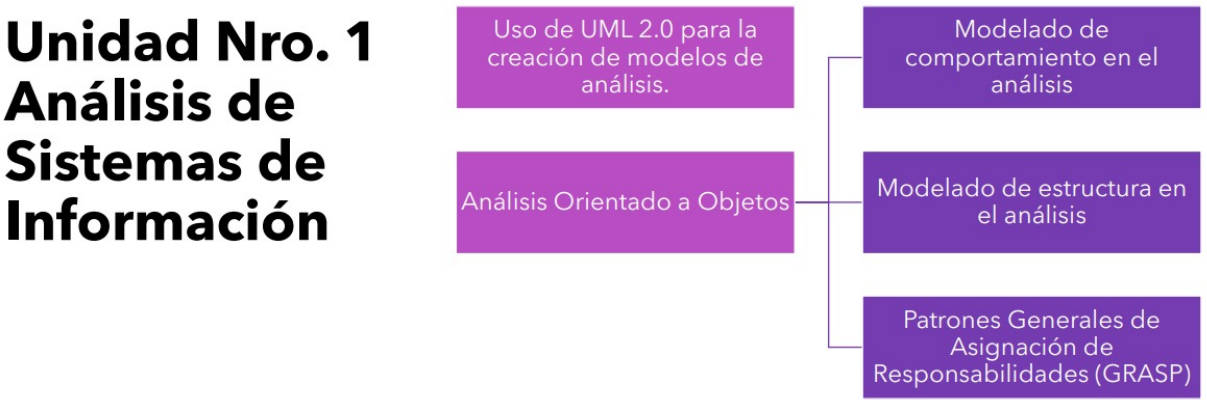
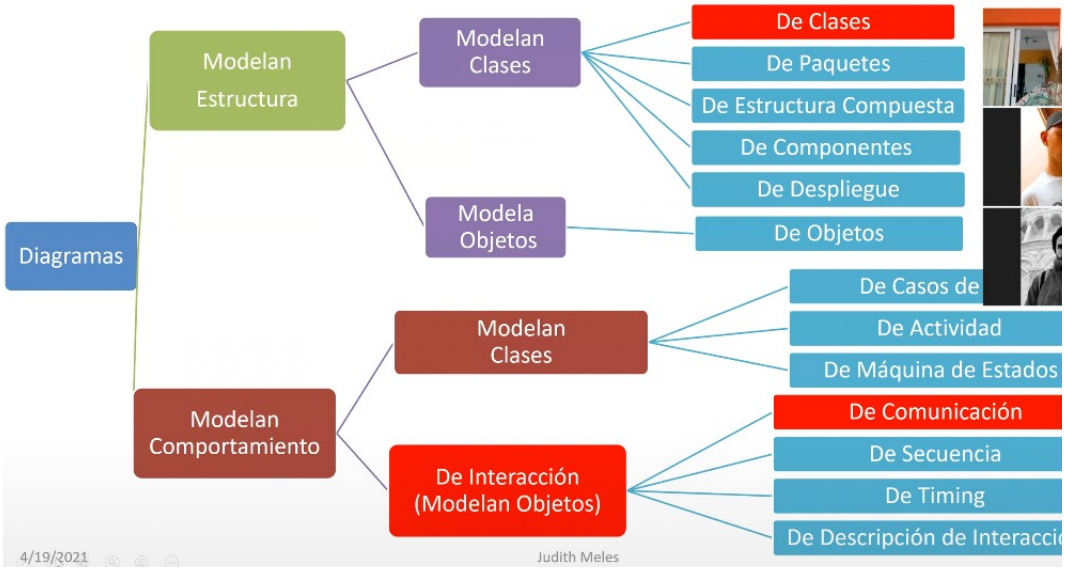
RESUMEN DISEÑO DE SISTEMAS DE INFORMACIÓN





¿QUÉ ES UML?

El Lenguaje Unificado de Modelado es un lenguaje de modelado visual para sistemas. UML se diseñó para incorporar las mejores prácticas en las técnicas de modelado y la ingeniería de software. Como tal, está explícitamente diseñado para implementarse por las herramientas de modelado UML.

Es importante apreciar que UML no nos proporciona ningún tipo de metodología de modelado. Naturalmente, algunos aspectos de la metodología se ven implicados por los elementos que componen el modelo UML, pero UML simplemente proporciona una sintaxis visual que podemos utilizar para construir modelos.

El Proceso Unificado (UP) es una metodología; nos dice los recursos humanos, las actividades y artefactos que necesitamos utilizar, desarrollar o crear para modelar un sistema de software.

UP utiliza UML como su sintaxis de modelado visual subyacente y por lo tanto se puede pensar a UP como el método preferido para UML, pero UML puede y proporciona el soporte de modelados visual para otros métodos.

El objetivo de UML y UP siempre ha sido dar soporte y encapsular las mejores prácticas en la ingeniería de software.

¿POR QUÉ UNIFICADO?

UML trata de estar unificado en diferentes dominios:

- Ciclo de vida de desarrollo: UML proporciona sintaxis visual para modelar directamente del ciclo de vida del desarrollo de software desde los requisitos a la implementación.
- Dominios de desarrollo: UML se ha utilizado para modelar todo tipo de sistemas.
- Lenguajes y plataformas de implementación: UML es neutro tanto en lenguaje como en plataforma. Dispone de un excelente soporte para lenguajes orientados a objetos, híbridos y no orientados a objetos.
- Procesos de desarrollo: UML puede soportar muchos otros procesos de ingeniería de software además del UP.
- Sus propios conceptos internos: UML trata de ser coherente y uniforme en su aplicación de un pequeño grupo de conceptos internos.

UML es un lenguaje para:

Un lenguaje de modelado es un lenguaje cuyo vocabulario y reglas se centran en la representación conceptual y física de un sistema. Un lenguaje de modelado como UML es, por lo tanto, un lenguaje estándar para los planos del software.
El modelado proporciona una comprensión de un sistema.
Para sistemas con gran cantidad de software, se requiere un lenguaje que abarque las diferentes vistas de la arquitectura de un sistema conforme evoluciona a través del ciclo de vida del desarrollo del software.
El vocabulario y las reglas de UML indican cómo crear y leer modelos bien formados pero no dicen qué modelos se deben crear ni cuándo se deberían crear. Esta es la tarea del proceso de desarrollo.

- Visualizar: UML es algo más que un simple montón de símbolos 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.
- Especificar: en este contexto, significa construir modelos precisos, no ambiguos y completos. En particular, UML cubre la especificación de todas las decisiones de análisis, diseño e implementación que deben realizarse al desarrollar y desplegar un sistema con gran cantidad de software.
- Construir: UML no es un lenguaje de programación visual pero sus modelos pueden conectarse a una gran variedad de lenguajes de programación. Las cosas que se expresan mejor gráficamente se representan mediante UML y las que se expresan mejor textualmente en código.

Esta correspondencia permite la ingeniería directa, es decir, la generación de código a partir de un modelo UML en un lenguaje de programación. Además, con ayuda de herramientas e intervención humana, se puede aplicar ingeniería inversa (reconstruir un modelo en UML a partir de una implementación).
La combinación de estas dos produce una ingeniería de ida y vuelta que permite trabajar en una vista textual o gráfica mientras las herramientas mantienen la consistencia entre ambas.

- Documentar: UML cubre la documentación de la arquitectura de un sistema y todos sus detalles. UML también proporciona un lenguaje para expresar requisitos y pruebas. Por último, UML proporciona un lenguaje para modelar las actividades de planificación de proyectos y gestión de versiones.

UN MODELO CONCEPTUAL DE UML
Está compuesto por:

A) Bloques básicos de UML

UML incluye tres clases de bloques básicos:

1. Elementos: son abstracciones que constituyen los ciudadanos de primera clase en un modelo
2. Relaciones: ligan los elementos entre sí
3. Diagramas: agrupan colecciones interesantes de elementos

Elementos en UML:

- Elementos Estructurales: son los nombres de los modelos UML. En su mayoría son las partes estáticas de un modelo, y representan conceptos o cosas materiales
  Dentro de estos elementos se encuentran:
  - Clase: es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Una clase implementa una o más interfaces.
  - Interfaz: es una colección de operaciones que especifican un servicio de una clase o componente. Por lo tanto, una interfaz describe el comportamiento visible externamente de ese elemento
  - Colaboración: define 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.
  - Caso de Uso: Es una descripción de un conjunto de secuencias de acciones que ejecuta un sistema y que produce un resultado observable de interés para un actor particular.

- Elementos de comportamiento: son las partes dinámicas de los modelos UML. Estos son los verbos y representan comportamiento en el tiempo y el espacio.
  Existen tres principales:
  - 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.

  - Máquina de estados: es un comportamiento que especifica 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.
  - Actividad: es un comportamiento que especifica la secuencia de pasos que ejecuta un proceso computacional. Un paso de una actividad se denomina acción.
- Elementos de agrupación: son las partes organizativas de los modelos UML. Estas son las cajas en las que se puede descomponer un modelo. El principal es:
  - Paquete: es un mecanismo de propósito general para organizar el propio diseño, en oposición a las clases, que organizan construcciones de implementación. Un paquete es puramente conceptual
- Elementos de anotación: son las partes explicativas de los modelos UML. Son los comentarios que se pueden aplicar para describir, clarificar y hacer observaciones sobre cualquier elemento de un modelo. El principal es:
  - Nota: es simplemente un símbolo para mostrar restricciones y comentarios junto a un elemento o una colección de elementos.

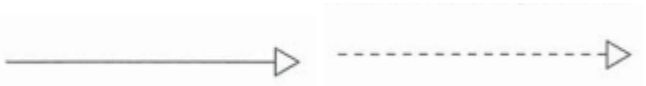
Relaciones en UML: Hay cuatro tipos de relaciones en UML

1. Dependencia: es una relación semántica entre dos elementos, en la cual un cambio a un elemento puede afectar a la semántica del otro elemento.

2. Asociación: es una relación estructural entre clases que describe un conjunto de enlaces, los cuales son conexiones entre objetos que son instancias de clases. La agregación es un tipo especial de asociación, que representa una relación estructural entre un todo y sus partes.

3. Generalización: es una relación de especialización/generalización en la cual el elemento especializado se basa en la especificación del elemento generalizado.

4. Realización: es una relación semántica entre clasificadores, en donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá.



Diagramas en UML: es la representación gráfica de un conjunto de elementos

1. **Diagrama de clases**: muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones.
2. **Diagrama de objetos**: muestra un conjunto de objetos y sus relaciones. Estos representan instantáneas estáticas de instancias de los elementos existentes en los diagramas de clases.

3. Diagrama de componentes
4. Diagrama de estructura compuesta
5. **Diagrama de casos de uso:** muestra un conjunto de casos de uso y actores y sus relaciones.
6. **Diagrama de secuencia**: tanto los diagramas de secuencia como los diagramas de comunicación son tipos de diagramas de interacción. Un diagrama de interacción muestra una interacción, que consta de un conjunto de objetos o roles, incluyendo los mensajes que pueden ser enviados entre ellos.
   Un diagrama de secuencia es un diagrama de interacción que resalta la ordenación temporal de los mensajes; un diagrama de comunicación es un diagrama de interacción que resalta la organización estructural de los objetos o roles que envían o reciben mensajes.
7. Diagrama de comunicación
8. **Diagrama de estados**: muestra una máquina de estados, que consta de estados, transiciones, eventos y actividades.
9. Diagrama de actividades: muestra la estructura de un proceso u otra computación como el flujo de control y datos paso a paso en la computación.
10. Diagrama de despliegue
11. Diagrama de paquetes
12. Diagrama de tiempos
13. Diagrama de visión global de interacciones

B) Reglas de UML

UML tiene un conjunto de reglas que especifican a qué debe parecerse a un modelo bien formado. Un modelo bien formado es aquel que es semánticamente autoconsistente y esta en armonía con todos sus modelos relacionados.

