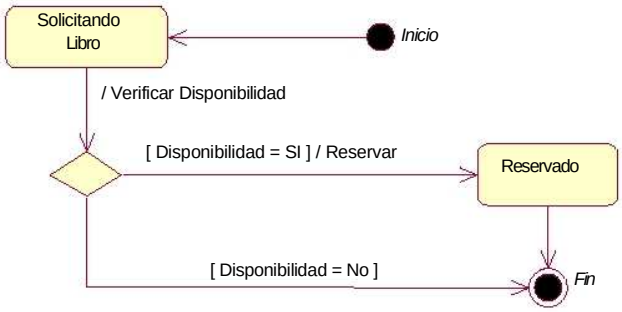
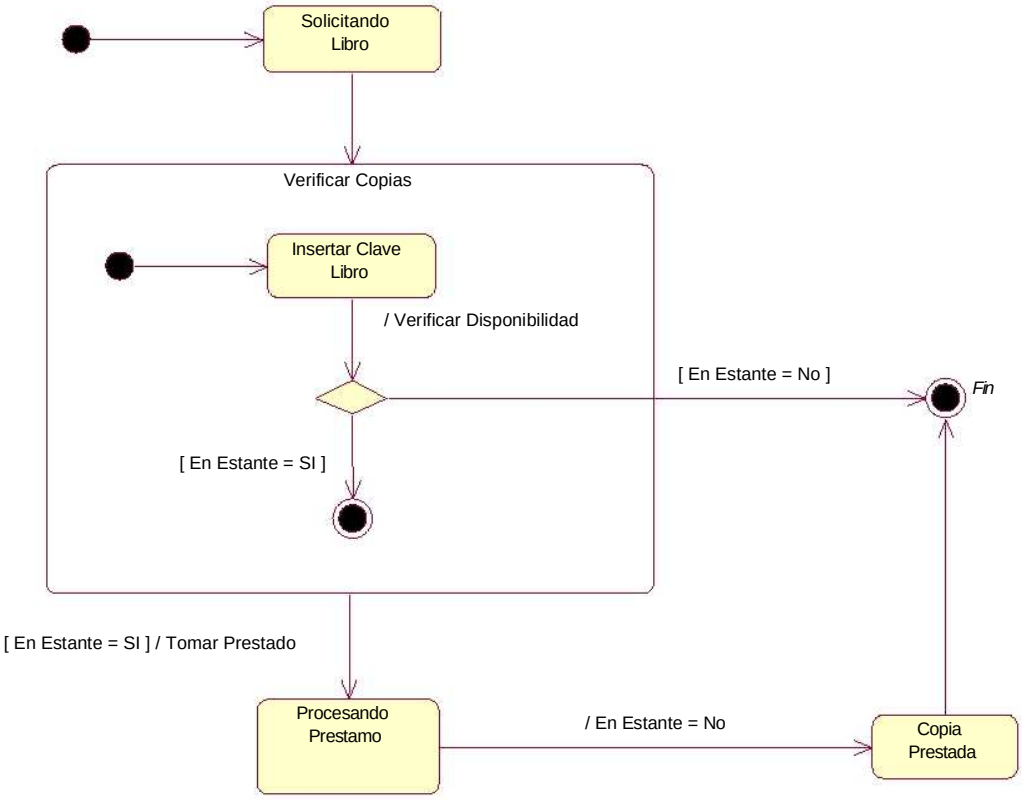


Diagrama Especifico

Si se activa el disparador solicitar_libro, si la guarda dice que está disponible, entonces el estado cambia a reservado.

Solución problema 2

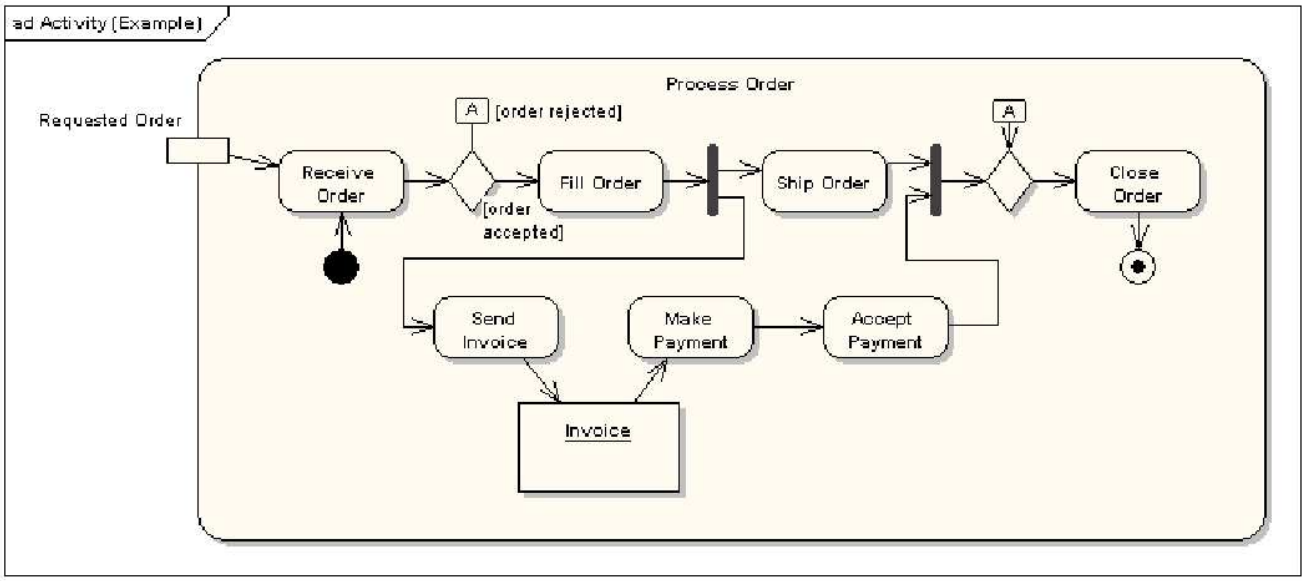


3.7 DIAGRAMA DE ACTIVIDADES

En UML un diagrama de actividades se usa para mostrar la secuencia de actividades. Los diagramas de actividades muestran el flujo de trabajo desde el punto de inicio hasta el punto final detallando muchas de las rutas de decisiones que existen en el progreso de eventos contenidos en la actividad. Estos también pueden usarse para detallar situaciones donde el proceso paralelo puede ocurrir en la ejecución de algunas actividades. Los Diagramas de Actividades son útiles para el Modelado de Negocios donde se usan para detallar el proceso involucrado en las actividades de negocio.

También describen como se coordinan las actividades se utiliza para indicar como podría implementarse una operación. Es útil cuando se sabe que una operación tiene que alcanzar un numero de cosas distintas y se quiere modelar cuales son las dependencias fundamentales entre ellas.

Son útiles para describir como se expresan los casos de uso individuales y pueden depender de otros casos de usos. Muchas veces los casos de uso que descubren no suceden en un orden arbitrario sino como parte del flujo de trabajo general de un área de las actividades de un cliente.

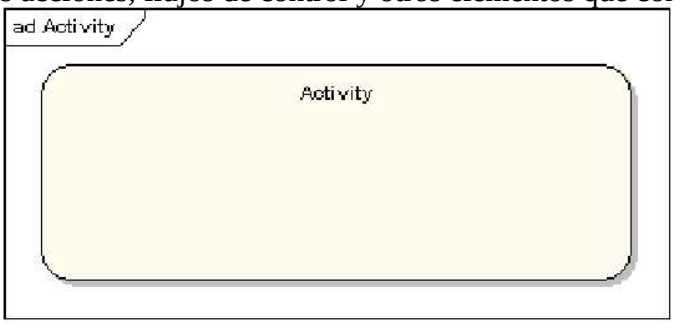


Ejemplo de un diagrama de actividades.

ELEMENTOS DEL DIAGRAMA DE ACTIVIDADES

Actividades

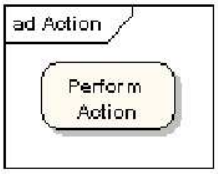
Una actividad es la especificación de una secuencia parame rizada de comportamiento. Una actividad muestra un rectángulo con las puntas redondeadas adjuntando todas las acciones, flujos de control y otros elementos que constituyen la actividad.



