Arquitectura de software

En los inicios de la informática, la programación se consideraba un arte y se desarrollaba como tal debido a la dificultad que entrañaba para la mayoría de las personas, pero con el tiempo se han ido descubriendo y desarrollando formas y guías generales, con base a las cuales se puedan resolver los problemas. A estas, se les ha denominado arquitectura de software, porque, a semejanza de los planos de un edificio o construcción, estas indican la estructura, funcionamiento e interacción entre las partes del software. En el libro "An introduction to Software Architecture", David Garlan y Mary Shaw definen que la arquitectura es un nivel de diseño que hace foco en aspectos "más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema".

La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema.

- Una arquitectura de software, también denominada arquitectura lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan un marco definido y claro para interactuar con el código fuente del software.
- Una arquitectura de software se selecciona y diseña con base en objetivos (requisitos) y restricciones. Los objetivos son aquellos prefijados para el sistema de información, pero no solamente los de tipo funcional, también otros objetivos como el mantenimiento, la auditoria, flexibilidad e interacción con otros sistemas de información. Las restricciones son aquellas limitaciones derivadas de las tecnologías disponibles para implementar sistemas de información. Unas arquitecturas son más recomendables de implementar con ciertas tecnologías mientras que otras tecnologías no son aptas para determinadas arquitecturas. Por ejemplo, no es viable emplear una arquitectura de software de tres capas para implementar sistemas en tiempo real.

- La arquitectura de software define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos. Toda arquitectura debe ser implementable en una arquitectura física, que consiste simplemente en determinar qué computadora tendrá asignada cada tarea.

Toda arquitectura de software debe describir diversos aspectos del software. Generalmente, cada uno de estos aspectos se describe de una manera más comprensible si se utilizan distintos modelos o vistas. Es importante destacar que cada uno de ellos constituye una descripción parcial de una misma arquitectura y es deseable que exista cierto solapamiento entre ellos. Esto es así porque todas las vistas deben ser coherentes entre sí, evidente dado que describen la misma cosa.

Cada paradigma de desarrollo exige diferente número y tipo de vistas o modelos para describir una arquitectura. No obstante, existen al menos tres vistas absolutamente fundamentales en cualquier arquitectura:

- La visión estática: describe qué componentes tiene la arquitectura.
- La visión funcional: describe qué hace cada componente.
- La visión dinámica: describe cómo se comportan los componentes a lo largo del tiempo y como interactúan entre sí.

Las vistas o modelos de una arquitectura de software pueden expresarse mediante uno o varios lenguajes. El más obvio es el lenguaje natural, pero existen otros lenguajes tales como los diagramas de estado, los diagramas de flujo de datos, etc. Estos lenguajes son apropiados únicamente para un modelo o vista. Afortunadamente existe cierto consenso en adoptar UML (Unified Modeling Language, lenguaje unificado de modelado) como lenguaje único para todos los modelos o vistas. Sin embargo, un lenguaje generalista corre el peligro de no ser capaz de describir determinadas restricciones de un sistema de información (o expresarlas de manera incomprensible).

Generalmente, no es necesario inventar una nueva arquitectura de software para cada sistema de información. Lo habitual es adoptar una arquitectura conocida en función de sus ventajas e inconvenientes para cada caso en concreto. Así, las arquitecturas más universales son:

- Descomposición Modular. Donde el software se estructura en grupos funcionales muy acoplados.
- Cliente-servidor. Donde el software reparte su carga de cómputo en dos partes independientes pero sin reparto claro de funciones.
- Arquitectura de tres niveles. Especialización de la arquitectura cliente-servidor donde la carga se divide en tres partes (o capas) con un reparto claro de funciones: una capa para la presentación (interfaz de usuario), otra para el cálculo (donde se encuentra modelado el negocio) y otra para el almacenamiento (persistencia). Una capa solamente tiene relación con la siguiente.

Otras arquitecturas menos conocidas son:

- Modelo Vista Controlador.
- En pipeline.
- Entre pares.
- En pizarra.
- Orientada a servicios (SOA del inglés Service-Oriented Architecture).
- Arquitectura de microservicios (MSA del inglés MicroServices Architecture). Algunos consideran que es una especialización de una forma de implementar SOA.
- Dirigida por eventos.
- Máquinas virtuales

Qué es el lenguaje unificado de modelado (UML)

¿Qué es UML?

El Lenguaje Unificado de Modelado (UML) fue creado para forjar un lenguaje de modelado visual común y semántica y sintácticamente rico para la arquitectura, el diseño y la implementación de sistemas de software complejos, tanto en estructura como en comportamiento. UML tiene aplicaciones más allá del desarrollo de software, p. ej., en el flujo de procesos en la fabricación.

Es comparable a los planos usados en otros campos y consiste en diferentes tipos de diagramas. En general, los diagramas UML describen los límites, la estructura y el comportamiento del sistema y los objetos que contiene.

