UNIDAD 3

MODELADO PARA EL ANÁLISIS DE REQUERIMIENTOS

INTRODUCCIÓN

Para que los analistas de sistemas puedan comprender los requerimientos de información de los usuarios, deben ser capaces de conceptualizar la forma en que los datos se mueven a través de la organización, los procesos o la transformación por la que pasan los datos y las salidas de los mismos.

Aunque las entrevistas y la investigación de datos “duros” proveen una narrativa verbal del sistema, una descripción visual puede cristalizar esta información para los usuarios y analistas de una manera útil.

El análisis y diseño orientados a objetos pueden ofrecer una metodología que facilita los métodos lógicos, rápidos y detallados para crear sistemas que respondan a un panorama de negocios en evolución. Las técnicas orientadas a objetos funcionan bien en situaciones en las que los sistemas de información complicados pasan a través de un proceso continuo de mantenimiento, adaptación y rediseño.

MODELADO

Un modelo captura las propiedades estructurales (estática) y de comportamiento (dinámicas) de un sistema.

- Es una abstracción de dicho sistema, considerando un cierto propósito.
- El modelo describe completamente aquellos aspectos del sistema que son relevantes al propósito del modelo, y a un apropiado nivel de detalle.
- Un proceso de desarrollo de software debe ofrecer un conjunto de modelos que permitan expresar el producto desde cada una de las perspectivas de interés.
- El código fuente del sistema es el modelo más detallado del sistema (y además, es ejecutable). Sin embargo, se requieren otros modelos.
- Cada modelo es completo desde un punto de vista del sistema. Sin embargo, existen relaciones de trazabilidad entre los diferentes modelos.

MODELADO

Durante el desarrollo de un sistema de software se requiere que el sistema sea visto desde varias perspectivas. Diferentes usuarios miran el sistema de formas diferentes en momentos diferentes.

La arquitectura del sistema, clave para poder manejar estos puntos de vista diferentes, puede describirse mejor a través de vistas arquitecturales interrelacionadas.

UML - UNIFIED MODELING LANGUAGE

El Lenguaje de Modelado Unificado (UML, *Unified Modeling Language*) es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. UML entrega una forma de modelar cosas conceptuales como lo son procesos de negocio y funciones de sistema, además de cosas concretas como lo son escribir clases en un lenguaje determinado, esquemas de base de datos y componentes de software reutilisables.

Sirve para especificar, visualizar, construir y documentar los artefactos de los sistemas software, así como para el modelado del negocio y otros sistemas no software [OMGOl].

UML se ha convertido en la notación visual estándar de facto para el modelado orientado a objetos. Comenzó como una iniciativa de Grady Booch y Jim Rumbaugh en 1994 para combinar las notaciones visuales de sus dos populares métodos. Más tarde se les unió Ivar Jacobson, el creador del método Objectory.

UML fue adoptado en 1997 como estándar por el OMG (*Object Management Group,* organización que promueve estándares para la industria), y continúa siendo refinado en nuevas versiones.

Actualmente es un lenguaje refinado para que pueda ser eficientemente aplicado en el desarrollo de aplicaciones software, tanto si son proyectos pequeños llevados por una o dos personas o si son grandes proyectos desarrollados por cientos de personas.

UML - UNIFIED MODELING LANGUAGE

UML provee un conjunto estandarizado de herramientas para documentar el análisis y diseño de un sistema de software. El conjunto de herramientas de UML incluye diagramas que permiten a las personas visualizar la construcción de un sistema orientado a objetos, algo similar a la forma en que los planos de construcción permiten a las personas visualizar la construcción de un edificio. La documentación que puede crear con UML provee un medio efectivo de comunicación entre el equipo de desarrollo y el equipo de negocios en un proyecto.

CATEGORÍA DE UML	ELEMENTOS DE UML	DETALLES ESPECÍFICOS
ELEMENTOS	Elementos Estructurales	Clases / Interfaces / Colaboraciones Casos de uso / Clases activas / Objeto Componentes / Nodos
	Elementos de comportamiento	Interacciones / Máquinas de estado
	Elementos de agrupamiento	Paquetes
	Elementos de anotaciones	Notas
RELACIONES	Relaciones estructurales	Dependencias / Agregaciones Asociaciones / Generalizaciones
	Relaciones de comportamiento	Comunica / Incluye Extiende / Generaliza
DIAGRAMAS	Diagramas estructurales Estáticos	Diagramas de clases Diagramas de componentes Diagramas de despliegue
	Diagramas de comportamiento Dinámicos	Diagramas de casos de uso Diagramas de secuencia Diagramas de comunicación Diagramas de estados Diagramas de actividad

UML – MODELO 4+1

Vista de Diseño

Diagramas de Clases
Diagramas de Actividad
Diagramas de Estados
Diagramas de Objetos
Diagramas de Secuencia

- Vocabulario
- Funcionalidad

Vista de Implementación

Diagramas de Estructura Compuesta
Diagramas de Componentes
Diagramas de Paquetes
Diagramas de Clases

- Ensamblado del sistema
- Gestión de la Configuración

Vista de Casos de Uso

Diagramas de Casos de Uso
Diagramas de Secuencia
Diagramas de Actividad

- Comportamiento

Vista de Interacción

Diagramas de Secuencia
Diagramas de Comunicación
Diagramas de Tiempos

- Rendimiento
- Escalabilidad
- Capacidad de Procesamiento

Vista de Despliegue

Diagramas de Despliegue
Diagramas de Paquetes

- Topología del Sistema
- Distribución
- Entrega
- Instalación

UML - UNIFIED MODELING LANGUAGE

A la hora de realizar un modelo de un sistema software, éste debe hacerse desde diferentes puntos de vista, de forma que recojan tanto la dimensión estática y estructural de los objetos como su componente dinámica.

Vista de Casos de Uso

- Captura la funcionalidad del sistema tal y como es percibido por los usuarios finales, analistas y encargados de pruebas.
- Comprende los casos de uso que describen el comportamiento asociado a dicha funcionalidad.
- En esta vista no se especifica la organización real del sistema software.
- Los diagramas que le corresponden son:
- Aspectos estáticos: diagramas de casos de uso.
- Aspectos dinámicos: diagramas de interacción, de estados y de actividades.

UML - UNIFIED MODELING LANGUAGE

Vista de Diseño

- Captura las clases, interfaces y colaboraciones que forman el vocabulario del problema y su solución.
- Soporta, principalmente, los requisitos funcionales del sistema (servicios que el sistema debe proporcionar a sus usuarios finales).
- Los diagramas que le corresponden son:
- Aspectos estáticos: diagramas de clases y de objetos. También son útiles los diagramas de estructura compuesta de clases.
- Aspectos dinámicos: diagramas de interacción, de estados y de actividades.

Vista de Interacción

- 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 como el rendimiento, escalabilidad y capacidad de procesamiento.

UML - UNIFIED MODELING LANGUAGE

Los diagramas que le corresponden son los mismos que la vista de diseño:

- Aspectos estáticos: diagramas de clases y de objetos.
- Aspectos dinámicos: diagramas de interacción, de estados y de actividades. Pero atendiendo más las clases activas que controlan el sistema y los mensajes entre ellas.

Vista de Implementación

- Captura los artefactos que se utilizan para ensamblar y poner en producción el sistema software real.
- Se centra en La configuración de las distintas versiones de los archivos físicos, correspondencia entre clases y ficheros de código fuente, correspondencia entre componentes lógicos y artefactos físicos.
- Los diagramas que le corresponden son:
- Aspectos estáticos: diagramas de componentes (especialmente la versión de artefactos) y de estructura compuesta.
- Aspectos dinámicos: diagramas de interacción, de estados y de actividades.

UML - UNIFIED MODELING LANGUAGE

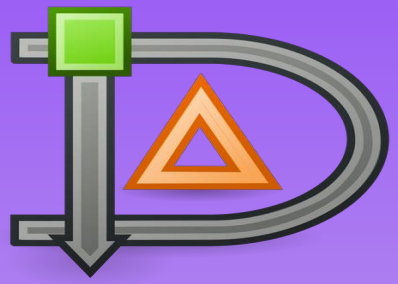
Vista de Despliegue

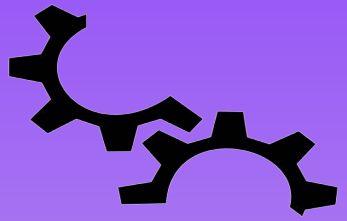
- Captura las características de instalación y ejecución del sistema.
- Contiene los nodos y enlaces que forman la topología hardware sobre la que se ejecuta el sistema software.
- Se ocupa principalmente de la distribución, entrega e instalación de las partes que forman el sistema software real.
- Los diagramas que le corresponden son:
- Aspectos estáticos: diagramas de despliegue.
- Aspectos dinámicos: diagramas de interacción, de estados y de actividades.

HERRAMIENTAS CASE

Visual Paradigm

gliffy





Visio

draw.io

Visual Paradigm Online

Lucidchart

IBM

MODELADO DE CASOS DE USO

UML se basa fundamentalmente en una técnica de análisis orientado a objetos conocida como modelado de casos de uso. Un modelo de casos de uso muestra una vista del sistema desde la perspectiva del usuario, por lo cual describe qué hace el sistema sin indicar cómo lo hace. Podemos utilizar UML para analizar el modelo de casos de uso y derivar los objetos del sistema junto con sus interacciones entre sí y con los usuarios del sistema. Al utilizar técnicas de UML podemos analizar con más detalle los objetos y sus interacciones para derivar su comportamiento, atributos y relaciones.