Representación en UML de una actividad

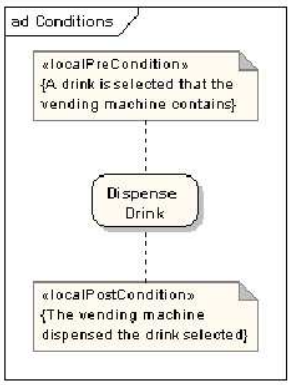
Acciones

Una acción representa un solo paso dentro de una actividad. Las acciones se denotan por rectángulos con las puntas redondeadas.



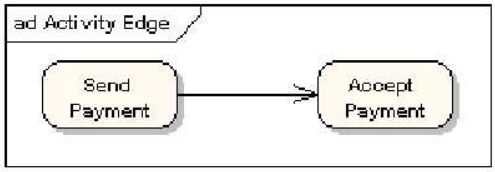
Restricciones de Acción.

Las restricciones se pueden adjuntar a una acción. El siguiente diagrama muestra una acción con pre y post condiciones locales.



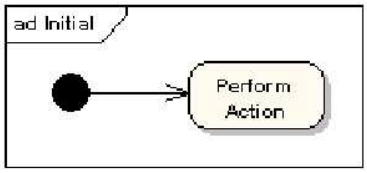
Flujo de Control

Un flujo de control muestra el flujo de control de una acción a otra. Su notación es una línea con una punta de flecha.



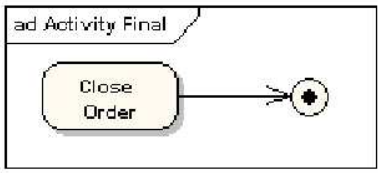
Nodo Inicial

Un nodo inicial o de comienzo se describe por un gran punto negro, como se muestra a continuación.

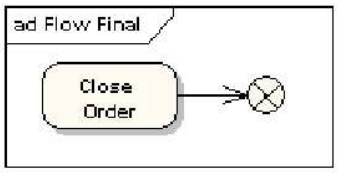


Nodo Final

Hay dos tipos de nodos finales: nodos finales de actividad y de flujo. El nodo final de actividad se describe como un círculo con un punto dentro del mismo.



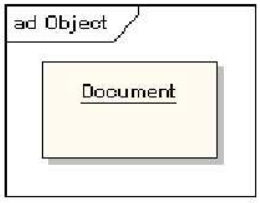
El nodo final de flujo se describe como un círculo con una cruz dentro del mismo.



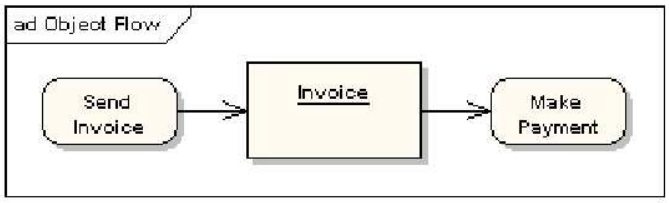
La diferencia entre los dos tipos de nodos es que el nodo final del flujo denota el final de un solo flujo de control, y el nodo final de actividad denota el final de todos los flujos finales dentro de la actividad.

Flujos de Objetos y Objeto

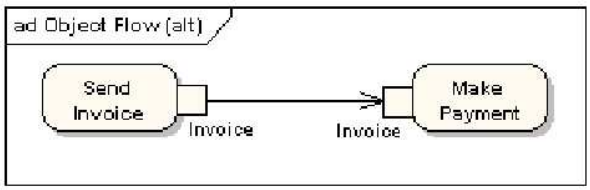
Un flujo de objeto es la ruta a lo largo de la cual pueden pasar objetos o datos. Un objeto se muestra cómo un rectángulo.



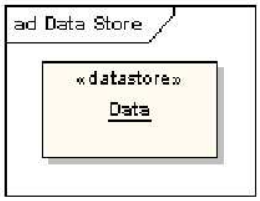
Un flujo de objeto se muestra como un conector con una punta de flecha denotando la dirección a la cual se está pasando el objeto.



Un flujo de objeto debe tener un objeto en por lo menos uno de sus extremos. Una notación de acceso rápido para el diagrama de arriba sería usar los pins de salidas y entradas.

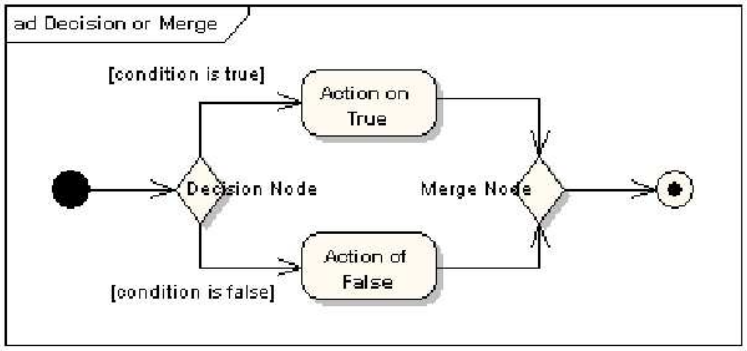


Un almacén de clave se muestra como un objeto con las clave «datastore».



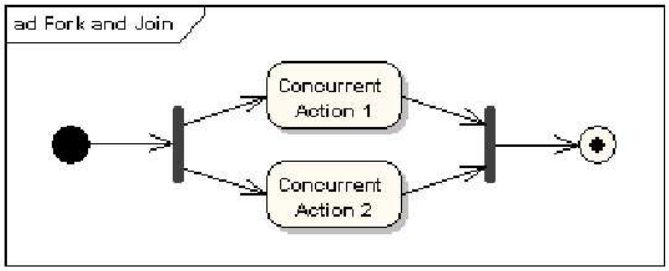
Nodos de Decisión y Combinación

Los nodos de decisión y combinación tienen la misma notación: una forma de diamante. Los dos se pueden nombrar. Los flujos de control que provienen de un nodo de decisión tendrán condiciones de guarda que permitirán el control para fluir si la condición de guarda se realiza. El siguiente diagrama muestra el uso de un nodo de decisión y un nodo de combinación.



Nodos de Bifurcación y Unión.

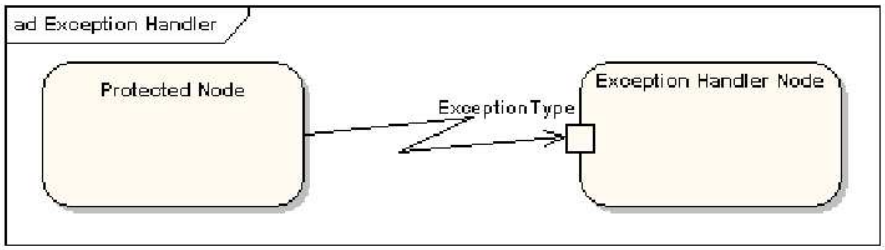
Las bifurcaciones y uniones tienen la misma notación: tanto una barra horizontal como vertical (la orientación depende de si el flujo de control va de derecha a izquierda o hacia abajo y arriba. Estos indican el comienzo y final de hilos actuales de control. El siguiente diagrama muestra un ejemplo de su uso.



Una unión es diferente de una combinación ya que la unión sincroniza dos flujos de entrada y produce un solo flujo de salida. El flujo de salida desde una unión no se puede ejecutar hasta que todos los flujos se hayan recibido. Una combinación pasa cualquier flujo de control directamente a través de esta. Si dos o más flujos de entrada se reciben por un símbolo de combinación, la acción a la que el flujo de salida apunta se ejecuta dos o más veces.

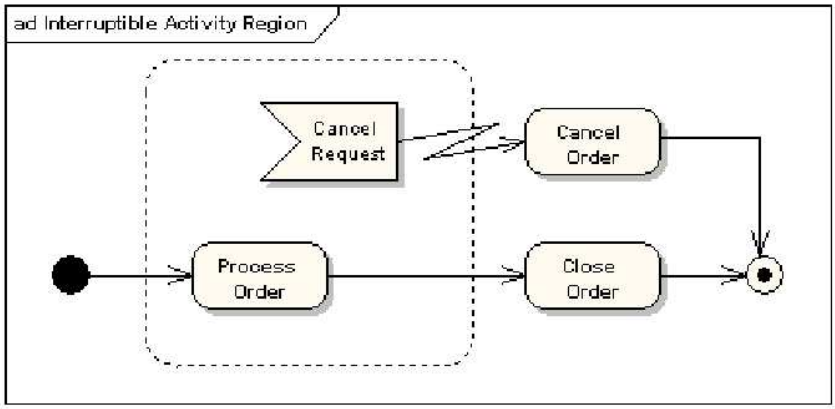
Gestores de Excepción.

