CAPÍTULO II

METODOLOGÍAS ORIENTADAS A OBJETOS

2.1 Introducción

Las tecnologías orientadas a objetos se han convertido en uno de los motores clave de la industria del software y a la par se han hecho innovaciones en las metodologías utilizadas para el desarrollo de software. Esta evolución de metodologías tiene como propósito la creación de sistemas cada vez más complejos, acordes a las necesidades presentes.

Las tecnologías más comunes para el modelado de software son la perspectiva del análisis estructurado y la perspectiva orientada a objetos. En el análisis estructurado, la construcción principal del sistema es la función y el proceso, y el modelado se realiza en descomposición de funciones. En la perspectiva de modelado orientada a objetos el bloque principal de todo el sistema es la clase y el objeto, y el problema se analiza mediante el estudio de los objetos dentro del ámbito en que se presentan.

El modelado contemporáneo se ha inclinado por la metodología orientada a objetos, pues ha logrado derribar problemas que se resisten al análisis estructurado. Si bien las metodologías estructuradas marcaron la pauta para la construcción de software, la metodología orientada a objetos es un paradigma que ha demostrado ser de gran utilidad para la solución de problemas clásicos que se presentan al elaborar software, brindando, entre otros beneficios: software reutilizable, costos y tiempo de desarrollo menores, sistemas de alta calidad, fáciles de modificar y de mejorar.

Una metodología orientada a objetos abarca métodos de análisis, diseño e implantación, permitiendo especificar la conducta de la estructura del sistema, documentar el problema desde su construcción, además de proporcionar un anteproyecto del sistema final.

El auge de las tecnologías orientadas a objetos se debe fundamentalmente a la existencia y popularidad de varios lenguajes de programación orientados a objetos, tales como C++ o Java, por lo que es de vital importancia tener un buen entendimiento del tema.

Este capítulo es una introducción a las metodologías orientadas a objetos, en el cual analizaremos los conceptos básicos y la terminología sobre las metodologías más utilizadas, esto es, Booch, OMT y UML, siendo esta última la estandarización de las anteriores.

Debido a las ventajas que UML nos brinda al momento de especificar, visualizar, construir y documentar un sistema de software, es la metodología utilizada en el presente trabajo y, por consiguiente, será especialmente tratada.

2.2 Conceptos Fundamentales de Metodologías Orientadas a Objetos

Para comenzar con este capítulo, consideramos primordial mencionar los conceptos relacionados con las tecnologías orientadas a objetos, para un mejor entendimiento de éstas. *La orientación a objetos es un conjunto de disciplinas (ingeniería) que*

desarrollan y modelan software que facilitan la construcción de sistemas complejos a partir de componentes[1].

El atractivo de la orientación a objetos es que proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tan fielmente como sea posible. La orientación a objetos trata de cubrir las necesidades de los usuarios finales, así como las propias de los desarrolladores de productos de software.

A continuación mencionaremos las propiedades más importantes de la orientación a objetos.

2.2.1 Objetos

Definimos un objeto como un concepto, una abstracción o cosa con límites definidos y significativos para el problema en cuestión. Los objetos sirven para dos propósitos: promueven el entendimiento del mundo real y proporcionan una base práctica para la implantación computacional. La descomposición de un problema en objetos depende de la naturaleza del problema.

Un objeto es una instancia de una clase. Una idea fundamental en la orientación a objetos es la comunicación de los objetos a través del paso de mensajes.

2.2.2 Clases

Una clase es un concepto discreto dentro de la aplicación que se está modelando: una cosa física, de negocios, lógica o de una aplicación. Las clases son los focos alrededor de los cuales se organizan los sistemas orientados a objetos.

Una clase se puede definir como una descripción abstracta de un grupo de objetos con las mismas características, esto es, siguen una misma estructura, comportamiento y relaciones similares, cada una de las cuales se diferencia por su estado específico (atributos) y por la posibilidad de realizar una serie de operaciones (métodos).

Un atributo es una característica o propiedades de una clase y aplica a todos los objetos de dicha clase. Un método es la realización de una función, servicio o acción que se puede aplicar a todos los objetos de una clase para comportarse de cierta manera. El nombre de un método es generalmente un verbo que representa claramente el comportamiento de la clase.

2.2.3 Comunicación entre objetos

Un sistema orientado a objetos consiste en un conjunto de objetos que se comunican unos con otros llamando a sus métodos. Estos últimos residen en el objeto y determinan cómo actúan en el mismo cuando reciben un mensaje.

Un mensaje es la acción que hace un objeto. Un método es el procedimiento que se invoca para actuar sobre un objeto, esto es, especifica cómo se ejecuta un mensaje. El conjunto de mensajes a los cuales puede responder un objeto se denomina protocolo del objeto. Estos mensajes son los únicos conductos que conectan al objeto con el mundo externo.

[1] Joyanes Aguilar Luis, Programación Orientada a Objetos, 2ª. Ed., Osborne McGraw-Hill, 1998.

2.2.4 Herencia

La herencia es una propiedad que permite a los objetos ser construidos a partir de otros objetos. El objetivo final es la reutilización de código, es decir, utilizar las clases anteriormente desarrolladas.

La herencia supone una *clase base* y una *jerarquía de clases* que contiene las clases derivadas de la clase base. Las clases derivadas pueden heredar características de su clase base, pero añaden otras nuevas características propias. Las clases que heredan propiedades de una clase base pueden a su vez servir como definiciones base de otras clases.

Existen dos mecanismos utilizados normalmente: herencia simple y herencia múltiple. En la herencia simple, una clase puede tener solo un ascendente, o dicho de otro modo, una subclase puede heredar características de una única clase. La herencia múltiple es la propiedad de una clase de poder heredar características de más de una clase. Básicamente, todo lo que se puede hacer con herencia múltiple se puede hacer con herencia simple, aunque a veces resulta más difícil utilizar herencia múltiple debido a que se llegan a combinar diferentes tipos de objetos, cada uno de los cuales define métodos o atributos iguales y con esto pueden surgir problemas de ambigüedad.

2.2.5 Polimorfismo

Esta propiedad supone que un mismo mensaje puede producir acciones (resultados) totalmente diferentes cuando se reciben por objetos diferentes.

Con esta propiedad un usuario puede enviar un mensaje genérico y dejar los detalles de la implantación exacta para el objeto que recibe el mensaje (ya sea un objeto padre o hijo), por esto último, podemos deducir que el polimorfismo se fortalece con el mecanismo de herencia.

2.2.6 Sobrecarga

La sobrecarga es una clase de polimorfismo. La sobrecarga es una propiedad en la cual se utiliza el mismo nombre de una operación para representar operaciones similares que se comportan de modo diferente cuando se aplican a clases diferentes o de la misma. Por consiguiente, los nombres de las operaciones se pueden sobrecargar, esto es, las operaciones se definen en clases diferentes o en la misma, pudiendo tener nombre idénticos, aunque su código programado puede diferir.

Si el nombre de una operación se utiliza para nuevas definiciones en clases de una jerarquía, la operación del nivel inferior anula la operación del nivel más alto.

La sobrecarga elimina la necesidad de definir métodos diferentes que en esencia hacen lo mismo, también hace posible que un método se comporte de una u otra forma según el número de argumentos con el que sea invocado.

2.2.7 Abstracción

La abstracción es la propiedad que permite representar las características esenciales de un objeto, sin preocuparse de las características restantes (no esenciales).

Una abstracción se centra en la vista externa de un objeto, de modo que sirva para separar su comportamiento esencial de la implantación. Definir una abstracción significa describir una entidad del mundo real, no importa lo compleja que pueda ser.

Es importante aclarar que una clase abstracta sirve como base común para otras clases, pero no tendrá instancias.

2.2.8 Encapsulamiento

El encapsulamiento o encapsulación es la propiedad que permite asegurar que el contenido de la información de un objeto está oculto al mundo exterior. La encapsulación es ver al objeto como una caja negra, la cual, sólo sabemos utilizarla pero no conocemos cómo fue hecha ni su contenido.

La encapsulación permite la división de un sistema en módulos. Éstos se implantan mediante clases, de forma que una clase representa la encapsulación de la abstracción. Cada clase debe constar de dos partes: una interfaz y una implantación. La interfaz de una clase captura sólo su vista externa y la implantación contiene la representación de la abstracción, así como los mecanismos que realizan el comportamiento deseado.

2.2.9 Modularidad

La modularidad es la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), en las que cada una debe ser tan independiente como sea posible de la aplicación en sí y de las partes restantes.

2.2.10 Jerarquía

La jerarquía es una propiedad que permite una ordenación de las abstracciones. Las dos jerarquías más importantes de un sistema complejo son: estructura de clases (generalización/especialización) y estructura de objetos (agregación), las cuales serán detalladas más adelante.

