UNIVERSIDAD DE ORIENTE

NÚCLEO DE ANZOÁTEGUI

ESCUELA DE INGENIERIA Y CIENCIAS APLICADAS

DEPARTAMENTO DE COMPUTACIÓN Y SYSTEMAS

DESARROLLO DE SOFTWARE



**LENGUAJE UNIFICADO DE MODELADO(UML).**

**ELEMENTOS Y DIAGRAMAS.**

Presentado por:

Christian Yánez C.I. 18.099.810

Aldo Olivares C.I. 19.854.994

Dixon Marrufo C.I. 20.358.837

Cesar Roldan C.I. 21.231.177

**Barcelona, Mayo de 2015**

Índice

[Introducción 4](#_Toc418627577)

[1. Bloque Básico del UML 5](#_Toc418627578)

[2. Elementos del UML 7](#_Toc418627579)

[3. Elementos de Comportamiento 8](#_Toc418627580)

[3.1. Interacción 8](#_Toc418627581)

[3.2. Máquina de Estados 8](#_Toc418627582)

[3.3. Actividad 9](#_Toc418627583)

[4. Elementos de Agrupación 9](#_Toc418627584)

[4.1. Paquete 9](#_Toc418627585)

[5. Notación del lenguaje de modelado unificado (UML) 10](#_Toc418627586)

[6. UML ofrece una notación y semántica estándar 10](#_Toc418627587)

[7. Elementos Notacionales 10](#_Toc418627588)

[8. Elementos comunes entre diagramas. 11](#_Toc418627589)

[9. Dependencias 12](#_Toc418627590)

[10. Diagrama de Estructura estática 12](#_Toc418627591)

[10.1. Clase 12](#_Toc418627592)

[10.2. Objetos 13](#_Toc418627593)

[10.3. Asociaciones 13](#_Toc418627594)

[a) Asociación binaria 14](#_Toc418627595)

[b) Sentido 14](#_Toc418627596)

[c) Multiplicidad 15](#_Toc418627597)

[d) Roles 15](#_Toc418627598)

[e) Composición 15](#_Toc418627599)

[10.4. Generalización 16](#_Toc418627600)

[10.5. Clase paramétrica 16](#_Toc418627601)

[10.6. Paquete 17](#_Toc418627602)

[10.7. Dependencia 17](#_Toc418627603)

[10.8. Agregación 18](#_Toc418627604)

[a) Agregación por contenido físico o por valor 18](#_Toc418627605)

[b) Agregación conceptual o por referencia 19](#_Toc418627606)

[10.9. Elementos Derivados 19](#_Toc418627607)

[11. Diagrama de caso de uso 20](#_Toc418627608)

[11.1. Actor 20](#_Toc418627609)

[11.2. Relaciones en un diagrama de caso de uso 20](#_Toc418627610)

[11.3. Caso de uso 21](#_Toc418627611)

[11.4. Límite de Sistema: 21](#_Toc418627612)

[12. Diagrama de secuencia 21](#_Toc418627613)

[12.1. Línea de vida de un objeto 22](#_Toc418627614)

[12.2. Activación 22](#_Toc418627615)

[12.3. Mensaje 23](#_Toc418627616)

[13. Diagrama de Colaboración 23](#_Toc418627617)

[13.1. Objeto 23](#_Toc418627618)

[13.2. Enlaces 23](#_Toc418627619)

[13.3. Flujo de mensajes 24](#_Toc418627620)

[13.4. Objeto compuesto 24](#_Toc418627621)

[14. Diagrama de Estados 24](#_Toc418627622)

[14.1. Estado 25](#_Toc418627623)

[14.2. Eventos 25](#_Toc418627624)

[14.3. Envío de mensajes 26](#_Toc418627625)

[14.4. Transición simple 26](#_Toc418627626)

[14.5. Transición interna 27](#_Toc418627627)

[14.6. Diagrama de Actividades 27](#_Toc418627628)

[14.7. Estado de acción 28](#_Toc418627629)

[14.8. Transiciones 28](#_Toc418627630)

[14.9. Decisiones 29](#_Toc418627631)

[15. Diagrama de Implementación 29](#_Toc418627632)

[16. Diagrama de Componentes 29](#_Toc418627633)

[17. Diagrama de Ejecución 30](#_Toc418627634)

[17.1. Nodos 30](#_Toc418627635)

[17.2. Componentes 30](#_Toc418627636)

[Conclusión 32](#_Toc418627637)

Introducción

El lenguaje unificado de diagrama o notación (UML) sirve para especificar, visualizar y documentar esquemas de sistemas de software orientado a objetos. UML no es un método de desarrollo, lo que significa que no sirve para determinar qué hacer en primer lugar o cómo diseñar el sistema, sino que simplemente le ayuda a visualizar el diseño y a hacerlo más accesible para otros. UML está controlado por el grupo de administración de objetos (OMG) y es el estándar de descripción de esquemas de software.

UML está diseñado para su uso con software orientado a objetos, y tiene un uso limitado en otro tipo de cuestiones de programación.

UML se compone de muchos elementos de esquematización que representan las diferentes partes de un sistema de software. Los elementos UML se utilizan para crear diagramas, que representa alguna parte o punto de vista del sistema. En este trabajo haremos una introducción a los de estos diagramas más utilizados.

1. Bloque Básico del UML

La construcción y diseño de Software se ha simplificado al pasar de los tiempos, y los lenguajes de modelos de diseño han ayudado a eso.

El UML, es un lenguaje unificado de modelado que se encarga de simplificar la perspectiva de un sistema en forma de diagrama para su posterior estudio. La perspectiva viene dada por la realidad que conforma el sistema, más aun el producto final será una representación de lo que “supuestamente” ocurre en dicho sistema.

El UML es un lenguaje tanto como Pascal, C# (C sharp), el alemán, el inglés y el latín; el UML fue inventado alrededor de 1997.

Como sucede con otros lenguajes, el UML fue inventado por necesidad.

Es más como con muchos lenguajes, en el UML se usan símbolos para transmitir significado. Sin embargo, a diferencia de los lenguajes orgánicos, como el inglés y el alemán, que evolucionan con el transcurso del tiempo a partir del uso común y la adaptación, el UML fue inventado por científicos, lo cual, por desgracia, es un problema. Los científicos son muy inteligentes pero con frecuencia no son muy buenos para explicar las cosas a aquellos menos científicos.

El uml representa una formalización del análisis y el diseño, y la formalización siempre parece llegar tarde. Considere los fabricantes de automóviles del siglo pasado. Al principio del siglo pasado, todos los fabricantes de coches en Flint, Michigan, estaban convirtiendo los carruajes ligeros tirados por un solo caballo en automóviles. Esto ocurrió mucho antes de que las grandes universidades, como la Michigan State University (msu), graduaran ingenieros mecánicos capacitados para construir automóviles y herramientas de software, como programas para diseño con ayuda de computadora (cad) que son especialmente buenos en el dibujo de artículos complejos, como las partes de los automóviles. La evolución de la ingeniería formalizada de los automóviles es consecuente con la evolución de la ingeniería formalizada del software.