Un caso de uso provee a los desarrolladores un panorama sobre lo que desean los usuarios. Está libre de detalles técnicos o de implementación. Podemos pensar en un caso de uso como una secuencia de transacciones en un sistema. El modelo de casos de uso se basa en las interacciones y relaciones de los casos de uso individuales.

La esencia es descubrir y registrar los requisitos funcionales, mediante la escritura de historias del uso de un sistema, para ayudar a cumplir los objetivos de varias de las personas involucradas; esto es, los casos de uso. Se supone que no es una idea difícil, aunque, de hecho podría ser difícil descubrir o decidir lo que es necesario, y escribirlo de manera coherente con un nivel de detalle útil.

MODELADO DE CASOS DE USO

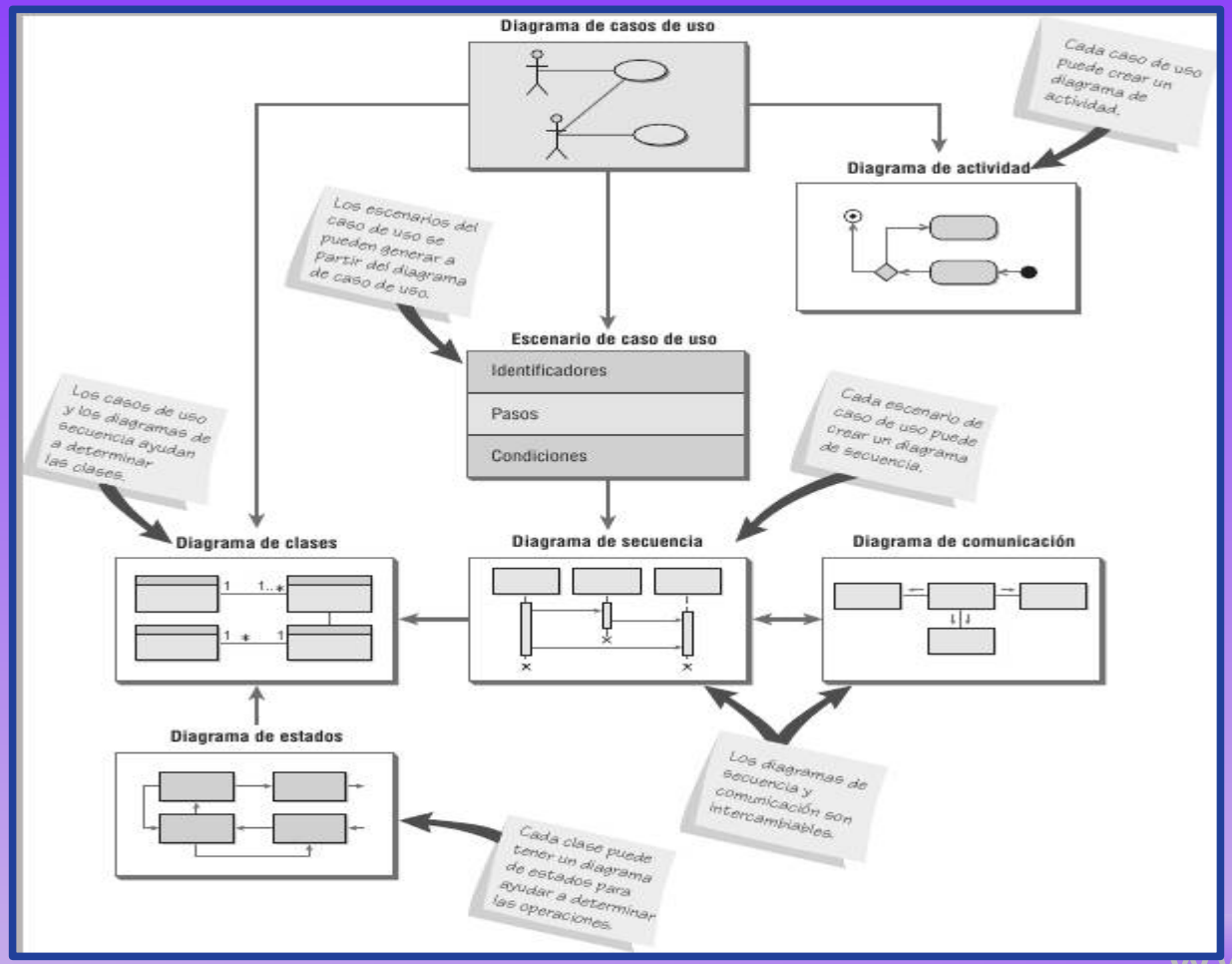
Los propósitos primarios de los casos de uso son:

- Decidir y describir los requerimientos funcionales del sistema, dando lugar a un acuerdo entre el cliente (y/o usuario final) y los programadores que desarrollan el sistema.
- Dar una descripción clara y consistente de lo que debería hacer el sistema, de modo que el modelo se use a lo largo del proceso de desarrollo.
- Proporcionar una base para realizar verificaciones (tests) del sistema que comprueben su funcionamiento.
- Proporcionar la capacidad para rastrear requerimientos funcionales en clases y operaciones reales del sistema, verificando los casos de uso afectados por cambios y extensiones al sistema.
- Guían todo el proceso de desarrollo del sistema. Ayudan a los jefes de proyecto a planificar, asignar y controlar las tareas de desarrollo.

El modelado de casos de uso también se utiliza cuando se desarrolla una nueva versión o actualización del sistema.

- Se añade la nueva funcionalidad al modelo de casos de uso existente insertando nuevos actores y casos de uso, o modificando la especificación de los casos de uso actuales.

MODELADO DE CASOS DE USO



Los diagramas de casos de uso proveen la base para crear otros tipos de diagramas, como los diagramas de clases y los diagramas de actividad. Los escenarios de casos de uso son útiles para dibujar diagramas de secuencia. Tanto los diagramas de casos de uso como los escenarios de casos de uso son potentes herramientas para ayudarnos a comprender la forma en que un sistema funciona en general.

DIAGRAMA DE CASOS DE USO

El diagrama de casos de uso representa la forma en como un Cliente (Actor) opera con el Sistema en desarrollo, además de la forma, tipo y orden en como los elementos interactúan (operaciones o casos de uso).

Un caso de uso siempre describe tres cosas: un actor que inicia un evento, el evento que desencadena un caso de uso y el caso de uso que realiza las acciones desencadenadas por el evento. En un caso de uso, un actor que utiliza el sistema inicia un evento que a su vez genera una serie relacionada de interacciones en el sistema. Los casos de uso se utilizan para documentar una transacción o evento individual. Se introduce un evento en el sistema, el cual ocurre en un tiempo y lugar específicos para provocar que el sistema haga algo.

- Un caso de uso describe una interacción entre el sistema y un agente externo.
- Un caso de uso capta siempre una función visible para el usuario.
- Un caso de uso logra un objetivo concreto y específico para el usuario.
- Un caso de uso puede ser algo simple o algo complejo, en este caso se puede formular en función de otros casos de uso.
- Los diagramas de casos de uso hacen referencia a la funcionalidad del sistema y no hacen referencia a la implementación. Especifican el comportamiento deseado del sistema. El qué, no el cómo.

DIAGRAMA DE CASOS DE USO

Un caso de uso es una operación/tarea específica que se realiza tras una orden de algún agente externo, sea desde una petición de un actor o bien desde la invocación desde otro caso de uso. Se representa en el diagrama por una elipse, denota un requerimiento solucionado por el sistema.

Un caso de uso en UML se define como una secuencia de acciones que realiza un sistema y que conduce a un resultado observable.

Las acciones pueden envolver comunicación con diversos actores (usuarios y otros sistemas) así como realización de cálculos y trabajos dentro del sistema.

Un escenario de caso de uso es cada uno de los diferentes caminos que pueden darse en un caso de uso, dependiendo de las condiciones que se den en su realización. Es la secuencia específica de acciones que ilustra un comportamiento.

Para describir un escenario debemos especificar

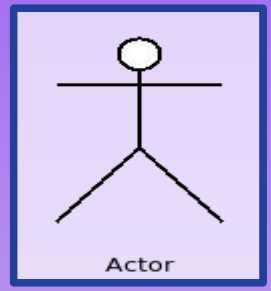
- Flujo básico de eventos:
- Cómo y cuándo empieza y acaba el caso de uso
- Cuándo interactúa con los actores y qué información se intercambian
- Flujos alternativos de comportamiento.

DIAGRAMA DE CASOS DE USO

Un Actor es un rol que un usuario juega con respecto al sistema. Es importante destacar el uso de la palabra rol, pues con esto se especifica que un Actor no necesariamente representa a una persona en particular, sino más bien la labor que realiza frente al sistema.

Un actor es algo o alguien que se encuentra fuera del sistema y que interactúa con él. En general los actores son personas u otros sistemas. Un único actor puede representar a varios usuarios distintos, y un usuario puede actuar como diferentes actores. Por ejemplo:

- el actor Cliente representa a varias personas distintas
- la persona M. García puede actuar como el actor Cliente y como el actor Proveedor



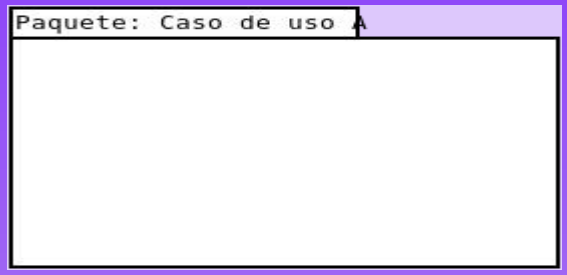
El actor envía o recibe mensajes a y desde el sistema, o intercambia información con el sistema.