2.3 Metodología orientada a objetos "Booch"

Este método marcó la conducta en el análisis del diseño orientado a objetos. Booch lo modela desde un punto de vista lógico para representar gráficamente clases, objetos y las relaciones entre ellos.

La orientación a objetos que Booch propone es iterativa, creciente y evolutiva, utilizando la combinación de un concepto llamado *microproceso* y *macroproceso* de desarrollo.

2.3.1 Macroproceso

El *Macroproceso* se inclina a observar el problema desde el punto de vista global, esto es, la visión que podría tener un administrador o el líder de proyecto. Sirve como marco de referencia para controlar al microproceso y dicta una serie de procesos y actividades, que permiten al equipo de desarrollo evaluar el riesgo de forma que se centren mejor las actividades de análisis y diseño de equipo.

Las fases que constituyen al macroproceso son:

- La conceptualización. Establece los requerimientos básicos.
- Análisis. Desarrollo de un modelo de acuerdo al comportamiento del problema.
- Diseño. Creación de la arquitectura a detalle.
- Evolución o Puesta en Marcha. Se realiza la prueba, corrección y entrega, tantas veces como sea necesario y evoluciona hacia la versión para el usuario final.
- Mantenimiento. Se realiza después de la entrega. Cabe mencionar que si esta etapa provoca nuevas ampliaciones del sistema que en la primera fase no fueron previstas, entonces se comenzará nuevamente el macroproceso.

2.3.2 Microproceso.

El *Microproceso* está orientado a una enfoque más estrecho; es la visión que un programador podría tener del sistema. El *microproceso* consta de 4 etapas:

- Identificación de las clases y objetos a un nivel de abstracción.
- Identificación de semánticas de estas clases y objetos.
- Identificación de las relaciones entre esas clases y objetos.
- Implantación de las clases y objetos.

Además, ofrece cuatro técnicas para la documentación de:

- *Diagrama de clases*, los cuales indican la existencia de clases y sus relaciones.
- *Diagrama de objetos*, los cuales son usados para mostrar la existencia de objetos y su comportamiento.
- *Diagrama de estado-transición*, los cuales muestran los estado posibles de cada objeto.
- *Diagramas de tiempo*, los cuales indican la secuencia de operaciones de los objetos.

2.3.3 Notación Booch

Booch propone su notación pre-UML para representar gráficamente clases, objetos y relaciones entre ellos, siendo sus nubes lo más sobresaliente. Con ellas denota clases (nubes con líneas punteadas) y objetos (nubes con líneas continuas) ambas conteniendo su nombre, atributos y métodos. También proporcionó la notación para las diferentes relaciones entre clases (de asociación, agregación y herencia).

2.3.3.1 Objetos

Los objetos se dibujan usando una nube con el borde continuo (Figura 2.1). Cada objeto contiene su nombre, atributos y operaciones.

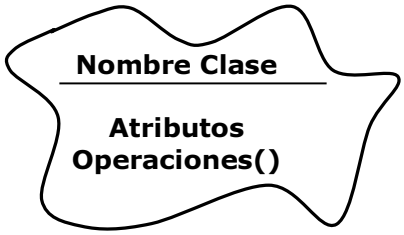

Figura 2.1 Notación de un Objeto en Booch

2.3.3.2 Clases

Una clase está representada por una nube punteada (Figura 2.2). Dentro de ella contiene su nombre, sus atributos y operaciones. Los atributos son las características que representan a la clase y las operaciones son las funciones o acciones que se pueden aplicar a los objetos de una clase.

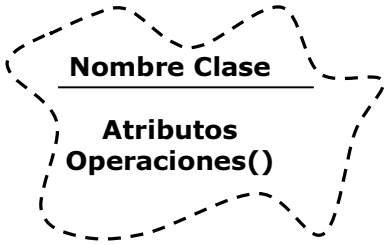

Figura 2.2 Notación de una Clase en Booch

Para proporcionar información adicional de una clase, esta notación permite utilizar símbolos que a continuación se describen:

A – Abstracta. Es una clase que puede usarse solamente como base de alguna otra clase. Es usada para definir el comportamiento común de otras clases llamadas comúnmente subclases. Además, no puede ser instanciada porque representa una gran variedad de objetos. Su representación gráfica se muestra en la Figura 2.3.

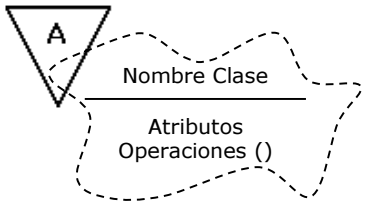

Figura 2.3 Notación de una Clase Abstracta en Booch

F – Friend (Amiga). Las clases de este tipo admiten funciones no públicas y otras clases (Figura 2.4).

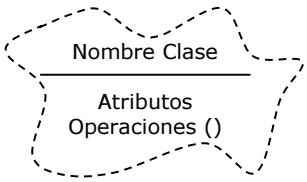

Figura 2.4 Notación de una Clase Amiga en Booch

S – Static. Una clase estática (static) provee datos Su representación se muestra en la Figura 2.5.

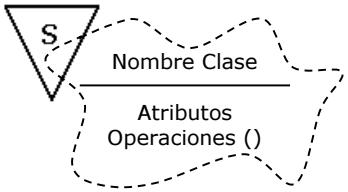

Figura 2.5 Notación de una Clase Estática en Booch

V – Virtual. Una clase Virtual comparte una clase base, es la clase más general en el sistema (Figura 2.6).

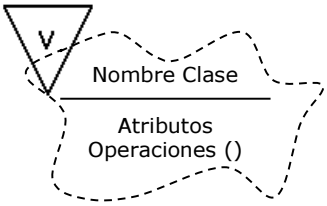

Figura 2.6 Notación de una Clase Virtual en Booch

Esta metodología también incorporó un símbolo para englobar clases, que representan un cluster de clases similares. Su notación es un rectángulo con compartimentos (Figura 2.7).

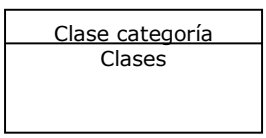

Figura 2.7 Notación de un Cluster en Booch

2.3.3.3 Relaciones

Para denotar relaciones entre objetos, Booch utilizó líneas y flechas (Figura 2.8).

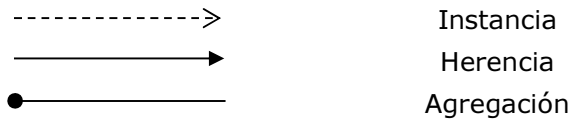

Figura 2.8 Notación de Relaciones en Booch

La metodología Booch, sin embargo, no alcanzó a cubrir totalmente la necesidad de crear sistemas cada vez más novedosos y complejos, por lo que surgieron otras metodologías.

2.4 Metodología orientada a objetos "OMT"

Aunque en la evolución de las diferentes metodologías se han utilizado modelos orientados a objetos, muchas de éstas no han tenido soportes suficientes para desarrollar aplicaciones de bases de datos o no ponen énfasis en la importancia de la estructura de datos.

OMT (Object Modeling Technique), sin embargo, pone atención en la importancia del modelo y uso del mismo para lograr una abstracción, en la cual el análisis está enfocado al mundo real para un nivel de diseño, además, agrega especial interés para el modelado de los recursos computacionales.

El objetivo principal de OMT es la conceptualización de una entidad que permita manejar atributos, asociaciones y sus entidades, así como obtener una mejor interacción con la base de datos. Por todo ello, OMT llegó a ser líder en el avance de las metodologías orientadas a objetos.

La metodología de modelado de objetos OMT descrito por Rumbaugh, tiene como objetivo desarrollar un modelo de sistemas, mediante objetos y las relaciones entre ellos.

2.4.1 Fases de la metodología OMT

Las fases utilizadas en la metodología OMT son: análisis del sistema, diseño del sistema y diseño de objetos.

2.4.1.1 Análisis del sistema

Su objetivo es desarrollar un modelo del sistema. El modelo se expresa en función de objetos, relaciones entre ellos, flujo dinámico de control y las transformaciones funcionales. Los pasos a seguir en el análisis son:

- Se obtiene una descripción inicial del problema.
- Se construye el *modelo de objetos* y sus relaciones. Su objetivo es describir la estructura estática del software, esto es: se abstraen los conceptos de los datos que son más importantes para la aplicación y se describen gráficamente por los diagramas de objetos que definen las clases y sus relaciones.

- Se desarrolla un *modelo dinámico*. Su objetivo es describir los aspectos del sistema que cambian conforme pasa el tiempo.
- Se construye un *modelo funcional*. Su objetivo es describir las transformaciones de los datos del sistema.
- Se verifican, iteran y depuran los tres modelos.

