

# Taller de Sistemas de Información 2

---

## Introducción



Instituto de  
Computación



Facultad de  
Ingeniería



Universidad de la  
República de Uruguay

# Agenda

- ❑ Introducción
- ❑ Aplicaciones Empresariales
- ❑ Arquitectura de Software
- ❑ Multitenancy

- ❑ Una **Aplicación Empresarial** es una aplicación de software desarrollada para administrar las operaciones, activos y recursos de una empresa
- ❑ Algunos ejemplos:
  - Contabilidad
  - Seguimiento de envíos
  - Servicio al cliente
  - Nómina de empleados
  - Procesos de negocio (ej: Órdenes de compra)

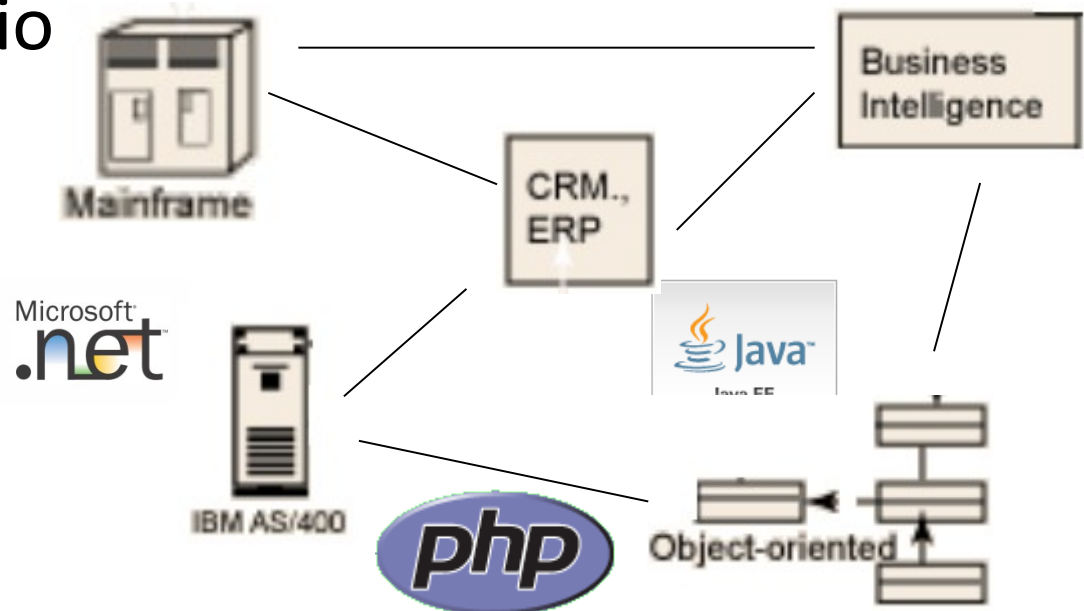
También denominados Enterprise Information Systems (EIS)

- ❑ Las aplicaciones empresariales tienen en general las siguientes características:
  - Involucran persistencia de datos
  - Se manejan grandes cantidades de datos
  - Existen varias interfaces de usuario, para distintos tipos de usuario
  - En general se deben integrar con otras aplicaciones
  - Se accede a los datos de forma concurrente

- ❑ El proceso de desarrollo de una aplicación empresarial típicamente:
  - Programadores de aplicaciones
  - Administradores de base de datos
  - Diseñadores de interfaz de usuario
  - Integradores de aplicaciones

- La creación y mantenimiento de las aplicaciones presenta varias complejidades:
  - Administración
  - Mantenibilidad
  - Escalabilidad
  - Interoperabilidad
  - Seguridad
  - Confiabilidad
  - Accesibilidad y Usabilidad
  - Internacionalización

- Integración de Aplicaciones Empresariales (EAI) es la tarea de hacer que aplicaciones desarrolladas de forma independiente trabajen de forma conjunta con el fin de compartir datos y procesos de negocio



- ❑ Al integrar Aplicaciones Empresariales surgen varios desafíos:
  - Las redes no son confiables
  - Las redes son lentas
  - Las aplicaciones son diferentes
    - a nivel de lenguajes de programación, formato de datos, etc
  - El cambio en las aplicaciones es inevitable
  - Las aplicaciones **están gobernadas por distintos grupos humanos**, con intereses que no alineados



- ❑ Históricamente se han utilizado distintos enfoques para la integración:
  - Transferencia de archivos
  - Base de datos compartida
  - Invocación de procedimientos remotos
    - Comunicación sincrónica
  - Mensajería
    - Comunicación asincrónica

- ❑ Existen plataformas que facilitan el desarrollo e integración de aplicaciones empresariales, brindando solución a los problemas típicos
- ❑ Ejemplos
  - .Net
  - Java EE (la que se utilizará en este curso)
- ❑ Permiten que el desarrollador se concentre en los aspectos relevantes para el negocio
- ❑ Incluyen distintas tecnologías de middleware

- Una de las definiciones más aceptadas es la del Software Engineering Institute (SEI) de la Universidad de Carnegie-Mellon

“La arquitectura de un sistema de software es la **estructura o estructuras** del sistema, que comprende **elementos de software**, las **propiedades visibles externamente** de dichos elementos y la **relación entre ellos**”

# Arquitectura de Software

¿Por qué es importante?

- ❑ Como cualquier otra estructura compleja, el SW debe ser construido sobre una base sólida

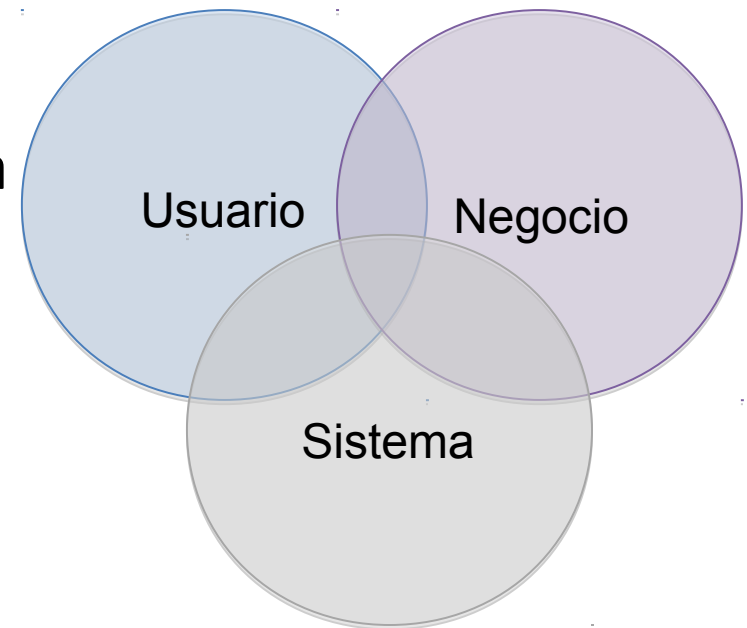


- ❑ Bien definida, la arquitectura permite guiar el proceso de construcción de la aplicación
- ❑ Una arquitectura pobre atenta contra:
  - la simpleza, extensibilidad y mantenibilidad de la aplicación
- ❑ Si bien las plataformas y herramientas modernas simplifican la construcción de aplicaciones, sigue siendo necesario un diseño cuidadoso basado en escenarios y requerimientos específicos.

# Arquitectura de Software

## ¿Por qué es importante?

- ❑ Los sistemas deben ser diseñados teniendo en cuenta:
  - El usuario del mismo
  - La infraestructura tecnológica existente
  - Las metas del negocio
- ❑ Existe un “*trade off*” entre todos estos participantes...



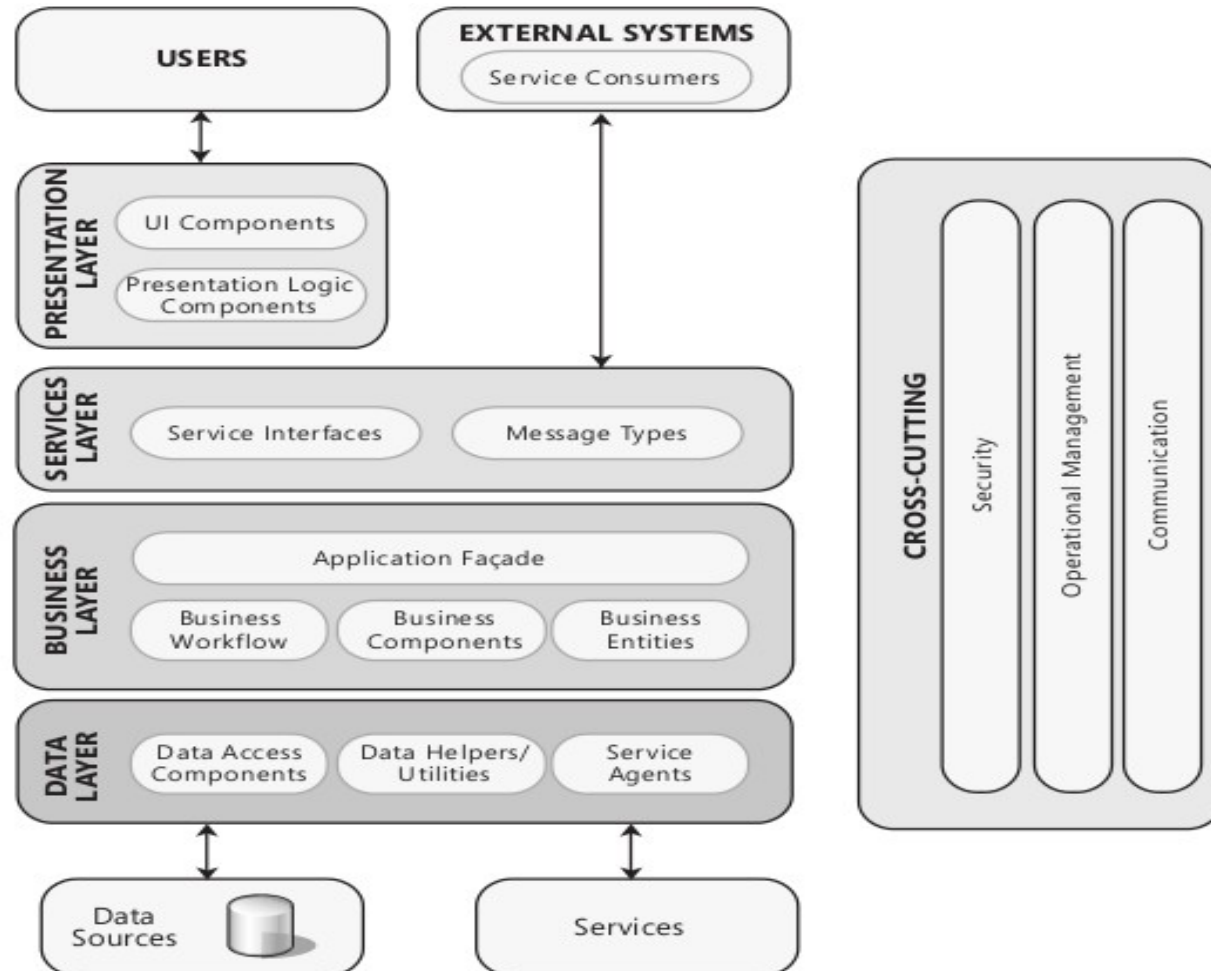
- ❑ La arquitectura debe:
  - Exponer la estructura del sistema pero esconder los detalles de implementación
  - Realizar todos los casos de uso y escenarios de negocio
  - Cumplir con los requerimientos de los involucrados en el sistema
  - Manejar requerimientos funcionales y no funcionales

