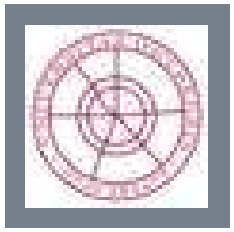


Arquitectura Software Dirigida por Modelos.

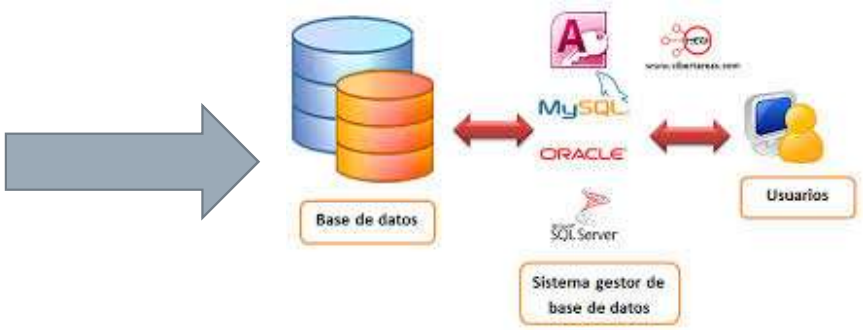
TEMA 1: ARQUITECTURA SOFTWARE

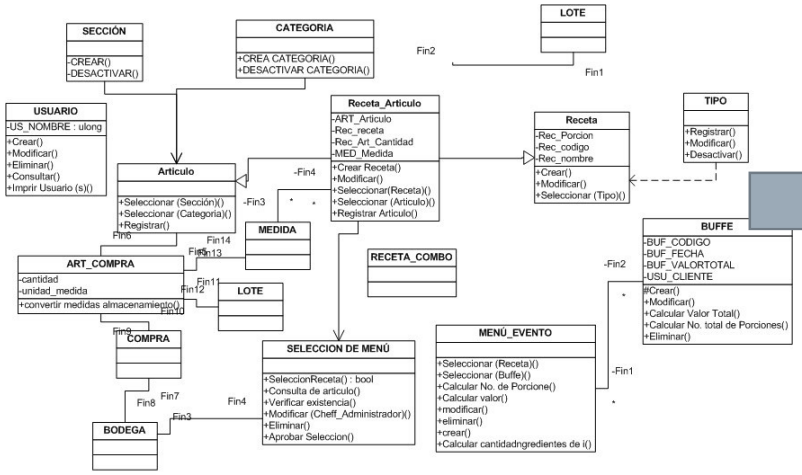
- 1.1. Introducción
- 1.2. Estilos arquitectónicos.
- 1.3. Patrones Arquitectónicos
  - 1.3.1 .-Arquitectura Software en Subsistemas
  - 1.3.2.-Arquitectura Software Orientada a Objetos
  - 1.3.3-Arquitectura Software Cliente/Servidor
  - 1.3.4.- Arquitectura Software Orientada a Servicios
  - 1.3.5.- Arquitectura Software Orientada a Componentes

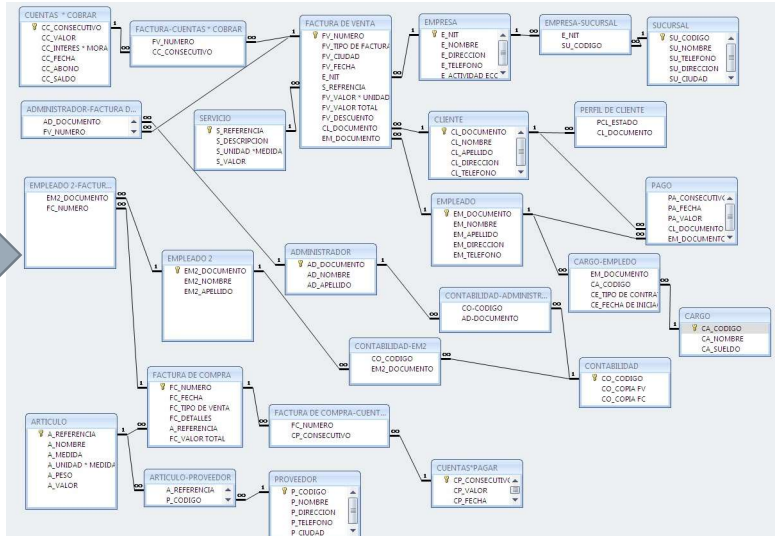
Del modelo estático al modelo de datos relacional.

