CAPÍTULO IV. DISEÑO TÉCNICO CON UML

4.1. Introducción

Los sistemas o aplicaciones, toman forma cuando una o varias personas tienen la visión de cómo la tecnología puede mejorar las cosas. Los desarrolladores deben entender la idea mientras crean el sistema, para ello debe existir un enlace entre quien tiene la idea y el desarrollador.

El UML (Lenguaje unificado de modelado) es una herramienta que cumple con esta función, se basa en símbolos y diagramas que permiten a los creadores generar diseños que capturen la idea de un sistema para comunicárselo de una forma fácil de comprender a quien realice el proceso de desarrollo.

Existen diversas herramientas que permiten la realización de diagramas UML e integrarlos en un modelo de diseño. Los más notables son Select Enterprise, Visual UML y Rational Rose que es la utilizada a lo largo del presente proyecto.

4.2. Necesidad del lenguaje unificado de modelado

En los comienzos de la programación, los desarrolladores no realizaban análisis profundos del problema a resolver. Se evaluaban los requerimientos de los clientes, generando un análisis en algún tipo de notación que ellos mismos comprendieran (aunque el cliente no la comprendiera). El análisis llegaba a los programadores con la esperanza de que el producto final cumpliese con lo que el cliente deseaba.

Dado que la ingeniería de sistemas es una actividad humana, existe la posibilidad de cometer errores en cualquier etapa del proceso, por ejemplo malentendidos por parte del analista sobre los requerimientos del cliente. Estos no eran descubiertos hasta que el producto final era entregado y no cumplía con las funcionalidades requeridas.

Hoy en día es necesario contar con un buen plan de análisis, donde el cliente comprenda el sistema que se va a desarrollar, señalando cambios si no se han entendido claramente sus necesidades. El incremento de complejidad de los sistemas informáticos que cuentan con hardware, software y bases de datos, hace imprescindible organizar el proceso de diseño de tal forma que los analistas, clientes y desarrolladores y otras personas involucradas en el proceso de desarrollo del sistema lo comprendan. UML proporciona esta organización.

Así mismo, la mayoría de los sistemas sufren cambios, mejoras o modificaciones una vez han sido entregadas al cliente. Un proyecto bien diseñado facilitará los cambios, es decir, si el diseño es sólido, un cambio en la implementación se afrontará sin problemas.

La necesidad de diseños sólidos ha traído consigo la creación de una notación que los analistas, desarrolladores y clientes acepten. UML proporciona tal notación.

4.3. Historia del UML

El lenguaje UML aparece en octubre de 1994, cuando James Rumbaugh se unió a la compañía Rational fundada por Grady Booch. El objetivo de ambos era unificar dos métodos que habían desarrollado, el método Booch y el OMT (Object Modelling Tool). Un año después, otro reputado investigador, Ivar Jacobson, se unió a Rational y aportó sus ideas.

El anteproyecto de UML que estas tres personas, conocidas como los "tres amigos", estaban creando, empezó a circular en la industria del software y las reacciones resultantes trajeron consigo considerables modificaciones.

Conforme diversas empresas vieron que UML era útil, se formó un consorcio de UML. Entre los miembros se encuentran DEC, Hewlett-Packard, Intellicorp, Microsoft, Oracle, Texas Instruments y Racional.

En 1997 el consorcio produjo la versión 1.0 de UML como respuesta a la necesidad de un lenguaje de modelado estándar. A finales de este mismo año, apareció la versión 1.1, que como la anterior fue puesta en consideración del OMG (Grupo de administración de objetos). Posteriormente se han realizado nuevas versiones gestionadas por la OMG, hasta la 2.0, y su evolución continúa.

4.4. Tipos de diagramas

UML (lenguaje unificado de modelado) está compuesto de diversos elementos gráficos que se combinan para conformar diagramas. El objetivo de estos diagramas es realizar una simplificación de un sistema presentando diversas perspectivas del mismo, a lo que se conoce como modelo.

El modelo visual creado permite simplificar la complejidad de los sistemas a analizar o diseñar de modo que sea comprendido fácilmente por todas las personas que intervienen en el proceso de desarrollo.

Para poder representar correctamente un sistema, UML ofrece una amplia variedad de diagramas para visualizar el sistema desde varias perspectivas:

4.4.1. Diagrama de Casos de Uso

Un caso de uso es una descripción de las acciones de un sistema desde el punto de vista de un usuario. Se representa la interacción con el sistema a desarrollar donde se muestran los requisitos funcionales (figura 4.1).

Cada caso de uso, que representa el sistema, es una colección de situaciones y cada una de estas una secuencia de pasos. A las entidades que inician las secuencias se las conoce como **actores.**

Figura 4.1. Representación de un caso de uso simple

Hay un actor que inicia un caso de uso y otro (o él mismo) que recibirá la acción que se ha generado. En la representación gráfica, el actor que inicia se encuentra a la derecha del caso de uso, indicado mediante una elipse, y el que recibe, a la derecha. Tanto el nombre del actor como el del caso de uso, deberán mostrarse en el diagrama. Una línea de trazo continuo, conecta el caso de uso con los actores (figura 4.2).

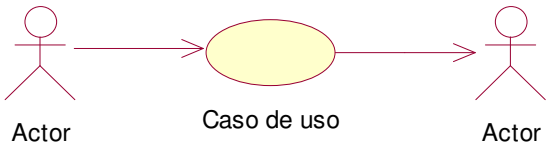


Figura 4.2. Interacción de actor con los casos de uso

Relaciones entre casos de uso:

Los casos de uso se pueden relacionar entre si. Una relación que aparece es la de **inclusión**, en la que un caso de uso utiliza los pasos de otro.
Para representar la inclusión, se utilizará una línea discontinua terminada en punta de flecha en un extremo, y una etiqueta <<include>> (figura 4.3).

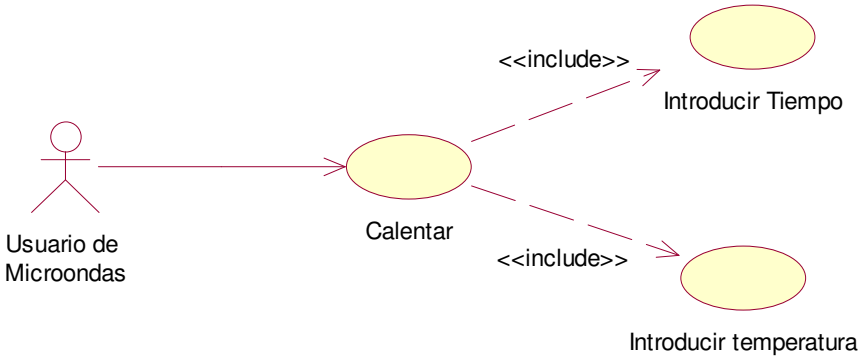


Figura 4.3. Relación de incluye en los casos de uso

Otra relación entre los casos de uso es la **extensión** que permite crear un caso de uso mediante la adición de pasos a uno existente, denominado caso de uso base. La

representación es análoga a la dada en la relación de inclusión, dónde la etiqueta es la <<extends>> (figura 4.4).

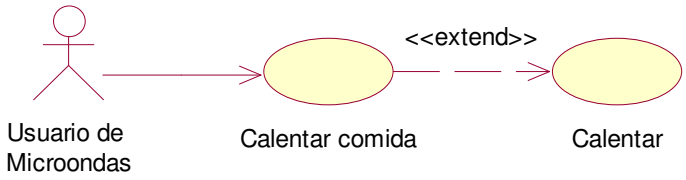


Figura 4.4. Relación de extend en los casos de uso

Si un caso de uso hereda de otro, se dice que tienen una relación de **generalización**. El caso de uso secundario hereda las acciones y significado del primario, y además agrega sus propias acciones. Se representa con una línea continua, que parte del caso de uso secundario, terminado en una punta de flecha sin relleno apuntando al primario (figura 4.5).