- ❑ Construir para cambiar, en vez de para durar
- ❑ Modelar para analizar y reducir riesgos
- ❑ Utilizar modelos y visualizaciones como una herramienta de comunicación y colaboración
- ❑ Identificar las decisiones arquitectónicas principales
- ❑ Considerar el utilizar un enfoque iterativo incremental para refinar la arquitectura



# Arquitectura de Software

## Arquitectura Típica de una Aplic. Empresarial



Microsoft Patterns & Practices. Microsoft Application Architecture Guide v2.0




- ❑ Separation of concerns
  - Dividir la aplicación en diferentes bloques, con el mínimo de solapamiento funcional posible
  - Lograr alta cohesión, bajo acoplamiento
- ❑ Single responsibility
  - Cada componente debe ser responsable de una única funcionalidad, o ser agregador de funcionalidad cohesiva
- ❑ Least knowledge (Law of Demeter)
  - Un componente no debe conocer detalles internos de otro componente

- ❑ Don't Repeat Yourself (DRY)
  - La funcionalidad no debe ser duplicada en diferentes componentes
- ❑ You ain't gona need it (YAGNI)
  - Evitar realizar un esfuerzo excesivo en el diseño, sobre todo si los requerimientos no están claros, o si pueden haber posibilidades de evolución
- ❑ Keep it simple (KISS)
  - *A designer knows he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away.*  
(Antoine de Saint-Exupery)

- ❑ Un patrón ofrece soluciones a problemas conocidos
  - Permiten clasificar y re-aprovechar el conocimiento
  - Generan vocabulario
- ❑ Patrones arquitectónicos dado un contexto de aplicación
  - proponen la organización estructural y de comportamiento del software
  - especifican las propiedades que tendrá el sistema al incorporarlos
  - provee un marco abstracto para una familia de sistemas

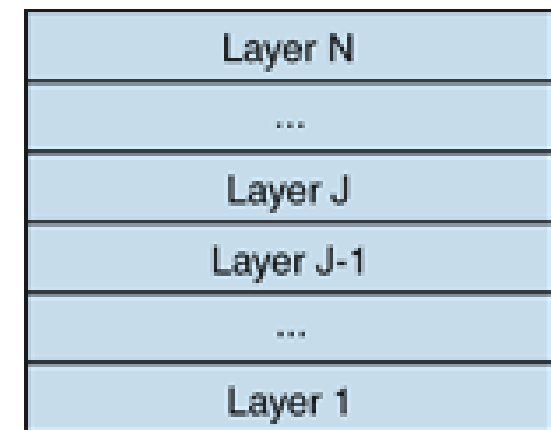
# Arquitectura de Software

## Patrones / Estilos Arquitectónicos (ejemplos)

- ❑ Cliente / Servidor
- ❑ Arquitectura Basada en Componentes
- ❑ Diseño Dirigido por Modelos
- ❑ Arquitectura en Capas 
- ❑ 3-Tier / N-Tier 
- ❑ Arquitectura Orientada a Servicios (SOA) 
- ❑ Message Bus

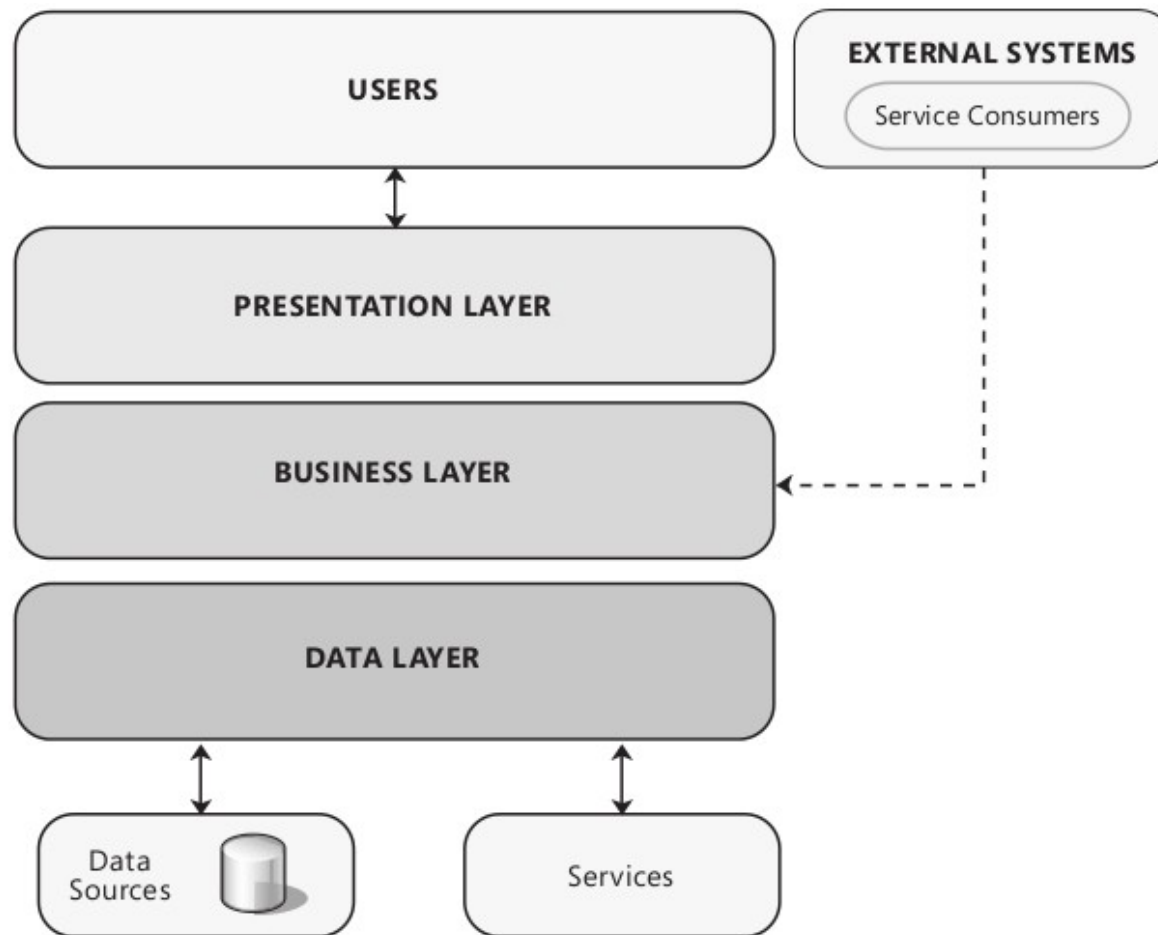
La arquitectura de un sistema de software es a menudo una combinación de estilos para formar el sistema completo.