Del modelo estático al modelo de datos relacional.

› Datos clase entidad







Del modelo estático al modelo de datos relacional.

- La mayoría de las bases de datos son bases de datos relacionales, por lo tanto el objetivo es llevar a cabo el diseño lógico de la base de datos relacional del modelo estático conceptual, particularmente para aquellas clases de entidad que necesitan ser persistentes.
  - Clase entidad. Son clases de datos conceptuales - es decir, su propósito principal es almacenar datos y facilitar el acceso a estos datos.

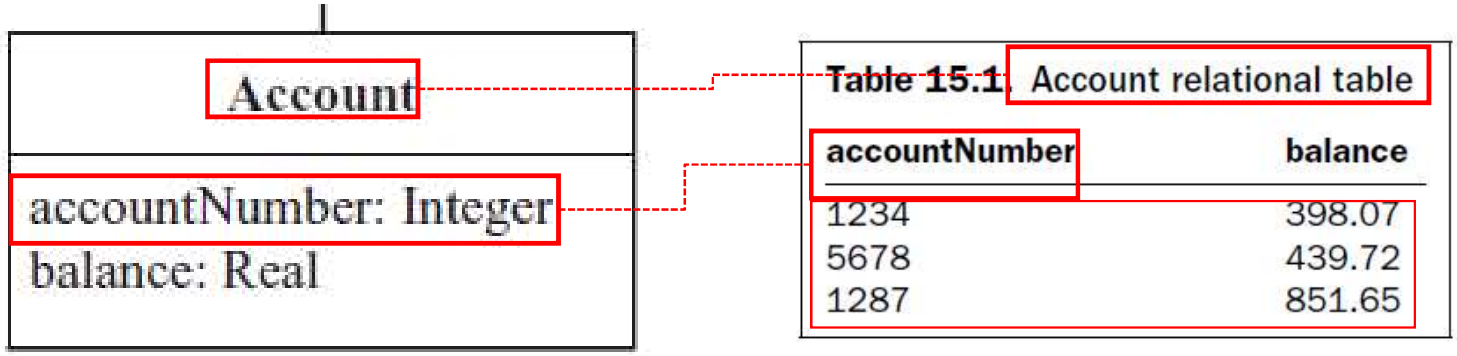
Del modelo estático al modelo de datos relacional.

- Pasos para el diseño relacional de la base de datos:
  1. Diseño de las tablas relacionales y las claves principales.
  2. Diseño de las claves externas para representar las asociaciones.
  3. Diseño de tablas de asociación para representar clases asociación.
  4. Diseño de las relaciones todo/parte (agregación).
  5. Diseño de las relaciones de generalización/especialización.

Del modelo estático al modelo de datos relacional.

1.- Diseño de las tablas relacionales y las claves principales

› En el caso más simple, una clase entidad se diseña como una tabla relacional, el nombre de la clase entidad se le asigna a la tabla. Cada atributo de la clase entidad se asigna a una columna de la tabla. Cada instancia de objeto se asigna a una fila de la tabla.



Del modelo estático al modelo de datos relacional.

- La identificación de las claves principales . Cada tabla relacional en una base de datos relacional debe tener una clave primaria. En su forma más simple, la clave principal es un atributo que se utiliza para localizar de forma exclusiva una fila en una tabla.
  - Account(accountNumber, balance)
  - Primary key: accountNumber
- Si fuera necesario se concatenarían varios atributos

Del modelo estático al modelo de datos relacional.

2.- Diseño de las claves externas para representar las asociaciones.

› Las asociaciones de uno a uno y uno a muchos en las bases de datos relacionales se pueden representar por una clave externa, (foreign key) dicha clave externa permitirá la navegación entre tablas.

Del modelo estático al modelo de datos relacional.

