Ingeniería en Desarrollo de Software

4º semestre

Programa de la unidad didáctica:

Diseño y arquitectura de software

Unidad 3. Arquitectura de sistemas informáticos

Clave:

Ingeniería:	TSU:
15142424	**/16142525**

Universidad Abierta y a Distancia de México

México, Ciudad de México, marzo del 2023

DCEIT

Índice

3. Arquitectura de sistemas informáticos

Presentación

En la presente unidad revisarás y aplicarás nuevos patrones arquitectónicos en los ambientes distribuidos, interactivos y adaptables como base para la elaboración de un diseño de arquitectura de software.

Conocerás nuevos estilos arquitectónicos tales como proxy, MVC, orientado a servicios, entre otros, los cuales forman parte de la clasificación de los ambientes antes mencionados. Aprenderás a decidir y proponer soluciones que cumplan con los requerimientos funcionales y no funcionales de un problema específico de desarrollo de software para determinar qué patrón arquitectónico de los analizados se debe aplicar para el diseño de una arquitectura de software según sus características y conveniencia para el tipo de problema que se presente.

Logros:

Al término de esta unidad lograrás:

- Generar una arquitectura “tipo”, que aplique los patrones arquitectónicos Proxy, Orientado a servicios y MVC analizados en la presente unidad.
- Aplicar la arquitectura en ambientes distribuidos, ambientes interactivos y ambientes adaptables.
- Tomar decisiones sobre la aplicación de arquitectura a utilizar.

Competencia específica

- Diseña un modelo arquitectónico que aplique e identifique los componentes de los sistemas distribuidos, interactivos y adaptables para el desarrollo de una aplicación informática dirigida a un objetivo específico mediante un patrón de arquitectura de software.

3.1. Patrones arquitectónicos en sistemas distribuidos

Distribuir el trabajo siempre se ha considerado una buena solución para aminorar la carga entre los participantes y, al mismo tiempo, que todos tengan una participación de la misma magnitud en la solución de un problema. Cuando se debe realizar alguna actividad, independientemente de su origen, puede ser dividida en partes. Cada una de éstas puede verse como un problema aislado y se puede solventar como tal, la unión de las soluciones será, por tanto, la solución al problema global.

Figura 1. Sistemas distribuidos

Los sistemas computacionales no quedan exentos de aplicar la división del trabajo entre las partes que lo conforman. Un sistema computacional distribuido es la unión de varias computadoras en un mismo espacio físico, que tienen como objetivo compartir sus recursos de hardware y software para lograr un objetivo común. Se busca tener este tipo de sistemas computacionales distribuidos por varias razones: la constante evolución de las redes, la importancia creciente que están tomando los sistemas multimedia, la tecnología móvil y sobretodo la visión

de los sistemas distribuidos como una modalidad eficiente que permite explotar y distribuir la carga de trabajo entre los recursos disponibles.

Para el diseño de un sistema computacional distribuido, el arquitecto dispone de varios patrones aplicables a éste tipo de sistemas que le permitirán contextualizar la arquitectura del software de tal manera que posibilite la división del trabajo entre los recursos de software y hardware.

3.1.1. Ámbito de aplicación

En la actualidad, las computadoras personales (no especializadas) tienen una capacidad de procesamiento de alto rendimiento comparado con las computadoras de hace 5 años; pero esta capacidad de procesamiento y además de almacenamiento no es aprovechada en su totalidad por un usuario normal, porque el uso que le da a la computadora se centra en meras actividades que demandan sólo entre el 10% y el 40% de la capacidad total que proporciona el hardware y/o software. No obstante, hay problemas del ámbito científico y/o tecnológico que sobrepasan a las capacidades mencionadas de las computadoras no especializadas que se poseen actualmente. Para resolver este tipo de problemas se crean arquitecturas distribuidas, en donde se conectan en una misma red una serie de computadoras (desde dos en adelante) que cooperan con su capacidad de procesamiento para la resolución del problema, solventando, cada una, una pequeña parte de él. Como se dijo anteriormente, la suma de estos pequeños resultados dará la solución total al problema. Así, no dependerá de una sola computadora para resolver el problema, ni del tiempo que ello implica.

La solución de problemas complejos requiere, por necesidad, la aplicación de una arquitectura distribuida. La arquitectura del software que genere el arquitecto dictará la forma más simple, pero efectiva, de resolver el problema.

Para tener más claro el concepto de un sistema distribuido, se verá de la siguiente manera: Una empresa tiene la necesidad de lanzar un nuevo servicio a través de internet a sus clientes, en donde éstos vayan haciendo un historial de lo que pasa en sus vidas. La empresa quiere hacer esta recolección de datos sin que el cliente tenga que hacer algo diferente a lo habitual.

Tiene proyecciones que indican que el crecimiento de los datos puede llegar a ser exponencial debido a la cantidad de usuarios que tiene registrados como “activos” en su base de datos. Los datos que se guardarán en su base de datos serán:

1. Información estructurada sobre los usuarios, por ejemplo, sus datos personales.
2. Información no estructurada como correos electrónicos y su contenido.
3. Información sobre las fotografías donde el cliente aparece.
4. Información sobre los lugares que visita el cliente.
5. Búsquedas y sus resultados.
6. Llamadas telefónicas y su duración, así como el destinatario.
7. Compras en línea.
8. Lugar de trabajo y estatus socioeconómico.
9. Entre otros.

.

La cantidad de información que se genera en un lapso de tiempo muy corto (horas incluso) llega a ser inmenso, pero la empresa no quiere perder ningún detalle de cada ocurrencia que tiene cada usuario. Al final de un periodo de tiempo establecido se tiene una cantidad de información bastante considerable y se necesita obtener conocimiento de esa información, en este punto es donde conviene introducir un sistema distribuido, porque para poder procesar tal cantidad de información con todas las variables que involucra sería una tarea poco práctica para una computadora única. La cooperación entre distintos nodos de un sistema distribuido será esencial, por ejemplo, para predecir el comportamiento de

consumo de cada usuario con base en toda la información histórica que ha generado.

Una implementación sugerida es que cada nodo procese cada uno de los 8 puntos de la lista anterior y obtenga un resultado, luego envíe esta información al nodo central y éste se encargue de compilarlos para generar el conocimiento que se busca obtener sobre cada usuario a partir de sus datos.

Un sistema computacional distribuido da al arquitecto de software una visión completamente distinta sobre la posible solución que pueda proponer, pero lanzar una solución de este tipo sin conocer los puntos más importantes que lo distinguen podría resultar perjudicial para la solución, por tanto, el arquitecto debe conocer las características de un sistema distribuido.

Características de sistemas distribuidos

Un sistema distribuido es “un conjunto de procesadores de información independientes conectados entre sí, que dan la sensación de ser una sola unidad” (Tanenbaum, 2006) y debe cumplir con ciertas características para poder considerarse como tal.