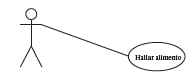
Hace alrededor de 5000 años, los chinos crearon una de las primeras computadoras: el ábaco. Hace cerca de 150 años, Charles Babbage inventó una máquina mecánica de cálculo. En 1940, Alan Turing definió la máquina Turing de cálculo, y Presper Eckert y John Mauchly inventaron la Eniac. Después de las máquinas de cálculo, vinieron las tarjetas perforadas y el análisis y diseño estructurados de Grace Hopper para apoyar el desarrollo de Cobol. En la década de 1960, se inventó Smalltalk, un lenguaje orientado a objetos, y en 1986, Bjarne Stroustrop inventó lo que ahora se conoce como C++. No fue sino hasta alrededor de este mismo periodo —la década de 1980— cuando hombres muy inteligentes, como Ivar Jacobson, James Rumbaugh y Grady Booch, empezaron a definir los elementos del análisis y diseño modernos de software, lo que ahora llamamos el uml.

A finales de la década de 1980 y principios de la de 1990, las guerras sobre la notación del modelado estaban plenamente entabladas, con diferentes facciones apoyando a Jacobson, Rumbaugh o Booch. Recuerde, no fue sino hasta 1980 cuando la persona promedio pudo comprar y poseer una computadora personal (pc), y hacer algo útil con ella. Jacobson, Rumbaugh y Booch, cada uno por su lado, usaron símbolos y reglas diferentes para crear sus modelos. Finalmente, Rumbaugh y Booch empezaron a colaborar

en relación con los elementos de sus respectivos lenguajes de modelado, y Jacobson se les unió en Rational Software. A mediados de la década de 1990, se fusionaron los elementos de modelado de Rumbaugh [Object Modeling Technique (omt, técnica de modelado de objetos)], Booch (método de Booch) y Jacobson (Objectory and Use Cases, cajas objetos y de usos) —a Rumbaugh, Jacobson y Booch se les mencionaba como “los tres amigos”— para formar el *proceso unificado de modelado*. Poco tiempo después, se eliminó *proceso* de la especificación del modelado y nació el uml. Esto ocurrió hace muy poco tiempo, apenas en 1997. La especificación uml 2.0 se estabilizó en octubre de 2004. Es correcto, ahora sólo estamos en la versión 2.

En la práctica se puede apreciar que es un lenguaje, y como todo lenguaje tiene reglas. Por decir un ejemplo, la utilización de relaciones de inclusión en un sistema puede conllevar a la descomposición del mismo, por lo que se aconseja no usarla en exceso.

Al hablar de un bloque básico nos viene a la mente un modelo estándar usado en diversas áreas para definir cualquier tipo de sistema. Pero la realidad es otra, no existe dicho modelo, pero existe una forma de simplificar la correcta obtención de la representación de un sistema. Los Elementos, Relaciones y Diagramas son los tres pilares fundamentales para la representación de la construcción de un Bloque Básico en UML.



Bloque básico de construcción de UML (Elementos (Actor y alimento), Relación(Linea), Diagrama(Casos de uso)

El UML es una definición oficial de un lenguaje pictórico con símbolos y relaciones comunes que tienen un significado común. Si todos los participantes hablan UML, entonces las imágenes tienen el mismo significado para todos aquellos que las observen. Por lo tanto, aprender UML es esencial para ser capaz de usar imágenes para experimentar barata, flexible y rápidamente con las soluciones. Es importante reiterar aquí que es más rápido, más barato y más fácil resolver problemas con imágenes que con código. La única barrera para obtener beneficios del modelado es aprender el lenguaje del mismo.

El UML es un lenguaje precisamente como lo son el inglés o el afrikaans. El UML comprende símbolos y una gramática que define la manera en que se pueden usar estos símbolos. Aprenda los símbolos y la gramática, y sus imágenes serán comprensibles para todo aquel que reconozca estos símbolos y conozca la gramática. Aunque, ¿por qué el UML? Usted podría usar cualesquiera símbolos y reglas con el fin de crear su propio lenguaje de modelado, pero el truco estaría en hacer que otros también lo usaran. Si sus aspiraciones son inventar un mejor lenguaje de modelado, entonces no me corresponde detenerlo. Debe saber que el UML se considera un estándar y que lo que este lenguaje es o no es lo define un consorcio de empresas que constituyen el Object Management Group (omg, Grupo de Administración de Objetos).

1. Elementos del UML

Hay 4 tipos de elementos en UML, elementos estructurales, elementos de comportamiento, elementos de agrupación y elementos de anotación:

|  |  |  |
| --- | --- | --- |
| **Interfaz** | Es una colección de operaciones que especifican un servicio de una clase o componente, representa el comportamiento completo de una clase o componente. | Interfaz |
| **Colaboración** | Se definen como una interacción y es una sociedad de roles y otros elementos que colaboran para proporcionar un comportamiento cooperativo mayor que la suma de los comportamientos de sus elementos las colaboración representan la implementación de patrones que forman un sistema. | Colaboración |
| **Caso de Uso** | Es una descripción de secuencias de acciones que un sistema ejecuta y que produce un resultado observable de interés para un actor particular. Se utiliza para estructurar el comportamiento en un modelo. | Caso de Uso |
| **Clase Activa** | Es un objeto que tienen uno o más procesos o hilos de ejecución y por lo tanto pueden dar origen a actividades de control.  Son iguales a las clases excepto en que sus objetos representan a elementos cuyo comportamiento es concurrente con otros elementos. | |  | | --- | | Clase 1 | |  | |  | |
| **Componente** | Es una parte física y reemplazable de un sistema que conforma con un conjunto de interfaces y proporciona la implementación de dicho conjunto.  Representa típicamente el empaquetamiento físico de diferentes elementos lógicos, como clases interfaces y colaboraciones. | Componente |
| **Nodo** | Es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional que dispone de memoria y con frecuencia capacidad de procesamiento. | **Nodo 1.**  Nodo |

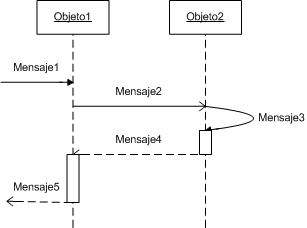
Estos seis elementos (clases, interfaces, colaboraciones, casos de uso, clases activas, componentes y nodos) son los elementos estructurales básicos que se pueden incluir en el modelo UML. También existen variaciones de estos siete elementos, tales como actores, señales, procesos, hilos y aplicaciones, documentos, archivos, bibliotecas, páginas y tablas.

1. Elementos de Comportamiento

Los elementos de comportamiento son las partes dinámicas de los modelos UML. Estos son los verbos de un modelo, y representan comportamientos en el tiempo y el espacio. Suelen estar conectados semánticamente a elementos estructurales. En total **hay tres tipos principales de elementos de comportamiento que se pueden incluir en el modelo UML.**