› Ejemplo:

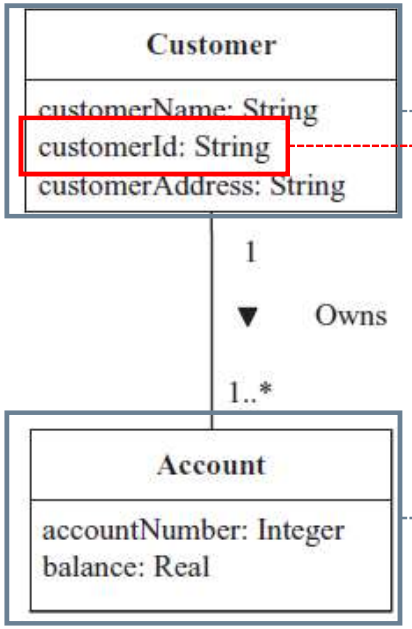


Table 15.2. Navigation between relational tables

Navigation from *customerId* (foreign key) in Account table . . .

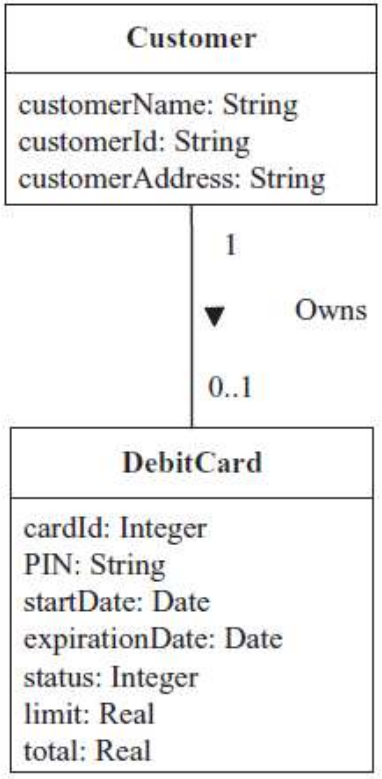
accountNumber	Balance	*customerId*
1234	398.07	24193
5678	439.72	26537
1287	851.65	21849

. . . to customerId (primary key) in Customer Table

customerName	customerId	customerAddress
Smith	21849	New York
Patel	26537	Chicago
Chang	24193	Washington

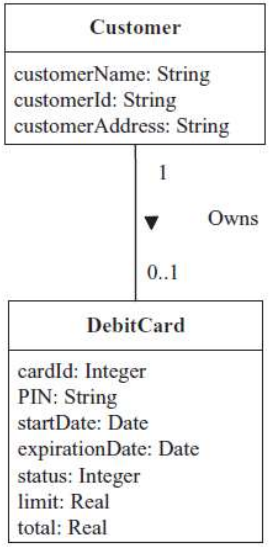
Del modelo estático al modelo de datos relacional.

- En el caso de una asociación de cero a uno, la colocación de la clave externa en la tabla relacional debe ser "opcional" para evitar referencias nulas, que se considera no deseable en los diseños de las bases de datos.



Del modelo estático al modelo de datos relacional.

› Ejemplo: Consideremos la relación el cliente es dueño de una tarjeta de débito.



Customer(customerName, customerId, customerAddress)

Debicard (CardID, PIN , starDate, expirationDate , status, limit, total, *CustomerID*)

Del modelo estático al modelo de datos relacional.

- Ejemplo: Consideremos la relación el cliente es dueño de una tarjeta de débito.

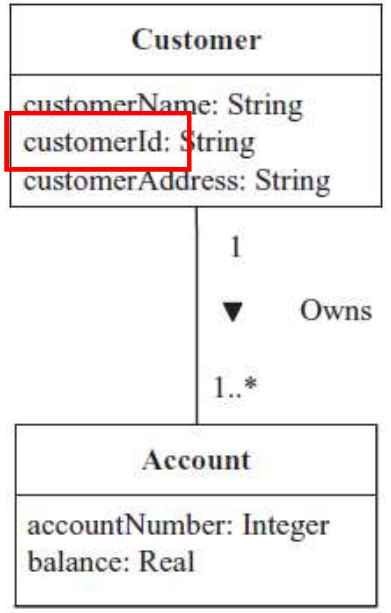
Customer(customerName, customerId, customerAddress)

Debicard (CardID, PIN , starDate, expirationDate , status, limit, total)

- Debido a que es posible que un cliente no tenga tarjeta de débito (relación opcional) haciendo CardID una clave externa en la tabla Cliente daría lugar a algunos clientes que tienen un valor nulo para el id de tarjeta.
- Por otro lado, debido a que cada tarjeta de débito siempre es propiedad de un cliente (relación obligatoria), por lo que customerId como clave externa en la tabla de Tarjeta de Débito es mejor solución, ya que nunca tendría un valor nulo.

