1. LENGUAJE UNIFICADO DE MODELADO (UML) 1

2. CLASIFICACIÓN DIAGRAMAS DE UML 2

**1.** **DEFINICIÓN DE UML:** 16

**2.** **DIAGRAMA DE COMUNICACIÓN** 18

**3.** **DIAGRAMA DE ESTADO** 20

**4.** **DIAGRAMA DE PAQUETES** 22

**5.** **DIAGRAMA DE ACTIVIDAD** 23

**6.** **CONCLUSIONES**: 26

## LENGUAJE UNIFICADO DE MODELADO (UML)

El Lenguaje Unificado de Modelado (Unified Modeling Language, UML) es un lenguaje estándar para escribir planos de software, centrado en la representación gráfica de un sistema, nos indica cómo crear y.. leer los modelos, pero no dice como crearlos ya que esto último es el objetivo de las metodologías de desarrollo.

UML es algo más que un simple montón de gráficos. Detrás de cada símbolo en la notación UML hay una semántica bien definida. De esta manera, un desarrollador puede escribir un modelo en UML, y otro desarrollador, o incluso otra herramienta, puede interpretar ese modelo sin ambigüedad.

* 1. Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones
* Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
* Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.
* Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
* Documentar: Los propios elementos gráficos sirven como documentación del sistema des- arrollado que pueden servir para su futura revisión.
  1. UML está compuesto por tres clases de bloques de construcción
* Elementos: Los elementos son abstracciones de cosas reales o ficticias (objetos, acciones, etc.)
* Relaciones: relacionan los elementos entre sí.
* Diagramas: Son colecciones de elementos con sus relaciones.
  1. Hay cuatro tipos de elementos UML
* Elementos estructurales
* Elementos de comportamiento
* Elementos de agrupación.
* Elementos de anotación.

## CLASIFICACIÓN DIAGRAMAS DE UML

La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. Recordemos que un modelo es una representación simplificada de la realidad; el modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementar dicho sistema.

2.1 DIAGRAMAS DE ESTRUCTURA

* **DIAGRAMA DE CLASES**

Los diagramas de clases describen la estructura estática de un sistema.

Elementos de un diagrama de clase:

Elementos estructurales

|  |
| --- |
| NOMBRE DE LA CLASE |
| Atributos: Tipo |
| Acciones() |

Para poder relacionar varias clases es necesario usar otro elemento conocido como asociaciones. Estas representan las relaciones estáticas entre una o más clases. El nombre de la asociación va por sobre o por debajo de la línea que la representa. Una flecha rellena indica la dirección de la relación.

Nombre

CLASE 1

CLASE 2

CLASE 1

CLASE 2

Las notaciones utilizadas para señalar la multiplicidad se colocan cerca del final de una asociación. Estos símbolos indican el número de instancias de una clase vinculadas a una de las instancias de la otra clase. Por ejemplo, una empresa puede tener uno o más empleados, pero cada empleado trabaja para una sola empresa solamente.

Podemos hablar sobre dos tipos de especialización de la relación de asociación las cuales son la agregación y la composición. La agregación Es muy similar a la relación de Asociación solo varía en la multiplicidad ya que en lugar de ser una relación "uno a uno" es de "uno a muchos" Se grafica con un rombo diamante vacío como lo vemos a continuación.

CLASE 1

CLASE 2

La Composición es una relación más fuerte y similar a la relación de asociación. Aporta documentación conceptual ya que es una "relación de vida", es decir, el tiempo de vida de un objeto está condicionado por el tiempo de vida del objeto que lo incluye.

CLASE 1

CLASE 2

Generalización es otro nombre para herencia. Se refiere a una relación entre dos clases en donde una Clase “Específica” es una versión especializada de la otra, o Clase “General”. Por ejemplo, Honda es un tipo de auto, por lo que la Clase “Honda” va a tener una relación de generalización con la Clase “Auto”. Esta se representa a través de un triángulo vacío hacia arriba.

CLASE 1

CLASE 2

**DIAGRAMA DE OBJETOS**

Los Diagramas de Objetos están vinculados con los Diagramas de Clases. Un objeto es una instancia de una clase, por lo que un diagrama de objetos puede ser visto como una instancia de un diagrama de clases. Los diagramas de objetos describen la estructura estática de un sistema en un momento particular y son usados para probar la precisión de los diagramas de clases.

Cada objeto es representado como un rectángulo, que contiene el nombre del objeto y su clase subrayadas y separadas por dos puntos.

Nombre Objeto: clase

Como con las clases, los atributos se listan en un área inferior. Sin embargo, los atributos de los objetos deben tener un valor asignado.

|  |
| --- |
| Nombre Objeto: clase |
| Atributo tipo= ‘valor’  Atributo tipo= ‘valor’  Atributo tipo= ‘valor’  Atributo tipo= ‘valor’ |

**DIAGRAMA DE CASOS DE USO**

Un caso de uso es una descripción de las acciones de un sistema desde el punto de vista del usuario. Es una herramienta valiosa dado que es una técnica de aciertos y errores para obtener los requerimientos del sistema, justamente desde el punto de vista del usuario. Los diagramas de caso de uso modelan la funcionalidad del sistema usando actores y casos de uso. Los casos de uso son servicios o funciones provistas por el sistema para sus usuarios

**Actores**

El rectángulo representa los límites del sistema que contiene los casos de uso. Los actores se ubican fuera de los límites del sistema.