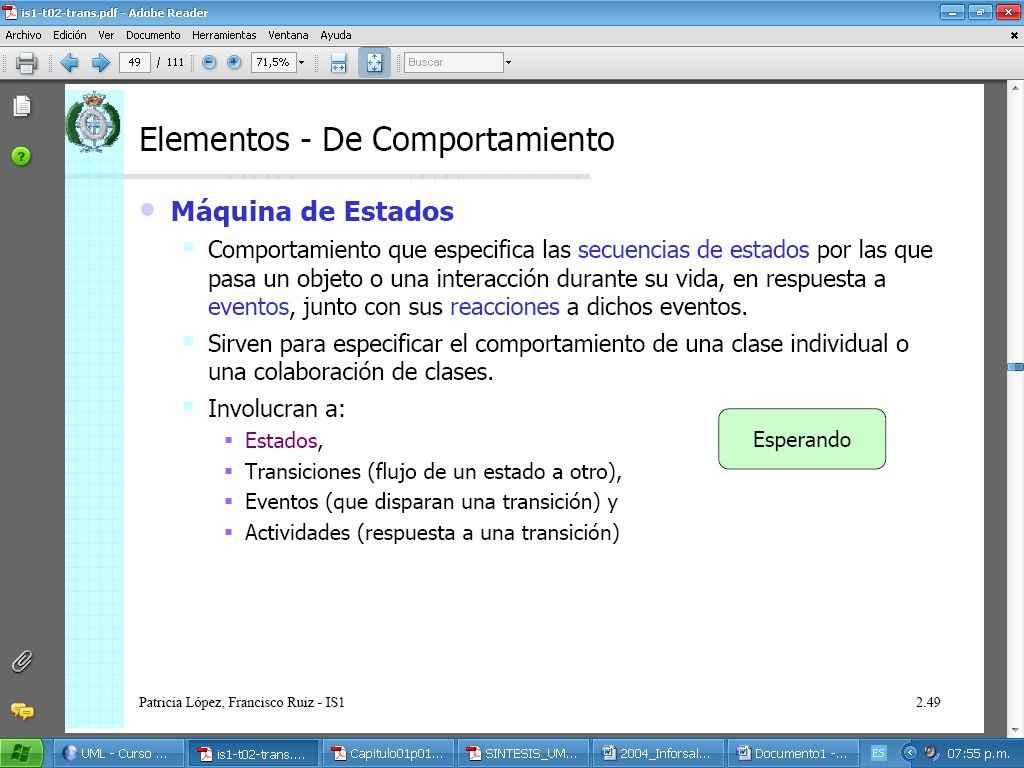
* 1. Interacción

Es un comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos, dentro de un contexto en particular, para alcanzar un propósito específico. Una interacción involucra a muchos otros elementos como mensajes, acciones y enlaces. Sirven para modelar el comportamiento de una sociedad de objetos, o una operación individual. Se representa con una línea dirigida.

****

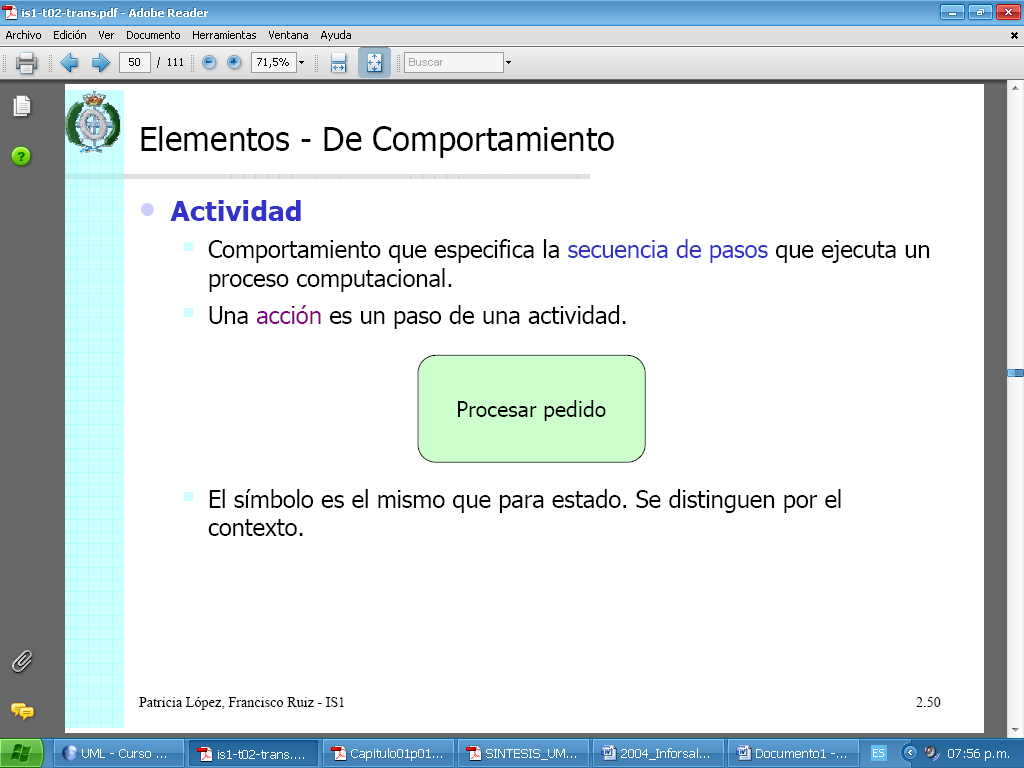
* 1. Máquina de Estados

Es un comportamiento que específica las secuencias de estados por las que pasa un objeto o una interacción durante su vida en respuesta a eventos, junto con sus reacciones a estos eventos. Sirven para especificar el comportamiento de una clase individual o una colaboración de clases. Un estado se representa como un rectángulo de esquinas redondeadas.

****

* 1. Actividad

Es un comportamiento que específica la secuencia de pasos que ejecuta un proceso computacional. En una interacción, el énfasis se pone en el conjunto de objetos que interactúan. En una máquina de estados, el énfasis se pone en el ciclo de vida de un objeto cada vez. En una actividad, el énfasis se pone en los flujos entre los pasos, sin mirar qué objeto ejecuta cada paso. Un paso de una actividad se denomina acción. Gráficamente, una acción se representa como un rectángulo con las esquinas redondeadas, con un nombre que indica su propósito. Los estados y las acciones se distinguen por sus diferentes contextos.

****

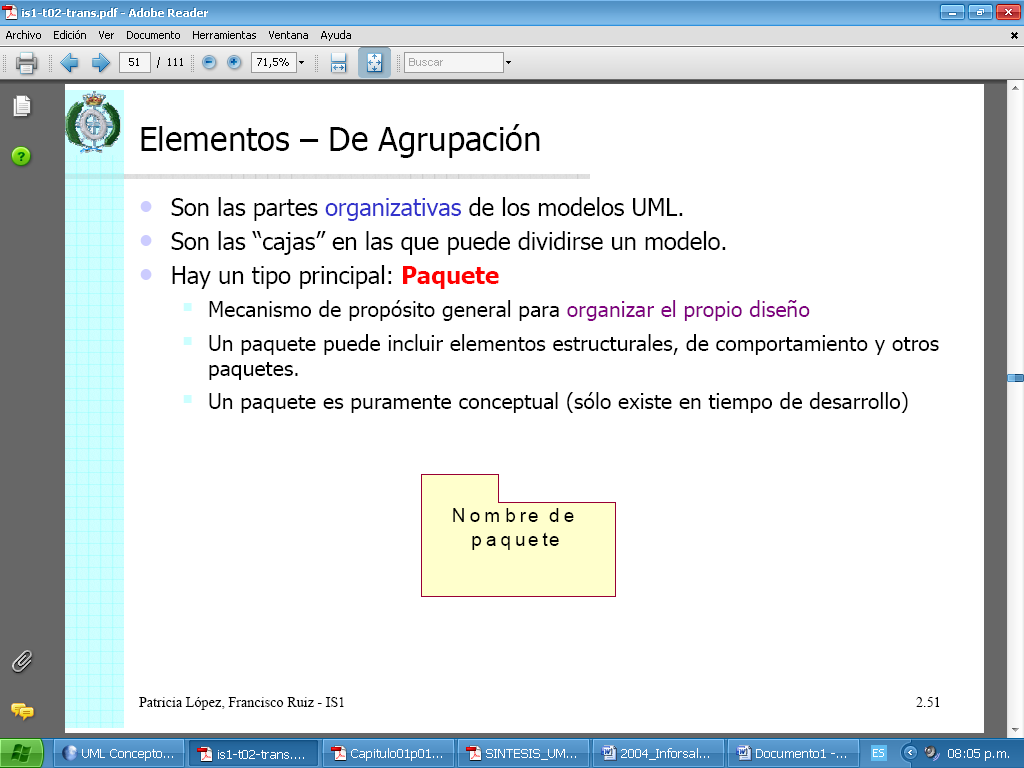
1. Elementos de Agrupación

Los elementos de agrupación son las partes organizativas de los modelos UML. Estos son las cajas en las que se puede descomponer un modelo. Hay un tipo principal de elementos de agrupación.