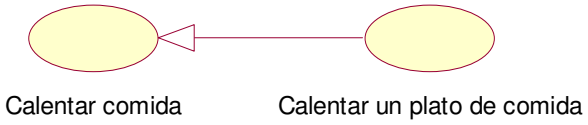


Figura 4.5. Relación de generalización

La relación de generalización también puede darse entre actores (figura 4.6).

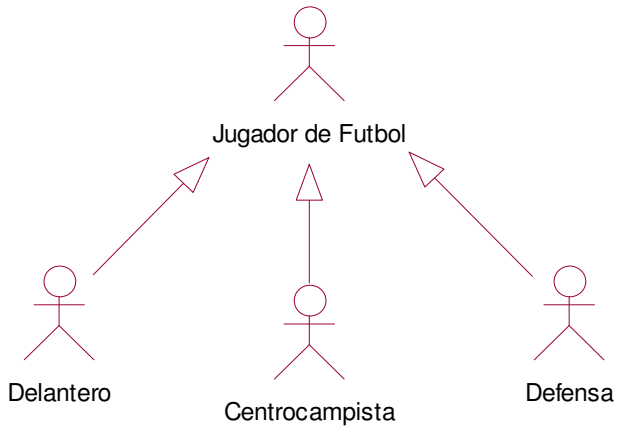


Figura 4.6. Relación de generalización entre actores

Por otro lado, el **agrupamiento** es una manera sencilla de organizar, agrupando en paquetes los casos de uso, cuando estos se relacionan.

4.4.2. Diagrama de clases

Una clase es una categoría o grupo de cosas que tienen atributos (propiedades) y que realizan determinadas acciones.

Un diagrama de clases representa el sistema a través de un conjunto de clases y sus relaciones.

En UML, una **Clase** es representada por un rectángulo que posee tres áreas (figura 4.7).

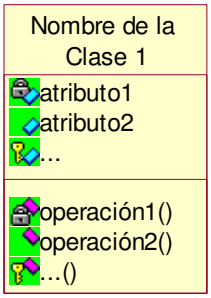


Figura 4.7. Representación de una clase

En donde:

- *Área Superior*: Contiene el **nombre** de la Clase
- *Área Intermedia*: Contiene los **atributos** (o propiedades) que caracterizan a la Clase, pueden ser de tres tipos, según el grado de comunicación y visibilidad de ellos con el entorno:
  - **Public** (): Indica que el atributo será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.
  - **Private** (): Indica que el atributo sólo será accesible desde dentro de la clase (sólo sus métodos lo pueden usar).
  - **Protected** (): Indica que el atributo no será accesible desde fuera de la clase, pero si podrá ser usado por métodos de la clase además de las subclases que se deriven.
- *Área Inferior*: Contiene los **métodos** (u operaciones), los cuales definen la forma de cómo la Clase interactúa con su entorno. Dependiendo de la visibilidad, los métodos pueden ser:
  - **Public** (): Indica que el método será visible tanto dentro como fuera de la clase, es decir, es usado desde todos lados.

- **Private** (): Indica que el método sólo será accesible desde dentro de la clase (sólo otros métodos de la clase lo pueden usar).

- **Protected** (): Indica que el método no será accesible desde fuera de la clase, pero si podrá ser usado por métodos de la clase además de métodos de las subclases que se deriven (ver herencia).

Un diagrama de clases se forma con varios rectángulos de este tipo conectados por líneas que representan la manera en que las clases se relacionan entre si.

Relaciones entre Clases:

Cuando las clases se conectan entre si, esta conexión se conoce como **asociación**. En el diagrama de clases, la asociación se indica con una flecha de una clase a otra (figura 4.8).

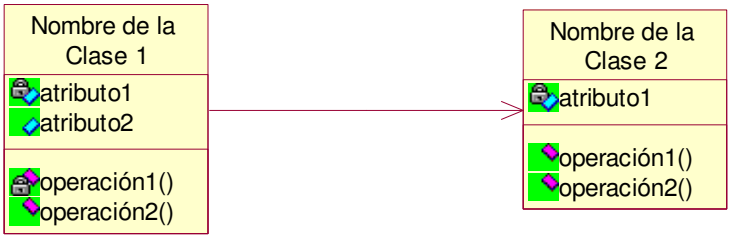


Figura 4.8. Relación de asociación entre clases

Dentro de una relación de asociación, cada clase juega un papel, que se indica en la parte superior de la línea que conecta a dichas clases (figura 4.9).

Figura 4.9. Detalle de la asociación

Una asociación puede funcionar de forma inversa, en el ejemplo, se observa que un trabajador se encuentra empleado en una empresa. Pero también es cierto, que esa empresa, emplea a dicho trabajador (figura 4.10).

Figura 4.10. Asociación en ambos sentidos

Las asociaciones pueden ser más complejas, interviniendo más de una clase (figura 4.11).

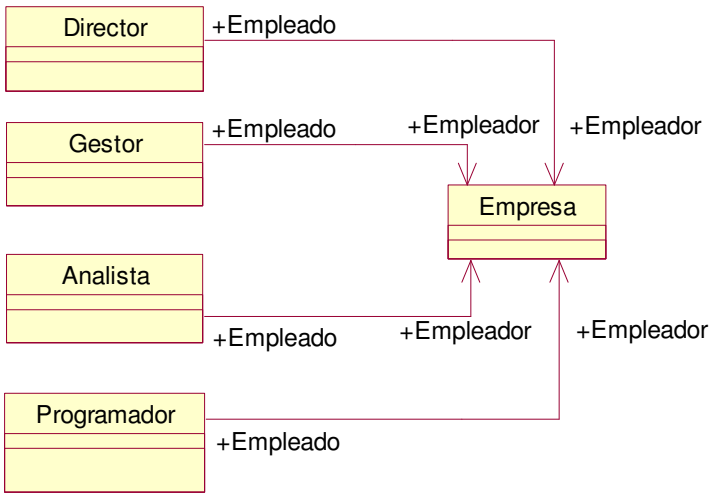


Figura 4.11. Asociación entre varias clases

Una asociación, al igual que una clase, puede contener atributos y operaciones. Cuando este sea el caso, se tendrá una *clase de asociación*. Se conecta a una asociación mediante una línea discontinua, pudiendo asociarse a otra clase (figura 4.12).

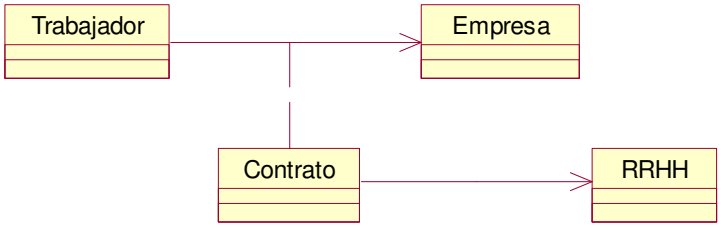


Figura 4.12. Clase de asociación

Así como un objeto es la instancia de una clase, una asociación también cuenta con instancias denominadas *vínculos* y que se representan como una línea que conecta dos objetos.

La asociación trazada entre el trabajador y la empresa de la figura 4.13, sugiere que las dos clases tienen una relación de uno a uno. No obstante, no es lo normal, una empresa cuenta con más trabajadores. Es decir que un trabajador pertenece sólo a una empresa, mientras que una empresa contiene varios trabajadores. Estas especificaciones son ejemplos de multiplicidad, que indica la cantidad de objetos de una clase que se relacionan con un objeto de la clase asociada.