Los casos de usos se representan con óvalos. La etiqueta en el óvalo indica la función del sistema, y los actores son los usuarios del sistema.

**DIAGRAMA DE COMPONENTES**

Este tipo de diagramas se utilizan para representar entidades reales, esto es, componentes de software. Un componente de software puede ser una tabla, un archivo de datos, un ejecutable, documentos etc.

El símbolo del componente es un rectángulo con otros dos rectángulos sobrepuestos.

Componente

Se puede añadir en su interior, información que indique detalles del componente.

Una interface describe a un grupo de operaciones usada o creada por componentes, estas están representadas con una línea y al final un círculo.

Las dependencias entre los componentes se grafican usando flechas con puntos.

Componente1

Componente2

Componente3

**DIAGRAMAS DE SECUENCIAS**

Los diagramas de clases y los de 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 la interacción con base en tiempos.

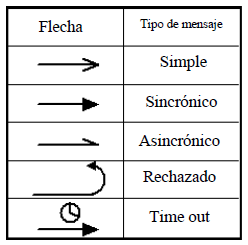
Los diagramas de secuencia tienen varios elementos importantes, entre estos tenemos los roles de la clase, las activaciones, mensajes, líneas de vida, repeticiones etc.

El rol de la clase describe la manera en que un objeto se va a comportar en el contexto. En estas no se listan los atributos del objeto.

Objeto: clase

Los cuadros de activación representan el tiempo que un objeto necesita para completar una tarea.

Los mensajes son flechas que representan comunicaciones entre objetos. Las medias flechas representan mensajes asincrónicos. Los mensajes asincrónicos son enviados desde un objeto que no va a esperar una respuesta del receptor para continuar con sus tareas.

****

Las líneas de vida son verticales y en línea de puntos, ellas indican la presencia del objeto durante el tiempo.

Los objetos pueden ser eliminados tempranamente usando una flecha etiquetada "<<destruir>>" que apunta a una X.

Una repetición o loop en un diagrama de secuencias, es representado como un rectángulo. La condición para abandonar el loop se coloca en la parte inferior entre corchetes [ ].

[Condición para salir]

Objeto: clase

Objeto: clase

<<Destruir>>

**X**

**ARQUITECTURA**

La visualización, especificación, construcción y documentación de un sistema con gran cantidad de software requiere que el sistema sea visto desde varias perspectivas. Diferentes usuarios (usuarios finales, analistas, desarrolladores, integradores de sistemas, encargados de las pruebas, encargados de la documentación técnica y jefes de proyectos) siguen diferentes agendas en relación al proyecto, y cada uno mira a ese sistema de formas diferentes en diversos momentos a lo largo de la vida del proyecto. La arquitectura de un sistema es quizás el artefacto más importante que puede emplearse para manejar estos diferentes puntos de vistas y controlar el desarrollo iterativo e incremental de un sistema a lo largo de su ciclo de vida

La Arquitectura es el conjunto de decisiones significativas sobre:

* La organización de un sistema software.
* La selección de elementos estructurales y sus interfaces a través de los cuales se construye el sistema.
* Su comportamiento, cómo se especifica en las colaboraciones entre esos elementos.
* La composición de esos elementos estructurales y de comportamiento en subsistemas progresivamente más grandes.
* El estilo arquitectónico que guía esta organización: los elementos estáticos y dinámicos y sus interfaces, sus colaboraciones y su composición.

La arquitectura del software no tiene que ver solamente con la estructura y el comportamiento, sino también con el uso, la funcionalidad, el rendimiento, la capacidad de adaptación, la reutilización, la capacidad de ser comprendido, las restricciones económicas y tecnológicas y los compromisos entre alternativas, así como los aspectos estéticos.

La arquitectura de un sistema con gran cantidad de software puede describirse mejor a través de cinco vistas interrelacionadas. Cada vista es una proyección de la organización y la estructura del sistema, centrada en un aspecto particular de ese sistema.

