
4


Lenguaje
UML

4. Lenguaje UML

UML (*Unified Modeling Language, Lenguaje Unificado de Modelamiento*) es un lenguaje que permite modelar, construir y documentar los elementos que forman un Sistema Software Orientado a Objetos. Se ha convertido en el estándar debido a que ha sido concebido por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh. Estos autores fueron contratados por la empresa Rational Software Co. para crear una notación unificada en la que basar la construcción de sus herramientas CASE. En el proceso de creación de UML han participado, no obstante, otras empresas de gran peso en la industria como Microsoft, Hewlett-Packard, Oracle o IBM, así como grupos de analistas y desarrolladores.

Esta notación ha sido ampliamente aceptada debido al prestigio de sus creadores y debido a que incorpora las principales ventajas de cada uno de los métodos particulares en los que se basa: Booch, OMT y OOSE. UML ha puesto fin a las llamadas “guerras de métodos” que se han mantenido a lo largo de los 90´s, en las que los principales métodos lanzaban nuevas versiones que incorporaban las técnicas de los demás (Figura 4.1). Con UML se fusiona la notación de estas técnicas para formar una herramienta compartida entre todos los ingenieros software que trabajan en el desarrollo orientado a objetos.

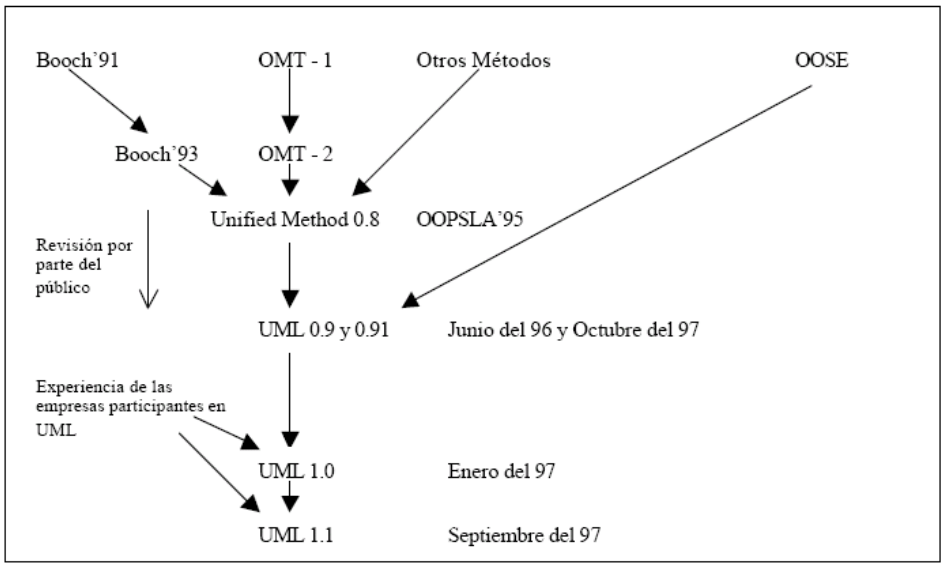


Figura 4.1. Historia de UML

UML es un "lenguaje" para especificar y pretende ser un método de desarrollo completo o un proceso de desarrollo paso a paso, tiene como propósito general el modelado orientado a objetos. Incluye todos los conceptos que se consideran necesarios para utilizar un proceso moderno iterativo, basado en construir una sólida arquitectura para resolver requisitos dirigidos por casos de uso.

Es simple, ya que tiene la capacidad de modelar toda la gama de sistemas que se necesita construir, es expresivo para manejar todos los conceptos que se originan en un sistema moderno, tales como la concurrencia y distribución, así como también los mecanismos de la ingeniería de software.

4.1 Definición de UML

Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar un sistema de software. Se usa para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir.

UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos que interactúan para realizar un trabajo que finalmente beneficia a un usuario externo.
El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar.

Es un lenguaje de propósito general para el modelado orientado a objetos. UML es también un lenguaje de modelamiento visual que permite una abstracción del sistema y sus componentes.

Existían diversos métodos y técnicas Orientadas a Objetos, con muchos aspectos en común pero utilizando distintas notaciones, se presentaban inconvenientes para el aprendizaje, aplicación, construcción y uso de herramientas, etc., además de pugnas entre enfoques, lo que genero la creación del UML como estándar para el modelamiento de sistemas de software principalmente, pero con posibilidades de ser aplicado a todo tipo de proyectos.

Los conceptos y modelos de UML pueden agruparse en las siguientes áreas conceptuales:

Estructura estática:

Cualquier modelo preciso debe primero definir su universo, esto es, los conceptos clave de la aplicación, sus propiedades internas, y las relaciones entre cada una de ellas. Este conjunto de construcciones es la estructura estática. Los conceptos de la aplicación son modelados como clases, cada una de las cuales describe un conjunto de objetos que almacenan información y se comunican para implementar un comportamiento. La información que almacena es modelada como atributos; La estructura estática se expresa con diagramas de clases y puede usar para generar la mayoría de las declaraciones de estructuras de datos en un programa.

Comportamiento dinámico:

Hay dos formas de modelar el comportamiento, una es la historia de la vida de un objeto y la forma como interactúa con el resto del mundo y la otra es por los patrones de comunicación de un conjunto de objetos conectados, es decir la forma en que interactúan entre sí. La visión de un objeto aislado es una máquina de estados, muestra la forma en que el objeto responde a los eventos en función de su estado actual. La visión de la interacción de los objetos se representa con los enlaces entre objetos junto con el flujo de mensajes y los enlaces entre ellos. Este punto de vista unifica la estructura de los datos, el control de flujo y el flujo de datos.

Construcciones de implementación:

Los modelos UML tienen significado para el análisis lógico y para la implementación física. Un componente es una parte física reemplazable de un sistema y es capaz de responder a las peticiones descritas por un conjunto de interfaces. Un nodo es un recurso computacional que define una localización durante la ejecución de un sistema. Puede contener componentes y objetos.

Organización del modelo:

La información del modelo debe ser dividida en piezas coherentes, para que los equipos puedan trabajar en las diferentes partes de forma concurrente. El conocimiento humano requiere que se organice el contenido del modelo en

paquetes de tamaño modesto. Los paquetes son unidades organizativas, jerárquicas y de propósito general de los modelos de UML. Pueden usarse para almacenamiento, control de acceso, gestión de la configuración y construcción de bibliotecas que contengan fragmentos de código reutilizable.

4.2 Diagramas de UML

El UML está compuesto de diversos elementos gráficos que se combinan para conformar diagramas.
La finalidad de los diagramas es representar diversas perspectivas de un sistema, a los cuales se les conoce como métodos, es importante recalcar que UML describe el funcionamiento de un sistema, pero no indica la forma de implementarlo.
De acuerdo a la especificación UML 2.0 los diagramas se clasifican en:

Diagramas Estructurales: Describe los elementos del sistema y sus relaciones con otros elementos:

- Diagrama de clase.
- Diagrama de Objetos.
- Diagrama de casos de uso.
- Diagrama de componentes.
- Diagrama de despliegue.
-

Diagramas de Comportamiento dinámico: describe el comportamiento del sistema a través del tiempo:

- Diagrama de estado.
- Diagrama de actividades.
- Diagrama de secuencia.
- Diagrama de colaboración.
-

Diagramas de Gestión de modelo: Describe la organización de los modelos mismos en unidades jerárquicas. El paquete es la unidad de organización para los modelos, y existen tipos especiales de paquetes con los modelos y subsistemas:

- Diagrama de clases (paquetes, subsistemas y modelos).

La clasificación de los diagramas y sus principales conceptos se presentan a continuación (Cuadro 4.2):

Clasificación	**Diagramas**	**Principales Conceptos**
Estructurales	Diagrama de clases	Clase, asociación, generalización, dependencia, realización, interface
	Diagrama de objetos	
	Diagrama de casos de uso	Caso de uso, actor, asociación, extensión, inclusión, generalización.
	Diagrama de componentes	Componente, interface, dependencia, realización.
	Diagrama de despliegue	Nodo, componente, dependencia, localización.
Comportamiento Dinámico	Diagrama de estados	Estado, evento, transición, acción.
	Diagrama de actividades	Estado, actividad, transición, concurrencia (fork), reunión (join).
	Diagrama de secuencia	Interacción, Objeto, Mensaje, Activación.

	Diagrama de colaboración	Colaboración, interacción, rol de colaboración, mensaje.
Gestión de Modelo	Diagrama de clases	Paquete, subsistema, modelo.

Cuadro 4.2. Clasificación de los diagramas UML

A continuación se describen los diagramas más usados por UML y por la metodología RUP.

4.2.1 Diagrama de casos de uso

Un caso de uso es una descripción de las acciones de un sistema desde el punto de vista del usuario. Los diagramas de casos de usos son una técnica de aciertos y errores para obtener los requerimientos del sistema.