Una característica es una cualidad propia de una cosa o persona que lo distingue inequívocamente de los demás, se listarán estas cualidades inherentes deseables para cualquier sistema distribuido (Tanenbaum, 2006):

- **Cooperación.** Para poder considerarse un sistema computacional como un sistema distribuido debe tener por lo menos dos procesadores de información actuando al mismo momento para generar una solución. En la actualidad y con el avance de la tecnología, una computadora puede tener

más de un núcleo en su procesador y, por tanto, puede considerarse como procesamiento distribuido y paralelo, al mismo tiempo.

- **Comunicación entre los nodos o "**punto, momento o espacio en donde todos los elementos de una red que comparten las mismas características se vinculan e interactúan. Estos elementos son a su vez nodos y pueden relacionarse en una red horizontal o de otro tipo donde cada ordenador y servidor constituye un nodo" (Fuente: http://www.definicionabc.com/tecnologia/nodo.php): sin ella no tendría sentido dividir el problema, porque no se podrían "distribuir" las partes en que se dividió el problema.
- **Un procesador central.** Administra la carga, entrega y recepción de información. Su trabajo no es en sí el procesamiento, sino la administración del trabajo de los demás agentes (nodos) que componen la red de procesamiento.
- **Procesadores dispuestos físicamente separados**, debido a que por la misma definición de distribución del trabajo, los procesadores de información deben estar separados, aunque estén dentro del mismo núcleo de la unidad central de procesamiento.
- **Tiempo de respuesta reducido.** El tiempo de respuesta es menor en comparación con el procesamiento que hace una sola computadora.
- **Evita el procesamiento central** porque va en contra de la misma definición de un sistema distribuido.

Con estas características, el arquitecto de software debe hacer una diferencia básica en el ámbito donde él trata de aplicar soluciones a través de AS. Es distinto hablar de la teoría de la computación distribuida y la aplicación de una arquitectura distribuida, porque en la teoría de la computación distribuida se analiza y aplica el uso del hardware y software, en la aplicación de una arquitectura distribuida se diseña cómo será el uso de este hardware y software para solventar un

determinado problema (Barnaby, 2002). Será diferente la concepción de una solución en donde se involucre la división del trabajo a la implementación de los detalles técnicos de computación distribuida.

No es forzoso el cumplimiento al pie de la letra de todas las características que se mencionan en la lista de características de sistemas distribuidos, sólo son deseables. Idealmente debería cumplir con todas las características, sin embargo, sólo está exento el cumplimiento de la disposición física en lugares distintos, las demás características no deben faltar.

La aplicación de una arquitectura distribuida no se debe considerar siempre como la mejor para cualquier tipo de problema, ya que podría ocasionar ser más cara la solución que el problema en sí, conocer sus ventajas y desventajas podrá guiar para tener un punto más de referencia respecto a su elección a la hora de solventar un problema.

Ventajas y desventajas de los sistemas distribuidos

La factibilidad de implementar o no una arquitectura distribuida radica en función de qué tanta ventaja competitiva dará a la solución propuesta, y de qué valor agregado tendrá la propuesta de una arquitectura de tipo distribuida sobre una arquitectura no distribuida tradicional.

Ventajas y desventajas. Tomada de http://www.opcionesbinariaspro.com/sites/default/files/styles/large/public/field/image/ventajas-desventajas-binarias.jpg

Cuando el arquitecto de software ha decidido qué arquitectura de software resolverá de manera adecuada (tal vez no la mejor) el problema enfrentado, se deben considerar estas ventajas y quizá, sin perder el enfoque central, las desventajas que ofrece una arquitectura u otra. A continuación se listan las principales ventajas de la arquitectura distribuida,

desde el punto de referencia de la teoría de sistemas distribuidos (Tanenbaum, 2006):

Ventajas

- **Los datos son comunes a todos los nodos:** varios procesadores de información pueden acceder a la misma información que existe en una base de datos común.
- **Los dispositivos son compartidos para algunos nodos:** como cada nodo es sólo una unidad central de procesamiento que puede compartir, por ejemplo, una unidad de almacenamiento común a todos los nodos.
- **La comunicación es directa en ambos sentidos:** contrastado con un sistema tradicional no distribuido que depende de terceras partes (servidores de internet, servidores web, servidores de bases de datos) para poder tener una comunicación efectiva; la comunicación es más rápida, siguiendo con el contraste citado al inicio del párrafo.
- **La carga del trabajo es flexible:** las tareas se distribuyen de manera equitativa, dependiendo de la disponibilidad de cada nodo.

La visión contraria, las desventajas, también debe ser tomada en cuenta al momento de decidir:

- **Diseño lógico:** es poca la experiencia diseñando e implementando arquitecturas de software que sean capaces de aprovechar todo el potencial de la computación distribuida.
- **Sistemas de comunicación:** si la red de transporte de datos no es lo suficientemente robusta para soportar la carga de trabajo que genera en sistema distribuido, se convertirá en “un cuello de botella” y todas las bondades antes citadas de los sistemas distribuidos se perderán.

- **Aseguramiento de los datos:** los datos estarán viajando y procesándose en distintos sistemas no centralizados, la importancia que se dé a su seguridad dependerá de la delicadeza de su origen.

Estas consideraciones debe tenerlas en cuenta el arquitecto de software (Tanenbaum, 2006; Barbaby, 2002; De la Torre et ál., 2010; Farley, 1998 y Couluris et ál., 2011) al proponer la solución que está basada en una arquitectura distribuida, porque el transporte de la teoría a la realidad puede ser compleja.

Como recomendación se sugiere, en la literatura especializada en arquitectura de software, que se trate de ver la solución distribuida como una cooperación de agentes que son parte de un suprasistema y no caer en los detalles finos de la implementación, eso no es trabajo del arquitecto de software, deberá ser responsabilidad de quien programe e implemente el software que refleja y soporta la arquitectura (Tanenbaum, 2006).

Para que quede de manera clara y no se tome como una mera explicación de cosas preconcebidas, las ventajas y desventajas de una arquitectura distribuida se hacen contra la comparación de una arquitectura tradicional.

Dentro de los modelos de arquitectura de software analizados en la unidad anterior pueden ser aplicables a un sistema distribuido los modelos en capas y cliente servidor.
Para obtener mayor información sobre los sistemas distribuidos y analices de qué forma se relacionan los modelos antes mencionados, te invitamos a que revises el documento *U3. Sistemas distribuidos.*

3.1.2 Arquitectura orientada a servicios

Con la evolución de internet se vuelve necesaria la integración de sistemas heterogéneos. Muchas organizaciones, no importa el giro, proveen servicios a sus clientes, empleados, asociados, etc. Un ejemplo de lo anterior es el siguiente: en un banco los ejecutivos proveen servicios a los clientes, dentro de los cuales podemos encontrar como típicos los siguientes (Newcomer E., Lomow G. (2005). *Understanding SOA with Web Services*. 1ra edición. Independent Technology Guides series. Upper Saddle River, NJ : Addison-Wesley, Chapter 1: Introduction to SOA with Web Services):

- Manejo de cuentas (aperturas, cierres, etc.)
- Préstamos y créditos (aplicaciones, términos y condiciones, etc.)
- Operaciones como depósitos, transferencias, retiros.
- Intercambio de divisas, etc.