- ❑ “Layers” es un estilo arquitectónico que comúnmente se utiliza para las Aplicaciones Empresariales
- ❑ En este esquema las capas más altas utilizan servicios definidos por las capas más bajas
- ❑ Esta división lógica entre capas de funcionalidad pueda basarse en distintas responsabilidades



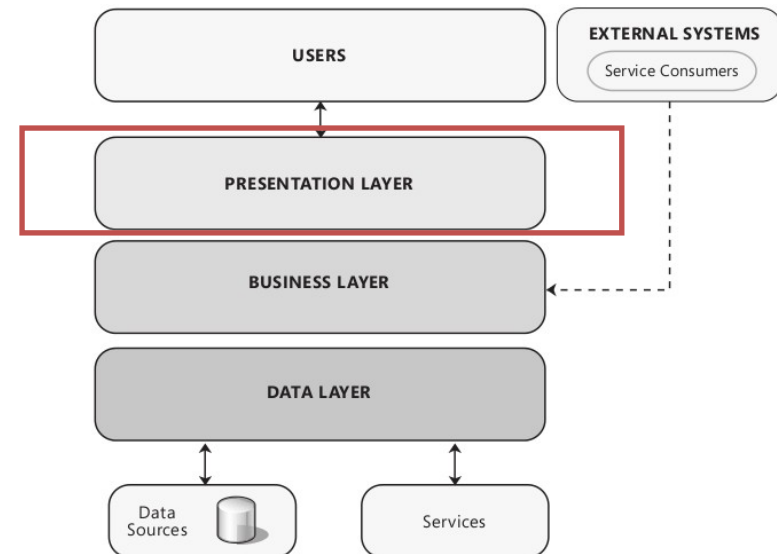
# Arquitectura de Software

## Arquitectura en Capas



### □ Capa de presentación

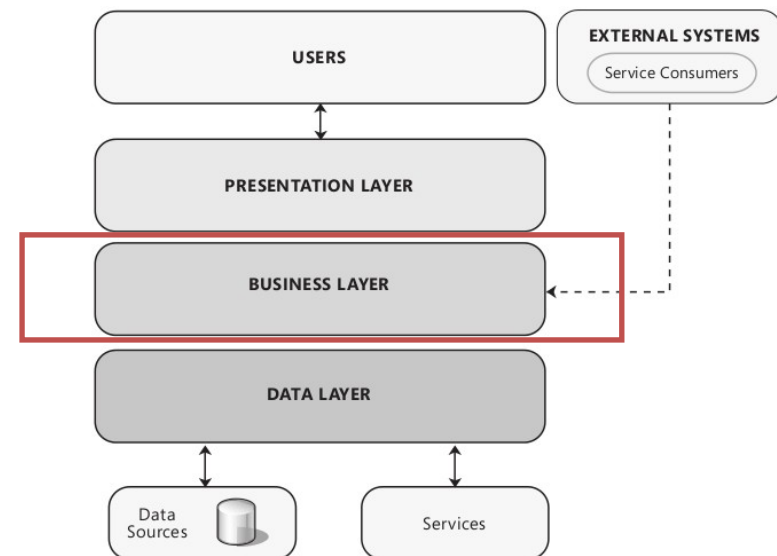
- Contiene la funcionalidad responsable de gestionar la interacción del usuario con el sistema
- Actúa como puente entre el usuario y la lógica de negocio





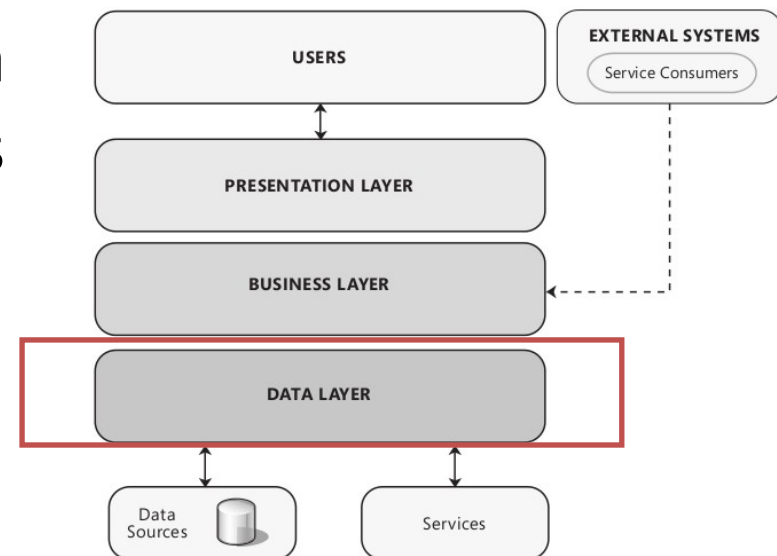
### □ Capa de negocio

- Implementa la funcionalidad central de la aplicación
- Encapsula la lógica de negocio relevante para la aplicación
- Consiste en componentes, los cuales exponen (en algunos casos) interfaces para que otros utilicen



### □ Capa de acceso a datos

- Provee acceso a los datos almacenados en las fronteras de la aplicación, así como a los datos expuestos por otros sistemas de información a los que se tiene conexión
- Los componentes en la capa de negocio hacen uso de los datos provistos por estos componentes



- ❑ Las capas antes presentadas pueden estar ubicadas en la misma locación física (tier) o en diferentes locaciones físicas
- ❑ Si se encuentran en locaciones físicas diferentes, existen fronteras físicas que deben ser tomadas en cuenta en el diseño

- ❑ Al definir la estrategia de deployment hay que optar por un esquema distribuido o no distribuido
- ❑ Si se trata de una aplicación para una Intranet, accedida por un conjunto pequeño de usuarios, en general es conveniente considerar un enfoque no distribuido
- ❑ Si la aplicación es mas compleja, la cual debe se mantenible y escalable, un enfoque distribuido debería ser la elección

- ❑ Este enfoque minimiza el número de servidores requeridos
- ❑ Minimiza el impacto en performance inherente a la comunicación entre capas de diferentes lugares físicos
- ❑ Sin embargo, compartir el mismo hardware, puede impactar la performance, por ejemplo, al acceder a recursos compartidos

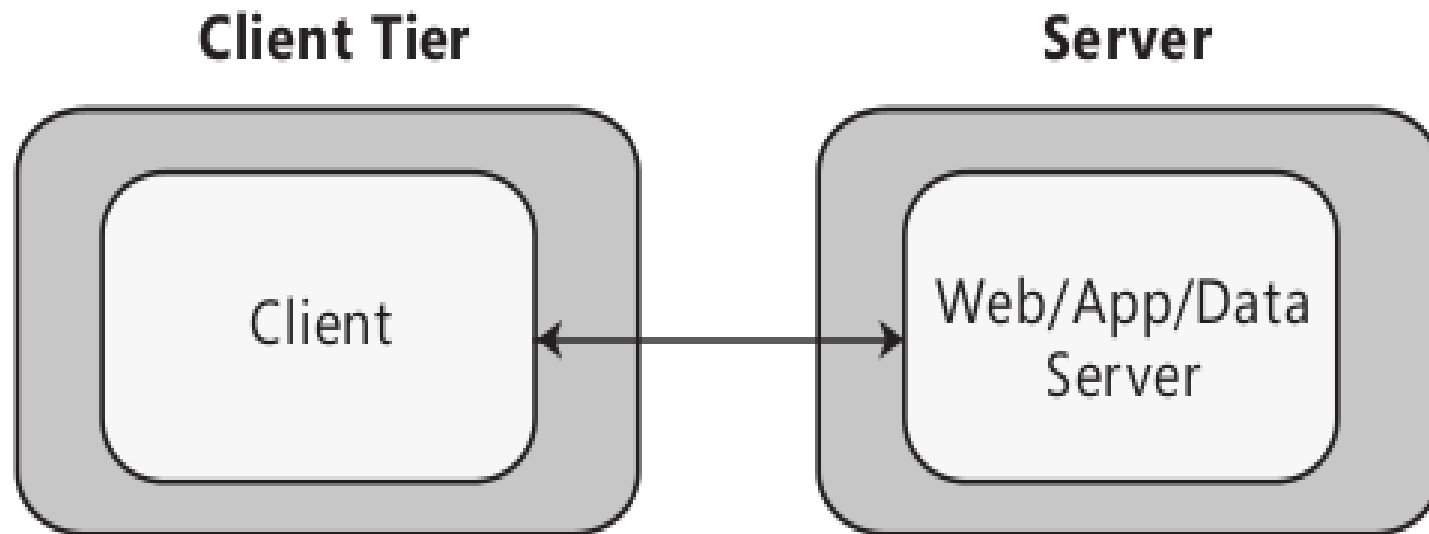
- ❑ Este enfoque permite configurar el hardware según las necesidades de cada capa
- ❑ Esto permite ajustar las necesidades de escalabilidad según cada capa de la aplicación
- ❑ Sin embargo, el uso de componentes distribuidos, impacta la performance a la hora de realizar llamadas remotas entre diferentes locaciones físicas

# Arquitectura de Software

## Patrones de Deployment Distribuido (N-Tier)

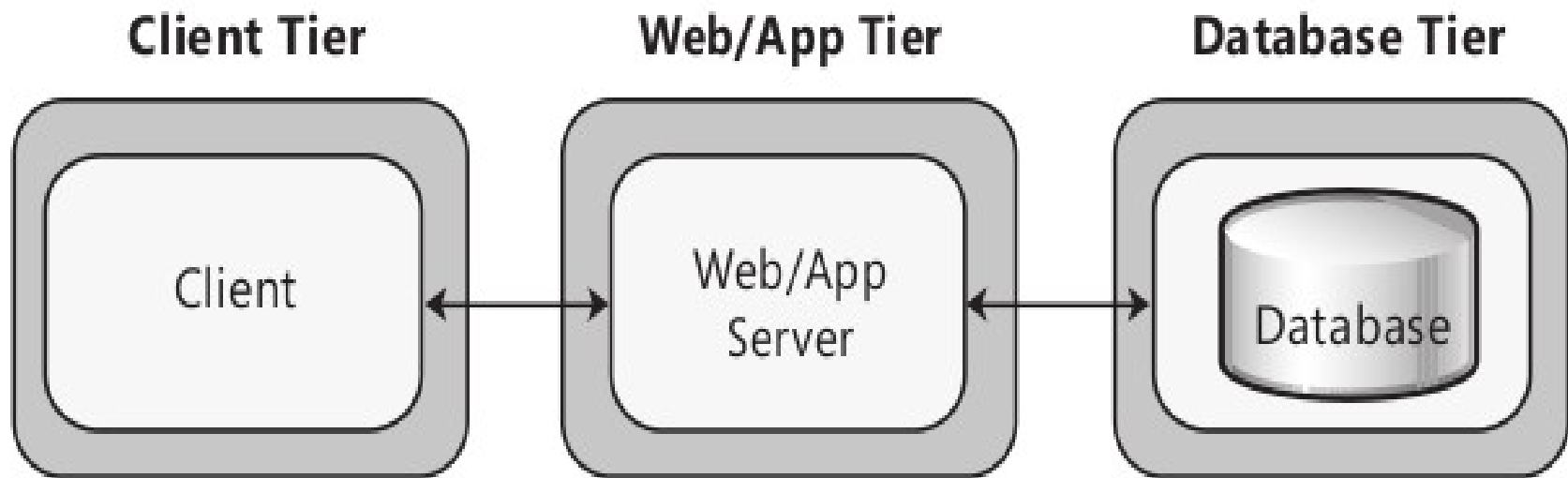
- ❑ Cliente / Servidor
- ❑ 2-Tier
- ❑ 3-Tier
- ❑ N-Tier