Actor: Es una idealización de una persona externa, de un proceso, o de una cosa que interactúa con el sistema (Figura 4.3). Los actores son objetos que residen *fuera* del sistema, en tanto que los casos de uso son objetos que residen *dentro* del sistema. Un actor puede ser una persona, otro sistema, o un proceso.

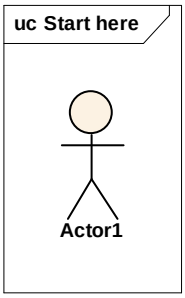


Figura 4.3. Actor

Caso de uso: Es una secuencia de transacciones realizadas que brinda un resultado de valor a un actor en particular (Figura 4.4). El propósito de los casos de usos es definir una pieza de comportamiento coherente, sin revelar la estructura interna del sistema.

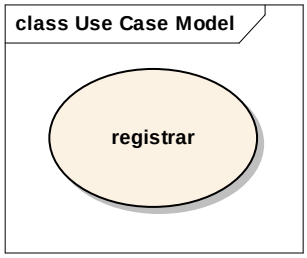


Figura 4.4. Caso de uso

´

Los casos de uso cumplen dos funciones importantes:

a) Capturan requerimientos funcionales del sistema:

El modelo de casos de uso define el comportamiento del sistema a través de un conjunto de casos de uso. El *entorno* del sistema es descrito por un conjunto de actores que usan el sistema a través de los casos de uso. El modelo de casos de uso es una vista *externa* del sistema.

b) Estructuran los modelos de objetos en vistas manejables:

En orden de manejar la complejidad de un sistema real, es práctico construir modelos de objetos para cada caso de uso con los objetos que participan en dicho caso de uso. Un objeto puede participar en varios casos de uso, esto significa que el modelo de objetos completo se obtiene a partir de un conjunto de vistas de modelos de objetos, uno por cada caso de uso.

Un caso de uso es especificado por interacciones de UML, y representadas en diagramas de secuencia, diagramas de colaboración, descripciones de texto informales.

Relaciones entre casos de uso:

Las relaciones indican la forma de interacción entre cada caso de uso, existen 3 tipos de relaciones para los casos de uso:

a) Inclusión:

Un caso de uso puede incluir, en su comportamiento, el comportamiento de otro caso de uso base a través de una relación de inclusión.

Cuando varios casos de uso comparten descripciones similares en orden de evitar redundancia y maximizar reutilización se puede extraer dichas *secuencias comunes*. La relación de inclusión es una relación de dependencia entre casos de uso (Figura 4.5).

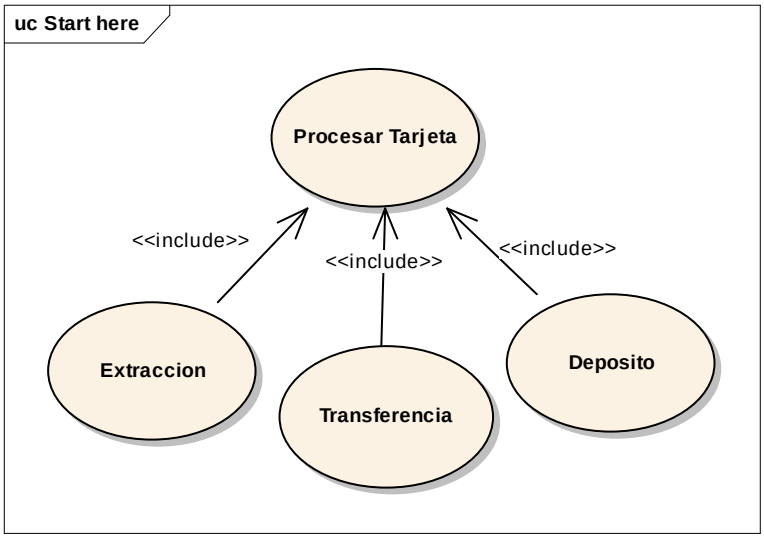


Figura 4.5. Relación de inclusión en los casos de uso

b) Extensión:

Su usa la relación de extensión para:

- Partes opcionales de un caso de uso.
- Cursos complejos y alternativos.
- Subsecuencias que se ejecutan solo bajo ciertas condiciones.

La relación de extensión (Figura 4.6) permite un desarrollo incremental, comenzando el desarrollo con casos de uso más simples e ir agregando comportamientos específicos (extensiones a los caso de uso base).

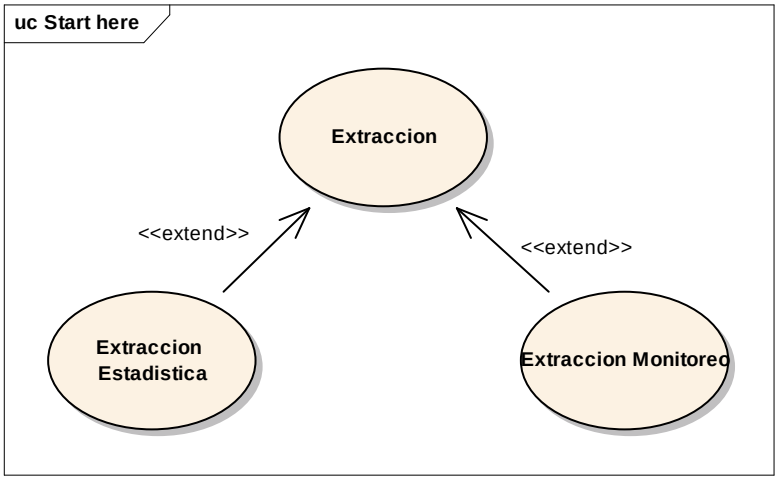


Figura 4.6 Relación de extensión para los casos de uso

c) Generalización

Un caso de uso se puede especializar en uno o más casos de uso hijos, utilizando una relación de generalización (Figura 4.7).

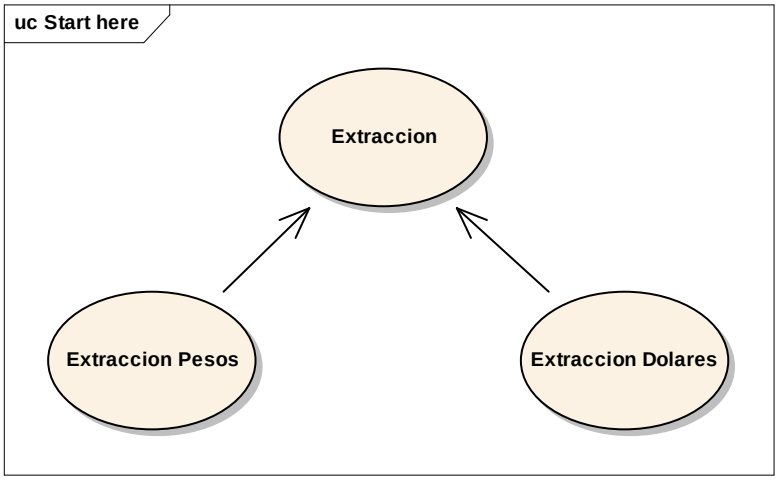


Figura 4.7 Generalización

4.2.2 Diagrama de secuencia

Muestra la mecánica de interacción con base al tiempo. Es un diagrama bidimensional. La dimensión vertical es el eje de tiempo y la dimensión horizontal muestra los roles de clasificadores que representan objetos individuales en la colaboración.

Objetos:

Está representado por un rectángulo con un nombre subrayado, debajo de el existe un línea (hilo de control) discontinua que representa el ciclo de vida del objeto. En la línea de vida existen elementos rectangulares llamadas activación, la cual representa una operación que realiza el objeto (Figura 4.8).

Un objeto activo es un objeto que mantiene la pila de activaciones. Cada objeto activo tiene su propio hilo de control que se ejecuta en paralelo con otros objetos. Los objetos que son llamados por los objetos activos se denominan *objetos pasivos*, estos reciben el control solo cuando reciben un mensaje solicitando una operación.

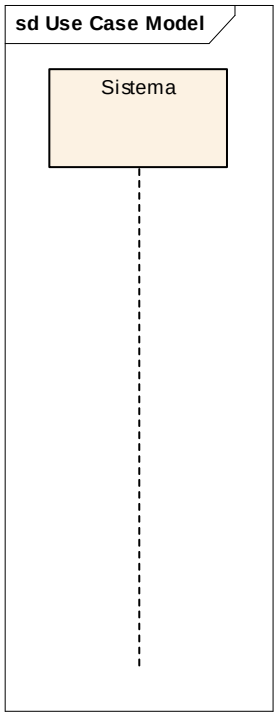


Figura 4.8. Objeto y línea del ciclo de vida

Mensajes:

Un mensaje va desde un objeto a otro pasando por su línea del ciclo de vida activando una operación en el objeto receptor.