Posee reglas sintacticas y semanticas para:

- Nombres: como llamar a los elementos, relaciones y diagramas
- Alcance: el contexto que da un significado específico a un nombre
- Visibilidad: cómo se pueden ver utilizar esos nombre por otros
- Integridad: como se relacionan apropiada y consistentemente unos elementos con otros
- Ejecución: que significan ejecutar o simular un modelo dinámico

Debido a que los modelos construidos en el desarrollo tienden a evolucionar y a ser vistos por muchos usuarios, además de desarrollar modelos bien formados, deben ser:

- Abreviados: Ciertos elementos se ocultan para simplificar la vista
- Incompletos: Pueden estar ausentes ciertos elementos
- Inconsistentes: No se garantiza la integridad del modelo.

C) Mecanismos comunes en UML

- Especificaciones: UML es algo más que un lenguaje gráfico. Detrás de cada elemento de su notación gráfica hay una especificación que proporciona una explicación textual de la sintaxis y semántica de ese bloque de construcción. La especificación de UML se utiliza para expresar los detalles del sistema. Además

proporcionan una base semántica que incluye a todas las partes de todos los modelos de un sistema, y cada parte está relacionada con las otras de manera consistente.

- Adornos: La mayoría de los elementos de UML tienen una notación gráfica única y clara que proporciona una representación visual de los aspectos más importantes del elemento. Todos los elementos de UML parten de un símbolo básico, al cual pueden añadirse una variedad de adornos específicos de ese símbolo.
- Divisiones comunes:
    - Divisiones dicotómicas: en UMl se pueden tanto clases como objetos,CU y ejecuciones de CU, componentes e instancias de componentes, nodos e instancias de nodos,etc.
    - Separación entre interfaz e implementación: Una interfaz declara un contrato, y una implementación representa una realización concreta de ese contrato. UML permite modelar tanto interfaces como sus implementaciones: Por ejemplo: CU y las colaboraciones que los realizan, así como operaciones y los métodos que las implementan.
    - Separación entre tipo y rol: El tipo declara la clase de una entidad, como un objeto, atributo o parámetro. Un rol describe el significado de una entidad en un contexto, como una clase, componente o colaboración.

- Mecanismos de extensibilidad:UML es un lenguaje abierto-cerrado, siendo posible extender el lenguaje de manera controlada. Los mecanismos que utiliza son:
    - Estereotipos: extienden el lenguaje UML permitiendo crear nuevos tipos de bloques de construcción que derivan de los existentes pero que son específicos a un problema.
    - Valores etiquetados: extiende las propiedades de un estereotipo de UML permitiendo añadir nueva información en la especificación de ese estereotipo
    - Restricciones: extienden la semántica de un bloque de construcción de UML permitiendo añadir nuevas reglas o modificar las existentes

  Estos tres mecanismos permiten a UML adaptarse a nuevas tecnologías de software.

ARQUITECTURA

La arquitectura es el artefacto más importante que puede emplearse para controlar el desarrollo iterativo e incremental de un sistema a lo largo de su vida
Consiste en un conjunto de decisiones significativas sobre:

- La organización de un sistema de software
- La selección de elementos estructuras y sus interfaces a través de los cuales se constituye el sistema.
- Su comportamiento, como se especifican 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 de software también se relaciona con el uso, la funcionalidad, el rendimiento, la capacidad de adaptación, la reutilización, la capacidad de ser comprendido, etc.
La arquitectura de un sistema con gran cantidad de software puede describirse mejor a través de cinco vistas interrelacionan(cada vista es una proyección de la organización y la estructura del sistema, centrada en un aspecto particular del mismo)

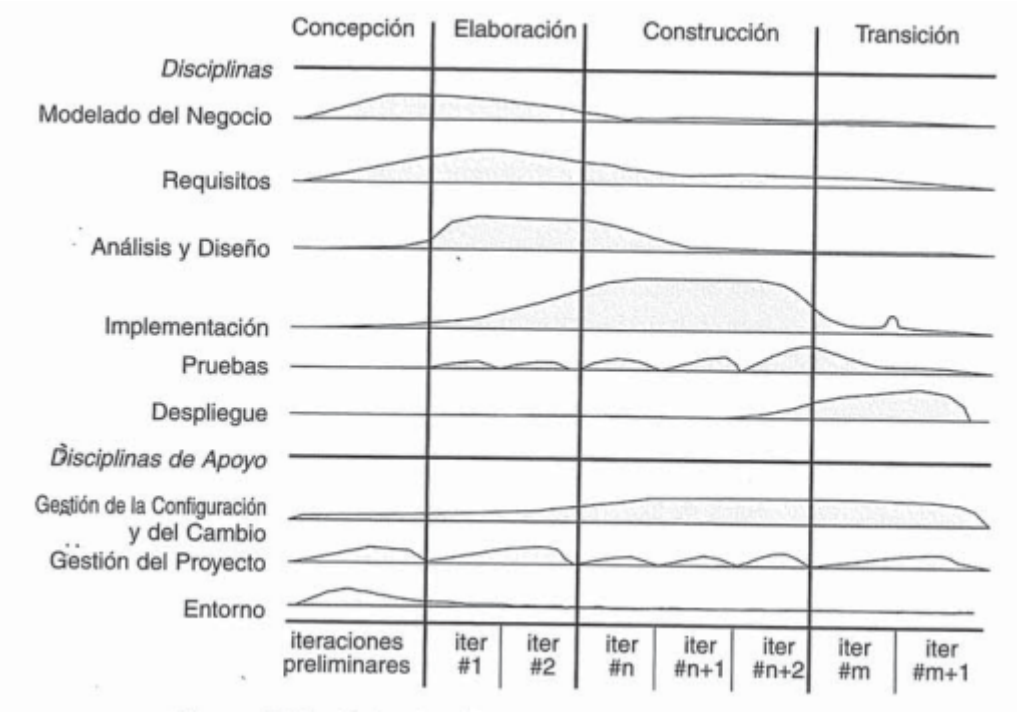
1. Vista de casos de uso: comprende los casos de uso que describen el comportamiento del sistema tal y como es percibido por los usuarios finales, analistas y encargados de pruebas. Aspectos estáticos: diagrama de CU. Aspectos dinámicos: diagrama de interacción, diagramas de estados y diagramas de actividades
2. Vista de diseño: comprende las clases, interfaces y colaboraciones que forman el vocabulario del problema y su solución. Aspectos estáticos: diagrama de clases. Aspectos dinámicos: diagramas de interacción, diagramas de estados y diagramas de actividades
3. Vista de interacción: muestra el flujo de control entre sus diversas partes, incluyendo los posibles mecanismo de concurrencia y sincronización. Aspectos estáticos y dinámicos: utilizan los mismo que en la vista de diseño respectivamente
4. Vista de implementación: comprende los artefactos que se utilizan para ensamblar y poner en producción el sistema físico. Aspectos estáticos: diagramas de artefactos. Aspectos dinámicos: diagramas de interacción, diagramas de estados y diagramas de actividades
5. Vista de despliegue:contiene los nodos que forman la topología hardware sobre la que se ejecuta el sistema. aspectos estáticos: diagramas de despliegue. Aspectos dinámicos: diagramas de interacción, diagramas de estados, diagramas de actividades.

CICLO DE VIDA DEL DESARROLLO DE SOFTWARE

UML no esta ligados ningún ciclo de vida de desarrollo de software en particular, pero para aprovecharlo al máximo se debe considerar un proceso que fuese:

- Dirigido por casos de uso: significa que los CU 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 conceptualizar, construir, gestionar y hacer evolucionar el sistema en desarrollo.
- Iterativo e incremental:
    - Iterativo: es aquel proceso que involucra la gestión de un flujo de versiones ejecutables.
    - Incremental: es aquel que implica la integración continua de la arquitectura del sistema para producir esas, donde cada nuevo ejecutable incorpora mejoras incrementales sobre los otros

Los ciclos de vida del desarrollo de software se dividen en cuatro fases:



- Concepción: es la primera fase del proceso, cuando la idea pra el desarrollo se lleva al punto de estar suficientemente bien fundamentada para garantizar la entrada en la fase de elaboración.
- 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 sólida base arquitectónica.
- 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.
- Transición: es la cuarta fase del proceso, cuando el software es puesto en las manos de la comunidad de usuarios

Nótese que todas estas fases componen una iteración en el proceso de desarrollo de software

"*Fin del desarrollo de la vista general de UML*"

ANÁLISIS

Durante el análisis, analizamos los requisitos que se describieron en la captura de requisitos, refinandolos y estructurandolos. El objetivo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que nos ayude a estructurar el sistema entero.

Además el propósito fundamental del análisis es resolver los temas que quedaron sin resolver, por causa de usar el lenguaje del cliente en la etapa de captura de requisitos, analizando los requisitos con mayor profundidad utilizando el lenguaje de los desarrolladores.

Agregado a esto, en el análisis podemos estructurar los requisitos de manera que nos facilite su comprensión, su preparación, su modificación y, en general, su mantenimiento. Esta estructura es independiente de la estructura que se dio a los requisitos. De todas maneras, existe una trazabilidad directa entre esas distintas estructuras, de forma que podemos hacer la traza de diferentes descripciones, en diferentes niveles de detalle, del mismo requisito y mantener su consistencia mutua con facilidad.

Modelo de casos de uso	Modelo de análisis
Descrito con el lenguaje del cliente.	Descrito con el lenguaje del desarrollador.
Vista externa del sistema.	Vista interna del sistema.
Estructurado por los casos de uso; proporciona la estructura a la vista externa.	Estructurado por clases y paquetes estereotipados; proporciona la estructura a la vista interna.
Utilizado fundamentalmente como contrato entre el cliente y los desarrolladores sobre qué debería y qué no debería hacer el sistema.	Utilizado fundamentalmente por los desarrolladores para comprender cómo debería darse forma al sistema, es decir, cómo debería ser diseñado e implementado.
Puede contener redundancias, inconsistencias, etc., entre requisitos.	No debería contener redundancias, inconsistencias, etc., entre requisitos.
Captura la funcionalidad del sistema, incluida la funcionalidad significativa para la arquitectura.	Esboza cómo llevar a cabo la funcionalidad dentro del sistema, incluida la funcionalidad significativa para la arquitectura; sirve como una primera aproximación al diseño.
Define casos de uso que se analizarán con más profundidad en el modelo de análisis.	Define realizaciones de casos de uso, y cada una de ellas representa el análisis de un caso de uso del modelo de casos de uso.

El lenguaje que utilizamos en el análisis se basa en un modelo de objetos conceptual, que llamamos modelo de análisis. El modelo de análisis nos ayuda a refinar los requisitos y nos permite razonar sobre los aspectos internos del sistema, incluidos sus recursos compartidos internos. Además, el modelo de análisis nos ofrece un mayor poder expresivo y una mayor formalización.

El modelo de análisis también nos ayuda a estructurar los requisitos y nos proporciona una estructura centrada en el mantenimiento, en aspectos tales como la flexibilidad ante los cambios y la reutilización. Esta estructura se utiliza además como entrada para las actividades de diseño e implementación.