Los gestores de Excepción se pueden modelar en diagramas de actividad como en siguiente ejemplo.



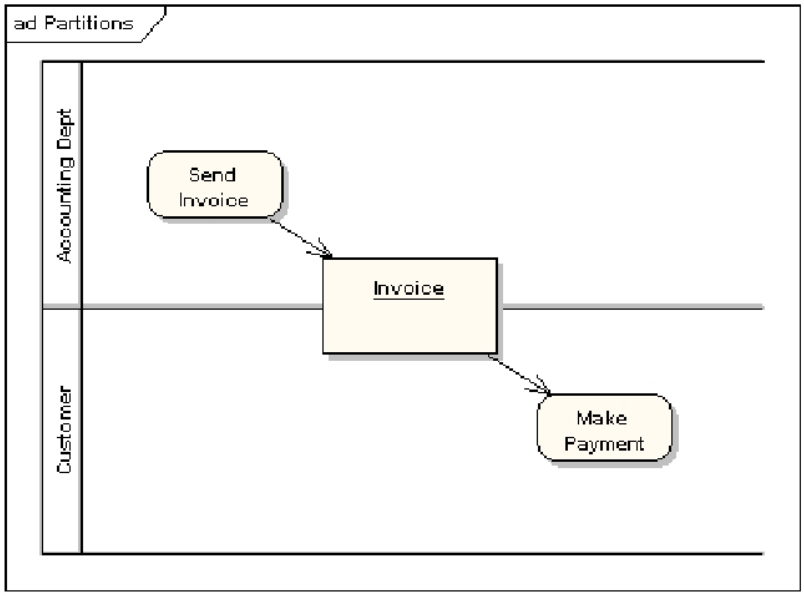
Región de Actividad Interrumpible

Una región de actividad interrumpible rodea un grupo de acciones que se pueden interrumpir. En un ejemplo simple como el siguiente, la acción Procesar Orden se ejecutará hasta su cumplimiento cuando pase control a la acción Cerrar Orden, a menos que una interrupción Cancelar Pedido se reciba, la cual pasará el control a la acción Cancelar Orden.



Partición

Una partición de una actividad se muestra como rutas horizontales o verticales. En el siguiente diagrama, las particiones se usan para separar acciones dentro de una actividad en aquellas realizadas por el departamento de contabilidad y aquellas realizadas por el cliente.



PASOS PARA LA ELABORACION DE ESTE DIAGRAMA

1.- Se debe seleccionar la operación que será diagramada; se recomienda seleccionar operaciones importantes que puedan ser, costosas repetitivas y que causen dificultades en el proceso.

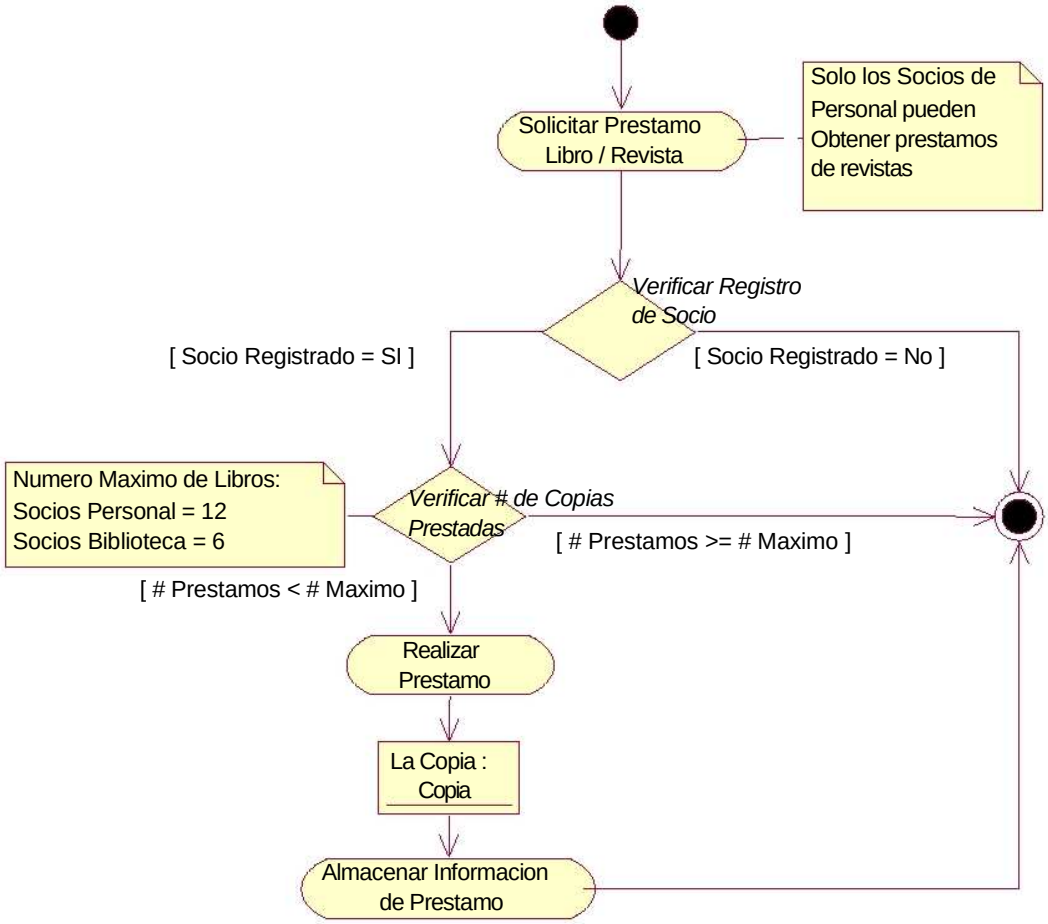
2.- Determinar dónde empieza y dónde termina el ciclo que se quiere diagramar.

3.- Observar varias veces la operación, para dividirla en sus elementos e identificarlos claramente.

4.- Identificar los elementos de la operación, entonces se procede a medir el tiempo de duración de cada uno.

5.- Con los datos anteriores y siguiendo la secuencia de elementos, se construye el diagrama.

EJEMPLO: REALIZAR DIAGRAMA DE ACTIVIDADES DE LA BIBLIOTECA

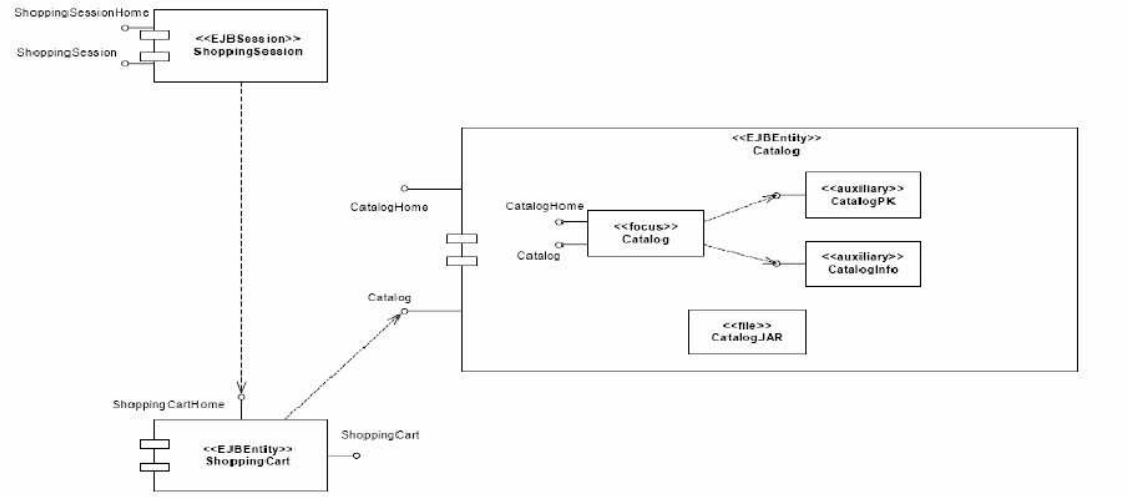

Solicitar Prestamo
Libro / Revista
Solo los Socios de
Personal pueden
Obtener prestamos
de revistas
Verificar Registro
de Socio
[Socio Registrado = SI]
[Socio Registrado = No]
Numero Maximo de Libros:
Socios Personal = 12
Socios Biblioteca = 6
Verificar # de Copias
Prestadas
[# Prestamos >= # Maximo]
[# Prestamos < # Maximo]
Realizar
Prestamo
La Copia :
Copia
Almacenar Informacion
de Prestamo

1.3.8 DIAGRAMA DE COMPONENTES

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes *software*, sean éstos componentes de código fuente, binarios o ejecutables. Desde el punto de vista del diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del *software*, la reutilización, y las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo. Los elementos de modelado dentro de un diagrama de componentes serán componentes y paquetes. En cuanto a los componentes, sólo aparecen tipos de componentes, ya que las instancias específicas de cada tipo se encuentran en el diagrama de despliegue.