Figura 4.13. Multiplicidad

Se pueden dar varios tipos de multiplicidades:

Uno a uno:

Figura 4.14. Multiplicidad de uno a uno

Uno a muchos:

Figura 4.15. Multiplicidad de uno a muchos

Uno a uno o más:

Figura 4.16. Multiplicidad de uno a uno o más

Uno a ninguno o uno:

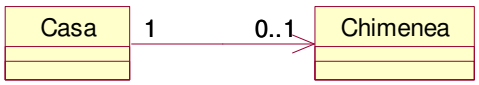


Figura 4.17. Multiplicidad de uno a ninguno o uno

En ocasiones, una clase es una asociación consigo misma. Se denominan *asociaciones reflexivas* y se dan cuando una clase tiene objetos que pueden jugar diversos papeles (figura 4.18). Un ocupanteDeAutomovil puede ser un conductor o un pasajero. En el papel de conductor, un ocupanteDeAutomovil puede llevar ninguno o más ocupanteDeAutomovil.

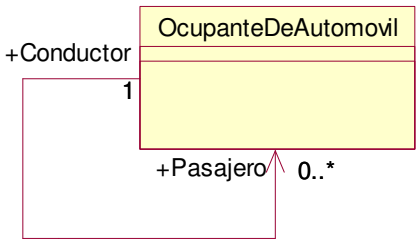


Figura 4.18. Asociación reflexiva entre clases

Una de las características fundamentales de la orientación a objetos es la **herencia** (en UML también denominada **generalización),** indica que una clase (clase secundaria o *subclase*) hereda los atributos y métodos de otra (clase principal o *superclase*). La superclase es más genérica y por tanto, la subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la superclase (public y protected).

En UML, se representa la herencia con una línea que conecta la clase principal con la clase secundaria. En el extremo de la línea próximo a la clase principal, se encuentra una flecha sin relleno, apuntando a dicha clase principal.

En la figura 4.19 se especifica que un Coche y un Camión heredan de Vehículo, es decir, coche posee las Características de Vehículo (precio, velMax, etc.) además posee algo particular que puede ser Descapotable, en cambio Camión también hereda las características de Vehículo (precio, velMax, etc.) pero posee como particularidad propia tara y carga.

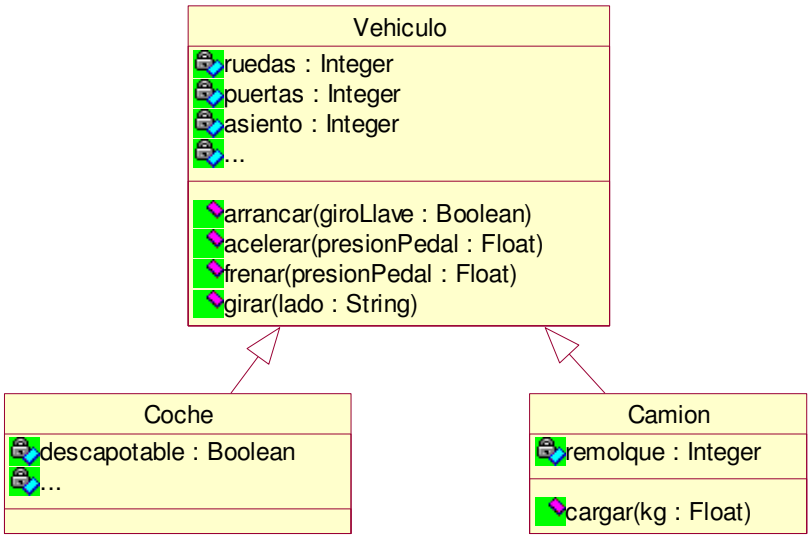


Figura 4.19. Ejemplo de herencia

Una clase puede no provenir de una clase principal, en cuyo caso será una *clase base* o *clase raíz*. Así mismo, una clase podría no tener clases secundarias, denominándose *clase final* o *clase hoja.* Si una clase tiene una única clase principal, se dice que tiene herencia simple, mientras que tendrá herencia múltiple, si proviene de varias clases principales.

En otro tipo de relación, una clase utiliza a otra. A esta relación se le denomina **dependencia.** Se representa por una línea discontinua, que parte de una clase, y termina en flecha sin relleno en el extremo de la clase de la que depende.

Figura 4.20. Relación de dependencia

Existe un tipo especial de relación denominada **agregación** o **acumulación** que se da cuando una clase consta de otras clases.

Véase el ejemplo de la figura 4.21, donde varias clases: PrimerPlato, SegundoPlato y Postre se relacionan con una clase comida como una relación de agregación, es decir, que la clase Comida se compone de las clases mencionadas.

Una relación de agregación se representa por una línea que parte de los componentes y finaliza con un rombo sin relleno en el extremo más cercano al todo.

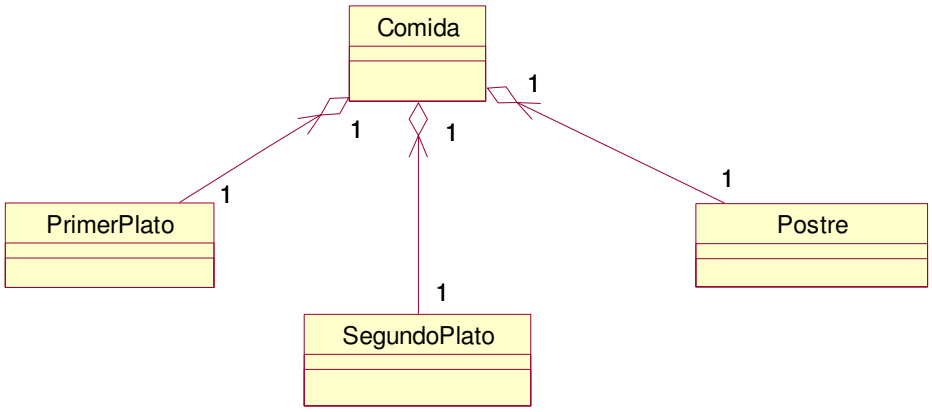


Figura 4.21. Relación de agregación

Una **composición** es un tipo muy representativo de una agregación. En dicha relación, cada componente pertenece solamente a un todo (figura 4.22).

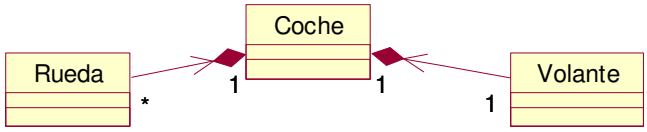


Figura 4.22. Relación de composición

Una **interfaz** es un conjunto de operaciones que especifica cierto aspecto de la funcionalidad de una clase, y es un conjunto de operaciones que una clase presenta a otras.

Un interfaz no tiene atributos, sus métodos son abstractos, es decir, que no se implementan dentro de la interfaz, sino dentro de la clase que utiliza dicha operación. La utilidad de esto es que varias clases pueden utilizar este método creado por una clase e introducida en el interfaz.

Se representa como una clase pero indicando la palabra <<interfaz>> antes del nombre. En los diagramas UML, de la clase que utiliza el interfaz parte una línea discontinua acabada en flecha sin relleno, en el extremo más próximo al interfaz (figura 4.23).