2.4.1.2 Diseño del sistema

En esta etapa se define la arquitectura del sistema. Los pasos para el diseño del sistema son:

- Organizar el sistema en subsistemas y ordenarlos en capas y divisiones.
- Identificar la concurrencia inherente en el problema.
- Asignar subsistemas a procesos.
- Definir la estrategia de implantación del administrador de datos.
- Identificar las fuentes globales y definir el mecanismo para controlar el acceso a ellos.
- Elegir un enfoque para la implantación de control de software.
- Considerar las condiciones de los límites.
- Establecer cambios fuera de las prioridades.

2.4.1.3 Diseño de objetos

Su objetivo es depurar el modelo del análisis y proporcionar una base detallada para la implantación, tomando en cuenta el ambiente en que se construye. Los pasos que se realizan en el diseño de objetos son los siguientes:

- Se depuran las operaciones para el modelado de objetos.
- Se diseñan algoritmos para las operaciones y estructuras de datos.
- Se optimizan las vías de acceso a los datos.
- Se construye un sistema controlado por procedimientos.
- Se empaquetan las clases y las asociaciones en módulos.

2.4.2 Notación OMT

La metodología OMT, igual que las metodologías predecesoras, representa gráficamente conceptos claves en la estructura del sistema tales como: clases, objetos y asociaciones. Esta notación si bien, no tiene la gama de opciones que las metodologías actuales manejan, sí mantiene características afines con las actuales.

2.4.2.1 Objetos

Los objetos en OMT se representan con un rectángulo redondeado con su respectivo nombre, atributos y valores (Figura 2.9).

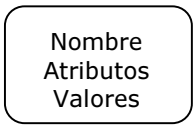

Figura 2.9 Notación de un Objeto en OMT

2.4.2.2 Clases

Las clases se representan por rectángulos con tres compartimentos. En el primero se pone el nombre de la clase, en el segundo los atributos y en el tercero las operaciones o métodos (Figura 2.10).

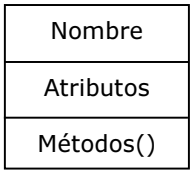

Figura 2.10 Notación de una Clase en OMT

2.4.2.3 Asociaciones

La asociación es una dependencia entre dos o más clases. Las asociaciones se representan por líneas que unen a las clases (Figura 2.11).

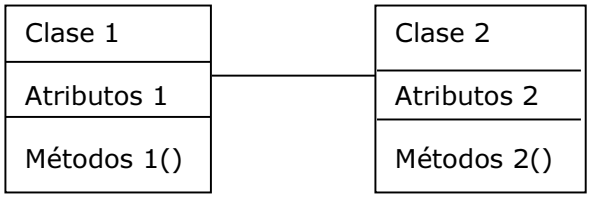

Figura 2.11 Notación de una Asociación en OMT

La multiplicidad de asociaciones especifica cuantas instancias de una clase pueden relacionarse a cada instancia de otra clase. En el caso más general, la multiplicidad puede ser especificada con un número o un conjunto de intervalos, tales como "1" (exactamente uno), "1+" (uno o más), "3-5" (tres a cinco) y "2,4,18" (dos, cuatro o dieciocho).

Existen terminadores de línea especiales para indicar ciertos valores de multiplicidad comunes. Un círculo relleno es un símbolo de OMT para "muchos" y significa cero o más. Un círculo vacío indica "opcional", indica cero o uno. Las líneas sin símbolos de multiplicidad indican una asociación uno a uno. En el caso general, el grado de multiplicidad se escribe a un lado del extremo de la línea, por ejemplo, "1+" para indicar uno o más (Figura 2.12).

Esta metodología, a pesar de que cubrió puntos que Booch u otras metodologías no habían satisfecho, quedó muy lejos de ser el estándar para las nuevas tecnologías en el desarrollo de sistemas.

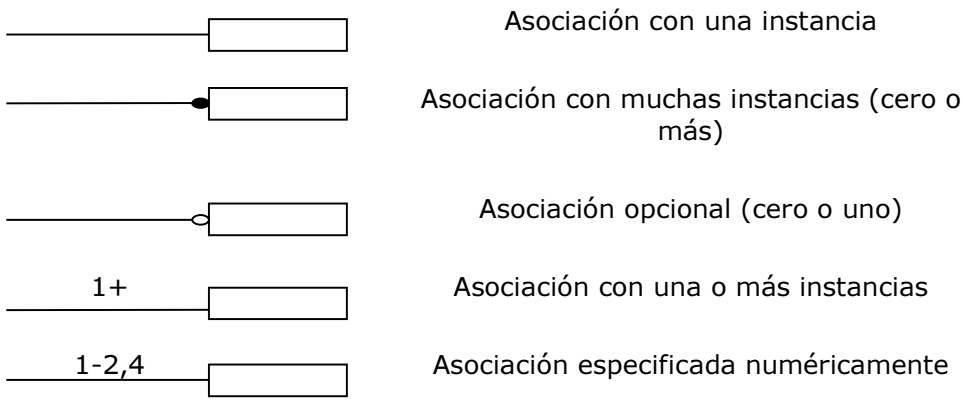

Figura 2.12 Notación de Multiplicidad en OMT

2.5 Metodología orientada a objetos "UML"

UML (Unified Modeling Language), es probablemente, una de las mayores innovaciones conceptuales en el mundo tecnológico del desarrollo del software. Es actualmente el estándar de la industria que requiera la construcción de modelos como condición previa al diseño y después a la construcción de prototipos.

UML se diseñó combinando una gran cantidad de estándares, si embargo se rige a través de tres metodologías precedentes (Booch, OMT y Objectory) y de la colaboración de J. Rumbaugh, G. Booch e I. Jacobson, además del análisis y estudio de alrededor de 20 métodos estándares.

Este lenguaje de modelado es de propósito general, el cual puede ser usado por todos los modeladores, no tiene propietario y está basado en el común acuerdo de gran parte de la comunidad informática. Está pensado en reemplazar al menos los modelos de Booch, OMT, entre otros, y es utilizado para entender, diseñar, configurar, mantener y controlar la información de sistemas computacionales.

2.5.1 Notación UML

La notación que esta metodología maneja comprende el comportamiento de todo el sistema, permitiendo la realización de diagramas que lo describen. Las metas claves de UML se encuentran en integrar la terminología más aceptada comúnmente y, con ello, permitir contar con software avanzado que proporcione diagramas mejor definidos para el desarrollo del sistema.

2.5.1.1 Objetos

La notación del objeto es un rectángulo con dos compartimentos. El compartimiento superior contiene el nombre del objeto y el nombre de la clase a la que pertenece, ambos subrayados para distinguirlo como individuo. El compartimiento inferior

contiene la lista de nombres y valores de atributos. En ellos no existe necesidad de mostrar sus operaciones porque son las mismas para todos los objetos de la clase (Figura 2.13).

Figura 2.13 Notación de un Objeto en UML

2.5.1.2 Clases

Las clases en UML se dibujan como rectángulos. Las listas de atributos y operaciones se dibujan en compartimentos separados (Figura 2.14). Los compartimentos pueden ser suprimidos cuando no es necesario el detalle completo. Además una clase puede aparecer en varios diagramas.

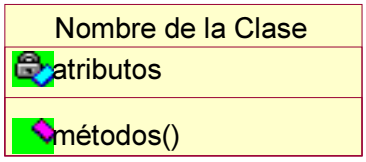

Figura 2. 14 Notación de una Clase en UML

2.5.1.3 Relaciones

Las relaciones entre clases son asociaciones, generalizaciones, agregaciones y varias clases de dependencia, que incluyen la de realización y de uso.

Las relaciones entre clases se dibujan como líneas que conectan rectángulos de clases. Los distintos tipos de relaciones se diferencian por la textura de la línea y por los adornos de las mismas en sus extremos.

2.5.1.3.1 Relación de asociación

La relación de asociación proporciona las conexiones con las cuales los objetos de diversas clases pueden interactuar. Una asociación describe conexiones discretas entre objetos u otras instancias de un sistema.

Las asociaciones llevan la información sobre relaciones entre objetos en un sistema. Éstas mantienen unido un sistema. Sin ellas no hay más que clases aisladas que no trabajan juntas.

Cada conexión de una asociación a una clase se llama extremo. Los extremos de una asociación pueden tener nombres llamados *roles* y éstos son nombrados de acuerdo al tipo de asociación que exista entre las clases. La propiedad más importante que tienen las asociaciones es la multiplicidad, ésta indica cuántas instancias de una clase se pueden relacionar con una instancia de otra clase.

La direccionalidad es otra característica importante de las asociaciones y se determina mediante una triángulo indicando la dirección de la asociación.