### ❑ Cliente / Servidor

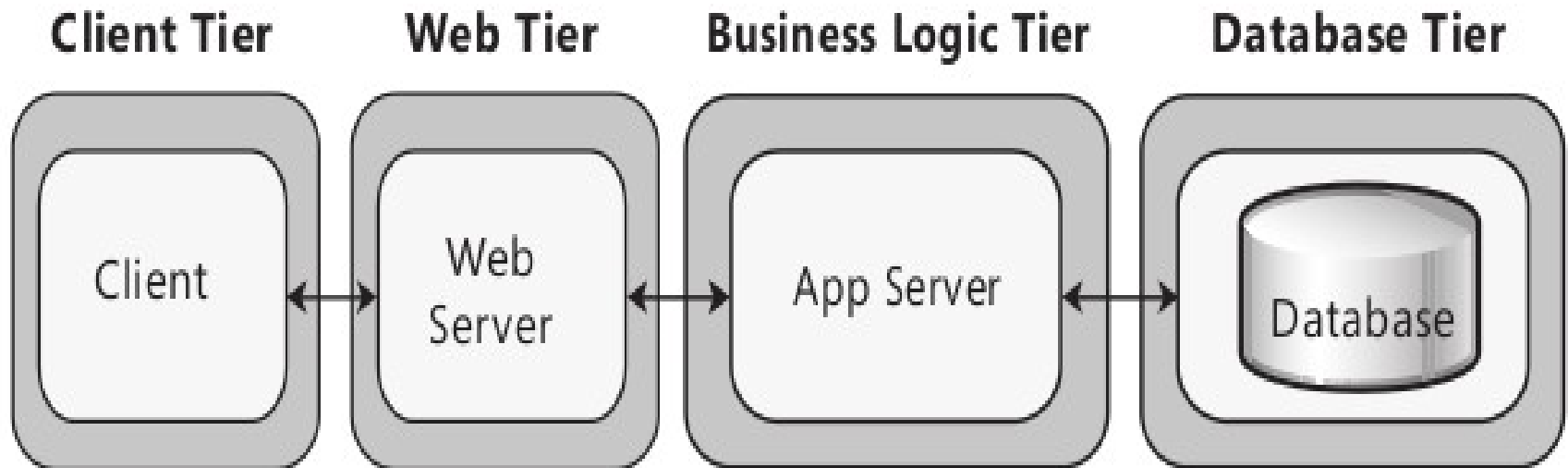




### □ 3-Tier



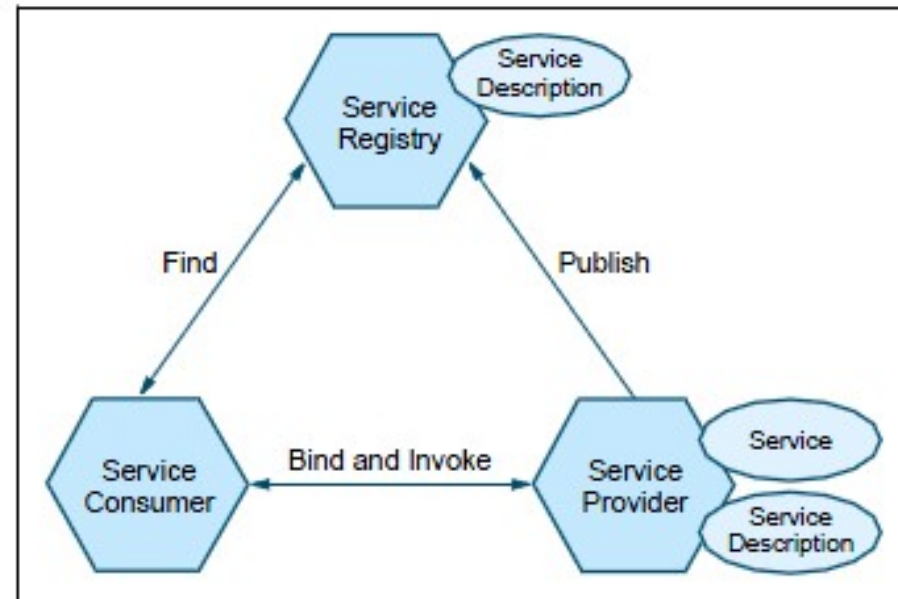
### □ 4-Tier



- Una Arquitectura Orientada a Servicios (Service Oriented Architecture, SOA) es una forma lógica de diseñar un sistema de software para proveer servicios, a aplicaciones u otros servicios distribuidos en la red, a través de interfaces que son publicadas y puede ser descubiertas

(Papazoglou and Heuvel 2007)

- Los tres roles principales en una SOA son:
  - Proveedor de Servicios
  - Registro de Servicios
  - Consumidor de Servicios



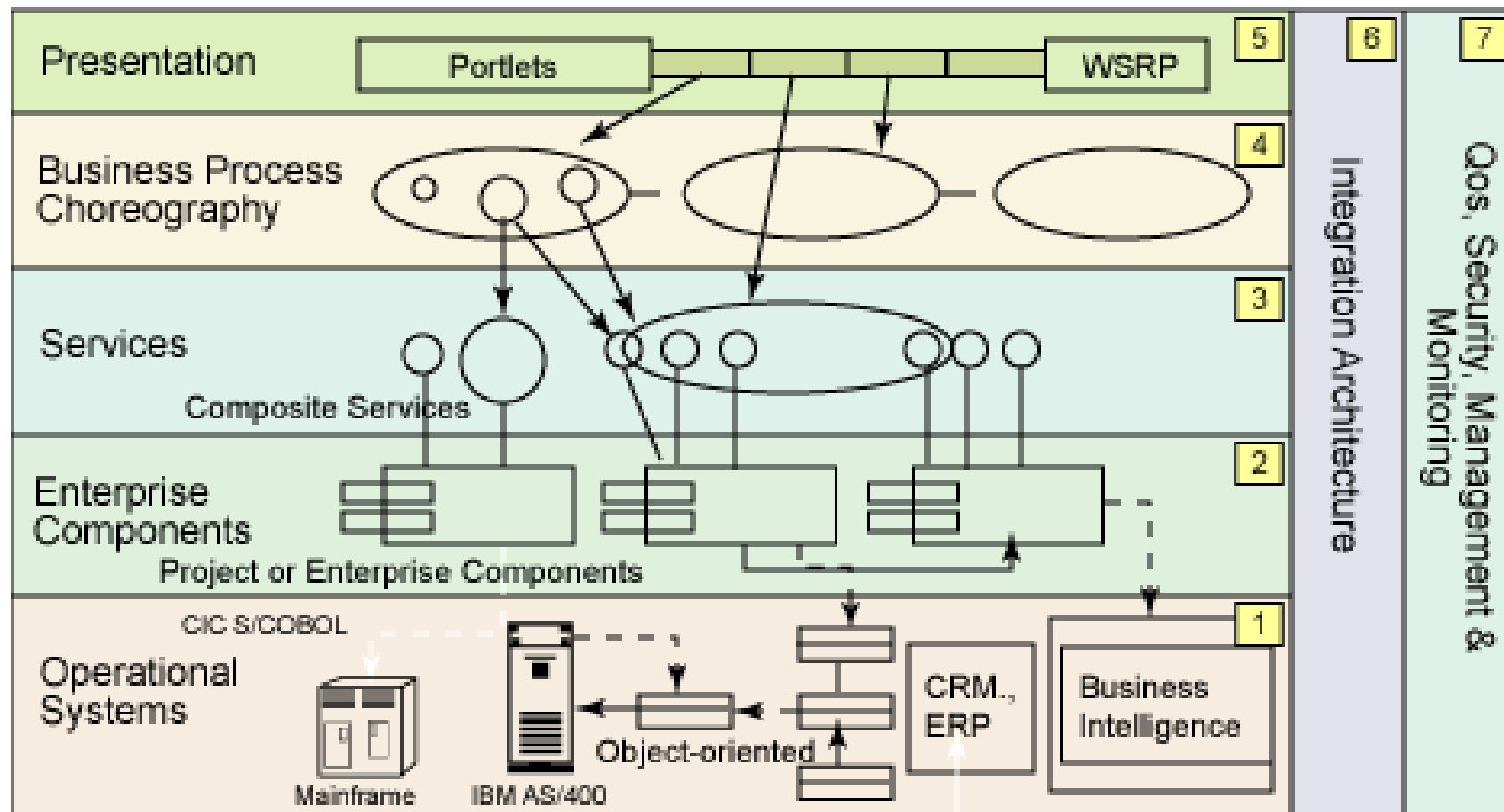
(Endrei et al. 2004)