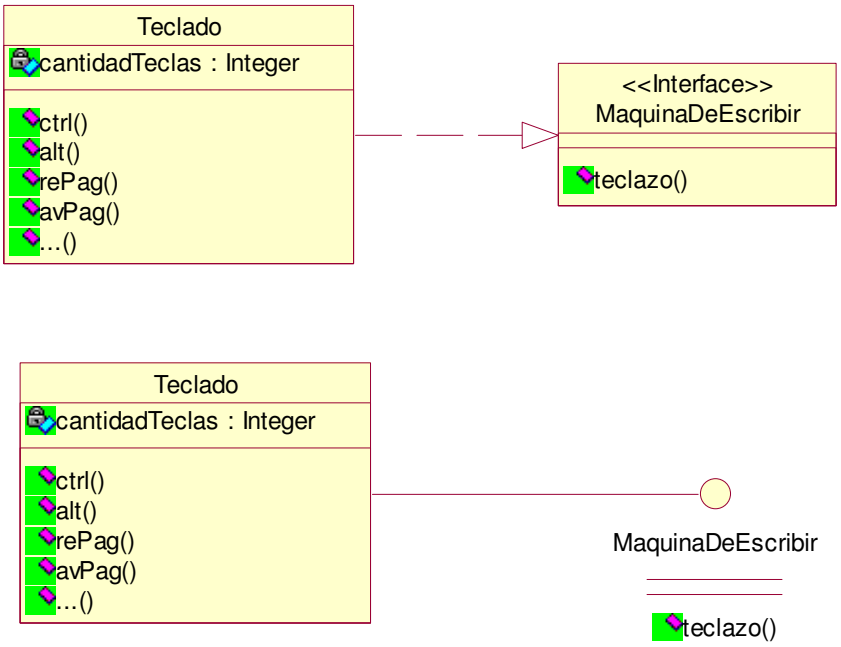


Figura 4.23. Interfaz

Destacar que una clase puede realizar más de un interfaz, y éste puede ser realizado por más de una clase.

Dentro del lenguaje unificado de modelado, y con el objetivo de aclarar aquellos puntos que el diseñador considera oportunos, aparecen las *notas*, que cuelgan de las clases y que pueden relacionar diagramas. Es posible asociar a una nota otro diagrama, de modo que se simplifica la compresión de diagramas de gran tamaño, que se dividen en varios y se relacionan mediante notas (figura 4.24).

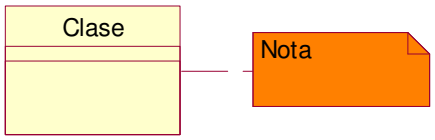


Figura 4.24. Notas

4.4.3. Diagrama de objetos

Un objeto es una instancia de la clase, dicho de otra manera, una entidad con valores específicos de sus atributos y operaciones. (Todos los seres humanos son instancias de una clase persona).

En UML, los objetos de representan mediante un rectángulo indicando el nombre del objeto seguido de dos puntos (:) y el nombre de la clase de la que se instancia.

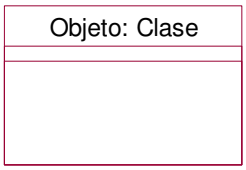


Figura 4.25. Representación de objetos en UML

4.4.4. Diagrama de estados

Un objeto, en un momento en particular, se encuentra en un estado definido, al igual que una persona es recién nacida, infante, adolescente, joven o adulta.
Un diagrama de estados representa los estados en que puede encontrarse un objeto, junto con las transiciones entre los estados.

Es necesario contar con diagramas de estados, ya que permiten a los analistas, diseñadores y programadores comprender el comportamiento de los objetos del sistema. Mientras que los diagramas de clases y de objetos muestran los aspectos estáticos, los diagramas de estados representan el comportamiento dinámico del sistema.

Fundamentalmente los desarrolladores deben saber como los objetos se comportarán, debido a que serán ellos quienes tendrán que reflejar tales comportamientos en el software.

Un diagrama de estados simple se representa en la figura 4.26. El punto relleno representa el estado inicial, de todos los que presentará el objeto. De este parte una flecha que indica transición hacia otro simbolizado mediante un rectángulo de esquinas redondeadas. Para finalizar se encuentra una diana que representa el estado final del objeto.

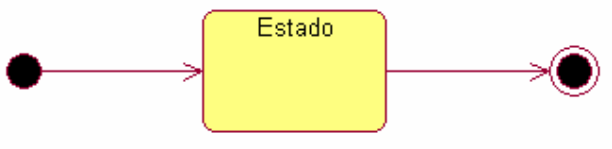


Figura 4.26. Diagrama de Estados

En UML al icono de estado (rectángulo redondeado), es posible agregarle más información con el objetivo de tener un diagrama más completo. Se puede dividir el símbolo en tres áreas, (nombre, atributos y operaciones). El área superior mostrará el nombre del estado, el área central las variables de estado, mientras que el área inferior contendrá las actividades u operaciones.

Las operaciones que actúan sobre el estado, deben servir de aclaración sobre la transición que sufre el objeto, por lo que se pueden dividir en tres fundamentales (figura 4.27): *entry* (que sucede cuando el sistema entra al estado), *do* (que ocurre cuando el sistema está en el estado), *event* (acción que aparece al darse un determinado evento dentro del estado) y *exit* (que sucede cuando el sistema sale del estado).

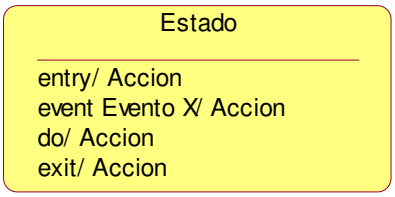


Figura 4.27. Representación de un estado en UML

También es posible agregar detalles a las líneas de transición, tales como un *suceso* que provoque una transición, y la *acción* que se ejecuta y haga que se lleve a cabo la modificación del estado. Los sucesos y acciones se representarán junto a la línea que indica la transición y separados de una barra diagonal (figura 4.28).

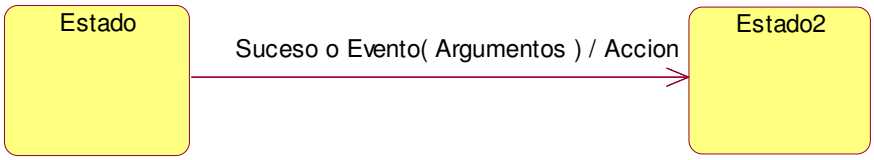


Figura 4.28. Detalle de la línea de transición de estados

En ocasiones un evento causará una transición sin acción asociada y algunas veces una transición se dará sin ser causada por un suceso sino porque finalizará una actividad. A estos tipos de relaciones se les conoce como *transición no desencadenada*.

En muchos sistemas, es necesario que se produzcan transiciones pasado un intervalo de tiempo, a estás se les suele llamar *condición de seguridad*. Véase el ejemplo de un protector de pantalla en la figura 4.29, pasado un tiempo sin pulsar una techa o accionar el ratón, la pantalla se desactiva mediante la transición al estado protector de pantalla

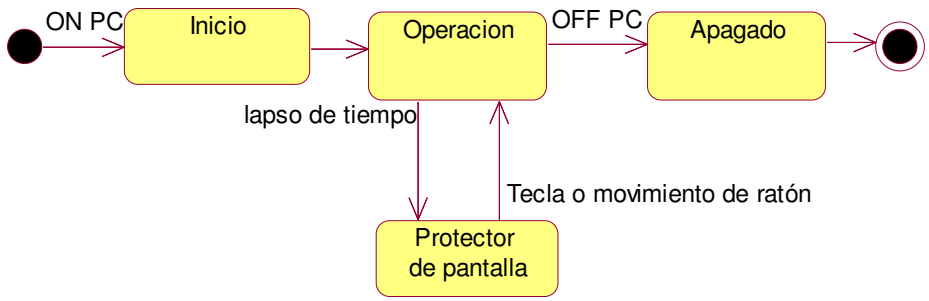


Figura 4.29. Ejemplo de condición de seguridad

En el ejemplo anterior, el sistema deberá analizar si no se ha presionado ninguna tecla o no se ha accionado el ratón. Luego, deberá registrar tales acciones y reaccionar si se han producido. El sistema atravesará varios cambios de estado mientras se encuentre en el estado de operación. Dado que estos estados se encuentran dentro de otros, se conocerán como *subestados*. Existen dos tipos: *secuencial* y *concurrente*.

Los *subastados secuenciales* como su nombre indican ocurren uno detrás del otro. Los subastados mencionados anteriormente tendrían la secuencia que aparece en la figura 4.30.