Algunos ejecutivos del banco están capacitados para ofrecer el mismo conjunto de servicios que permita un balance en la carga de trabajo y una mayor disponibilidad para los clientes.

Lo que sucede o que se encuentre detrás de cada proceso (programas utilizados, reglas de negocio, tecnología, etc.) no es importante para el cliente, mientras su trámite sea atendido. En algunos casos el proceso es una compleja transacción que requiere que el cliente acuda con diversos ejecutivos por lo que se establece un flujo de proceso de negocio.

Detrás de todo este esquema se encuentran los sistemas informáticos que automatizan los procesos del banco y apoyan los objetivos del negocio brindando los servicios mediante los ejecutivos, cajeros automáticos o la web. Un ejemplo de lo descrito anteriormente lo puedes observar en la siguiente imagen:

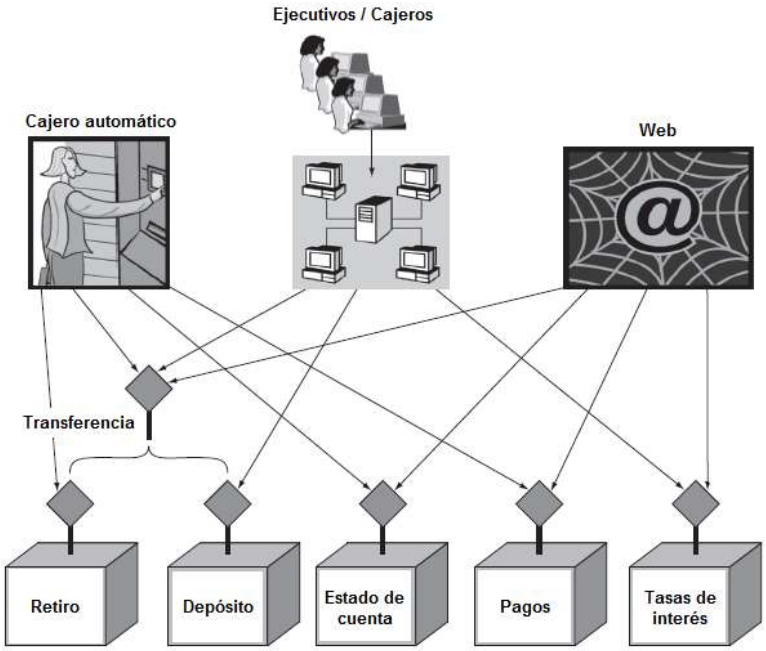


Figura: Ejemplo de los servicios brindados por un banco

Recuperada de: Newcomer E., Lomow G. (2005). *Understanding SOA with Web Services*. 1ra edición. Independent Technology Guides series. Upper Saddle River, NJ : Addison-Wesley, Chapter 1: Introduction to SOA with Web Services, Page 12

Implementar estos servicios (e incluso operaciones más complejas que las descritas en el ejemplo) en el contexto de una arquitectura orientada a servicios (SOA) hace esta tarea más sencilla y permite presentar la funcionalidad como un servicio accesible a todos los involucrados en el sistema.

SOA es una arquitectura de alto nivel que define la forma en que se comunicarán las aplicaciones, denominadas en este caso servicios, con otras aplicaciones. Se establece una relación con la arquitectura en capas ya que cada aplicación puede estar diseñada con base en la misma. SOA es a la vez un modelo de programación y de arquitectura a la vez que permite el diseño de sistemas de software que proveen servicios a usuarios y a otras aplicaciones mediante interfaces.

Dentro de este contexto es importante definir qué es un servicio, término que ha estado presente en la computación desde hace mucho tiempo y que se ha utilizado de diversas formas:

> "Un servicio es un componente reusable que representa funcionales operacionales y de negocio. Se aplica el término reusable para indicar que es ésta característica la que permite la creación de nuevos procesos basados en los servicios que se proporcionan. Cada servicio expone sus prestaciones mediante una bien definida interface." *(Lewis G., September 2010, White Paper, Software Engineering Institute, Carnegie Mellon)*

La arquitectura basada en servicios representa un enfoque para la construcción de sistemas distribuidos. La estructura de una arquitectura de este tipo puede dividirse en dos partes: en una se encuentran los aspectos funcionales y en la otra los aspectos de calidad de los servicios (M. Endrei, J. Ang, A. Arsanjani. S. Chua, P. Comte. P. Korgdahl, M. Lou, T. Newling (2004). *Patterns: Service Oriented Architecture and Web Services*. 1ra edición. IBM Reedbooks. IBM's International Technical Support Organization (ITSO), Research Triangle Park, NC):
Dentro de los aspectos funcionales podemos localizar:

- Transportación: mecanismo utilizado para mover las requisiciones de servicios desde el consumidor hacia al proveedor y la respuesta al mismo.
- Protocolo de comunicación de servicios: mecanismo mediante el cual se establece la comunicación entre el proveedor y el consumidor del servicio.
- Descripción del servicio: esquema que se utiliza para indicar lo que el servicio proporciona, cómo se invoca y la información requerida.
- Proceso de negocio: colección de servicios que son invocados en una secuencia en particular mediante un conjunto de reglas para cubrir un requerimiento específico.

- Descripción del servicio: indica la manera en que el servicio interactuará, así como el formato de la respuesta y solicitud del mismo.

Dentro de los aspectos de calidad de servicio podemos localizar:

- Políticas: condiciones o reglas sobre las cuales los servicios se hacen disponibles a los usuarios.
- Seguridad: reglas que aplican para la identificación, autorización y acceso a los servicios.
- Transacción: conjunto de atributos que deben ser aplicados a un grupo de servicios para proveer un resultado consistente.
- Administración: conjunto de atributos aplicados para el manejo y administración de los servicios, tanto proveídos como consumidos.

Los servicios son autónomos, ya que cada uno se diseña, desarrolla, implementa y mantiene de forma independiente y pueden estar situados en cualquier nodo de una red local o remota, mientras exista el soporte necesario para los protocolos de comunicación.

Mediante la implementación de una arquitectura SOA es posible modularizar los sistemas y aplicaciones en componentes de negocio que pueden combinarse mediante interfaces bien definidas. Derivado de lo anterior se pueden entonces mencionar los siguientes beneficios de la implementación de una arquitectura de este tipo (De la Torre, C., Zorrilla, U., Calvarro, J. y Ramos, M. A. (2010). *Guía de arquitectura N-Capas orientada al dominio con .NET 4.0*. España: Krasis Press):

- Aprovechamiento de la tecnología existente y de procesos de negocios que conllevan a la reducción de costos mediante la reutilización de servicios comunes con interfaces estándar.
- Alto nivel de abstracción, ya que cada servicio es autónomo y se accede a los mismos mediante un protocolo establecido.