Del modelo estático al modelo de datos relacional.

- En el caso de asociaciones de uno a muchos. Una asociación uno-a-muchos se diseña como una estructura relacional de tal forma que la clave externa se encuentra en la tabla cuya multiplicidad sea "muchos".
- Ejemplo:
  - Customer(customerName, customerId, customerAddress)
  - Account(accountNumber, balance, customerId)



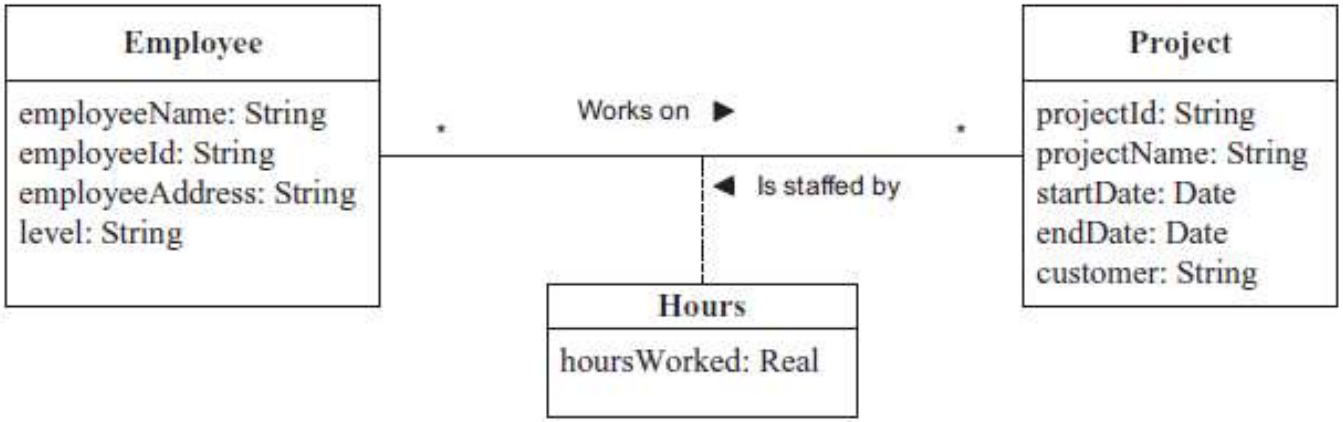
Del modelo estático al modelo de datos relacional.

3.- Diseño de tablas de asociación para representar clases asociación.

- Una clase asociación se suele usar para representar una asociación muchos-a-muchos.
- Una clase de asociación se asigna a una tabla de asociación. Una tabla de asociación representa la asociación entre dos o más relaciones. La clave principal de una tabla asociación es la clave concatenada formada por la clave primaria de cada tabla que participa en la asociación.

Del modelo estático al modelo de datos relacional.

- Ejemplo:



- Project(projectId, projectName, startDate, enDate, customer)
- Employee(employeeName, employeeId, employeeAddress, Level)
- Hours(projectId, employeeId, hoursWorked)