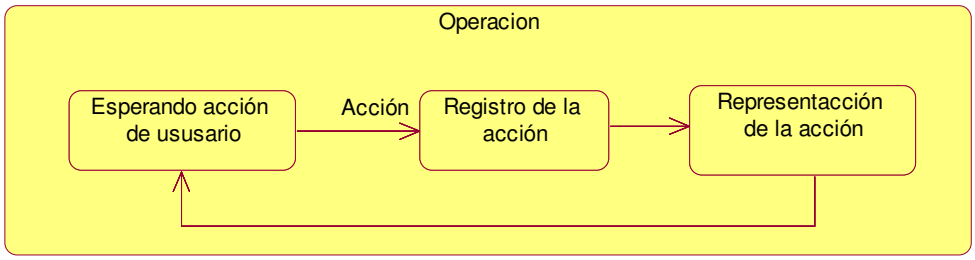


Figura 4.30. Subastados secuenciales

Los *subastados concurrentes* suceden al mismo tiempo. Se representan dentro del estado que los contiene, separados por una línea discontinua. En el ejemplo sería el control del cronómetro del sistema que dispara el protector de pantalla sino se dieron ninguno de los subastados anteriores (figura 4.31).

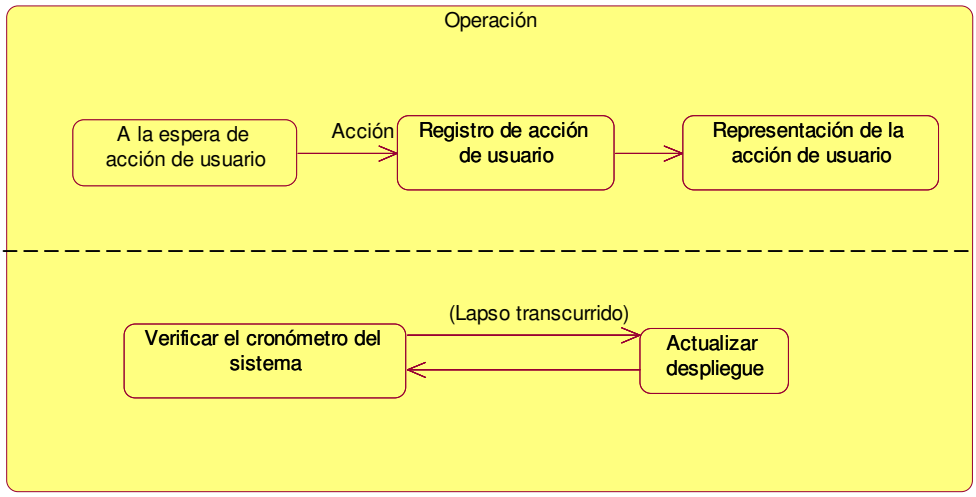


Figura 4.31. Subastados concurrentes

4.4.5. Diagrama de secuencia

Los diagramas de objetos y de clases, muestran el sistema desde un punto de vista estático, sin embargo en un sistema funcional los objetos interactúan entre si a lo largo del tiempo.

Los diagramas de estados se refieren a las transiciones que sufre un objeto, mientras que en UML los diagramas de secuencias muestra la forma en que un objeto interactúa con otros en base a los tiempos.

En los diagramas de secuencia, los **objetos** se representan con rectángulos con nombre subrayado. Se sitúan en la parte superior del diagrama, de izquierda a derecha. Debajo del rectángulo, de forma descendente, parte una línea discontinua que representa *la línea de vida* del objeto. Junto a la línea de vida se encuentra un rectángulo en posición vertical, que representa la ejecución de una operación por parte del objeto, conocido como *activación*. La longitud del rectángulo indica la duración de la activación del objeto (figura 4.32).

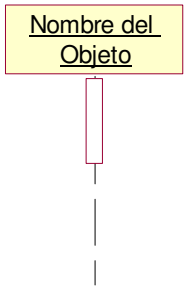


Figura 4.32. Representación de objetos en UML

Los **mensajes** parten de un objeto a otro, o a él mismo, indicando que objeto entra en ejecución en el instante de tiempo. Pueden ser simples, sincrónico, o asincrónico.

Un mensaje simple es la transferencia de control de un objeto a otro. Si el mensaje es sincrónico, el objeto que lo envía esperará respuesta antes de continuar con su ejecución. Si, por el contrario, el objeto envía un mensaje asincrónico, no esperará respuesta antes de continuar.

En el diagrama de secuencias, se representan por una línea terminada en punta de flecha que parte de la línea de vida de un objeto apuntando a la de otro objeto (o la suya propia).

El diagrama representa el **tiempo** en dirección vertical, comenzando en la parte superior y avanzando hacia la parte inferior. Por tanto, un mensaje que esté más cerca de la parte superior ocurrirá antes que otro que esté cerca de la parte inferior.

Así, el diagrama de secuencias tiene dos dimensiones, la vertical que muestra el paso del tiempo, y la horizontal que indica la disposición de los objetos.

Figura 4.33: los objetos, dentro del diagrama de secuencias, se colocan de izquierda a derecha en la parte superior. Cada línea de vida de un objeto es una línea discontinua que se desplaza hacia abajo del objeto. Una línea continua terminada en punta de flecha, que representa un mensaje entre objetos, conecta una línea de vida con otra. El tiempo se inicia en la parte superior y continúa hacia abajo.

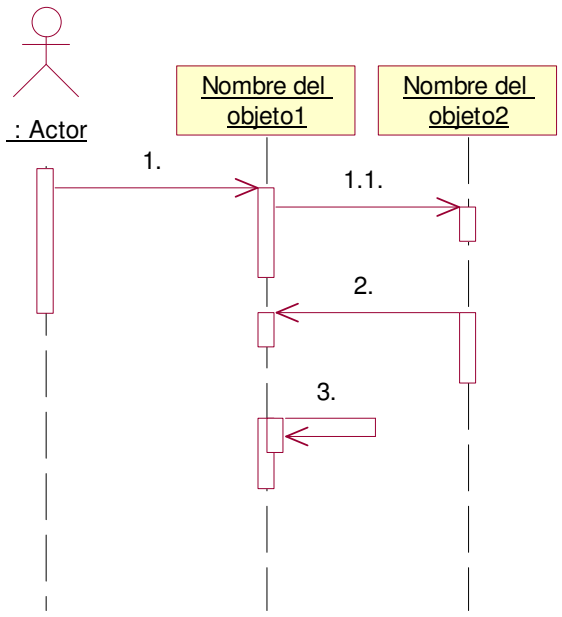


Figura 4.33. Diagrama de secuencia

En ocasiones un objeto cuenta con una operación que se invoca a sí misma. A esto se le conoce como **recursividad**. En UML se representa mediante una flecha que parte de la activación, indicando la operación, que apunta a un rectángulo pequeño sobrepuesto en la activación (figura 4.34).

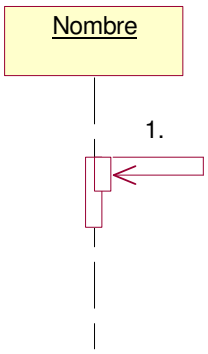


Figura 4.34. Recursividad

4.4.6. Diagrama de colaboraciones

Los diagramas de colaboraciones muestran la forma en que los objetos colaboran entre si, tal y como sucede en el diagrama de secuencias. Ambos diagramas son semánticamente equivalentes, esto significa que proporcionan la misma información y se podría convertir uno en el otro.

Los diagramas de secuencias destacan la sucesión de las interacciones, constituyendo una organización con respecto al tiempo. Los diagramas de colaboraciones destacan el contexto y la organización general de los objetos que interactúan., organizándose de acuerdo al espacio. Esta diferencia que presentan, hace útil contar con ambas representaciones del sistema a la hora de afrontar un proceso de desarrollo.