- Rápida adaptación y despliegue de servicios, clave para brindar capacidad de respuesta a clientes, socios y empleados.
- Es adaptable al cambio, lo cual añade flexibilidad y facilita cualquier modificación necesaria en el proceso de negocio.

SOA es un patrón de arquitectura a la vez simple y poderoso. Un hecho en relación al mismo es que permite la migración de arquitecturas TI tradicionales permitiendo la integración de diversas tecnologías donde la lógica de negocio es descompuesta en servicios bien definidos y reutilizables, disponibles a los usuarios del sistema. SOA expone las funcionalidades del sistema como servicios para ser consumido por las aplicaciones. Una actual reorganización de SOA la constituyen los servicios web que se ha convertido en una de las más populares implementaciones de este patrón arquitectónico.

3.1.3 Patrón de arquitectura de nube

La arquitectura de nube se refiere a cómo varios componentes de la tecnología de nube, como el hardware, los recursos virtuales, las capacidades del software y los sistemas de red virtual, interactúan y se conectan para crear entornos de computación en la nube. Actúa como un plano que define la mejor manera de combinar de manera estratégica los recursos a fin de crear un entorno de nube para una necesidad empresarial específica. (Google (2024). What is cloud architecture? Benefits & Components Recuperado de: https://cloud.google.com/learn/what-is-cloud-architecture).
Los componentes de este patrón incluyen:

- Frontend
- Backend
- Modelo de entrega basado en la nube
- Red (internet, intranet, otras)

Funcionamiento de la Arquitectura de Nube

El backend alberga todos los recursos de computación en la nube, servicios, almacenamiento de datos y aplicaciones ofrecidos por el proveedor. La conexión entre los componentes de frontend y backend se establece a través de una red que facilita la transmisión bidireccional de datos. Cuando los usuarios interactúan con el frontend, envían consultas al backend mediante middleware, donde el modelo de servicio ejecuta tareas específicas.

Modelos de Servicio en la Nube

Los servicios disponibles para usar dentro de la arquitectura dependen del modelo de entrega basado en la nube o del modelo de servicio. Existen tres modelos principales de servicios de computación en la nube:

Infraestructura como Servicio (IaaS): Proporciona acceso bajo demanda a la infraestructura de nube, incluyendo servidores, almacenamiento y herramientas de redes. Este modelo elimina la necesidad de adquirir, administrar y mantener la infraestructura local.

Plataforma como Servicio (PaaS): Ofrece una plataforma de procesamiento completa con todas las herramientas de infraestructura y software subyacentes necesarias para desarrollar, ejecutar y administrar aplicaciones.

Software como Servicio (SaaS): Suministra aplicaciones basadas en la nube proporcionadas y mantenidas por el proveedor de servicios, eliminando la necesidad de que los usuarios implementen software localmente.

Capas de arquitectura de nube

Una forma más simple de comprender cómo funciona la arquitectura de la nube es pensar en todos estos componentes como varias capas ubicadas una encima de

la otra para crear una plataforma en la nube. A continuación se describen las capas básicas de la arquitectura de nube:

Hardware: servidores, almacenamiento, dispositivos de red y otros elementos de hardware.

Virtualización: capa de abstracción que crea una representación virtual de los recursos físicos de almacenamiento y procesamiento. Esto permite que varias aplicaciones usen los mismos recursos.

Aplicación y servicio: esta capa coordina y admite solicitudes de la interfaz de usuario de frontend, y ofrece diferentes servicios según el modelo de servicio en la nube, desde la asignación de recursos hasta las herramientas de desarrollo de aplicaciones y las aplicaciones basadas en la Web.

Tipos de arquitecturas

La elección del tipo de arquitectura de nube es crucial y debe basarse en consideraciones específicas de inversión en tecnología, requisitos comerciales y objetivos organizacionales. Existen tres tipos principales de arquitecturas de nube:

Nube Pública: Utiliza recursos de computación en la nube y una infraestructura física que es propiedad y es gestionada por un proveedor externo de servicios de nube. Permite escalar recursos fácilmente sin invertir en hardware propio, pero implica el uso compartido de arquitecturas entre múltiples usuarios.

Nube Privada: Corresponde a una nube dedicada, con propiedad y gestión interna por parte de la organización. Se aloja de forma privada en las instalaciones locales del centro de datos de la empresa, brindando mayor control y seguridad sobre la infraestructura y los datos. Aunque más costosa, ofrece un nivel de personalización y seguridad que puede ser esencial, requiriendo mayor experiencia en TI para su mantenimiento.

Nube Híbrida: Combina arquitecturas de nube pública y privada. Permite la migración de cargas de trabajo, posibilitando el uso de servicios que mejor se adapten a las necesidades y cargas de trabajo específicas de la empresa. La arquitectura híbrida se destaca como la opción preferida para empresas que buscan controlar sus datos mientras aprovechan las ofertas de nube pública.

3.2. Patrones arquitectónicos en sistemas interactivos

La interacción es la relación de causa-efecto entre dos o más involucrados. Los sistemas interactivos nacieron por la necesidad de reaccionar en función de una respuesta dada por otro involucrado, que puede ser un humano u otro sistema de software. Cambiar la salida esperada del sistema con base en las entradas proporcionadas hacia el sistema. Cuando no existían este tipo de sistemas las entradas eran conocidas, el proceso era conocido y las salidas esperadas, del tipo del cálculo de una ecuación irresoluble por un humano, no por su complejidad, sino por el tiempo de operación que implica el resolverla; eran sistemas muy medidos y muy cuadrados en su ámbito, no permitían involucramiento externo alguno, sólo en las entradas proporcionadas inicialmente.

La riqueza y variación que se dé en tipos de respuesta que pueden proporcionar los sistemas interactivos dependerá del tipo de entradas que el usuario proporcione durante la ejecución del proceso que debe realizar el software.

Una vez que el arquitecto de software determina que en base a los requerimientos del problema a resolver es factible aplicar un diseño de sistema interactivo dispone de varios patrones arquitectónicos para su elaboración, como son: MVC, PAC. En las páginas siguientes analizaras las características principales de cada uno de ellos.

3.2.1. Ámbito de aplicación

Un sistema interactivo describe una interface de usuario que se ajusta con base en la interacción usuario-sistema para reflejar los requerimientos particulares. En tanto que las computadoras intentan servir más que nunca en la historia de las tecnologías, a más usuarios de diversas maneras la interface usuario-computadora toma una importancia crítica. Procesadores de texto, navegadores, hojas de cálculo son todas aplicaciones interactivas que permiten simplificar procesos o tareas al usuario. Dispositivos como teléfonos inteligentes, tabletas, reproductores de música son parte de nuestra vida diaria y requieren para su operación la implementación de sistemas interactivos.