- ❑ Una SOA facilita varias tareas del desarrollo de aplicaciones empresariales distribuidas:
  - la integración, la implementación de procesos de negocios y el aprovechamiento de sistemas legados
- ❑ Una SOA provee la flexibilidad y agilidad que requieren los usuarios de negocio:
  - les permite definir servicios de alta granularidad que pueden ser combinados y reutilizados para abordar necesidades de negocio actuales y futuras

(Papazoglou and Heuvel 2007)

# Arquitectura de Software

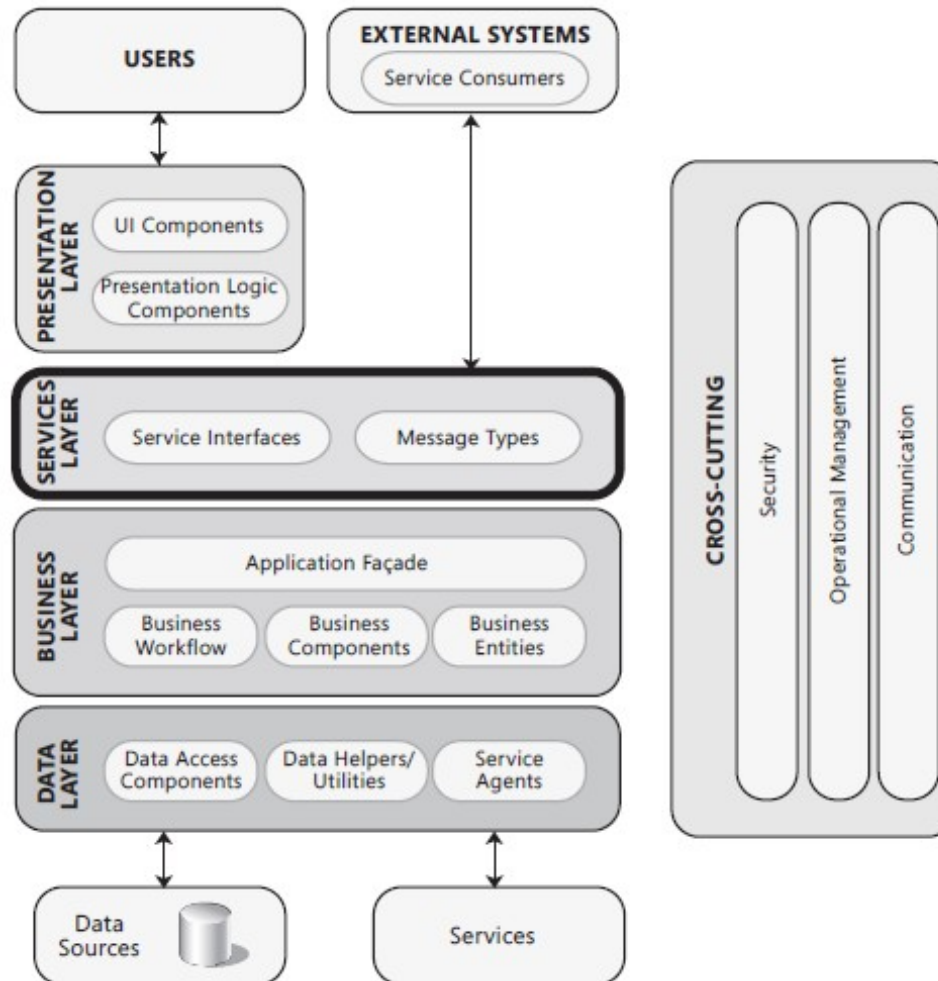
## Capas de una SOA



<http://www.ibm.com/developerworks/library/ws-soa-design1/>

# Arquitectura de Software

## SOA y Arquitectura en Capas



Microsoft Patterns & Practices. Microsoft Application Architecture Guide v2.0

- ❑ La “arquitectura” es un concepto abstracto por lo que se necesita un mecanismo de representación
- ❑ Un modelo es una simplificación de la realidad, creada con el objetivo de abstraer una porción del sistema, de forma de simplificar su comprensión

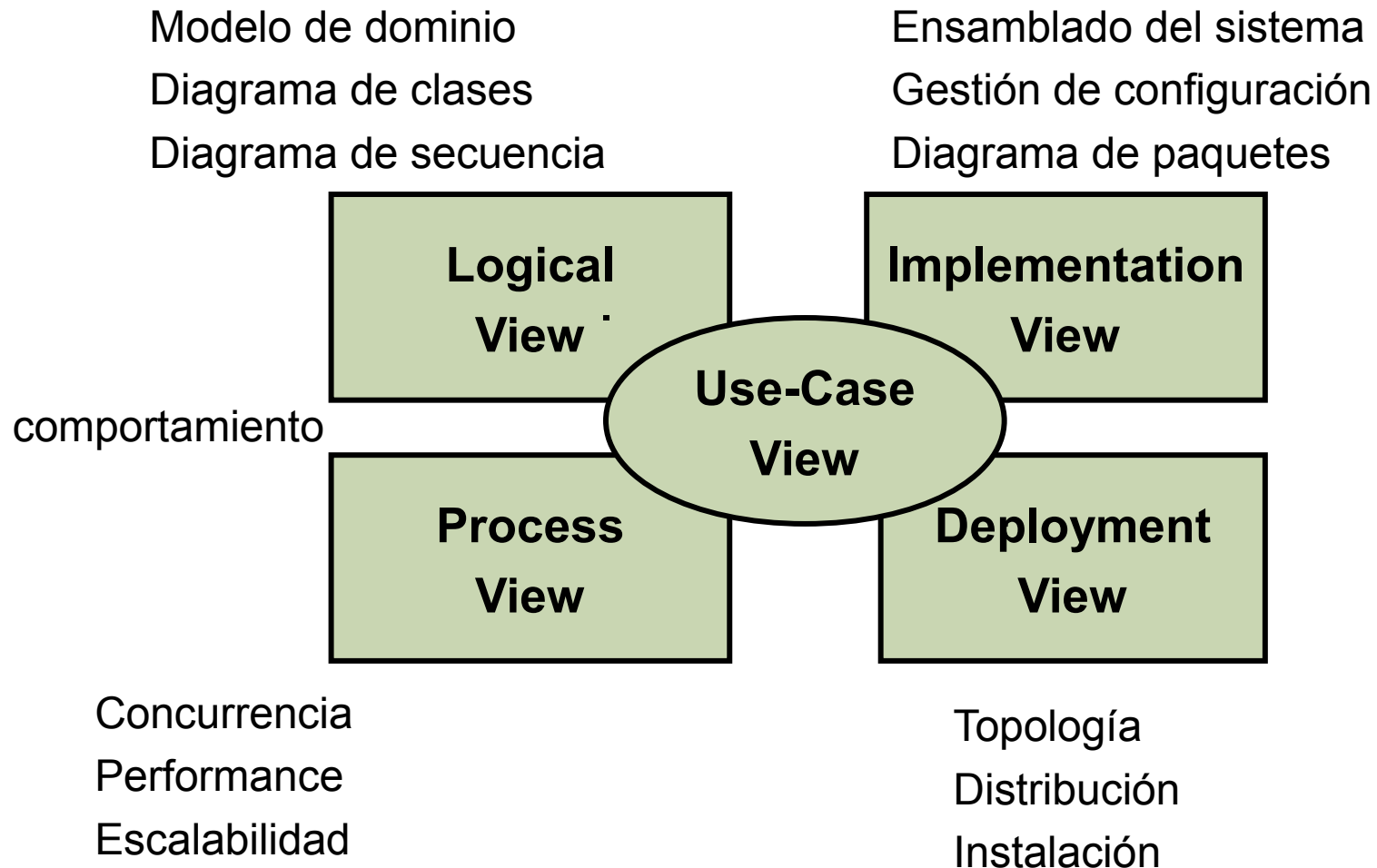


- Una vista es una representación de uno o más aspectos estructurales de una arquitectura que ilustra cómo la arquitectura lleva adelante uno o más *concerns* de uno o más *stakeholders*
  - stakeholders – persona, grupo o entidad con un interés sobre la realización de la arquitectura
  - concern (preocupación) de la arquitectura – es un requerimiento, objetivo o intención que pueda tener un *stakeholder* respecto a la arquitectura

- ❑ Modelo 4+1 vistas para la arquitectura de software
  - Propuesto por Phillippe Kruchten (1995)
  - Impulsa fuertemente la noción de vistas como modelo de representación de arquitecturas de software
- ❑ Sugiere
  - 4 vistas del sistema
  - 1 vista de casos de uso