Un diagrama de colaboraciones (figura 4.35) muestra los objetos como tales y sus relaciones entre sí. Así mismo, indica los mensajes que se envían los objetos.

Los mensajes se representan con una flecha cerca de la línea que asocia los objetos, esta flecha apunta al objeto receptor de la operación. El mensaje se etiqueta, indicando al objeto receptor la operación que debe realizar. El nombre de la operación se representa seguido de paréntesis en donde aparecerán los parámetros (en caso de haber alguno) con los que funcionará el método.

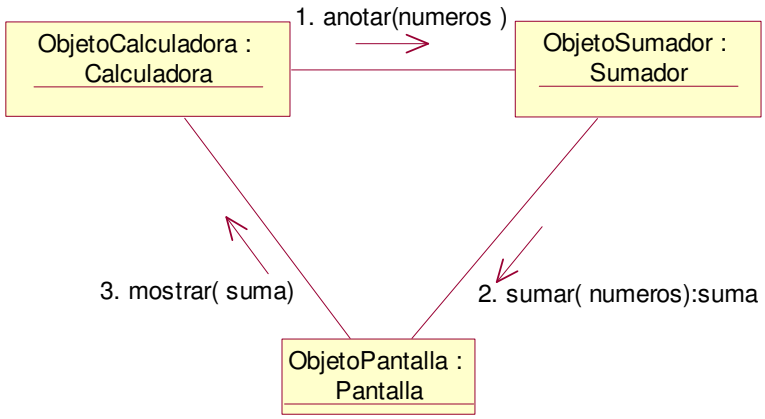


Figura 4.35. Diagrama de colaboraciones

4.4.7. Diagrama de actividades

Uno de los primeros métodos visuales aplicados a la programación son los diagrama de flujo, muestra una secuencia de pasos, procesos, puntos de decisión y bifurcaciones. Estos pretenden facilitar la comprensión del sistema, entendiendo todos y cada uno de los pasos y caminos que se pueden dar. Éste diagrama es convertido posteriormente en código.

El diagrama de actividades de UML, es muy parecido a los diagramas de flujo, muestra los pasos (conocidos como actividades) en una operación o proceso, así como los puntos de decisión y bifurcaciones.

Volviendo un momento atrás, se puede ver que los diagramas de estados muestran los estados de un objeto y las actividades representadas mediante flechas entre los estados. El diagrama de actividades extiende el diagrama de estados, refinando lo que ocurre dentro de cada actividad.

Cada actividad se representa mediante un rectángulo de esquinas redondeadas (más alargado y ovalado que la representación de un estado). La transición a otra actividad viene indicada con una flecha. Un punto inicial (círculo relleno) y un punto final (diana), completan los símbolos utilizados en el diagrama de actividades (figura 4.36).

Dentro de una secuencia de actividades, frecuentemente se llegará a un punto donde se realizará alguna decisión. Un **punto de decisión** se representa mediante un rombo que indica decisión. Entre corchetes se indicará la condición junto a la ruta correspondiente.

Los **eventos** indican el suceso que ocurre en la transición de una actividad a otra. En el diagrama, se indican encima de las conexiones entre actividades (Buen tiempo y Mal tiempo en el ejemplo de la figura 4.37).

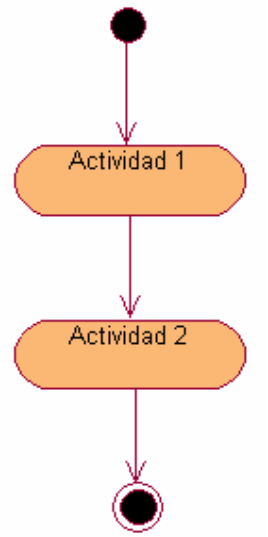


Figura 4.36. Diagrama de actividades

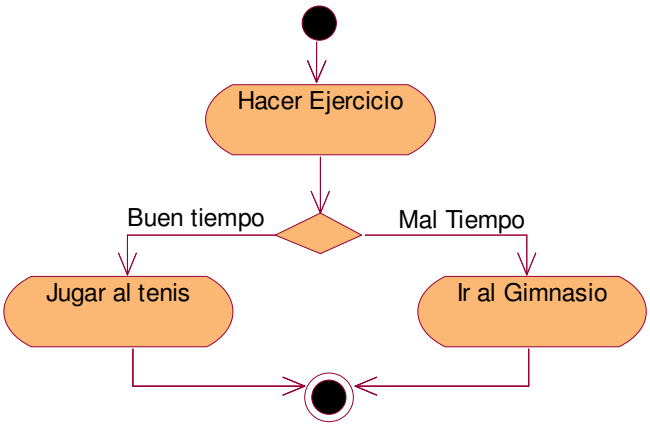


Figura 4.37. Punto de decisión

En ocasiones, dos actividades se realizan al mismo tiempo y se resumen en el mismo punto, denominándose **concurrentes.** Se representan mediante una línea gruesa perpendicular a la transición de donde parten ambas rutas, que apuntan a una nueva línea gruesa en donde finalizan (figura 4.38).

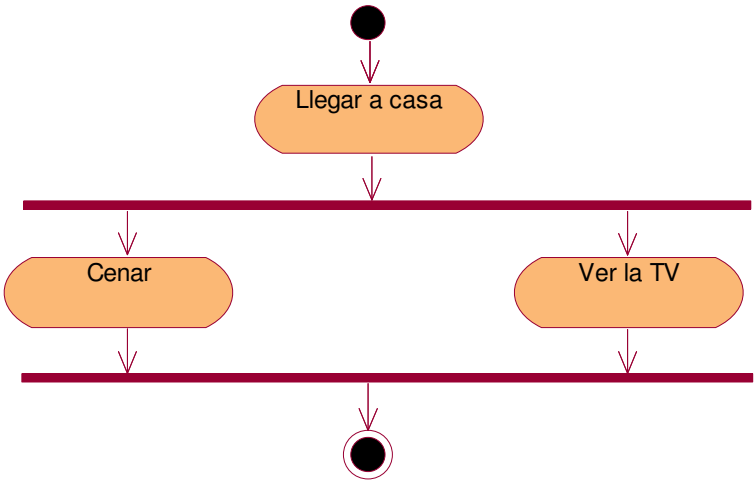


Figura 4.38. Actividades concurrentes

4.4.8. Diagramas de componentes

Los diagramas UML anteriores, representan el sistema desde el punto de vista conceptual, es decir, la abstracción de elementos. Existe un diagrama que representa a una entidad real, un componente de software.
Se define un componente de software como una parte física de un sistema, y se encuentra en la computadora, no en la mente del analista.

Un componente sería, por ejemplo, la representación dentro del software (código) de una clase. La clase se trata de una definición de atributos y operaciones, mientras que el componente es la realización física de estos. Un componente puede estar formado por varias clases.

Uno de los puntos fundamentales de la programación orientada a objetos es la reutilización. Los componentes se crean con el objetivo de poder ser reutilizados en diversos sistemas, de modo que se agilice la realización de nuevo software.

Componentes e interfaces

Se define interfaz como un conjunto de operaciones de una clase que son utilizadas por otras. De igual modo, un componente (implementación de software de una clase) utilizará un interfaz, y sólo a través de éste se podrán ejecutar sus operaciones.
La relación entre un componente y su interfaz se conoce como *realización*.

Un componente podrá hacer disponible su interfaz (*interfaz de exportación*) para que otros componentes puedan utilizar las operaciones que contiene. Dicho de otra manera, un componente puede acceder a los servicios de otro componente (*interfaz de importación*).

Sustitución y reutilización

Las interfaces son fundamentales dentro del modelado, ya que son primordiales en la reutilización y sustitución.

Un componente se puede sustituir por otro, si el nuevo contiene las mismas interfaces que el antiguo. Esto es posible debido a que los demás componentes no notan el cambio, pues se comunican entre si a través de su interfaz.