Un caso de uso siempre es iniciado por un actor que le envía un mensaje o estímulo (stimulus) al Sistema. Los actores llevan a cabo casos de uso.

Un actor se representa gráficamente mediante un monigote con su nombre debajo que debería reflejar el papel del actor.

DIAGRAMA DE CASOS DE USO

Los paquetes se emplean para la organización de diagramas. Se determina una partición del sistema y a cada una de las divisiones se le asocia un paquete en el que se incluyen los casos de uso correspondientes.



Se representa gráficamente con una carpeta con el nombre en su interior.

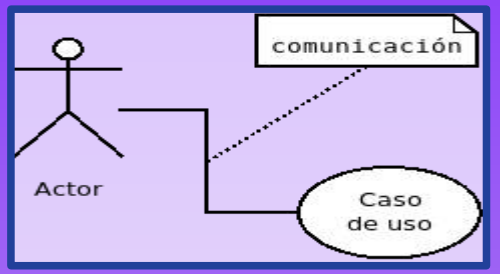
Las relaciones son interacciones entre los actores y los casos de uso y entre ellos.

Tipos de relaciones:

- Relaciones entre actores y casos de uso
- Relaciones de especialización entre actores
- Relaciones entre casos de uso:
- Generalización / Especialización
- Inclusión
- Extensión

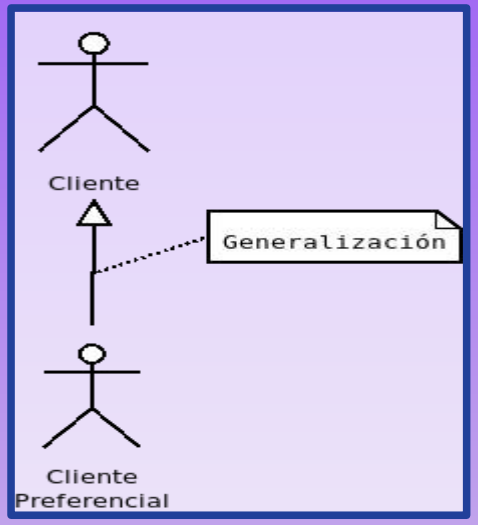
DIAGRAMA DE CASOS DE USO

Relación Actor - Caso de uso



- Es una relación de comunicación o asociación
- Representa comunicación en uno o dos sentidos (emisión y/o recepción)
- Se representa gráficamente mediante una línea continua, en algunas herramientas de diagramación puede ser una flecha simple.

Relación Actor-Actor

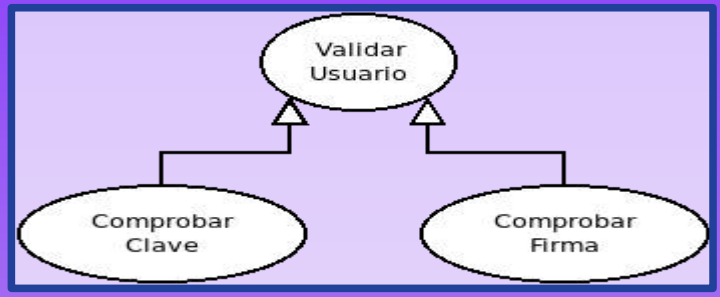


- Relación de generalización: se pueden definir categorías generales de actores (por ejemplo Cliente) y especializarlos (por ejemplo Cliente Preferencial)
- Esta relación se representa gráficamente mediante una flecha con la cabeza hueca dirigida al actor mas general
- Los actores especializados heredan el comportamiento y pueden extenderlo.

DIAGRAMA DE CASOS DE USO

Relación Generalización de casos de uso

- Un caso de uso (hijo) hereda el comportamiento y el significado de otro caso de uso (padre)
- El caso de uso hijo puede añadir comportamiento o bien redefinir el del padre.
- Se representa gráficamente con una flecha hueca dirigida al caso de uso padre



Relación Inclusión de casos de uso (dependencia)

- Representa un comportamiento común entre varios casos de uso: "un caso de uso base A incorpora explícitamente el comportamiento de otro caso de uso B en el lugar especificado en el caso de uso base"
- Gráficamente se representa mediante una flecha etiquetada con el estereotipo <<usa>> (<<include>>) que parte del caso de uso base hacia el incluido.
- Teóricamente el caso de uso incluido nunca se encuentra aislado (debe representar un comportamiento común a varios casos de uso)

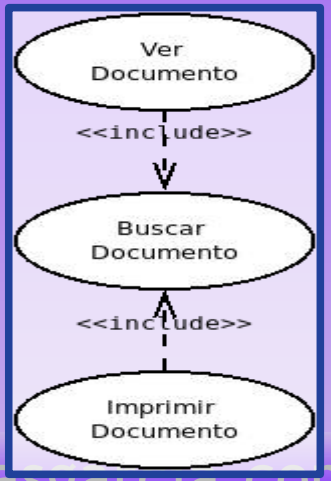


DIAGRAMA DE CASOS DE USO

Relación de extensión de casos de uso (dependencia)

- Un caso de uso añade acciones, que pueden ser opcionales, al comportamiento de un caso de uso general.
- El caso de uso base, por tanto, bajo ciertas condiciones incorpora el comportamiento de otro caso de uso en un punto concreto de su especificación denominado punto de extensión. Un caso de uso base puede tener varias relaciones de extensión y, en consecuencia, varios puntos de extensión.
- Se representa gráficamente mediante una flecha etiquetada con el estereotipo <<extiende>> (<<extend>>) que llega al caso de uso base desde el extendido.
- Para extender un caso de uso deben definirse puntos de extensión (extension points), esto es, dar una especificación de algún punto en el caso de uso donde insertar la extensión para añadir funcionalidad bajo las condiciones especificadas.
- Los puntos de extensión se muestran como una lista dentro del caso de uso extendido. Para dar más detalles, se puede conectar una nota a la línea de dependencia especificando bajo qué condiciones se ejecuta la extensión.

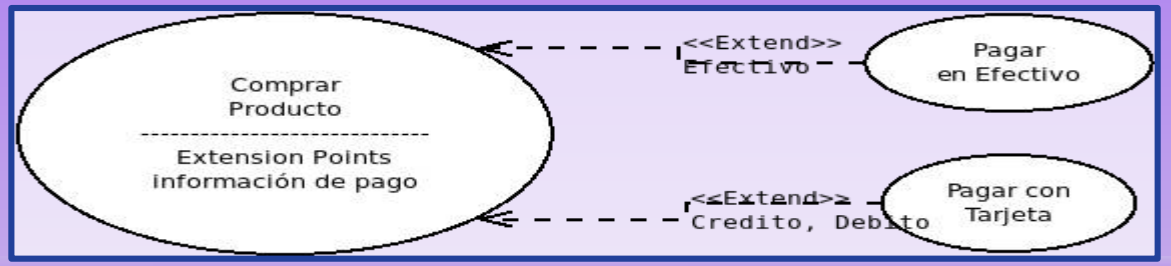


DIAGRAMA DE CASOS DE USO

Pasos para la elaboración de un diagrama de casos de uso:

1) Identificar los actores que interactúan con el sistema
2) Organizar los actores identificando tanto los roles generales como los especializados
3) Considerar las formas más importantes que tiene cada actor de interactuar con el sistema
4) Considerar las formas excepcionales en las que cada actor puede interactuar con el sistema
5) Organizar los comportamientos como casos de uso, utilizando las relaciones de Inclusión y extensión

Hay dos formas de aplicar los diagramas de casos de uso:

- Modelado del contexto de un sistema: Se determina qué actores quedan fuera del sistema e interactúan con él.
- Modelado de los requisitos de un sistema: Se especifica qué debe hacer el sistema, independientemente de cómo se haga. Se especifica el comportamiento del sistema como una caja negra.

DIAGRAMA DE CASOS DE USO

Modelado del contexto de un sistema

Se trata de diferenciar entre elementos del sistema y elementos externos al sistema:

- Los elementos del sistema desarrollan el comportamiento de éste
- Los elementos externos interactúan con el sistema

El contexto del sistema se puede definir como los elementos externos que interactúan con el sistema.

Para modelar el contexto de un sistema:

1) Identificar los actores entorno al sistema considerando:

- Qué grupos requieren funciones del sistema para desarrollar sus tareas.
- Qué grupos son necesarios para que el sistema desarrolle sus tareas.
- Qué interacción tiene el sistema con hardware externo o con otros sistemas.
- Qué grupos realizan tareas secundarias de administración y mantenimiento.

1) Organizar los actores similares en jerarquías de generalización/especialización.

2) Introducir los actores en el diagrama de casos de uso y especificar las vías de comunicación de cada actor con los casos de uso.

DIAGRAMA DE CASOS DE USO

Modelado de los requisitos de un sistema

- Un requisito se puede definir como una característica de diseño, propiedad o comportamiento de un sistema.
- El establecimiento de los requisitos determina un contrato entre el sistema y sus elementos externos.
- Los requisitos funcionales se pueden expresar mediante casos de uso.