La notación UML para una asociación es una línea continua que conecta a dos clases. El nombre de la asociación se pone a lo largo de la línea con el nombre del rol y la multiplicidad en cada extremo. Las multiplicidades más comúnmente utilizadas en UML son:

- Muchos (*)
- 0 *ó* 1 (0..1)
- 1 (1)
- m *a* n (m..n)

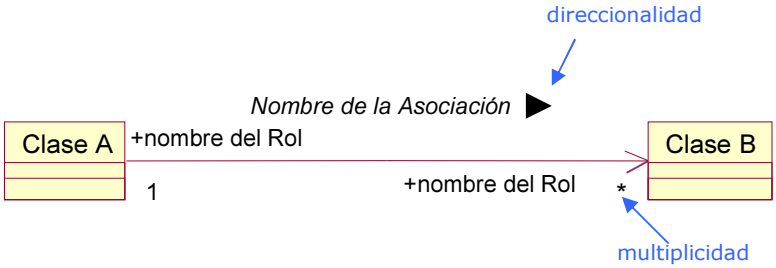

Figura 2.15 Notación de Asociación en UML

La Figura 2.15 muestra una multiplicidad de 1 a muchos (*), esto es, que una instancia de la *Clase A* puede relacionarse con muchas instancias de la C*lase B.* De la misma manera podemos hacer uso de las multiplicidades anteriormente mencionadas.

Una asociación puede también tener atributos por sí misma, en cuyo caso es una asociación y una clase, conocida como una clase asociación (Figura 2.16).

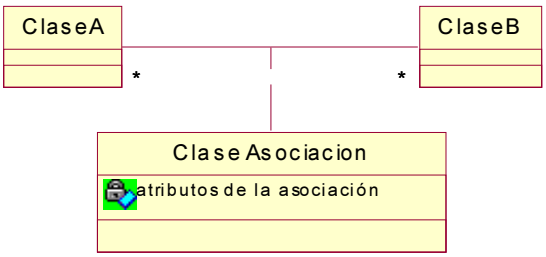

Figura 2.16 Notación de Clase Asociación en UML

Si un atributo de la asociación es único dentro de un conjunto de objetos relacionados, entonces es un calificador. El calificador es un valor que selecciona un objeto único del conjunto de objetos relacionados a través de la asociación (Figura 2.17).

Figura 2.17 Notación de Asociación con calificador en UML

2.5.1.3.2 Relación de generalización

La relación de generalización relaciona descripciones generales de la clase padre (superclase) con clases hijos (subclases). La generalización permite compartir atributos, operaciones y relaciones en común, de diferentes clases sin tener que repetirlas.

En la generalización la clase padre define los métodos y atributos de manera más general, mientras que la clase hija se extiende de la clase padre definiendo más específicamente los métodos de la superclase.

Una generalización se dibuja como una flecha desde el hijo al padre, con un triángulo hueco en el extremo conectado con el padre (Figura 2.18).

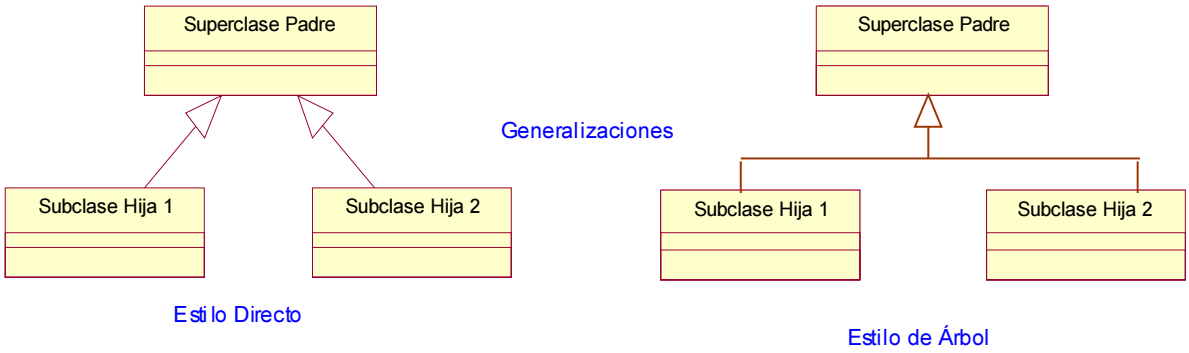

Figura 2.18 Notación de Generalización en UML

La generalización permite operaciones polimórficas, es decir, operaciones cuya implantación (método) es determinada por la clase objeto a la que se aplican, en vez de ser indicada explícitamente. Esto funciona porque una clase padre puede tener muchos hijos posibles, cada uno de los cuales implanta su propia variación de una operación que se define a través de todo el conjunto de clases.

El propósito de la generalización es permitir la descripción incremental de un elemento que comparte las descripciones de sus antecesores (herencia). Al compartir, se reduce el tamaño del modelo y más importante aún, se reduce el número de los cambios y la posibilidad de inconsistencia accidental al momento de realizar una actualización al modelo.

2.5.1.3.3 Relación de agregación y composición

Estas relaciones representan una relación todo – partes, donde *todo* se refiere a la clase principal y las *partes* son piezas de la clase principal.

La agregación es aquella en que las partes pueden ser partes en cualquier todo. Además la multiplicidad, en el *lado todo*, debe ser distinta de uno.

En la composición existe un alto grado de pertenencia, esto es, las partes existen dentro del todo y se destruyen junto con él. Además, la clase *todo* es la responsable de gestionar sus partes.

La notación de agregación es un diamante hueco en el extremo de la trayectoria unida a la clase agregada (Figura 2.19 (a)). Por otra parte, en la composición la *clase todo* se muestra con un diamante relleno (Figura 2.19 (b)).

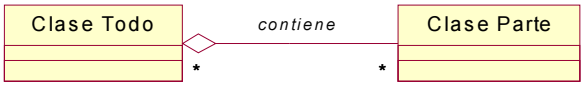

Figura 2.19(a) Notación para la agregación en UML

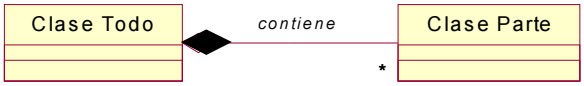

Figura 2.19(b)Notación para composición en UML

2.5.1.3.4 Relación de realización

La relación de realización relaciona una especificación con una implantación. Una interfaz es una especificación del comportamiento sin la implantación; una clase incluye la estructura de implantación. Una o más clases pueden realizar una interfaz, y cada clase implementa las operaciones de la interfaz.

La realización se indica con una flecha de línea discontinua con una punta de flecha hueca cerrada (Figura 2.20).

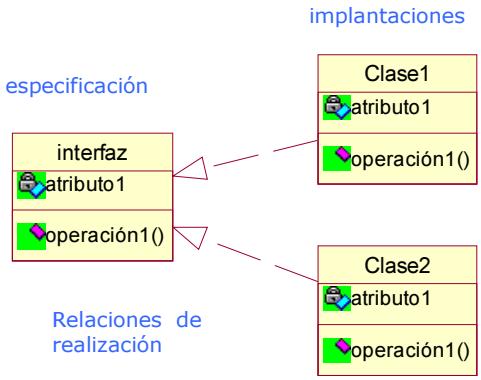

Figura 2.20 Notación de Realización en UML

Existe una notación reducida para mostrar interfaces sin su contenido y las clases o componentes que las realizan. Se muestra la interfaz con un círculo pequeño, unido al rectángulo de la clase por una línea continua (Figura 2.21).

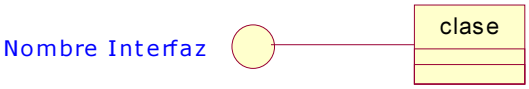

Figura 2.21 Notación reducida de Realización en UML

2.5.1.3.5 Relación de dependencia de uso

La dependencia de uso es una declaración en la cual el comportamiento de un elemento (el proveedor) afecta el comportamiento de otro elemento (cliente). En otras

palabras, establece que un elemento requiere la presencia de otro elemento para su correcto funcionamiento. Los estereotipos de uso incluyen la llamada y la instanciación, sin embargo, está abierto a otros tipos. La dependencia de llamada indica que un método, en una clase, llama a una operación en otra clase; la instanciación indica que un método, en una clase, crea una instancia de otra clase.

Una dependencia de uso se dibuja como una flecha discontinua desde la clase cliente hasta la clase proveedor, con la palabra clave para distinguirla (Figura 2.22).

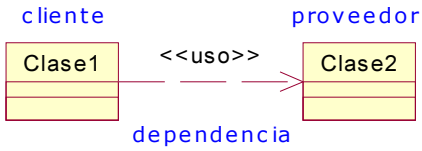

Figura 2.22 Notación de Dependencia de Uso en UML

2.5.2 Vistas de UML

En UML no hay ninguna línea entre los diferentes conceptos y construcciones, pero por conveniencia se puede dividir en varias vistas, donde una vista es un subconjunto de UML que modela construcciones que representan un aspecto del sistema.

En el nivel superior, las vistas se pueden dividir en tres áreas: clasificación estructural, comportamiento dinámico y gestión del modelo.