POR QUE EL ANÁLISIS NO ES DISEÑO NI IMPLEMENTACIÓN

El diseño y la implementación son mucho más que el análisis, por lo que se requiere una separación de intereses. En el diseño, por ejemplo, debemos moldear el sistema y encontrar su forma, incluyendo su arquitectura, además se deben tomar decisiones relativas a cómo debería el sistema tratar los requisitos.
Decimos que el diseño y la implementación son mucho más que simplemente el analizar los requisitos refinandolos y estructurandolos: el diseño y la implementación se preocupan en realidad de dar forma al sistema de manera que de vida a todos los requisitos.

Dicho simplemente, llevando a cabo el análisis conseguimos una separación de interés que prepara y simplifica las subsiguientes actividades de diseño e implementación, delimitando los temas que deben resolverse y las decisiones que deben tomarse en esas actividades.

EL PAPEL DEL ANÁLISIS EN EL CICLO DE VIDA DEL SOFTWARE

El propósito del análisis debe alcanzarse de algún modo en todo proyecto. Pero la manera exacta de ver y de emplear el análisis puede diferir de un proyecto a otro, y nosotros distinguimos tres variantes básicas:

1. El proyecto utiliza el modelo de análisis para describir los resultados del análisis, y mantiene la consistencia de este modelo a lo largo de todo el ciclo de vida del software.
2. El proyecto utiliza el modelo de análisis para describir los resultados del análisis pero considera a este modelo como una herramienta transitoria e intermedia. Más adelante, cuando el diseño y la implementación están en marcha durante la fase de construcción, se deja de actualizar el análisis.
3. El proyecto no utiliza en absoluto el modelo de análisis para describir los resultados del análisis. En cambio, el proyecto analiza los requisitos como parte integrada en la captura de requisitos o en el diseño. En el primero de los casos, hará falta un mayor formalismo en el modelo de casos de uso. Esto puede ser justificable si el cliente es capaz de comprender los resultados, aunque creemos que rara vez se da el caso. El segundo caso podría complicar el trabajo de diseño. Sin embargo, esto puede ser justificable si los requisitos son muy simples y/o son bien conocidos, si es fácil de identificar la forma del sistema, etc.

Al elegir entre las dos primeras variantes, debemos sopesar las ventajas de mantener el modelo de análisis con el coste de mantenerlo durante varias iteraciones y generaciones. Por tanto, tenemos que realizar un análisis coste/beneficio correcto y decidir si deberíamos dejar de actualizar el modelo de análisis o si deberíamos conservarlo y mantenerlo durante el resto de la vida del sistema.

En cuanto a la tercera variante, en el proyecto no solo se puede evitar el coste de mantener el modelo de análisis, sino también el de introducirlo al principio. Sin embargo, normalmente las ventajas de trabajar con un modelo de análisis por lo menos al principio del proyecto sobrepasan los costes de emplearlo, por lo tanto, solo deberíamos emplear esta variante en casos excepcionales para sistemas extraordinariamente simples.

ARTEFACTOS

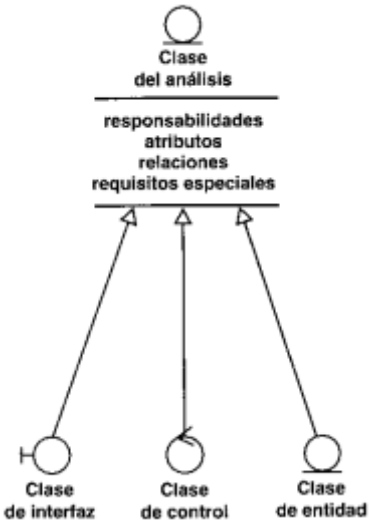
Modelo de análisis

El modelo de análisis es una jerarquía de paquetes del análisis que contienen clases del análisis y realizaciones de casos de uso.

Clase de análisis

Una clase de análisis representa una abstracción de una o varias clases del diseño del sistema. Poseen las siguientes características:

- Una clase de análisis se centra en el tratamiento de los requisitos funcionales y pospone los no funcionales
- Esto hace que una clase de análisis sea más evidente en el contexto del dominio del problema.
- Una clase de análisis raramente define un interfaz en términos de operaciones y de sus signaturas. En cambio, su comportamiento se define mediante responsabilidades en un nivel más alto y menos formal. Una responsabilidad es una descripción textual de un conjunto cohesivo del comportamiento de una clase.
- Una clase define atributos en un nivel bastante alto de abstracción los cuales son reconocibles en el dominio del problema
- Una clase de análisis participa en relaciones
- Las clases de análisis siempre encajan en uno de tres estereotipos: de interfaz, de control o de entidad



- Clases de interfaz: Se utilizan para modelar la interacción entre el sistema y sus actores. Esta interacción a menudo implica recibir información y peticiones de los usuarios y los sistemas externos.Las clases de interfaz modelan las partes del sistema que dependen de sus actores, lo cual implica que clarifican y reúnen los requisitos en los límites del sistema. Por ejemplo: ventanas, sensores, etc.
- Clases de control: Representan coordinación secuencia, transacciones, y control de otros objetos y se usan con frecuencia para encapsular el control de un caso de uso en concreto. También se utilizan para representar derivaciones y cálculos complejos, como la lógica del negocio, que no pueden asociarse con ninguna información concreta de larga duración, es decir a una clase entidad.
- Clases de identidad: Las clases de entidad se utilizan para modelar información que posee una vida larga y que es a menudo persistente. Modelan la información y el comportamiento asociado a un concepto por ejemplo una persona.
- Los aspectos dinámicos del sistema se modelan de las clases de control, debido a que ellas manejan y coordinan las acciones y los flujos de control principales, y delegan trabajo a otros objetos

Realización de caso de uso - Análisis

Una realización de caso de uso - análisis es una colaboración dentro del modelo de análisis que describe cómo se lleva a cabo y se ejecuta un caso de uso determinado en términos de las clases del análisis y de sus objetos del análisis en interacción.

Una realización de caso de uso posee una descripción textual del flujo de sucesos, diagramas de clases que muestran sus clases del análisis participantes, y diagramas de interacción que muestran la realización de un flujo o escenario particular del caso de uso en términos de interacciones de objetos del análisis.

- Diagrama de clases: una clase de análisis y sus objetos normalmente participan en varias realizaciones de casos de uso, y algunas de las responsabilidades, atributos, y asociaciones de una clase concreta suelen ser sólo relevantes para una única realización de caso de uso.
- Diagramas de interacción: la secuencia de acciones en un caso de uso comienza cuando un actor invoca el caso de uso mediante el envío de algún tipo de mensaje al sistema. En los diagramas de colaboración, mostramos las interacciones entre objetos creando enlaces entre ellos y añadiendo mensajes a esos enlaces. El nombre de un mensaje deberá denotar el propósito del objeto invocante en la interacción con el objeto invocado.
- Flujo de sucesos-análisis: los diagramas de una realización de caso de uso pueden ser difíciles de leer por sí mismos, de modo que puede ser útil un texto adicional que los explique. Este texto deberá escribirse en términos de objetos, particularmente objetos de control que interactúan para llevar a cabo el caso de uso.
- Requisitos especiales: son descripciones textuales que recogen todos los requisitos no funcionales sobre una realización de caso de uso. Algunos de estos ya se habían capturado en la etapa anterior o pueden ser nuevos.

Paquete del análisis

Los paquetes de análisis proporcionan un medio para organizar los artefactos del modelo de análisis en piezas manejables. Un paquete de análisis puede constar de clases de análisis, de realizaciones de casos de uso, y de otros paquetes de análisis.
Los paquetes de análisis deben ser cohesivos y deberían ser débilmente acoplados.

Características:

- Los paquetes del análisis pueden representar una separación de intereses de análisis.
- Los paquetes del análisis deberían crearse basándonos en los requisitos funcionales y en el dominio del problema, y deberían ser reconocibles por las personas con conocimiento del dominio. Los paquetes del análisis no deberían basarse en requisitos no funcionales o en el dominio de la solución.
- Los paquetes del análisis probablemente se convertirán en subsistemas en las dos capas de aplicación superiores del modelo de diseño, o se distribuirán entre ellos. En algunos casos, un paquete de análisis podría incluso reflejar una capa completa de primer nivel en el modelo de diseño.

- Paquetes de servicio: aparte de proporcionar casos de uso a sus actores, todo sistema también proporciona un conjunto de servicios a sus clientes. Un cliente adquiere una combinación adecuada de servicios para ofrecer a sus usuarios los casos de uso necesarios para llevar a cabo su negocio.
  En el Proceso Unificado, el concepto de servicio está soportado por los paquetes de servicio. Los paquetes de servicio se utilizan en un nivel más bajo de la jerarquía de paquetes de análisis para estructurar el sistema de acuerdo a los servicios que proporciona. Podemos observar lo siguiente acerca de los paquetes de servicio:
  - Un paquete de servicio contiene un conjunto de clases relacionadas funcionalmente.
  - Un paquete de servicio es indivisible. Cada cliente obtiene o bien todas las clases o bien ninguna del paquete de servicio.
  - Cuando se lleva a cabo un caso de uso, puede que sean participantes uno o más paquetes de servicio. Además, es frecuente que un paquete de servicio concreto participe en varias realizaciones de caso de uso diferentes.
  - Un paquete de servicio, depende a menudo de otro paquete de servicio.
  - Un paquete de servicio normalmente sólo es relevante para uno o unos pocos actores.
  - La funcionalidad definida por un paquete de servicio, cuando se diseña e implementa, puede gestionarse como una unidad de distribución separada. Un paquete de servicio puede, por tanto, representar cierta funcionalidad “adicional” del sistema. Cuando un paquete de servicio queda excluido, también lo queda todo caso de uso cuya realización requiera el paquete de servicio.
  - Los paquetes de servicio pueden ser mutuamente excluyentes, o pueden representar diferentes aspectos o variantes del mismo servicio.
  - Los paquetes de servicio constituyen una entrada fundamental para las actividades de diseño e implementación subsiguientes, dado que ayudarán a estructurar los modelos de diseño e implementación en términos de subsistemas de servicio. Los subsistemas de servicio tienen una influencia decisiva en la descomposición del sistema en componentes binarios y ejecutables.

  Los paquetes de servicio son reutilizables
  Aquellos paquetes de servicios cuyos servicios se centran alrededor de una o más clases de entidad son probablemente reutilizables en diferentes sistemas que soporten el mismo negocio o dominio. Esto es debido a que las clases de entidad se obtienen a partir de clases de entidad del negocio o de clases del dominio, lo cual hace a las clases entidad y sus servicios relacionados candidatos para la reutilización dentro del negocio o dominio entero y dentro de la mayoría de los sistemas que los soportan. Un paquete de servicio de este tipo es probable que se reutilice en varias realizaciones de caso de uso diferentes.

Descripción de la arquitectura

La descripción de la arquitectura contiene una vista de la arquitectura del modelo de análisis, que muestra sus artefactos significativos para la arquitectura.

Los siguientes artefactos del modelo de análisis se consideran significativos para la arquitectura:

- La descomposición del modelo de análisis en paquetes de análisis y sus dependencia
- Las clases fundamentales del análisis como las clases de entidad, las clases de interfaz, las clases de control y las clases de análisis. Suele ser suficiente con considerar significativa para la arquitectura una clase abstracta pero no sus subclases.
- Realizaciones de casos de uso que describen cierta funcionalidad importante y crítica que implican muchas clases de análisis y por lo tanto tienen una cobertura amplia, posiblemente a lo largo de varios paquetes de análisis; o se centran en un caso de uso importante que debe ser desarrollado al principio en el ciclo de vida del software, y por tanto es probable que se encuentre en la vista de la arquitectura del modelo de casos de uso.