Un mensaje puede ser:

a) Simple: indica la transferencia de control de un objeto a otro.
b) Síncrono: Cuando se manda este tipo de mensajes, el objeto esperara la respuesta a tal mensaje antes de continuar con su trabajo.
c) Asíncrono: El objeto no espera la respuesta del mensaje para continuar.

Cada uno de los mensajes tiene un conector específico (Figura 4.9):

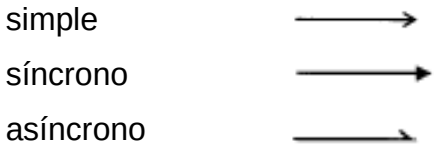


Figura 4.9. Conectores de mensajes

Tiempo:

El diagrama representa el tiempo en dirección vertical. El tiempo se inicia en la parte superior y avanza hacia la parte inferior del diagrama. El diagrama de secuencia tiene dos dimensiones (Figura 4.10), la dimensión horizontal es la posición de objetos y la dimensión vertical muestra el transcurso del tiempo.

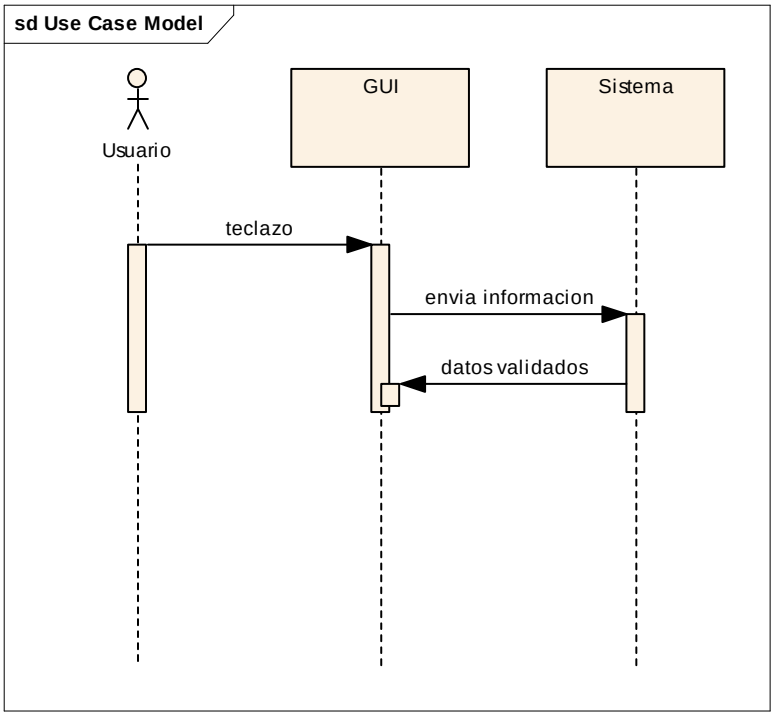


Figura 4.10. Diagrama de secuencia

Casos de uso y diagrama de secuencia:

El diagrama de secuencia (Figura 4.12) se centra en un escenario de un caso de uso (Figura 4.11):

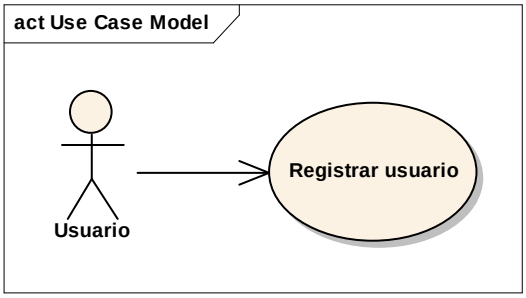


Figura 4.11. Diagrama de casos de uso para el proceso de registro de usuarios

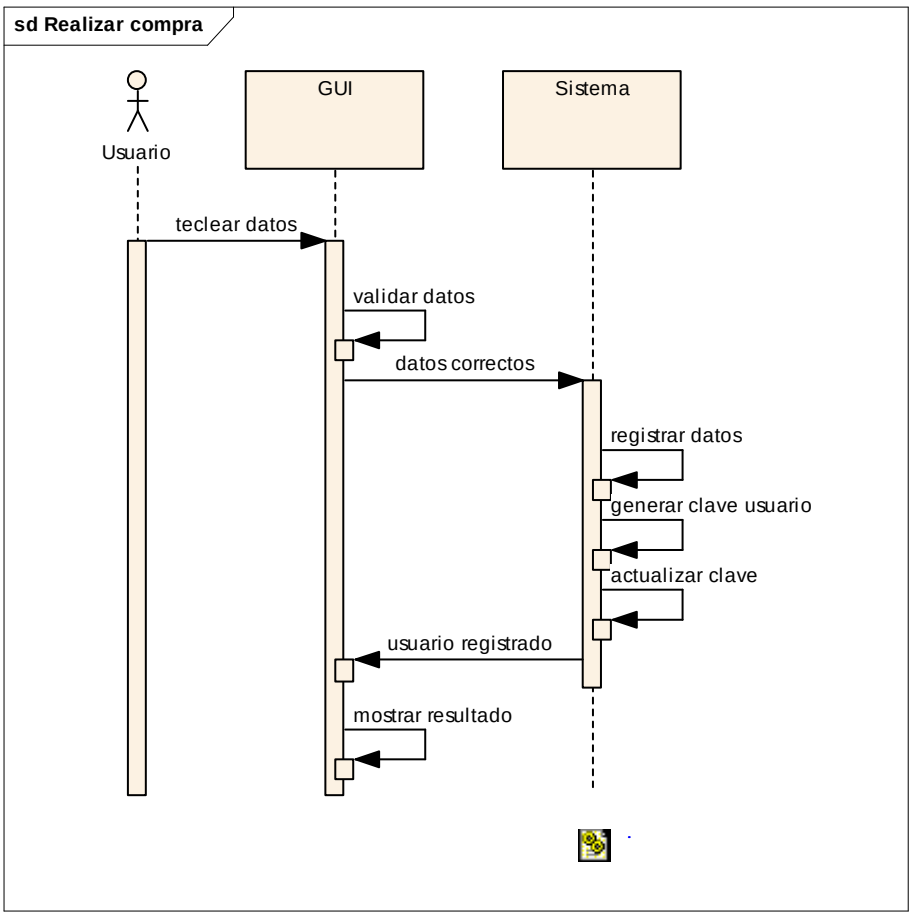


Figura 4.12. Diagrama de secuencia para el caso de uso de registro de usuario

4.2.3 Diagrama de estados

En cualquier momento, un objeto se encuentra en un estado en particular, el diagramas de estados es una manera de caracterizar los cambios en un sistema es decir que los objetos que lo componen cambien de estado como respuesta a los sucesos y al tiempo (Figura 4.16).

El diagrama de estados presenta los estados en los que se puede encontrar un objeto junto con las transiciones entre los estados, además de mostrar un punto inicial y final de una secuencia de cambios de estados

Simbología:

En la Figura (4.13) se muestra los elementos que integran un diagrama de estados:

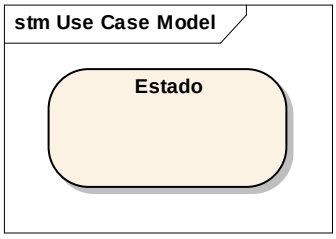
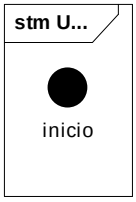
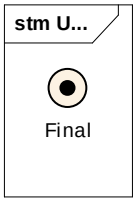
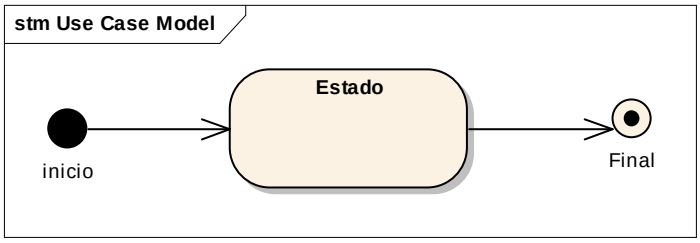
Elemento	Símbolo
Estado	
Nodo Inicial (inicio de la secuencia de estados)	
Nodo Final (final de la secuencia de estados)	
Transición	

Figura 4.13. Elementos del diagrama de estados

El elemento de estado se puede dividir en tres partes, el área superior corresponde al nombre del estado, en el área central corresponde a las variables de estado y el área inferior a los métodos (Figura 4.14).