La clasificación estructural describe los elementos del sistema y sus relaciones con otros elementos. La clasificación de las vistas en este punto incluye la vista estática, la vista de casos de uso, la vista de implantación y la vista de despliegue.

El comportamiento dinámico describe la conducta de un sistema en el tiempo. El comportamiento se puede describir como una serie de cambios en el sistema a partir de la vista estática. Las vistas de comportamiento dinámico incluyen la vista de máquina de estados, la vista de actividad y la vista de interacción.

La gestión del modelo describe la organización de los propios modelos en unidades jerárquicas. La gestión del modelo cruza las otras vistas y las organiza para el trabajo de desarrollo y control de configuración.

Resumiendo: dependiendo del área y la vista en que se encuentre, se desarrollan diferentes diagramas:

✓ Área Estructural.

Vista Estática: Diagrama de clases.

Vista de Casos de Uso: Diagrama de casos de uso.

Vista de Implantación: Diagrama de componentes.

Vista de Despliegue: Diagrama de despliegue.

✓ Área dinámica.

Vista de Máquina de Estados: Diagrama de estados.

Vista de Actividad: Diagrama de actividad.

Vista de Interacción: Diagrama de secuencia y diagrama de colaboración.

✓ Gestión del modelo.

Vista de Gestión del Modelo: Paquetes.

2.5.2.1 Vista estática

La vista estática modela conceptos principales de la aplicación. Esta visión no describe el comportamiento del sistema dependiente del tiempo. Los componentes principales de la vista estática son las clases y sus relaciones.

La visión estática se exhibe en los diagramas de clases (Figura 2.23), llamados así por que su objetivo principal es la definición de las características de cada una de las clases, las interfaces, las colaboraciones, relaciones de asociación, relaciones de generalización, de realización, de dependencia, de herencia y de agregación.

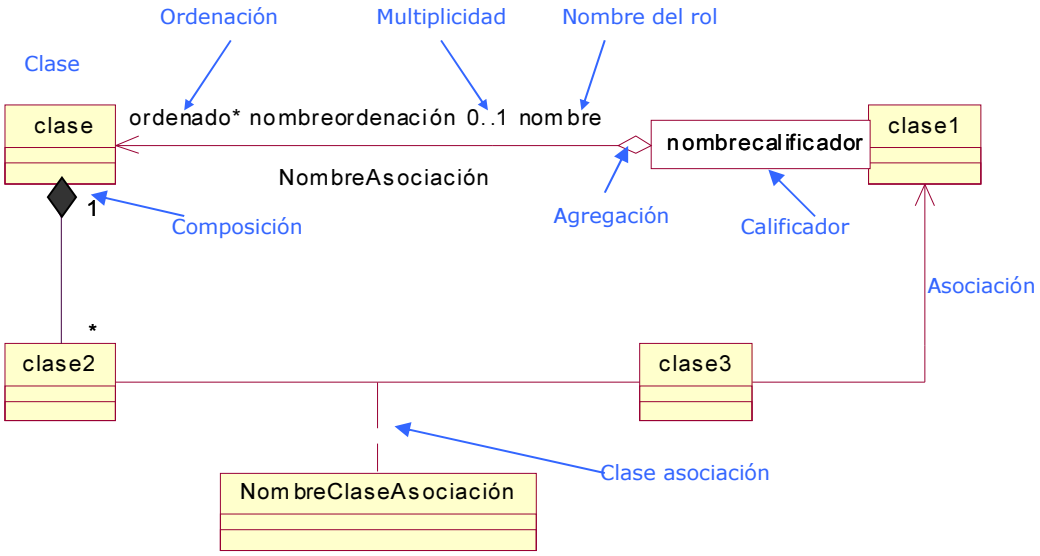

Figura 2.23 Diagrama de Clases en UML

2.5.2.2 Vista de casos de uso

La vista de casos de uso modela la funcionalidad del sistema según lo perciben los usuarios exteriores, llamados actores. El diagrama de caso de uso es muy útil para especificar que va hacer el sistema, su comportamiento, y no como va lo va a realizar.

El diagrama de casos de uso consiste en actores y casos de uso. Los actores son usuarios externos que interactúan con el sistema, aunque también se pueden ser otros sistemas. Los actores se dibujan como "muñecos" de palo (Figura 2.24).

Los casos de uso representan los escenarios que el sistema atraviesa en respuesta a un estímulo desde un actor. Éstos se dibujan como elipses.

Figura 2.24 Notación de Actor y Caso de Uso en UML

2.5.2.2.1 Relaciones de casos de uso

Existen varias relaciones para unir a los actores con los casos de uso y así describir el comportamiento del sistema (Figura 2.29).

La relación de asociación es una línea continua, denota que existe comunicación entre el actor y el caso de uso en el que participa (Figura 2.25).

Figura 2.25 Notación de Asociación en los Casos de Uso.

La relación de extensión se puede definir como la inserción de comportamiento adicional en un caso base que no tiene conocimiento sobre él. Puede haber varias extensiones del mismo caso de uso base, y pueden ser aplicadas conjuntamente. Esta relación se especifica con una flecha continua, donde la flecha está con dirección hacia el caso de uso base (Figura 2.26) además, debe llevar la leyenda de *extend* sobre la línea flecha.

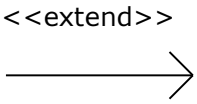

Figura 2.26 Notación de Extensión en los Casos de Uso

Por otra parte, aunque cada instancia de un caso de uso es independiente, la descripción de un caso de uso se puede descomponer en factores de otros casos de uso más simples. Un caso de uso puede incorporar el comportamiento de otros casos de uso como fragmentos de su propio comportamiento. Esto se llama relación de inclusión, se describe con una flecha continua con la leyenda *include* sobre la línea y está direccionada al caso de uso a ser incluido (Figura 2.27).

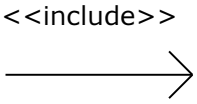

Figura 2.27 Notación de Inclusión en los Casos de Uso

Un tipo de relación muy utilizada también dentro de los diagramas de casos de uso es la generalización. Ésta es una relación entre un caso de uso general y un caso de uso más específico que hereda y añade propiedades. Se dibuja como una flecha continua, apuntando hacia el caso de uso padre o base (Figura 2.28).

Figura 2.28 Notación de Generalización en los Casos de Uso

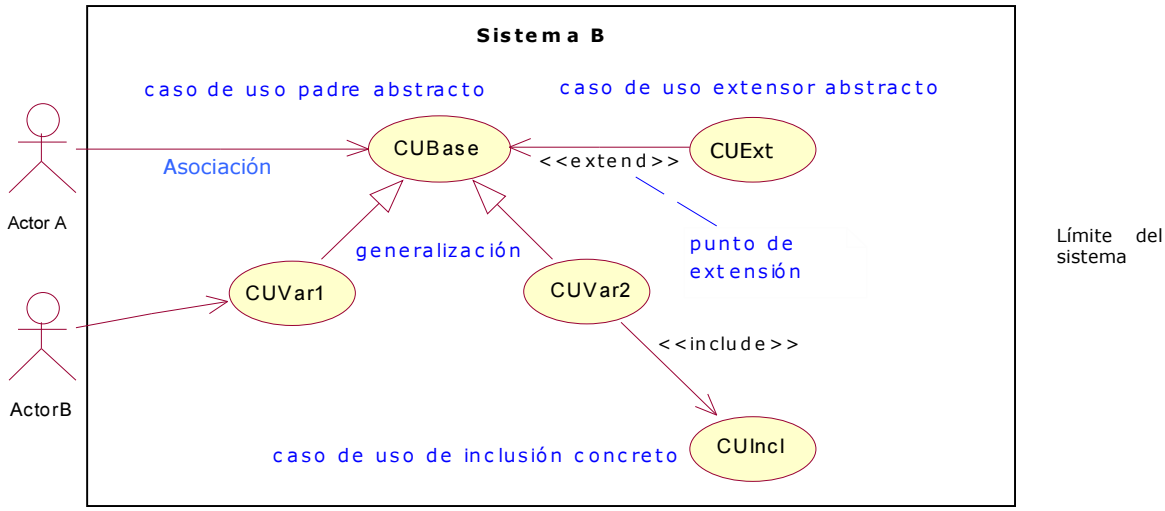

Figura 2.29 Diagramas de Caso de Uso en UML

2.5.2.3 Vista de implantación y vista de despliegue

La vista de implantación muestra el empaquetado físico de las partes reutilizables del sistema en unidades sustituibles, llamadas componentes. Esta vista muestra los elementos del diseño, tales como las clases, mediante componentes, así como sus interfaces y dependencias entre componentes. Los componentes son las piezas reutilizables a partir de las cuales se pueden construir los sistemas.

La vista de despliegue muestra la disposición física de los recursos de ejecución computacional, tales como las computadoras y sus interconexiones, llamados nodos.