El diseño de sistemas interactivos constituye un reto y a la vez una disciplina fascinante ya que afecta en distintas áreas de la vida cotidiana la forma en que se desarrollan las actividades actualmente.

Hay una gran variedad de sistemas interactivos, los cuales deben diseñarse con la premisa de estar “centrado en el usuario”, sus actividades y las tecnologías utilizadas. Se debe asegurar que los sistemas sean accesibles y que los diseños sean útiles y aceptados por el usuario.

La variedad de sistemas interactivos es amplia, a continuación se listarán y explicarán los más importantes.

3.2.2. Modelo-vista-controlador

A lo largo del desarrollo de la presente unidad se ha hecho énfasis en la importancia de la separación de las partes lógicas y físicas que conforman la solución de software. Una manera correcta de lograr esta división es la aplicación del patrón arquitectónico modelo-vista-controlador.

El patrón de arquitectura de software modelo-vista-controlador (MVC) se centra únicamente en la separación de las tareas de un sistema de software.

Un sistema de software se divide en tres partes:

- Lo que el usuario ve (pantallas), que es la parte específica que representa la capa de la **vista**.
- La aplicación de las reglas de negocio propias del contexto, que es la parte específica que representa la capa del **controlador.**
- En dónde se almacenan los datos, que es la parte específica que representa la capa del **modelo**.

El patrón de arquitectura MVC hace la separación lógica y física sobre la base a estas tres capas:

1. La interfaz de usuario.
2. Lógica del negocio.
3. Los datos de la aplicación.

La separación de las partes física y lógica de un software, en principio, puede parecer difícil y un proceso complejo, pero a la larga trae beneficios para muchas etapas que involucran el desarrollo del software.
A continuación se enlistan algunos beneficios de la separación de las partes de un software:

- La separación ayuda a resolver el problema independientemente, debido a que cada una de las partes trabaja aparte de las otras dos, se cumple con

el principio básico de la separación modular, “baja dependencia, alta cohesión”.

- La independencia modular que oferta el patrón, hace posible la reutilización de cualquiera de las tres partes; por ejemplo, la funcionalidad que proporciona la lógica del negocio puede ser llamada desde una computadora tradicional (con su interfaz tradicional) o desde un dispositivo móvil (con deferente interfaz) y los resultados serán los mismos para ambas plataformas, sólo cambiará la forma en cómo se presentan los datos procesados. A la capa de modelo y del controlador se les puede aplicar el mismo ejemplo descrito.
- El mantenimiento del sistema será más fácil, porque ante un posible fallo, es rápido identificar en qué capa lógica y/o física del patrón MVC se genera dicha falla, sin afectar a las otras dos.
- La separación física (por niveles) será de manera transparente para la aplicación, porque no tienen que “vivir” en el mismo lugar físico las capas del modelo y así dedicar más recursos de procesamiento computacional, almacenamiento o presentación, según sea el caso.

La posición que ocupará dentro de la arquitectura del software cada una de las partes del modelo arquitectónico MVC debe quedar perfectamente clara para el arquitecto. A continuación se explica cada una de las partes de un modelo arquitectónico MVC:

1. **Modelo:** es la representación de los datos de la aplicación, generados y almacenados dentro del ámbito de su competencia. Un sistema puede tener muchas fuentes de datos (manejadores de bases de datos, hojas de cálculo, archivos de texto plano, sistemas de información, entre otros) de las cuales toma información. La capa del **modelo** debe ser capaz de

recuperarlos y mostrarlos a las demás capas sin que “se enteren” del trabajo que tuvo que realizar para lograrlo.

2. **Vista:** es la representación del modelo en un formato amigable al usuario y permite su interacción. Está representada por la interfaz gráfica de usuario (GUI, por sus siglas en inglés), que es el conjunto de ventanas donde el usuario interactúa con la aplicación. En esta capa del software recibe información procesada y representada de manera clara y fácil de interpretar, ingresa datos si es que el uso así lo amerita.
3. **Controlador:** aplicación del funcionamiento propio del contexto, responde a peticiones del usuario hechas desde la vista y a su vez, hace peticiones al modelo para tomarlo como entrada para su proceso. Puede tomarse como la parte de comunicación entre el modelo y la vista, aplicando reglas de existencia entre ellos (De la Torre, et al., 2010).

La aplicación de este patrón arquitectónico es identificable en algunas de las plataformas más populares de internet. Para clarificar su uso se explicará un breve ejemplo de ello.

Una gran proporción de personas conocen el concepto de una red social y hasta son usuarios asiduos de ellas. Estás plataformas son una clara aplicación del patrón arquitectónico MVC. Algunas de estas redes sociales tienen millones de usuarios y la información que se genera cada día (comentarios, fotografías, redes de participación) es inconmensurable y aquí entra en función la capa del **modelo**; cuando una persona hace una publicación de cualquier índole puede elegir quién puede verla y quién puede participar haciendo comentarios sobre ella, hay reglas sobre qué tipo de palabras se pueden publicar y cuáles no, se puede configurar las relaciones que se tiene con las personas que pertenecen a nuestro círculo y aquí entra en función la capa del **controlador**.

La manera como se despliegue esta información, si está ordenada por fecha, por la persona que hace el comentario, la manera en que se pueden ver los álbumes de fotografías, la forma en que presenta las conversaciones que se tiene con las personas que pertenecen a un círculo social, dónde está situado el menú, dónde aparece la publicidad, es aquí que entra en función la capa de la **vista**.

Con frecuencia la configuración gráfica de la página (el acomodo de sus elementos) ha cambiado, tal vez el menú pasó de estar a la derecha a la izquierda o los colores son diferentes; sin embargo, nuestros datos siguen intactos, aun cuando estos cambios aparezcan de un día al otro, porque se cambió la capa de la vista, pero la capa del modelo sigue sin modificaciones. Aunque la información que se genera a diario es mucha en cantidad, debe pasar por todos estos filtros de seguridad y configuración antes de presentarse al usuario.

Para profundizar en el estudio de estos temas, puedes revisar el documento *U3. Arquitectura MVC* en los *Materiales de desarrollo de la unidad.*

3.2.3. Presentación-abstracción-control

Presentación-abstracción-control (PAC) es un patrón de arquitectura de software que pertenece a la categoría de sistemas interactivos.

La idea principal sobre la cual gira el PAC es la de agentes cooperativos que hacen una función bastante similar a una capa en el MVC. Los agentes tienen una organización jerárquica que define el arquitecto de software y con una función específica dentro del sistema en general; la jerarquía está compuesta por tres capas, de ahí viene el nombre del patrón: presentación, abstracción, control.

En gran medida es exactamente igual al patrón MVC, su diferencia está en la aplicación que se le da a cada patrón. Para conocer su correcta aplicación al

proponer la arquitectura solución que soportará al software, el arquitecto debe tener bien definido qué significa cada una de las partes mencionadas:

- **La presentación** tiene el mismo trabajo que Vista en el modelo MVC.
- **La abstracción** se encarga del manejo y almacenamiento de los datos, aunque puede haber más agentes encargados de esta tarea. Una representación multi-agente para la abstracción.
- **El control** se equipara al controlador en MVC. Recibe eventos disparadores desde la presentación o desde la abstracción y da respuesta a ellos respecto a las reglas de funcionamiento que se hayan impuesto dentro de él. De igual manera, puede tener una visión multi-agente.

