Arquitectura de Software Tutoría

Juan Ricardo Giadach

juan.giadach@mail.udp.cl

La Casa de la Cascada, Frank Lloyd Wright, 1939.

Contenidos

1. Introducción
2. Atributos de Calidad
3. Arquitecturas de Software Genéricas
4. Patrones de Arquitectura
5. Resolución de ejercicios

Definiciones

“Architecture is defined by the recommended practice as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.” (IEEE)

“The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.” (L. Bass)

Arquitectura

La arquitectura de un sistema

- define la estructura (componentes)
- especifica la comunicación entre componentes
- plantea los requerimientos funcionales
- plantea los requerimientos no funcionales
  - restricciones técnicas
  - restricciones de negocio
  - atributos de calidad


Arquitectura de Software
Requerimientos no Funcionales

Requerimientos no funcionales

- Performance
- Escalabilidad
- Mantenibilidad
- Seguridad
- Confiabilidad
- Integrabilidad
- Portabilidad
- Verificabilidad
- Soportabilidad

Atributos de calidad del software

Requerimientos no funcionales

- **Performance**: respuesta eficiente
  - Sistema debe tener pocos componentes
  - Operaciones criticas concentradas en uno o dos componentes
  - Disminuir la comunicación entre componentes
  - Bajo uso de la red y operaciones de entrada/salida

Requerimientos no funcionales

- **Escalabilidad**: aceptar mayor carga sin degradación
  - Componentes auto-contenidos
  - Componentes con bajo acoplamiento
  - Crecimiento horizontal

Requerimientos no funcionales

- **Mantenibilidad**: nuevos requerimientos funcionales
  - Componentes auto-contenidos
  - Especificación detallada
  - Componentes reemplazables
  - Evitar datos compartidos

Requerimientos no funcionales

- **Seguridad a nivel usuario**:
  - Autenticación
  - Autorización
  - Perfilamiento

- **Seguridad de datos**:
  - Encriptación
  - Integridad
  - No repudio

Requerimientos no funcionales

- **Confiabilidad**: disponible, recuperable
  - Componentes redundantes
  - Fácil migración de componentes
  - Bajo acoplamiento
  - Control distribuido
  - Operación activo-activo o activo-pasivo

Requerimientos no funcionales

- **Integrabilidad**: integración con otros sistemas
  - Interoperación
  - Publicación de API´s

Requerimientos no funcionales

- **Portabilidad**: independencia de plataforma

Requerimientos no funcionales

- **Verificabilidad**: autotesteo
  - Transacciones dummy
  - Procesos de verificación operativa

Requerimientos no funcionales

- **Soportabilidad**: diagnóstico y corrección de incidencias
  - Diseño modular
  - Integración continua (CI / CD)
  - Bitácora de procesos
  - Gestión de logs
  - Documentación actualizada

“The only way to go fast, is to go well” (R.Martin)

“If you think good architecture is expensive, try bad architecture” (B.Foote & J. Yoder)

“Todo beneficio obtenido tiene un costo asociado” (JRG)

Arquitecturas de Software Genéricas

Marina Bay Sands Hotel (2010, Singapur)

Diseño de Software

Fase 0: **Analizar el contexto**

- Requerimientos funcionales
- Requerimientos no funcionales
- Ambiente Operacional
- Restricciones

Fase 1: **Estructuración**

- Identificar componentes y sus relaciones

Fase 2: **Modelo de control**

- Comportamiento de los componentes

Fase 3: **Descomposición modular**

- Diseño detallado

Estructuración

* Descompone el sistema en un conjunto de componentes
* Utiliza un diagrama de bloques que muestra la estructura del sistema
* Indica el flujo de datos entre los componentes del sistema
* Muestra las interfaces que provee el sistema

Modelos de estructuración

- Cliente/Servidor
- Arquitectura Orientada a Servicios (SOA)
- Modelo de Capas
- Modelo Repositorio
- Objetos distribuidos
- Arquitectura Cloud

Cliente / Servidor

Estructurado en base a

- Servidores que ofrecen servicios específicos
- Clientes que requieren servicios
- Redes de comunicación que conectan ambos

Es un modelo parcialmente distribuido

Cliente / Servidor

Ventajas

- Procesamiento distribuido
- Datos distribuidos
- Escalable

Desventajas

- Datos no compartidos
- Administración de datos en cada servidor
- Performance deteriorada
- No hay registro centralizado de servicios

Ejercicio

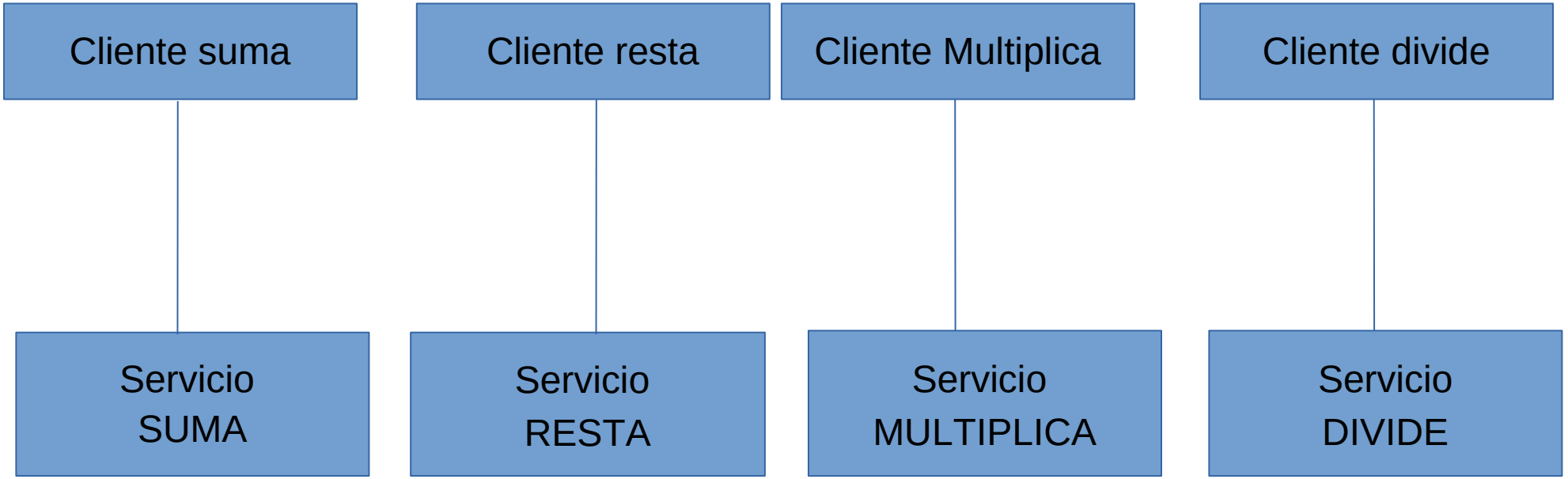
Una empresa requiere disponibilizar a sus clientes un sistema con una calculadora que les permita hacer las cuatro operaciones basicas