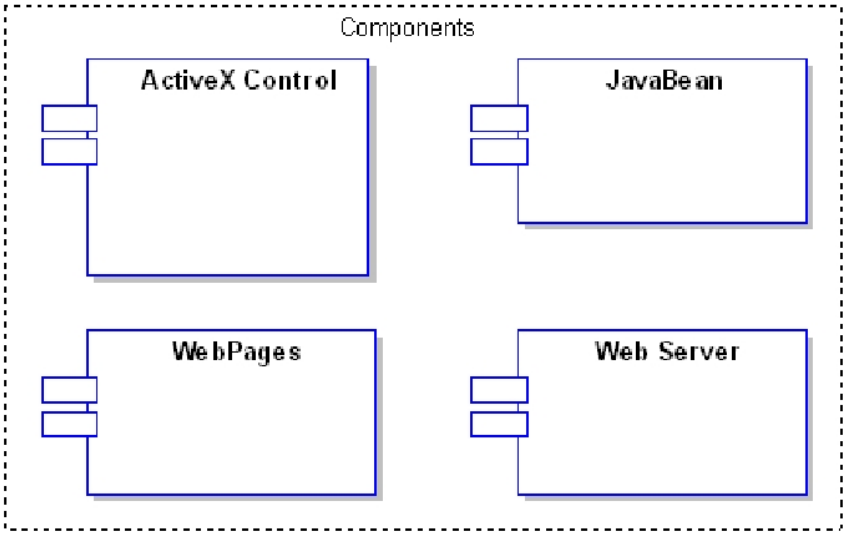
Los diagramas de componentes normalmente contienen componentes, interfaces y relaciones y sus relaciones entre ellos. Y como todos los diagramas, también puede contener paquetes utilizados para agrupar elementos del modelo.

Dado que los diagramas de componentes muestran los componentes *software* que constituyen una parte reusable, sus interfaces, y su interrelaciones, en muchos aspectos se puede considerar que un diagrama de componentes es un diagrama de clases a gran escala. Cada componente en el diagrama debe ser documentado con un diagrama de componentes más detallado, un diagrama de clases, o un diagrama de casos de uso.



Un paquete en un diagrama de componentes que representa un división física del sistema. Los paquetes se organizan en una jerarquía de capas donde cada capa tiene una interfaz bien definida. Un ejemplo típico de una jerarquía en capas de este tipo es: Interfaz de usuario; Paquetes específicos de la aplicación; Paquetes reusables; Mecanismos claves; y Paquetes *hardware* y del sistema operativo.

Un diagrama de componentes se representa como un grafico de componentes *software* unidos por medio de relaciones de dependencia (generalmente de compilación). Puede mostrar también que un componente *software* contiene una interfaz.



Control ActiveX: Componente que se puede insertar en una página Web para proporcionar una funcionalidad que no está directamente disponible en HTML, como secuencias de animación. Los controles ActiveX se pueden implementar en diferentes lenguajes de programación son pequeños programas que permiten mostrar páginas web dinámicas en el PC y que suplen las limitaciones que, al respecto, tiene el lenguaje HTML. Los controles ActiveX tienen que descargarse al disco duro de la computadora para que los documentos que los utilizan puedan visualizarse.

Java Bean: Es un componente hecho en software que se puede reutilizar y que puede ser manipulado visualmente por una herramienta de programación en lenguaje Java. Al igual se define un interfaz para el momento del *diseño* que permite a la herramienta de programación, interrogar (query) al componente y conocer las propiedades (properties) que define y los tipos de sucesos (events) que puede generar en respuesta a diversas acciones.

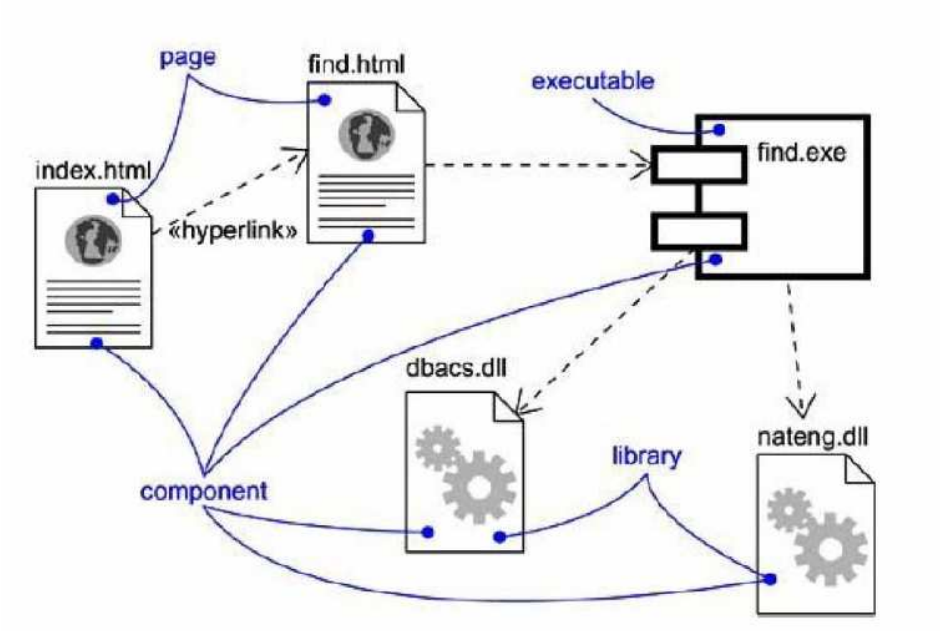
Normalmente los diagramas de componentes se utilizan para modelar código fuente, versiones ejecutables y bases de datos físicas.

- Código fuente: En el modelado de código fuentes se suelen utilizar para representar las dependencias entre los ficheros de código fuente, o para modelar las diferentes versiones de estos fichero. Para ello se deben identificar el conjunto de archivos de código fuente de interés y estereotiparlos como archivos file, agruparlos en paquetes, utilizar valores etiquetados para la información de versiones y modelar las dependencias de compilación.
- Código ejecutable: Se utiliza para modelar la distribución de una nueva versión a los usuarios. Para tal propósito se identifican el conjunto de componentes ejecutables que intervienen, se utilizan estereotipos para los diferentes tipos de componentes (ejecutables, bibliotecas, tablas, archivos, documentos, etc), se consideran las relaciones entre dichos componentes que la mayoría de las veces incluirán interfaces que son exportadas (realizadas) por ciertos componentes e importadas (utilizadas) por otros.
- Bases de datos física: UML permite el modelado de bases de datos físicas así como de los esquemas lógicos de bases de datos.

Así tenemos en cuenta las dependencias asociadas al proceso de compilación un componente podría ser:

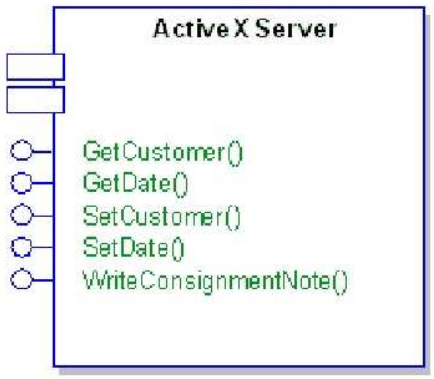
- Código fuente que depende de otros componentes (no necesariamente código fuente) que deben estar disponibles durante la compilación del componente. • Código objeto binario, como por ejemplo una librería, que puede depender de algún código objeto con el que se haga un link.
- Código ejecutable que puede depender de otros programas ejecutables con los que interacciona en tiempo de ejecución.

