**UNIVERSIDAD TECNOLÓGICA DE PANAMÁ**

**FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES**

**DEPARTAMENTO DE INGENIERÍA DE SOFTWARE**

**GUÍA DE ACTIVIDAD N°5**

**FC-FISC-1-8-2016**

Facilitador(a): ***Ing. Vanessa Castillo, M.Sc.*** ***Asignatura: Ingeniería de Software II***

Estudiantes: Fecha: 06/29/2023 Grupo:7

Antonio Reyna 8-964-1747

Joy Nelaton: 8-902-1282

Salazar, Pedro 8-937-444

Kevin Coronado 8-952-2444

Jose Angel Lugo 20-70-4252

**Patrones de Arquitectura**

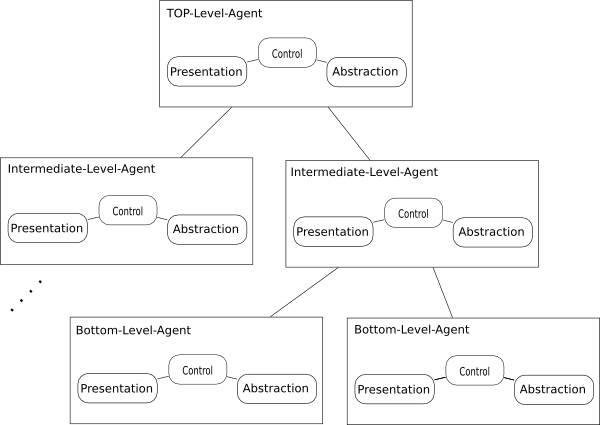
**Presentación-Abstracción-Control**

Definición: Es un patrón de arquitectura de software para sistemas interactivos, similar al Modelo Vista Controlador (MVC).

Se basa en una jerarquía de agentes cooperantes en la que, se divide un sistema interactivo en tres tipos de componentes o capas, destinadas a desarrollar funciones específicas dentro de la funcionalidad del sistema. Los componentes, que dan el nombre al patrón, son:

* Abstracción: Es el que recupera y procesa los datos y su funcionalidad. Sin embargo, solamente posee una parte de toda la estructura de datos.
* Presentación: Presenta la información y/o la lógica de operaciones en un formato previamente asignado, que corresponda con el tipo de información (audio, video...) y que sea adecuado para interactuar.
* Control: se encarga de asuntos como el flujo de control y de la comunicación entre los otros dos componentes. Éste es el que conecta la capa de presentación con los componentes de la capa de abstracción y provee una funcionalidad que permite a los agentes comunicarse con otros agentes PAC. Todos los agentes se comunican sólo a través de este tipo de componente.

Estructura: Tiene una estructura en la que los componentes de presentación, abstracción y control están conectados. La presentación se comunica con el control para enviar eventos y solicitudes. El control coordina la interacción entre la presentación y la abstracción, permitiendo la comunicación bidireccional entre estos componentes.

**Estructura de una aplicación con PAC**:  


Ejemplos de Aplicación:

* Smalltalk
* Adobe Photoshop
* Eclipse
* Ruby on Rails

**Reflexión**

Definición:

El patrón arquitectónico de Reflexión proporciona un mecanismo para cambiar dinámicamente la estructura y el comportamiento de un sistema de software. Permite la modificación de aspectos fundamentales, tales como los tipos de estructuras y mecanismos de llamada de funciones. En este patrón, una aplicación se divide en dos partes. Un nivel meta proporciona información sobre las propiedades del sistema y hace que el software este al tanto de sus propias características.

Estructura:

* Nivel Meta: Este nivel proporciona información sobre las propiedades del sistema y permite que el software sea autoconsciente. Este nivel es responsable de almacenar metadatos sobre la estructura y comportamiento del sistema, y ofrece la capacidad de modificar estas propiedades dinámicamente.
* Nivel Base: Este nivel consiste en la lógica de la aplicación, y su implementación depende del nivel meta. Los cambios en la información del nivel meta pueden afectar el comportamiento subsiguiente del nivel base. Esto significa que, si se realizan cambios en los metadatos de la estructura o comportamiento en el nivel meta, estos cambios pueden alterar dinámicamente la operación del nivel base.