* 1. Paquete

Un paquete es un mecanismo de propósito general para organizar el propio diseño. Un paquete puede incluir elementos estructurales, de comportamiento y otros paquetes.

Un paquete es puramente conceptual, sólo existe en la etapa de desarrollo. Gráficamente, un paquete se visualiza como una carpeta, normalmente incluye su nombre, y su contenido. Hay variaciones, tal como Frameworks, modelos y subsistemas (tipos de paquetes).



1. Notación del lenguaje de modelado unificado (UML)

La notación en UML es un conjunto de elementos gráficos y diagramas que tienen como finalidad modelar sistemas orientados a objetos y también con UML se pueden modelar otros sistemas ya sean de software, hardware u organizaciones del mundo real.

Al estar basado en una *notación gráfica*, es muy expresivo y permite representar en mayor o menor medida las fases de un proyecto.

1. UML ofrece una notación y semántica estándar

UML prescribe una notación estándar y semánticas esenciales para el modelado de un sistema orientado a objetos.

Previamente, un diseño orientado a objetos podría haber sido modelado con cualquiera de la docena de metodologías populares, causando a los revisores tener que aprender las semánticas y notaciones de la metodología empleada antes que intentar entender el diseño en sí.

Con UML, diseñadores diferentes modelando sistemas diferentes pueden entender casa uno los diseños de los otros.

Si tenemos varias perspectivas a la hora de representar un desarrollo de software, en otras palabras cada desarrollo al que nos enfrentemos podríamos analizarlo desde diversos ángulos, que hace que detectemos en los diversos aspectos del sistema. Con los modelos podremos representar las diversas perspectivas desde donde estaríamos evaluando el desarrollo del software.

1. Elementos Notacionales

A continuación se verán una serie de elementos notaciones pero sin entrar en detalle en la esencia de lo que es el concepto de cada uno de ellos.

También visualizaremos como representar gráficamente los conceptos principales de la orientación a objetos.

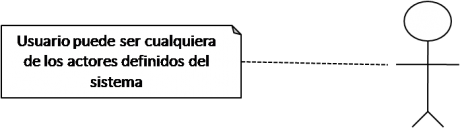
A continuación listaremos los siguientes elementos que serán tratados:

* Nota.
* Dependencia.
* Clase.
* Objetos.
* Atributos.
* Métodos.
* Asociaciones.
* Dependencias.
* Responsabilidades.
* Navegabilidad.
* Composición.
* Relaciones.
* Herencia.
* Generalización.
* Actores.
* Relaciones.
* Caso de uso.
* Línea de vida.
* Activación.
* Mensaje.

1. Elementos comunes entre diagramas.

Normalmente cuanto estamos construyendo un diagrama, surge información que nos puede ayudar a complementar el proceso o lo que estamos tratando de plasmar en el diagrama, información que se requiere en formato libre, información que denominamos comentario y que dentro de la estructura del diagrama no podemos expresarla en forma adecuada.

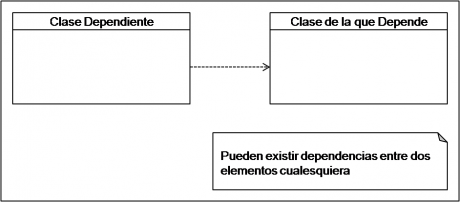
Una nota se representa como un rectángulo con una esquina doblada con texto en su interior.  Puede incluirse en un diagrama como una nota sola, o estar unida la nota a un elemento por medio de una línea descontinúa.  Puede contener restricciones, comentarios, el cuerpo de un procedimiento, etc.



Ejemplo de una nota

1. Dependencias

Debemos validar si entre los elementos que componen un diagrama existe alguna dependencia entre ellos, en estos casos hay que tomar en cuenta que al hacer un cambio en el elemento destino puede significar un cambio  en el elemento origen.  Una dependencia se representa por medio de una línea discontinua entre los dos elementos, con una flecha que sale del elemento origen y llega al elemento destino.



Ejemplo de una dependencia

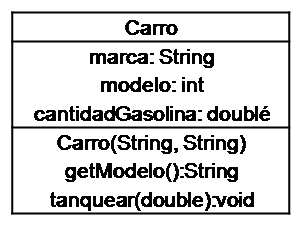
1. Diagrama de Estructura estática

Un diagrama de estructura estática muestra el conjunto de clases y objetos importantes que hacen parte de un sistema, junto con las relaciones existentes entre estas clases y objetos. Muestra de una manera estática la estructura de información del sistema y la visibilidad que tiene cada una de las clases, dada por sus relaciones con las demás en el modelo.

* 1. Clase

Una clase se representan como un rectángulo dividido en tres partes (tres compartimentos).

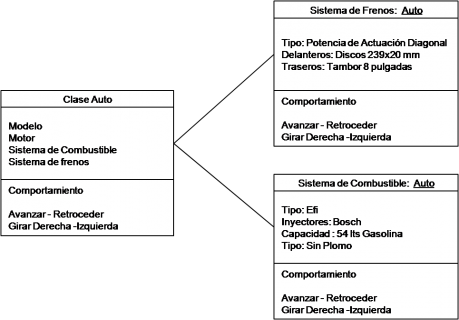
* En el compartimento superior, centrado, va el nombre de la clase.
* El segundo compartimento contiene los atributos, en la forma: visibilidad nombreAtributo: tipo
* El tercer compartimento contiene los métodos, en la forma: visibilidad nombreMétodo(parámetros): tipoRetorno.
* Los parámetros van separados por comas y sólo se especifica el tipo.
* Los constructores no tienen tipo de retorno.
* La visibilidad, de atributos y métodos, puede ser:  – privado, + público, # protegido, ~ de paquete



Ejemplo de clase

* 1. Objetos

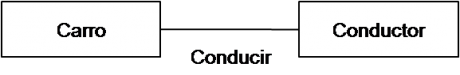
Se representa igual a una clase, debemos seguir la siguiente sintaxis nombre del objeto junto con el nombre de la clase subrayado. Nombre del objeto:nombre de la clase

[](http://1.bp.blogspot.com/-07gIuFpM2qM/TyNOJ5PoW-I/AAAAAAAAAvw/O2Yut30Th28/s1600/ejemploobjeto1.png)

Ejemplo de objetos

* 1. Asociaciones

Una asociación (relación entre dos clases) se representa como una línea continua entre dos Clases, y puede tener el nombre de la relación sobre esta línea. Ejemplo:

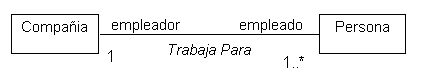
[](http://3.bp.blogspot.com/-fiUZI3Kazyk/TyNOLf0RuxI/AAAAAAAAAwg/nGFTGwyqqyU/s1600/ejemploasociacionconducir.png)

Ejemplo de asociaciones

En el ejemplo anterior cada Carro tiene una relación con Conductor, pero también cada Conductor tiene relación con Carro. Esta es una relación en ambos sentidos.