UML no es un lenguaje de programación, pero existen herramientas que se pueden usar para generar código en diversos lenguajes usando los diagramas UML. UML guarda una relación directa con el análisis y el diseño orientados a objetos.

UML y su función en el modelado y diseño orientados a objetos

Hay muchos paradigmas o modelos para la resolución de problemas en la informática, que es el estudio de algoritmos y datos. Hay cuatro categorías de modelos para la resolución de problemas: lenguajes imperativos, funcionales, declarativos y orientados a objetos (OOP). En los lenguajes orientados a objetos, los algoritmos se expresan definiendo 'objetos' y haciendo que los objetos interactúen entre sí. Esos objetos son cosas que deben ser manipuladas y existen en el mundo real. Pueden ser edificios, artefactos sobre un escritorio o seres humanos.

Los lenguajes orientados a objetos dominan el mundo de la programación porque modelan los objetos del mundo real. UML es una combinación de varias notaciones orientadas a objetos: diseño orientado a objetos, técnica de modelado de objetos e ingeniería de software orientada a objetos.

UML usa las fortalezas de estos tres enfoques para presentar una metodología más uniforme que sea más sencilla de usar. UML representa buenas prácticas para la construcción y documentación de diferentes aspectos del modelado de sistemas de software y de negocios.

La historia y los orígenes de UML

"The Three Amigos" (los tres amigos) de la ingeniería de software, como se los conocía, habían desarrollado otras metodologías. Se asociaron para brindar claridad a los programadores creando nuevos estándares. La colaboración entre Grady, Booch y Rumbaugh fortaleció los tres métodos y mejoró el producto final.

Los esfuerzos de estos pensadores derivaron en la publicación de los documentos UML 0.9 y 0.91 en 1996. Pronto se hizo evidente que varias organizaciones, incluidas Microsoft, Oracle e IBM, consideraron que UML era esencial para su propio desarrollo de negocios. Ellos, junto con muchas otras personas y compañías, establecieron los recursos necesarios para desarrollar un lenguaje de modelado hecho y derecho. "Los tres amigos" publicaron la Guía del usuario para el Lenguaje Unificado de Modelado en 1999, y una actualización que incluye información sobre UML 2.0 en la segunda edición de 2005.

OMG: Tiene un significado diferente

Según su sitio web, el Object Management Group® (OMG®) es un consorcio internacional sin fines de lucro y de membresía abierta para estándares tecnológicos, fundado en 1989. Los estándares de OMG son promovidos por proveedores, usuarios finales, instituciones académicas y agencias gubernamentales. Los grupos de trabajo de OMG desarrollan estándares de integración empresarial para una amplia gama de tecnologías y una gama incluso más amplia de industrias. Los estándares de modelado de OMG, incluidos UML y Model Driven Architecture® (MDA®), permiten un eficaz diseño visual, ejecución y mantenimiento de software y otros procesos.

OMG supervisa la definición y el mantenimiento de las especificaciones de UML. Esta supervisión ofrece a los ingenieros y programadores la capacidad de usar un lenguaje para muchos propósitos durante todas las etapas del ciclo de vida del software en sistemas de cualquier tamaño.

La finalidad de UML según OMG

El OMG define los propósitos de UML de la siguiente manera:

- Brindar a arquitectos de sistemas, ingenieros y desarrolladores de software las herramientas para el análisis, el diseño y la implementación de sistemas basados en software, así como para el modelado de procesos de negocios y similares.
- Hacer progresar el estado de la industria permitiendo la interoperabilidad de herramientas de modelado visual de objetos. No obstante, para habilitar un intercambio significativo de información de modelos entre herramientas, se requiere de un acuerdo con respecto a la semántica y notación.

UML cumple con los siguientes requerimientos:

- Establecer una definición formal de un metamodelo común basado en el estándar MOF (Meta-Object Facility) que especifique la sintaxis abstracta del UML. La sintaxis abstracta define el conjunto de conceptos de modelado UML, sus atributos y sus relaciones, así como las reglas de combinación de estos conceptos para construir modelos UML parciales o completos.

- Brindar una explicación detallada de la semántica de cada concepto de modelado UML. La semántica define, de manera independiente a la tecnología, cómo los conceptos UML se habrán de desarrollar por las computadoras.
- Especificar los elementos de notación de lectura humana para representar los conceptos individuales de modelado UML, así como las reglas para combinarlos en una variedad de diferentes tipos de diagramas que corresponden a diferentes aspectos de los sistemas modelados.
- Definir formas que permitan hacer que las herramientas UML cumplan con esta especificación. Esto se apoya (en una especificación independiente) con una especificación basada en XML de formatos de intercambio de modelos correspondientes (XMI) que deben ser concretados por herramientas compatibles.