# Arquitectura de Software

## Representación – 4 + 1

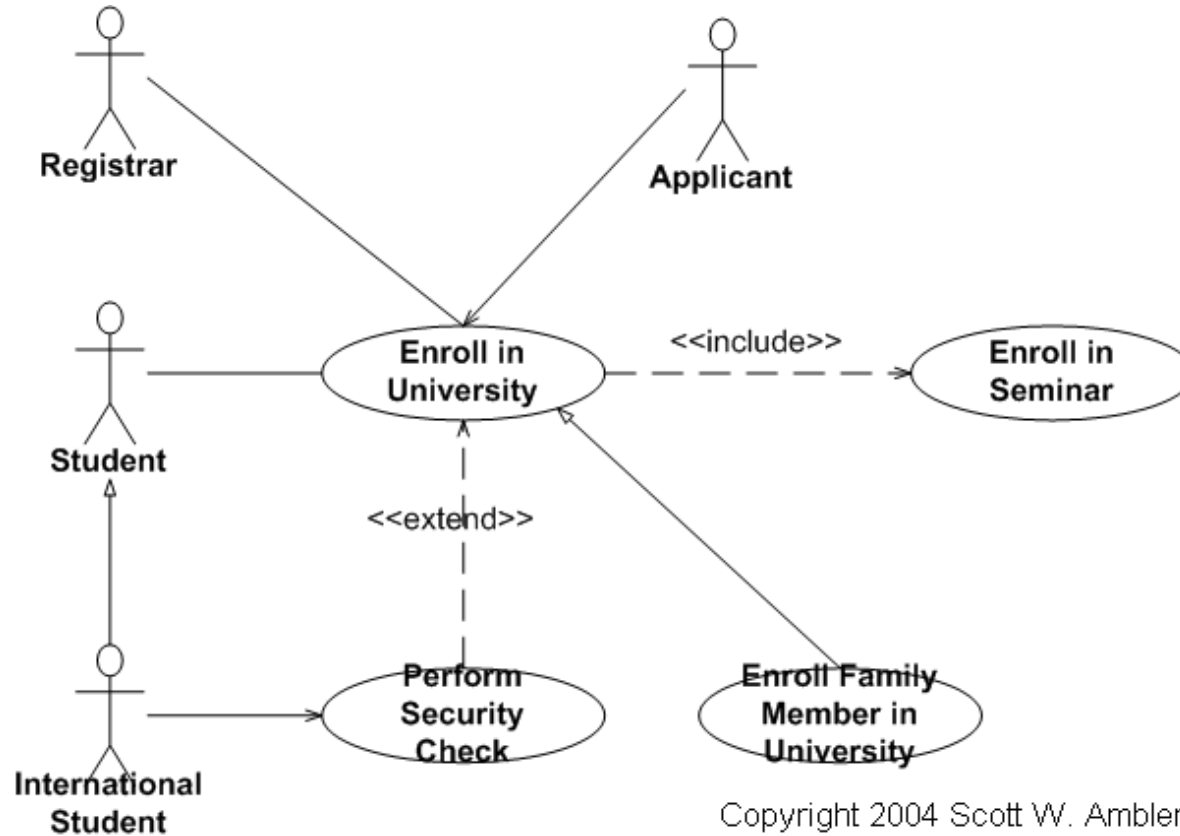


- ❑ Cada vista está más alineada al punto de vista de un *stakeholder*
  - Lógica: usuario final
  - Proceso: integrador, desarrollador
  - Implementación: desarrollador, *project manager*
  - *Deployment*: administrador del sistema

- ❑ La descripción de la funcionalidad provista por el sistema, desde el punto de vista de un actor externo
  - Selección de escenarios relevantes para la arquitectura
- ❑ Se usa para ilustrar las otras vistas de la arquitectura

# Arquitectura de Software

## 4 + 1: Vista de Casos de Uso

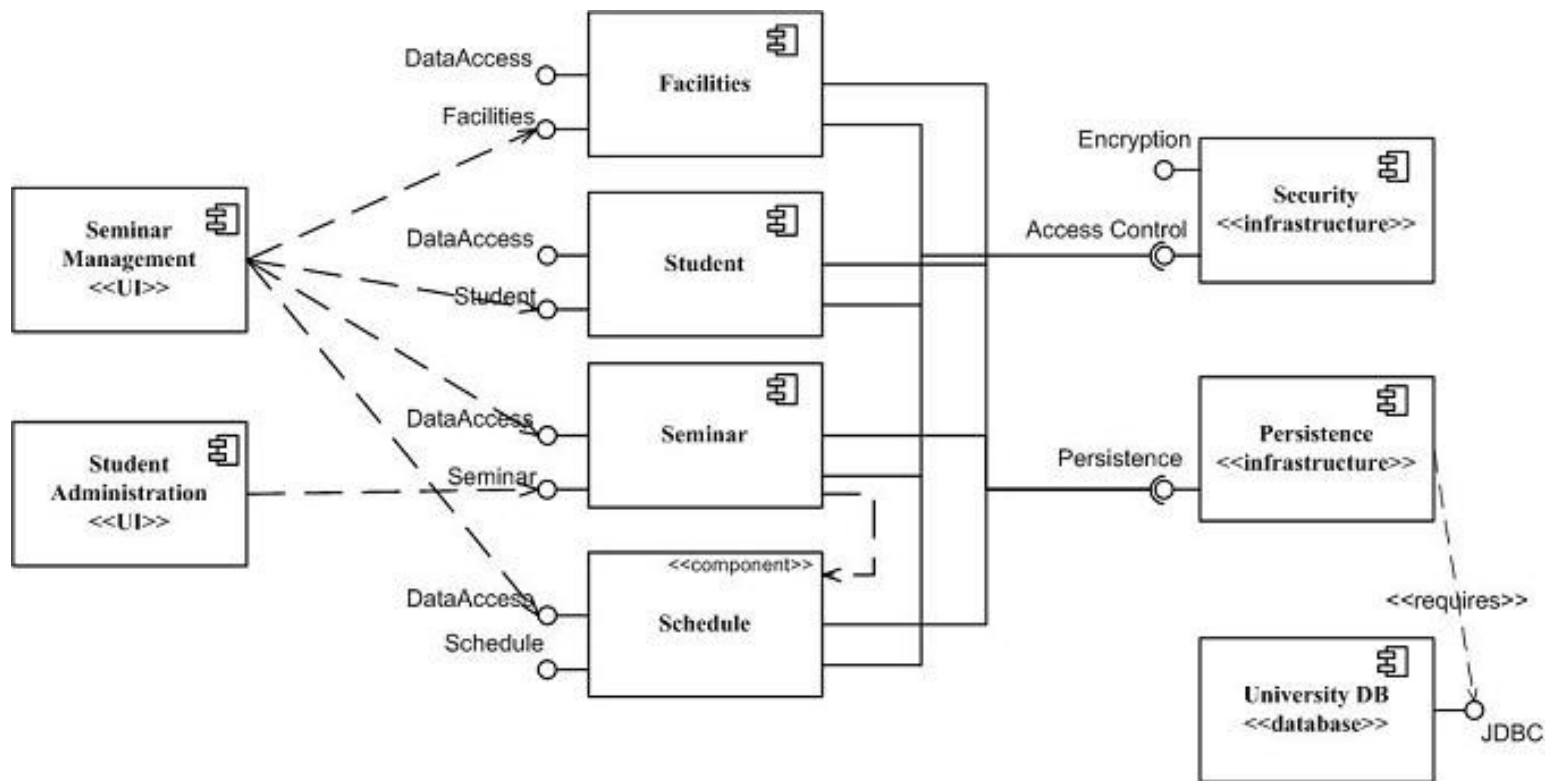


Copyright 2004 Scott W. Ambler

- ❑ Focalizada en los requerimientos funcionales fundamentalmente
- ❑ Presenta las abstracciones lógicas más importantes
  - ❑ potencialmente varios refinamientos
  - ❑ definición de las interfaces que ofrecen los componentes
- ❑ Generalmente se utilizan patrones o estilos arquitectónicos para organizar

# Arquitectura de Software

## 4 + 1: Vista lógica



<http://www.agilemodeling.com/artifacts/componentDiagram.htm>

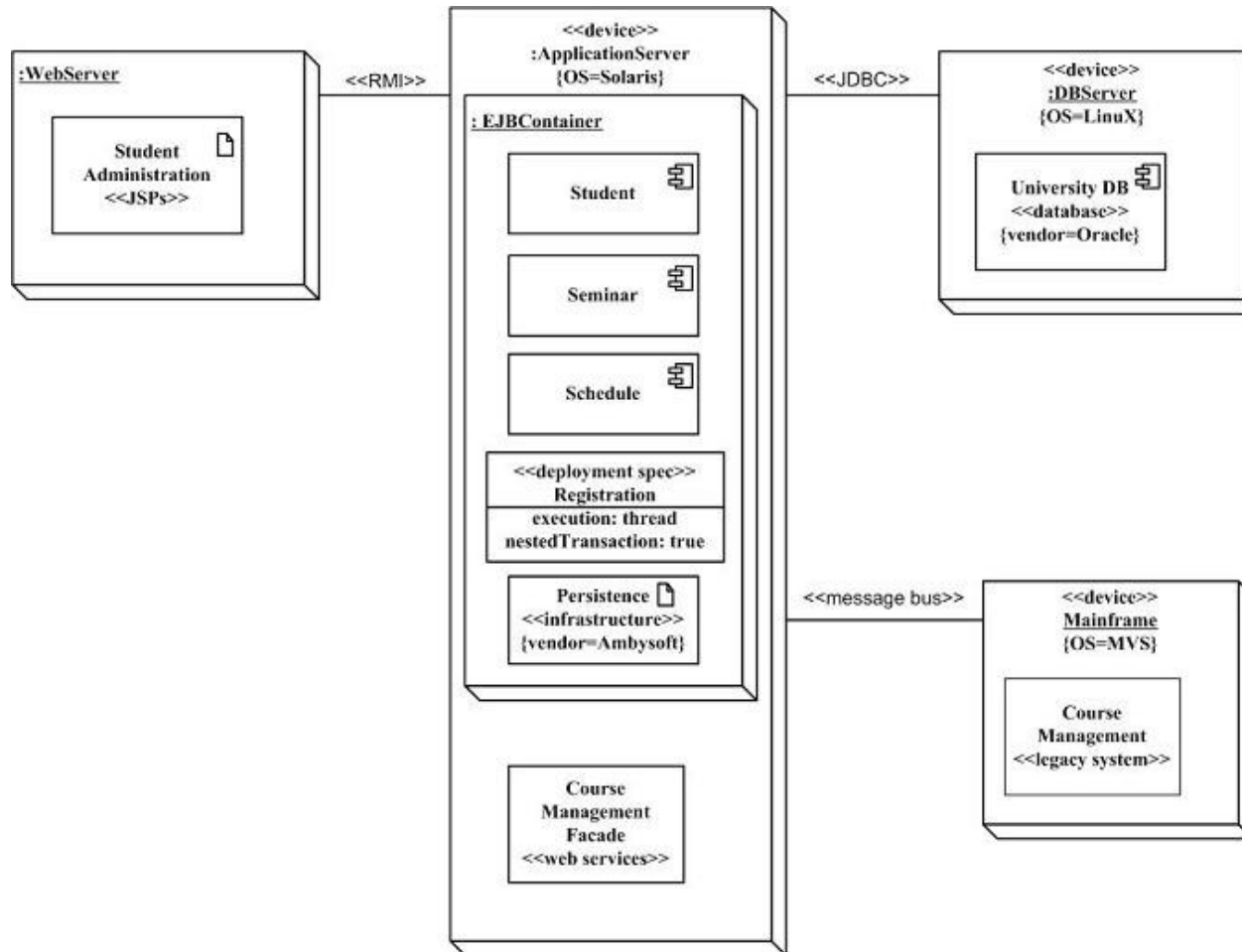


- ❑ Presenta los procesos o threads de ejecución presentes en el sistema
- ❑ Presenta la estrategia de sincronización y comunicación entre procesos
- ❑ Considera requerimientos no funcionales como performance, tolerancia a fallas, disponibilidad, escalabilidad, etc