- **Sumar**
- **Restar**
- **Multiplicar**
- **Dividir**

Se pide definir este sistema utilizando la arquitectura Cliente / Servidor.

Ejercicio - Calculadora

Estructura del sistema



Arquitectura Orientada a Servicios (SOA)

> Evolución del modelo cliente/servidor
  - Clientes
  - Bus (Enterprise Service Bus)
  - Servicios independientes
> Ambiente de integración basado en estándares:
  - XML, JSON, SOAP, REST
> Servicios administrados centralizadamente
> Conecta aplicaciones como servicios
> Flujo de datos controlado
> Provee escalabilidad y seguridad

Arquitectura Orientada a Servicios (SOA)

Ventajas

- Reutiliza sistemas existentes
- Bajo nivel de acoplamiento
- Escalabilidad, mantenibilidad, interoperabilidad

Desventajas

- BUS es punto único de falla
- Complejidad en entornos grandes

Ejercicio

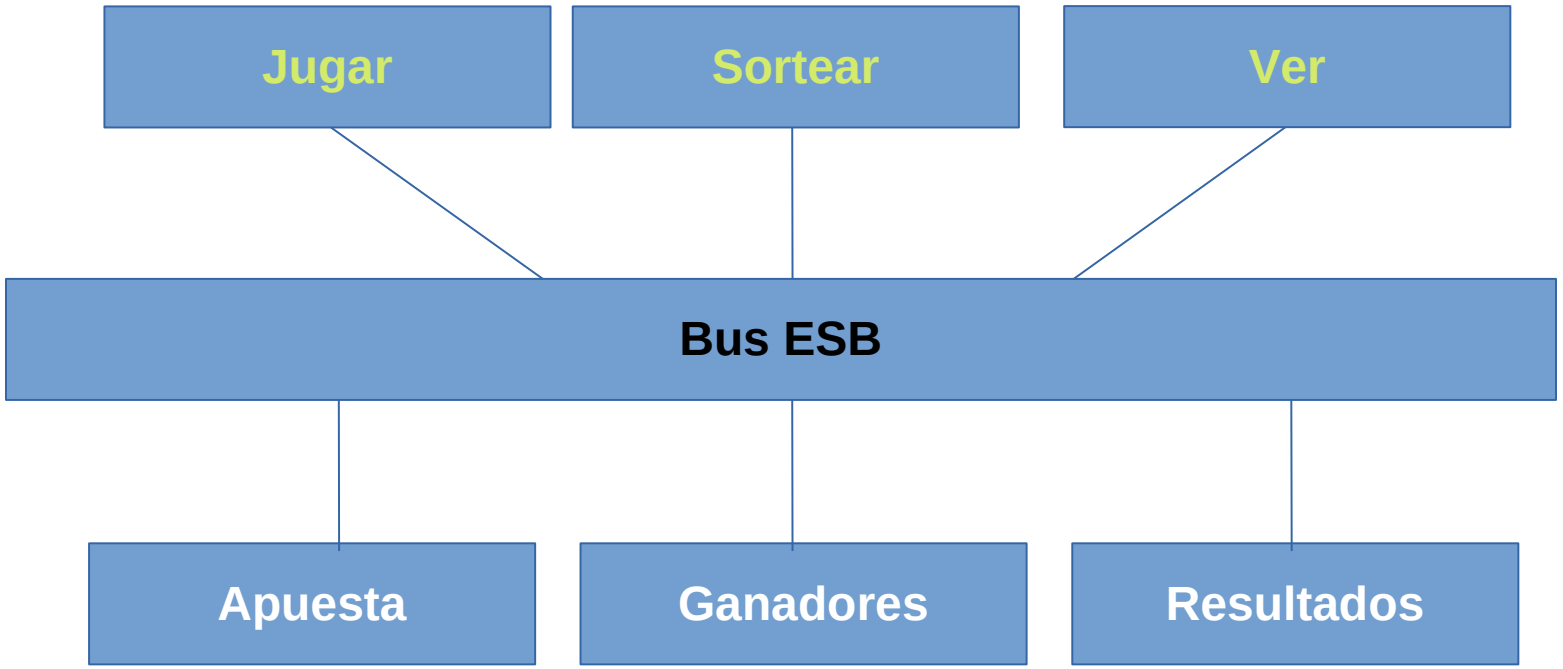
El juego del Loto consiste en escoger seis números entre el 1 y el 41, apostando a que ellos serán los números ganadores.

Se pide desarrollar un sistema que permita:

- **Ingresar las apuestas de los jugadores**
- **Ingresar los números ganadores**
- **Ver el premio obtenido por una apuesta**

Se pide definir este sistema utilizando la Arquitectura Orientada a Servicios.

