CARRERA: Ingeniería en Desarrollo de software
Cuatrimestre 04

Programa de la asignatura:
Diseño y Arquitectura de Software

Clave: 150920517/ 150920517

Índice

Unidad 3. Aplicación de sistemas

Presentación de la unidad

La consecución del conocimiento es la forma más segura de generar un aprendizaje significativo, en la presente unidad se tendrá que aplicar el conocimiento adquirido y analizado en las dos unidades anteriores, aquí aprenderás la aplicación de arquitecturas de software en ambientes distribuidos, ambientes interactivos y ambientes adaptables.

A la arquitectura base que se generó en la unidad anterior se le adaptarán los nuevos conceptos que se tocarán en el desarrollo de la presente unidad para hacer aplicable a los ambientes antes mencionados. Aprenderás además a decidir y proponer soluciones en base al problema específico y saber qué arquitectura se deberá aplicar según su conveniencia para el tipo de problema que se presente.

Propósito

Generar una arquitectura “tipo” que aplique los conceptos aprendidos en la presente unidad. La aplicación específica de la arquitectura en ambientes distribuidos, ambientes interactivos y ambientes adaptables y que el arquitecto decida el porqué de esta aplicación.

Competencia específica

Diseñar una propuesta de arquitectura para el diagnóstico de información de los usuarios mediante el análisis y uso de herramientas de diferentes tipos de sistema.

Actividad 1. Presentación de arquitectura preliminar

La aplicación correcta de los conceptos adquiridos a través del estudio y consulta de las dos unidades anteriores debe reflejarse de manera clara en una arquitectura que sea representativa del caso de estudio que se planteó en la unidad 2, hasta este momento ya eres capaz de proponer mejoras a diversas arquitecturas. Ahora, compartirás tu arquitectura con los participantes en el foro *Presentación de arquitectura preliminar;* los pasos son los siguientes:

1. **Sube** tu evidencia de aprendizaje de la unidad anterior en formato de imagen digital a un sitio de acceso gratuito (por ejemplo, Lickr o Picasa) y **comparte** la liga en el foro.

2. **Revisa**, por lo menos, tres evidencias de aprendizaje de tus compañeros(as) e **Identifica** en ellas las fortalezas y debilidades, **compártelas** en un comentario ligado al aporte de cada uno(a) de ellos (ellas).

3. Al final, el (la) Facilitador(a) escogerá la evidencia de aprendizaje con el mayor

número de fortalezas; esta arquitectura servirá de base para las siguientes actividades de la presente unidad.

La arquitectura electa estará disponible para todos desde su vínculo en el foro. Dicha arquitectura servirá de base para las siguientes actividades de la presente unidad.

3.1. 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 del 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.

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:

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, pues 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 mencionadas capacidades 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, lo veremos de la siguiente manera:
Una empresa tiene la necesidad de lanzar un nuevo servicio a través de internet a sus clientes 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, como 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, pues 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 en base a 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.

3.1.1. 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, incluso de su mismo género o especie, por lo tanto, las características de un sistema computacional distribuido deben distinguirlo de cualquier otro que no lo sea. Se listarán estas cualidades inherentes deseables para cualquier sistema distribuido (Tanenbaum, 2006), a saber:

- **Cooperación** pues para poder considerarse un sistema computacional 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** pues sin ella no tendría sentido el dividir el problema, pues no se podrían “distribuir” las partes en que se dividió el problema.
- **Un procesador central** que administre la carga, entrega y recepción de información. Su trabajo no es en sí el procesamiento si no la administración del trabajo de los demás agentes (nodos) que componen la red de procesamiento.

- **Dispuestos físicamente separados** pues 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** en comparación contra el procesamiento que hace una sola computadora.
- **Evita el procesamiento central** pues 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, pues 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 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, pues sólo son deseables. Idealmente debería cumplir con todas las características, sin embargo, sólo está exento del 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, pues podría ocasionar ser más cara la solución que el problema en sí, conocer sus ventajas y desventajas nos podrá guiar para tener un punto más de referencia respecto a su elección a la hora de solventar un problema.

3.1.2. Ventajas y desventajas de los sistemas distribuidos

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

Cuando el arquitecto de software ha decidido cuál 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), a saber:

- **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 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:** pues 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, a saber:

- **Diseño lógico:** pues 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:** pues 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 al.*, 2010), (Farley, 1998), (Couluris *et al*., 2011) al proponer la solución que está basada en una arquitectura distribuida, pues el transporte de la teoría a la realidad puede ser compleja.

Como recomendación que se sugiere en la literatura especializada en arquitectura de software, es que se trate de ver la solución distribuida como una cooperación de agentes que son parte de un supra sistema 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.

3.2. 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 en base a 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, si no 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 dio 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.

La variedad de sistemas interactivos es amplia, a continuación listaremos y explicaremos lo más importantes.