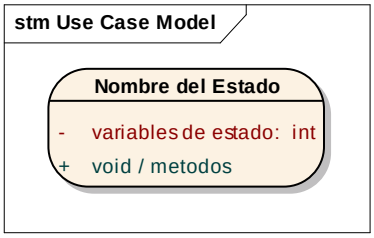


Figura 4.14. Estado

Sucesos y acciones:

Se pueden agregar detalles a las líneas de transiciones para indicar un suceso que provoque una transición y la actividad u operación que se ejecuto para provocar el cambio de estado. Los sucesos y las acciones se escriben cercas de la línea de transición y separadas por una diagonal. Existen transiciones que no son provocadas por un evento o acción asociada al cambio, por lo general estas transiciones con originadas por la finalización de algún estado, a estas transiciones se les llama no desencadenadas.

Cuando se dispara una transición, su acción (si la hay) es ejecutada. Una acción es un cómputo atómico y breve (Figura 4.15). A menudo es:

- Una sentencia de asignación
- Una operación aritmética
- El envío de una señal a otro objeto
- La invocación de una operación propia
- Asignación de valores de retorno
- Creación o destrucción de objetos
- Una secuencia de acciones simples

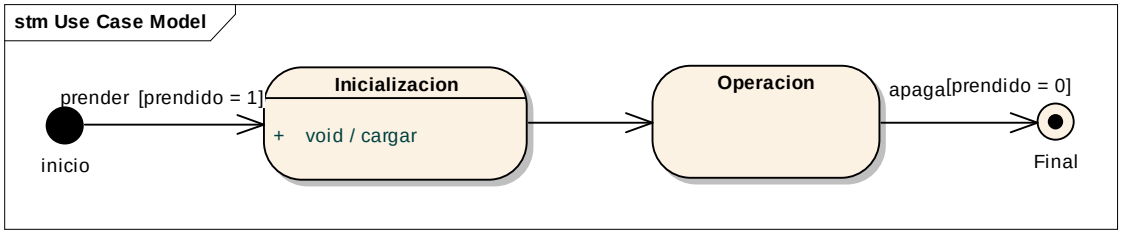


Figura 4.15. Acciones que disparan la transición

Estados anidados:

Los estados se pueden anidar dentro de otros estados compuestos. Una transición que deja el estado más externo es aplicable a todos los estados internos.

Acciones de entrada y salida:

Un estado puede tener acciones que se realicen siempre que se entre o se salga del estado. Si la transición sale del estado original, entonces su acción de salida se ejecuta antes de la acción de la transición y de la acción de entrada en el estado nuevo.

Transición interna:

Una transición interna tiene un estado origen pero ningún estado destino. Si una transición interna tiene acción, se ejecuta pero no existe cambio de estado.

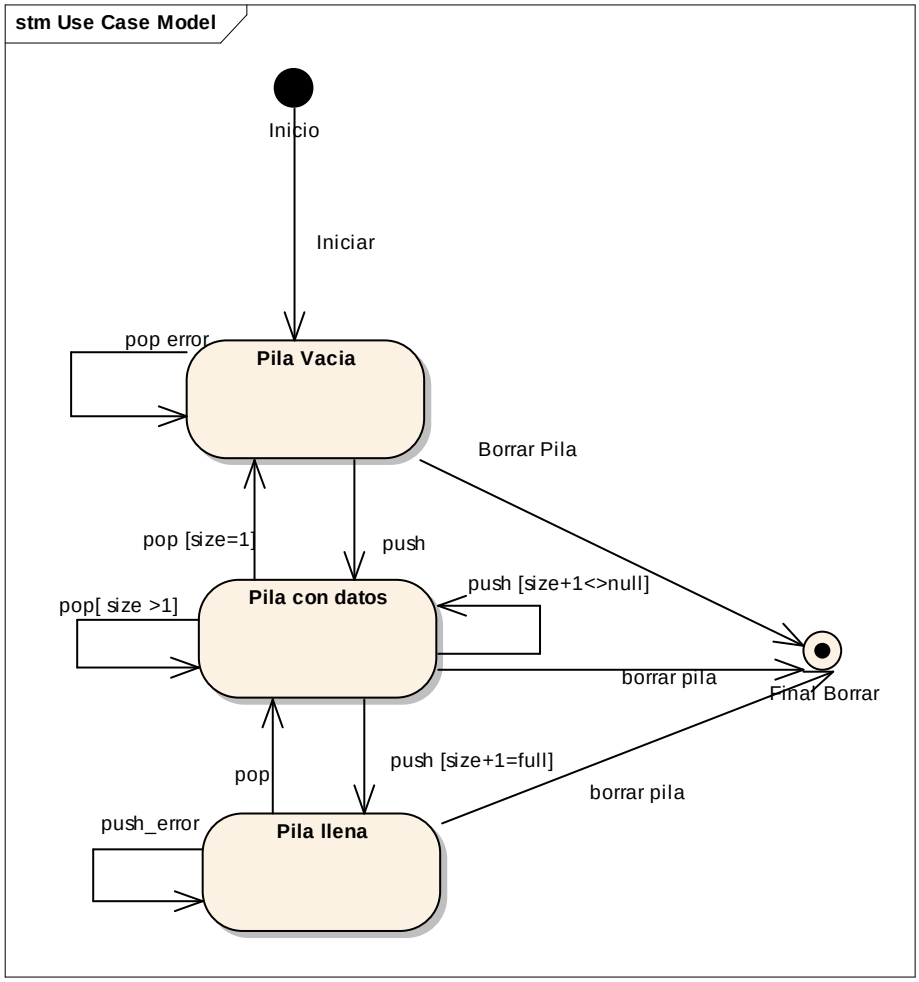


Figura 4.16. Diagrama de estados para la estructura de datos de pila

4.2.4 Diagrama de actividades

Las actividades de un caso de uso o dentro del comportamiento de un objeto se dan, normalmente, en secuencia. El diagrama de actividades esta diseñado para mostrar una visión simplificada de lo que ocurre durante una operación o proceso. Es una extensión del diagrama de estados. El diagrama de estados muestra los estados de un objeto y representa las actividades como flechas que conectan a los estados, el propósito del diagrama de actividades es resaltar estas actividades.

Simbología:

En la Figura 4.17 se muestra los elementos que integran un diagrama de actividades:

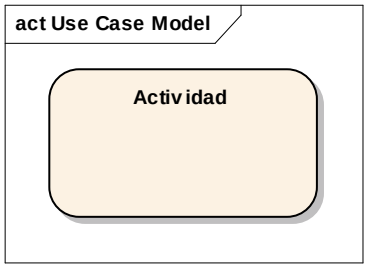
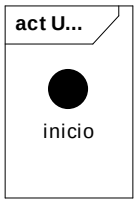
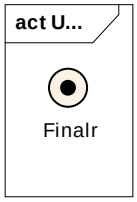
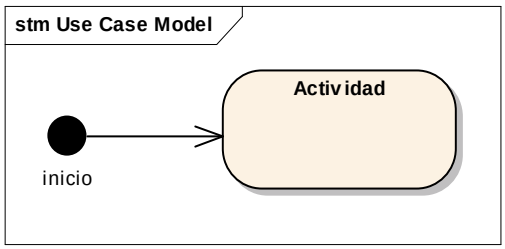
Elemento	**Símbolo**
Actividad (La actividad descrita es la que es ejecutada)	
Nodo inicial	
Nodo Final	
Transición	

Figura 4.17. Elementos del diagrama de actividades

Decisiones:

Hay dos formas de representar puntos de decisión (Figura 4.18), mediante la división de la transición o usando el rombo de dedición estableciendo actividades excluyentes:

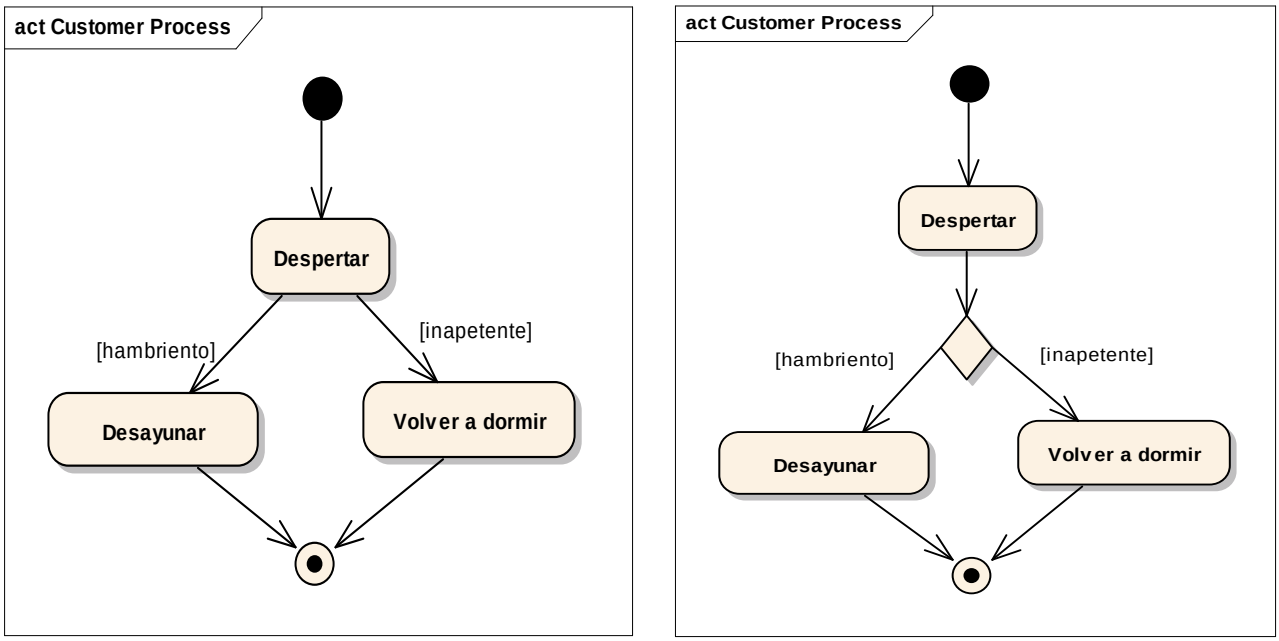


Figura 4.18. Uso de decisiones en el diagrama de actividades

Rutas Concurrentes:

Indica la división de una transición, está representada por una línea gruesa en donde llegan y salen flechas (Figura 4.19), establecen actividades realizándose simultáneamente.

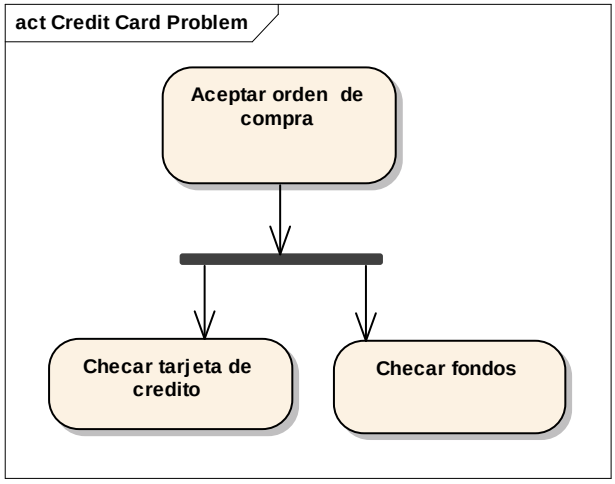


Figura 4.19 Rutas concurrentes

A continuación se presenta un ejemplo del uso de los diagramas de actividades (Figura 4.20):

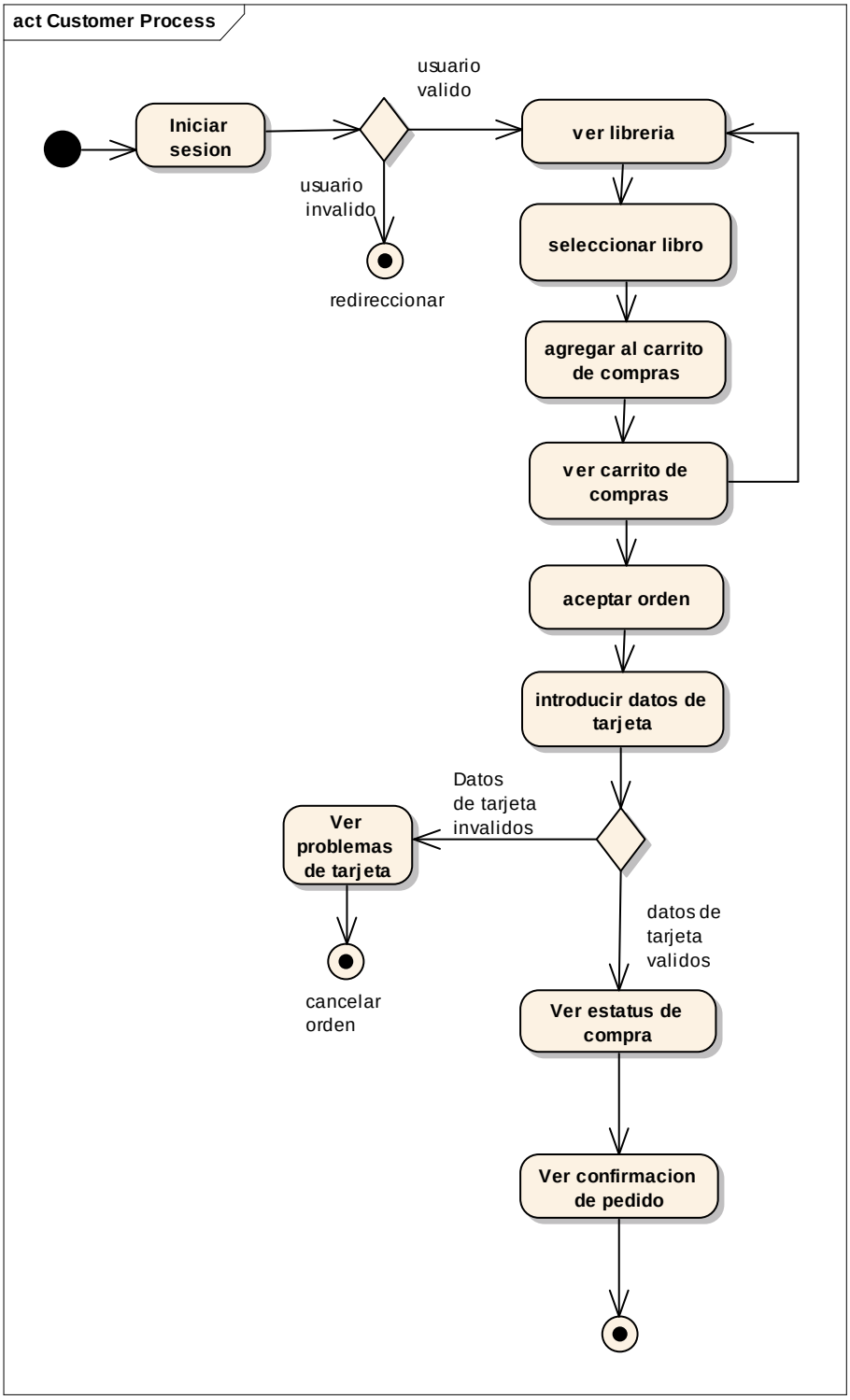


Figura 4.20. Diagrama de actividades para la compra electrónica de libros.

4.2.5 Diagrama de Clases

Un diagrama de clases está formado por rectángulos conectados entre si, estos rectángulos representan a las clases (Figura 4.21) involucradas en el sistema, en estos se especifican los métodos y atributos (datos) de dicha clase.

El diagrama de clases representa las relaciones e interacciones entre las clases que conforman al sistema y tiene la finalidad de facilitar el desarrollo de aplicaciones que simulen algún aspecto del mundo real mediante el uso de clases que representen cosas reales.

Los diagramas de clases colaboran en lo referente al análisis. Permiten al analista hablarles a los clientes en su propia terminología, lo cual hace posible que los clientes indiquen importantes detalles de los problemas que requieren ser resueltos.

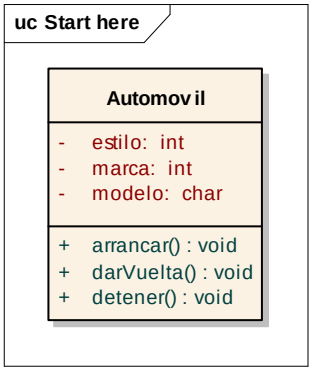


Figura 4.21. Clase

Visibilidad:
Define que objetos puede acceder y utilizar los atributos y operaciones de un objeto dado.

- (+) Público, todos pueden acceder a él.
- (#) Protegido, solo desde la mismo objeto y desde operaciones definidas en subclases.
- (-) Private, solo desde el mismo objeto.

Multiplicidad:

Es la cantidad de objetos que se relacionan con un objeto de la clase asociada.

Multiplicidad	Descripción
0	Opcional
1	Uno
*	Muchos
0..*	Cero a muchos
1 ..*	Uno a muchos

Relaciones

Indican la forma en que interactúan las clases del sistema, para el lenguaje UML existen las siguientes relaciones:

Asociación:

Describe **conexiones semánticas** entre los **objetos** individuales de clases dadas. Las asociaciones proporcionan las conexiones, con las cuales los objetos de diversas clases pueden interactuar (relación de conocimiento). La asociación se visualiza como una línea que conecta a ambas clases con el nombre de la asociación sobre la línea y la dirección de la asociación (Figura 4.22).