Ejercicio - Loto



Modelo de capas

Conjunto de capas de software que ofrecen servicios específicos

Cada capa tiene una interfaz claramente definida y aporta en lo que le corresponde al procesamiento de la transaccion

La comunicación es entre componentes de capas vecinas

Desarrollo independiente de las capas

Modelo de capas

Ventajas

- Desarrollo incremental
- Flexible
- Mantenible

Desventajas

- Difícil estructuración
- Dependencias cruzadas
- Baja performance

Ejercicio

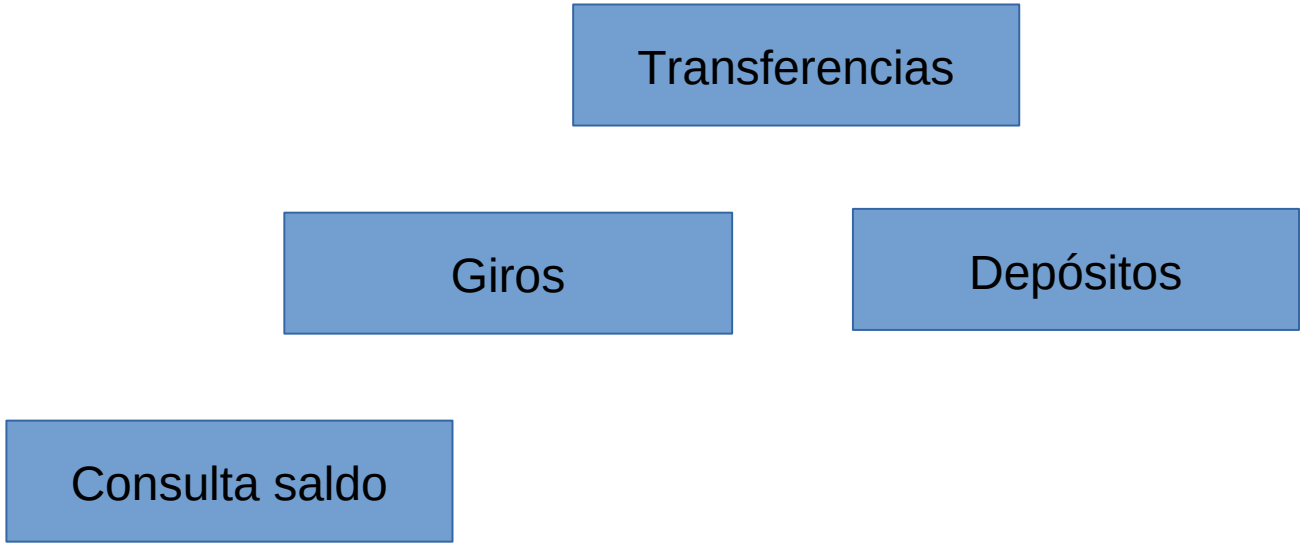
Un banco requiere un sistema para manejar las cuentas corrientes de sus clientes.
Específicamente, requiere tener las siguientes operaciones:

- **Consulta del saldo de una cuenta**
- **Depósito en una cuenta**
- **Giro desde una cuenta**
- **Transferencia entre cuentas**

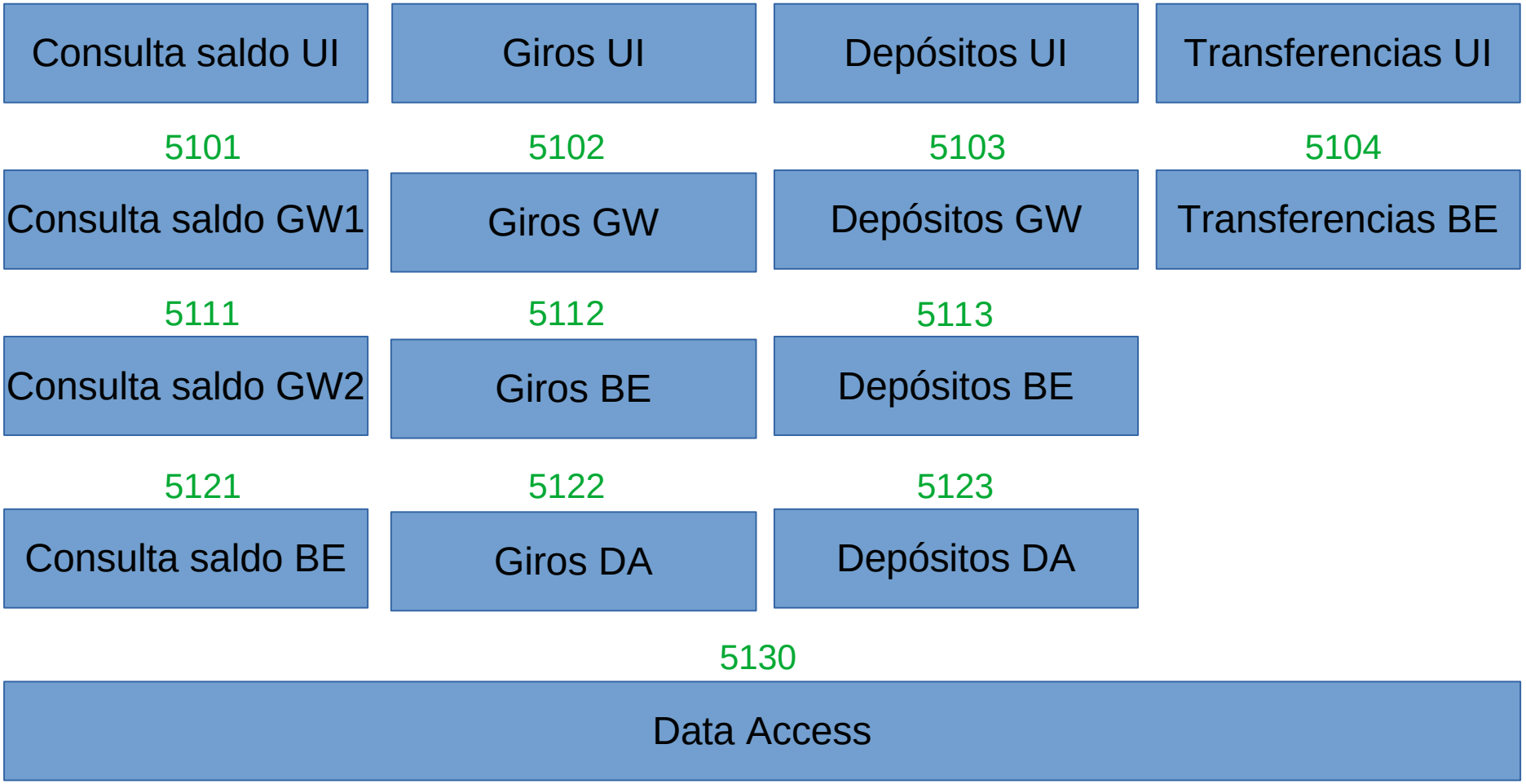
Se pide definir este sistema utilizando la arquitectura de Capas.