Para modelar los requisitos hay que seguir los siguientes pasos:

1) Establecer el contexto del sistema, identificando los actores.
2) Considerar qué comportamiento del sistema espera cada actor.
3) Nombrar los comportamientos comunes como casos de uso.
4) Factorizar el comportamiento común (relación de Inclusión) en nuevos casos de uso y factorizar el comportamiento variante (relación de extensión) como nuevos casos de uso.
5) Modelar los casos de uso, actores y relaciones en diagramas de casos de uso.
6) Adornar esos casos de uso con notas que enuncien los requisitos no funcionales.
7) Posteriormente, especificar el comportamiento de cada caso de uso identificado con diagramas de interacción.

ESPECIFICACIÓN DE CASOS DE USO

La especificación o descripción de un caso de uso normalmente es textual.

- Es una especificación simple y consistente de cómo interactúan los actores y los casos de uso (el sistema).
- Se concentra en el comportamiento externo del sistema e ignora cómo se hacen realmente las cosas dentro del sistema.
- El lenguaje y la terminología son los mismos que los usados por el cliente/usuario del sistema.

La descripción textual debería incluir:

- Objetivo del caso de uso: qué intenta conseguir.
- Cómo se inicia el caso de uso: qué actor inicia la ejecución del caso de uso y en qué situaciones.
- El flujo de mensajes entre actores y el caso de uso: qué mensajes o eventos intercambian el caso de uso y el actor para notificarse algo,actualizar o recuperar información.
- Flujo alternativo en el caso de uso: un caso de uso puede tener ejecuciones alternativas dependiendo de condiciones o excepciones.
- Cómo finaliza el caso de uso con un valor para el actor: cuándo se considera finalizado el caso de uso y la clase de valor devuelto al actor.

ESPECIFICACIÓN DE CASOS DE USO

- Escenarios:
  - Escenario principal (o de éxito)
  - Escenarios alternativos o de excepción (indicar la condición que determina la variación):
    - Datos de entrada incorrectos
    - Datos de entrada incompletos
    - Imposibilidad de completar el caso de uso por vulnerar reglas de negocio
  - Otras situaciones de excepción
- Observaciones:
  - En el modelo de negocio no se deben tener en cuenta detalles de interfaz de usuario
  - En cada paso, se especifica una acción realizada por el actor o por el sistema
  - Especificar claramente cómo se inicia el caso de uso

ESPECIFICACIÓN DE CASOS DE USO

- Pre-condiciones:
  - Condiciones que se asumen para ejecutar el caso de uso
    - Ejemplo: “El usuario debe tener acceso validado con el perfil de jefe de servicio”
- Post-condiciones (o garantías):
  - Mínimas: Condición que se verificará en cualquier escenario del caso de uso
    - Ejemplo: “En la auditoria de la aplicación quedará registrado el acceso a la validación de solicitudes de cambio de perfil”
  - De éxito: Condición que se verificará en los escenarios de éxito
    - Ejemplo: “El estado de la solicitud de cambio de perfil pasará a ser validado por el jefe de servicio”
- Prioridad:
  - Es esencial para la planificación del proyecto (y sus iteraciones o incrementos)
  - Los posibles valores varían en cada organización (alta, media, baja).

ESPECIFICACIÓN DE CASOS DE USO

Identificador	CU-<**id-requisito**>	
Nombre	<**nombre del requisito funcional**>	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso {concreto cuando <evento de activación> , abstracto durante la realización de los casos de uso <lista de casos de uso>}	
Precondición	<precondición del caso de uso>	
Secuencia Normal	**Paso**	**Acción**
	1	{El <actor> , El sistema} <acción realizada por el actor o sistema>, se realiza el caso de uso < caso de uso CU-x>
	2	Si <condición>, {el <actor> , el sistema} <acción realizada por el actor o sistema>>, se realiza el caso de uso < caso de uso CU-x>
	…	…
Postcondición	<postcondición del caso de uso>	
Excepciones	**Paso**	**Acción**
	1	Si <condición de excepción>,{el <actor> , el sistema} }<acción realizada por el actor o sistema>>, se realiza el caso de uso < caso de uso CU-x>, a continuación este caso de uso {continua, aborta}
	…	…
Rendimiento	**Paso**	**Cota de tiempo**
	1	n segundos
	…	…
Frecuencia esperada	<nº de veces> veces / <unidad de tiempo>	
Importancia	{sin importancia, importante, vital}	
Urgencia	{puede esperar, hay presión, inmediatamente}	
Comentarios	<comentarios adicionales>	

DIAGRAMA DE ACTIVIDADES

Los diagramas de actividades representan la dinámica del sistema.

Muestran el flujo del control en el sistema conforme pasa de una actividad a otra, cuales actividades pueden ser llevadas a cabo en paralelo y cualquier trayectoria alternativa del flujo.

En la etapa inicial (inception) del proyecto, los diagramas de actividades pueden ser creados para representar el flujo entre casos de uso, o para representar el flujo dentro de un caso de uso en particular.

En las etapas avanzadas del proyecto, los diagramas de actividades pueden ser creados para mostrar el flujo de una operación en el diseño del sistema.

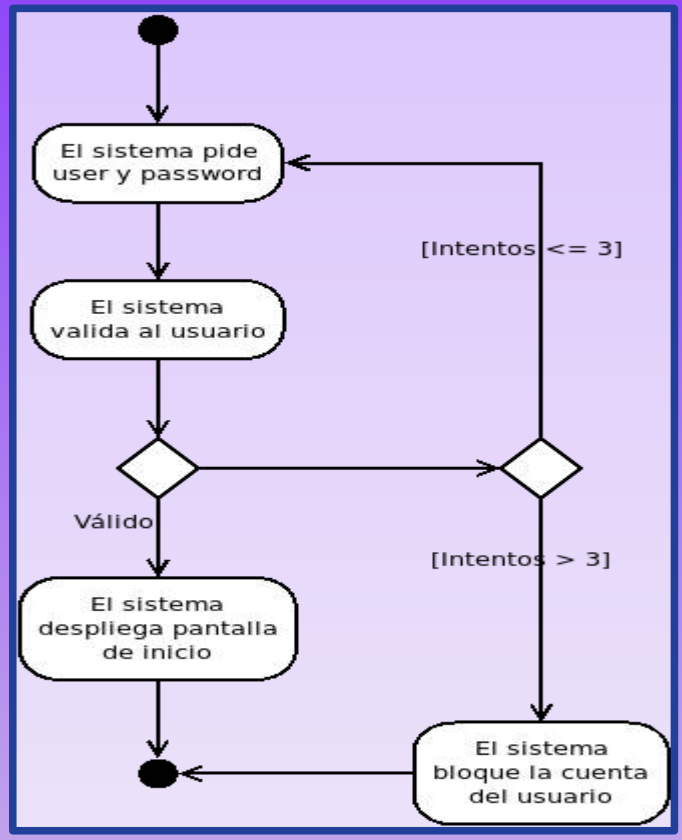


DIAGRAMA DE ACTIVIDADES

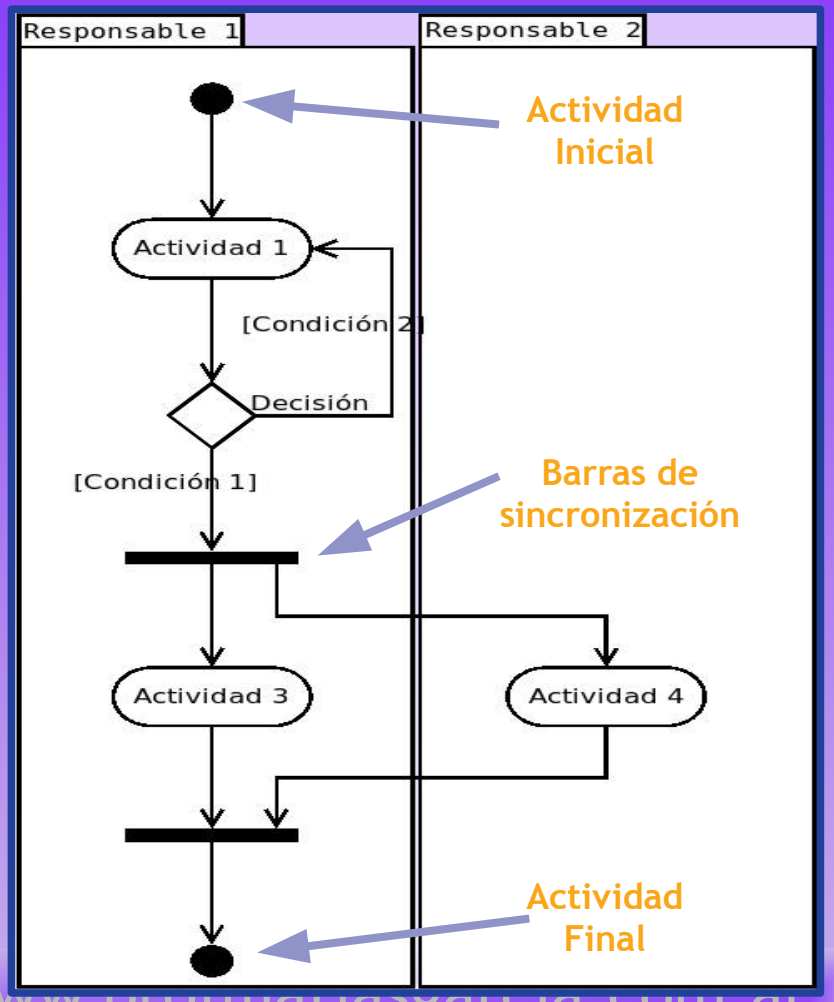
Diagrama de actividades con marcos de responsabilidad.