La reutilización de un componente en otro sistema es posible, si éste puede acceder al componente reutilizado mediante sus interfaces. Es por tanto, muy importante un diseño orientado a la reutilización, de modo que pequeños cambios en sus interfaces hagan reutilizables los componentes, con el considerable ahorro de tiempo para el desarrollador (nuevo código que debe ser integrado y probado).

Tipos de componentes

1. *Componentes de distribución*, que conforman el fundamento de los sistemas ejecutables (dll, ejecutables, controles Active X, Java Beans, etc.).
2. *Componentes para trabajar en el producto* (archivos de Base de datos y de código)
3. *Componentes de ejecución*, creados como consecuencia de un sistema de ejecución.

Representación en UML

Los diagramas de componentes UML contienen, obviamente, componentes, interfaces y sus relaciones.

Un **componente** se representa mediante un rectángulo que tiene otros dos sobrepuestos en su lado izquierdo (figura 4.39).

Figura 4.39. Representación de un componente

Se puede agregar al símbolo detalles sobre el componente (figura 4.40), como el nombre del paquete al que pertenece, las clases que contiene, así como el tipo de componente (Applet, Active X, etc.).

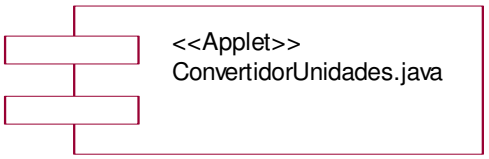


Figura 4.40. Detalle de un componente

Las **interfaces** se representan con rectángulos conectados al componente mediante línea continua (figura 4.41).

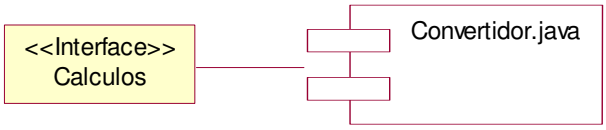


Figura 4.41. Representación de una interfaz

Otra representación de la interfaz es un círculo (figura 4.42).

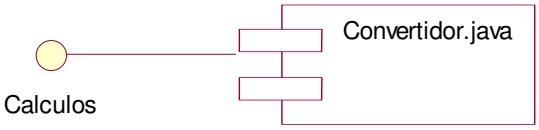


Figura 4.42. Representación de interfaz mediante su icono

Bajo ambas representaciones, se puede incluir la relación con otros componentes, a través de la interfaz de importación (figura 4.43).

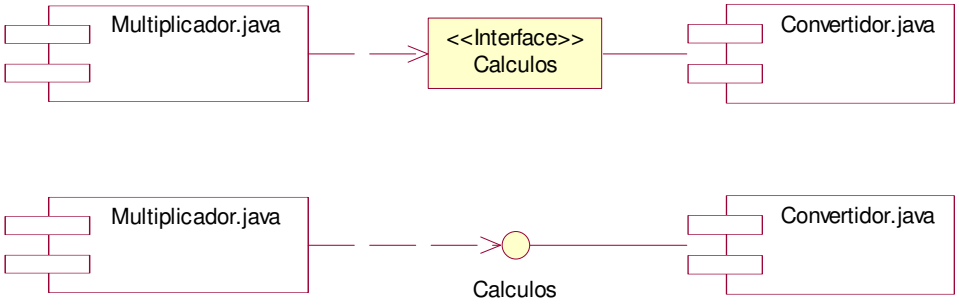


Figura 4.43. Relación de interfaces con más de un componente

4.4.9. Diagramas de distribución

Todo sistema está compuesto de hardware, siendo básico para un buen diseño del sistema, un diseño sólido de distribución del hardware.

A los elementos de hardware se les denomina de forma genérica **nodos**. Dentro del hardware, se pueden usar dos tipos de nodos:

- Un procesador, el cual puede ejecutar un componente.

- Un dispositivo (impresora, monitor, etc.), que no ejecuta componentes, pero que tiene la función de interacción con el mundo exterior.

En UML, un nodo viene representado por un cubo su nombre (figura 4.44).

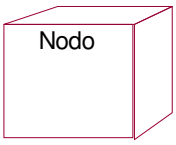


Figura 4.44. Representación en UML de un nodo

Así mismo, puede contener más información, tal como el nombre del paquete al que pertenece, componentes situados en dicho nodo, etc.

La relación entre los elementos de hardware o nodos, se representa mediante una línea que une los cubos en donde se puede utilizar el estereotipo de conexión que se da (figura 4.45).

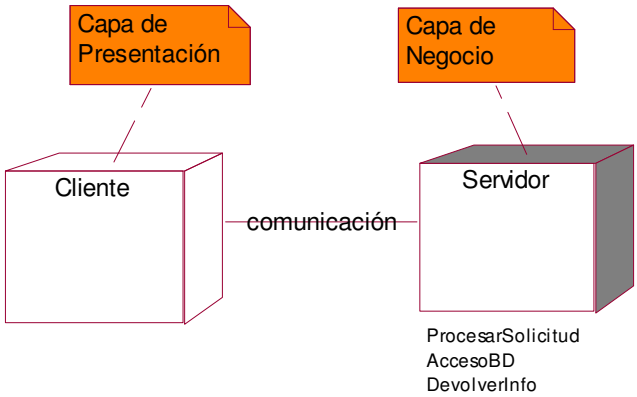


Figura 4.45. Diagrama de distribución

4.5. Diseño del convertidor de unidades

Los diagramas UML sirven al desarrollador a tener una visión global de la aplicación que se pretende desarrollar. No siempre son necesarios todos los diagramas, bastaría con aquellos necesarios para que el programador entienda lo que se pretende desarrollar y cómo se ha de hacer. Así mismo sirve para documentar la aplicación y así facilitar futuras actualizaciones.

Cómo ejemplo de un diseño básico, se ha incluido un diagrama de caso de uso de la aplicación que aquí se presenta, y un diagrama de clases del convertidor de unidades.

- *Diagrama de casos de uso*

Mediante el diagrama de casos de uso de la figura 4.46, se muestran aquellas acciones que el usuario, de la aplicación web desarrollada, puede hacer. De modo que se da una visión general de la funcionalidad de la aplicación.

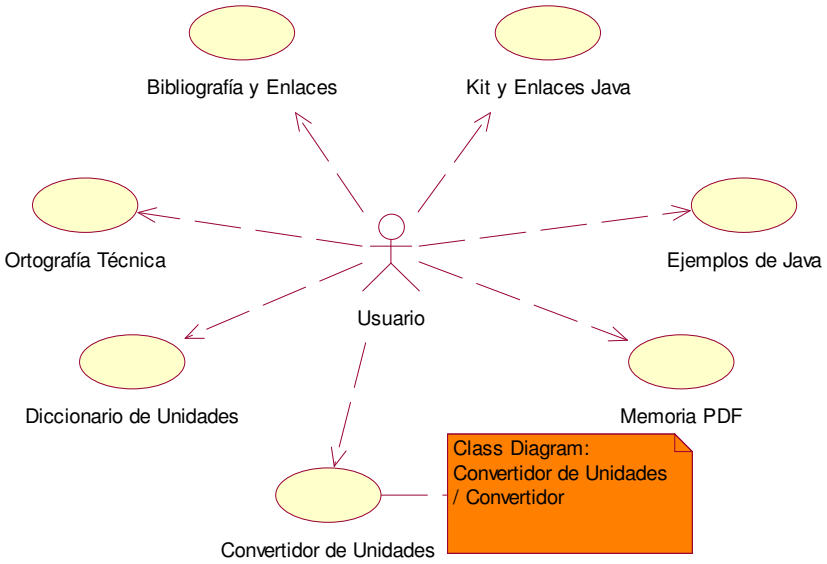


Figura 4.46. Caso de uso de la aplicación

El usuario puede acceder, según el diagrama, a un *Diccionario de Unidades*, a consultar las normas de *Ortografía técnica*, al *Kit y Enlaces Java* para conocer más de este lenguaje de programación, a *Ejemplos de Java* y la *Bibliografía y Enlaces*. Así mismo podrá utilizar la herramienta Java, *Convertidor de Unidades.*

➢ *Diagrama de Clases*

El diagrama anterior contiene una nota, donde se enlaza al diagrama de clases del Convertidor de unidades, de la figura 4.47.