1. Asociación binaria

Se identifica como una línea sólida que une dos clases. Representa una relación de algún tipo entre las dos clases, no muy fuerte ( es decir, no se exige dependencia existencial ni encapsulamiento).



Un ejemplo es la relación entre una compañía y sus empleados

En este caso la relación recibe el nombre genérico *Trabaja Para*, la compañía tiene uno o más instancias de la clase Persona denominadas *empleado*y cada empleado conoce su *empleador*(en este caso único).

1. Sentido

Para mostrar que la relación sólo tiene un sentido se muestra con una flecha que indica el sentido de la relación. Ejemplo:

[http://3.bp.blogspot.com/-41X-hGO6xZo/TyNOLM3sc0I/AAAAAAAAAwU/Dc6Dta2lGKA/s1600/ejemploasociacion2.png](http://3.bp.blogspot.com/-41X-hGO6xZo/TyNOLM3sc0I/AAAAAAAAAwU/Dc6Dta2lGKA/s1600/ejemploasociacion2.png)

En este ejemplo un Pasajero conoce el Carro(o carros) con el cual viaja, pero el Carro no tiene ninguna relación con los Pasajeros.

1. Multiplicidad

Es una restricción que se pone a una asociación, que limita el número de instancias de una clase que pueden tener esa asociación con una instancia de la otra clase.

[http://3.bp.blogspot.com/-DrrJUVYz7BU/TyNOJozrTeI/AAAAAAAAAvg/On53-yx69WM/s1600/ejemplomultiplicidad.png](http://3.bp.blogspot.com/-DrrJUVYz7BU/TyNOJozrTeI/AAAAAAAAAvg/On53-yx69WM/s1600/ejemplomultiplicidad.png)

En este caso las relaciones son:

- Un chofer tiene relación con cero o más autobuses.

- Un autobús tiene relación con uno o dos choferes.

- Una terminal de pasajero tiene relación con cero o muchos autobuses.

- Un autobús tiene relación con un terminal de pasajero.

1. Roles

Para indicar el papel que juega una clase en una asociación se puede especificar un nombre de rol.  Se representa en el extremo de la asociación junto a la clase que desempeña dicho rol.

ejemploroles

Ejemplo de roles.

1. Composición

Es una asociación fuerte, que implica tres cosas:

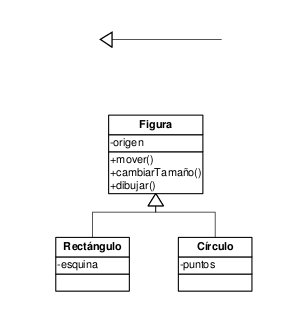
1. Dependencia existencial. El elemento dependiente desaparece al destruirse el que lo contiene y, si es de cardinalidad 1, es creado al mismo tiempo.
2. Hay una pertenencia fuerte. Se puede decir que el objeto contenido es parte constitutiva y vital del que lo contiene
3. Los objetos contenidos no son compartidos, esto es, no hacen parte del estado de otro objeto.

Se denota dibujando un rombo relleno del lado de la clase que contiene a la otra en la relación.

Existe también una relación de composición menos fuerte (no se exige dependencia existencial, por ejemplo) que es denotada por una un rombo sin rellenar en uno de los extremos.

* 1. Generalización

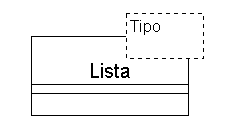
La relación de generalización denota una relación de herencia entre clases. Se representa dibujando un triángulo sin rellenar en el lado de la superclase. La subclase hereda todos los atributos y mensajes descritos en la superclase.



* 1. Clase paramétrica

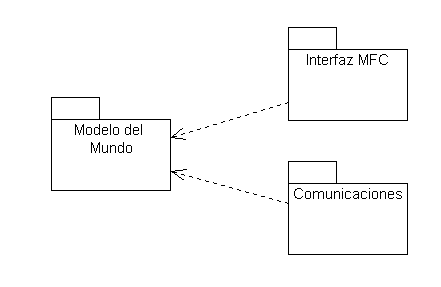
Una clase paramétrica representa el concepto de clase genérica en los conceptos básicos OO o de témplate en C++. Se dibuja como una clase acompañada de un rectángulo en la esquina superior derecha, con los parámetros del caso.

Por ejemplo, la clase Lista que utiliza un parámetro formal *Tipo*se vería de la siguiente manera



* 1. Paquete

Un paquete es una forma de agrupar clases (u otros elementos en otro tipo de diagramas) en modelos grandes. Pueden tener asociaciones de dependencia o de generalización entre ellos. Un ejemplo puede ser el siguiente

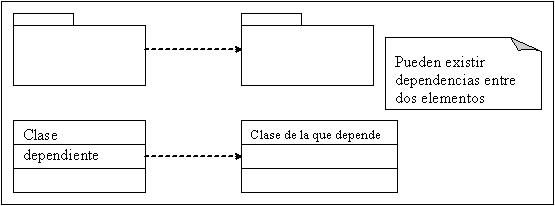


Ejemplo de paquetes

En este caso existen tres paquetes (que se muestran vacios en este caso, con su contenido encapsulado), con dos de ellos dependiendo del Modelo del Mundo.

* 1. Dependencia

Denota una relación semántica entre dos elementos (clases o paquetes, por el momento) del modelo. Indica que cambiar el elemento independiente puede requerir cambios en los dependientes. Se muestra como una línea punteada direccional, indicando el sentido de la dependencia. Puede tener por medio de estereotipos una explicación del tipo de dependencia presentada.



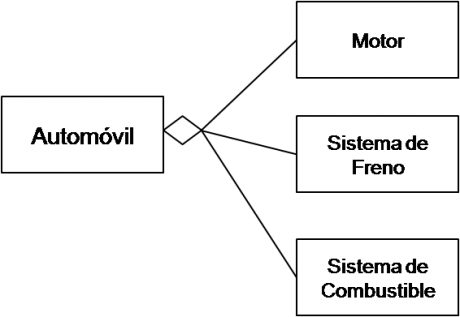
En el ejemplo anterior pueden verse dos relaciones de dependencia hacia el paquete Modelo del Mundo.

* 1. Agregación

La agregación representa el objeto compuesto.  Durante el desarrollo de una aplicación se nos presentara la necesidad de crear objetos complejos que no encajan con los tipos de datos básicos que proveen los lenguajes: tipo caracteres, enteros, reales, entre otros. El símbolo de agregación es un diamante colocado en el extremo en el que está la clase que representa el todo.  Podemos trabajar con dos tipos de agregación: Agregación por Valor y Agregación por Referencia.

1. Agregación por contenido físico o por valor

El contenedor contiene el objeto en sí. Cuando creamos un objeto contenedor, se crean también automáticamente los contenidos.  Ejemplo:

[](http://2.bp.blogspot.com/-7qR8Lkr2r34/TyNOK8F-SNI/AAAAAAAAAwQ/-78ymkfO3G4/s1600/ejemploagregacionvalor.png)

1. Agregación conceptual o por referencia

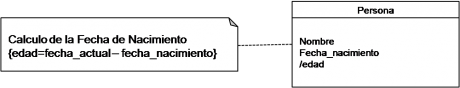
Se tienen punteros a objetos. No hay un acoplamiento fuerte. Los objetos se crean y se destruyen dinámicamente.