DIAGRAMAS

Un diagrama es una representación gráfica de un conjunto de elementos y relaciones. Se utilizan para visualizar un sistema desde diferentes perspectivas

Los diagramas se dividen en dos:

DIAGRAMAS ESTRUCTURALES (VISTA ESTÁTICA)

Existen para visualizar, especificar, construir y documentar los aspectos estáticos de un sistema, es decir, aquellos que conforman el esqueleto del mismo. Por ejemplo: la existencia y ubicación de clases, interfaces, colaboraciones, componentes y nodos.

Los diagramas estructurales en UML son:

1) Diagrama de clases: Presenta un conjunto de clases, interfaces y colaboraciones, y las relaciones entre ellas.Se utilizan en los sistemas orientados a objetos y sirven para describir la vista estática de un sistema.

2) Diagrama de componentes: Muestra las partes internas, los conectores y los puertos que implementan un componente.Cuando se instancia un componente, también se instancian las copias de sus partes internas

3) Diagrama de estructura compuesta: Muestra la estructura interna de una clase o colaboración. La diferencia entre componentes y estructura compuesta es mínima.

4) Diagrama de objetos: Representa un conjunto de objetos y sus relaciones. Los diagramas de objetos abarcan la vista estática de un sistema al igual que los diagramas de clase, pero desde la perspectiva de casos reales o prototípicos.

5) Diagrama de artefactos: Muestra un conjunto de artefactos y sus relaciones con otros artefactos y con las clases a las que implementan. Se utilizan para mostrar las unidades físicas de implementación del sistema.

6) Diagrama de despliegue: Muestra un conjunto de nodos y sus relaciones. Los diagramas de despliegue se relacionan con los diagramas de componentes en que un nodo normalmente incluye uno o más componentes.

DIAGRAMAS DE COMPORTAMIENTO (VISTA DINÁMICA)

Los diagramas de comportamiento en UML se utilizan para visualizar, especificar, construir y documentar los aspectos dinámicos de un sistema, es decir, aquellos que representan partes mutables. Por ejemplo: el flujo de mensajes a lo largo del tiempo y el movimiento físico de componentes de red.

Los diagramas de comportamiento en UML son:

1) Diagrama de casos de uso: Representa un conjunto de casos de uso y actores, y sus relaciones. Los diagramas de CU se utilizan para describir la vista de casos de uso ESTÁTICA de un sistema. Los diagramas de casos de uso son especialmente importantes para organizar y modelar el comportamiento de un sistema.

“Tanto los diagramas de secuencia como los diagramas de comunicación son diagramas de interacción. Estos diagramas comparten el mismo modelos subyacente el dual es el diagrama de CU”

2) Diagrama de secuencia: es un diagrama de interacción que resalta la ordenación temporal de los mensajes. Presenta un conjunto de roles y mensajes enviados y recibidos por las instancias que interpretan los roles. Se utilizan para definir la vista dinámica de un sistema.

3) Diagrama de comunicación: Es un diagrama de interacción que resalta la organización estructural de los objetos que envían y reciben mensajes. Muestra un conjunto de roles, enlaces entre ellos y los mensajes enviados y recibidos por las instancias que interpretan estilos roles. Se utilizan para describir la vista dinámica de un sistema

4) Diagrama de estados: Representa una máquina de estados, constituida por estados, transiciones, eventos y actividades. Modelan el comportamiento de una interfaz, clase o colaboración. Se utilizan para describir la vista dinámica de un sistema

5) Diagrama de actividades: Muestra un flujo paso a paso en una computación. Una actividad muestra un conjunto de acciones, el flujo secuencial o ramificado de acción en acción, y los valores que son producidos o consumidos por las acciones. Se utilizan para describir la vista dinámica de un sistema.

A la hora de modelar un sistema desde diferentes vistas es necesario:

- Decidir qué vistas se necesitan para expresar la arquitectura del sistema de identificar los riesgos técnicos del proyecto
- Definir para cada vista que artefactos hay que crear para capturar los detalles esenciales.
- Decidir cuáles de los diagramas se pondrán bajo algún tipo de control formal o semiformal, es decir, de cuales se guardaran como documentación.
- Tener en cuenta que habrá diagramas que se desecharan.

MODELADO A DIFERENTES NIVELES DE ABSTRACCIÓN

Debido a las distintas personas implicadas en el desarrollo, puede darse que muchas de ellas necesitan la misma vista del sistema pero a distintos niveles de abstracción.
Hay dos formas de modelar un sistema a diferentes niveles de abstracción: presentando diagrams con diferentes niveles de detalle para el mismo modelo o creando modelos a diferentes niveles de abstracción con diagramas que se relacionan de un modelo a otro.

Para modelar un sistema a diferentes niveles de abstracción presentando diagramas más con diferentes niveles de detalle:

- Hay que considerar las necesidades de las personas que utilizaran el diagrama
- Si se va a usar un modelo para implantación, debe ser de baja abstracción con muchos detalles. En cambio si se va a usar para presentar un modelo a un usuario final deberá ser de mayor abstracción ocultando muchos detalles.
- Dependiendo de en qué nivel de abstracción se encuentre se deberá ocultar o revelar más o menos las siguiente cuatro categorías:
    - Bloques de construcción y relaciones
    - Adornos
    - Flujo
    - Estereotipos

La ventaja de este enfoque es que siempre se está modelando desde un repositorio semántico común. La principal desventaja es que los cambios de los diagramas a un nivel de abstracción pueden dejar obsoletos a los diagramas a otros niveles de abstracción
Para modelar un sistema diferentes niveles de abstracción mediante la creación de modelos a diferentes niveles de abstracción es necesario:

- Considerar las necesidades de las personas que utilizaran el diagrama y decidir el nivel de abstracción que deberá ver cada una, creando modelos separados por cada nivel.
- Poblar los modelos de mayor nivel con abstracciones simples y los de bajo nivel con abstracciones detalladas. Establecer dependencias de traza entre los elementos relacionados de diferentes modelos.
- Si se siguen las cinco vistas de una arquitectura, al modelar un sistema a diferentes niveles de abstracciones se producen cuatro situaciones con frecuencia:
    - Casos de uso y su realización: los CU en un modelo de CU se corresponde con colaboraciones en un modelo de diseño
    - Colaboraciones y su realización: las colaboraciones corresponden con una sociedad de clases que trabajan juntas para llevar a cabo la colaboración

- Componentes y su diseño: Los componentes en un modelo de implementación se corresponde con los elementos en un modelo de diseño
- Nodos y sus componentes: Los nodos en un modelo de despliegue se corresponden con componentes en un modelo de implementación

La ventaja de este enfoque es que los diagramas a diferentes niveles de abstracción se mantienen poco acoplados. La desventaja principal es que se deben gastar recursos para mantener sincronizados estos modelos y sus diagramas

MODELADO DE VISTA COMPLEJAS

Para modelar vistas complejas, con diagramas grandes y complejos:

- Primero, hay que asegurarse de que no existe una forma significativa de presentar esta información a mayor nivel de abstracción.
- Si se han omitido tantos detalles como es posible y el diagrama es aún complejo, hay que considerar la agrupación de algunos elementos en paquetes o en colaboraciones de mayor nivel, y luego mostrar sólo esos paquetes o colaboraciones en el diagrama.
- Si el diagrama es aún complejo, hay que usar notas de colores como señales visuales para resaltar los puntos de interés.
- Si el diagrama aún es complejo, hay que imprimirlo completamente y colgarlo en una pared.

DIAGRAMA DE CLASES

Un diagrama de clases muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones.
Los diagramas de clases se utilizan para modelar la vista de diseño estática de un sistema. Esto incluye, principalmente, modelar el vocabulario del sistema, modelar las colaboraciones o modelar esquemas.Son importantes no solo para visualizar, especificar y documentar modelos estructurales, sino también para construir sistemas ejecutables.

PROPIEDADES COMUNES

Un diagrama de clases es un tipo especial de diagrama y comparte las propiedades comunes al resto de los diagramas. Lo que distingue a un diagrama de clases de los otros tipos de diagramas es su contenido particular.

CONTENIDO

Los diagramas de clases contienen normalmente los siguientes elementos:

- Clases
- Interfaces
- Relaciones de dependencia, generalización y asociación

USOS COMUNES

Los diagramas de clases se utilizan para modelar la vista de diseño estática de un sistema. Esta vista soporta principalmente los requisitos funcionales de un sistema, los servicios que el sistema debe proporcionar a sus usuarios finales.

Los diagramas de clases se utilizan de 3 maneras:

1. Para modelar el vocabulario de un sistema: implica tomar decisiones sobre qué abstracciones son parte del sistema en consideración y cuáles caen fuera de sus límites. Los diagramas de clases se utilizan para especificar estas abstracciones y sus responsabilidades.
2. Para modelar colaboraciones simples: una colaboración es una sociedad de clases, interfaces y otros elementos que colaboran para proporcionar un comportamiento cooperativo mayor que la suma de todos los elementos. Los diagramas de clases se emplean para visualizar y especificar este conjunto de clases y sus relaciones.
3. Para modelar un esquema lógico de base de datos: se pueden modelar esquemas para bases de datos mediante diagramas de clases.

TÉCNICAS COMUNES DE MODELADO

- Modelado de colaboraciones simples

  Ninguna clase se encuentra aislada. En cambio, cada una trabaja en colaboración con otras para llevar a cabo alguna semántica mayor que la asociada a cada clase individual. Estas colaboraciones se representan con los diagramas de clases.

  Para modelar una colaboración:
  - Hay que identificar los mecanismos que se quieren modelar. Un mecanismo representa una función o comportamiento de la parte del sistema que se está modelando que resulta de la interacción de una sociedad de clases, interfaces y otros elementos.
  - Para cada mecanismo, hay que identificar las clases, interfaces y otras colaboraciones que participan en esta colaboración.
  - Hay que usar escenarios para recorrer la interacción entre estos elementos.
  - Hay que asegurarse de rellenar estos elementos con su contenido.

- Modelado de un esquema lógico de base de datos

  Muchos de los sistemas que se modelen tendrán objetos persistentes, lo que significa que estos objetos podrán ser almacenados en una base de datos con el fin de poder recuperarlos posteriormente.
  Los diagramas de clases de UML son un superconjunto de los diagramas entidad-relación. Mientras que los diagramas E-R clásicos se centran solo en los datos, los diagramas de clases van un paso más allá, ya que permiten el modelado del comportamiento.

  Para modelar un esquema:
  - Hay que identificar aquellas clases del modelo cuyo estado debe trascender el tiempo de vida de las aplicaciones.

- Hay que crear un diagrama de clases que contenga estas clases.
- Hay que expandir los detalles estructurales de estas clases.
- Hay que buscar patrones comunes que complican el diseño físico de bases de datos, tales como asociaciones cíclicas y asociaciones uno a uno.
- Hay que considerar también el comportamiento de las clases persistentes expandiendo las operaciones que sean importantes para el acceso a los datos y la integridad de estos.
- Donde sea posible, hay que usar herramientas que ayuden a transformar un diseño lógico en un diseño físico.

- Ingeniería directa e inversa

  En la mayoría de los casos los modelos creados se corresponden con un código.

  La ingeniería directa es el proceso de transformar un modelo en código a través de una correspondencia con un lenguaje de implementación. La ingeniería directa produce pérdida de información porque los modelos escritos en UML son semánticamente más ricos que cualquier lenguaje de programación orientado a objetos actual.