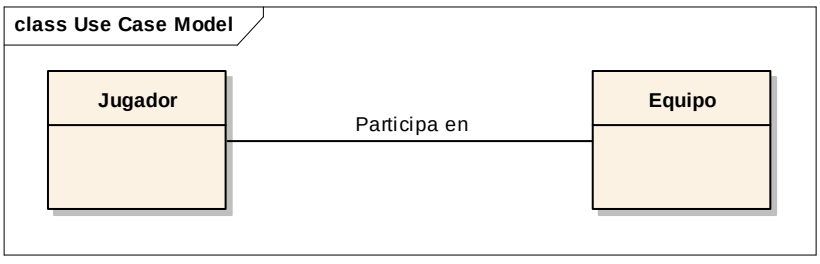


Figura 4.22. Relación de asociación

Una restricción (Figura 4.23) es una expresión booleana representada como una cadena interpretable en un determinado lenguaje. Para expresar restricciones se puede utilizar el lenguaje natural, notación de teoría de conjuntos, lenguajes de restricciones o varios lenguajes de programación.

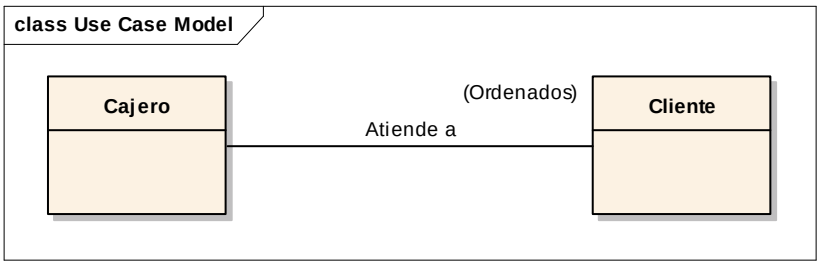


Figura 4.23. Restricciones

Agregación y composición:

Una agregación es una relación todo-partes donde los objetos de una clase son compuestos por objetos de otra (Figura 4.24). Por ejemplo un automóvil está compuesto por carrocería, motor, ruedas, etc.

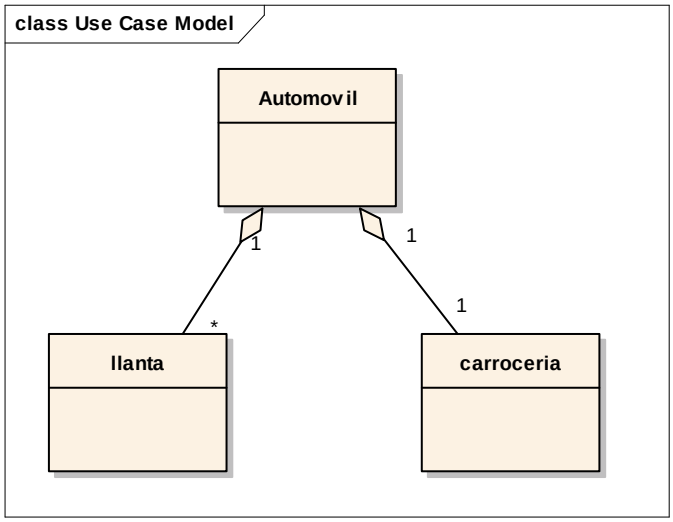


Figura 4.24. Agregación

Una composición es una forma de asociación más fuerte en la cual el compuesto es responsable de gestionar sus partes (Figura 4.25), por ejemplo asignación y desasignación.

La composición implica tres cosas

- Dependencia existencial: El elemento dependiente desaparece al destruirse el que lo contiene y, si es de multiplicidad 1, es creado al mismo tiempo.
- Hay una pertenencia fuerte: Se puede decir que el objeto contenido es parte constitutiva y vital del que lo contiene.

- Los objetos contenidos no son compartidos: esto es, no hacen parte del estado de otro objeto.

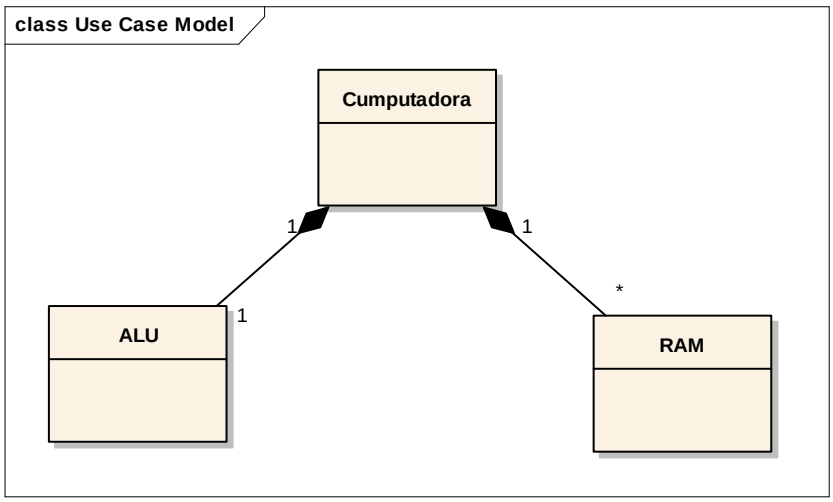


Figura 4.25. Composición

Herencia y generalización:

Propósitos de la generalización:

a) Principio de sustitución de Liskov:

Operaciones polimórficas. Una instancia de un descendiente se puede utilizar donde quiera que este declarado el antecesor.

b) Herencia:

Permitir la descripción incremental de un elemento que comparte descripciones con sus antecesores. Esto se llama Herencia.

La herencia es el mecanismo a través del cual los atributos, operaciones y restricciones, definidas para una clase, denominada *superclase*, pueden ser reutilizados (heredadas) por otras clase denominadas subclases (Figura 4.26). La herencia utiliza el principio *"es un tipo de ..."*. La subclase puede redefinir las operaciones heredadas, adicionalmente, una subclase puede definir nuevos atributos, operaciones y restricciones.

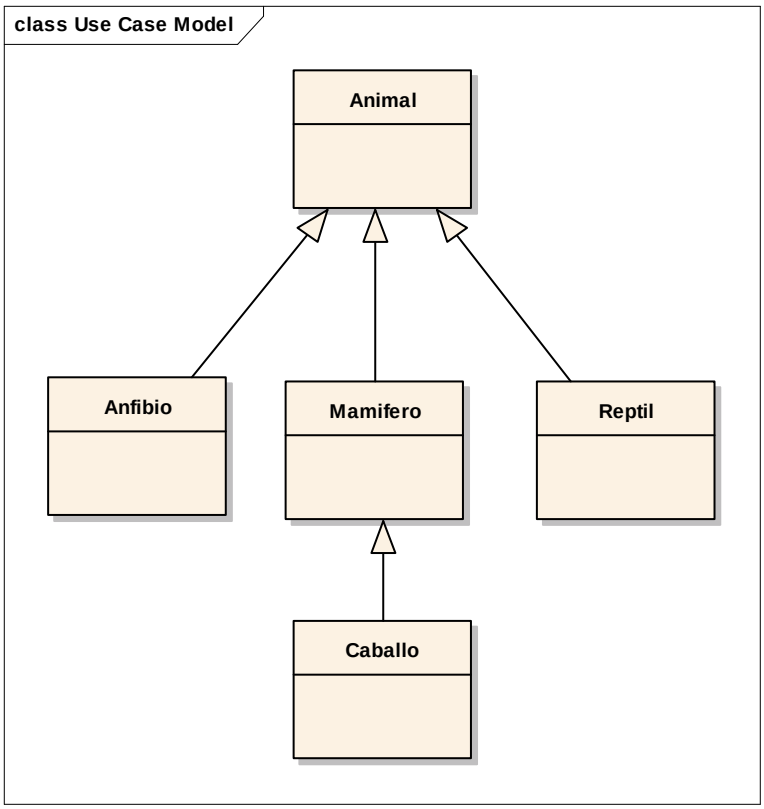


Figura 4.26. Herencia

Dependencias:

Es otro tipo de relación entre clases, indican que una clase utiliza a otra, su uso mas común es mostrar que la firma de un método de una clase utiliza a otra clase (Figura 4.27).