[](http://3.bp.blogspot.com/-jQ3eGWooiMU/TyNOKllO7sI/AAAAAAAAAwA/TijvtF8fCbA/s1600/ejemploagregacionreferencia.png)

En ambos casos la agregación se destaca por un rombo transparente.

* 1. Elementos Derivados

 En algunas ocasiones nos encontramos con la posibilidad de calcular un valor de un elemento a partir de otros elementos presentes en el modelo.  Se incluye para dar mayor claridad al diseño.  Se representa con una barra / precediendo al nombre del elemento derivado.

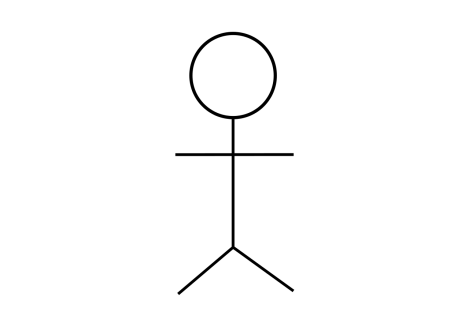
[](http://1.bp.blogspot.com/-BZJHi1k8vU8/TyNOL05EheI/AAAAAAAAAwk/Al2IR1Aj6BA/s1600/ejemploderivado.png)

1. Diagrama de caso de uso

Es la representación gráfica de cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico.  El caso de uso es la representación gráfica de cada proceso desde la perspectiva usuario, lo que se espera del sistema en cuanto a funcionalidad.

* 1. Actor

Representa un rol que es llevado a cabo por una persona, otro sistema, sensores, etc. El actor es representado por una figura en forma de persona.



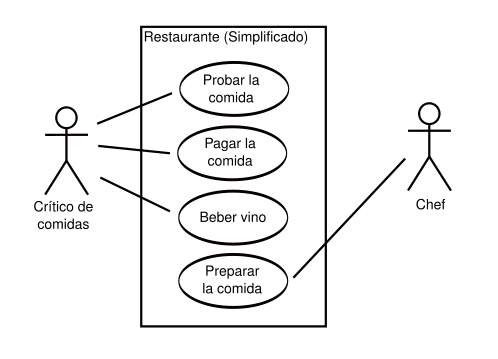
* 1. Relaciones en un diagrama de caso de uso

Este elemento representa la relación que existe entre un Caso-Uso y un Actor, dicho elemento es representado por una línea recta que se extiende de la figura del actor hacia el ovalo del caso-uso.

Entre dos casos de uso puede haber relaciones:

1. Extiende: Cuando un caso de uso especializa a otro extendiendo su funcionalidad.
2. Usa: Cuando un caso de uso utiliza a otro.

Se representa como una línea que une a los dos casos de uso relacionados, con una flecha en forma de triángulo y con una etiqueta “extiende” o “usa” según sea el tipo de relación.



* 1. Caso de uso

Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Se representa mediante un ovalo.

* 1. Límite de Sistema:

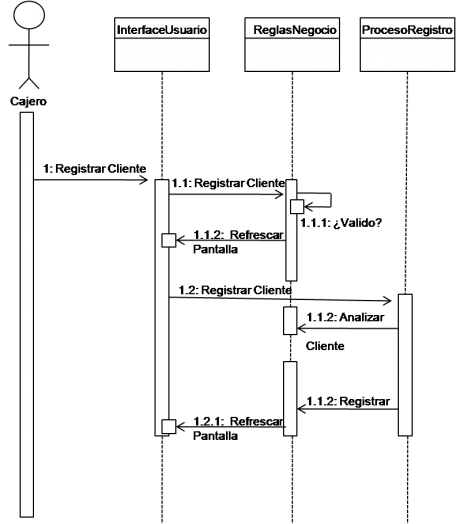
Empleado para delimitar los límites del sistema.  Se representa por un rectángulo con color de fondo distintivo.

1. Diagrama de secuencia

Muestra una interacción ordenada según la secuencia temporal de eventos.  El diagrama de secuencia muestra la interacción entre los objetos que tiene lugar a través del intercambio de mensajes, en particular, muestra los objetos participantes en la interacción y los mensajes que se intercambian ordenadas según la secuencia de tiempo.

* 1. Línea de vida de un objeto

Un objeto se representa como una línea vertical punteada con un rectángulo de encabezado y con rectángulos a través de la línea principal que denotan la ejecución de métodos. El rectángulo de encabezado contiene el nombre del objeto y el de su clase, en un formato *nombreObjeto: nombreClase*



* 1. Activación

Muestra el periodo de tiempo en el cual el objeto se encuentra desarrollando alguna operación, bien sea por sí mismo o por medio de delegación a alguno de sus atributos. Se denota como un rectángulo delgado sobre la línea de vida del objeto.

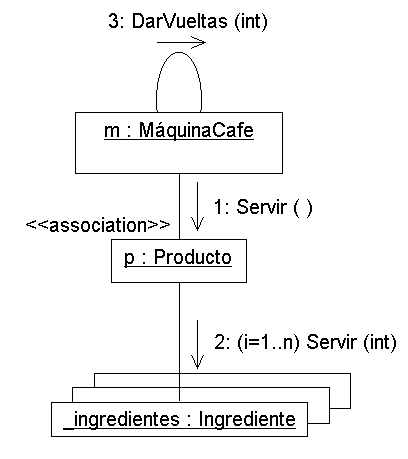
* 1. Mensaje

El envío de mensajes entre objetos se denota mediante una línea sólida dirigida, desde el objeto que emite el mensaje hacia el objeto que lo ejecuta.

Los diagramas de clases y los objetos representan información estática, no obstante en un sistema funcional los objetos interactúan entre sí, y tales interacciones suceden con el tiempo, el diagrama de secuencias UML muestra la mecánica de interacción con base en tiempos.

1. [Diagrama de Colaboración](http://webdocs.cs.ualberta.ca/~pfiguero/soo/uml/colaboracion01.html)

Un diagrama de colaboración es una forma de representar interacción entre objetos, alterna al diagrama de secuencia. A diferencia de los diagramas de secuencia, pueden mostrar el contexto de la operación (cuáles objetos son atributos, cuáles temporales, etc. ) y ciclos en la ejecución. Se toma como ejemplo el caso de uso PedirProducto ya descrito como diagrama de secuencia.



* 1. Objeto

Un objeto se representa con un rectángulo, que contiene el nombre y la clase del objeto en un formato *nombreObjeto: nombreClase*.

* 1. Enlaces

Un enlace es una instancia de una asociación en un diagrama de clases. Se representa como una línea continua que une a dos objetos. Esta acompañada por un número que indica el orden dentro de la interacción y por un estereotipo que indica que tipo de objeto recibe el mensaje. Pueden darse varios niveles de subíndices para indicar anidamiento de operaciones. Los estereotipos indican si el objeto que recibe el mensaje es un atributo (asociación y se asume por defecto), un parámetro de un mensaje anterior, si es un objeto local o global.

* 1. Flujo de mensajes

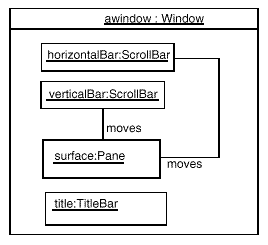
Expresa el envío de un mensaje. Se representa mediante una flecha dirigida cercana a un enlace.