  Para hacer ingeniería directa con un diagrama de clases:
  - Hay que identificar las reglas para la correspondencia al lenguaje o lenguajes de implementación elegidos.
  - Según la semántica de los lenguajes escogidos, quizás haya que restringir el uso de ciertas características de UML.
  - Hay que usar valores etiquetados para guiar las decisiones de implementación en el lenguaje destino. Esto se puede hacer a nivel de clases individuales si es necesario un control más preciso.
  - Hay que usar herramientas para generar código.

  La ingeniería inversa es el proceso de transformar código en un modelo a través de una correspondencia con un lenguaje de programación específico. La ingeniería inversa produce un aluvión de información, parte de la cual está a un nivel de detalle más bajo del que se necesita para construir modelos útiles. Al mismo tiempo, la ingeniería inversa es incompleta debido a que cuando se hace ingeniería directa se pierde información por lo que no se puede recrear un modelo completo a partir de código.

  Para hacer ingeniería inversa sobre un diagrama de clases:
  - Hay que identificar las reglas para la correspondencia desde el lenguaje o lenguajes de implementación elegidos.
  - Con una herramienta, hay que indicar el código sobre el que se desea aplicar ingeniería inversa.
  - Con la herramienta, hay que crear un diagrama de clases inspeccionando el modelo.
  - La información de diseño se puede añadir manualmente al modelo, para expresar el objetivo del diseño que no se encuentra o está oculto en el código.

DIAGRAMAS DE INTERACCIÓN

Los diagramas de secuencia y los diagramas de comunicación son dos de los tipos de diagramas de UML que se utilizan para modelar los aspectos dinámicos de los sistemas.
Un diagrama de interacción muestra una interacción, que consiste en un conjunto de objetos y sus relaciones, incluyendo los mensajes que se pueden enviar entre ellos.
Un diagrama de secuencia es un diagrama de interacción que destaca la ordenación temporal de los mensajes; un diagrama de comunicación es un diagrama de interacción que destaca la organización estructural de los objetos que envían y reciben mensajes.

CONTENIDOS

Normalmente, los diagramas de interacción contienen:

- Roles u objetos
- Comunicaciones o enlaces
- Mensajes

DIAGRAMAS DE SECUENCIA

Los diagramas de secuencia tienen dos características que los distinguen de los diagramas de comunicación.

En primer lugar está la línea de vida la cual es una línea discontinua vertical que representa la existencia de un objeto a lo largo de un periodo de tiempo. La mayoría de los objetos que aparecen en un diagrama de interacción existirán mientras dure la interacción, así que los objetos se colocan en la parte superior del diagrama, con sus líneas de vida dibujadas desde arriba hasta abajo.
Pueden crearse objetos durante la interacción. Sus líneas de vida comienzan con la recepción del mensaje estereotipado como create. Los objetos pueden destruirse durante la interacción. Sus líneas de vida acaban con la recepción del mensaje estereotipado como destroy.
En segundo lugar, está el foco de control. El foco de control es un rectángulo delgado y estrecho que representa el periodo de tiempo durante el cual un objeto ejecuta una acción, bien sea directamente o a través de un procedimiento subordinado.
La parte superior del rectángulo se alinea con el comienzo de la acción; la inferior se alinea con su terminación. Si se quiere ser preciso acerca de donde se encuentra el foco de control, también se puede sombrear la región del rectángulo durante la cual el método del objeto está ejecutándose.

El contenido principal de los diagramas de secuencia son los mensajes. Un mensaje se representa con una flecha que va de una línea de vida hasta otra. La flecha apunta al receptor.

- Si el mensaje es asíncrono, la flecha es abierta.
- Si el mensaje es síncrono, la flecha es un triángulo relleno.
- Una respuesta a un mensaje síncrono se representa con una flecha abierta discontinua.

El orden del tiempo a lo largo de una línea de vida es significativo. El conjunto de mensajes en líneas de vida separadas forma un orden parcial.

CONTROL ESTRUCTURADO EN LOS DIAGRAMAS DE SECUENCIA

Una secuencia de mensajes está bien para mostrar una secuencia sencilla y lineal, pero a menudo necesitamos mostrar condicionales y bucles. Este tipo de control de alto nivel puede mostrarse mediante operadores de control estructurados en los diagramas de secuencia.

- Ejecución opcional: la etiqueta es opt. El cuerpo del operador de control se ejecuta si una condición de guarda es cierta cuando se entra en el operador. La condición de guarda es una expresión booleana que puede aparecer entre corchetes encima de cualquier línea de vida dentro del cuerpo y puede referenciar a los atributos del objeto.
- Ejecución condicional: la etiqueta es alt. El cuerpo del operador de control se divide en varias subregiones con líneas discontinuas horizontales. Cada subregión representa una rama de la condición. Cada subregión tiene una condición de guarda. De todas formas, solo se ejecuta una subregión como máximo; si son ciertas más de una condición de guarda, la elección de la subregión no es determinista y podría variar de ejecución en ejecución. Puede haber una subregión con una condición de guarda especial (else); esta subregión se ejecuta si no es cierta ninguna de las otras condiciones de guarda.
- Ejecución paralela: la etiqueta es par. El cuerpo del operador de control se divide en varias subregiones con líneas discontinuas horizontales. Cada subregión representa una computación paralela o concurrente. En la mayoría de los casos, cada subregión implica diferentes líneas de vida. Cuando se entra en el operador de control, todas las subregiones se ejecutan concurrentemente. La ejecución de mensajes en cada subregión es secuencial, pero el orden relativo de los mensajes en las subregiones paralelas es completamente arbitrario.
- Ejecución en bucle (iterativa): la etiqueta es loop. Una condición de guarda aparece sobre una línea de vida dentro del cuerpo. El cuerpo del bucle se ejecuta repetidamente mientras la condición de guarda sea cierta antes de cada iteración.

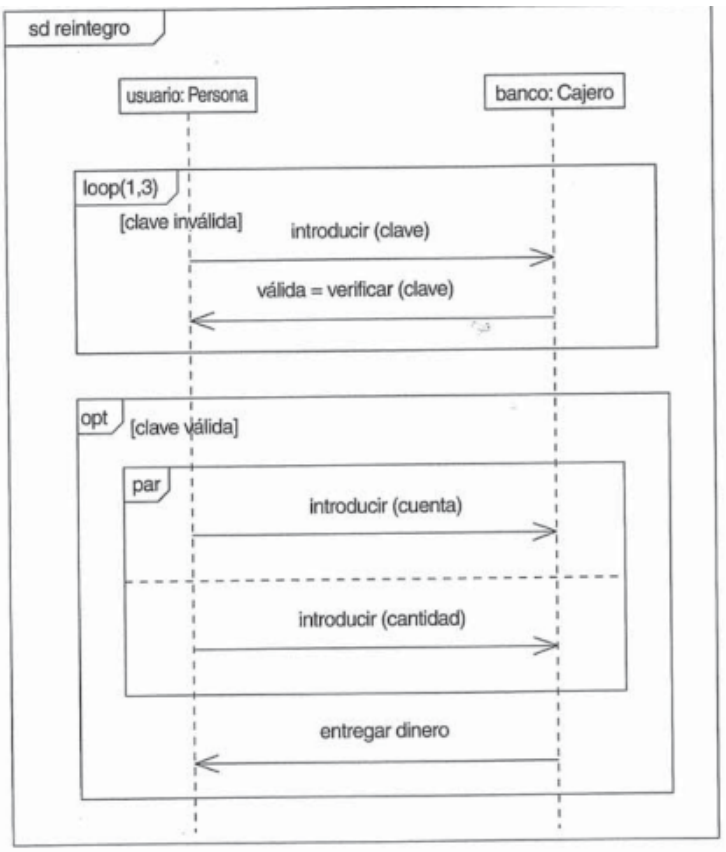
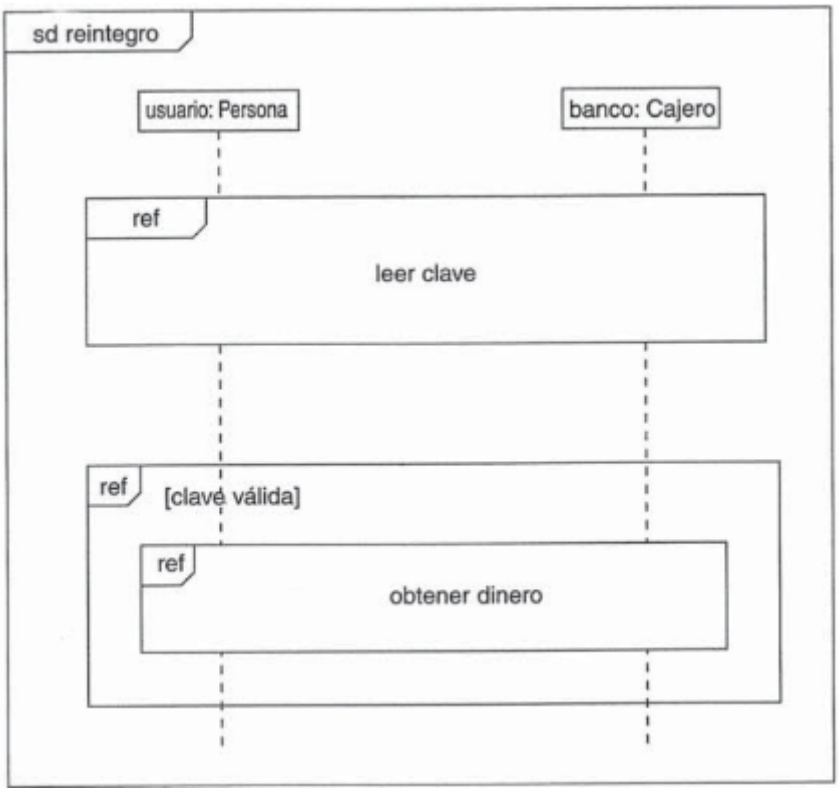


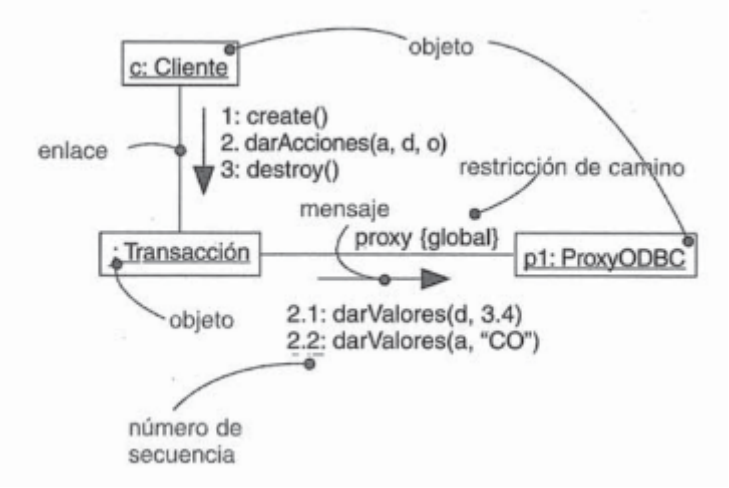
DIAGRAMA DE ACTIVIDAD ANIDADOS

Los diagramas de actividad que son demasiado grandes pueden ser difíciles de comprender. Las secciones estructuradas de una actividad pueden organizarse en una actividad subordinada, especialmente si la actividad subordinada se utiliza más de una vez dentro de la actividad principal. La actividad principal y las subordinadas se representan en diagramas diferentes.



DIAGRAMAS DE COMUNICACIÓN

Un diagrama de comunicación destaca la organización de los objetos que participan en una interacción. Un diagrama de comunicación se construye colocando en primer lugar los objetos que participan en la colaboración como nodos del grafo. Luego se representan los enlaces que conectan esos objetos como arcos del grafo que pueden tener nombres de roles para identificarlos. Por último, los enlaces se adornan con los mensajes que los objetos envían y reciben.