Ejercicio - Banco

Dependencia entre funcionalidades



Ejercicio - Banco



Modelo de Repositorio

Útil cuando hay grandes cantidades de datos que deben ser compartidos por varias aplicaciones

- Gestión centralizada de datos
- Almacenamiento centralizado / distribuido

Modelo pasivo

Modelo proactivo

Modelo de Repositorio

Ventajas

- Modo eficiente de compartir datos
- Administración centralizada de los datos
- Seguridad, Escalabilidad, Mantenibilidad

Desventajas

- Difícil cambio del modelo de datos
- Política centralizada de administración
- Punto único de falla

Objetos distribuidos

Cada componente (objeto) define los datos y metodos que pueden ser invocados

Cada objeto puede proveer y recibir servicios

Objetos se comunican a través del sistema ORB (Object Request Broker)

Arquitectura compleja

Objetos distribuidos

Ventajas

- Diseño flexible
- Fácil agregar objetos
- Configuración dinámica
- Escalabilidad, mantenibilidad

Desventajas

- Compleja construcción
- Bajo rendimiento

Arquitectura Cloud

Externalización de servicios computacionales

- Infraestructura - IaaS
- Plataforma - PaaS
- Aplicación – SaaS

Recursos elásticos (pay what you use)

Arquitectura Cloud

Ventajas

- Servicios ubicuos
- Reducción de costos (?)
- Disponibilidad, Escalabilidad, Elasticidad
- Flexibilidad, Movilidad

Desventajas

- Dependencia de ente externo
  - Seguridad
  - Confidencialidad
  - Conectividad

Modelos de Control

Modelos de Control

- Control del flujo de procesamiento entre los componentes
  - Centralizado: un componente controla la ejecución del sistema
    - Modelo Call–Return: cadena de procesos
    - Modelo Administrador: malla de procesos
  - Descentralizado Basado en Eventos: procesa eventos generados asincronamente
    - Transmisión Múltiple: modelo publicador/suscriptor
    - Manejo de Interrupciones: eventos de alta prioridad

Descomposición Modular

Museo Nacional de Catar - 2019

Descomposición modular

- Componentes divididos en módulos
- Posibilita visualizar el modelo de control
- Modelo de Objetos
  - Conjunto de clases débilmente acopladas
  - Crea objetos a partir de las clases
- Modelo de Flujo de Datos
  - Descomposición en procesos funcionales
  - Flujo predeterminado

Arquitecturas de Dominio Específico

- Modelos específicos para algún dominio
- Arquitecturas genéricas
  - Generalización de sistemas reales
  - Análisis de sistemas existentes
- Arquitecturas de referencia
  - Idealización de una arquitectura específica
  - Estudios del dominio de una aplicación
  - Estándar de facto en su dominio

Arquitectura de Software

Patrones de Arquitectura

Patrones de Arquitectura

- Solución de diseño a una necesidad
- Características
  - esquema genérico
  - probado
  - recurrente
- Especificación
  - contexto
  - componentes
  - responsabilidades
  - relaciones

Descripción de un Patrón

- Nombre del patrón
- Contexto
  - situación que origina la necesidad
- Requerimiento
  - necesidad a satisfacer
- Solución
  - estructura (componentes)
  - comportamiento

Clasificación

- Patrones simples
- Patrones para sistemas interactivos
- Patrones adaptables
- Patrones para sistemas distribuidos

Patrones simples

- Capas
- Tubos y filtros
- Pizarrón
- Repositorio

Capas (o Layers)

- Estructura aplicaciones en esquema multi-capa descomponiéndolas en tareas con diferentes niveles de abstracción
- **Solución**:
  - capa base con nivel de abstracción más bajo
  - avanzar capa a capa utilizando los servicios de la capa inmediatamente anterior
  - cada capa está integrada por distintos componentes
  - cada capa expone una interfaz con los servicios que ofrece

Implementación

- determinar el numero de capas según el nivel de abstracción requerido
- asignar responsabilidades a cada capa
- especificar los servicios ofrecidos por cada capa
- definir la estructura de cada capa
- especificar la interfaz de cada capa
- especificar el método de comunicación intercapas
- definir el esquema para el manejo de errores

Análisis

- **Ventajas**
  - componentes estandarizados
  - cambios afectan el nivel local
  - reutilización de capas / componentes

- **Desventajas**
  - cambios afectan en cascada
  - baja performance
  - complejo de definir