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

Se representan por un rectángulo con bordes redondeados.

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

Decisiones: Se representa mediante una transición múltiple que sale de un estado, donde cada camino tiene un label distinto.



TARJETAS CRC

CRC representa a las Clases, Responsabilidades y Colaboradores. El analista puede usar estos conceptos cuando empieza a hablar sobre el sistema o a modelarlo a partir de una perspectiva orientada a objetos. Las tarjetas CRC se utilizan para representar las responsabilidades de las clases y las interacciones entre ellas. Los analistas crean las tarjetas con base en escenarios que describen los requerimientos del sistema. Estos escenarios modelan el comportamiento del sistema que se está estudiando.

Las tarjetas CRC se pueden crear de manera interactiva con unos cuantos analistas que puedan trabajar en conjunto para identificar la clase en el dominio del problema presentado por la empresa. Una sugerencia es buscar todos los sustantivos y verbos en un enunciado del problema que se haya creado para capturarlo. Por lo general, los sustantivos indican las clases en el sistema; para encontrar las responsabilidades hay que identificar los verbos. Una vez identificadas todas las clases, los analistas pueden empezar a compilarlas, eliminar las ilógicas y escribir cada una en su tarjeta. Se asignara una clase a cada persona en el grupo, que será su “propietaria” durante la sesión CRC.

TARJETAS CRC

A continuación, el grupo crea escenarios que en realidad son recorridos de las funciones del sistema, para lo cual se toma la funcionalidad deseada del documento de requerimientos creado con anterioridad. Después el grupo procede a refinar la responsabilidad en tareas cada vez más pequeñas, de ser posible. Si es apropiado, el objeto pude cumplir con estas tareas o el grupo puede decidir que se cumplan mediante la interacción con otras cosas. Si no existen otras clases apropiadas, tal vez el grupo tenga que crear una.

Las responsabilidades que se enlistan evolucionarán en un momento dado en lo que se conoce como métodos en UML.

<nombre de la clase>	
Descripción:	
Superclase	Subclases
Responsabilidades	Colaboradores

Alumno	
Alumn@s de la institución	
Persona	--
Responsabilidades	Colaboradores
Inscribirse a curso	Curso
Modificar datos	
Mostrar datos	
Dar examen	Profesor, Materia, Curso
Concurrir a clase	Asistencia

REPASANDO CONCEPTOS

Los objetos son personas, lugares o cosas relevantes para el sistema a analizar. Los sistemas orientados a objetos describen las entidades como objetos. Algunos objetos comunes son clientes, artículos, pedidos, etc. Los objetos también pueden ser pantallas de GUI o áreas de texto en la pantalla.

Los objetos se representan y agrupan mediante clases, las cuales son óptimas para la reutilización y la facilidad de mantenimiento. Una clase define el conjunto de atributos compartidos y comportamientos que se encuentran en cada objeto de la clase. Un atributo describe cierta propiedad que poseen todos los objetos de la clase. Un método es una acción, un comportamiento, que se puede solicitar de cualquier objeto de la clase. Los métodos son los procesos que una clase sabe cómo llevar a cabo.

Al ejecutarse el programa se pueden crear objetos a partir de la clase establecida. El término instanciar se utiliza cuando se crea un objeto a partir de una clase.

Lo que distingue a la programación orientada a objetos (y por ende al análisis y diseño orientado a objetos) de la programación clásica es la técnica de colocar todos los atributos y métodos de un objeto dentro de una estructura autocontenida (encapsulamiento), la clase en sí.

Con la herencia se pueden crear clases derivadas de tal forma que hereden todos los atributos y comportamientos de una clase base.

DIAGRAMA DE CLASES

Un diagrama de clases describe la estructura estática de un sistema en términos de clases y de relaciones entre estas clases, mostrando los atributos y operaciones que caracterizan cada clase de objetos.

Un diagrama de objetos representa la estructura estática del sistema mostrando los objetos (instancias) en el sistema y las relaciones entre los objetos.

Un diagrama de clases dado corresponde a un conjunto infinito de diagramas de objetos. Se utilizan para modelar la vista de diseño estático de un sistema.

Por lo regular se utilizan para una de las siguientes cosas:

- Modelar el vocabulario del sistema
- Modelar colaboraciones simples
- Modelar el esquema lógico de la BD

El diagrama de clases puede utilizarse con distintos fines en distintas etapas del proceso de desarrollo.

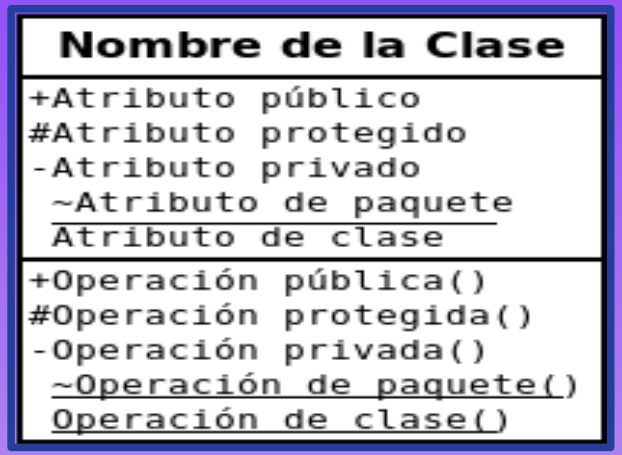
- Durante la etapa de análisis, el modelo de dominio es encargado de mostrar el conjunto de clases conceptuales del problema y las relaciones presentes entre sí.
- Durante la etapa de diseño, el modelo de diseño determina las futuras componentes de software (clases) y sus relaciones entre sí.

DIAGRAMA DE CLASES

Resumen de notación

Una clase se representa con un recuadro dividido en tres regiones

- El nombre de la clase
- Los atributos
- Las operaciones o métodos



Los atributos y operaciones pueden tener diferentes niveles de visibilidad:

- Público: visible por todos los clientes de la clase.
- Protegido: visible por las subclases de la clase.
- Privado: visible sólo para la clase.
- Paquete: visible para cualquier clase del mismo paquete.
- Atributos y operaciones estáticos (static) o de clase: son propios de la clase, no de la instancia. Son visibles por todos los objetos de la clase.

DIAGRAMA DE CLASES

COHESIÓN: “Relación lógica entre los atributos y los métodos de una misma clase”

La cohesión es una indicación cualitativa del grado que tiene un objeto para centrarse en una sola cosa.

La cohesión de una clase se determina examinando el grado en que el conjunto de propiedades que posee sea parte del diseño o dominio del problema.

Es decir, en la construcción de software se desea conseguir una alta cohesión. (highly cohesive)

ACOPLAMIENTO: “Relación lógica entre los atributos y los métodos de distintas clases (llamadas clases acopladas)”

El acoplamiento está representado por el número de conexiones físicas entre los elementos del diseño (clases, objetos, paquetes).

Por ejemplo, el número de colaboraciones entre clases o el número de mensajes intercambiados entre objetos.

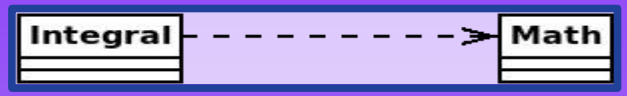
En el diseño de software la meta es conseguir un acoplamiento lo más bajo posible (loosely coupled).

De esta manera, los errores no se propagan tan fácilmente. Se evitan los daños colaterales cuando el software es modificado.

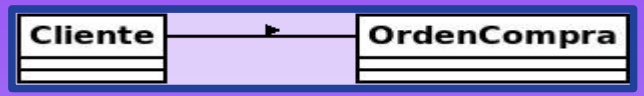
DIAGRAMA DE CLASES

Tipos de relaciones entre clases:

Dependencia: Una clase hace uso, o requiere visibilidad sobre otra.



Asociación: permite asociar objetos que colaboran entre si. Cabe destacar que no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro.

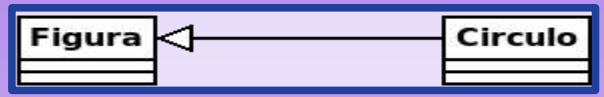


Agregación: Una clase esta contenida en otra pero puede ser compartida con otras clases. El tiempo de vida del objeto incluido es independiente del que lo incluye.

Composición: una clase se compone de otras clases a las cuales posee en exclusividad. Es responsabilidad de la clase contenedora la creación y destrucción de las contenidas.

Generalización: Una clase es un tipo de otra

Herencia: Una clase extiende a la otra con atributos y métodos. Las subclases heredan de la superclase.



Especialización: Una clase especializa la funcionalidad de la clase raíz.