Vista de Diseño

Vista de Interacción

Vista de Despliegue

Vista de Implementación

Vocabulario,

Funcionalidad

Comportamiento

Rendimiento,

Escalabilidad,

Capacidad de procesamiento

Ensamblado del Sistema,

Gestión de la configuración

Topología del Sistema, Distribución,

Entrega, Instalación

* **La vista de casos de uso:** comprende los casos de uso que describen el comportamiento del sistema tal y como es percibido por los usuarios finales, analistas y encargados de las pruebas. Esta vista no especifica realmente la organización de un sistema software. Más bien, existe para especificar las fuerzas que configuran la arquitectura del sistema. Con UML, los aspectos estáticos de esta vista se capturan en los diagramas de caso de uso; los aspectos dinámicos de esta vista se capturan en los diagramas de interacción, diagramas de estado y diagramas de actividades.
* **La vista de diseño:** comprende las clases, interfaces y colaboraciones que forman el vocabulario del problema y su solución. Esta vista soporta principalmente los requisitos funcionales del sistema, entendiendo por ellos los servicios que el sistema debería proporcionar a sus usuarios finales. Con UML, los aspectos estáticos de esta vista se capturan en los diagramas de clases y de objetos; los aspectos dinámicos se capturan en los diagramas de interacción, diagramas de estado y diagramas de actividades.
* **La vista de interacción:** esta vista muestra el flujo de control entre sus diversas partes, incluyendo los posibles mecanismos de concurrencia y sincronización. Esta vista abraca principalmente el rendimiento, la escalabilidad, y la capacidad de procesamiento del sistema. Con UML, los aspectos estáticos y dinámicos de esta vista se capturan en el mismo tipo de diagramas que la vista de diseño, pero con mayor atención sobre las clases activas que controlan el sistema y los mensajes que fluyen entre ellas.
* **La vista de implementación:** comprende los artefactos que se utilizan para ensamblar y hacer posible el sistema físico. Esta vista se preocupa principalmente de la gestión de configuraciones de las distintas versiones de un sistema, a partir de componentes y archivos un tanto independientes y que pueden ensamblarse de varias formas para producir un sistema en ejecución. Con UML, los aspectos estáticos de esta vista se capturan en los diagramas de componentes; los aspectos dinámicos de esta vista se capturan en los diagramas de interacción, diagramas de estados y diagramas de actividades.
* **La vista de despliegue:** contiene los nodos que forman la topología hardware sobre la que se ejecuta el sistema. Esta vista se preocupa principalmente de la distribución, entrega e instalación de las partes que constituyen el sistema físico. Con UML, los aspectos estáticos de esta vista se capturan en los diagramas de despliegue; los aspectos dinámicos de esta vista se capturan en los diagramas de interacción, diagramas de estados y diagramas de actividades.

Cada una de estas cinco vistas puede existir por sí misma, de forma que diferentes usuarios puedan centrarse en las cuestiones de la arquitectura del sistema que más les interesen. Estas cinco vistas también pueden interactuar entre si (los nodos en la vista de despliegue contienen componentes de la vista de implementación que, a su vez, representan la realización física de las clases, interfaces, colaboraciones y clases activas de la vistas de diseño y de procesos). UML permite expresar cada una de estas cinco vistas y sus interacciones.

**CICLO DE VIDA DEL DESARROLLO DE SOFTWARE**

UML es bastante independiente del proceso, lo que significa que no está ligado a ningún ciclo de vida de desarrollo de software particular. Sin embargo, para obtener el máximo beneficio de UML, se debería considerar un proceso que fuese:

* Dirigido por los casos de uso.
* Centrado en la arquitectura.
* Iterativo e incremental.

Dirigido por los casos de uso significa que los casos de uso se utilizan como un artefacto básico para establecer el comportamiento deseado del sistema, para verificar y validar la arquitectura del sistema, para las pruebas y para la comunicación entre las personas involucradas en el proyecto.

Centrado en la arquitectura significa que la arquitectura del sistema se utiliza como un artefacto básico para conceptuar, construir, gestionar y hacer evolucionar el sistema en desarrollo.

Un proceso iterativo es aquel que involucra la gestión de un flujo de versiones ejecutables del sistema.

Un proceso Incremental es aquel que involucra la continua integración de la arquitectura del sistema para producir esas versiones, donde cada nuevo ejecutable incorpora mejoras incrementales sobre los otros. En conjunto, un proceso iterativo e incremental está dirigido por el riesgo, lo que significa que cada nueva versión se encarga de atacar y reducir los riesgos más significativos para el éxito del proyecto.

Este proceso dirigido por casos de uso, centrado en la arquitectura, iterativo e incremental puede descomponerse en fases. Una fase es el intervalo de tiempo entre dos hitos importantes del proceso, cuando se cumplen un conjunto de objeticos bien definidos, se completan los artefactos y se toman las decisiones sobre si pasar o no a la siguiente fase.

Existen cuatro fases en el ciclo de vida del desarrollo de software: concepción, elaboración, construcción y transición. En el diagrama, los flujos de trabajo se representan frente a estas fases, mostrando como varia a lo largo del tiempo el nivel de atención que una fase presta a un flujo de trabajo.



* **La concepción:** es la primera fase del proceso, cuando la idea inicial para el desarrollo se lleva al punto de estar suficientemente bien fundamentada para garantizar la entrada en la fase de elaboración.
* **La elaboración:** es la segunda fase del proceso, cuando se definen los requisitos del producto y su arquitectura. En esta fase se expresan con claridad los requisitos del sistema, son priorizados y se utilizan para crear una solidad base arquitectónica. Los requisitos de un sistema pueden variar desde enunciando de carácter general hasta criterios precisos de evaluación, especificando cada uno un comportamiento funcional o no funcional y proporcionando y una referencia para las pruebas.
* **La construcción:** es la tercera fase del proceso, cuando el software se lleva desde una base arquitectónica ejecutable hasta su disponibilidad para la comunidad de usuarios. Aquí los requisitos del sistema y especialmente sus criterios de evaluación son constantemente reexaminados frente a las necesidades del proyecto, y los recursos se asignan al proyector de forma apropiada para atacar los riesgos.
* **La transición:** es la cuarta fase del proceso, cuando el software es puesto en las manos de la comunidad de usuarios. El proceso del software raramente termina aquí, porque incluso durante esta fase el sistema es mejorado continuamente, se erradican errores de programación, y se añaden características que no se incluían en las versiones anteriores.