UML y el modelado de datos

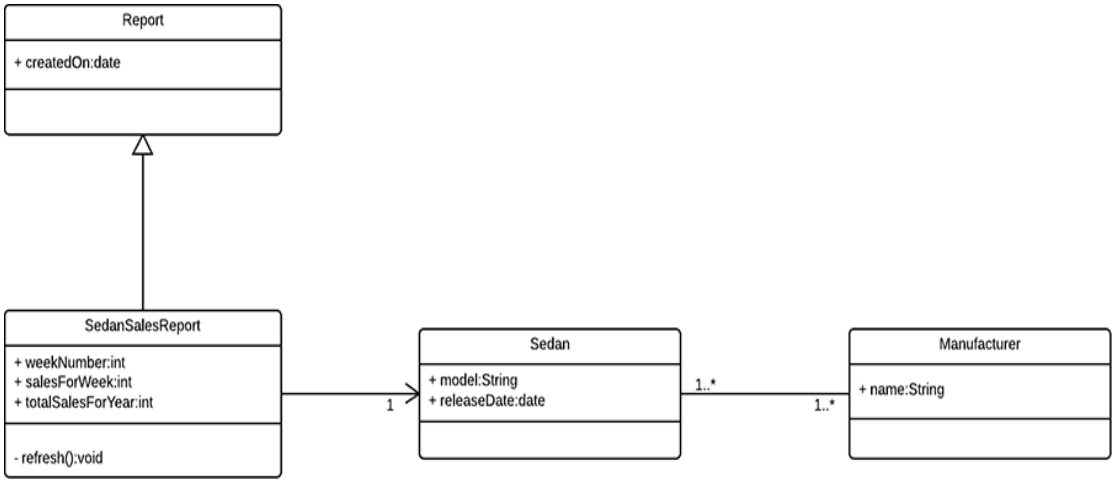
El UML es popular entre programadores, pero no suele ser usado por desarrolladores de bases de datos. Una razón es sencillamente que los creadores de UML no se enfocaron en las bases de datos. A pesar de ello, el UML es efectivo para el modelado de alto nivel de datos conceptuales y se puede usar en diferentes tipos de diagramas UML. Puedes encontrar información sobre la multidimensionalidad de un modelo de clases orientado a objetos en una base de datos relacional en este artículo sobre Modelado de bases de datos en UML.

Crear diagramas es rápido y sencillo con Lucidchart. Inicia una prueba gratuita hoy mismo para empezar a crear y colaborar.

DIAGRAMAS DE CLASES

Los diagramas de clases representan las estructuras estáticas de un sistema, incluidas sus clases, atributos, operaciones y objetos. Un diagrama de clases puede mostrar datos computacionales u organizacionales en la forma de clases de implementación y clases lógicas, respectivamente. Puede haber superposición entre estos dos grupos.

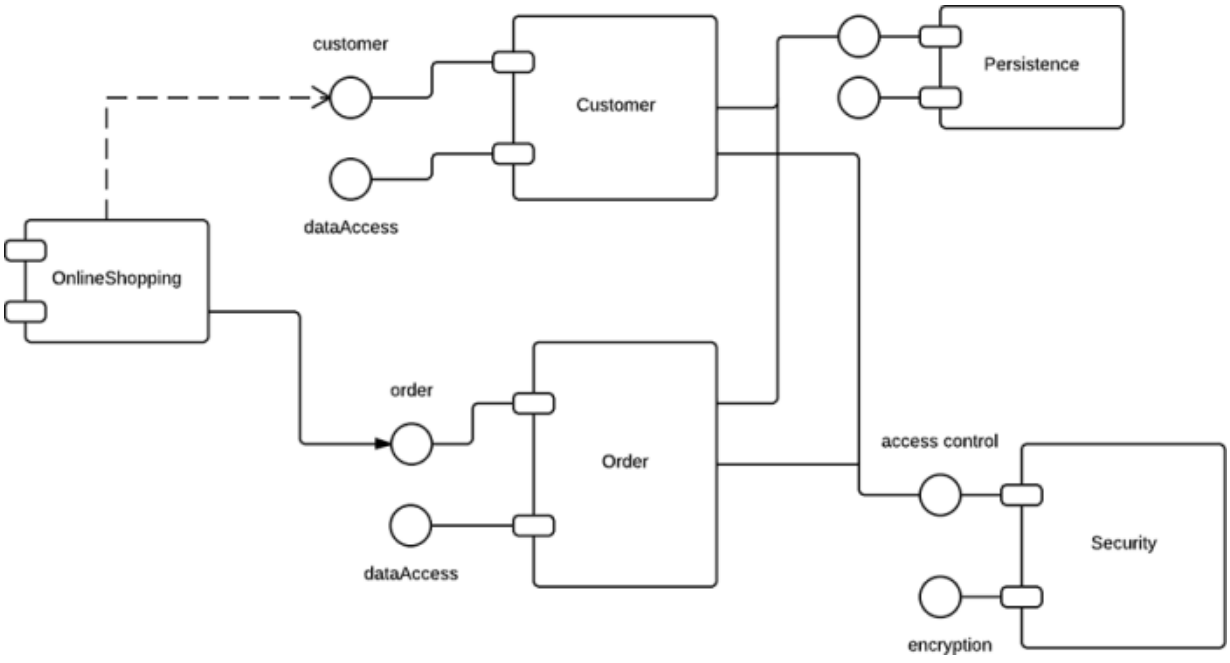
1. Las clases se representan con una forma rectangular dividida en tercios. La sección superior muestra el nombre de la clase, mientras que la sección central contiene los atributos de la clase. La sección inferior muestra las operaciones de la clase (también conocidas como métodos).
2. Agrega formas de clases a tu diagrama de clases para modelar la relación entre esos objetos. Además, podría ser necesario que agregues subclases.
3. Usa líneas para representar asociación, traspaso, multiplicidad y otras relaciones entre clases y subclases. Tu estilo de notación preferido informará la notación de estas líneas.