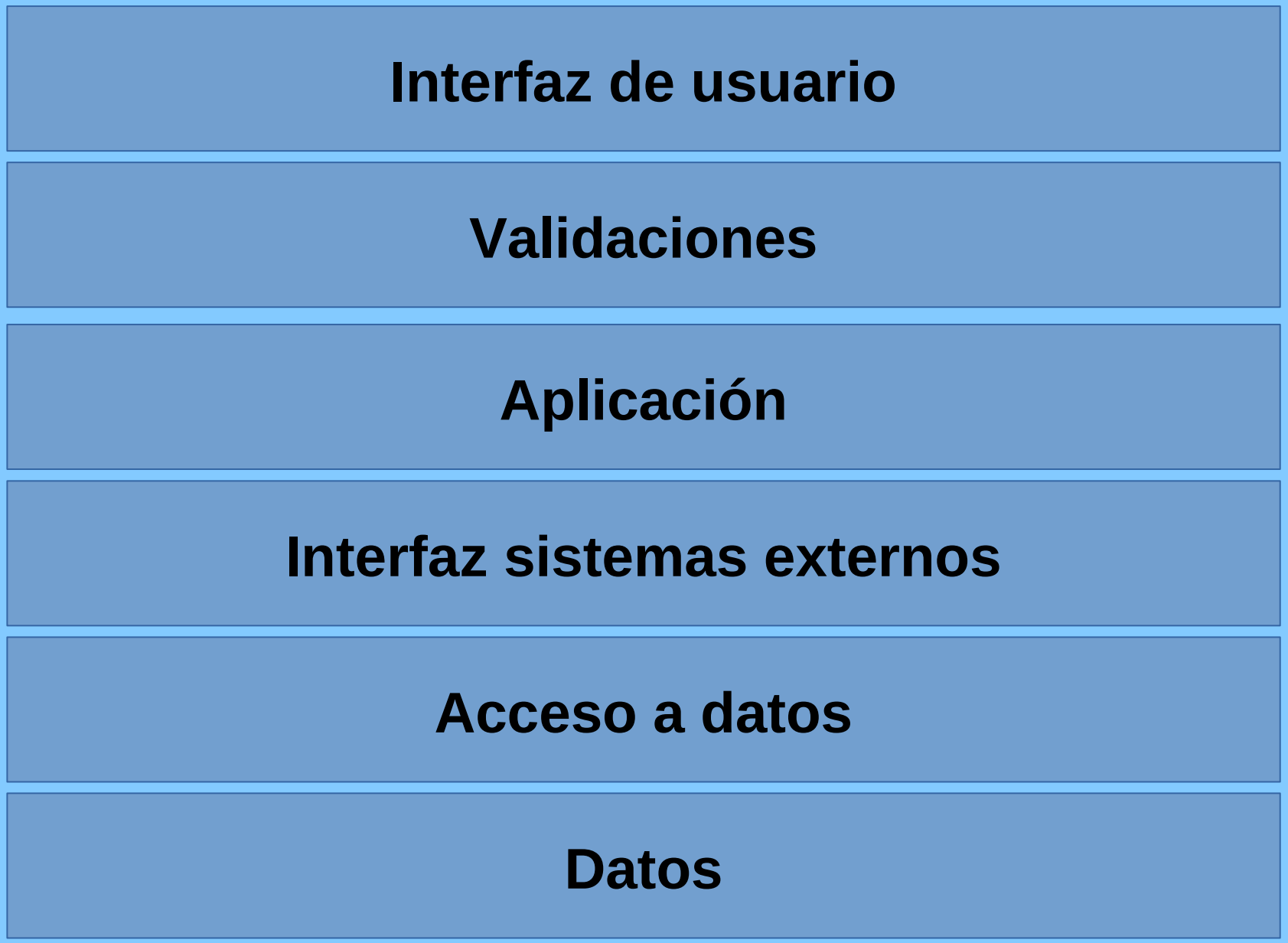
Ejercicio

Una compañía de arriendo de autos requiere mejorar su proceso operativo. En la actualidad, todo el proceso es manual y consta de los siguientes pasos:

- El cliente busca en un catalogo de fotos el vehículo que desea arrendar
- El vendedor verifica la disponibilidad del vehículo seleccionado
- Si esta disponible, el vendedor hace la reserva del vehículo
- El vendedor verifica los datos del cliente
- El vendedor habilita el sistema externo de pagos
- Hecho el pago, se confirma el arriendo del vehículo

Se pide diseñar este sistema utilizando el patrón de Capas

Arriendo de vehículos



Tubos y Filtros

- Procesamiento de flujo de datos mediante
  - Filtros: procesos que transforman datos de entrada en datos de salida
  - Tubos: medio de comunicación entre dos filtros contiguos
- Filtros son independientes
  - no comparten su estado
  - desconocen la existencia de otros filtros

Implementación

- dividir el sistema en una secuencia de procesos ordenados e independientes
- definir el formato de los datos transmitidos por los tubos (rendimiento o flexibilidad ?)
- especificar el procesamiento de cada filtro
- construir los filtros
- definir el esquema para el manejo de errores

Análisis

- **Ventajas**
  - arquitectura flexible
  - no requiere de archivos intermedios
  - filtros reutilizables
  - procesamiento paralelo (eficiencia)
  - construcción independiente
- **Desventajas**
  - información no compartida
  - conversión de datos (ineficiencia)
  - errores pueden afectar el flujo de procesamiento