- ❑ Describe la organización del sistema en el ambiente donde va a ser instalado
  - nodos que conforman la topología física del ambiente
  - hardware y/o software que soporta la ejecución en estos nodos
  - particularidades de configuración
  - mecanismos de comunicaciones entre dichos nodos

# Arquitectura de Software

## 4 + 1: Vista de Deployment

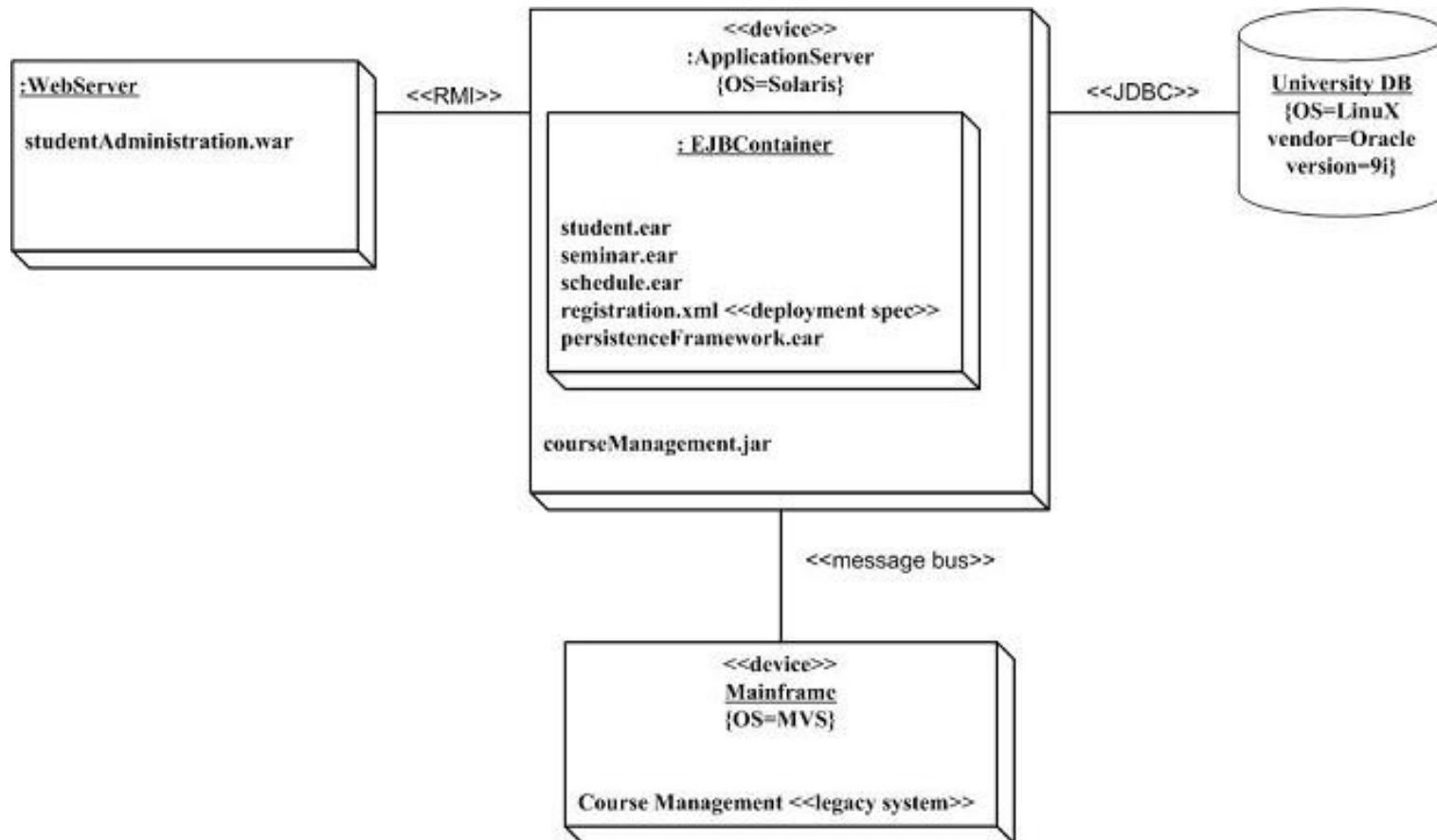


<http://www.agilemodeling.com/artifacts/deploymentDiagram.htm>

- ❑ Se focaliza en los módulos en que está organizado el ambiente de desarrollo
  - Incorpora restricciones particulares de la plataforma, lenguaje de programación o herramientas que se estén utilizando
- ❑ Presenta los componentes run-time que conforman el sistema

# Arquitectura de Software

## 4 + 1: Vista de Implementación



<http://codeidol.com/other/learnuml2/Modeling-Your-Deployed-System-Deployment-Diagrams/Deployed-Software-Artifacts/>

- ❑ 2003. Andrés Vignaga, Daniel Perovich. SAD del subsistema de reservas del sistema de gestión hotelera.
  - <http://www.fing.edu.uy/inco/pedeciba/bibliote/reptec/TR0314.pdf>

# Multi-Tenancy

## Definición y Beneficios

- ❑ Es la habilidad de proveer una aplicación de software a múltiples organizaciones cliente (o tenants) a través de una única instancia compartida
- ❑ Uno de los beneficios de multi-tenancy es en relación a los costos
  - Se comparte software, hardware, etc...
- ❑ Otros beneficios incluyen la posibilidad de actualizar simultáneamente a todos la aplicación para todos tenants

# Multi-Tenancy

## Desafíos

- ☐ Aislamiento
- ☐ Seguridad
- ☐ Customización
- ☐ Actualizaciones
- ☐ Recuperación

<http://www.ibm.com/developerworks/library/ws-middleware/>

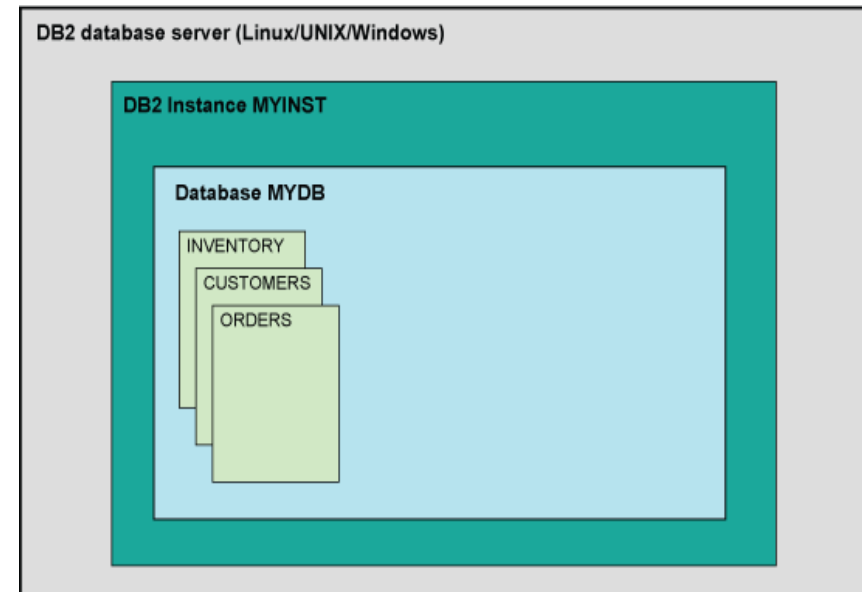


# Multi-Tenancy

## Compartir Tablas (Single Schema, Single DB)

### ❑ Compartir Tablas

- Compartir tablas entre todos los clientes
- Ventajas:
  - Bajo costo, menos cantidad de almacenamiento, etc.
- Desventajas:
  - Complejidad de las consultas, si una tabla se corrompe afecta a todos, etc.