DIAGRAMAS DE COMPONENTES

Los diagramas de componentes muestran cómo se combinan los componentes para formar componentes más grandes o sistemas de software. Estos diagramas están diseñados para modelar las dependencias de cada componente en el sistema. Un componente es algo necesario para ejecutar una función de estereotipo. Un estereotipo de componente puede constar de ejecutables, documentos, tablas de bases de datos, archivos o archivos de bibliotecas.

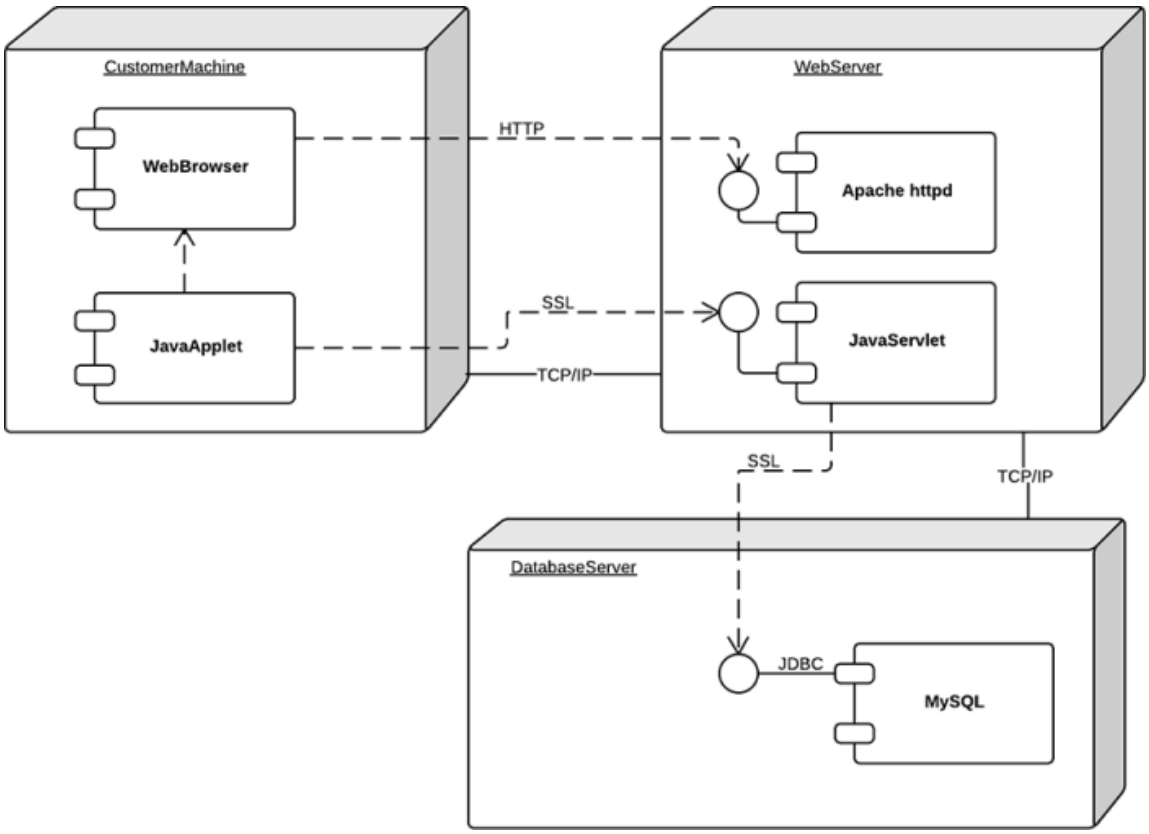
1. Representa un componente con una forma rectangular. Debe tener dos rectángulos pequeños en un lado o mostrar un icono con esa forma.
2. Agrega líneas entre formas de componentes para representar las relaciones pertinentes.