Los diagramas de comunicación tienen dos características que los distinguen de los diagramas de secuencia.

En primer lugar, el camino, el cual se dibuja haciéndolo corresponder con una asociación. También se dibujan caminos haciéndolos corresponder con variables locales, parámetros, variables globales o accesos propios. Un camino representa una fuente de conocimiento de un objeto.

En segundo lugar está el número de secuencia, que se utiliza para la ordenación temporal de un mensaje, se precede de un número, que se incrementa secuencialmente por cada nuevo mensaje en el flujo de control. Para representar el anidamiento, se utiliza la numeración decimal de Dewey (1 es el primer mensaje; 1.1 es el primer mensaje dentro del mensajes, etc.). El anidamiento se puede representar a cualquier nivel de profundidad.

La mayoría de las veces se modelan flujos de control simples y secuenciales. Sin embargo, también se pueden modelar flujos más complejos, que impliquen iteracion y bifurcación. Una iteración representa una secuencia repetida de mensajes, cada iteración tiene un número de secuencia diferente. Por otro lado, en una bifurcación los caminos alternativos tendrán el mismo número de secuencia, pero cada camino debe ser distinguible de forma única por una condición que no se solape con las otras.

EQUIVALENCIA SEMÁNTICA

Los diagramas de secuencia y de comunicación son semánticamente equivalentes, ya que ambos derivan de la misma información del metamodelo de UML. Como consecuencia de esto, se puede partir de un diagrama en una forma y convertirlo a la otra sin pérdida de información.

USOS COMUNES

Cuando se modelan los aspectos dinámicos de un sistema, normalmente se utilizan los diagramas de interacción de dos formas.

1. Para modelar flujos de control por ordenación temporal: Para ello se utilizan los diagramas de secuencia. El modelado de un flujo de control por ordenación temporal destaca el paso de mensajes tal y como se desarrolla a lo largo del tiempo, lo que es una forma útil de visualizar el comportamiento dinámico en el contexto de un escenario de un caso de uso.
2. Para modelar flujos de control por organización: para ello se utilizan los diagramas de comunicación. El modelado de un flujo de control por organización destaca las relaciones estructurales entre las instancias de la interacción, a través de las cuales pasan los mensajes.

TÉCNICAS COMUNES DE MODELADO

Modelado de flujos de control por ordenación temporal

Para modelar un flujo de control por ordenación temporal:

- Hay que establecer el contexto de la interacción, bien sea un sistema, un subsistema, una operación, o una clase, o bien un escenario de un caso de uso o de una colaboración.

- Hay que establecer el escenario de la interacción, identificando qué objetos juegan un rol en ella.
- Hay que establecer la línea de vida de cada objeto.
- A partir del mensaje que inicia la interacción, hay que ir colocando los mensajes subsiguientes de arriba abajo entre las líneas de vida.
- Si es necesario visualizar el anidamiento de mensajes o el intervalo de tiempo durante el cual tiene lugar la computación.
- Si es necesario especificar restricciones de tiempo o espacio, hay que adornar cada mensaje con una marca de tiempo y asociar las restricciones.
- Si es necesario especificar este flujo de control más formalmente, hay que asociar pre y poscondiciones a cada mensaje.

Modelado de flujos de control por organización

Para modelar el flujo de control por organización:

- Hay que establecer el contexto de la interacción, bien sea un sistema, un subsistema, una operación, o una clase, o bien un escenario de un caso de uso o de una colaboración.
- Hay que establecer el escenario de la interacción, identificando qué objetos juegan un rol en ella.
- Hay que especificar los enlaces entre esos objetos, junto a los mensajes que pueden pensar.
    1. Colocar los enlaces de asociaciones en primer lugar.
    2. Colocar los demás enlaces a continuación, y adornarlos con las anotaciones de camino adecuadas.
- Comenzando por el mensaje que inicia la interacción, hay que asociar cada mensaje subsiguiente al enlace apropiado, estableciendo su número de secuencia.
- Si es necesario especificar restricciones de tiempo o espacio, hay que adornar cada mensaje con una marca de tiempo y asociar las restricciones adecuadas.
- Si es necesario especificar el flujo de control más formalmente, hay que asociar pre y poscondiciones a cada mensaje.

<u>MÁQUINA DE ESTADO</u>

Una máquina de estados es un comportamiento que especifica las secuencias de estados por las que pasa un objeto durante su vida, en respuesta a eventos, junto con sus respuestas a esos eventos. Se utilizan para modelar los aspectos dinámicos de un sistema.

COMPONENTES

Estado:
El estado de un objeto es un periodo de tiempo durante el cual satisface alguna condición, realiza alguna actividad o espera algún evento. Un objeto permanece en un estado una cantidad de tiempo finito
Un estado esta compuesto por:

- Nombre
- Efectos de entrada/Salida: acciones ejecutadas al entrar y salir del estado
- Transiciones internas: transiciones que no generan un cambio de estado

- Subestados: estructura anidada de un estado
- Eventos diferidos: Una lista de eventos que no se manejan en este estado sino que se posponen y se añaden a una cola para ser manejados por el objeto en otro estado.

Estado inicial: Indica el punto de comienzo por defecto para la máquina de estados o el subestado.
Estado final: indica que la ejecución de la máquina de estados ha finalizado.

Transiciones:
Una transición es una relación entre dos estados que indica que un objeto que esté en el primer estado realiza ciertas acciones y entrará en el segundo estado cuando ocurra un evento especificado y se satisfagan unas condiciones específicas
Una transición posee:

- Estado de origen: Es el estado afectado por la transición
- Evento disparador: El evento cuya recepción por el objeto que está en el estado origen provoca el disparo de la transición si se satisface su condición
- Condición de guarda: Una expresión booleana que se evalúa cuando la transición se activa por la recepción de un evento disparador
- Efecto: Un comportamiento ejecutable
- Estado destino: El estado activo tras completarse la transición

Otros conceptos:

- Un efecto es la especificación de la ejecución de un comportamiento dentro de una máquina de estados.
- Un evento es la especificación de un acontecimiento significativo situado en el tiempo y en el espacio.
- Una actividad-Do es una ejecución no atómica en curso, dentro de una máquina de estados. Esta actividad se puede interrumpir por la recepción de un evento que cause una transición que fuerza un cambio de estado.
- Una acción es una computación atómica ejecutable que produce un cambio de estado del modelo o devuelve un valor
- La dinámica de ejecución se puede ver de dos formas: con diagramas de actividades o con diagramas de estados.
- Evento diferido: es una lista de eventos cuya aparición en el estado se pospone hasta que se activa otro estado; si los eventos no han sido diferidos en ese estado, se maneja el evento y se pueden disparar transiciones como si realmente acabaran de producirse
- Submaquinas: Una máquina de estado puede ser referenciada desde dentro de otra máquina. Una máquina de estados así referenciada se denomina submáquina . Son útiles para construir grandes modelos de estados.

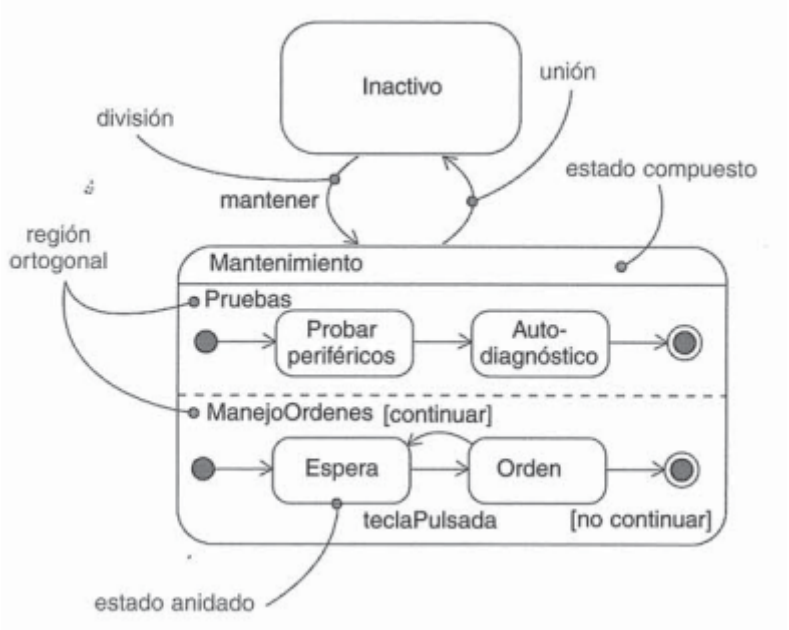
SUBESTADOS
Un subestado es un estado anidado dentro de otro.
Un estado con subestados dentro, se denomina compuesto.
Un estado compuesto puede tener subestados concurrentes(ortogonales) o secuenciales (no ortogonales).

Subestados no Ortogonales:

Los subestados como *validación* y *procesamiento*, se llaman subestados no ortogonales o disjuntos. Dado un conjunto de estados no ortogonales en el contexto de un estado compuesto, se dice que el objeto está en el estado compuesto, y solo en uno de sus subestados (o en el final) en un momento dado. Por lo tanto, los subestados no ortogonales particionan el espacio de estado del estado compuesto en estados disjuntos.

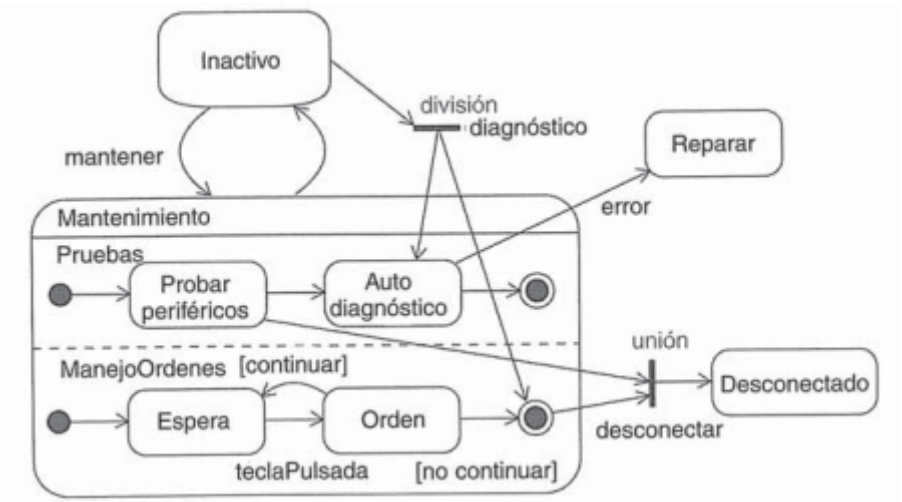
Estados de historia:
Un estado de historia permite que un estado compuesto que contiene subestados secuenciales recuerde el último subestado activo antes de la transición que provocó la salida del estado compuesto

Subestados Ortogonales:
Los estados ortogonales permiten dividir el espacio del estado en regiones ortogonales. Estas regiones permiten especificar dos o más máquinas de estados que se pueden ejecutar en paralelo en el contexto del objeto que las contiene. Si una región ortogonal alcanza su estado final antes que la otra, está la espera a que alcance su estado final y luego el control de las dos regiones se vuelve a unir en un único flujo



División y Unión:
Normalmente, la entrada a un estado compuesto con regiones ortogonales va al estado inicial de cada región ortogonal. También es posible pasar del estado externo directamente a uno o más estados ortogonales. Esto se denomina división o fork, porque el control pasa de un estado simple a varios estados ortogonales. Se representa con una línea negra y gruesa a la que llega una flecha y de la que salen varias flechas

Una unión o join es una transición con dos o más flechas entrantes y una flecha saliente. Cada flecha entrante debe venir desde un estado en regiones ortogonales diferentes del mismo estado compuesto



Objetos activos:
Otra forma de modelar la concurrencia es utilizar objetos activos. En vez de particionar la máquina de estados de un objeto en dos regiones concurrentes, se pueden definir dos objetos activos, cada uno de los cuales es responsable del comportamiento de una de las regiones concurrentes.
Si el comportamiento de uno de estos flujos concurrentes se ve afectado por el estado del otro, se puede modelar utilizando regiones ortogonales.
Si el comportamiento de uno de estos flujos concurrentes se ve afectado por los mensajes enviados a y desde otro, se puede modelar esto utilizando objetos activos.