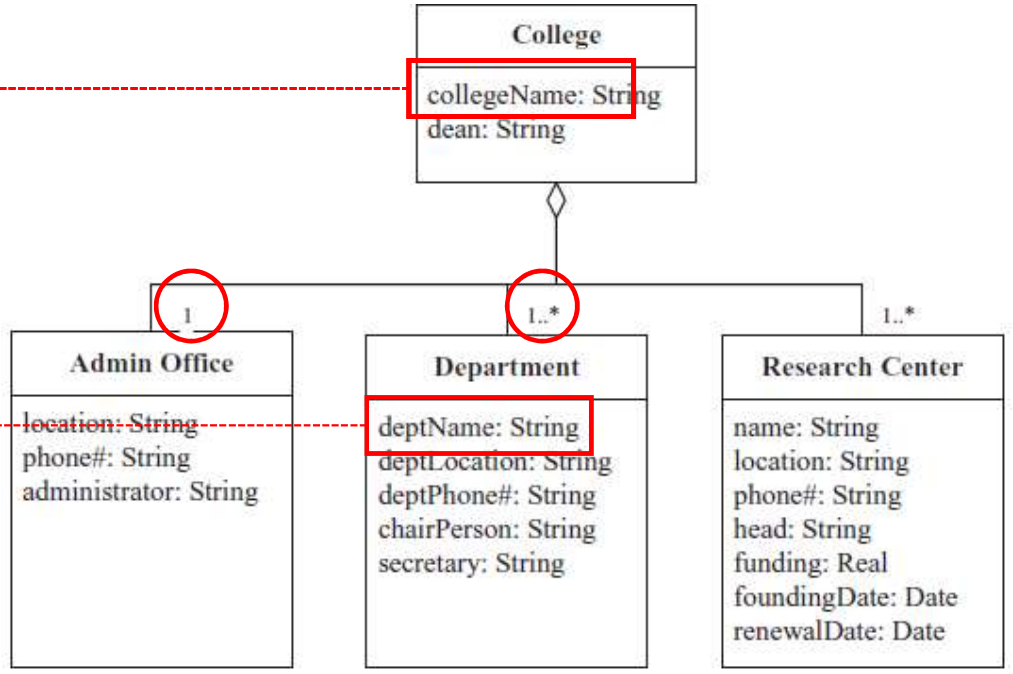
Del modelo estático al modelo de datos relacional.

4.- Diseño de las relaciones todo/parte (agregación).

- Una relación todo/parte o agregación o composición: Al asignar una relación todo/parte a una base de datos relacional , la clase agregada o compuesta (el todo) se diseña como una tabla relacional y cada clase parte también se diseña como una tabla relacional.
- La clave primaria del conjunto de tablas relacionales (compuesto o agregado) se hace de las siguientes formas:
  - Si la asociación es uno-a-uno entre clase agregada y la clase de parte, se usa como clave primaria la clave primaria de la tabla agregada o compuesta.
  - Si la asociación es de uno a muchos, se usa como clave primaria la clave de la tabla agregada y la primaria de la tabla parte

Del modelo estático al modelo de datos relacional.

› Ejemplo:



› College(collegeName, dean)

› AdminOffice(collegeName, location, ….)

› Department(departmentName, collegeName, location, ….)

Del modelo estático al modelo de datos relacional.

5.- Diseño de las relaciones de generalización/especialización.

Hay tres maneras diferentes de la asignación de un jerarquía de generalización/especialización a una base de datos relacional:

- La superclase y subclases están asignadas cada una a una tabla relacional.
- Sólo las subclases se asigna a tablas relacionales.
- Sólo la superclase se asigna a una tabla relacional.

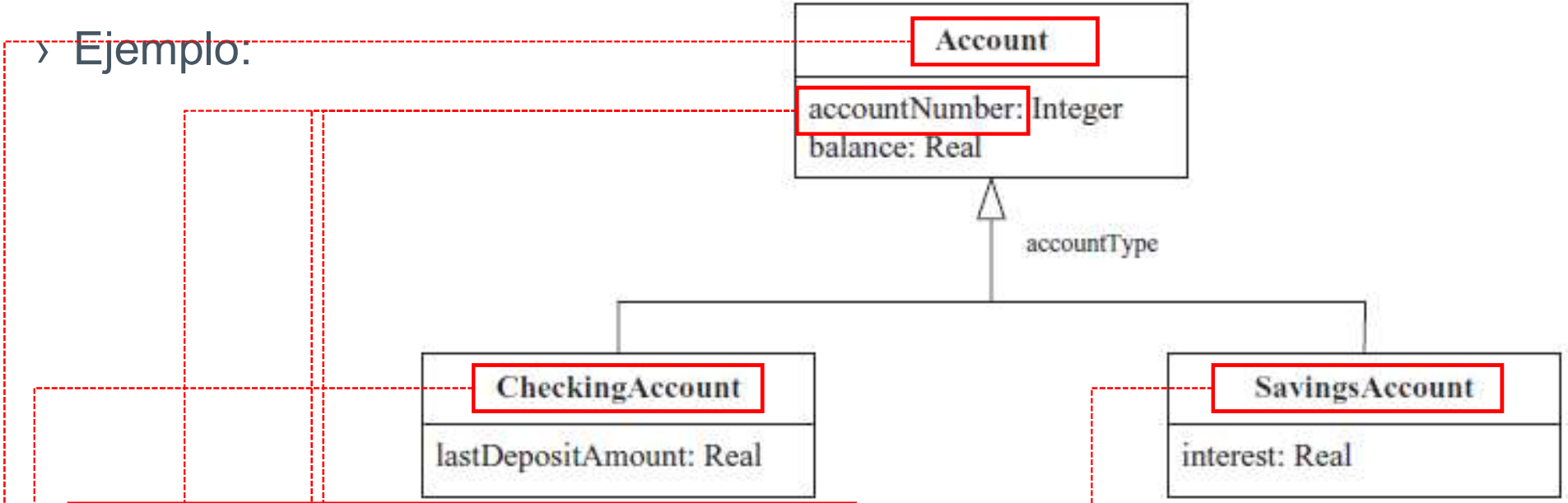
Del modelo estático al modelo de datos relacional.