Ejemplos de Aplicación:

* Inversión de control (IoC): Frameworks como Spring (Java) y ASP.NET Core (.NET) utilizan reflexión para instanciar y configurar objetos automáticamente.
* Frameworks de prueba unitaria: JUnit y TestNG en Java, NUnit en .NET y otras herramientas similares utilizan reflexión para descubrir e invocar métodos de prueba.
* Mapeo objeto-relacional (ORM): Herramientas como Hibernate (Java) y Entity Framework (.NET) usan reflexión para mapear entre objetos en código y registros en una base de datos.

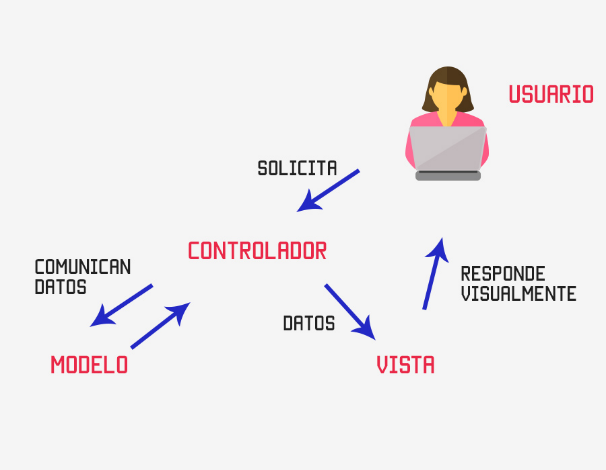
**Modelo-Vista-Controlador**

Definición:

El Modelo-Vista-Controlador (MVC) es un patrón de diseño de software ampliamente utilizado en el desarrollo de aplicaciones. Proporciona una forma estructurada de organizar el código y separar las responsabilidades en diferentes componentes. MVC se usa inicialmente en sistemas donde se requiere el uso de interfaces de usuario, aunque en la práctica el mismo patrón de arquitectura se puede utilizar para distintos tipos de aplicaciones. Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos.

Su fundamento es la separación del código en tres capas diferentes, acotadas por su responsabilidad, en lo que se llaman Modelos, Vistas y Controladores

Estructura:



* Modelos

Es la capa donde se trabaja con los datos, por tanto, contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos los tendremos habitualmente en una base de datos, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán los correspondientes selects, updates, inserts, etc.

* Vistas

Las vistas, como su nombre nos hace entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML. En las vistas nada más tenemos los códigos HTML y PHP que nos permite mostrar la salida.

En la vista generalmente trabajamos con los datos, sin embargo, no se realiza un acceso directo a éstos. Las vistas requerirán los datos a los modelos y ellas se generará la salida, tal como nuestra aplicación requiera.

* Controladores

Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc.

En realidad, es una capa que sirve de enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo.

Ejemplos de Aplicación:

* CakePHP: Es un framework de desarrollo rápido de aplicaciones open source muy flexible, basado en MVC e inspirado en Ruby On Rails. Sitios como MapMe, Copify, Piano Marvel y Flipcomp usan CakePHP
* Ruby On Rails: Se trata de una aplicación experimentada, la cual todas sus capas usan el patron MVC para organizar la programación de aplicaciones. Webs tan conocidas como Twitter, GitHub, Hulu o Slideshare usan Ruby

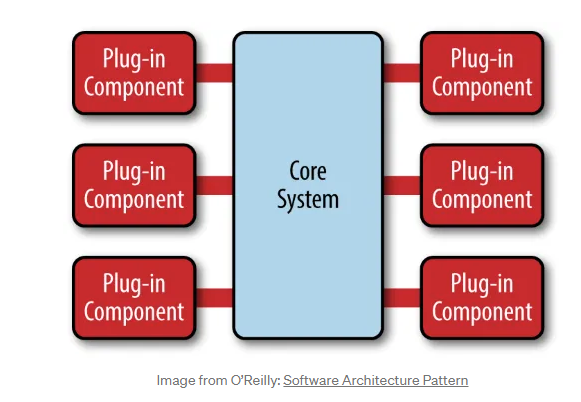
**Microkernel**

Definición:

Patrón arquitectónico usado en sistemas de software que deben adaptarse a los cambios de requisitos. Este supera el núcleo funcional, la funcionalidad extendida, y los aspectos relativos al cliente.