PARA MODELAR LA VIDA DE UN OBJETO:

- Establecer el contexto de la máquina de estados. En el contexto de una clase, considerar las vecinas. En el contexto del sistema global, considerar sólo uno de los comportamientos del sistema.
- Hay que establecer los estados inicial y final del objeto
- Hay que decidir a qué eventos puede responder el objeto
- Hay que diseñar los estados de alto nivel en los que puede estar el objeto, comenzando con el estado inicial y acabando en el estado final.
- Hay que especificar cualquier acción de entrada o salida
- Hay que expandir los estados con subestados si es necesario
- Hay que comprobar que todos los eventos mencionados en la máquina de estados estan incluido en el conjunto posible de eventos proporcionado por la interfaz del objeto
- Hay que comprobar que todas las acciones mencionadas en la máquina de estados están soportadas por las relaciones, métodos y operaciones del objeto que la contiene.
- Hay que recorrer la máquina de estados, para comprobar las secuencias esperadas de eventos y sus respuestas
- Después de reorganizar la máquina de estados, hay que contrastrla con las secuencias esperadas de nuevo, para asegurarse de que no se ha cambiado la semántica del objeto.

TIEMPO Y ESPACIO

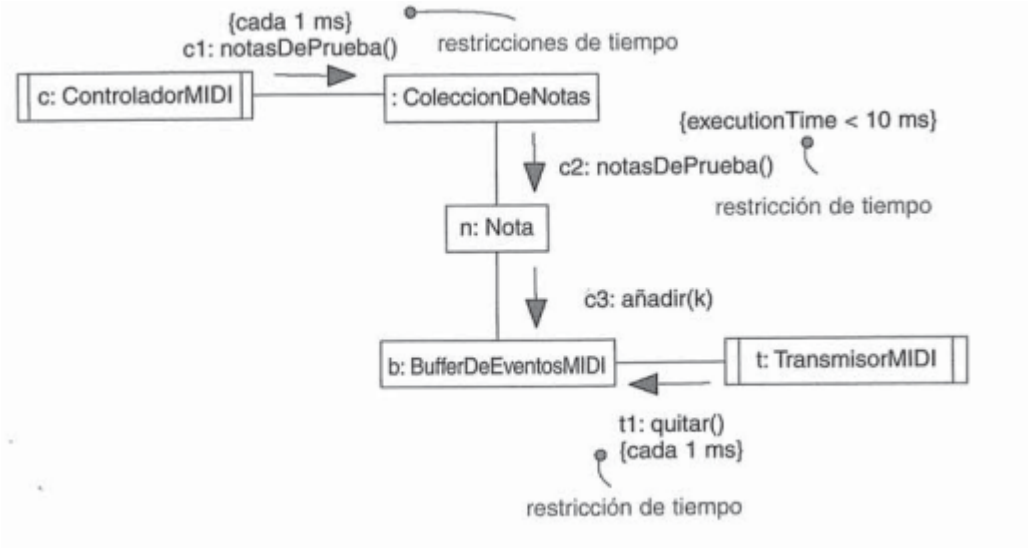
Cuando empezamos a modelar, consideramos un entorno sin conflictos(los mensajes se envían en un tiempo cero, las redes nunca se caen, etc.); cosa que en el mundo real no es así.

CONCEPTOS

- Marca de tiempo: denota el instante en el que ocurre un evento. Se representa como un guión en el borde del diagrama de secuencia.
- Expresión de tiempo: es una expresión que al evaluarse genera un valor de tiempo absoluto o relativo.
- Restriccion de tiempo: es un enunciado semántico sobre el valor absoluto o relativo del tiempo
- Localización: es la asignación de un componente a un nodo

TIEMPO

Los sistemas de tiempo real son aquellos sistemas en los cuales el tiempo desempeña un papel crítico. Los eventos pueden ocurrir a intervalos de tiempo regulares o irregulares, la respuesta puede ocurrir en un lapso de tiempo predecible absoluto o predecible relativo al propio evento.
Los mensajes representan el aspecto dinámico de un sistema, por ello, se le puede dar nombre a cada mensaje de una interacción, para poder utilizarlo en expresiones temporales. Las expresiones de tiempo se pueden insertar en una restricción de tiempo para especificar un comportamiento temporal del sistema. Estas restricciones se pueden representar colocándolas junto al mensaje apropiado o conectandolas mediante relaciones de dependencia



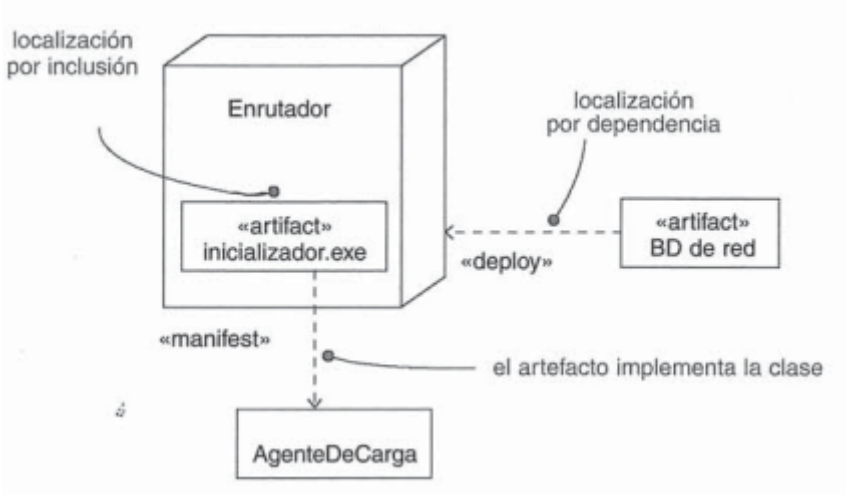
LOCALIZACIÓN

Los sistemas distribuidos incluyen componentes dispersos físicamente en distintos nodos

La vista de despliegue es la que modela mediante diagramas de despliegue, la topología de los procesadores y dispositivos sobre los que se ejecuta el sistema. Los artefactos como los programas ejecutables, y las bibliotecas residen en los nodos.

La localización de un elemento se puede modelar de dos formas:

- El elemento se puede incluir fisicamente(textual o gráficamente) en un comportamiento extra del nodo que lo contiene.
- Se puede utilizar una dependencia con la palabra clave *deploy* que una el artefacto con el nodo que lo contiene.



TÉCNICAS COMUNES DE MODELADO

Modelado de restricciones de tiempo

Para modelar restricciones de tiempo:

- Hay que considerar si cada evento incluido en una interacción debe comenzar en un tiempo absoluto
- Para cada secuencia interesante de mensajes en una interacción, hay que considerar si hay un tiempo máximo relativo asociado a ella

Modelado de la distribución de objetos

Para modelar la distribución de objetos:

- Para cada clase de objeto del sistema que desempeñe un papel importante, hay que considerar su localidad de referencia, es decir, hay que considerar todos los objetos vecinos y su localización. Hay que colocar los objetos, de forma tentativa, lo mas cerca posible de los actores que los manipulan.
- Considerar los patrones de interacción entre conjuntos relacionados de objetos. Colocar junto los conjuntos de objetos de alto grado de interacción y separar los conjuntos de objetos con un bajo grado de interacción.
- Considerar la distribución de responsabilidades en el sistema. Hay que redistribuir los objetos para equilibrar la carga de cada nodo.
- Considerar cuestiones de seguridad, volatilidad y calidad del servicio; redistribuyendo los objetos según convenga.
- Asignar los objetos altamente acoplados al mismo artefacto

- Asignar los artefactos a los nodos, de forma que las necesidades de computación de cada nodo estén de acuerdo con su capacidad.
- Equilibrar los costes de rendimiento y de comunicación asignando los artefactos que están estrechamente acoplados al mismo nodo.

DIAGRAMAS DE ESTADOS

Los diagramas de estados son uno de los cinco tipos de diagramas de UML que se utilizan para modelar los aspectos dinámicos de un sistema. Un diagrama de estados muestra una máquina de estados. Tanto los diagramas de actividades como los diagramas de estados son útiles para modelar la vida de un objeto.

Cuando se modelan sistemas con gran cantidad de software, la forma más natural de visualizar, especificar, construir y documentar el comportamiento de ciertos tipos de objetos es centrarse en el flujo de control entre estados en vez del flujo de actividades.

TÉRMINOS Y CONCEPTOS

Un diagrama de estados muestra una máquina de estados, destacando el flujo de control entre estados. Una máquina de estados es un comportamiento que especifica las secuencias de estados por las que pasa un objeto a lo largo de su vida en respuesta a eventos, junto con sus respuestas a esos eventos.

USOS COMUNES

En el modelado de los aspectos dinámicos de un sistema, una clase o un caso de uso, los diagramas de objetos se utilizan normalmente para modelar objetos reactivos.

Un objeto reactivo (o dirigido por evento) es aquel para el que la mejor forma de caracterizar su comportamiento es señalar cuál es su respuesta a los eventos lanzados desde fuera de su contexto.

TÉCNICAS COMUNES DE MODELADO

Modelado de objetos reactivos

Cuando se modela el comportamiento de un objeto reactivo, normalmente se especifican tres cosas:

- Los estados estables en los que puede encontrarse el objeto
- Los eventos que disparan una transición entre estados
- Las acciones que tienen lugar durante cada cambio de estado

Un estado estable representa una condición que puede satisfacer un objeto durante un periodo de tiempo identificable.

Para modelar un objeto reactivo:

- Hay que elegir el contexto para la máquina de estados, ya sea una clase, un caso de uso o el sistema global.

- Hay que elegir los estados inicial y final del objeto.
- Hay que elegir los estados estables del objeto considerando las condiciones que puede satisfacer el objeto durante algún periodo identificable de tiempo.
- Hay que elegir un orden parcial significativo de los estados estables a lo largo de la vida del objeto.
- Hay que elegir los eventos que pueden disparar una transición de un estado a otro.
- Hay que asociar acciones a estas transiciones y/o a los estados.
- Hay que considerar diferentes formas de simplificar la máquina con subestados, bifurcaciones, divisiones, uniones y estados de historia.
- Hay que comprobar que todos los estados son alcanzables mediante alguna combinación de eventos.
- Hay que comprobar que ningún estado es un punto muerto del cual no se puede salir con ninguna combinación de eventos.
- Hay que hacer trazas a través de la máquina de estados, ya sea manualmente o con herramientas, para verificar las secuencias esperadas de eventos y sus respuestas.

Ingeniería directa e inversa

La ingeniería directa es posible con los diagramas de estados, especialmente si el contexto del diagrama es una clase.

La ingeniería inversa es teóricamente posible, pero no es muy útil en la práctica. Las herramientas de ingeniería inversa no tienen capacidad de abstracción y, por lo tanto, no pueden producir diagramas de estados significativos.