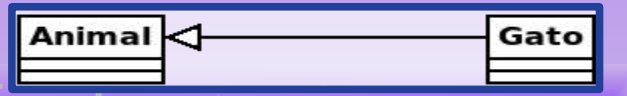
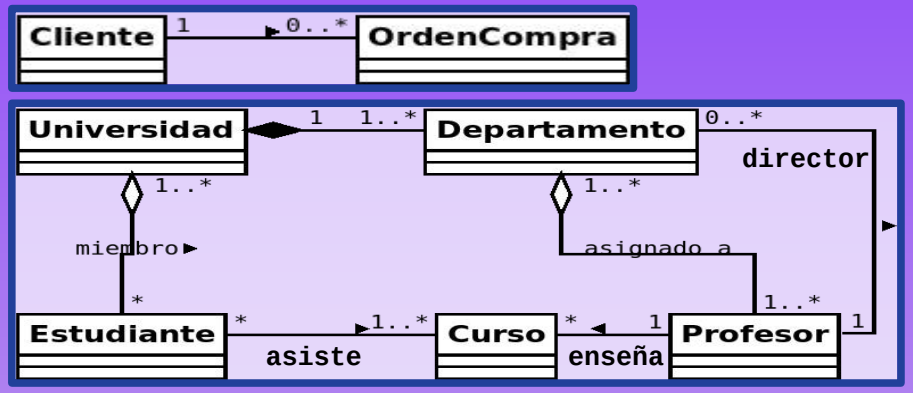


DIAGRAMA DE CLASES

Cardinalidad o multiplicidad de relaciones: En UML, la cardinalidad de las relaciones indica el grado y nivel de dependencia, se anotan en cada extremo de la relación y éstas pueden ser:

1	Una y solo una
0..1	Cero o una
0..*	Cero o mas
1..*	Una o mas
n..m	De n a m (enteros)
-	Sin especificar



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

Nombre: Una asociación puede tener un nombre, que se utiliza para describir la naturaleza de la relación. Para evitar ambigüedades, se puede indicar una dirección al nombre, es decir, la dirección en que se debe leer el nombre.

DIAGRAMA DE PAQUETES

Los paquetes ofrecen un mecanismo general para la partición de los modelos y la organización de los elementos de modelado en grupos relacionados semánticamente.

Los paquetes ofrecen un mecanismo general para la partición de los modelos y la organización de los elementos de modelado en grupos relacionados semánticamente.

Para controlar el acceso a los elementos del paquete, se puede indicar la visibilidad de un elemento del paquete precediendo el nombre del elemento por un símbolo de visibilidad, los elementos con visibilidad pública son accesibles fuera del paquete y los elementos con visibilidad privada sólo están disponibles para elementos internos al paquete.

Los paquetes pueden importar elementos de otros paquetes, a través de una relación de dependencia entre paquetes con el estereotipo <<import>>. La relación de importación puede ser pública (por defecto, <<import>>) o privada (<<access>>).

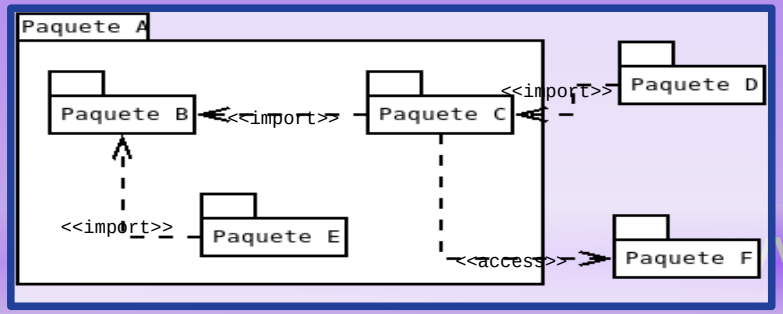


DIAGRAMA DE ESTADOS

Un Diagrama de Estados muestra la secuencia de estados por los que pasa un caso de uso o un objeto a lo largo de su vida, indicando qué eventos hacen que se pase de un estado a otro y cuáles son las respuestas y acciones que genera.

En cuanto a la representación, un diagrama de estados es un grafo cuyos nodos son estados y cuyos arcos dirigidos son transiciones etiquetadas con los nombres de los eventos. Un estado se representa como una caja redondeada con el nombre del estado en su interior. Una transición se representa como una flecha desde el estado origen al estado destino.

La caja de un estado puede tener 1 o 2 compartimentos. En el primer compartimento aparece el nombre del estado. El segundo compartimento es opcional, y en él pueden aparecer acciones de entrada, de salida y acciones internas.

Una acción de entrada aparece en la forma entrada/acción_asociada donde acción_asociada es el nombre de la acción que se realiza al entrar en ese estado. Cada vez que se entra al estado por medio de una transición la acción de entrada se ejecuta.

Una acción de salida aparece en la forma salida/acción_asociada. Cada vez que se sale del estado por una transición de salida la acción de salida se ejecuta.

DIAGRAMA DE ESTADOS

Una acción interna es una acción que se ejecuta cuando se recibe un determinado evento en ese estado, pero que no causa una transición a otro estado. Se indica en la forma nombre_de_evento/acción_asociada.

Los diagramas de estados muestran cómo reaccionan los objetos a los eventos y cómo cambian su estado interno. Un diagrama de estados puede representar ciclos continuos o bien una vida finita, en la que hay un estado inicial de creación y un estado final de destrucción (del caso de uso o del objeto). El estado inicial se muestra como un círculo sólido y el estado final como un círculo sólido rodeado de otro círculo. En realidad, los estados inicial y final son pseudoestados, pues un objeto no puede “estar” en esos estados, pero nos sirven para saber cuáles son las transiciones inicial y final(es).

El estado de un objeto puede ser caracterizado por el valor de uno o más atributos de su clase. Los valores de los atributos de un objeto y los enlaces que mantiene constituyen su estado.

Los casos de uso y los escenarios sirven para describir el comportamiento del sistema, es decir, la interacción entre los objetos participantes. Los diagramas de estado sirven para describir el comportamiento dentro de un objeto.

Se crean sólo para los objetos con comportamiento dinámico significativo.

DIAGRAMA DE ESTADOS

Una transición de estados representa un cambio de un estado origen a un estado sucesor, la transición puede estar acompañada de una acción.

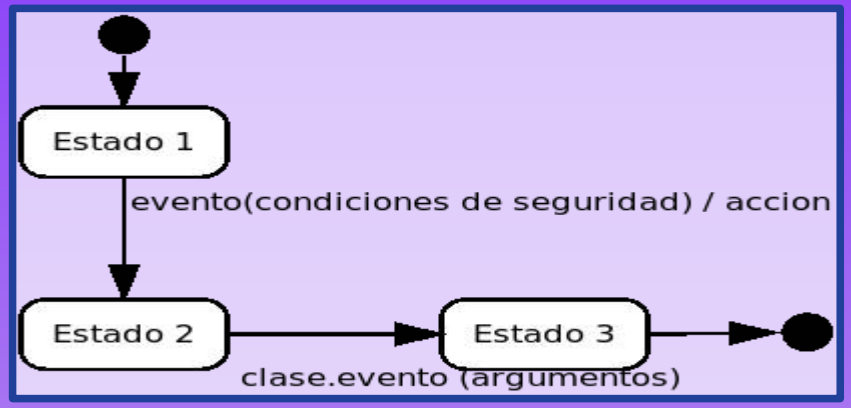
Pueden ser:

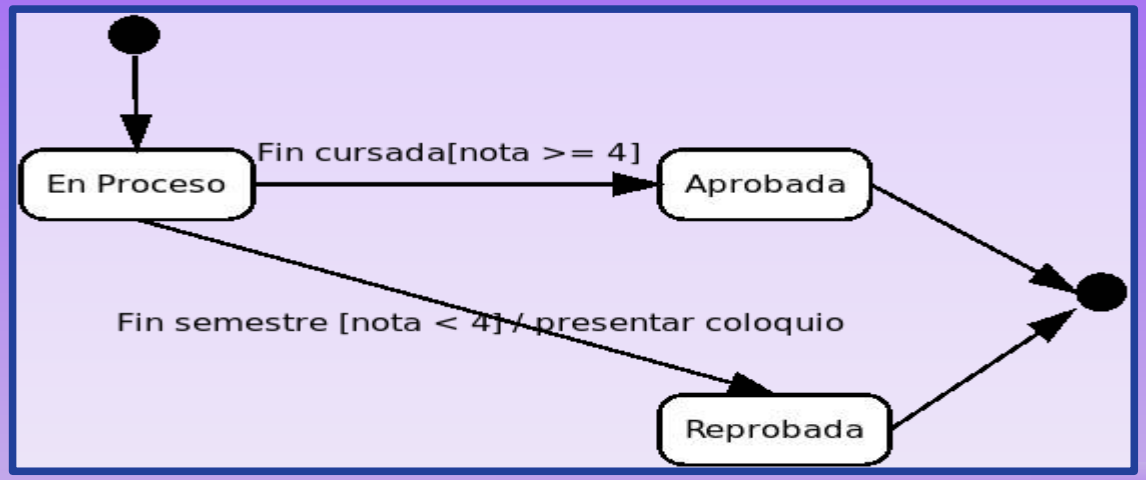
- Automática: Se da al terminar la actividad que origina el estado.
- No-automática: Se da a causa de un evento.
- Los dos tipos son inmediatas y no se pueden interrumpir.

Constan de cinco partes:

- Estado Origen: es el estado afectado por la transición
- Evento: El evento cuya recepción por parte del objeto en el estado origen, hace posible el disparo de la transición si la condición de seguridad se cumple.
- Condición de Seguridad: Expresión booleana que se evalúa cuando se recibe el evento, si es verdadera, la transición se dispara.
- Acción: Operación atómica ejecutable que puede actuar directamente en el objeto dueño de la máquina de estado e indirectamente en otros objetos visibles por el objeto.
- Estado Destino: El estado que está activo después de completarse la transición.

DIAGRAMA DE ESTADOS





Para mostrar explícitamente la sincronización de dos (o más) actividades en concurrencia se usan los pseudoestados fork (división de control) y join (fusión de control).