Se pueden utilizar estereotipos como <<link>> o <<compile>> para distinguir la distinta naturaleza de las dependencias. Igualmente se pueden definir estereotipos para las distintas clases de componentes. UML proporciona algunos estereotipos predefinidos como: <<file>>, <<library>>, <<executable>>, <<table>> y document>>.



Dynamically Linked Library (DLL): Es una librería enlazada a un programa ejecutable en tiempo de ejecución. Es un archivo que contiene funciones que se pueden llamar desde aplicaciones u otras DLL. Los desarrolladores utilizan las DLL para poder reciclar el código y aislar las diferentes tareas. Las DLL no pueden ejecutarse directamente, es necesario llamarlas desde un código externo.

Los componentes también pueden exponer las interfaces. Estas son los puntos visibles de entrada o los servicios que un componente está ofreciendo y dejando disponibles a otros componentes de software y clases. Típicamente, un componente está compuesto por numerosas clases y paquetes de clases internos. También se puede crear a partir de una colección de componentes más pequeños.



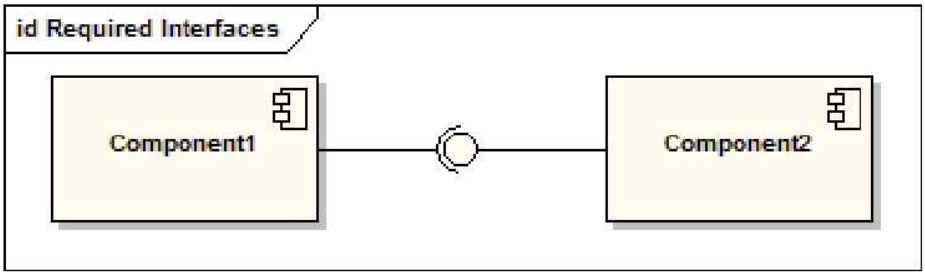
Requisitos: Los componentes pueden tener requisitos adjuntos para indicar sus obligaciones contractuales; esto es, qué servicios proveen en el modelo. Los requisitos ayudan a documentar el comportamiento funcional de los elementos de software.

Restricciones: Los componentes pueden ser restricciones asignadas que indican el entorno en el que operan. Las pre-condiciones especifican lo que debe ser verdadero antes de que un componente pueda realizar alguna función; las post-condiciones indican lo que debe ser verdadero después de que un componente haya realizado algún trabajo y los invariantes especifican lo que debe permanecer verdadero durante la vida del componente.

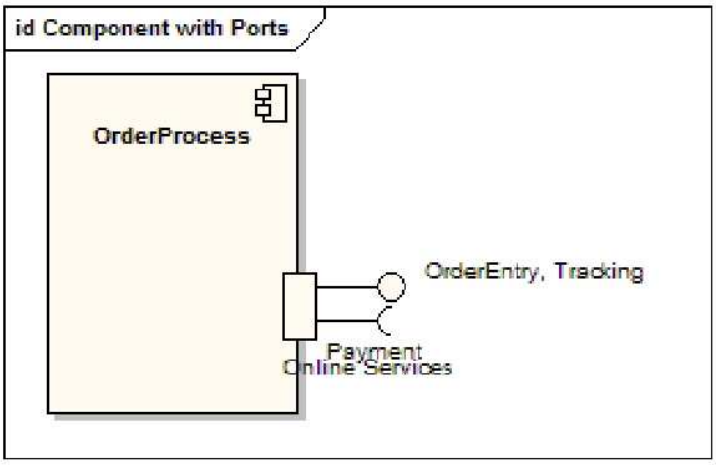
Escenarios: Los escenarios son descripciones textuales y procedimentales de las acciones de un objeto a lo largo del tiempo y describen la forma en la que un componente trabaja. Se pueden crear múltiples escenarios para describir tanto el camino básico (una ejecución perfecta) como las excepciones, errores y otras condiciones.

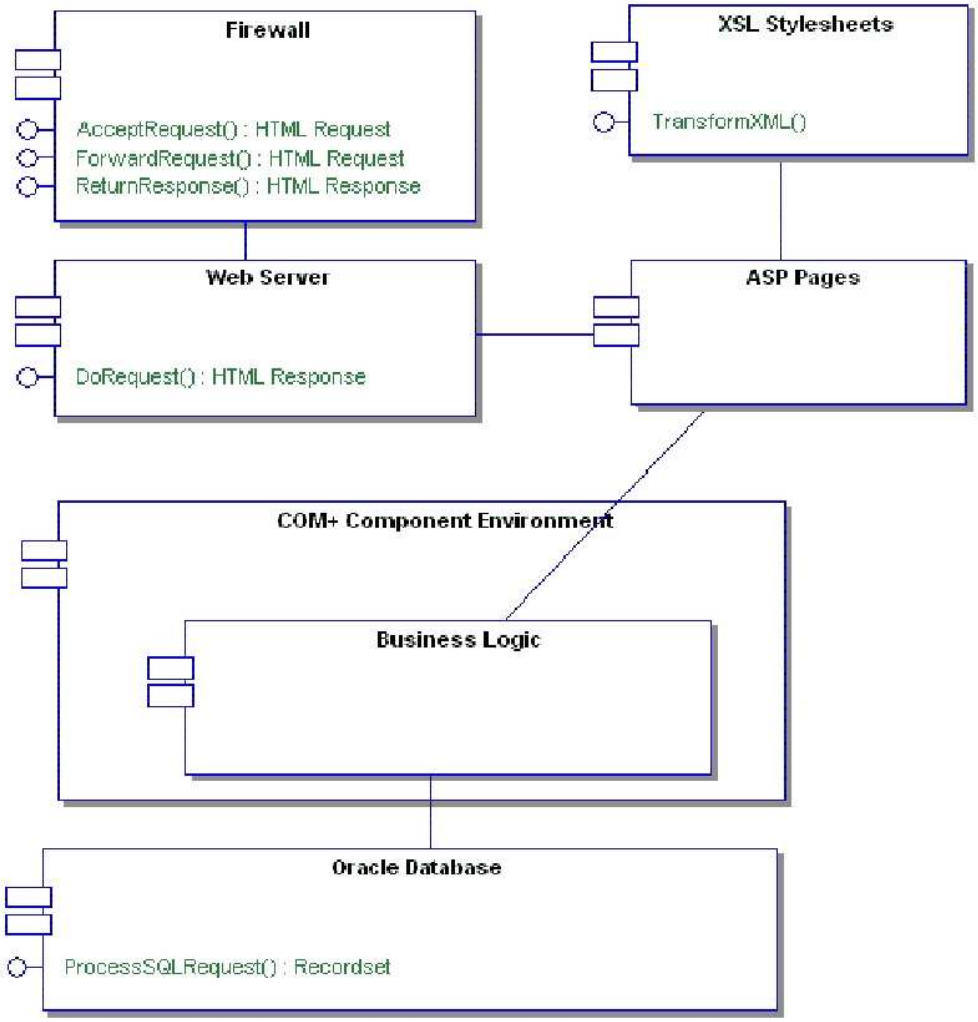
Trazabilidad: Puede indicar la trazabilidad por medio de vínculos de realización. Un componente puede implementar otro elemento del modelo (por ejemplo un caso de uso) o un componente puede ser implementado por otro elemento (por ejemplo un paquete de clases). Al emplear las relaciones de realización desde y hacia los componentes, se pueden seguir las dependencias entre los elementos del modelo y la trazabilidad desde los requisitos iniciales hasta la implementación final.

Interfaces Requeridas: El conector Ensamble une la interfaz requerida del componente (Componente1) con la interfaz proporcionada de otro componente (Component2); esto permite que un componente provea los servicios que otro componente requiere. Las Interfaces son colecciones de uno o más métodos que pueden o no contener atributos.