En el contexto del patrón PAC un agente es un componente de proceso de información que incluye: eventos para recepción y transmisión, estructuras de datos para mantener un estado, un proceso que administra eventos entrantes, actualizaciones, su propio estado y puede producir también nuevos eventos. Los agentes pueden ser tan simples como un objeto o tan complejos como un sistema de software completo.
(Fuente: http://moosehead.cis.umassd.edu/cis390/slides/08_PAC.pdf)

Un ejemplo clásico de una arquitectura PAC es el sistema para el control de tráfico aéreo, que como bien sabemos es una de las aplicaciones de software con alta demanda y que involucra un tiempo real absoluto: Un agente PAC toma los datos de entrada de un sistema de radar acerca de la localización de un avión 747 que está por llegar y usa el componente de presentación para “dibujar” una imagen en una pantalla. Otro agente, de forma independiente, toma los datos de entrada de otro avión que está despegando en ese momento y dibuja también una imagen del suceso en la pantalla. Otro agente más se encuentra monitoreando los datos sobre el clima, etc.

(Fuente: http://www.garfieldtech.com/blog/mvc-vs-pac)

Se puede observar que cada agente mantiene su propio estado e información asociada de manera coordinada pero al mismo tiempo independiente del resto de los agentes que conforman el sistema.

No tendría sentido evaluar detalladamente dos patrones que sólo se diferencian en el nombre, supuestamente los tres componentes de ambos patrones, MVC y PAC, son idénticos. No obstante, la diferencia radica, principalmente, en el número de capas o agentes que se dediquen en la solución del problema que se ataca, y en la división lógica de cada una de ellas.

Esta división lógica se puede explicar al ver una capa que conforma al patrón MVC, por ejemplo, para la vista en PAC su equivalente es la presentación; en el patrón PAC puede estar conformada por una cantidad más amplia de subsistemas que en la capa del patrón MVC.

Al final, se debe tomar como una única entidad, sea cual sea su conformación (uno o muchos subsistemas) la capacidad de abstracción de su funcionamiento dará paso a la implementación limpia y de baja dependencia.
Este patrón arquitectónico tiene la posibilidad de ser flexible a la hora de conformar su estructura, la misma flexibilidad lo hace adaptable a las necesidades del usuario y del mismo entorno donde sea que se instale. Por lo tanto, también puede describirse como un sistema adaptable.

3.3. Patrones arquitectónicos en sistemas adaptables

Un sistema adaptable es aquel que se modifica en función de las circunstancias específicas que se presenten en ese momento particular. Las circunstancias

pueden ser modificaciones no predecibles en el ámbito de aplicación de sistema, variables no consideradas en la concepción inicial del diseño de éste. La entropía, que es el **desorden** de un sistema y que puede tener origen en el interior o el exterior. Los sistemas adaptables son tolerantes a la entropía, a estas variables no consideradas y a las modificaciones no esperadas, sea cual sea el origen de éstas.

Un sistema adaptable, además de tolerar las modificaciones del contexto de aplicación (entropía externa), también tolera fallos de diseño (entropía interna) compensándolas con su principal característica, la adaptabilidad.

Dentro del contexto de los sistemas adaptables surgen retos para el arquitecto de software al generar un diseño que resuelva un sistema que abarque la funcionalidad y características anteriormente descritas: tolerar modificaciones y fallos así como la habilidad de adaptarse a los cambios en su entorno. Para el logro de este objetivo se aplican los patrones arquitectónicos proxy, microkernel y reflection que serán analizados a continuación.

3.3.1. Ámbito de aplicación

Con el avance de la tecnología los sistemas se vuelven cada vez más complejos. La adaptabilidad está emergiendo como un importante requerimiento no funcional de los sistemas actuales, entre los cuales se pueden mencionar sistemas de información, e-business, entre otros. Como ya se analizó, la adaptabilidad es la habilidad de un sistema de acomodarse a los cambios en su entorno. Por entorno se entiende cualquier aspecto observable por el sistema de software, tal como: información capturada por el usuario final, dispositivos de hardware, sensores, etc.

El sistema observa su propio entorno y analiza estos resultados para determinar las modificaciones apropiadas y para su administración se requiere una variedad de agentes. (“An Architecture-Based Approach to Self-Adaptive Software” Whitepaper. 1999. Oreizy P., Gorlick M., Taylor R., Heimbigner D., Johnson G., Medvidovic N., IEEE Intelligence systems)

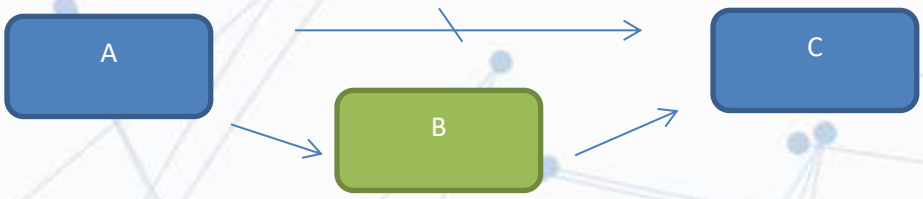
Un sistema adaptable pretende atacar y resolver algunos retos de la ingeniería de software tales como: el creciente costo de mantenimiento de un sistema debido a su constante evolución y modificación, los requerimientos de flexibilidad, los cada vez más impredecibles ambientes de operación, cambios en hardware, etc.

3.3.2. Proxy y sistemas adaptables

El *proxy* en el contexto de sistemas adaptables se entiende como un patrón de diseño, y se llama así porque tiene el mismo modo de hacer las cosas que un proxy de red.

Un *proxy de red* es aquel programa o dispositivo (cualquier tipo de hardware) que toma una acción a manera de representar a otro dispositivo (Couluris et ál., 2011).

Por ejemplo, si un dispositivo A tiene la necesidad de hacer una petición a un dispositivo C, envía esta petición por medio del dispositivo B, y así C nunca se enterará que en realidad la solicitud viene de A. Una representación gráfica del ejemplo anterior se presenta en la siguiente imagen:



El patrón de diseño proxy tiene un propósito específico: controlar el acceso de un objeto hacia otros mediante una intermediación. La aplicación de este tipo de patrón de diseño toma sentido en algunos casos muy especiales.