Un elemento que distingue a este proyecto y que afecta a las cuatro fases es una iteración. Una Iteración es un conjunto bien definidos de actividades, con un plan y unos criterios de evaluación bien establecidos, que acaba en un sistema que puede ejecutarse, probarse y evaluarse. El sistema ejecutable no tiene por qué ser entregado externamente. Como la iteración produce un producto ejecutable, después de cada iteración se puede juzgar el progreso y se pueden reevaluar los riesgos. Esto significa que el ciclo de vida del desarrollo de software puede caracterizarse por involucrar un flujo continuo de versiones ejecutables de la arquitectura del sistema. Este énfasis en la arquitectura como un artefacto importante es el que conduce a UML a centrarse en el modelado de las diferentes vistas de la arquitectura de un sistema.

1. **DEFINICIÓN DE UML:**

UML es un lenguaje de modelado visual que sirve para:

* **Visualizar**
* **Especificar**
* **Construir**
* **Documentar**

Trabaja modelando sistemas, independientemente de la metodología de análisis y diseño, siempre y cuando se realice bajo una perspectiva orientada a objeto

Es un modelo explícito, que facilita la comunicación y a la construcción de modelos precisos, no ambiguos y completos.

**CONCEPTOS DE MODELO**

* SISTEMA: colección de elementos, posiblemente divididos en subsistemas, organizados para lograr un propósito. Está descrito por un conjunto de modelos.
* MODELO: simplificación completa y auto consistente de la realidad, creado para comprender un sistema.
* VISTA (Arquitectural): es una proyección de la organización y estructura de un modelo de un sistema, centrada en su aspecto. Incluye un subconjunto de los elementos en el modelo.
* DIAGRAMA: representación gráfica de un conjunto de elementos del modelo y sus relaciones. En UML generalmente corresponde a un grafo conexo de nodos (Objetos) y sus arcos (Relaciones).

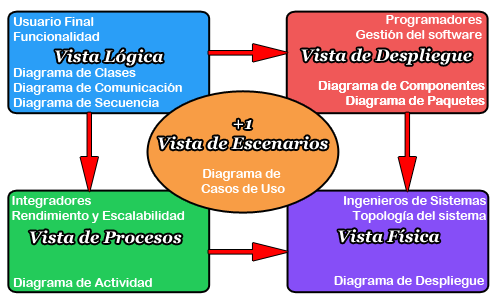
Un modelo captura las propiedades estructurales y de comportamiento de un sistema, es una abstracción de dicho sistema. El modelo describe completamente aquellos aspectos del sistema que son relevantes al propósito del modelo

**VISTAS ARQUITECTURALES**

Durante el desarrollo de un sistema de Software se requiere que este sea visto desde varias perspectivas

Diferentes usuarios miran el sistema de formas diferentes en momentos diferentes. La arquitectura del sistema es clave para poder manejar estos puntos de vista diferentes:

* Se organiza mejor a través de vistas arquitecturales interrelacionadas
* Proyecciones del modelo del sistema centrada en un aspecto particular.



**VISTA DE ESCENARIOS**

Captura la funcionalidad del sistema tal y como es percibido por los usuarios finales. Los diagramas correspondientes a esta vista son el diagrama de Casos de Uso

**VISTA LÓGICA**

Captura las clases, las interfaces, y colaboraciones que describen al sistema:

* En el dominio del problema
* En el dominio de la solución

Los elementos que la conforman dan soporte a los requisitos funcionales del sistema. Los diagramas correspondientes son los diagramas de clase, secuencia y comunicación.

**VISTA DE PROCESOS**

Captura el flujo de control entre las diversas partes del sistema, incluyendo los posibles mecanismos de concurrencia y sincronización. Abarca en especial requisitos no funcionales el rendimiento, la escalabilidad y capacidad de procesamiento

Los diagramas correspondientes son los diagramas de actividad

**VISTA FÍSICA**

Captura las características de instalación y ejecución del sistema. Contiene los nodos y enlaces que forman la topología hardware sobre la que se ejecuta el sistema software

Se ocupa principalmente de la distribución de las partes que forman el sistema software real. Los diagramas que corresponden es el diagrama de despliegue.

**VISTA DE DESPLIEGUE**

Captura los artefactos que se utilizan para ensamblar y poner en producción el sistema software real. Aquí se define la arquitectura física del sistema. Corresponde en esta vista los diagramas de paquetes y de componentes

Cada vista puedes existir de forma independiente, pero actúan entre sí los nodos

En resumen, el modelo UML es un conjunto de elementos de modelado que definen la estructura, el comportamiento y la funcionalidad del sistema. La presentación de estos conceptos se realiza a través de múltiples diagramas con el fin de hacerlo comprensibles. Estos diagramas se agrupan en vistas, cada una enfocada a un aspecto particular del sistema.