Componentes con puertos: Usar puertos con Diagramas de Componentes permite que se especifique un servicio o comportamiento a su entorno así como también un servicio o comportamiento que un componente requiere. Los puertos pueden especificar entradas, salidas así como también operar bi-direccionalmente. El siguiente diagrama detalla un componente con un puerto para servicios En Línea conjuntamente con dos interfaces proporcionadas Ordenar Entrada y Seguimiento así como también una interfaz requerida Pago.





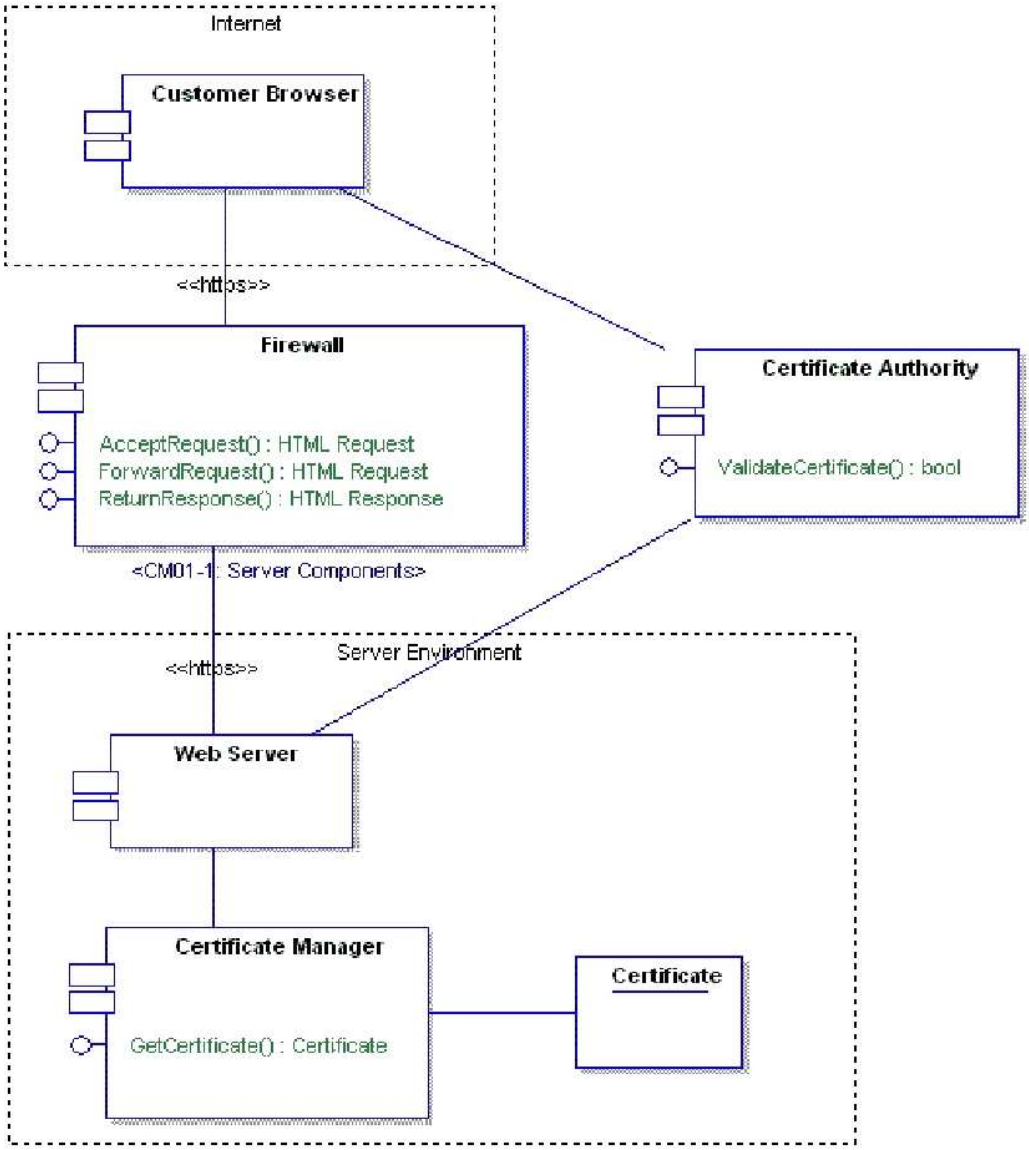
Ejemplo de un diagrama de componentes

XML: Extensible Markup Language («lenguaje de marcas extensible»), es un metalenguaje extensible de etiquetas. Es una simplificación y adaptación que permite definir la gramática de lenguajes específicos (de la misma manera que HTML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Los RSS utilizan (Really Simple Syndication "publicar artículos simultáneamente en diferentes medios a través de una fuente a la que pertenece".

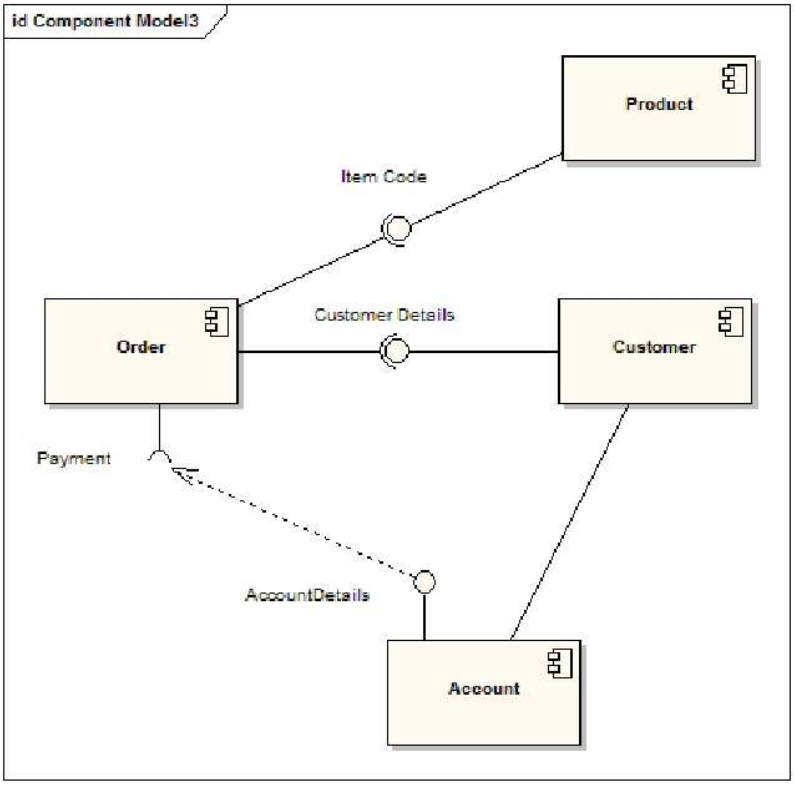
ASP: Active Server Pages es una tecnología del lado servidor de Microsoft para páginas web generadas dinámicamente, que ha sido comercializada como un anexo a Internet Information Server (IIS)

Componentes de seguridad

El diagrama de componentes de la seguridad muestra cómo trabaja en conjunto el software de seguridad, tal como la Autoridad Certificadora (Certificate Authority), el navegador (Browser), el servidor WEB y otros elementos del modelo para asegurar la provisión de la seguridad en el sistema propuesto.



Un diagrama de Componentes tiene un nivel más alto de abstracción que un diagrama de clase - usualmente un componente se implementa por una o más clases (u objetos) en tiempo de ejecución. Estos son bloques de construcción, como eventualmente un componente puede comprender una gran porción de un sistema.



En este diagrama se muestran algunos componentes y sus relaciones internas. Los conectores Ensamble 'vinculan' las interfaces proporcionadas suministrada por el Producto y el Cliente a las interfaces requeridas especificadas por orden. Una relación de dependencia traza los detalles de la cuenta asociada del cliente a la interfaz requerida, 'pago', indicada por orden

PASOS PARA LA REALIZACION DE ESTE DIAGRAMA

Componentes

- Utiliza nombres descriptivos para los componentes arquitecturales
- Usa nombramientos convencionales específicos al área para el diseño detallado de componentes.
- Aplica consistentemente estereotipos textuales a los componentes.
- Evita componentes de modelación de información (DATA) y de interface de usuario.