DIAGRAMAS DE IMPLEMENTACIÓN

Un diagrama de implementación modela la implementación física y la estructura de los componentes de hardware. Los diagramas de implementación muestran dónde y cómo operarán los componentes de un sistema en conjunto con los demás.

1. Al trazar un diagrama de implementación, usa la misma notación que usas para un diagrama de componentes.
2. Usa un cubo 3D para modelar un nodo (lo cual representa una máquina física o máquina virtual).
3. Etiqueta el nodo con el mismo estilo que se usa para los diagramas de secuencia. Agrega otros nodos según sea necesario, luego conéctalos con líneas.



Ejemplos de tutoriales de diagramas de comportamiento

Diagrama de actividades

Los diagramas de actividades muestran el flujo de control de procedimiento entre objetos de clases, junto con procesos organizacionales, como los flujos de trabajo de negocios. Estos diagramas se integran con formas especializadas que luego se conectan con flechas. La notación establecida para los diagramas de actividades es similar a la de los diagramas de estados.

1. Empieza tu diagrama de actividades con un círculo negro.
2. Conecta el círculo a la primera actividad, la cual se modela con un rectángulo redondeado.

3. Ahora, conecta cada actividad a otras actividades con líneas que muestren el flujo paso a paso de todo el proceso.
4. También puedes probar usar carriles para representar los objetos que realizan cada actividad.

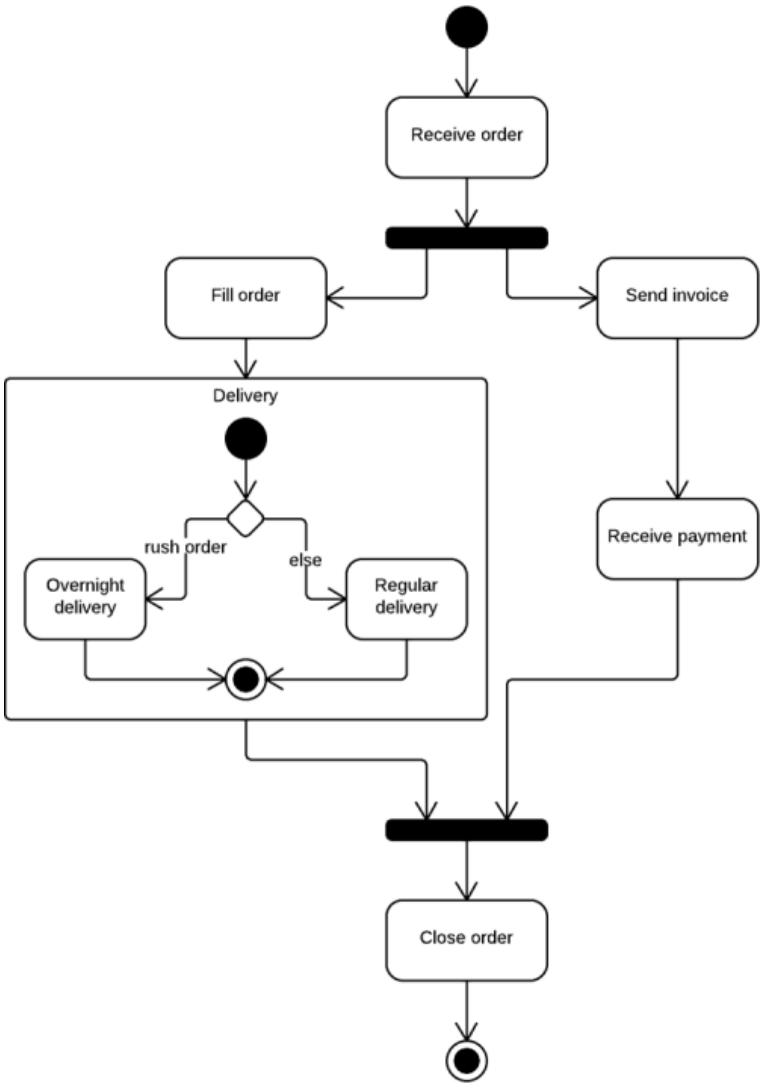


Diagrama de caso de uso

Un caso de uso es una lista de pasos que definen la interacción entre un actor (un humano que interactúa con el sistema o un sistema externo) y el sistema propiamente dicho. Los diagramas de casos de uso representan las especificaciones de un caso de uso y modelan las unidades funcionales de un sistema. Estos diagramas ayudan a los equipos de desarrollo a comprender los requisitos de su sistema, incluida la función de la interacción humana en el mismo y las diferencias entre diversos casos de uso. Un diagrama de caso de uso podría mostrar todos los casos de uso del sistema o solo un grupo de casos de uso con una funcionalidad similar.

1. Para iniciar un diagrama de casos de uso, agrega una forma ovalada en el centro del dibujo.
2. Escribe el nombre del caso de uso dentro del óvalo.

3. Representa a los actores con una figura humana cerca del diagrama, luego usa líneas para modelar las relaciones entre los actores y los casos de uso.