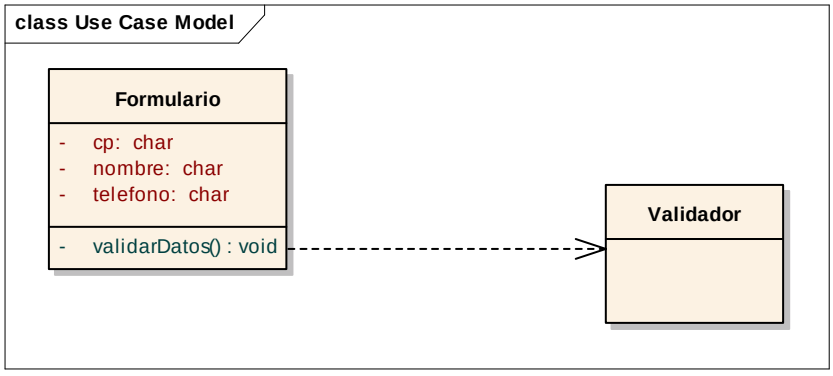


Figura 4.27. Relación de dependencia

Realización:

Una vez que se hayan creado las clases que conformaran al sistema, tal vez se den cuenta de que no pertenecen a una clase en particular o principal, sin embargo su comportamiento debe de incluir operaciones con las mismas firmas de la primer clase, esto se puede resolver mediante un elemento que conjunte una serie de operaciones comunes para un conjunto de clases del sistema y reutilizarlas para clases de otro sistema (Figura 4.28). La interfaz es un elemento de UML que permite especificar cierto aspecto de la funcionalidad de una clase y contiene un conjunto de operaciones o métodos que una clase presenta a otra.

La relación entre una clase y una interfaz es llamada realización.

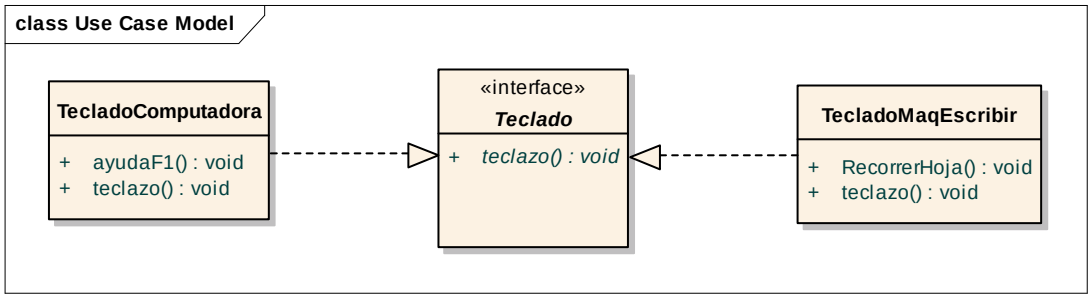


Figura 4.28. Realización

4.2.6 Diagrama de Objetos

Un objeto es una instancia de una clase, es una entidad que tiene valores específicos para sus atributos y métodos. El diagrama de objetos representa la interacción de los objetos en el sistema (Figura 4.29). La forma de representar a los objetos en UML es la siguiente:

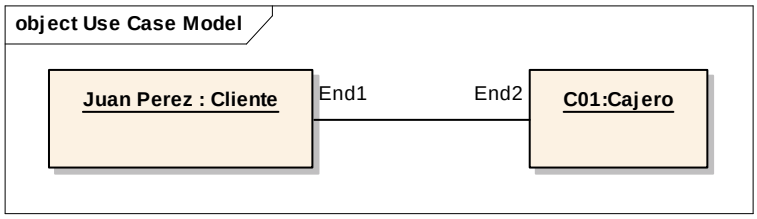


Figura 4.29. Diagrama de objetos

4.2.7 Diagrama de Colaboración

Es una extensión de un diagrama de objetos, además de las relaciones entre objetos, el diagrama de colaboraciones muestra los mensajes que se envían los objetos entre sí (Figura 4.30).

Para representar a los mensajes se utiliza una flecha cerca de la línea de asociación apuntando al objeto receptor (Figura 4.31).

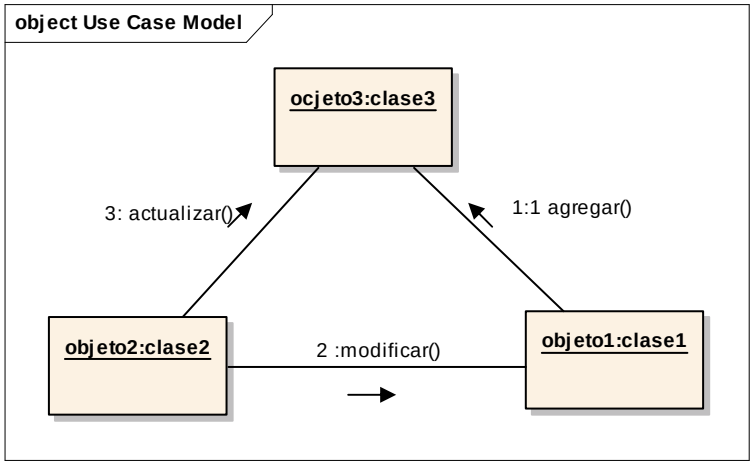


Figura 4.30. Mensajes entre objetos

Cambios de estado:

Es posible que representar cambios de estado de un objeto. Generalmente en un diagrama de colaboración un objeto se representa una sola vez. Sin embargo, si un objeto tiene distintos estados que se deban hacer explícitos (cambio de localización, o cambio de asociaciones) el objeto se puede representar más de una vez vinculando los diferentes símbolos con un flujo etiquetado <<become>> o <<conversion>>. Un flujo <<become>> es una transición de un estado a otro de un objeto. Se dibuja como una flecha con línea discontinua con el estereotipo <<become>>.

También se puede utilizar el estereotipo <<copy>> o <<copia>> que representa una copia de un objeto que a partir de dicho momento es independiente.

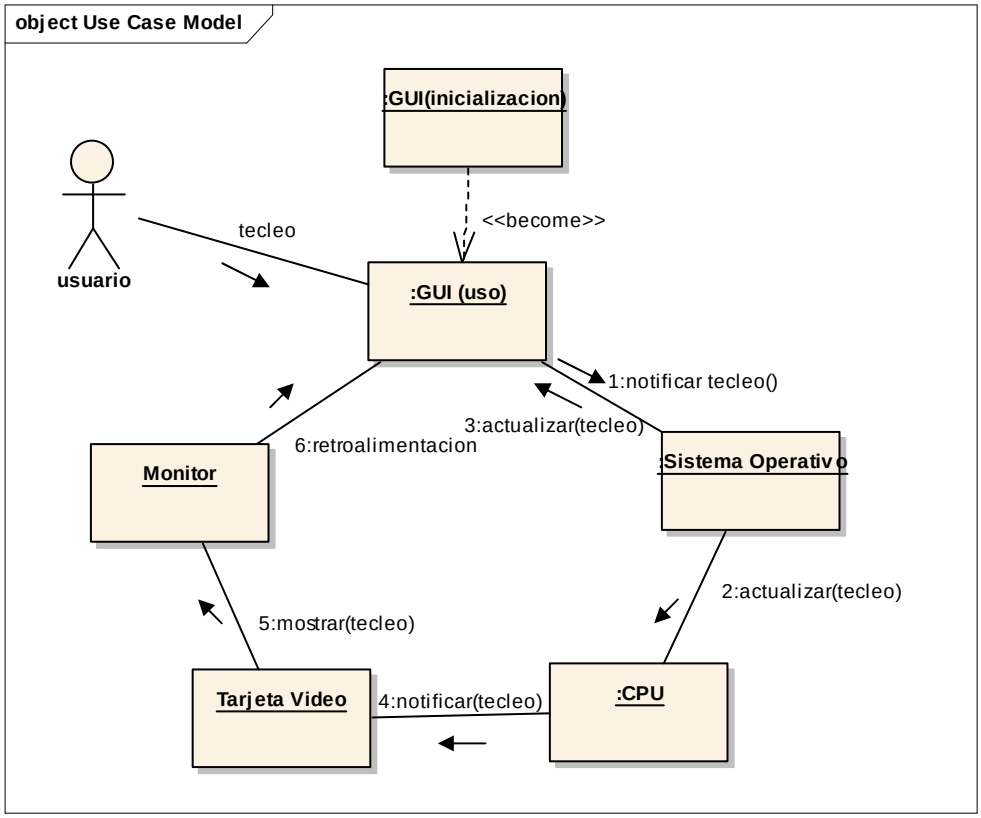


Figura 4.31. Diagrama de colaboración para el proceso de la PC para despliegue de datos en pantalla

4.2.8 Diagrama de Componentes

Un componente es una unidad física de implementación con interfaces bien definidas pensada para ser utilizada como parte reemplazable de un sistema. Cada componente incorpora la implementación de ciertas clases del diseño del sistema. Los componentes bien diseñados no dependen directamente de otros componentes sino de sus interfaces. El uso de interfaces permite evitar la dependencia directa entre componentes.

El diagrama de componentes contiene componentes, interfaces y las relaciones entre ellos.