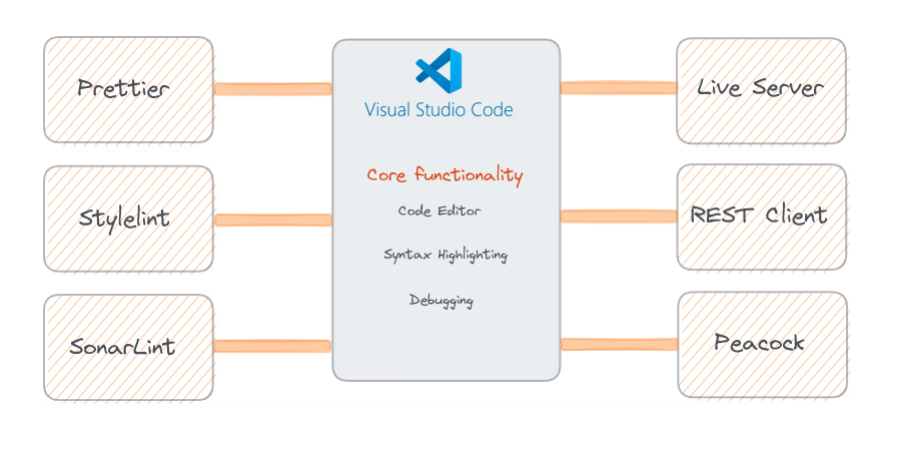
Estructura:

Está compuesto por el core (micro kernel principal) y los componentes o plugins (funcionalidades extras agregadas).



Ejemplos de aplicación: este patrón es ideal para aplicaciones basadas en productos en donde se pueden añadir más versiones y funciones a lo largo del proceso con pocos cambios.

El editor visual studio code constituye un ejemplo práctico. El core de visual studio code es la funcionalidad de edición y depuración y los plugins añaden más funcionalidad al programa sin alterarlo de forma drástica.



**Broker**

El Broker es un patrón de arquitectura que se utiliza para estructurar los sistemas de software distribuidos con componentes desacoplados que interactúan por invocaciones de servicios remotos. Esto quiere decir que, si un componente necesita un servicio de algún otro que se encuentra en otra ubicación que no conoce, el broker se encarga de proporcionar la conexión con el servidor apropiado, transmitir las peticiones al servidor y transmitir los resultados y excepciones de regreso al cliente.

**Estructura**

Servidor:

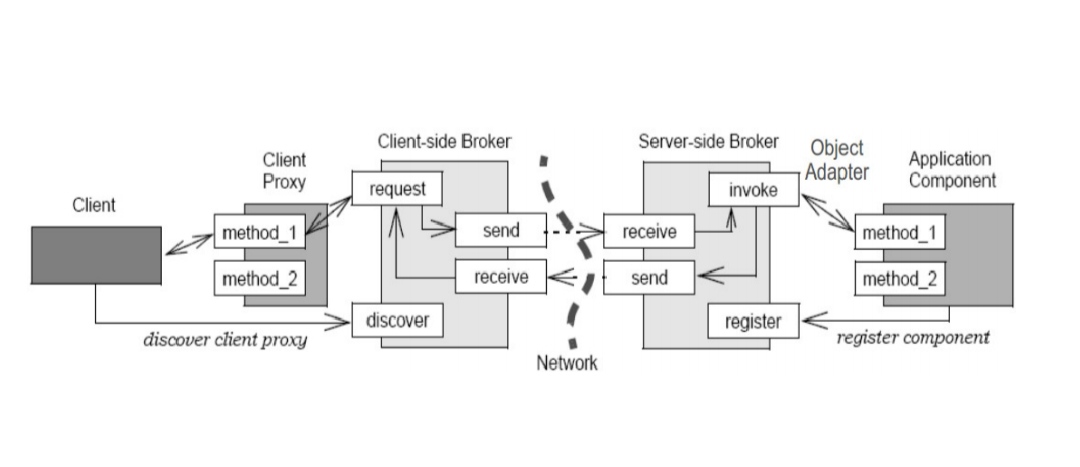
Implementan objetos que exponen su funcionalidad a través de interfaces que consisten en operaciones y atributos. Dichas interfaces están disponibles a través de un lenguaje de definición de interfaz (IDL) o un estándar binario.

Se pueden clasificar en dos tipos:

* Servidores que ofrecen servicios comunes a muchos dominios de aplicación.
* Servidores específicos para dominios específicos.

Cliente:

Los clientes son aplicaciones que acceden a los servicios de, al menos, un servidor. La interacción que sucede entre clientes y servidores se basa en un modelo dinámico, lo cual significa que los servidores también pueden actuar como clientes. Los clientes no necesitan conocer la ubicación de los servidores a los cuales acceden; esto es importante porque permite la inclusión de nuevos servicios, y el movimiento de los servicios existentes a otras ubicaciones, aún mientras el sistema este ejecutándose.