3.2.1. 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 3 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** y,
- 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. la lógica del negocio y,
3. los datos de la aplicación.

La separación de las partes de un software, en principio, puede hacerse difícil ante arquitectos inexpertos, pero a la larga trae beneficios para muchas etapas que involucran el desarrollo del software:

- La separación ayuda a resolver el problema por separado, pues cada una de las partes trabaja independiente 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, pues 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, pues 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 claro para el arquitecto. A continuación se explica cada una de ellas:

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 nosotros conocemos el concepto de una red social y hasta somos 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 cómo se despliegue esta información, si está ordenada por fecha, por la persona que hace el comentario, la manera en que podemos ver nuestros álbumes de fotografías, la forma en que nos presenta las conversaciones que tenemos con las

personas que pertenecen a nuestro círculo, dónde está situado el menú, dónde aparece la publicidad, aquí entra en función la capa de la Vista. Analicemos esto: muchas veces nos encontramos con que la configuración gráfica de la página (el acomodo de sus elementos) ha cambiado, tal vez el menú pasó de estar a la derecha por la izquierda, o los colores son diferentes; sin embargo nuestros datos siguen intactos aun cuando estos cambios aparezcan de un día al otro, pues 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.

Actividad 2. Aplicación del patrón Modelo-vista-controlador

En la actividad anterior, el/la Facilitador(a) escogió la mejor arquitectura del grupo. Ahora trabajaremos en pro de mejorarla. Acabamos de revisar el patrón arquitectónico Modelo-Vista-Controlador, por lo tanto, toca añadir sus conceptos a nuestra arquitectura base:

1. **Identifica**, sobre la arquitectura base, los elementos arquitectónicos-modulares que la conforman.

2. **Propón** un cambio de estos elementos identificados para la aplicación del patrón arquitectónico MVC, deberás basarte en las descripciones hechas sobre las capas involucradas en el patrón.

3. **Asegúrate** de que los elementos que sugieres pertenecen a cada una de las 3 capas del patrón MVC; **responde** a la siguiente pregunta: ¿el elemento actual responde afirmativamente a la descripción expuesta para la capa en la cual se está colocando?

4. **Plasma** tu propuesta de mejora de la arquitectura base en formato de imagen digital.

5. **Guarda** la actividad con el nombre DRS_U3_A2_XXYZ. **Ingresa** al apartado de Tareas y **Envía** el archivo a tu Facilitador(a) para recibir retroalimentación.

3.2.2. 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 multiagente 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. Igual puede tener una visión multiagente.

No tendría sentido evaluar 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 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 darán 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 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.

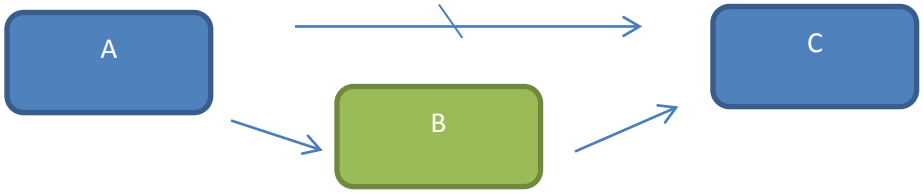
Un sistema adaptable, además de tolerar las modificaciones del contexto de aplicación (entropía externa), también tolera fallos de diseño (entropía interna) compensándolas con su principal características, la adaptabilidad.

3.3.1. 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 al*., 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 venido a tomar 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, pues si hay cobro por cantidad de datos transmitidos la cuenta se puede elevar.

Pensemos 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, pues 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, pues 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.

El documento de la hoja de cálculo, en lugar de hacer referencia directa a los gráficos, lo hace a un elemento proxy, y este hace referencia a los gráficos, cuando se necesiten. Este proceso debe ser totalmente transparente al usuario pues 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, pues el lugar de hacer de manera directa, lo hace a través de un tercero. Aunque 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 este.
- **Protección:** verificación de permisos de acceso.

Las consecuencias directas del uso del proxy puede 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 pues 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).

3.3.2. Administración de los sistemas

En los sistemas interactivos la administración se refiere simplemente a la manera en cómo se hace uso de ellos, cuál es la mejor forma de utilizarlos y cómo se espera que trabajen para que den resultados óptimos. Sistemas que se adaptan a los requerimientos tecnológicos y de demanda de mercado.