2.5.2.3.1 Componente

Un componente es una unidad física de implantación con interfaces bien definidas pensada para ser utilizada como parte reemplazable de un sistema. Cada componente incorpora la implantación de ciertas clases del diseño del sistema.

Los componentes soportan interfaces y en ocasiones éstas se requieren de otros componentes. Una interfaz es una lista de las operaciones que una pieza de software o de hardware ofrece y puede realizar. El uso de las interfaces permite evitar dependencias directas entre componentes, facilitando una sustitución más fácil de nuevos componentes.

Un componente se dibuja como un rectángulo, con dos rectángulos pequeños a un lado. Puede ser unido por medio de líneas sólidas a los círculos que representan sus interfaces (Figura 2.30).

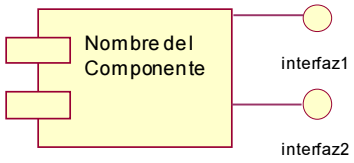

Figura 2.30 Notación de Componente e Interfaces en UML

2.5.2.3.2 Nodo

Un nodo es un objeto físico de ejecución que representa un recurso computacional, teniendo generalmente memoria y capacidad de proceso. Los nodos tienen estereotipos diferentes para reconocer recursos, tales como CPU, dispositivos, memorias etc. Además, los nodos pueden contener objetos e instancias.

Un nodo se representa mediante un cubo con el nombre del nodo y opcionalmente su clasificación. Las asociaciones entre nodos representan líneas de comunicación. Los nodos pueden tener relaciones de generalización para relacionar una descripción general de un nodo, con una variación más específica (Figura 2.31).

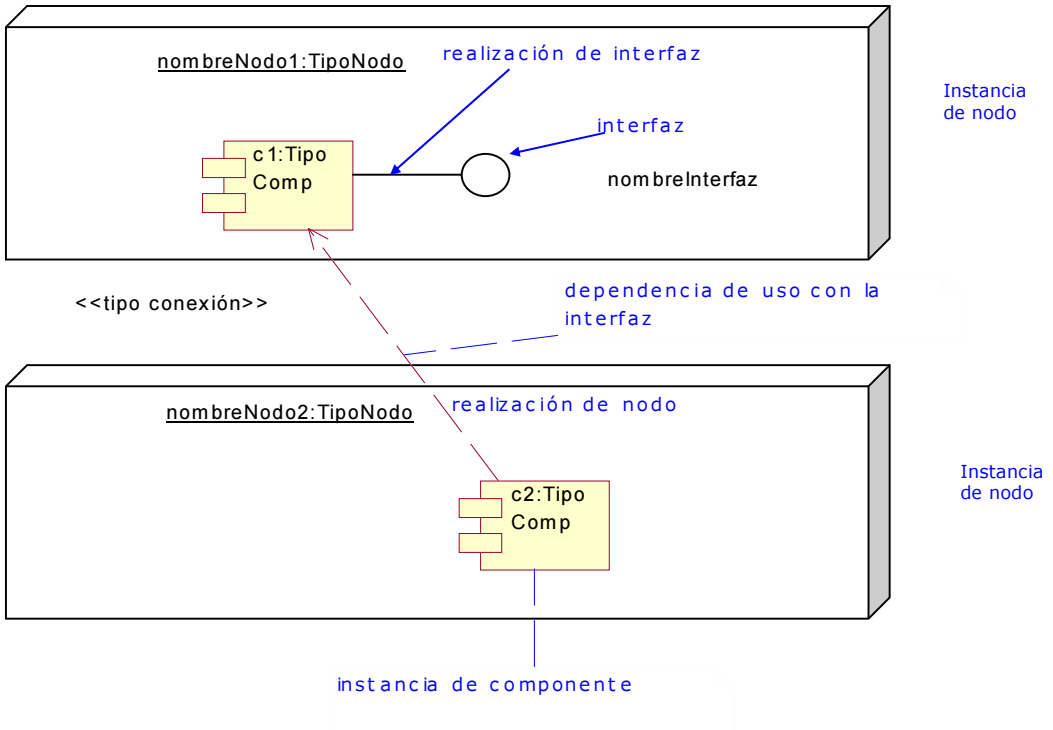

Figura 2.31 Diagramas de Componentes y Nodos en UML

2.5.2.4 Vista de máquina de estados.

Una máquina de estados es un modelo de todas las historias posibles de la vista de un objeto de una clase (Figura 2.34). El objeto se examina aisladamente. Cualquier influencia externa del resto del mundo se resume como evento. Cuando ocurre un evento, se puede desencadenar una transición que lleve el objeto a un nuevo estado. Cuando se dispara una transición, se puede ejecutar una acción a la transición.

Un evento es una ocurrencia significativa que tiene una localización en el tiempo. Ocurre en un punto en el tiempo y no tiene duración. Los eventos se pueden dividir en varios tipos, explícitos e implícitos: eventos de señal, eventos de llamada, eventos de cambio y eventos de tiempo.

En los eventos de señal, la señal es una entidad con nombre, que es como el vehículo de comunicación entre dos objetos. Las señales incorporan la comunicación unidireccional asíncrona, esto es, el remitente no espera que el receptor se ocupe de la señal, sino continúa con su propio trabajo independiente. Para modelar la bidireccionalidad se pueden utilizar varias señales, por lo menos una en cada dirección.

Un evento de llamada es la recepción de una llamada por un objeto que elige poner una operación en ejecución. Por otro lado el evento de cambio es un cambio en el valor de una expresión booleana. Por último, el evento de tiempo representa el paso del tiempo y se puede especificar de modo absoluto (hora) o de modo relativo (tiempo que transcurrió desde un evento dado).

Además, los estados describen un período durante la vida de un objeto de una clase. Un estado puede tener un nombre, aunque a menudo es anónimo y viene descrito simplemente por sus acciones.

En la máquina de estados, un conjunto de estados está conectado mediante transiciones. Las transiciones son procesadas por el estado del que salen. Los estados se muestran como rectángulos con las esquinas redondeadas (Figura 2.32).

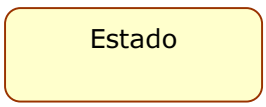

Figura 2.32 Notación de Estado en UML

Una transición es una acción que se ejecuta cuando pasa de un estado a otro. Existen dos tipos de transiciones: externa e interna. La transición externa es una respuesta a un evento que causa un cambio de estado o una transición a sí mismo, junto con la acción especificada. La transición interna es una respuesta a un evento que causa la ejecución de una acción, pero no causa cambio de estado o ejecución de acciones de salida o entrada.

Una acción de entrada, es aquella que se ejecuta cuando se entra en el estado y una acción de salida, es la que se ejecuta cuando sale del estado. La transición se representa por una flecha continua (Figura 2.33).

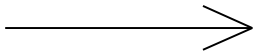

Figura 2.33 Notación de transición en UML

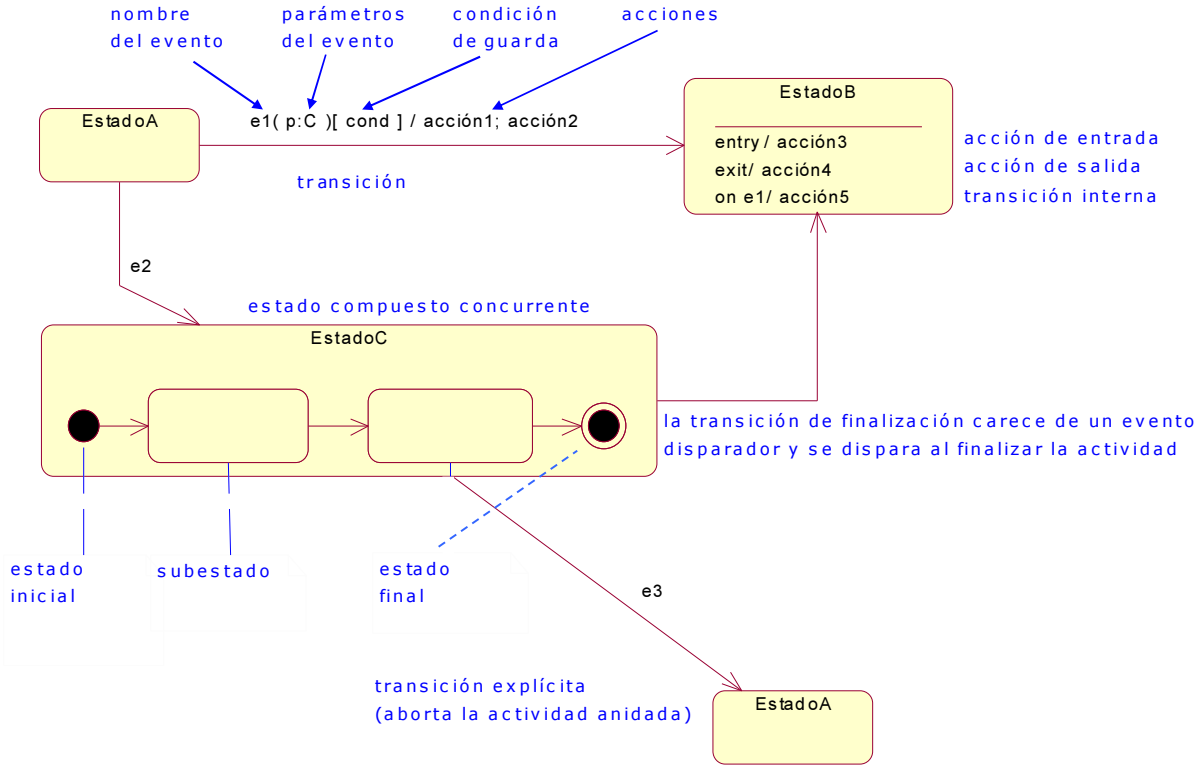

Figura 2.34 Diagrama de Estados en UML

2.5.2.5 Vista de interacción

La vista de interacción describe secuencias de intercambios de mensajes entre objetos para representar el comportamiento de un sistema. Esta visión proporciona una vista integral de su comportamiento.

