**UNIVERSIDAD DE TARAPACÁ**



**DEPARTAMENTO DE INGENIERÍA EN**

**COMPUTACIÓN E INFORMATICA**



Carrera de Ingeniería en Computación e Informática



**ENSAYO I**

**Autor: Eduardo Caldera Coltters**

**Antonio Labraña Núñez**

**Douglas Gómez Jiménez**

**Curso: Sistemas Distribuidos**

**Profesor: Diego Aracena Pizarro**

ARICA, 9 Septiembre 2019

**1.- Arquitectura basada en Niveles**

La arquitectura en capas es muy importante para separar o simplificar la complejidad de cualquier aplicación de red, en donde los desarrolladores definen diferentes módulos o reglas de interconexiones entre equipos informáticos.

* Arquitecturas de dos niveles principalmente cliente/servidor. Consiste que un cliente realiza peticiones a otro programa en donde este envía respuesta.
* Arquitecturas de tres niveles: Es la aplicación donde la interfaz, la lógica del negocio, el acceso a los datos y los datos se encuentran separados.

**Arquitectura de tres capas**

Se define cómo organi­zar el modelo de diseño en capas, que pueden estar físicamente distribuidas, lo cual quiere decir que los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores. Este patrón es importante porque simplifica la comprensión y la organización del desarrollo de sistemas complejos, reduciendo las dependencias de forma que las capas más bajas no son conscientes de ningún detalle o interfaz de las superiores.

**Capa de presentación**

Es la que se encarga de que el sistema interactúe con el usuario y viceversa, muestra el sistema al usuario, le presenta la información y obtiene la información del usuario en un mínimo de proceso.

**Capa de negocio**

Es donde residen las funciones que se ejecutan, se reciben las peticiones del usuario, se procesa la información y se envían las respuestas tras el proceso. Se denomina capa de negocio o capa de lógica del negocio, porque es aquí donde se establecen todas las reglas que deben cumplirse.

**Capa de acceso a datos**

Esta capa es la encargada de almacenar los datos del sistema y de los usuarios. Su función es almacenar y devolver datos a la capa de negocio, aunque para esto también es necesario en algunos casos, que tengan procedimientos almacenados y funciones dentro de la capa.

**Arquitectura de dos capas**

La arquitectura en 2 niveles es, por lo tanto, una arquitectura cliente/servidor en la que el servidor es polivalente, es decir, puede responder directamente a todas las solicitudes de recursos del cliente.

La presentación de la arquitectura de 2 niveles es con la capa de presentación y capa de acceso a datos.

Su importancia es muy importante ya que, si esta arquitectura no existiría la red, ya que esta arquitectura permite cumplir una petición de un cliente a uno o varios servidores y este mostrar los resultados de la petición.

**2.- Arquitectura basada en objetos**

La tecnología de objetos está progresando y para poder proveer los principios de organización para la siguiente generación de la arquitectura cliente/servidor se fundamenta en la Computación Distribuida Basada en Objetos (ODC), la cual está encaminada a la computación cliente/servidor orientada a objetos. Las ventajas que resultan de este ambiente de computación distribuida basada en objetos son las siguientes:

1. Las aplicaciones conformadas por objetos distribuidos son una imagen única de la empresa.

2. Los servidores son transparentes hacia los clientes, los cuales pueden ser implementados como agentes migratorios.

3. Tanto los objetos remotos, como locales, se comunican de la misma manera a través de mensajes.

4. El particionamiento permite a los usuarios un cambio de recursos de cómputo en sus accesos.

 5. La envoltura de los objetos (interfaces orientadas a objetos) está elaborada para la comunicación entre ellos.

 La arquitectura basada en servicios es una arquitectura en la que se pueden establecer conexiones para consumir servicios que provee un servidor determinado, estos servicios pueden ser consumidos por terceras personas ya que esta arquitectura permite brindar información en interfaces estándares. En esta arquitectura hay tres componentes que juegan un papel importante que son: el proveedor del servicio, el registro del servicio y el consumidor del servicio. Generalmente a estos servicios son conocidos como webservices (servicios web) usando como formato principal para el intercambio de datos el estándar XML.

**3.- Arquitectura centrada en los Recursos**

**ARQUITECTURE STYLE**

* Componentes
* Conexión entre componentes (conector)
* Datos enviados entre componentes
* Como estos componentes son reunidos en un sistema

Un componente es una unidad modular bien definida y provee de interfaces que son reemplazables en un sistema. Que un componente pueda ser reemplazado y que el sistema pueda seguir funcionando es importante. Ante la mantención de un sistema, se espera que los componentes puedan estar fuera de orden y que no se baje el sistema completo para tal caso.