Patrones Generales de Asignación de Responsabilidades (GRASP)

Los patrones GRASP constituyen un apoyo para la enseñanza que ayuda a uno a entender el diseño de objetos esencial, y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable. Este enfoque para la comprensión y utilización de los principios de diseño se basa en los patrones de asignación de responsabilidades.

Las responsabilidades están relacionadas con las obligaciones de un objeto en cuanto a su comportamiento.

Entre las responsabilidades de hacer de un objeto se encuentran:
Hacer algo él mismo, como crear un objeto o hacer un cálculo - Iniciar una acción en otros objetos - Controlar y coordinar actividades en otros objetos.
Entre las responsabilidades de conocer de un objeto se encuentran*:* Conocer los datos privados encapsulados - Conocer los objetos relacionados - Conocer las cosas que puede derivar o calcular.

"Una responsabilidad no es lo mismo que un método", los métodos se implementan para llevar a cabo responsabilidades. Las responsabilidades se implementan utilizando métodos que o actúan solos o colaboran con otros métodos u objetos.

Responsabilidades y los diagramas de interacción

En resumen, los diagramas de interacción muestran elecciones en la asignación de responsabilidades a los objetos. Cuando se crean, se han tomado las decisiones acerca de la asignación de responsabilidades, lo que se refleja en los mensajes que se envían a diferentes clases de objetos.

Patrones

Los desarrolladores orientados a objetos con experiencia acumulan un repertorio de principios generales que los ayudan a aplicar ciertos estilos que les guían en la creación de software. **Estos principios y estilos, si se codifican con un formato estructurado que describa el problema y la solución, se los denomina patrones.**

GRASP: Patrones de Principios Generales para Asignar Responsabilidades

¿Qué son los patrones GRASP?: Describen los principios fundamentales del diseño de objetos y la asignación de responsabilidades, expresados como patrones.

GRASP es un acrónimo de General Responsibility Assignment Software Patterns (patrones generales de software para asignar responsabilidades)

TIPOS PATRONES GRASP

1. **Experto en Información (o Experto): Asignar una responsabilidad al experto en información (la clase que tiene la información necesaria para realizar la responsabilidad).**
   El Experto en Información se utiliza con frecuencia en la asignación de responsabilidades; es un principio de guía básico que se utiliza continuamente en el diseño de objetos. Nótese que el cumplimiento de la responsabilidad a menudo requiere información que se encuentra dispersa por diferentes clases de objetos. Esto implica que hay muchos expertos en información “parcial” que colaborarán en la tarea.
   DESVENTAJA: En algunas ocasiones la solución que sugiere el Experto no es deseable, normalmente debido a problemas de acoplamiento y cohesión.
   BENEFICIOS: Se mantiene el encapsulamiento de la información, puesto que los objetos utilizan su propia información para llevar a cabo las tareas. Normalmente, esto conlleva un bajo acoplamiento, lo que da lugar a sistemas más robustos y más fáciles de mantener.

2. **Creador: B es un creador de objetos A. Si se puede aplicar más de una opción, inclínese por una clase B que agregue o contenga la clase A:**

- B *agrega* objetos de A.
- B *contiene* objetos de A.
- B *registra* instancias de objetos de A.
- B *utiliza más estrechamente* objetos de A.
- B *tiene los datos de inicialización* que se pasarán a un objeto de A cuando sea creado (por tanto, B es un Experto con respecto a la creación de A).

La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia, es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien, el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización. **La intención básica del patrón Creador es encontrar un creador que necesite conectarse al objeto creado en alguna situación. Eligiéndolo como el creador se favorece el bajo acoplamiento. El Agregado agrega Partes, el Contenedor contiene Contenido, y el Registro registra Registros, son todas ellas relaciones comunes entre las clases en un diagrama de clases. El Creador sugiere que la clase contenedor o registro es una buena candidata para asignarle la responsabilidad de crear lo que contiene o registra. Por supuesto, esto es sólo una guía.**

DESVENTAJAS: A menudo, la creación requiere una complejidad significativa, como utilizar instancias recicladas por motivos de rendimiento, crear condicionalmente una instancia a partir de una familia de clases similares basado en el valor de alguna propiedad externa, etcétera. En estos casos, es aconsejable delegar la creación a una clase auxiliar denominada Factoría.

VENTAJAS: Se soporta bajo acoplamiento (descrito a continuación), lo que implica menos dependencias de mantenimiento y mayores oportunidades para reutilizar

INTRODUCCIÓN A BAJO ACOPLAMIENTO Y ALTA COHESIÓN:

Alta cohesión y bajo acoplamiento

¿Qué significa? ¿Qué es cohesión y acoplamiento?

El grado de cohesión mide la coherencia de una clase, esto es, lo coherente que es la información que almacena una clase con las responsabilidades y relaciones que ésta tiene con otras clases.

El grado de acoplamiento indica lo vinculadas que están unas clases con otras, es decir, lo que afecta un cambio en una clase a las demás y por tanto lo dependientes que son unas clases de otras.

3. **Bajo Acoplamiento: Asignar una responsabilidad de manera que el acoplamiento permanezca bajo.**

Una clase con alto (o fuerte) acoplamiento confía en muchas otras clases. Tales clases podrían no ser deseables; algunas adolecen de los siguientes problemas:

- Los cambios en las clases relacionadas fuerzan cambios locales.
- Son difíciles de entender de manera aislada.
- Son difíciles de reutilizar puesto que su uso requiere la presencia adicional de las clases de las que depende.

El patrón de Bajo Acoplamiento es un principio a tener en mente en todas las decisiones de diseño; es un objetivo subyacente a tener en cuenta continuamente. Impulsa la asignación de responsabilidades de manera que su localización no incremente el acoplamiento hasta un nivel que nos lleve a los resultados negativos que puede producir un acoplamiento alto. El Bajo Acoplamiento soporta el diseño de clases que son más independientes, lo que reduce el impacto del cambio.

En general, las clases que son inherentemente muy genéricas por naturaleza, y con una probabilidad de reutilización alta, debería tener un acoplamiento especialmente bajo.

El caso extremo de Bajo Acoplamiento es cuando no existe acoplamiento entre clase.Si el Bajo Acoplamiento se lleva al extremo, producirá un diseño pobre porque dará lugar a unos pocos objetos inconexos, saturados, y con actividad compleja que hacen todo el trabajo, con muchos objetos muy pasivos, sin acoplamiento que actúan como simples repositorios de datos.

4. **Alta Cohesión: Asignar una responsabilidad de manera que la cohesión permanezca alta.** En cuanto al diseño de objetos, la cohesión es una medida de la fuerza con la que se relacionan y del grado de focalización de las responsabilidades de un elemento.

   Una clase con baja cohesión hace muchas cosas no relacionadas, o hace demasiado trabajo. Tales clases no son convenientes; adolecen de los siguientes problemas:

   - Difíciles de entender.
   - Difíciles de reutilizar.
   - Difíciles de mantener.
   - Delicadas, constantemente afectadas por los cambios.

   A menudo, las clases con baja cohesión representan un “grano grande” de abstracción, o se les han asignado responsabilidades que deberían haberse delegado en otros objetos.
   Igual que el Bajo Acoplamiento, el patrón de Alta Cohesión es un principio a tener en mente durante todas las decisiones de diseño; es un objetivo subyacente a tener en cuenta continuamente.

   Grados de cohesión funcional:

- ➢ **Muy baja cohesión.** Una única clase es responsable de muchas cosas en áreas funcionales muy diferentes.
- ➢ **Baja cohesión.** Una única clase tiene la responsabilidad de una tarea compleja en un área funcional
- ➢ **Alta cohesión**. Una clase tiene una responsabilidad moderada en un área funcional y colabora con otras clases para llevar a cabo las tareas.
- ➢ **Moderada cohesión.** Una clase tiene responsabilidades ligeras y únicas en unas pocas áreas diferentes que están lógicamente relacionadas con el concepto de la clase, pero no entre ellas

VENTAJAS:
• Se incrementa la claridad y facilita la comprensión del diseño.
• Se simplifican el mantenimiento y las mejoras.
• Se soporta a menudo bajo acoplamiento.
• El grano fino de funcionalidad altamente relacionada incrementa la reutilización porque una clase cohesiva se puede utilizar para un propósito muy específico.

5. **Controlador: Asignar la responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase que representa una de las siguientes opciones:**
   - Representa el sistema global, dispositivo o subsistema (controlador de fachada).
   - Representa un escenario de caso de uso en el que tiene lugar el evento del sistema, a menudo denominado Manejador, Coordinador o Sesión (controlador de sesión o de caso de uso).
   - Utilice la misma clase controlador para todos los eventos del sistema en el mismo escenario de caso de uso.
   - Informalmente, una sesión es una instancia de una conversación con un actor.

Un evento del sistema de entrada es un evento generado por un actor externo. Se asocian con operaciones del sistema, tal como se relacionan los mensajes y los métodos.

Un Controlador es un objeto que no pertenece a la interfaz de usuario, responsable de recibir o manejar un evento del sistema. Un Controlador define el método para la operación del sistema.
El patrón Controlador proporciona guías acerca de las opciones generalmente aceptadas y adecuadas. El controlador es una especie de fachada en la capa del dominio para la capa de la interfaz.

A menudo, es conveniente utilizar la misma clase controlador para todos los eventos del sistema de un caso de uso de manera que es posible mantener la información acerca del estado del caso de uso en el controlador.
Un error típico del diseño de los controladores es otorgarles demasiada responsabilidad.

VENTAJAS:

- **Aumenta el potencial para reutilizar y las interfaces conectables (pluggable): Asegura que la lógica de la aplicación no se maneja en la capa de interfaz.** Un diseño de una interfaz como controlador reduce la oportunidad de reutilizar la lógica en futuras aplicaciones, puesto que está ligada a una interfaz particular (por ejemplo, objetos ventana) que es raramente aplicable en otras aplicaciones. En cambio, delegando la responsabilidad de una operación del sistema a un controlador ayuda a la reutilización de la lógica en futuras aplicaciones
- **Razonamiento sobre el estado de los casos de uso:** A veces es necesario asegurar que las operaciones del sistema tienen lugar en una secuencia válida, o ser capaces de razonar sobre el estado actual de la actividad y operaciones del caso de uso que está en marcha.

DESVENTAJAS: Una clase controlador pobremente diseñada tendrá baja cohesión —no centrada en algo concreto y que gestiona demasiadas áreas de responsabilidad—; este controlador se denomina **controlador saturado**:

- Existe una *única* clase controlador que recibe *todos* los eventos del sistema en el sistema, y hay muchos. Esto ocurre a veces si se elige un controlador de fachada.
- El propio controlador realiza muchas de las tareas necesarias para llevar a cabo los eventos del sistema, sin delegar trabajo. Normalmente esto conlleva una violación de los patrones Experto en Información y Alta Cohesión.
- Un controlador tiene muchos atributos y mantiene información significativa sobre el sistema o el dominio, que debería haberse distribuido a otros objetos, o duplica información que se encuentra en otros sitios.

Hay varios remedios para un controlador saturado entre los que se encuentran:

1. Añadir más controladores: un sistema no tiene que tener sólo uno. En lugar de un controlador de fachada, utilice controladores de casos de uso. Por ejemplo, considere una aplicación con muchos eventos del sistema, como un sistema de reservas de vuelos.

2. Diseñe el controlador de manera que, ante todo, delegue el cumplimiento de cada responsabilidad de una operación del sistema a otros objetos.