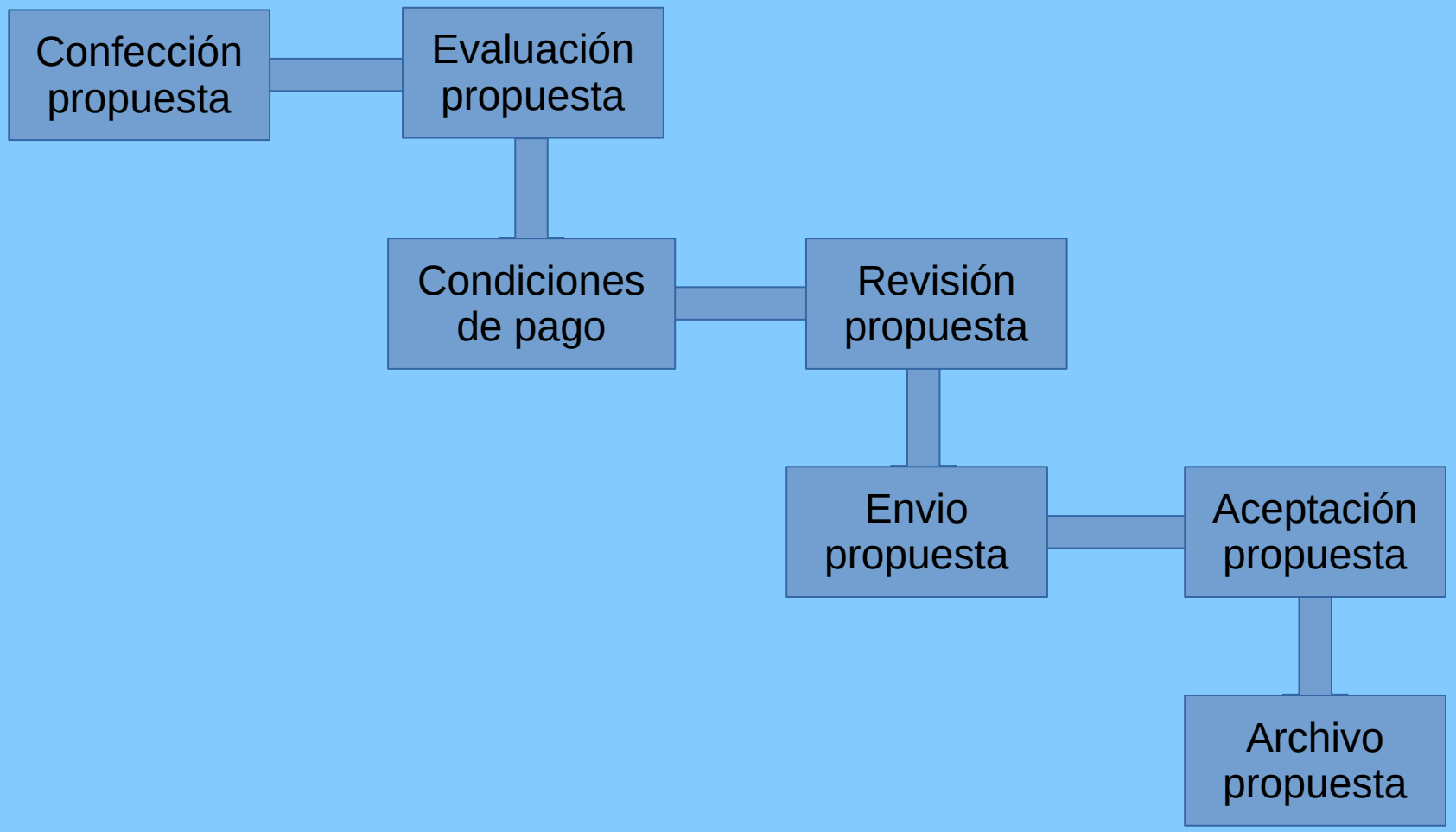
Ejercicio

Una compañía de seguros requiere mejorar su proceso de venta de seguros. En la actualidad, todo el proceso es manual y consta de los siguientes pasos:

- confección de la propuesta de seguro en la que se individualiza al cliente y el objeto a asegurar, hecha por el depto de Ventas
- evaluación de la propuesta por el depto de Análisis de Riesgo para determinar el valor de la póliza
- determinación de las condiciones de pago, hecha por el depto de Finanzas
- revisión de la propuesta, a cargo del depto de Siniestros
- envío de la propuesta de seguro al cliente, coordinado por el depto de Distribución
- recepción de la propuesta aceptada por el cliente, en el depto de Ingreso de Documentos
- archivo de la propuesta en el departamento de Seguros

Se pide diseñar este sistema utilizando el patrón de Tubos y Filtros

Venta de Seguros



Pizarrón

- Patrón útil cuando
  - no hay una solución completa y específica para un problema
  - participan varios sistemas que aportan su conocimiento
  - Ejemplos: inteligencia artificial, reconocimiento de imágenes , toma de decisiones

- **Contexto**: dominio en el que no hay una solución completa

Pizarrón

- **Problema**
  - conocimiento parcial de la solución
  - cada solución requiere diferentes paradigmas
  - el problema abarca muchas especialidades
  - no es factible una solución completa
  - hay soluciones parciales que cubren parte del problema
  - módulos aportan parcialmente a la solución
  - cada sistema contribuyente usa sus datos en diferentes representaciones

Pizarrón

- **Solución**
  - conjunto de sistemas independientes
  - trabajo colaborativo
  - datos compartidos a través del pizarrón
  - cada sistema se especializa en parte del problema
  - controlador centralizado que coordina la ejecución de los sistemas
  - genera soluciones parciales que pueden ser desechadas
  - no hay comunicación directa entre los sistemas

Pizarrón

- **Solución**
  - almacenamiento centralizado de información
  - diccionario de datos del contenido del pizarrón
  - sistemas especializados independientes leen y escriben en el pizarrón
  - monitoreo centralizado del estado del sistema
  - decisión centralizada de las acciones a seguir basada en el progreso alcanzado
  - determinación si la solución es aceptable o insoluble

Análisis

- **Ventajas**
  - arquitectura flexible
  - procesamiento paralelo (eficiencia)
  - construcción independiente
  - integración de sistemas colaborativos
- **Desventajas**
  - información no compartida directamente
  - conversión de datos (ineficiencia)
  - controlador complejo y posible cuello de botella
  - sobrecarga en el proceso

Ejemplos de uso

- **Reconocimiento de textos, voz e imágenes**
  - Combinación de diferentes técnicas de análisis y procesamiento de audio
- **Soporte a la toma de decisiones**
  - Integración de diversos sistemas de análisis de datos
- **Sistemas ETL**
  - Datos provenientes de variadas fuentes
- **Sistemas de Inteligencia Artificial**
  - *“La novedad del año, … pa’ los regalones”*

Repositorio

Grandes cantidades de datos en diferentes plataformas que deben ser compartidos

Es una capa intermedia entre las aplicaciones y los datos

Implementa operaciones CRUD basado en esquemas lógicos

Separa la lógica del negocio de las fuentes de datos => independencia de datos

Análisis

- **Ventajas**
  - arquitectura flexible
  - independencia de datos
  - desarrollo independiente
  - reutilización, modularidad, paralelismo
- **Desventajas**
  - sobrecarga en los procesos
  - duplicación de código
  - agrega una capa adicional (disminuye eficiencia)

Arquitectura de Software

Patrones de Arquitectura

(2)

Patrones para Sistemas Interactivos

- Modelo Vista Controlador
- Presentación Abstracción Control

MVC – Solución

- Tres componentes
  - Comunicación (vista):
    - envía requerimientos del usuario
    - recibe datos (del modelo) y los despliega al usuario
  - Administración (controlador):
    - define el comportamiento del sistema
    - recibe eventos (desde la interfaz de usuario) y solicita servicios al modelo
  - Procesamiento (modelo):
    - provee la funcionalidad requerida por las vistas
    - encapsula el manejo de los datos

MVC – Implementación

- Separar la funcionalidad de la interacción del usuario
- Diseñar e implementar
  - el modelo
  - las vistas
  - los controladores
  - la relación entre vistas y controladores

MVC – Análisis

- **Ventajas**
  - Modelo soporta múltiples vistas
  - Flexible, mantenible, adaptable
  - Frameworks implementan MVC
- **Desventajas**
  - Modelo acoplado con vistas y controladores
  - Vistas sin acceso a los datos (ineficiencia)
  - Complejidad