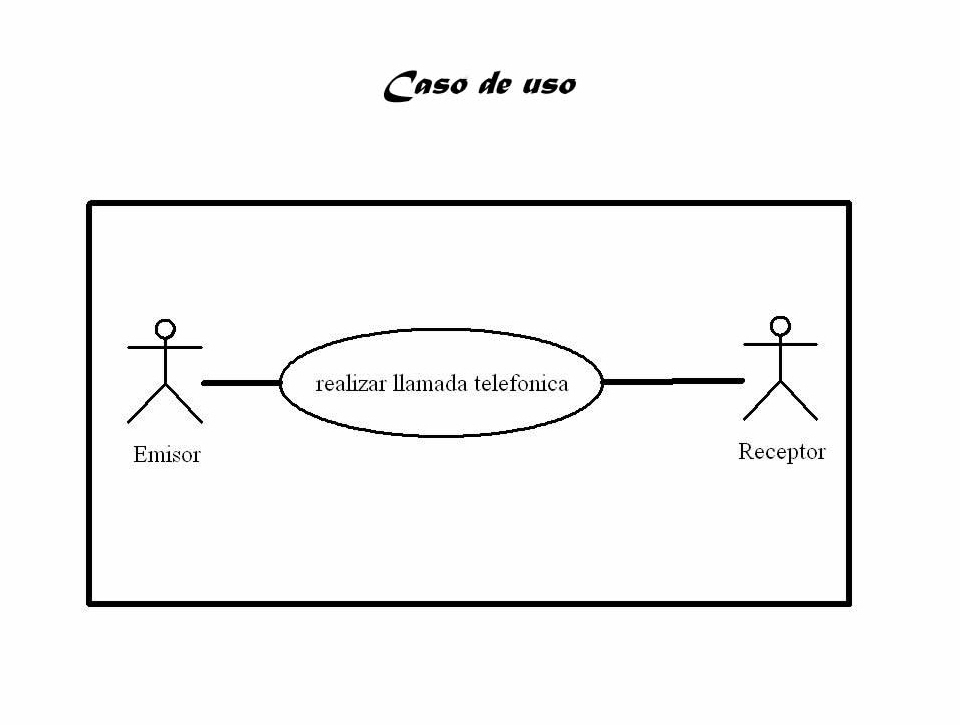
Un diagrama es una representación gráfica de un conjunto de elementos que con sus relaciones, ofrece una vista del sistema a modelar. Para poder representar correctamente un sistema, UML ofrece una amplia variedad de diagramas para visualizar el sistema desde varias perspectivas.

1. **DIAGRAMA DE COMUNICACIÓN**

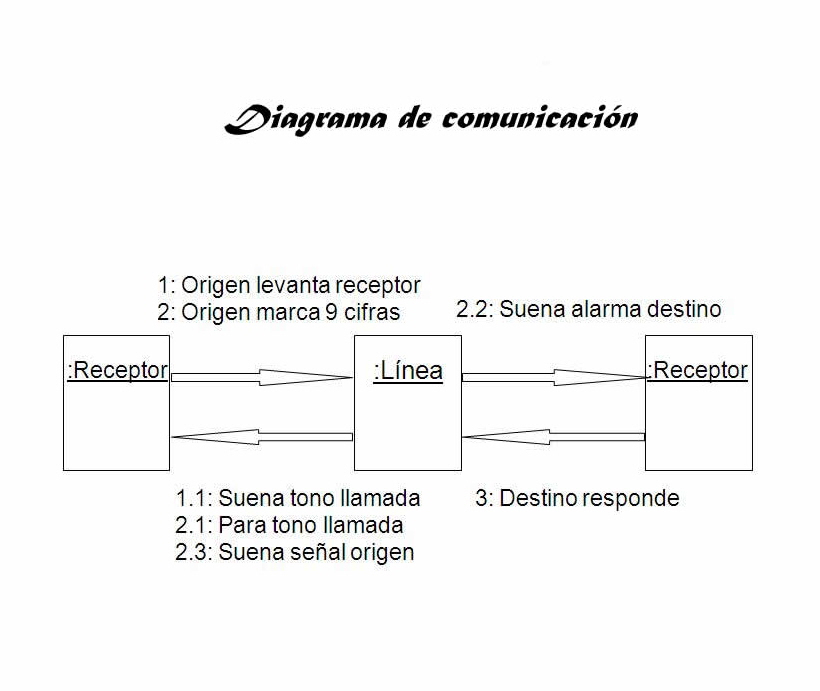
Un diagrama de comunicación es una versión simplificada del diagrama de colaboración de la versión de UML 1.X

Este diagrama describe las interacciones entre los objetos en términos de mensajes secuenciales. Es una forma interactiva del diagrama de secuencias. Existen herramientas que automáticamente convierten estos diagramas el uno en el otro

Este diagrama representa una combinación de información tomada desde el diagrama de clases, secuencial y el diagrama de caso de uso, describiendo tanto la estructura estática como el comportamiento dinámico de un sistema