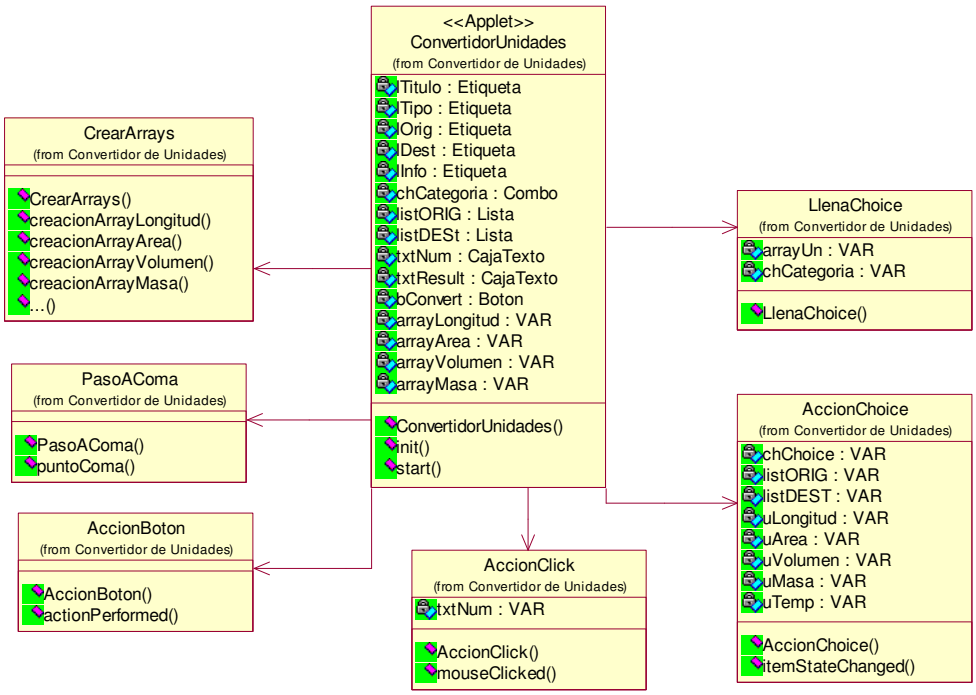


Figura 4.47. Diagrama de clases del convertidor

La clase **ConvertidorUnidades**, es un Applet de java y constituye la base del convertidor de unidades, denominándose clase principal . Esta clase es la encargada de la representación gráfica y la interacción con el usuario. Contiene atributos, de distintos tipos, y métodos que se describen a continuación:

ATRIBUTOS

- *Etiqueta*, serán los distintos nombres que aparecen en la pantalla.
- *Combo*, desplegable en el que aparecen distintas opciones a elegir por el usuario.
- *Lista*, área de texto que contiene diversas opciones todas ellas visibles desde el inicio, al contrario que en el caso anterior
- *Caja de texto*, espacio reservado para mostrar texto desde la aplicación o introducir texto por parte del usuario.
- *VAR*, variables internas de la aplicación.

MÉTODOS

- *Init(),* es el método que inicia el applet.
- *Start()*, es llamado en cada reload del applet.

Dentro de la clase ConvertidorUnidades se utilizan otras clases, que no contienen representación gráfica, es decir que únicamente contienen variables y métodos y por tanto realizan algún procedimiento interno de la aplicación.

La clase **CrearArrays**, se encarga de almacenar los factores de conversión de las unidades de medida. Contiene para ello varios métodos:

MÉTODOS

- *creacionArrayLongitud()*, método que llena el array de factores de conversión de unidades de longitud .
- *creacionArrayArea()*, llena el array de factores de área.
- *creacionArray…()*, diversos métodos para llenar los factores de conversión de las distintas unidades de medida.

La clase **LlenaChoice**, llena el desplegable de la clase principal, que se muestra por pantalla, con las unidades de medida. Contiene variables internas y el método que realiza dicha función:

ATRIBUTOS

- *VAR*, variables internas de la aplicación.

MÉTODOS

- *LlenaChoice()*, es el mismo método constructor de la clase el que llena el desplegable con las unidades de medida.

La clase **PasoAComa**, se encarga de pasar el resultado de punto a coma o de coma a punto, dependiendo de si el usuario trabaja con punto como separador de la parte decimal o por el contrario lo hace con coma. El método que lleva a cabo esta función es:

MÉTODOS

- *puntoComa()*, método que pasa de punto a coma y viceversa.

La clase **AccionClick**, lleva a cabo las funciones de control de la caja de texto

ATRIBUTOS

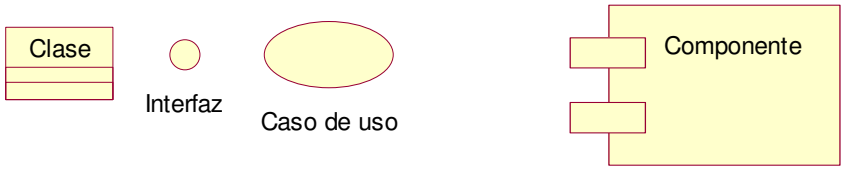
- *VAR*, variables internas de la aplicación.

MÉTODOS

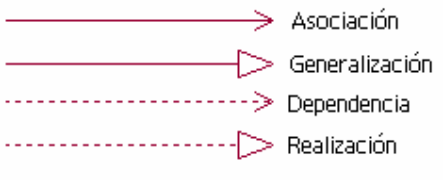
- *LlenaChoice()*, es el mismo método constructor de la clase el que llena el desplegable con las unidades de medida.

La figura 4.48, muestra las representaciones UML utilizadas a lo largo del presente capítulo.

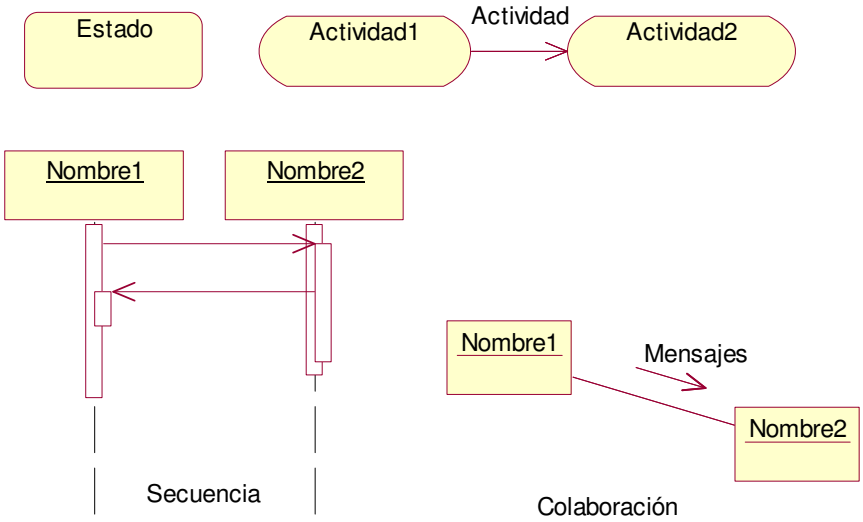
Elementos Estructurales



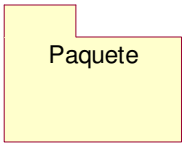
Relaciones



Elementos de comportamiento



Agrupaciones



Anotación

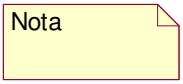


Figura 4.48. Representaciones UML