- La superclase y subclases están asignadas cada una a una tabla relacional.
- La superclase se asigna a una tabla relacional. Cada subclase también se asigna a una tabla relacional. Hay un atributo compartido para la clave principal, en otras palabras, La misma clave principal se utiliza en la superclase y las tablas de subclases.

  La principal ventaja de este enfoque es que es limpio y extensible, ya que cada clase se asigna a una tabla. Sin embargo, la principal desventaja es que la navegación superclase/subclase podría ser lento ya que se accede a la tabla de superclase y a la subclase cada vez que se quiera acceder a alguna de ellas.

Del modelo estático al modelo de datos relacional.

- Ejemplo:



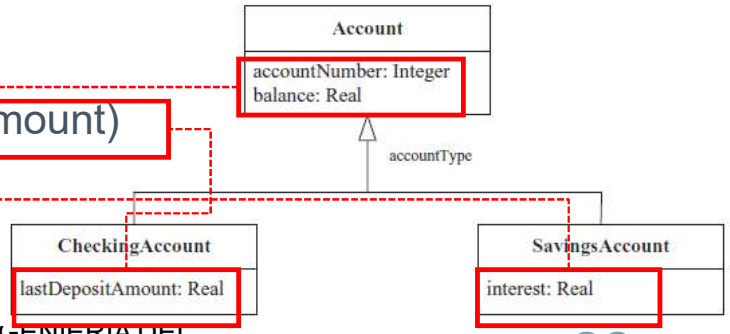
- Account (accountNumber, accountType, balance)
- Checking Account (accountNumber, lastDepositAmount)
- Savings Account (accountNumber, interest)

Del modelo estático al modelo de datos relacional.

Sólo las subclases se asigna a tablas relacionales.

- Con este enfoque, cada subclase está diseñada como una tabla. Sin embargo, no existe una tabla de superclase. En su lugar, los atributos de la superclase se repiten en cada tabla de la subclase.
- Este enfoque funciona especialmente bien si la subclase tiene muchos atributos y la superclase tiene pocos atributos.
- Ejemplo:

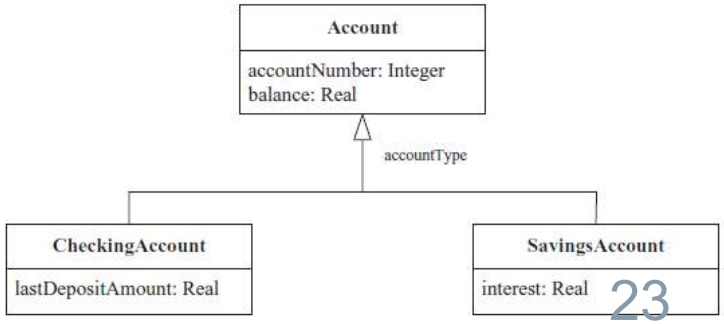
- CheckingAccount (accountNumber, balance, lastDepositAmount)
- SavingsAccount (accountNumber, balance, interest)