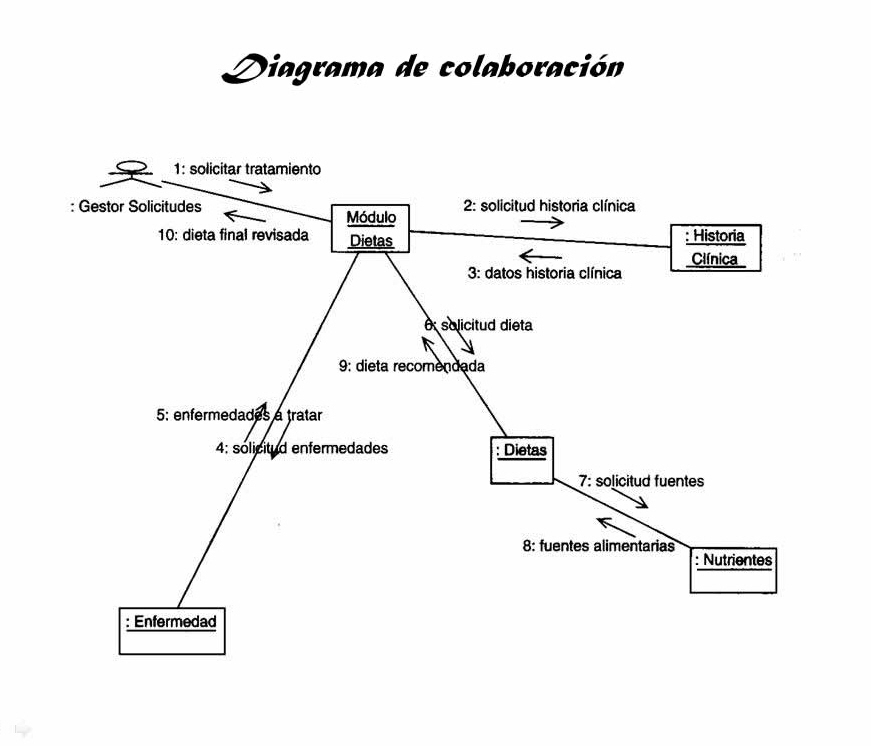


Para mantener el orden de los mensajes en un diagrama de comunicación, estos son etiquetados con un número cronológico y colocado cerca del enlace por el cual se desplaza el mensaje.



Leer un diagrama de comunicación conlleva comenzar en el mensaje 1.0 y seguir los mensajes desde un objeto hasta el siguiente sucesivamente. Algunas de sus ventajas son que si el esquema es simple, el comportamiento también será simple. Existe una equivalencia semántica, además de que este diagrama muestra claramente los objetos que están conectados a un determinado objeto. Su debilidad es que no muestra el orden en el que se ejecutan los mensajes tan bien como el diagrama de secuencia. Elementos:

* Instancias de clases: Rectángulo con el nombre de la instancia subrayado.
* Mensajes: Los mensajes se muestran como flechas etiquetadas unidas a los enlaces. Cada mensaje tiene un número de secuencia, una lista opcional de mensajes precedentes, una condición opcional de guarda, un nombre, una lista de argumentos y un nombre de valor de retorno opcional.
* Flujos: Generalmente, un diagrama de comunicación contiene un símbolo para un objeto durante una operación completa.

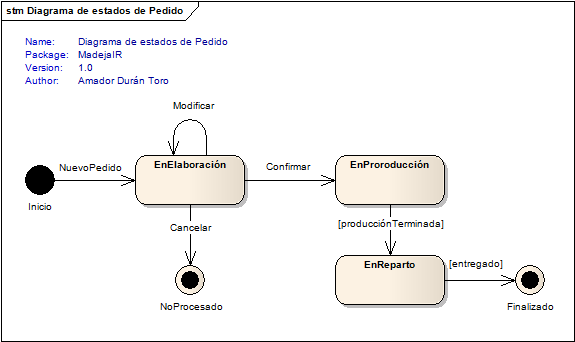


1. **DIAGRAMA DE ESTADO**

Es el diagrama que muestra a una máquina de estado, y son útiles para modelar la vida de un objeto

Es el que muestra el flujo de control entre estados (en que estados posibles puede estar algo y como se producen los cambios entre estos estados)

Una máquina de estado es un comportamiento que especifica las secuencias de estados por las que pasa un objeto a los largo de su vida en respuesta a eventos, junto con sus respuestas a esos eventos.



**CONCEPTOS CLAVES**

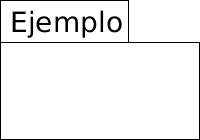
* ESTADO: un estado es una condición o situación en la vida de un objeto durante la cual satisface una condición, realiza alguna actividad o espera algún evento.
* EVENTO: es la especificación de un acontecimiento significativo que ocupa un lugar en el tiempo y en el espacio. Es la aparición de un estímulo que puede o no activar una transición de estado.
* TRANSICION: es una relación entre dos estados que indica que un objeto que está en el primer estado, realizará ciertas acciones y entrará en el segundo estado cuando ocurra un evento especificado y se satisfagan unas condiciones específicas.

Elementos:

* Círculo relleno, señalando al estado inicial.
* Círculo hueco que contiene un círculo más pequeño lleno, lo que indica el estado final (si existe)
* Rectángulo redondeado, lo que denota un estado. Parte superior del rectángulo contiene un nombre del estado. Puede contener una línea horizontal en el medio, por debajo del cual las actividades que se realizan en ese estado se indican
* Flecha, que denota la transición. El nombre del evento (si los hay) que causa esta transición etiqueta el cuerpo de la flecha. Una expresión de guardia puede añadirse antes un "/" y encerrado en corchetes cuadrados (eventName [guardExpression]), denotando que esta expresión debe ser cierto para la transición se lleve a cabo. Si se realiza una acción durante esta transición, que se añade a la siguiente etiqueta un "/" (eventName [guardExpression] / acción).
* Línea horizontal gruesa, ya sea con x> 1 líneas de entrada y 1 línea de salida o 1 línea de entrada y x> 1 líneas de salir. Estos denotan unirse / tenedor, respectivamente.