Interfaces

- Prefiere notaciones "lollipop" para indicar la realización de interfaces por componentes.
- Prefiere el lado izquierdo de un componente para las interfaces "lollipop". • Muestra solamente interfaces relevantes.

Dependencias y herencia

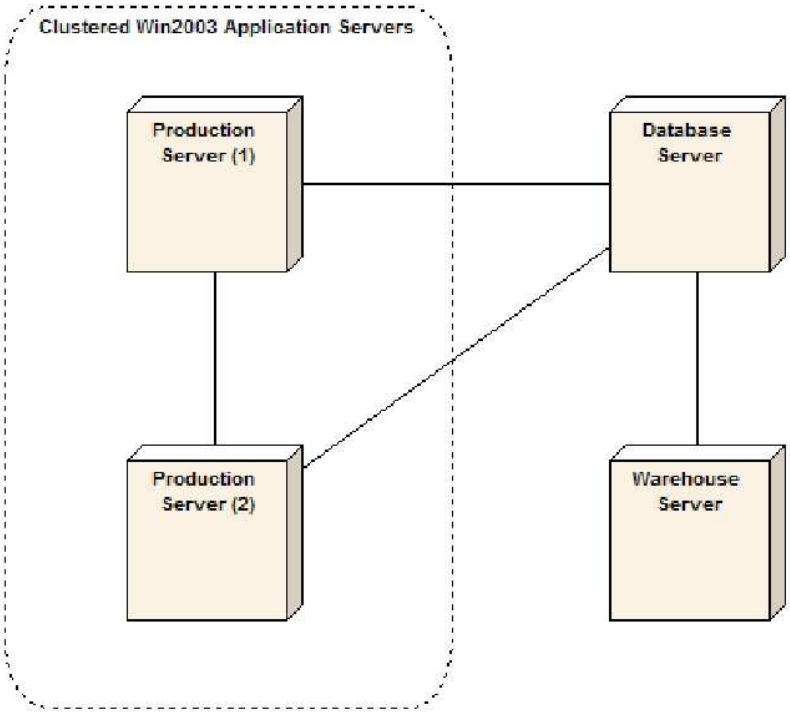
- Modela las dependencias de izquierda a derecha.
- Pon los componentes hijos por debajo de los componentes padres. • Los componentes deben depender solamente de las interfaces. • Evita dependencias de compilaciones modeladoras.

1.3.9 DIAGRAMA DE DESPLIEGUE

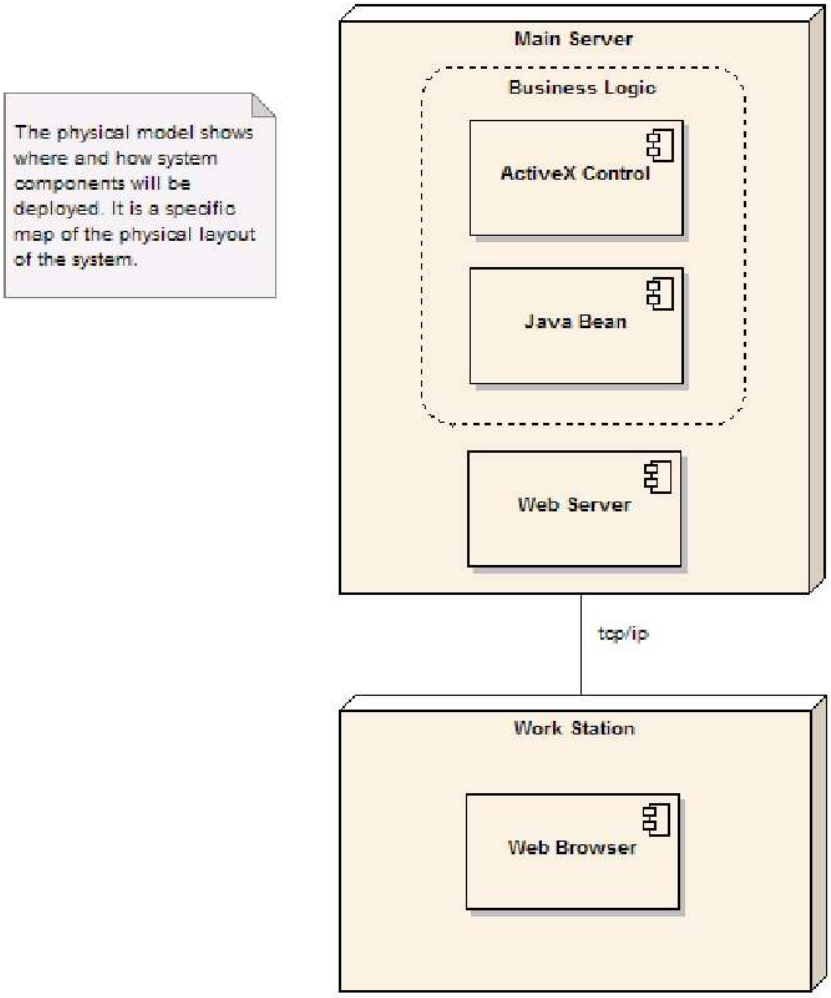
El Modelo Físico/de Despliegue provee un modelo detallado de la forma en la que los componentes se desplegarán a lo largo de la infraestructura del sistema. Detalla las capacidades de red, las especificaciones del servidor, los requisitos de hardware y otra información relacionada al despliegue del sistema propuesto.

Un diagrama de despliegue muestra las relaciones físicas entre los componentes *hardware* y *software* en el sistema final, es decir, la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes *software* (procesos y objetos que se ejecutan en ellos). Estarán formados por instancias de los componentes *software* que representan manifestaciones del código en tiempo de ejecución (los componentes que sólo sean utilizados en tiempo de compilación deben mostrarse en el diagrama de componentes).

Un diagrama de despliegue es un grafico de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes *software*, objetos, procesos (caso particular de un objeto). En general un nodo será una unidad de computación de algún tipo, desde un sensor a un *mainframe*. Las instancias de componentes *software* pueden estar unidas por relaciones de dependencia, posiblemente a interfaces (ya que un componente puede tener más de una interfaz).



Un nodo es un objeto físico en tiempo de ejecución que representa un recurso computacional, generalmente con memoria y capacidad de procesamiento. Pueden representarse instancias o tipos de nodos que se representa como un cubo 3D en los diagramas de implementación.



Se utiliza un nodo para identificar cualquier servidor, terminal de trabajo u otro hardware host que se utiliza para desplegar componentes en el ambiente de producción. Usted también puede especificar los vínculos entre los nodos y asignarles estereotipos (tales como TCP/IP) y requisitos. Los nodos también pueden tener documentados características de performance, estándares mínimos de hardware, niveles de sistema operativo, etc. La pantalla de abajo ilustra las propiedades comunes que puede establecer para un nodo.

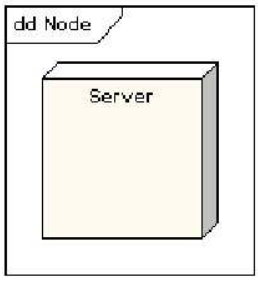
Los diagramas de despliegue muestran la configuración en funcionamiento del sistema, incluyendo su hardware y su software. Para cada componente de un diagrama de despliegue se deben documentar las características técnicas requeridas, el tráfico de red esperado, el tiempo de respuesta requerido, etc.

La mayoría de las veces el modelado de la vista de despliegue estática implica modelar la topología del hardware sobre el que se ejecuta el sistema. Los diagramas de despliegue son fundamentalmente diagramas de clases que se ocupan de modelar los nodos de un sistema. Aunque UML no es un lenguaje de especificación hardware de propósito general, se ha dise nado para modelar muchos de los aspectos hardware de un sistema a un nivel suficiente para que un ingeniero software pueda especificarla plataforma sobre la que se ejecuta el software del sistema y para que un ingeniero de sistemas pueda manejar la frontera entre el hardware y el software cuando se trata de la relación entre hardware y software se utilizan los diagramas de despliegue para razonar sobre la topología de procesadores y dispositivos sobre los que se ejecuta el software.