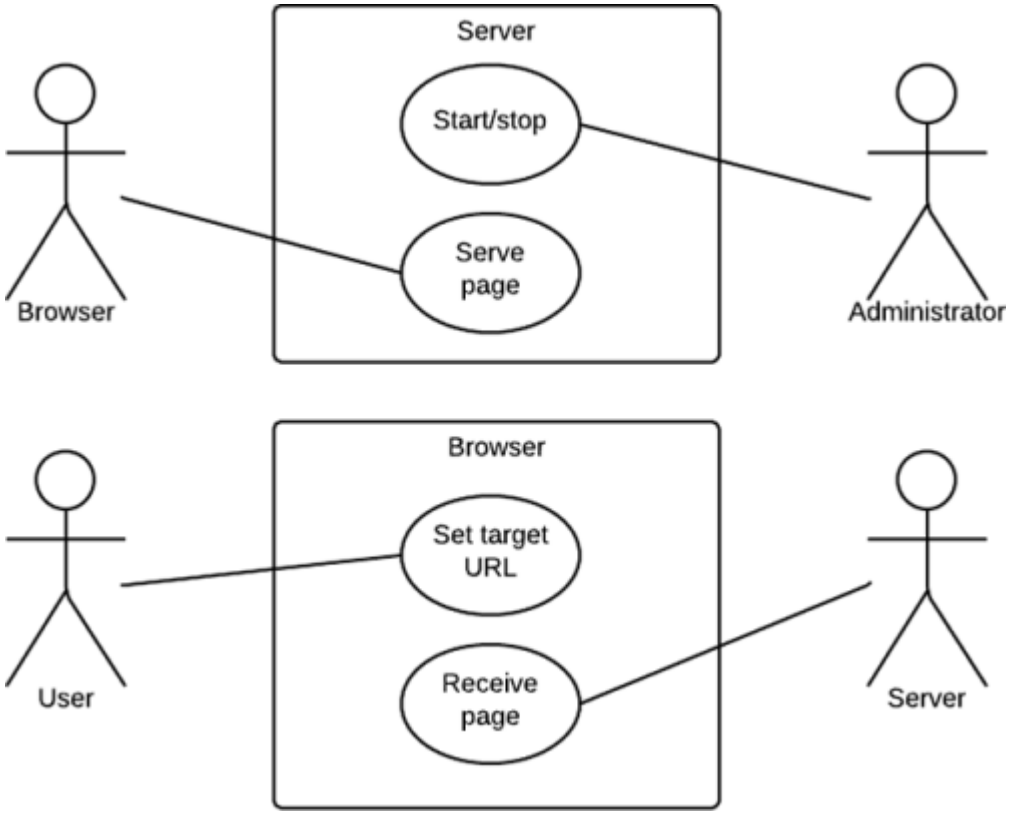
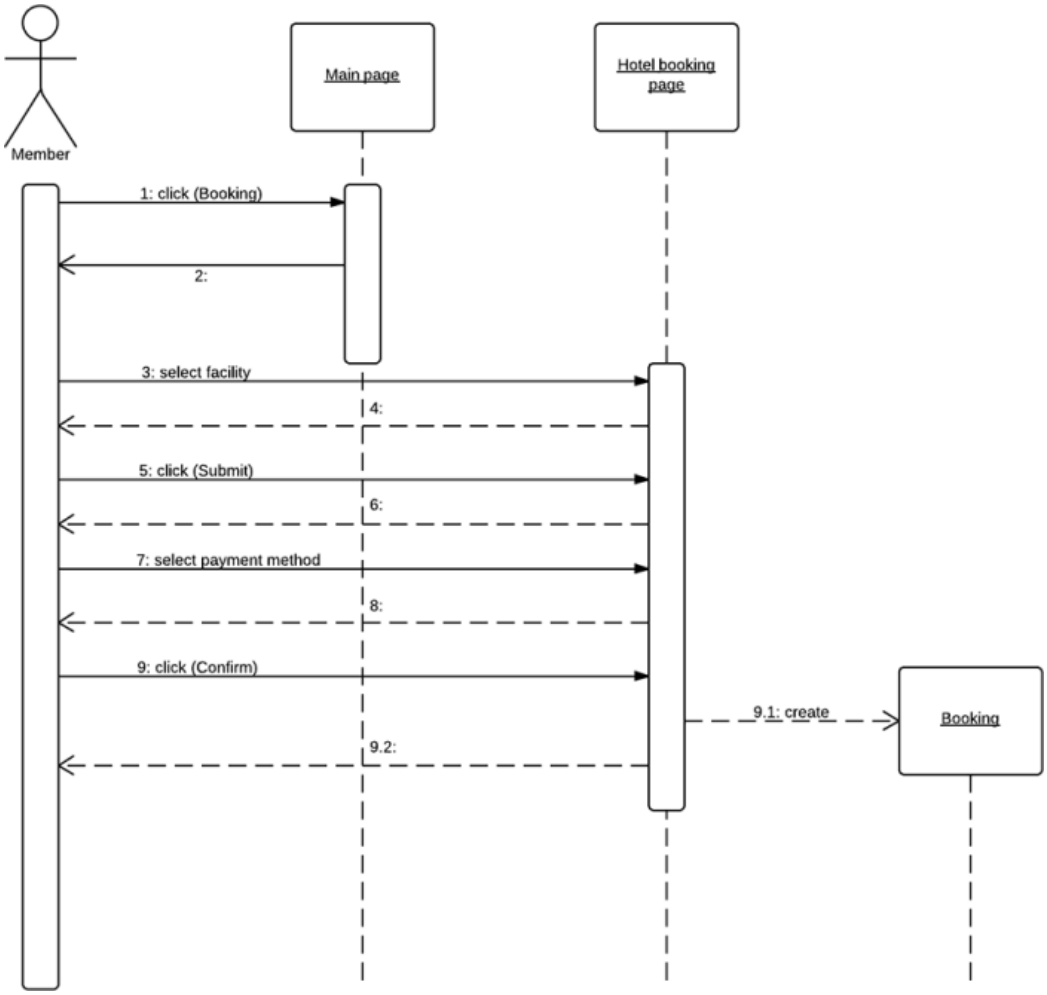