Broker:

El Broker es el responsable de la comunicación entre los clientes y servidores. Cuando un cliente necesita un servicio de un servidor, envía una petición al broker, que a su vez se la envía al servidor. El servidor realiza la funcionalidad correspondiente y envía la respuesta (o excepción) de vuelta al broker, que se la devuelve al cliente. También ofrece API’s a clientes y servidores que incluyen operaciones para el registro de servidores, y la invocación de métodos de servidores.

De otro modo si el servidor especificado es mantenido por otro broker, el broker local encuentra una ruta al broker remoto y le envía la solicitud.

Proxies:

Podemos distinguirlos por los dos tipos de proxies existentes los cuales son: los proxies del lado del cliente y los proxies del lado del servidor:

* Proxies del cliente: Se ubican entre el cliente y el broker para esconder detalles de implementación como el marshalling (empaquetado de parámetros), la creación y eliminación de bloques de memoria y el mecanismo de comunicación es utilizado para enviar el mensaje.
* Proxies del servidor: Son análogos a los proxies del cliente, con el añadido de que reciben las solicitudes, las desempaquetan, llaman al servicio apropiado y hacen el marshalling de la respuesta o excepción.

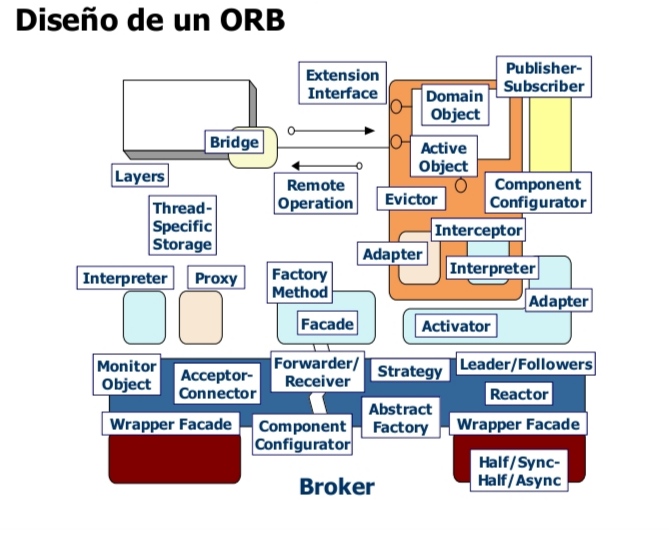
Puentes:

Son un componente opcional utilizado para esconder detalles de implementación cuando dos brokers se encuentran inter-operando entre si.

**Usos conocidos**

www:

EL World Wide Web es el sistema Broker disponible más grande del mundo. Los browsers de hipertexto, tales como Hot Java, Mosaic y Netscape actúan como brokers, y los servidores WWW hacen el papel de servidores de servicio.

CORBA:

Tecnología orientada a objetos distribuidos en sistemas heterogéneos. Un lenguaje de definición de interface esta disponible para soportar la interoperabilidad de los objetos cliente y servidor. Muchas implementaciones de CORBA reconocen la variante del sistema Broker de comunicación directa.

OLE 2.x de Microsoft:

La tecnología OLE 2.x de Microsoft provee otro ejemplo del uso de patrón de arquitectura Broker. Mientras que CORBA garantiza interoperabilidad utilizando un lenguaje de definición de interfaz, OLE 2.x define un estándar binario para mostrar y acceder a la interfaz de los servidores.

**Referencias**

<https://es.wikipedia.org/wiki/Presentaci%C3%B3n%E2%80%93abstracci%C3%B3n%E2%80%93control>

<http://software-pattern.org/Reflection>

<https://desarrolloweb.com/articulos/que-es-mvc.html>

<https://www.bbvaapimarket.com/es/mundo-api/mvc-desarrollo-fiable-de-aplicaciones-web/#:~:text=Webs%20tan%20conocidas%20como%20Twitter,puedes%20consultar%20sus%20principales%20características>.

Tavargere, Z. (2023, February 27). Microkernel Architecture: How It Works and What It Offers. *Zahiruddin Tavargere (zahere.com)*. <https://zahere.com/microkernel-architecture-how-it-works-and-what-it-offers>