Marcadores de creación y destrucción de objetos

Puede mostrarse en la gráfica cuáles objetos son creados y destruidos, agregando una restricción con la palabra *new*o *delete*, respectivamente, cercana al rectángulo del objeto

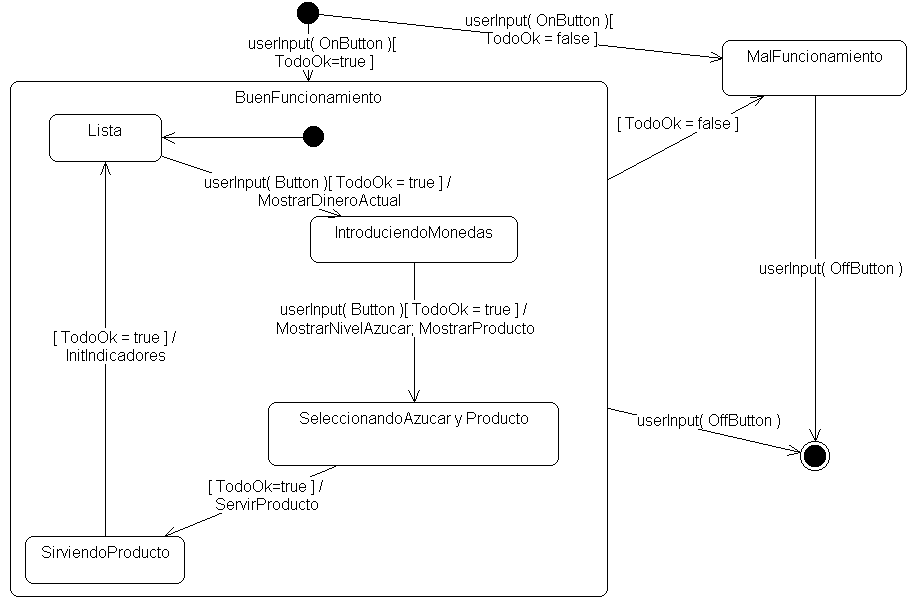
* 1. Objeto compuesto

Es una representación alternativa de un objeto y sus atributos. En esta representación se muestran los objetos contenidos dentro del rectángulo que representa al objeto que los contiene. Un ejemplo es el siguiente objeto ventana



1. Diagrama de Estados

Muestra el conjunto de estados por los cuales pasa un objeto durante su vida en una aplicación, junto con los cambios que permiten pasar de un estado a otro.Un ejemplo en el caso de la cafetera son los estados posibles para la clase MaquinaCafe:



* 1. Estado

Identifica un periodo de tiempo del objeto (no instantáneo) en el cual el objeto esta esperando alguna operación, tiene cierto estado característico o puede recibir cierto tipo de estímulos. Se representa mediante un rectángulo con los bordes redondeados, que puede tener tres compartimientos: uno para el nombre, otro para el valor característico de los atributos del objeto en ese estado y otro para las acciones que se realizan al entrar, salir o estar en un estado (entry, exit o do, respectivamente). En el caso del ejemplo anterior, se tienen cuatro estados (EnFuncionamiento, SinCambio, SinIngredientes, MalFuncionamiento) , en los cuales se desarrollan ciertas acciones al entrar; por ejemplo, al entrar al estado SinIngredientes se debe realizar la accion "Indicador SinIngredientes en On".

Se marcan también los estados iniciales y finales mediante los símbolos Estado inicial y Ejemplo de estado final, respectivamente.

* 1. Eventos

Es una ocurrencia que puede causar la transición de un estado a otro de un objeto. Esta ocurrencia puede ser una de varias cosas:

\*Condición que toma el valor de verdadero o falso

\*Recepción de una señal de otro objeto en el modelo

\*Recepción de un mensaje

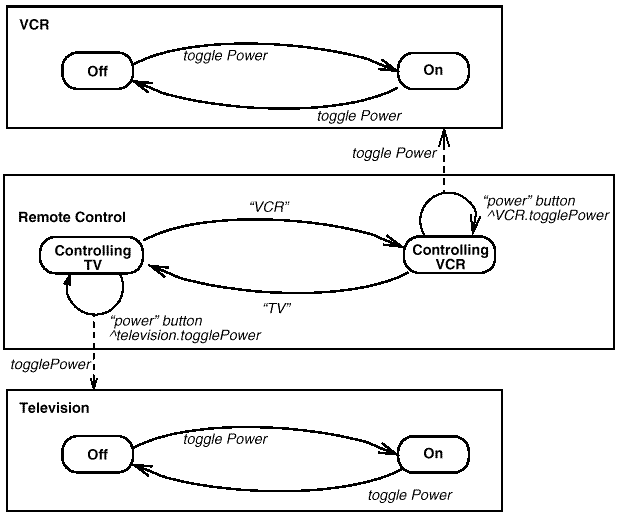
\*Paso de cierto período de tiempo, después de entrar al estado o de cierta hora y fecha particular

El nombre de un evento tiene alcance dentro del paquete en el cual está definido, no es local a la clase que lo nombre.

En el caso del ejemplo anterior se encuentra nombrado en varias transiciones el evento **userInput**, que recibe como parámetro un **Button**, para indicar el botón que ha sido presionado por el usuario de la máquina de café.

* 1. Envío de mensajes

Además de mostrar y transición de estados por medio de eventos, puede representarse el momento en el cual se envían mensajes a otros objetos. Esto se realiza mediante una línea punteada dirigida al diagrama de estados del objeto receptor del mensaje. Si tomamos como ejemplo un control remoto que puede enviar órdenes de encender o apagar al televisor o a la videograbadora se puede obtener un diagrama de estados como el siguiente:



Los tres aparatos tienen diagramas de estados separados y algunas de las transiciones del control remoto causan el envío de mensajes (*togglePower*) a los otros aparatos.

* 1. Transición simple

Una transición simple es una relación entre dos estados que indica que un objeto en el primer estado puede entrar al segundo estado y ejecutar ciertas operaciones, cuando un evento ocurre y si ciertas condiciones son satisfechas. Se representa como una línea sólida entre dos estados, que puede venir acompañada de un texto con el siguiente formato:

event-signature [guard-condition] /action-expression send-clause

event-signature es la descripción del evento que da a lugar la transición, guard-condition son las condiciones adicionales al evento necesarias para que la transición ocurra, action-expression es un mensaje al objeto o a otro objeto que se ejecuta como resultado de la transición y el cambio de estado y send-clause son acciones adicionales que se ejcutan con el cambio de estado, por ejemplo, el envio de eventos a otros paquetes o clases.

En el caso del ejemplo inicial de esta hoja se tiene una transición entre los estados IntroduciendoMoneda y SeleccionadoAzucaryProducto que tiene una transición con el siguiente detalle:

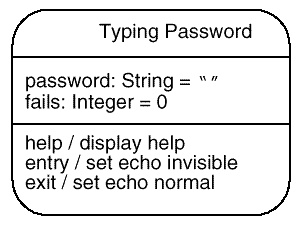
userInput( Button ) | [TodoOk=true} / MostrarNivelAzucar, MostrarProducto

El evento que dispara el cambio de estado es userInput( Button). Se requiere como condición adicional que no se haya detectado ninguna falla (TodoOk = true) y se ejecuta MostrarNivelAzucar y MostrarProducto, que deberían ser ejecutables por el objeto al cual pertenece el diagrama.