Una interacción es un conjunto de mensajes dentro de una colaboración. Un mensaje puede ser una señal (una comunicación explícita entre objetos, con nombre y asíncrona) o una llamada (la invocación síncrona de una operación con un mecanismo para el control, que retorna posteriormente al remitente).

Una colaboración es una descripción de una colección de objetos que interactúan para llevar a cabo un cierto comportamiento dentro de un contexto. Detalla una sociedad de objetos cooperantes unidos para realizar un cierto propósito.

Existen dos tipos de diagramas en una vista de interacción: un diagrama de secuencia, que se centra en las secuencias de tiempo de los mensajes y un diagrama de colaboración, que se centra en las relaciones de los objetos que intercambian los mensajes.

2.5.2.5.1 Diagrama de secuencia

Un diagrama de secuencia representa una interacción como un gráfico bidimensional (Figura 2.35). La dimensión vertical es el eje del tiempo, que avanza hacia abajo. La dimensión horizontal muestra los objetos individuales en la colaboración. Durante el tiempo que existe un objeto se muestra por medio de una línea discontinua.

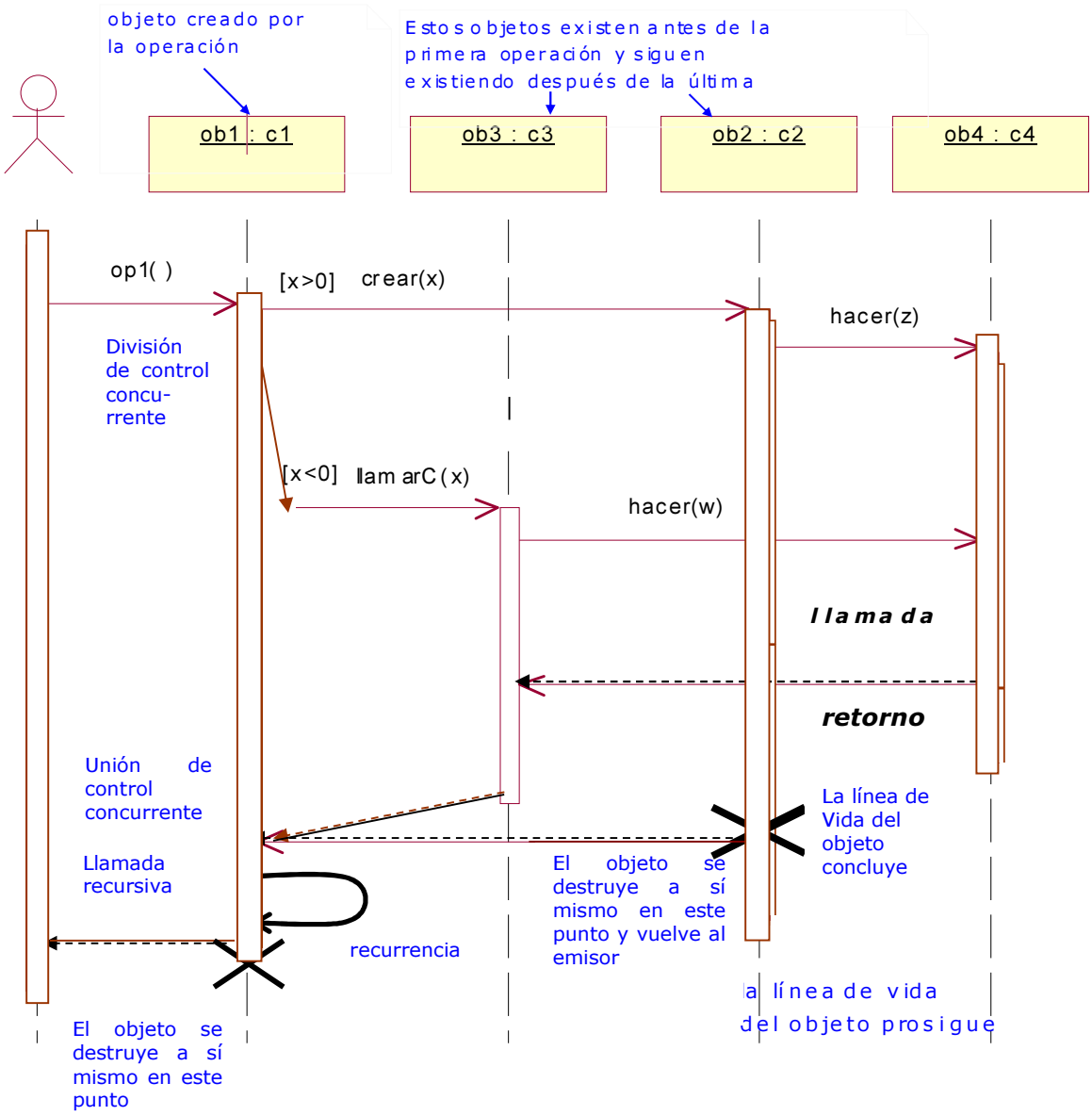

Figura 2.35 Diagrama de Secuencia en UML

Un mensaje se muestra con una flecha, desde la línea de vida de un objeto a la de otro. Las flechas se organizan en el diagrama en orden cronológico hacia abajo.

2.5.2.5.2 Diagrama de colaboración

Un diagrama de colaboración modela los objetos y los enlaces implicados en la interacción (Figura 2.36 y 2.37). Este diagrama muestra eventos (mensajes entre los objetos) y enumera el orden relativo de esos eventos.