El siguiente caso de estudio te servirá para comprender el patrón de diseño proxy: La computación en la nube ha tomado un lugar muy importante dentro de la infraestructura de tecnologías de la información. Si un usuario tiene sus documentos personales publicados en algún servicio **en la nube** depende completamente que estén disponibles en cualquier lugar y en cualquier dispositivo (computadora, teléfono inteligente, tableta electrónica). Cuando se trabaja con este tipo de plataformas de nueva generación se depende completamente de tener disponible una conexión a internet para acceder a la información del usuario. El transporte de la cantidad de datos (el tamaño total del archivo al que se quiere acceder) es un tema importante, porque si hay cobro por cantidad de datos transmitidos la cuenta se puede elevar.

Debe pensarse en un caso hipotético: un usuario desea revisar una presentación que tiene hecha en una hoja de cálculo, donde obtiene estadísticas sobre la población de México, estas estadísticas están ligadas a gráficos de gran calidad visual en alta definición. El acceso a su hoja de cálculo lo hace en su tableta electrónica mientras hace un viaje de la ciudad **A** a la ciudad **B**. Para el usuario que requiere soluciones tecnológicas, lo importante es tener sus datos en el momento y el lugar que los requiere. Según el caso de estudio presentado, cuando el usuario acceda a su documento no es necesario crear todas las gráficas en un solo momento, porque es un proceso que involucra el transporte de una gran cantidad de datos y un consumo importante de memoria física del dispositivo y tiempo del procesador. La solución ideal es sólo dibujar los elementos gráficos

que estén a la vista del usuario, porque no tiene caso consumir recursos computacionales en elementos que, tal vez, no se vayan a necesitar.

El lugar de estas imágenes que no se cargan está ocupado por un elemento proxy, que hace la emulación de la carga, pero **bajo demanda**. La carga bajo demanda es el concepto aplicable cuando se accede a un elemento (o sus datos) sólo cuando se requiere, cuando se demanda su uso.

La hoja de cálculo en lugar de hacer referencia directa a los gráficos, lo hace a un elemento proxy, y éste hace referencia a los gráficos, cuando se necesiten. Este proceso debe ser totalmente transparente al usuario, porque la arquitectura que se elabore deberá solventar la problemática descrita.

Otro concepto importante del patrón proxy es la palabra **referencia** que se entiende como la triangulación de acceso, porque en lugar de hacerlo de manera directa, lo hace a través de un tercero.

Se debe aclarar que su uso es más allá que una simple liga, su uso se centra cuando se requiere un objeto más sofisticado que la simple ligadura, que tenga un comportamiento controlable, como la carga dinámica en el ejemplo del caso de estudio. Los tipos de proxy pueden ser:

- **Remoto:** accede a la distancia a un objeto y lo representa en el contexto local.
- **Virtual:** una representación simulada del objeto en caso de que se requiera el acceso a éste.
- **Protección:** verificación de permisos de acceso.

Las consecuencias directas del uso del proxy pueden listarse dependiendo del tipo que se use; en general el uso del proxy oculta al usuario y a algunos elementos del propio sistema el acceso a través de referencia a otros objetos. El uso adecuado de cada tipo de proxy se lista a continuación:

- **Proxy remoto:** oculta a varias capas de aplicación y al usuario final el acceso a datos u objetos remotos dando representación como referencia local.
- **Proxy virtual:** utilizado en objetos bajo demanda (como en el caso de estudio), optimiza el uso de espacio, memoria o procesamiento, debido a que sólo adquiere los objetos que necesita cuando los necesita.
- **Proxy de protección:** verifica el acceso a los objetos referenciados y permite tareas de mantenimiento distintas al acceso, por ejemplo disminuir la carga de trabajo de un objeto referenciado.

Este patrón de diseño puede tener utilidad en la resolución de problemas arquitectónicos donde se tenga la necesidad de hacer representación de objetos (locales o remotos).

3.3.3. Microkernel

Uno de los patrones que es aplicable en el diseño de sistemas adaptables es el *microkernel*, el cual se utiliza en sistemas con requerimientos cambiantes, es un patrón complejo y sofisticado que a su vez guarda cierta similitud con el modelo en capas. Este patrón realiza una separación entre un núcleo de funcionalidad mínima de la funcionalidad extendida del sistema. Los sistemas microkernel principalmente se han aplicado en relación al diseño de sistemas operativos sin embargo este patrón también es factible de ser aplicado en otro tipo de ámbitos como por ejemplo, aplicaciones financieras y sistemas de bases de datos.

Fue primordialmente creado para el diseño de arquitecturas de software enfocadas al desarrollo de diversas aplicaciones con interfaces de programación similar, las cuales se construyen sobre el mismo núcleo pequeño, eficiente y portable proveyendo al mismo tiempo el soporte para agregar extensiones con nuevos servicios.

Provee la base de la arquitectura para algunos sistemas operativos, donde las operaciones principales o centrales se controla por medio de un núcleo **Kernel** y se presenta una interfaz al usuario controlada por el entorno de aplicación **Shell**, ambos conceptos conocidos en un sistema de este tipo. Tal como se mencionó anteriormente, una característica esencial del patrón es que se realiza una separación entre las funciones de bajo nivel del sistema operativo y las funciones de interpretación de comando.

Algunas de las funciones del microkernel son (F. Bushmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal (1996). *Pattern-Oriented Software Architecture*. Wiley. West Sussex PO19 IUD. England):

- Manipulación de las interrupciones del sistema.
- Sincronizar las operaciones del multiprocesador.
- Manejo de las excepciones que se puedan presentar en el procesador.
- Brinda un soporte para la recuperación del sistema en caso de una posible falla eléctrica.

Al mismo tiempo el patrón microkernel define cinco tipos de componentes participantes (F. Bushmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal (1996). *Pattern-Oriented Software Architecture*. Wiley. West Sussex PO19 IUD. England):

- **Servidores internos:** Comunicación con el microkernel. Pueden ser considerados como extensiones del microkernel y solo son accesibles por el mismo. Un ejemplo son los drivers de los dispositivos.
- **Servidores externos:** Comunicación con el cliente. Es implementado como un proceso por separado que provee su propia interface de usuario.
- **Adaptadores:** Representan la interface entre el cliente y el servidor externo y permite acceder a sus servicios
- **Clientes:** aplicaciones que acceden a la funcionalidad del microkernel por medio de las interfaces proveídas por los servidores externos.
- **Microkernel:** Constituye el elemento principal e implementa los servicios centrales.

Finalmente, el patrón microkernel ofrece los beneficios siguientes (F. Bushmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal (1996). *Pattern-Oriented Software Architecture*. Wiley. West Sussex PO19 IUD. England):

Portabilidad: Debido principalmente a que no es necesario portar los servidores externos o las aplicaciones del cliente, si se exporta el sistema microkernel a un nuevo ambiente.

Flexibilidad y extensibilidad: Si se requiere implementar una nueva vista todo lo que se necesita hacer es agregar un nuevo servidor externo que extienda el sistema con funcionalidad adicional.

Separación de las políticas y del mecanismo: El componente microkernel provee todos los mecanismos necesarios para habilitar los servidores externos y permitir la implementación de sus políticas. Esta estricta separación de políticas y mecanismos facilita cualquier mantenimiento o cambio del sistema completo.