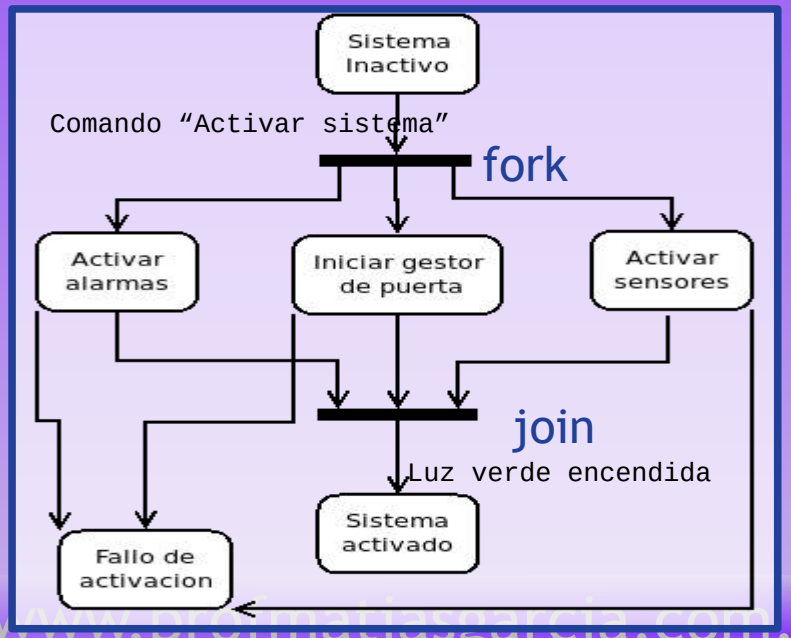


DIAGRAMA DE SECUENCIA

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo. Esta descripción es importante porque puede dar detalle a los casos de uso, aclarándolos al nivel de mensajes de los objetos existentes, como también muestra el uso de los mensajes de las clases diseñadas en el contexto de una operación.

El eje vertical representa el tiempo, y en el eje horizontal se colocan los objetos y actores participantes en la interacción, sin un orden prefijado. Cada objeto o actor tiene una línea vertical, y los mensajes se representan mediante flechas entre los distintos objetos. El tiempo fluye de arriba abajo.

Muestra las interacciones de los objetos ordenados en una secuencia de tiempo.

Muestra los objetos y las clases involucradas en el escenario y la secuencia de mensajes intercambiados entre los objetos necesaria para llevar a cabo la funcionalidad del escenario.

El envío y la recepción de un mensaje se consideran normalmente eventos.

DIAGRAMA DE SECUENCIA

Los mensajes pueden ser señales, llamadas a operación o algo similar.

Los distintos tipos de mensaje se representan con distintos símbolos para las flechas horizontales:

- Mensaje síncrono: Sólo desencadena una operación cuando el destinatario acepta el mensaje, y el emisor del mensaje se bloquea hasta ese momento.

- Mensaje asíncrono: Representa un envío de mensaje donde no hay retorno explícito y no se interrumpe la ejecución del emisor.

- Mensaje de retorno: Es una representación opcional para mostrar explícitamente que el flujo de control vuelve al emisor original del mensaje.

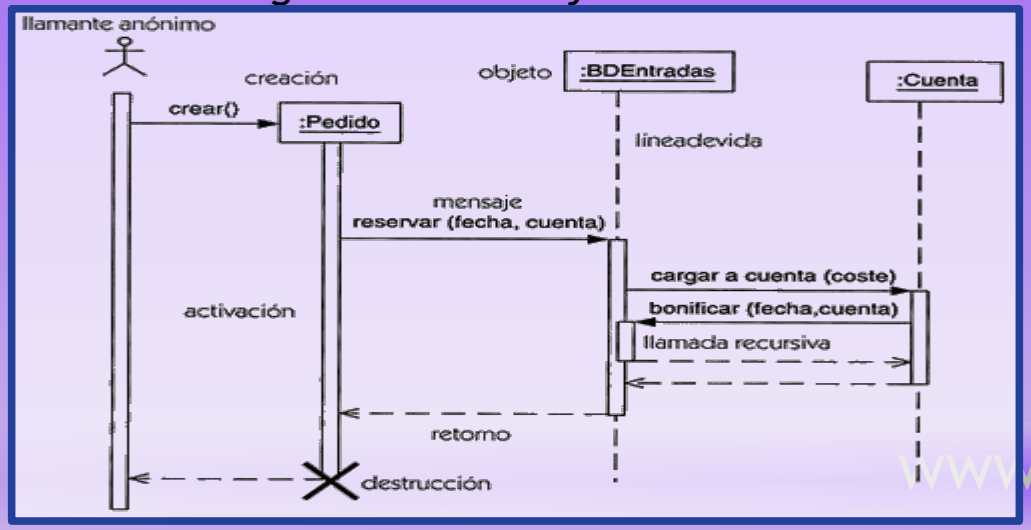


DIAGRAMA DE SECUENCIA

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

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

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

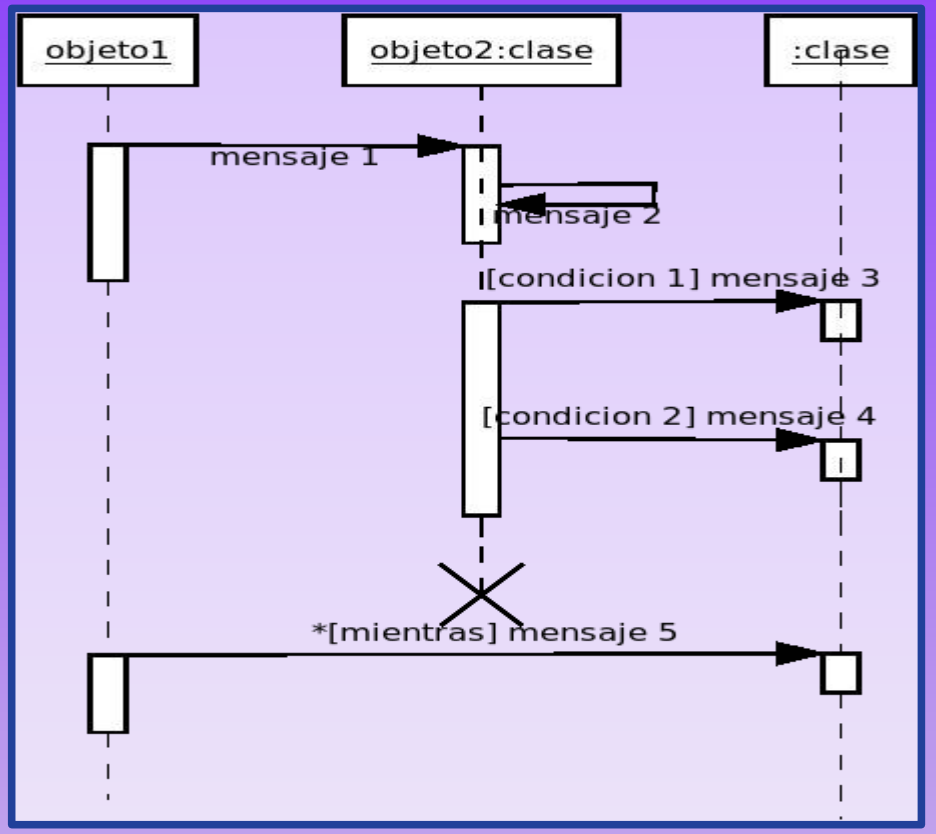


DIAGRAMA DE COLABORACIÓN

Un Diagrama de Colaboración muestra una interacción organizada basándose en los objetos que toman parte en la interacción y los enlaces entre los mismos (en cuanto a la interacción se refiere).

A diferencia de los Diagramas de Secuencia, los Diagramas de Colaboración muestran las relaciones entre los roles de los objetos. La secuencia de los mensajes y los flujos de ejecución concurrentes deben determinarse explícitamente mediante números de secuencia.

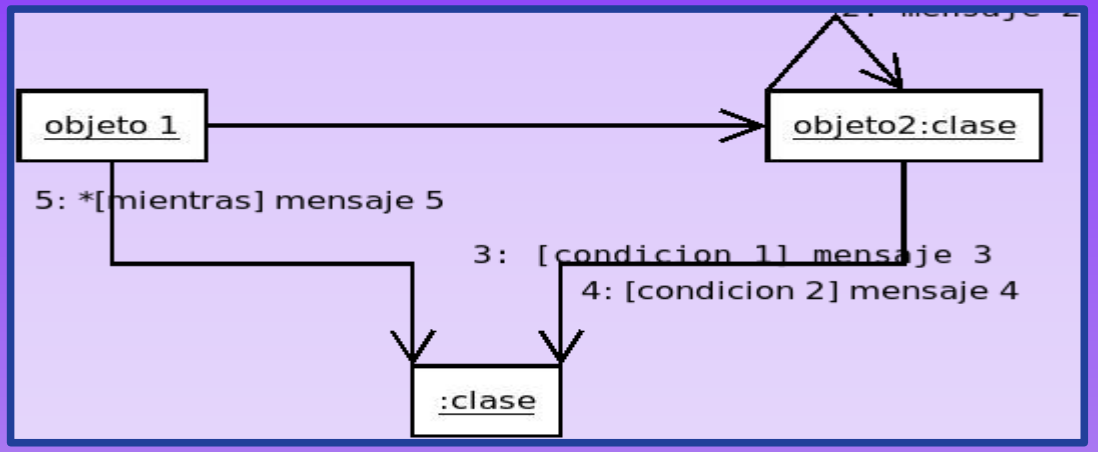
En cuanto a la representación, un Diagrama de Colaboración muestra una serie de objetos con los enlaces entre los mismos, y con los mensajes que se intercambian dichos objetos. Los mensajes son flechas que van junto al enlace por el que “circulan”, y con el nombre del mensaje y los parámetros (si los tiene) entre paréntesis.