* 1. Transición interna

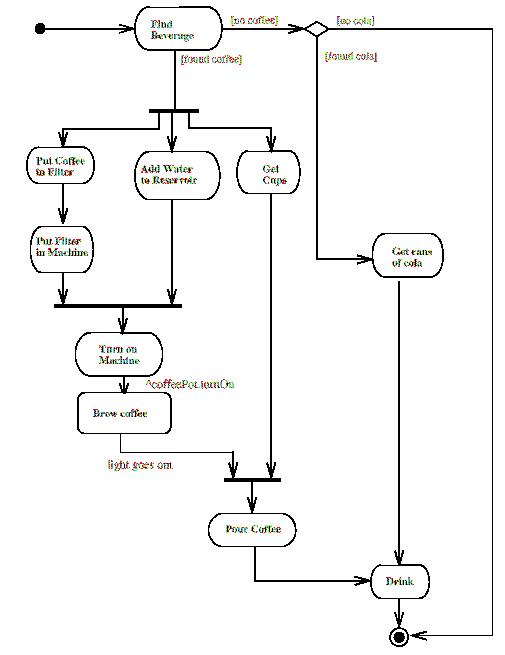
Es una transición que permanece en el mismo estado, en vez de involucrar dos estados distintos. Representa un evento que no causa cambio de estado. Se denota como una cadena adicional en el compartimiento de acciones del estado.

Supongamos el estado de una interfaz pidiendo password al usuario. En este caso puede tenerse una transición interna que muestre una ayuda al usuario. Esta transición se muestra en el siguiente diagrama con la cadena "help / display help " dentro del cuerpo del estado.



* 1. [Diagrama de Actividades](http://webdocs.cs.ualberta.ca/~pfiguero/soo/uml/actividades01.html)

Un diagrama de actividades es un caso especial de un diagrama de estados en el cual casi todos los estados son estados de acción (identifican que acción se ejecuta al estar en él) y casi todas las transiciones son enviadas al terminar la accion ejecutada en el estado anterior. Puede dar detalle a un caso de uso, un objeto o un mensaje en un objeto. Sirven para representar transiciones internas, sin hacer mucho énfasis en transiciones o eventos externos. Se presenta a continuación un ejemplo de diagrama de actividades para un mensaje de un objeto. Generalmente modelan los pasos de un algoritmo.



* 1. Estado de acción

Representa un estado con acción interna, con por lo menos una transición que identifica la culminación de la acción (por medio de un evento implícito). No deben tener transiciones internas ni transiciones basadas en eventos (Si este es el caso, represéntelo en un diagrama de estados). Permite modelar un paso dentro del algoritmo. Se representan por un rectángulo con bordes redondeados. 

* 1. Transiciones

Las flechas entre estados representan transiciones con evento implícito. Pueden tener una condición en el caso de decisiones.

* 1. Decisiones

Se representa mediante una transición múltiple que sale de un estado, donde cada camino tiene un label distinto. Se representa mediante un diamante al cual llega la transición del estado inicial y del cual salen las múltiples transiciones de los estados finales. Un ejemplo se ve en la figura cuando *no hay cafe* y se toma una decisión entre *hay cola* o *no hay cola*.

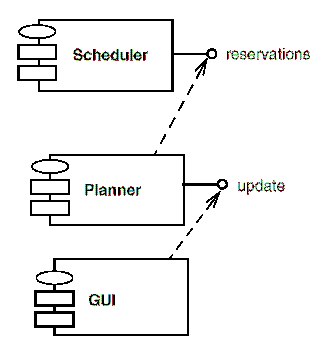
1. [Diagrama de Implementación](http://webdocs.cs.ualberta.ca/~pfiguero/soo/uml/implementacion01.html)

Un diagrama de implementación muestra la estructura del código (Diagrama de componentes) y la estructura del sistema en ejecución (Diagrama de ejecución).

1. Diagrama de Componentes

Un diagrama de componentes muestra las dependencias lógicas entre componentes software, sean éstos componentes fuentes, binarios o ejecutables. Los componentes software tienen tipo, que indica si son útiles en tiempo de compilación, enlace o ejecución. Se consideran en este tipo de diagramas solo tipos de componentes. Instancias específicas se encuentran en el diagrama de ejecución.

Se representa como un grafo de componentes software unidos por medio de relaciones de dependencia (generalmente de compilación). Puede mostrar también contenencia de entre componentes software e interfaces soportadas. Un ejemplo es el siguiente:



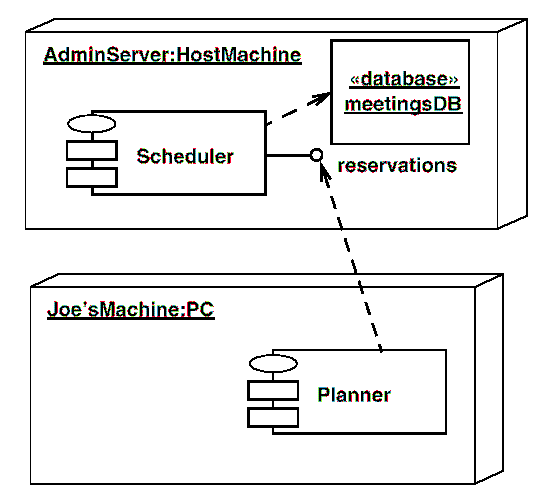
En este caso tenemos tres componentes, GUI dependiendo de la interfaz *update* provista por *Planner*, *Planner*dependiendo de la interfaz *reservations*provista por *Scheduler*.

1. Diagrama de Ejecución

Un diagrama de ejecución muestra la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes software, procesos y objetos que se ejecutan en ellos. Instancias de los componentes software representan manifestaciones en tiempo de ejecución del código. Componentes que solo sean utilizados en tiempo de compilación deben mostrarse en el diagrama de componentes.

Un diagrama de ejecución es un grafo de nodos conectados por asociaciones de comunicación. Un nodo puede contener instancias de componentes software, objetos, procesos (un caso particular de un objeto). Las instancias de componentes software pueden estar unidos por relaciones de dependencia, posiblemente a interfaces.

Un ejemplo de diagrama de ejecución es el siguiente:



En este caso se tienen dos nodos, *AdminServer*y *Joe'sMachine*. *AdminServer* contiene la instancia del componente *Scheduler*y un objeto activo (proceso) Denominado *meetingsDB*. En*Joe'sMachine* se encuentra la instancia del componente software *Planner*, que depende de la interfaz *reservations*, definida por *Scheduler*.

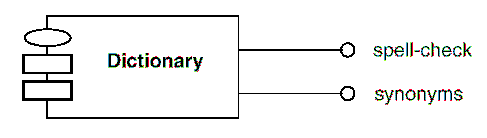
* 1. Nodos

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

* 1. Componentes

Un componente representa una unidad de código (fuente, binario o ejecutable) que permite mostrar las dependencias en tiempo de compilación y ejecución. Las instancias de componentes de software muestran unidades de software en tiempo de ejecución y generalmente ayudan a identificar sus dependencias y su localización en nodos.  Pueden mostrar también que interfaces implementan y qué objetos contienen. Su representación es un rectángulo atravesado por una elipse y dos rectángulos más pequeños.

Un ejemplo de componente que implementa dos interfaces es



Conclusión

No siempre utilizaremos todos los elementos que ofrece el UML, una vez conozcamos que función cumple cada uno, usaremos en cada momento las más adecuadas a nuestras necesidades. Eso significa que no es fácil su utilización y que solo después de entender para qué sirven cada uno de ellos y qué limitaciones tienen podremos conocer la utilidad e implementarlos.

Por consiguiente la premisa de que este conocimiento se puede alcanzar con un poco de práctica y sentido común.