1. **DIAGRAMA DE PAQUETES**

Básicamente agrupan diferentes elementos de modelado en una entidad cohesiva. Estos por lo general se representan como una carpeta con el nombre de la misma en la pestaña.



Existen dos formas de escribir el nombre de lo que contiene el paquete, el simple, donde solo se coloca el nombre y el calificado, que muestra el nombre del paquete de la siguiente manera: Paquete::clase, donde "Paquete" el nombre del paquete que lo contiene y clase el elemento de modelado que contiene el paquete.

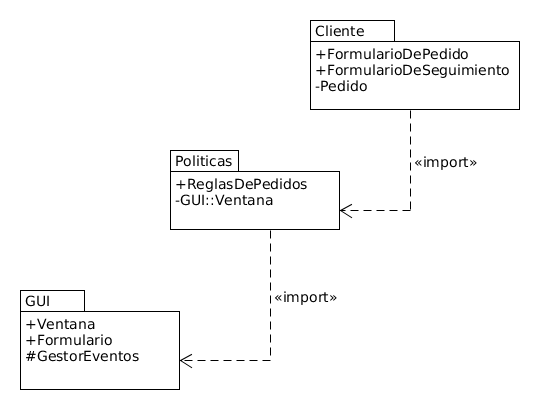
Cada paquete crea su propio espacio de nombre, es decir que puede haber dos o mas nombres iguales siempre que no estén en el mismo paquete; elementos de diferentes tipos pueden tener el mismo nombre pero no se recomienda.

Los paquetes pueden tener elementos tales como: clases, interfaces, componentes, nodos, colaboraciones, casos de uso, diagramas e incluso otro paquetes. Pero no es recomendable crear anidamientos muy profundos, como 2 o cuatro son suficientes

Al igual que las clases, se puede especificar la visibilidad de los elementos de un paquete. La visibilidad puede ser público, o privado colocando un signo "+", "-" o "#" a la izquierda respectivamente.

También existe el concepto de importación, que permite que si el paquete A importa al paquete B entonces el espacio de nombres de B se incluyen en A.

La importación se representa como una flecha punteada con la palabra <<import>> cerca de ella:



Elementos:

* Paquete: un mecanismo de propósito general para la organización de elementos y diagramas de modelos en grupos. Proporciona un espacio de nombres encapsulado dentro del cual todos los nombres deben ser únicos.
* Clase: una representación de un objeto que refleja su estructura y comportamiento dentro del sistema.
* Interfaz: una especificación de comportamiento. Una clase de implementación debe ser escrita para apoyar el comportamiento de una clase de interfaz.
* Objeto: una instancia de una clase. A menudo se utiliza en el análisis para representar un artefacto u otro elemento.
* Tabla: una clase estereotipada.

1. **DIAGRAMA DE ACTIVIDAD**

Los diagramas de actividad modelan los aspectos dinámicos de un proceso y especifican en que orden ocurren los hechos. Estos diagramas son muy parecidos a los diagramas de flujos pero con la capacidad de especificar otros comportamientos como concurrencia.

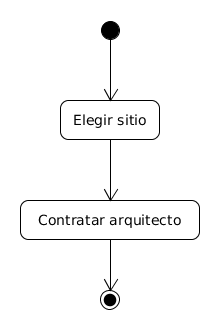
Estos contienen acciones, nodos de actividad, flujos. objetos valor:

Acciones: estos representan un paso que va a ser ejecutado en el proceso que se quiere modelar y se representa como un recuadro redondeado:

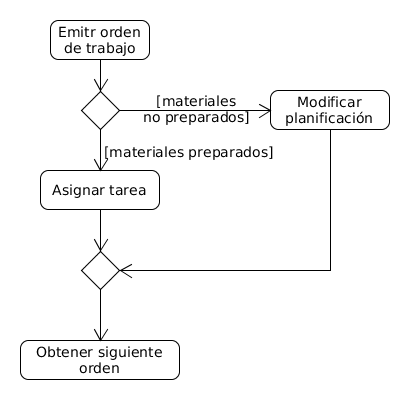


Nodos de actividad: Estos son elementos que encapsulan una serie de pasos en uno solo. A diferencia de las acciones, estos pueden ser divididos en otras acciones, es decir no son atómicos y se representan igual a las acciones excepto con paréntesis al final.

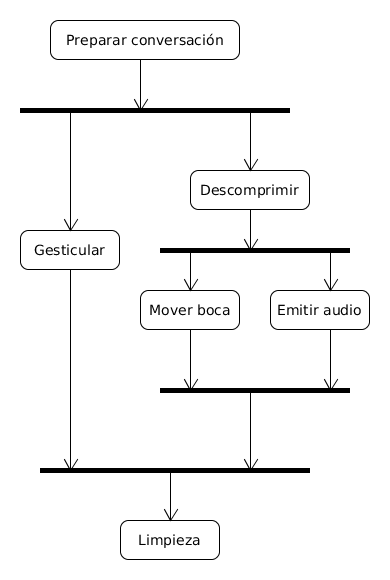
Flujos: Estos permiten describir el flujo u orden en que se ejecutan las acciones en el diagrama de actividad. Se describen con unas flechas y especifican cual acción ocurre después de otra. Para especificar el comienzo del proceso a modelar se coloca un punto relleno y se coloca una flecha saliendo de ella hacia la primera acción, y para especificar el fin se coloca un círculo con un círculo relleno dentro:



En estos diagramas se pueden tener bifurcaciones, es decir, puntos donde pueden haber flujos alternativos por recorrer. Las bifurcaciones se representan con un rombo y se debe colocar las condiciones de cada bifurcación cerca de las flechas de flujo. También se puede utilizar rombos como punto de unión de diferentes flujos:



Otra herramienta útil es la división y unión. La división representan momentos en donde dos o más flujos se ejecutan concurrentemente y la unión representa cuando se deben volver a sincronizar los flujos. Estos se representan con una barra vertical o horizontal.