Cada mensaje lleva un número de secuencia que denota cuál es el mensaje que le precede, excepto el mensaje que inicia el diagrama, que no lleva número de secuencia. Se pueden indicar alternativas con condiciones entre corchetes (por ejemplo 3 [condición_de_test] : nombre_de_método()). También se puede mostrar el anidamiento de mensajes con números de secuencia como 2.1, que significa que el mensaje con número de secuencia 2 no acaba de ejecutarse hasta que no se han ejecutado todos los 2. x .

DIAGRAMA DE COLABORACIÓN

Muestra a los objetos con sus relaciones entre sí, además de los mensajes que se intercambian entre ellos.

En una colaboración se muestran solamente los aspectos que son necesarios para explicar un comportamiento particular, suprimiendo el resto.



Los objetos de una colaboración no le pertenecen, pueden existir antes y después de la colaboración.

Aunque los objetos de una colaboración estén enlazados, no se comunican necesariamente fuera de la colaboración.

Un diagrama de colaboración es una forma de representar interacción entre objetos, alterna al diagrama de secuencia. A diferencia de los diagramas de secuencia, pueden mostrar el contexto de la operación (cuáles objetos son atributos, cuáles temporales, ...) y ciclos en la ejecución.

DIAGRAMA DE COLABORACIÓN

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

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

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

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

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

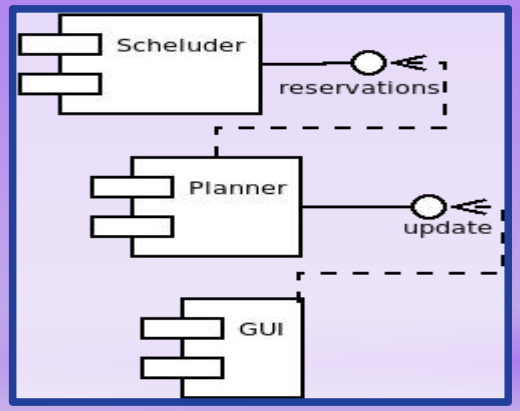
DIAGRAMA DE IMPLEMENTACIÓN

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

Diagrama de Componentes

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

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



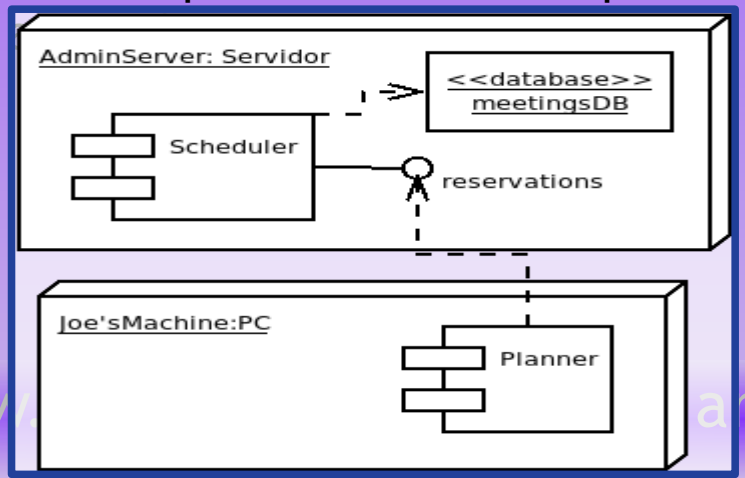
En este caso tenemos tres componentes, GUI dependiendo de la interfaz *update* provista por *Planner*, *Planner* dependiendo de la interfaz *reservations* provista por *Scheduler*.

DIAGRAMA DE IMPLEMENTACIÓN

Diagrama de Ejecución

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

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



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

DIAGRAMA DE IMPLEMENTACIÓN

- Nodos

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

- Componentes

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

UML PARA DESARROLLAR

En un principio el analista usará el conjunto de herramientas de UML para descomponer los requerimientos del sistema en un modelo de casos de uso y en un modelo de objetos.

1. Definir el modelo de casos de uso.

- Buscar los actores en el dominio del problema; revisar los requerimientos del sistema y entrevistar algunos expertos de negocios.
- Identificar los eventos principales iniciados por los actores y desarrollar un conjunto de casos de uso primarios en un nivel muy alto, que describan los eventos desde la perspectiva de cada actor.
- Desarrollar los diagramas de casos de uso para comprender cómo se relacionan los actores con los casos de uso que definirán el sistema.
- Refinar los casos de uso primarios para desarrollar una descripción detallada de la funcionalidad del sistema para cada uno. Proveer detalles adicionales al desarrollar los escenarios de casos de uso que documenten los flujos alternos de los casos de uso primarios.
- Revisar los escenarios de casos de uso con los expertos del área de negocios para verificar los procesos e interacciones. Hacer las modificaciones necesarias hasta que los expertos del área de negocios estén de acuerdo en que los escenarios de los casos de uso son completos y precisos.

UML PARA DESARROLLAR

2. Continuar con la creación de diagramas de UML para modelar el sistema durante la fase de análisis de sistemas.

- Derivar los diagramas de actividad a partir de los diagramas de casos de uso.
- Desarrollar los diagramas de secuencia y de comunicación a partir de los escenarios de los casos de uso.
- Revisar los diagramas de secuencia con los expertos del área de negocios para verificar los procesos y las interacciones. Hacer las modificaciones necesarias hasta que los expertos del área de negocios estén de acuerdo en que los diagramas de secuencia son completos y precisos. Con frecuencia, la revisión adicional de los diagramas de secuencia gráficos provee a los expertos del área de negocios la oportunidad de reconsiderar y refinar los procesos con un detalle más atómico que la revisión de los escenarios de los casos de uso.

3. Desarrollar los diagramas de clases.

- Buscar los sustantivos en los casos de uso y hacer una lista. Son objetos potenciales. Una vez que se identifican los objetos, buscar similitudes y diferencias en los objetos debido a sus estados o comportamiento y después crear las clases.
- Definir las principales relaciones entre las clases. Buscar las relaciones “tiene un” y “es un” entre las clases.

UML PARA DESARROLLAR

4. Dibujar diagramas de estados.

- Desarrollar diagramas de estados para ciertos diagramas de clase, de manera que pueda proveer un análisis más detallado del sistema en este punto. Usar los diagramas de estados como ayuda para comprender los procesos complejos que no se puedan derivar por completo de los diagramas de secuencia.
- Determinar los métodos al examinar los diagramas de estados. Derivar los atributos de las clases de los estados (datos) a partir de los casos de uso, expertos de las áreas de negocios y métodos de las clases.

5. Diseño de sistemas; refinar los diagramas de UML y utilizarlos para derivar las clases y sus atributos y métodos.

- Revisar todos los diagramas de UML existentes para el sistema. Escribir especificaciones de clases para cada clase que incluyan los atributos, métodos y descripciones de la clase. Revisar los diagramas de secuencia para identificar otros métodos de clases.
- Desarrollar especificaciones de métodos que muestren con detalle los requerimientos de entrada y de salida para el método, junto con una descripción detallada del procesamiento interno del método.

UML PARA DESARROLLAR

- Crear otro conjunto de diagramas de secuencia (si es necesario) para reflejar los métodos reales de las clases, además las interacciones entre las clases y las interfaces del sistema.
- Crear diagramas de clases mediante los símbolos de clase especializados para la clase de límite o de interfaz, la clase de entidad y la clase de control.
- Analizar los diagramas de clases para derivar los componentes del sistema; es decir, clases relacionadas en función y lógica que se compilarán y desplegarán en conjunto como archivos.
- Desarrollar diagramas de despliegue para indicar cómo se desplegarán los componentes de su sistema en el entorno de producción.

6. Documentar el diseño de su sistema en forma detallada. Este paso es imprescindible. Entre más completa sea la información que provea al equipo de desarrollo por medio de la documentación y los diagramas de UML, más rápido será el desarrollo y más sólido será el sistema de producción final.

UML FUTURO

Tendencias de Futuro

- Nueva versión
- UML 2.5 (2020)
- Extensiones de UML
- SysML - Systems Modeling Language (SysML, www.sysml.org)
- Generación automática de código a partir de modelos
- Model-Driven Engineering (MDE)
- Extendiendo UML mediante perfiles
- MARTE – Modeling and Analysis of Real-Time Embedded Systems
- Entornos avanzados basados en metamodelado
- Eclipse Modeling Framework (EMF)
- Modelado y generación de código en dominios específicos
- Domain-Specific Modeling (DSM, www.dsmforum.org)

BIBLIOGRAFÍA Y LICENCIA

- Kendall Kenneth y Kendall Julie. “Análisis y Diseño de Sistemas” (8va Ed). Prentice Hall, 2011,México.
- Rumbaugh J., Jacobson I. y Booch G.. “El Lenguaje Unificado de Modelado Manual de Referencia”. Addison Wesley, 2007, España.
- Weitzenfeld Alfred. “Ingeniería de Software Orientada a Objetos con UML, Java e Internet”. Thomson. 2007, España.

- Autor:

Matías E. García

Prof. & Tec. en Informática Aplicada
www.profmatiasgarcia.com.ar
info@profmatiasgarcia.com.ar