Escalabilidad: El sistema microkernel es aplicable ya sea para el desarrollo de sistemas operativos, sistemas de bases de datos para computadoras distribuidas en una red donde es sencillo escalar el microkernel a una nueva configuración cuando una computadora es agregada a la red.

Transparencia: Permite que cada componente acceda a otros componentes sin necesidad de conocer su distribución física.

Como ya se mencionó anteriormente, el patrón de arquitectura Microkernel es utilizado principalmente para el desarrollo de sistemas operativos, de hecho un sistema operativo de amplia difusión estructurado bajo este modelo es Windows NT.

3.3.4. Reflection

El patrón de arquitectura Reflection, perteneciente a la categoría de sistemas adaptables, provee un mecanismo dinámico para una estructura cambiante del software. Como bien sabemos, los sistemas de software involucran de forma constante cambios a lo largo del tiempo y cambios no anticipados pueden ser con frecuencia requeridos y es complicado hacer frente de una forma automática a situaciones que no se prevén. (F. Bushmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal (1996). *Pattern-Oriented Software Architecture*. Wiley. West Sussex PO19 IUD. England)

En el patrón de arquitectura Reflection, todos los aspectos estructurales y de entorno del sistema son almacenados en meta-objetos y separados de la aplicación en componentes lógicos, de tal manera que el sistema es diseñado bajo

una arquitectura que permite hacer frente a situaciones imprevistas de una forma automática.

Se organiza en dos capas principales: el **Meta Nivel**, que brinda al software un conocimiento adecuado sobre su estructura y comportamiento y está compuesto por meta objetos con estructuras, algoritmos, tipo y llamado a funciones y el **Nivel Base,** que define las aplicaciones lógicas y utiliza los meta objetos para mantener la independencia de los factores cambiantes y a su vez la comunicación a través de ellos.

El Meta Nivel encapsula los componentes internos de la aplicación que pueden adaptarse en Meta Objetos y la información almacenada en el mismo (metadato) describe los atributos de la aplicación y los aspectos que pueden cambiar. El metadato puede ser representado mediante XML pero no es la única forma.

El Nivel Base implementa la lógica de la aplicación y utiliza la información proporcionada por el Meta-Nivel la cual interpreta en tiempo de ejecución para adaptar la aplicación. (F. Bushmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal (1996). *Pattern-Oriented Software Architecture*. Wiley. West Sussex PO19 IUD. England)

El patrón Reflection tiene otro nivel adicional: el Meta Data Protocol (MOP), el cual define como deben ser realizados los cambios en el Meta Nivel. Muchos lenguajes de programación actuales implementan ciertas capacidades que pueden ser consideradas como Reflection, la idea básicamente es la misma pero no todo lenguaje de este tipo puede ser utilizado para una arquitectura de software base.

Por ejemplo, en una aplicación distribuida los componentes del Nivel Base deben únicamente comunicarse entre ellos mediante un meta-objeto que implementa un

mecanismo específico para el paso se mensajes. Cambiar esta meta-objeto ajustaría la manera en que los componentes del nivel base se comunicarían pero no el código en sí de la aplicación (código del Nivel Base).

Muchas aplicaciones existentes como intérpretes y máquinas virtuales son reflectivas, en el sentido de que su base de implementación es realizada a través de este patrón de arquitectura, también se ha vinculado con el área de la Inteligencia Artificial a través de los lenguajes LISP (lenguaje funcional) y PROLOG (lenguaje declarativo).

En una herramienta de análisis de código el Nivel Base lo constituye el código objeto que será evaluado; el Meta Nivel son las clases que analizan el código mientras que el MOP está compuesto por la interfáz de usuario proporcionada para la herramienta de análisis y para la presentación de los resultados.

Por último, podemos deducir que el patrón Reflection ofrece los siguientes beneficios:

- No es necesario modificar el software de forma explícita una vez que ha sido creada una aplicación adaptable bajo este patrón de arquitectura.
- El MOP permite que cualquier cambio que en un determinado momento sea requerido se realice mediante una interface que permita adaptar e integrar de forma segura cualquier modificación.

Cierre de la unidad

En esta unidad se abordó el significado de los distintos patrones arquitectónicos y la importancia de su correcta aplicación en la propuesta de soluciones arquitectónicas.

Se expuso la forma correcta de distinguir las partes que conforman cada patrón arquitectónico; a través de este conocimiento podrás analizar el modo correcto de aplicarlo en cualquier tipo de arquitectura. El conocimiento adquirido y la experiencia de aplicar los patrones y estilos arquitectónicos estudiados en esta unidad didáctica te llevará a ser capaz, incluso, de combinar distintos tipos de patrones arquitectónicos si es que así lo amerita la situación, hacer modificaciones a ellos y tener un estilo propio de solventar la problemática que se presente.

El uso de los patrones y estilos arquitectónicos puede ser una excelente herramienta al momento de diseñar soluciones de software, hacerlo de manera correcta lleva a que la solución sea escalable, sostenible y adaptable al entorno.

Para saber más

Si deseas saber acerca de Arquitecturas de software distribuidas, consulta los siguientes vínculos:

- *Arquitecturas distribuidas cliente/servidor* (s. f.). Recuperado de http://www.fceia.unr.edu.ar/ingsoft/unidad14-4.pdf

- Costilla, C. (2009). *Arquitecturas de los SGBD distribuidos*. Recuperado de: https://vdocuments.mx/ddbms-55cac53a4b625.html

- Scott, R. (s. f). *Eligiendo una arquitectura distribuida para su empresa*.

Fuentes de consulta

- Barnaby, T. (2002). *Distributed .NET Programming in C#*. Apress.
- Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). *Distributed Systems: Concepts and Design* (5ª ed.). Addison Wesley.
- De la Torre, C., Zorrilla, U., Calvarro, J., & Ramos, M. A. (2010). *Guía de arquitectura N-Capas orientada al dominio con .NET 4.0*. Krasis Press.
- Farley, J. (1998). *Java Distributed Computing*. O'Reilly.
- Bushmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-Oriented Software Architecture*. Wiley.
- GarfieldTec. (2006). *MVC VS PAC*. Recuperado de https://www.garfieldtech.com/blog/mvc-vs-pac
- Google. (2024). *What is cloud architecture? Benefits & Components*. Recuperado de https://cloud.google.com/learn/what-is-cloud-architecture
- Jaramillo, S., Augusto, S., & Villa, D. (2008). *Programación avanzada en Java*. Elizcom.
- Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Korgdahl, P., Lou, M., & Newling, T. (2004). *Patterns: Service Oriented Architecture and Web Services*. IBM Reedbooks.
- Newcomer, E., & Lomow, G. (2005*). Understanding SOA with Web Services*. Addison-Wesley.
- Tanenbaum, A. S. (2006). *Distributed Systems: Principles and Paradigms* (2ª ed.). Prentice Hall.
- WINPAL. (2016). *Architecture of Distributed Systems*.