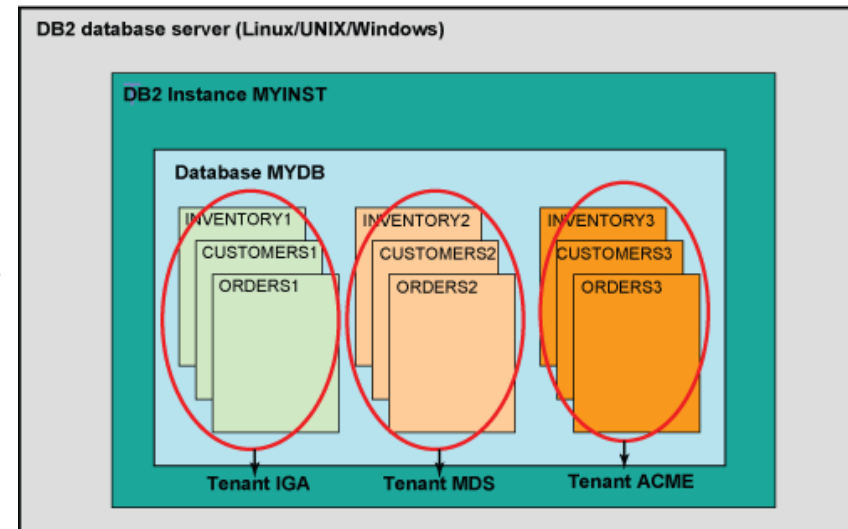
<http://www.ibm.com/developerworks/data/library/techarticle/dm-1201dbdesigncloud/>

# Multi-Tenancy

## Single Schema, Single DB op II

### ❑ Compartir Base de Datos

- Tablas diferentes para cada cliente
- Ventajas:
  - Bajo costo, mejora aislamiento y personalización con respecto al anterior, etc.
- Desventajas:
  - Incremento de almacenamiento (más tablas), consultas complejas (nombres de tablas diferentes para cada cliente), etc.



<http://www.ibm.com/developerworks/data/library/techarticle/dm-1201dbdesigncloud/>

# Multi-Tenancy

## Multiple Schemas, Shared DB

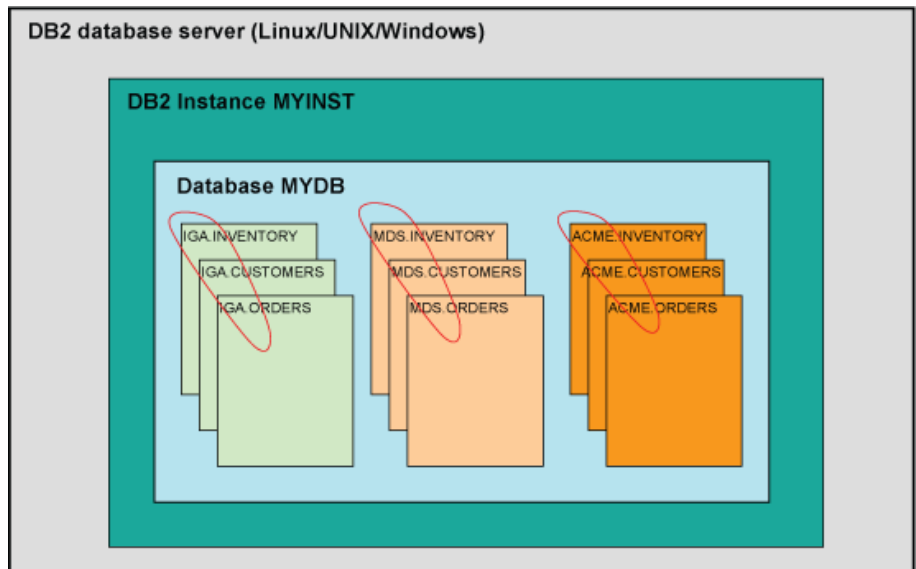
### ❑ Compartir Base de Datos usando distintos esquemas para cada cliente

#### ○ Ventajas:

- Mejora aislamiento, las consultas son las mismas (cambia el esquema), etc.

#### ○ Desventajas:

- Requiere más almacenamiento



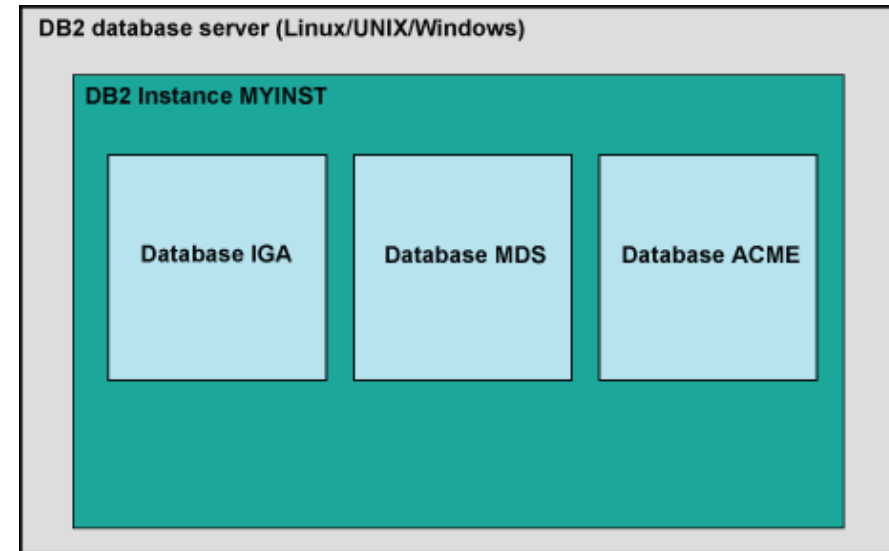
<http://www.ibm.com/developerworks/data/library/techarticle/dm-1201dbdesigncloud/>

# Multi-Tenancy

## Multiple Databases

### ❑ Compartir Instancia del Manejador de BD

- Cada cliente tiene su base de datos
- Ventajas:
  - Mejor aislamiento, mantenimiento independiente de BD, menos complejidad de la aplicación..
- Desventajas:
  - Más almacenamiento, puede limitaciones de cant de BD, consumo de memoria..



<http://www.ibm.com/developerworks/data/library/techarticle/dm-1201dbdesigncloud/>

# Multi-Tenancy

## Multiple Servers

### ❑ Compartir Servidor

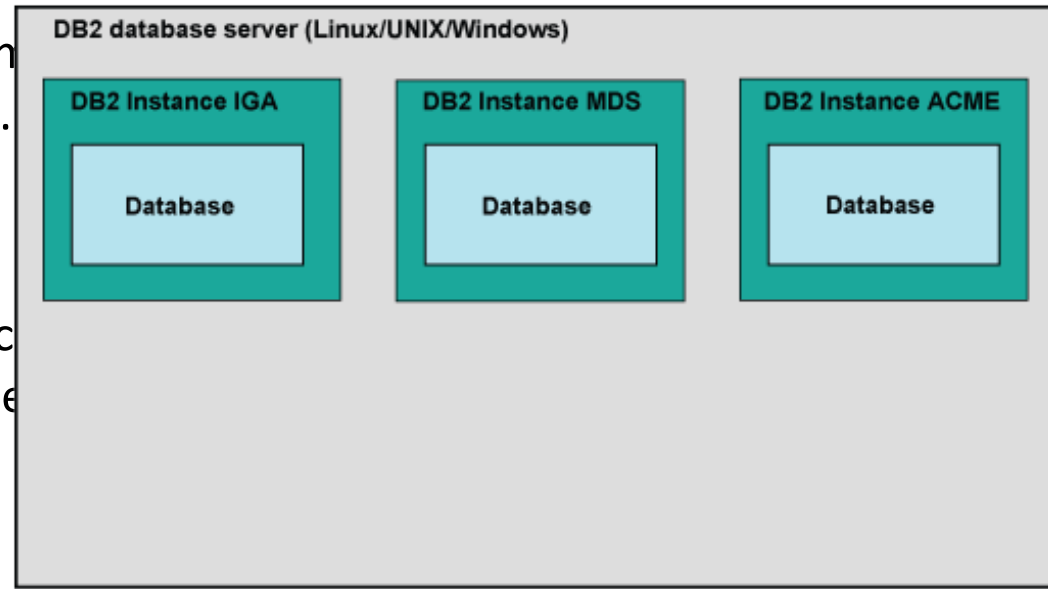
- Cada cliente tiene su propia instancia de manejador

#### ➤ Ventajas:

- Buen aislamiento, mantenimiento independiente, etc.

#### ➤ Desventajas:

- Mayor almacenamiento, puede implicar mas costos de licencias, puede haber limitación en cantidad de instancias



<http://www.ibm.com/developerworks/data/library/techarticle/dm-1201dbdesigncloud/>

- ❑ Microsoft Patterns & Practices. Microsoft Application Architecture Guide v2.0
- ❑ Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.
- ❑ G. Hohpe and B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional, October 2003.

- ❑ Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. N. Rozanski, E. Woods. Addison-Wesley, 2005
- ❑ Software Architecture in practice, Second Edition. L. Bass, P. Clemens, R. Kazman. Addison-Wesley, 2003
- ❑ Architectural Blueprints — The “4+1” View Model of Software Architecture. Kruchten, Philippe. 1995.  
<http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>

- ❑ M. Papazoglou, *Web Services: Principles and Technology*, 1st ed. Prentice Hall, 2007.
- ❑ Web Services Concepts, Architectures and Applications. Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju. Springer 2004  
<http://www.inf.ethz.ch/personal/alonso/WebServicesBook>
- ❑ Understanding Web Services Specifications. IBM developersWorks.  
[http://www.ibm.com/developerworks/views/webservices/libraryview.jsp?search\\_by=Understanding+Web+%20Services+specifications+Part](http://www.ibm.com/developerworks/views/webservices/libraryview.jsp?search_by=Understanding+Web+%20Services+specifications+Part)



- ❑ Designing a database for multi-tenancy on the cloud  
<http://www.ibm.com/developerworks/data/library/techarticle/dm-1201dbdesigncloud/>
- ❑ Multi-Tenant Data Architecture <http://msdn.microsoft.com/en-us/library/aa479086.aspx>
- ❑ Develop and Deploy Multi-Tenant Web-delivered Solutions using IBM middleware: Part 1: Challenges and architectural patterns  
<http://www.ibm.com/developerworks/library/ws-middleware/>

# PREGUNTAS

---