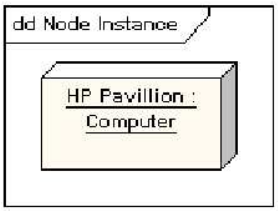
Esta vista cubre principalmente la distribución, entrega e instalación de las partes que configuran un sistema físico. Los diagramas de despliegue se suelen utilizar para modelar:

- Sistemas empotrados: Un sistema empotrado es un colección de hardware con una gran cantidad de software que interactúa con el mundo físico. Los sistemas empotrados involucran software que controla dispositivo (motores, actuadores) que a su vez están controlados por estímulos externos como sensores.
- Sistemas cliente-servidor: Los sistemas cliente-servidor son un extremos del espectro de los sistemas distribuidos y requieren tomar decisiones sobre la conectividad de red de los clientes a los servidores y sobre la distribución física de los componentes software del sistemas a través de nodos.
- Sistemas completamente distribuidos: En el otro extremo encontramos aquellos sistemas que son ampliamente o totalmente distribuidos y que normalmente incluyen varios niveles de servidores Tales sistemas contienen a menudo varias versiones de componentes software, alguno de los cuales pueden incluso migrar de un nodo a otro. El diseño de tales sistemas requiere tomar decisiones que permitan un cambio continuo de la topología del sistema.

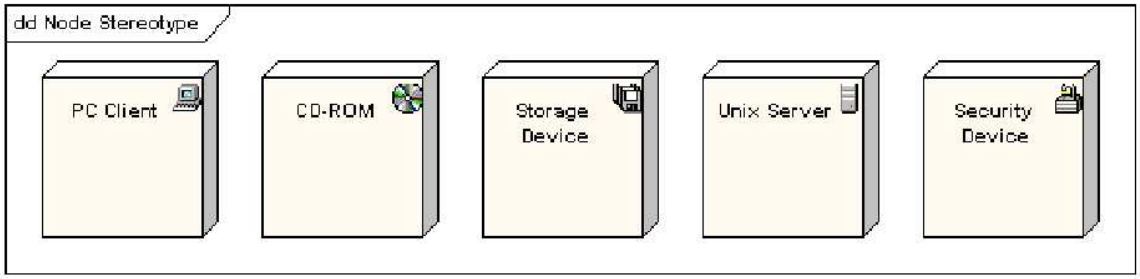
Nodo: Un Nodo es un elemento de hardware o software. Esto se muestra con la forma de una caja en tres dimensiones, como a continuación.



Instancia de Nodo: Una instancia de nodo se puede mostrar en un diagrama. Una instancia se puede distinguir desde un nodo por el hecho de que su nombre esta subrayado y tiene dos puntos antes del tipo de nodo base. Una instancia puede o no tener un nombre antes de los dos puntos. El siguiente diagrama muestra una instancia nombrada de una computadora.

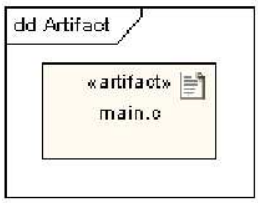


Estereotipo de Nodo: Un número de estereotipos estándar se proveen para los nodos, nombrados «cdrom», «cd-rom», «computer», «disk array», «pc», «pc client», «pc server», «secure», «server», «storage», «unix server», «user pc». Estos mostrarán un icono apropiado en la esquina derecha arriba del símbolo nodo.

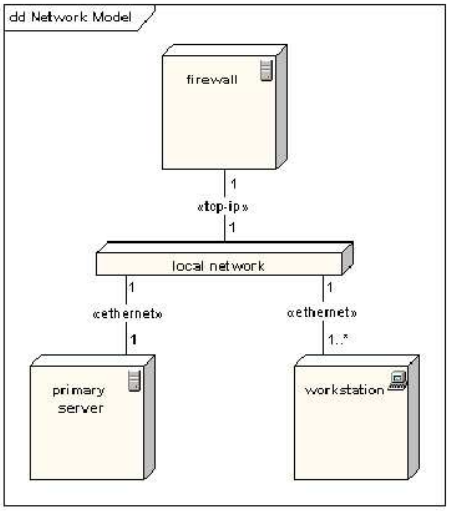


Artefacto: Un artefacto es un producto del proceso de desarrollo de software, que puede incluir los modelos del proceso (e.g. modelos de Casos de Uso, modelos de Diseño, etc.), archivos fuente, ejecutables, documentos de diseño, reportes de prueba, prototipos, manuales de usuario y más.

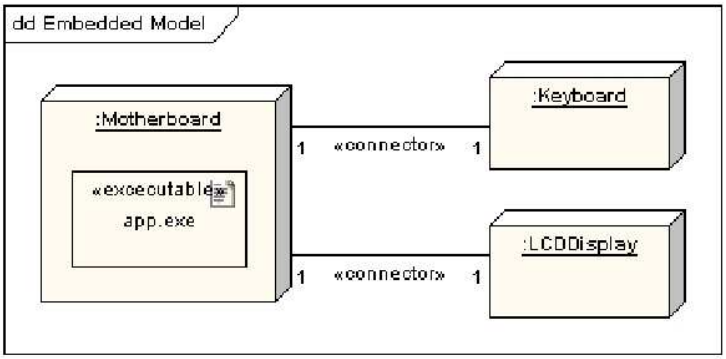
Un artefacto se denota por un rectángulo mostrando el nombre del artefacto, el estereotipo «artifact» y un icono de documento, como a continuación.



Asociación: En el contexto del diagrama de despliegue, una asociación representa una ruta de comunicación entre los nodos. El siguiente diagrama muestra un diagrama de despliegue para una red, mostrando los protocolos de red como estereotipos y también mostrando multiplicidades en los extremos de la asociación.



Nodo como contenedor: Un nodo puede contener otros elementos, como componentes o artefactos. El siguiente diagrama muestra un diagrama de despliegue para una parte del sistema y muestra un artefacto ejecutable como contenido por el nodo madre (motherboard).



PASOS PARA LA ELABORACION DE ESTE DIAGRAMA

General

1. Indica los componentes del software o sistema en Diagramas Específicos al proyecto.
2. Enfocarse en los Nodos y las Asociaciones de comunicación en los niveles mas importantes.

Nodos y Componentes

1. Nombra los nodos con términos descriptivos.
2. Modela solamente los componentes vitales del software o sistema. 3.
Aplica Estereotipos consistentes a los componentes.
4. Aplica Estereotipos visuales a los nodos

Dependencias y Asociaciones de Comunicación

1. Indica los protocolos de comunicación vía estereotipos
2. Modela solamente las dependencias críticas entre componentes

Bibliografía:

1.- Ingeniería de Software Orientada a Objetos con UML Java e Internet
Autor: Alfredo Weitzenfeld
Editorial: Thomson

2.- Análisis y Diseño Estructurado y Orientado a Objetos de Sistemas Informáticos
Autor: Antonio de Amescua Seco
Editorial: Mc Graw Hill

3.- Ingeniería de Software Clásica y Orientada a Objetos
Autor: Stephen R. Schach
Editorial: Mc Graw Hill

4.- Utilización de UML en Ingeniería de Software con Objetos y Componentes
Autor: Stevens Perdita
Editorial: Addison Wesley

5.- Análisis y Diseño de sistemas de Información
Autor: Jeffrey L. Whitten
Editorial: Mc Graw Hill

6.- Teach yourself UML in 24 Hours
Autor: Joseph Schmuller
Editorial: SAMS

7.- http://www.sparxsystems.com.ar/resources/tutorial/uml2_statediagram.html

Bibliografía:

1.- Ingeniería de Software Orientada a Objetos con UML Java e Internet
Autor: Alfredo Weitzenfeld
Editorial: Thomson

2.- Análisis y Diseño Estructurado y Orientado a Objetos de Sistemas Informaticos
Autor: Antonio de Amescua Seco
Editorial: Mc Graw Hill

3.- Ingeniería de Software Clásica y Orientada a Objetos
Autor: Stephen R. Schach
Editorial: Mc Graw Hill

4.- Utilización de UML en Ingeniería de Software con Objetos y Componentes
Autor: Stevens Perdita
Editorial: Addison Wesley

5.- Análisis y Diseño de sistemas de Información
Autor: Jeffrey L. Whitten
Editorial: Mc Graw Hill

6.- Teach yourself UML in 24 Hours
Autor: Joseph Schmuller
Editorial: SAMS