Actividad 3. Sistemas distribuidos

Recuperando la actividad anterior, la arquitectura base ya cuenta con elementos del patrón MVC, por lo tanto, el objetivo de la presente actividad es **dotar** a nuestra propuesta de elementos de arquitectura distribuida. Lo que se te pide es:

1. **Identifica**, sobre la arquitectura base, las distintas capas que la conforman respecto del patrón MVC.
2. **Ubica** cada una de las capas que conforman la arquitectura en un nivel diferente al que está actualmente, de esta manera dejará de ser local y pasará a ser un ambiente distribuido, haciendo uso de la teoría vista sobre el tema.
3. **Asegura** la correcta comunicación de las capas y los niveles que ahora conforman la propuesta de arquitectura distribuida.
4. **Plasma** tu propuesta en una mejora a la arquitectura base. El resultado de este punto será una nueva propuesta arquitectónica en formato de imagen digital.
5. **Guarda** la actividad con el nombre DRS_U3_A3_XXYZ.
6. **Ingresa** al apartado de Tareas.
7. **Envía** el archivo a tu Facilitador(a) para recibir retroalimentación.

Autoevaluación

Para reforzar los conocimientos relacionados con los temas que se abordaron en esta tercera unidad del curso, es necesario que resuelvas la actividad integradora. Recuerda que es muy importante leer cuidadosamente los planteamientos indicados y elegir la opción adecuada para cada uno.

Evidencia de aprendizaje: Expansión modular y escalable de la arquitectura

Como parte de la evaluación de esta unidad es necesario **modificar** y **adaptar** el modelo de arquitectura que se ha generado en la unidad anterior para que se incluya el soporte de acceso desde dispositivos móviles. Esta modificación deberás hacerla tomando como base lo expuesto en las tres unidades que conforman la materia (Arquitectura, Modelos de arquitectura y Aplicación de sistemas), utlizando capas lógicas y físicas en distintos niveles de la arquitectura propuesta para el caso de la tienda de conveniencia. La secuencia que deberás seguir para la aceptación será la siguiente:

1. **Usa** la arquitectura base, resultado de la consecución de las actividades de esta unidad.
2. **Adapta** y **modifica** la arquitectura base. La propuesta de arquitectura deberá hacerse en un software especializado para realizar esquemas (Día, Microsoft Visio, entre otros).
3. **Añade** a la arquitectura base la propuesta de aplicación sobre el acceso a dispositivos móviles.
4. **Guarda** tu archivo con la nomenclatura DRS_U3_EA_XXYZ.
5. **Envía** el archivo a tu Facilitador(a) a través de la sección Evidencia de

aprendizaje. **Espera** retroalimentación y, en caso de ser necesario, vuelve a enviar tu archivo con los ajustes pertinentes.

*Recuerda que tu documento no deberá pesar más de 4MB.

Autorreflexiones

Además de enviar tu trabajo de la Evidencia de aprendizaje, es importante que **ingreses** al foro *Preguntas de Autorreflexión* y **consultes** las preguntas que tu Facilitador(a) presente, a partir de ellas, debes elaborar tu Autorreflexión en un archivo de texto llamado DRS_U3_ATR_XXYZ. Posteriormente **envía** tu archivo mediante la herramienta *Autorreflexiones*.

Cierre de la unidad

Has concluido el contenido que comprende la unidad número tres de la presente asignatura. Has aprendido el significado de los distintos patrones arquitectónicos y la importancia de su correcta aplicación en la propuesta de soluciones arquitectónicas.

Además conoces 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 arquitectónicos estudiados en esta asignatura te llevará a ser capaz, incluso, de mezclar 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 arquitectónicos puede ser una excelente herramienta a la hora 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:

- Robinson Scott (s/f). *Eligiendo una Arquitectura Distribuida para su Empresa*. (Traductor: Carlos Alexander Zuluaga). Consultado el 12 de Junio de 2012 en: http://mononeurona.org/pages/display/577

- Costilla, C. (2009). *Arquitecturas de los SGBD distribuidos*. Consultado el 12 de Junio de 2012 en:

http://sinbad.dit.upm.es/docencia/grado/curso0910/Tema%20VII%20Arquitecturas%20SGBD%20Distribuidos/2009-10%20Docu%20Todo%20el%20Tema%20VII%20BSDT.pdf

- *Arquitecturas Distribuidas Cliente/Servidor* (s/f). Consultado el 12 de Junio de 2012 en: http://www.fceia.unr.edu.ar/ingsoft/unidad14-4.pdf

Fuentes de consulta

- Tanenbaum, Andrew S. (2006). *Distributed Systems: Principles and Paradigms* (2nd Edition). Estados Unidos. Prentice Hall.
- De la Torre Llorente, C., Zorrilla Castro, U., Calvarro Nelson, J., Ramos Barroso, M. A. (2010). *Guía de Arquitectura N-Capas orientada al Dominio con .NET 4.0*. España: Krasis PRESS.
- Coulouris, G., Dollimore, J., Kindberg, T., Blair, G. (2011). *Distributed Systems: Concepts and Design* (5th Edition). Estados Unidos: Addison Wesley.
- Barnaby, T. (2002). *Distributed .NET Programming in C#*. Estados Unidos: Apress.
- Barley, J. (1998). *Java Distributed Computing*. Estados Unidos: O'Reilly.