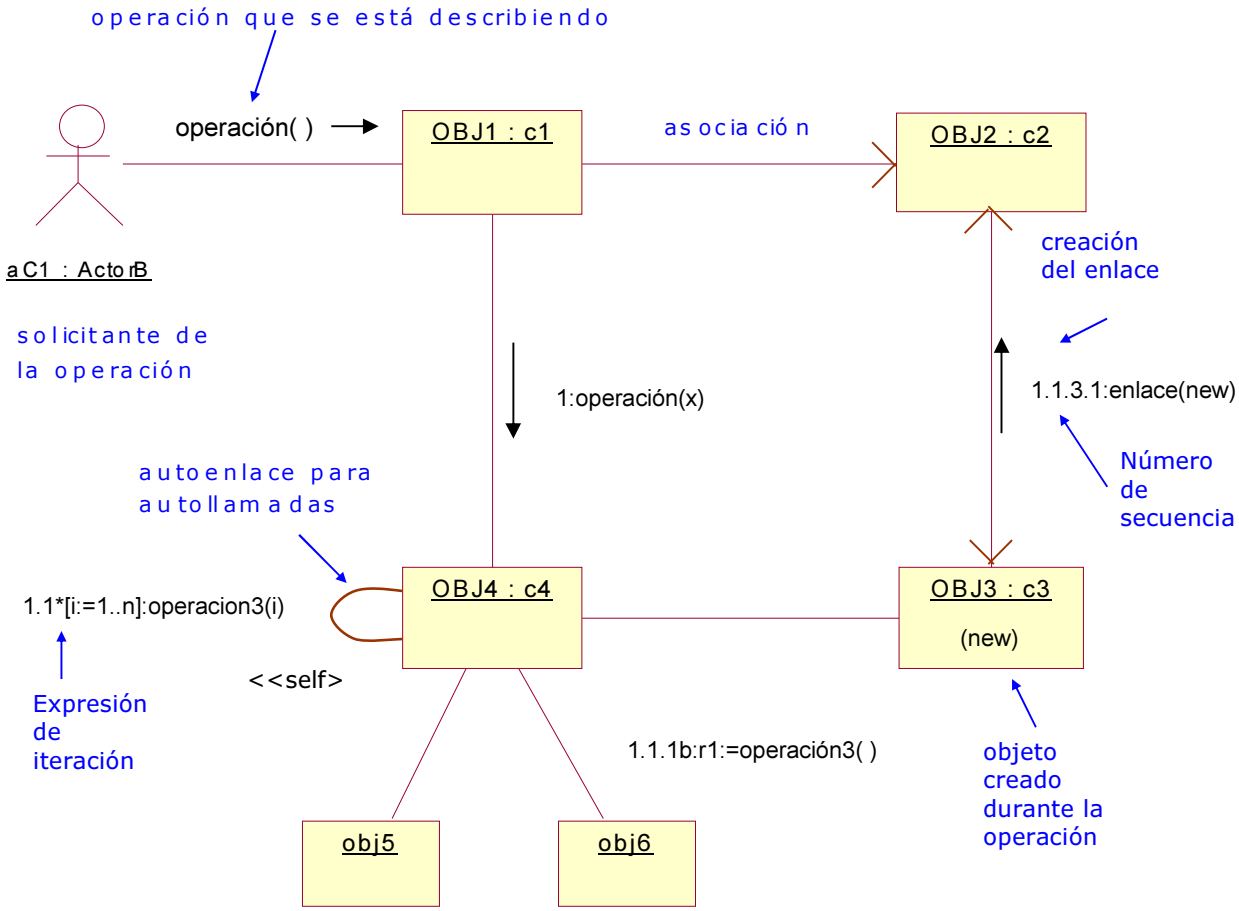

Figura 2.36 Diagrama de Colaboración en UML

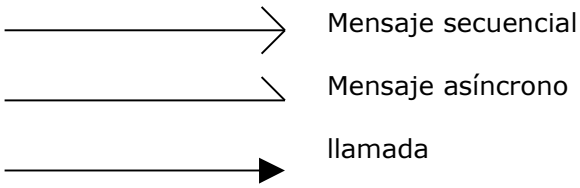

Figura 2.37 Notación de Mensajes en UML

2.5.2.6 Vista de actividades

Una vista de actividades es una variante de una máquina de estados. En esta vista se muestran las actividades implicadas en la ejecución de un cálculo. Su objetivo es mostrar relaciones temporales entre actividades, además de mostrar el flujo de trabajo entre los casos de uso. La vista de actividades se refleja en los diagramas de actividades.

2.5.2.6.1 Diagrama de actividades

Un diagrama de actividades describe grupos de secuencias. Los diagramas de actividades contienen estados de actividades. Éstos se representan como un rectángulo con los extremos redondeados que contiene una descripción de actividad.

Las transiciones simples de terminación se muestran como flechas. Las ramas se muestran como diamantes con múltiples flechas de salida etiquetadas. Una división o unión de control se muestra con múltiples flechas que entran o salen de una barra gruesa de sincronización (Figura 2.38).

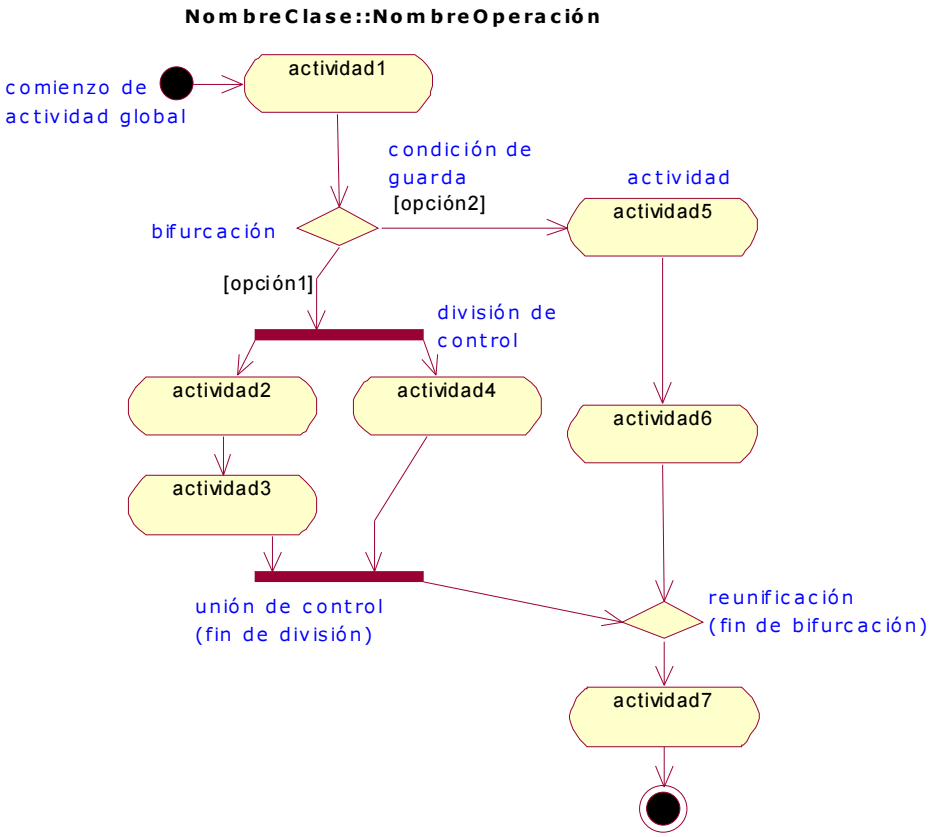

Figura 2.38 Diagrama de Actividades en UML

2.5.2.7 Vista de gestión del modelo

La vista de gestión de modelo, constituye la organización del modelo. Esta vista consiste en mostrar paquetes y relaciones de dependencia entre paquetes. Un conjunto de paquetes contiene elementos del modelo, tales como: clases, máquinas de estados y casos de uso.

Un paquete es una parte del modelo (Figura 2.39). Éstos favorecen al control del contenido de un modelo, así como las unidades para el control de acceso y el control de configuración. Además, cabe señalar que los paquetes pueden contener otros paquetes.

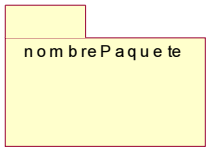

Figura 2.39 Diagrama de Actividades en UML

Las dependencias entre paquetes resumen dependencias entre los elementos internos, esto es, se derivan de las dependencias de los elementos individuales. Éstas se muestran como flechas con líneas discontinuas.

Existe también la relación de generalización y se describe con una flecha hueca del paquete hijo al paquete padre. Por otra parte, un modelo es un paquete que abarca una descripción completa de una vista particular de un sistema, mientras que un subsistema es un paquete que tiene piezas separadas, el cual representa generalmente la partición del sistema (Figura 2.40).

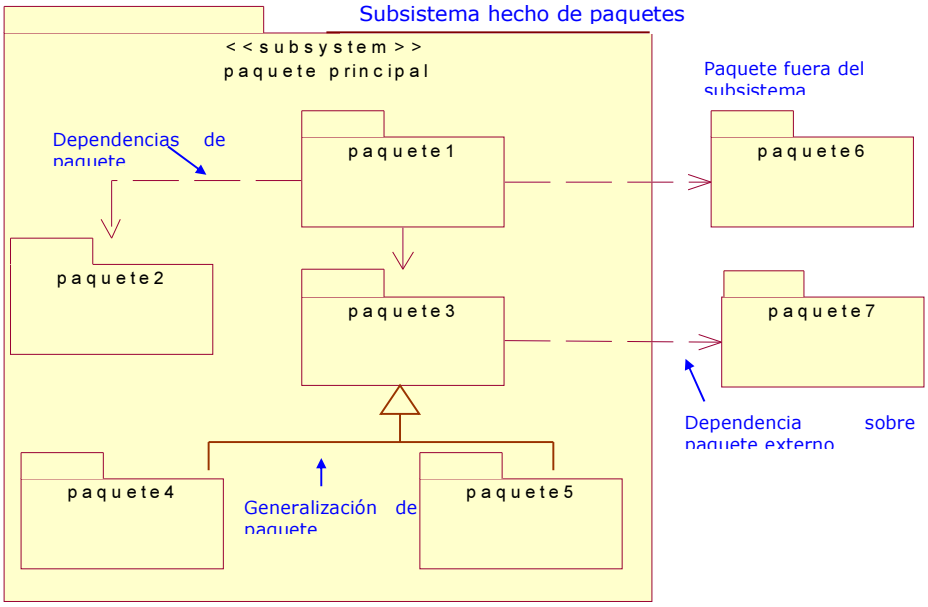

Figura 2.40 Diagrama de Actividades en UML

Las vistas y diagramas explicados para esta última metodología, hacen que sea la más completa y apropiada para proporcionar mejores técnicas y paradigmas en el desarrollo de sistemas, además de hacer más sencillo el modelado del sistema.