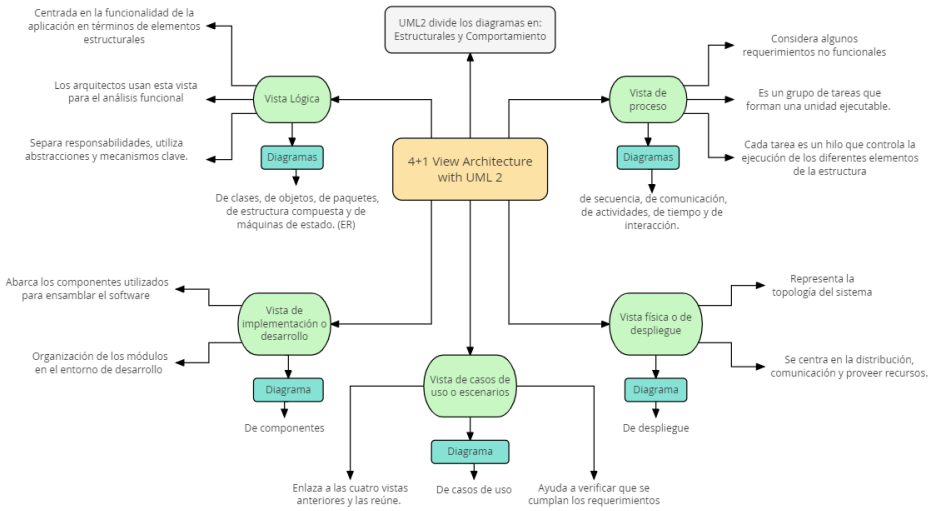
Diagrama de secuencia

Los diagramas de secuencia, también conocidos como diagramas de eventos o escenarios de eventos, ilustran cómo los procesos interactúan entre sí mostrando llamadas entre diferentes objetos en una secuencia. Estos diagramas tienen dos dimensiones: vertical y horizontal. Las líneas verticales muestran la secuencia de mensajes y llamadas en orden cronológico y los elementos horizontales muestran instancias de objetos en las que se transmiten los mensajes.

1. Para crear un diagrama de secuencia, escribe el nombre de la instancia de clase y el nombre de la clase en un cuadro rectangular.
2. Dibuja líneas entre las instancias de clases para representar al emisor y receptor de los mensajes.
3. Usa puntas de flecha oscuras para simbolizar mensajes sincrónicos, puntas de flecha abiertas para mensajes asincrónicos y líneas discontinuas para mensajes de respuesta.