Simbología:

La siguiente Figura (4.32) muestra los elementos utilizados en el diagrama de componentes:

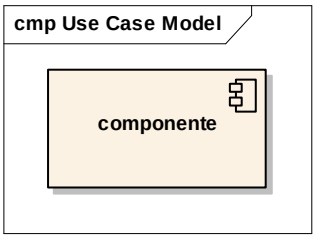
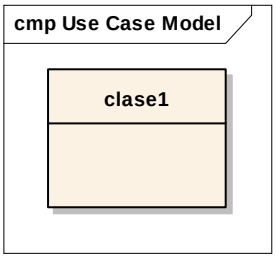
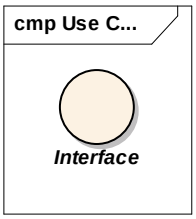
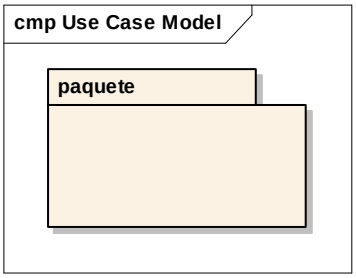
Elemento	**Símbolo**
Componente	
Clase (son las clases que implementa el componente)	
Interfaz	
Paquete	

Figura 4.32. Elementos del diagrama de componentes

En la Figura (4.33) se presenta un sistema de procesador de texto, este sistema implementa dos clases, procesador Texto y validador Ortográfico.

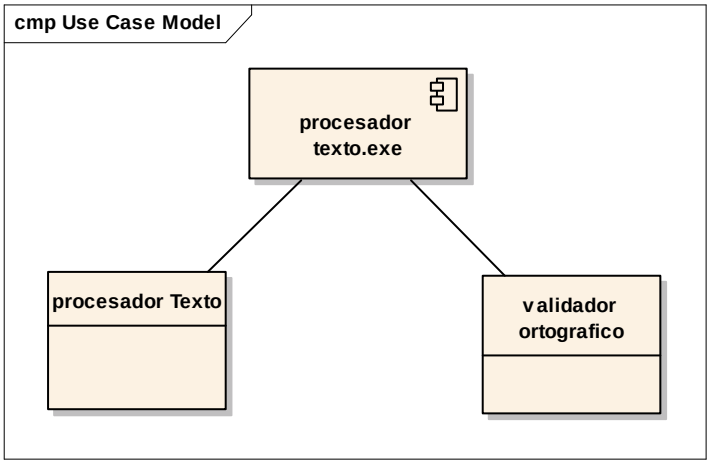


Figura 4.33. Relación entre componentes

En la Figura (4.34) se presenta la interacción entre los componentes de un sistema.

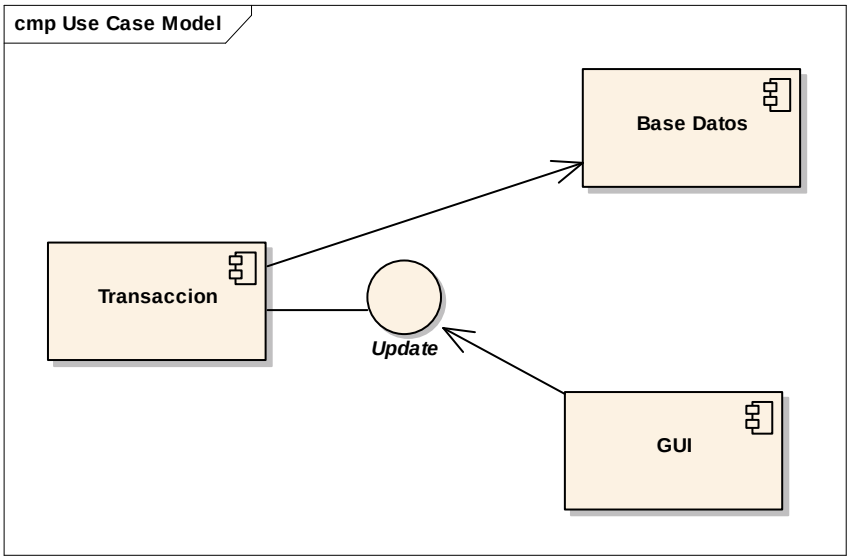


Figura 4.34. Diagrama de componentes para un sistema de transacciones

4.2.9 Diagrama de Distribución

El elemento primordial de hardware es un nodo, que es un nombre genérico para todo tipo de recurso de cómputo. Es posible usar 2 tipos de nodos: un procesador, el cual puede ejecutar un componente, y un dispositivo que no lo ejecuta. Normalmente un dispositivo tiene contacto con el mundo exterior.

En UML un cubo representa a un nodo, puedes usar un nombre para el nodo y un esteriotipo. El nombre es una cadena de texto, si el nodo es parte de un paquete, su nombre puede contener también el paquete, (Figura 4.35):

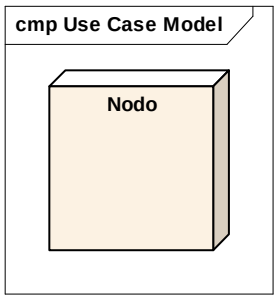


Figura 4.35. Nodo

Se puede dividir el cubo en comportamientos que agreguen información, como en la siguiente Figura 4.36:

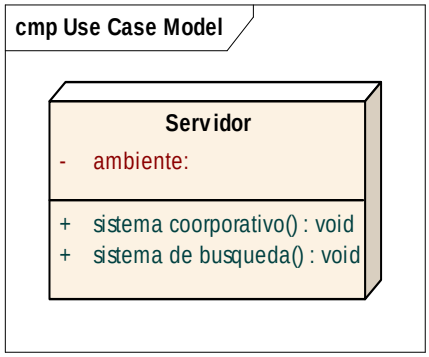


Figura 4.36. Dispositivo

Otra forma de indicar los componentes distribuidos es la de mostrarlos en relaciones de dependencia con un nodo, (Figura 4.37):

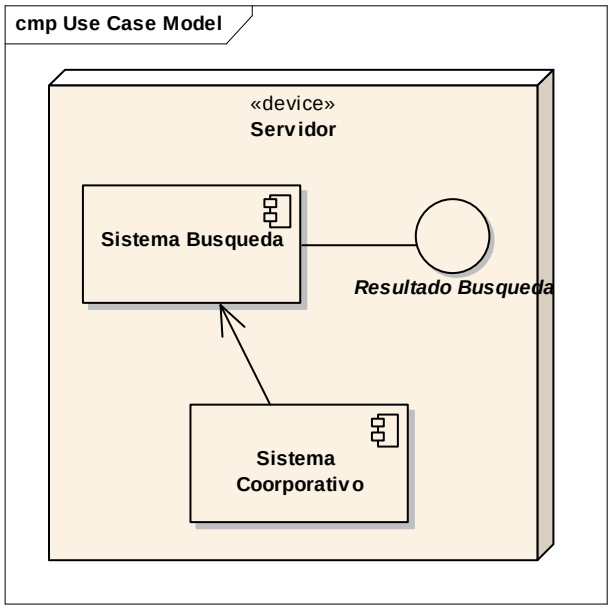


Figura 4.37. Relación de los componentes del dispositivo

Una línea asocia a los cubos representando una conexión entre nodos (Figura 4.38):

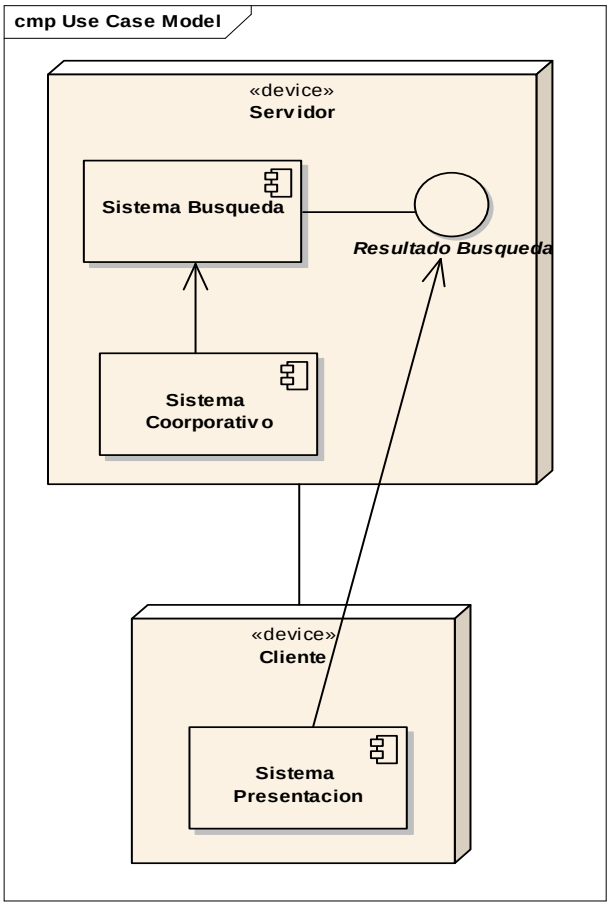


Figura 4.38 Diagrama de distribución para un sistema cliente servidor