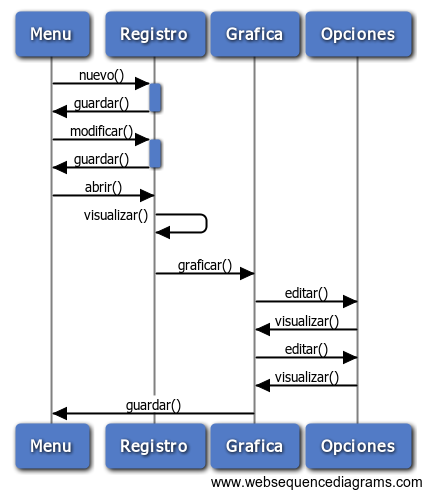


Flujos de objetos: especifican creación y manipulación de objetos en el diagrama de actividad. Estos se representan con recuadros, dentro se debe especificar el nombre del objeto y el estado entre corchetes.

Elementos del diagrama de actividades:

* **Círculo**: Procedimiento estandarizado.
* **Cuadrado**: Proceso de control.
* **Línea continua**: Flujo de información vía formulario o documentación en soporte de papel escrito.
* **Línea interrumpida**: Flujo de información vía formulario digital.
* **Rectángulo**: Formulario o documentación. Se grafica con un doble de ancho que su altura.
* **Rectángulo Pequeño**: Valor o medio de pago (cheque, pagaré, etc.). Se grafica con un cuádruple de ancho que su altura, siendo su ancho igual al de los formularios.
* **Triángulo (base inferior)**: Archivo definitivo.
* **Triángulo Invertido (base superior)**: Archivo Transitorio.
* **Semióvalo**: Demora.
* **Rombo**: División entre opciones.
* **Trapezoide**: Carga de datos al sistema.
* **Elipsoide**: Acceso por pantalla.
* **Hexágono**: Proceso no representado.
* **Pentágono**: Conector.
* **Cruz de Diagonales**: Destrucción de Formularios.

**Diagrama de secuencia:** es un tipo de diagrama usado para modelar interacción entre objetos en un sistema.



Objetos usados en el diagrama de secuencia:

* Mensaje: pasa de la línea de vida de un objeto a otro.
* Objeto: rectángulos con nombres subrayados, el tiempo se Representa como una progresión vertical.
* Línea de vida activa: el tiempo se representa en forma vertical inicia en la parte superior y avanza a la parte inferior un mensaje que este en la parte superior ocurrirá antes que uno en la parte inferior.

1. **CONCLUSIONES**:

Luis Correa

* En el diagrama de tiempo, un sistema secundario se activa como consecuencia de la activación de un sistema primario.

Ejemplo: El sistema secundario (Alarma y Aspersores) se activa una vez que se activa el sistema primario (Detector de Humo).

* En el diagrama de la visión global de interacciones: Tendremos un diagrama de secuencia de ofimática y otro de un software de almacenamiento en la nube (internet).

Para almacenar el software de ofimática en la nube se tendrá que cumplir con la condición de que haya iniciado sesión en el software de almacenamiento en la nube y luego se podrá enviar el archivo del diagrama de ofimática al diagrama de almacenamiento en la nube.

Orlando Medina

* Es fácil predecir que UML será el lenguaje de modelado de software de uso universal. Esto debido a que la mayoría de las empresas importantes de informática la han apoyado, y la han aceptado como un estándar.
* Con respecto a los diagramas, no es una actividad de dibujar grafos. Se trata de escribir con el detalle necesario, el flujo principal de un sistema y los flujos alternativos.
* El objetivo esencial es identificar los actores involucrados y a partir de sus objetivos, encontrar los casos de uso. Los diagramas de comunicación en especial, son una ayuda visual del comportamiento de un sistema.

Manuel Dun

* Los diagramas de paquetes son herramientas poderosas en proyectos de gran escala, logran simplificar el desarrollo de manera eficaz y son fáciles de entender. Además de esto, son simple de implementar y ayudan a disminuir la complejidad de proyectos de desarrollo de software.
* Los diagramas de despliegue se utilizan para documentar cómo el hardware y el software interactúan en el proyecto, especifican los diferentes elementos que se involucran y cómo se relacionan entre ellos.
* Los diagramas de actividad ayudan a modelar los comportamientos involucrados en algún proceso del proyecto a desarrollar. Estos diagramas son muy parecidos a los diagramas de flujos que todos conocemos, añadido a esto, se pueden utilizar tanto para modelar algoritmos específicos a implementar en un lenguaje de programación, como para contextos m[as generales que talvez no se vayan a implementar como un programa para ejecutarse.

**Nota: investigar sobre los elementos de los diagrama que están en el material**