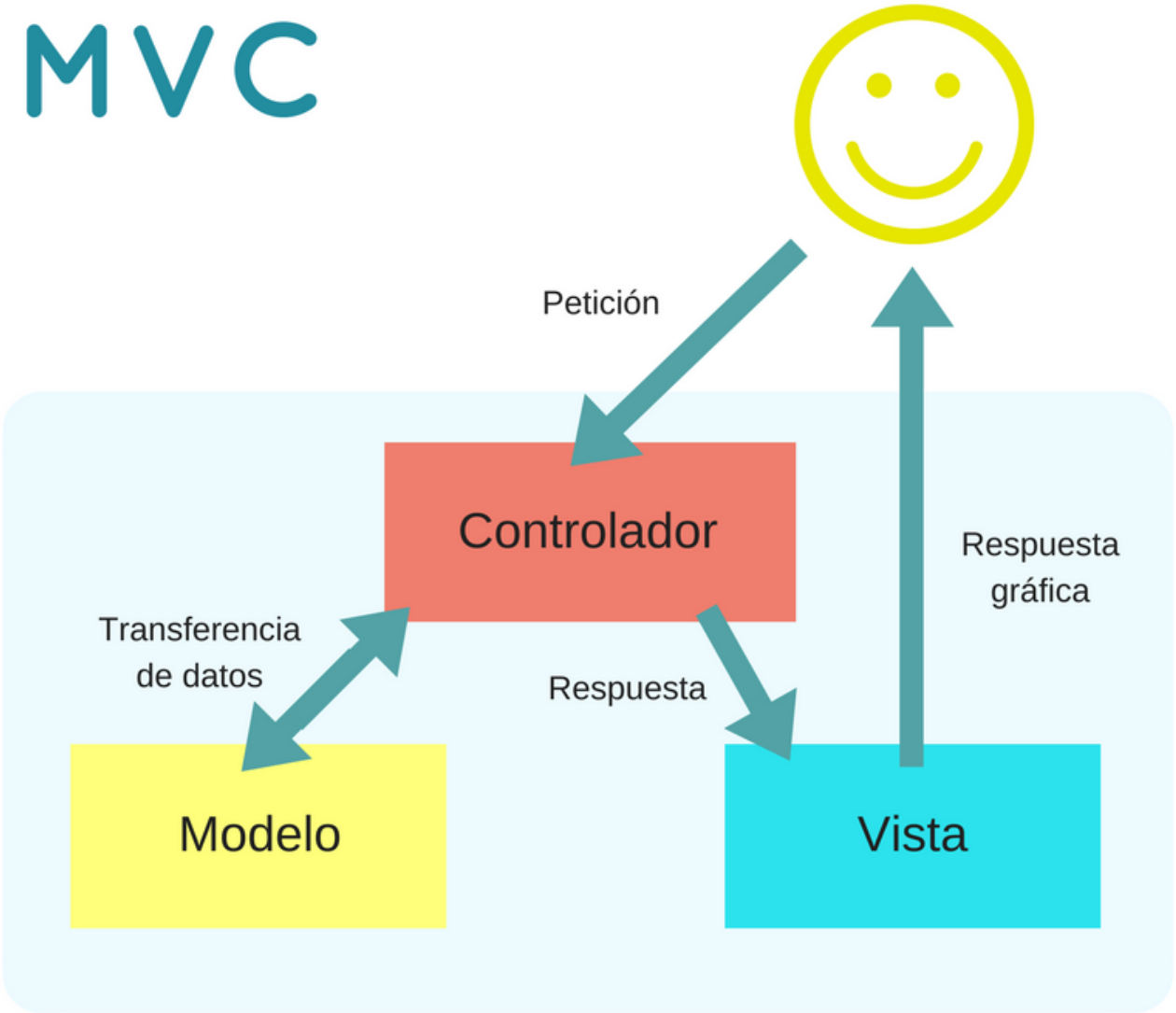
Arquitectura de software 4+1 UML



MVC (Modelo-Vista-Controlador): ¿qué es y para qué sirve?

El MVC es un patrón de diseño arquitectónico de software, que sirve para clasificar la información, la lógica del sistema y la interfaz que se le presenta al usuario. En este tipo de arquitectura existe un sistema central o controlador que gestiona las entradas y la salida del sistema, uno o varios modelos que se encargan de buscar los datos e información necesaria y una interfaz que muestra los resultados al usuario final. Es muy usado en el desarrollo web porque al tener que interactuar varios lenguajes para crear un sitio es muy fácil generar confusión entre cada componente si estos no son separados de la forma adecuada. Este patrón permite modificar cada uno de sus componentes si necesidad de afectar a los demás.

Componentes



Modelo: este componente se encarga de manipular, gestionar y actualizar los datos. Si se utiliza una base de datos aquí es donde se realizan las consultas, búsquedas, filtros y actualizaciones.

Vista: este componente se encarga de mostrarle al usuario final las pantallas, ventanas, páginas y formularios; el resultado de una solicitud. Desde la perspectiva del programador este componente es el que se encarga del frontend; la programación de la interfaz de usuario si se trata de un aplicación de escritorio, o bien, la visualización de las páginas web (CSS, HTML, HTML5 y Javascript).

Controlador: este componente se encarga de gestionar las instrucciones que se reciben, atenderlas y procesarlas. Por medio de él se comunican el modelo y la vista: solicitando los datos necesarios; manipulándolos para obtener los resultados; y entregándolos a la vista para que pueda mostrarlos.

Este patrón es uno de los más usados, en la actualidad se puede encontrar tanto en pequeños como en grandes sistemas, en el mundo laboral es indispensable llevarlo a la practica. Si apenas te estás iniciando en el mundo de la programación, te recomiendo que adoptes este modelo lo más pronto posible, para que en un futuro con sistemas más complejos no tengas ningún inconveniente.