El conector es el mecanismo que media la comunicación, coordinación y cooperación entre los componentes. Un conector, por ejemplo, puede ser formado por llamadas a procedimientos, paso de mensajes, flujo de datos. En otras palabras, un conector maneja el flujo de control y datos entre los componentes.

Con los componentes y conectores se pueden crear y clasificar estilos de arquitecturas.

**RESOURCES-BASED ARQUITECTURE**

Uno de los problemas con la composición de servicios es conectar varios componentes que puedan ser fácilmente cambiadas a una integración.

Como una alternativa, un sistema distribuido puede verse como una colección grande de recursos que son individualmente manejadas por componentes. Los recursos pueden ser quitados como añadidos por aplicaciones remotas y, del mismo modo, pueden ser obtenidos o modificados.

Esta nueva aproximación ha sido ampliamente adoptada por la web y conocida como “Representational State Transfer” (REST). Hay 4 características claves que son conocidas como “*RESTful* *arquitectures”*.

* Los recursos son identificados a través de un esquema individual
* Todos los servicios ofrecen la misma interface, que consisten en a lo más 4 operaciones (PUT, GET, DELETE, POST)
* Los mensajes enviados o recibidos por un servicio están completamente descritos
* Luego de la ejecución de un servicio, ese componente olvida todo sobre quien hizo la llamada (esta propiedad es también referenciada como “*stateless execution*”)

Para crear un *bucket* (un objeto en otras palabras) una aplicación puede enviar una consulta *PUT* a través del protocolo que usa este servicio, que es HTTP. Si el objeto o *bucket* ya existe, un mensaje de error HTTP será devuelto.

De forma similar, para saber que objetos están contenidos en un *bucket*, una aplicación puede enviar una consulta *GET* a dicho *bucket*. Será devuelto entonces una respuesta común a través de HTTP con la lista de nombres de los objetos.

**4.- Arquitectura basada en eventos**

**PUBLISH-SUSCRIBE ARQUITECTURES**

A medida que un sistema crece y los procesos pueden fácilmente unirse o salir, se hace importante tener una arquitectura en que las dependencias procesos sea tan ligera posible.

La idea es ver al sistema como una colección de procesos que operan autónomamente. En este modelo la coordinación abarca la comunicación y cooperación entre los procesos, esto enlaza las actividades enteras de los procesos.

Cabri provee una taxonomía de modelos de coordinación que puede ser aplicada igualmente a muchos tipos de sistemas distribuidos. Se puede hacer una distinción entre los modelos a lo largo de dos dimensiones diferentes:

Cuando los procesos son temporal y referencialmente acoplados, la coordinación toma el lugar de un camino directo, que se conoce como “*direct* *coordination*”.

Para “*referential coupling*”, por ejemplo, un proceso puede comunicarse solamente si conoce el nombre o identificador de otro proceso que quiere intercambiar información. Para “*temporal coupling*”, los procesos que intentan comunicarse deberán estar tanto activos como en ejecución. Una llamada telefónica es un buen ejemplo de comunicación directa

Un tipo diferente de coordinación ocurre cuando los procesos están temporalmente desacoplados, pero referencialmente acoplados. Por ejemplo, un buzón de correo. En este caso, no hay necesidad de que dos procesos de comunicación estén ejecutándose al mismo tiempo para el caso de que se quiera una comunicación.

En su lugar, la comunicación toma el rol de poner el mensaje en el buzón de correo. Es necesario ser especificar la dirección de correo del buzón de correo que retendrá el mensaje.

En los sistemas “*referential decoupled*” los procesos no se conocen unos a otros explícitamente. Lo único que un proceso puede hacer es publicar una notificación describiendo la ocurrencia de un evento.

La combinación entre “*referential* *decoupled*” y “*temporally coupled”* de un sistema, forma un grupo de modelos para *event-based coordination*.

Un modelo conocido es la combinación de “*referential and temporally decoupled processes*”, conocidos como *shared data space.* La idea clave es que los procesos se comunican a través a través tuplas que son datos estructurados que consisten en un número de campos (similar a las tablas de una base de datos).

Los procesos pueden poner cualquier tipo de tupla en un “*shared data space*” en función de recuperar una tupla, en donde cualquier tupla que coincida es devuelta. Si un proceso quiere traer una tupla de un *shared data space*, específica los valores de los campos en los que está interesado. Nuevamente, cualquier campo en el que se encuentre alguna coincidencia será retornado.

Obs: Buen resumen, se olvidaron de la pregunta inicial sobre Ingeniería de software 6.0