Presentación Abstracción Control

- Estructura jerárquica de agentes cooperativos
- Proveen la funcionalidad de la aplicación
- Agente es responsable de una parte de la funcionalidad
- Cada agente compuesto por tres componentes
  - Presentación: interfaz
  - Abstracción: modelo de datos y funcionalidad
  - Control: comunicación interna y con agentes

PAC - Implementación

- Definir la funcionalidad central del sistema
- Estructurar la jerarquía de agentes
- Definir e implementar cada agente
  - funcionalidad
  - interfaz
  - modelo de datos
  - mecanismo de control

PAC - Análisis

- **Ventajas**
  - Asigna responsabilidades específicas
  - Funcionamiento independiente
  - Soporta multitarea
- **Desventajas**
  - Sistema complejo
  - Baja eficiencia
  - Complejo mecanismo de control

Tower Bridge, Londres

Ejercicio

Una pastelería de barrio quiere ampliar la cobertura de atención de clientes abriéndose al comercio electrónico. Para ello, requiere el desarrollo de un sistema que cubra los siguientes requerimientos funcionales:

* Recepción de pedidos online
* Disponibilidad de diversos medios de pago (webpay, mercadoPago, transferencias, etc.)
* Organización de los envíos a domicilio
* Gestión de los repartidores
* Estadísticas de venta, diaria, semanal, mensual, anual

Se pide lo siguiente:

* Justificar la elección del patrón de arquitectura a usar
* Diagrama global de la solución indicando como se va a satisfacer la funcionalidad requerida
* Adicionalmente a Confiabilidad y Rendimiento, que deben estar considerados de todas maneras, describa dos requerimientos no funcionales a implementar en el sistema.

Solución

a) Justificar la elección del patrón de arquitectura a usar

Se elige SOA dado que el sistema tiene diversos requerimientos que se pueden implementar en servicios independientes.

Estos estarán enfocados en satisfacer los requerimientos de los distintos tipos de usuarios (clientes, repartidores, administración, gestión).

Entonces, cada tipo de usuario tendrá un conjunto de servicios que los van a atender.

Solución

b) Diagrama global de la solución indicando como se va a satisfacer la funcionalidad requerida

Tal como ya se mencionó, cada funcionalidad va a ser satisfecha por un conjunto de servicios independientes.

Por lo tanto, los clientes generarán transacciones que serán procesadas por ese conjunto.

Entonces, en el nivel cliente estarán los procesos captadores de los requerimientos de los usuarios.

Estos procesos clientes generarán transacciones y las enviarán al bus.

Este, redirigirá la transacción al servicio correspondiente, en donde se hará el procesamiento y retornará la información solicitada hacia el bus, y de ahí al cliente.

Servicios

* *Gestión de productos*
  - *listado de productos*
  - *búsqueda de productos*
  - *detalle de productos*
* *Carro de compra*
  - *ingreso de productos*
  - *eliminación de productos*
  - *listado contenido carro*
  - *valorización contenido*
* *Proceso de compra*
  - *ingreso datos despacho*
  - *selección medio de pago*
  - *redirección a entidad financiera*
  - *recepción respuesta MP*
  - *confirmación compra*

Servicios

* *Despacho de Pedidos*
  - *listado de despachos*
  - *organización delivery*
  - *asignación de repartidores*
  - *confirmación recepción*
  - *cierre de pedidos*
* *Gestión de Repartidores*
  - *ingreso de repartidores*
  - *eliminación de repartidores*
* *Estadísticas de Ventas*
  - *ventas diarias*
  - *ventas semanales*
  - *ventas mensuales*
  - *ventas anuales*
  - *etc.*

Servicios Anexos

* *Gestión de usuarios*
  - *Registro de usuarios*
  - *Administración de claves*

* *Autenticación de usuarios*
  - *validación credenciales de acceso*
  - *perfilamiento de usuarios*

* *Servicio de Datos*
  - *interfaz de acceso a base de datos*

Clientes

* *Cliente Comprador*
  - *identificación del usuario*
  - *construcción del pedido*
  - *pago del pedido*
  - *confirmación del pedido*
* *Cliente Repartidor*
  - *identificación del repartidor*
  - *asignacción de pedidos*
  - *confirmación de entrega*
* *Cliente Operativo*
  - *confección del pedido*
  - *cierre del pedido*
  - *gestión de productos*
* *Cliente Gerencial*
  - *selección de estadística*

Solución

c) Adicionalmente a Confiabilidad y Rendimiento, que deben estar considerados de todas maneras, describa dos requerimientos no funcionales a implementar en el sistema.

Seguridad: sobre todo en el proceso de pago se debe asegurar el correcto procesamiento de este, implementando comunicación encriptada con los diversos procesadores. Ademas, grabar la traza de las transacciones para responder a posibles cuestionamientos.

Mantenibilidad: dado que el negocio esta creciendo, este RNF ayuda a implementar nuevas funcionalidades que vayan siendo requeridas.

Clasificación

- Patrones simples
- Sistemas interactivos
- **Patrones para sistemas adaptables**
  - **MicroKernel**
  - **Reflection**

MicroKernel

- Sistemas que deben adaptarse a cambios en los requerimientos
- Requisitos cambiantes
- Separa funcionalidad en
  - Mínima: común a todos los clientes
  - Extendida: específica de un cliente

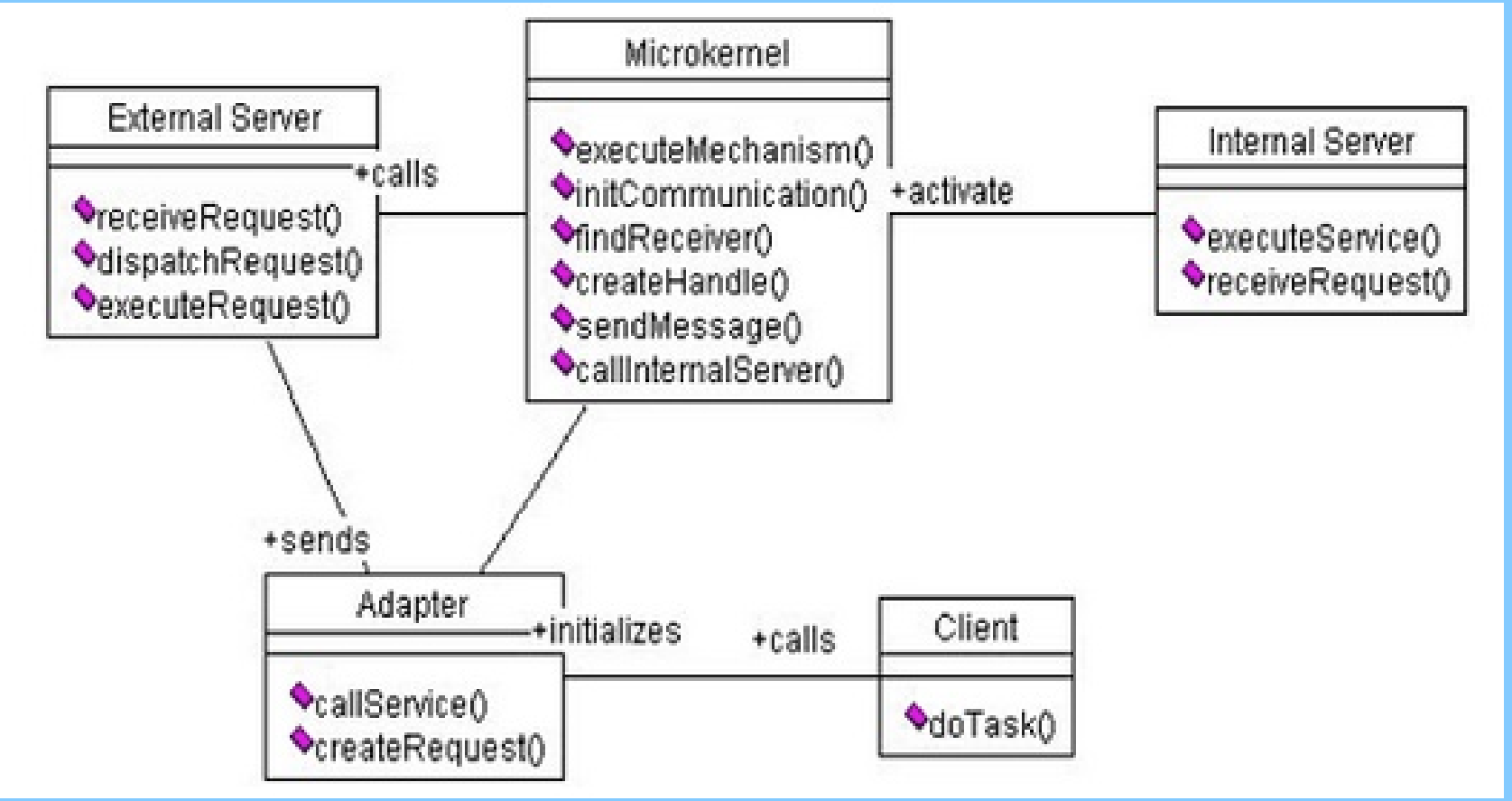
MicroKernel - Requerimiento

- Sistemas que perduran en el tiempo (legacy)
- Deben adaptarse a cambios en la infraestructura (hardware / software)
- Debe ser
  - extensible a nuevas tecnologías
  - portable
  - adaptable

MicroKernel - Solución

- **Microkernel**: componente central (núcleo)
  - Funcionalidad básica
  - Administración de recursos
  - Servicio de comunicación entre componentes
- **Servidores internos**: servicios específicos de la plataforma
- **Servidores externos**: interfaz de acceso al núcleo
- **Adaptadores**: mecanismo de comunicación cliente con servidor externo
- **Clientes**

MicroKernel - Solución



MicroKernel - Análisis

- **Ventajas**
  - Adaptabilidad
  - Portabilidad
  - Flexibilidad
  - Escalabilidad, confiabilidad
- **Desventajas**
  - Sistema complejo
  - Baja eficiencia
  - Compleja implementación

Reflection

- Sistemas que deben adaptarse a cambios en los requerimientos
- Cambio dinámico de la estructura de un sistema
- No requiere intervención manual
- Ejemplo: Frameworks de desarrollo

Reflection

- Separa la funcionalidad en
  - Nivel base
    - lógica de la aplicación
    - funcionalidad principal
  - Nivel meta
    - define el comportamiento
    - manipula las propiedades del sistema
    - interacción de niveles

Reflection - Análisis

- **Ventajas**
  - Adaptabilidad dinamica
  - Extensibilidad
  - Flexibilidad
  - Mantenibilidad
- **Desventajas**
  - Sistema complejo
  - Baja eficiencia
  - Seguridad comprometida
  - Baja soportabilidad

Patrones para Sistemas Distribuidos

- Microservicios

Microservicios

- Colección de pequeños procesos independientes
- Implementan una funcionalidad especifica
- Tienen un bajo nivel de acoplamiento
- Cada microservicio es independiente y solo se comunican de ser necesario
- Un sistema está constituido en base a un conjunto de microservicios

Microservicios – Componentes

- **Servicios**: mini procesos que satisfacen una funcionalidad especifica
- **API Gateway**: controlador y autorizador de acceso
- **Servicio de localización**: ubicación de servicios
- **Plataforma de comunicación**:
  - Sincrónica vía protocolo de comunicaciones – TCP/IP, API REST
  - Asincronica vía colas de mensajes – RabbitMQ, KAFKA

Características

- Servicios sin estado (stateless vs stateful)
- Gestión independiente de datos
- Seguridad centralizada
  - autenticación
  - autorización
  - comunicación encriptada
- Manejo de token de acceso

Microservicios - Análisis

- Ventajas
  - Flexibilidad
  - Mantenibilidad
  - Escalabilidad
  - Soportabilidad
  - Desarrollo en paralelo
- Desventajas
  - Sistema complejo
  - Multiplicidad de servicios
  - Baja eficiencia
  - Seguridad dependiente del microservicio
  - Monitoreo del estado de los microservicios

Arquitectura de Software
Tutoría

Juan Ricardo Giadach

juan.giadach@mail.udp.cl

Puente Magdeburgo Water