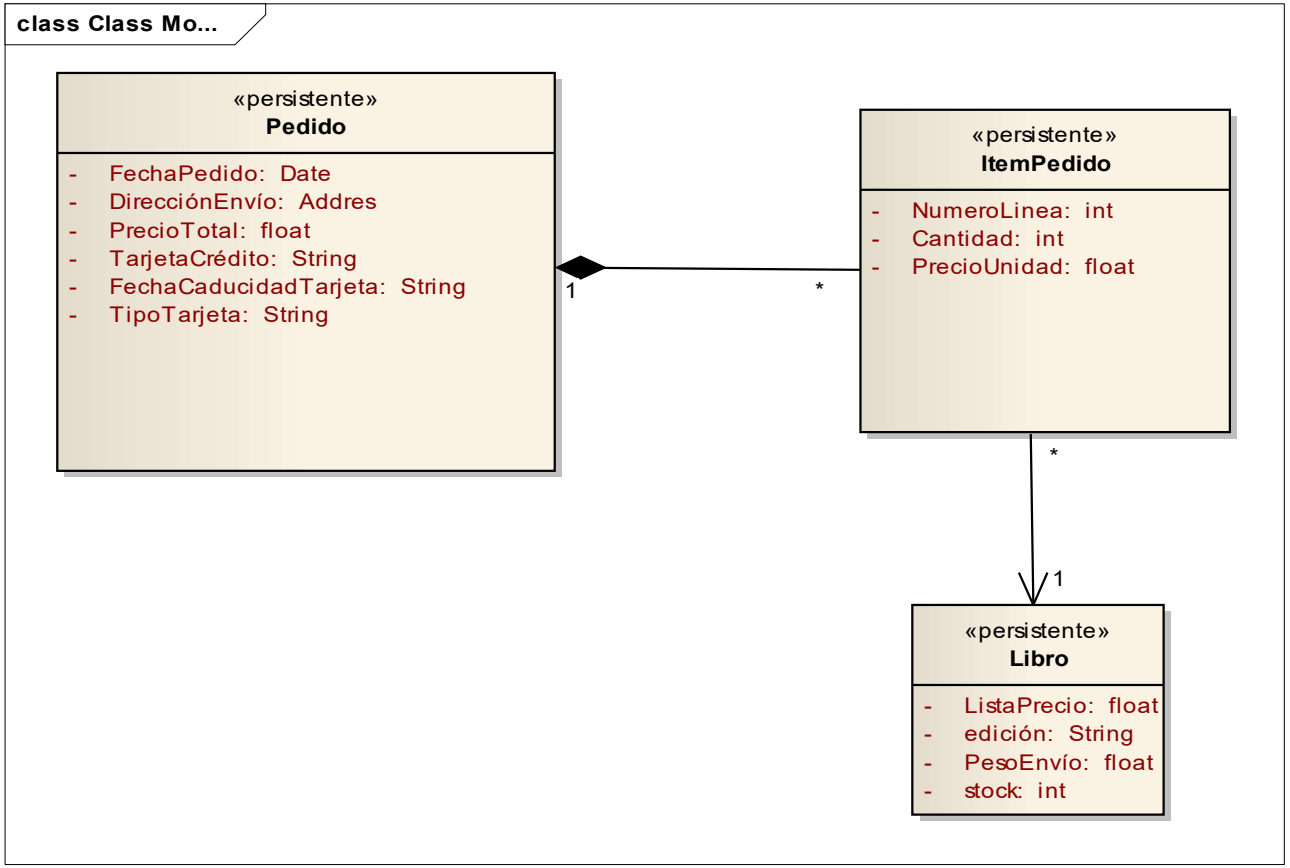
Del modelo estático al modelo de datos relacional.

Sólo la superclase se asigna a una tabla relacional.

- En este enfoque, hay una tabla de superclase pero no hay tablas de subclases. En su lugar, todos los atributos de la subclase se traen a la superclase. Es necesario añadir un discriminador de tipo (Ej:AccountType) que se añade como un atributo de la tabla de superclase. Cada fila de la tabla de superclase utiliza los atributos correspondientes a la subclase , el resto de valores de los atributos están a null. Este enfoque se puede utilizar si la superclase tiene muchos atributos, cada subclase tiene sólo unos pocos atributos, y hay un pequeño número de subclases.
- Ejemplo: Account (accountNumber, accountType, balance, lastDepositAmount, interest)



Del modelo estático al modelo de datos relacional.

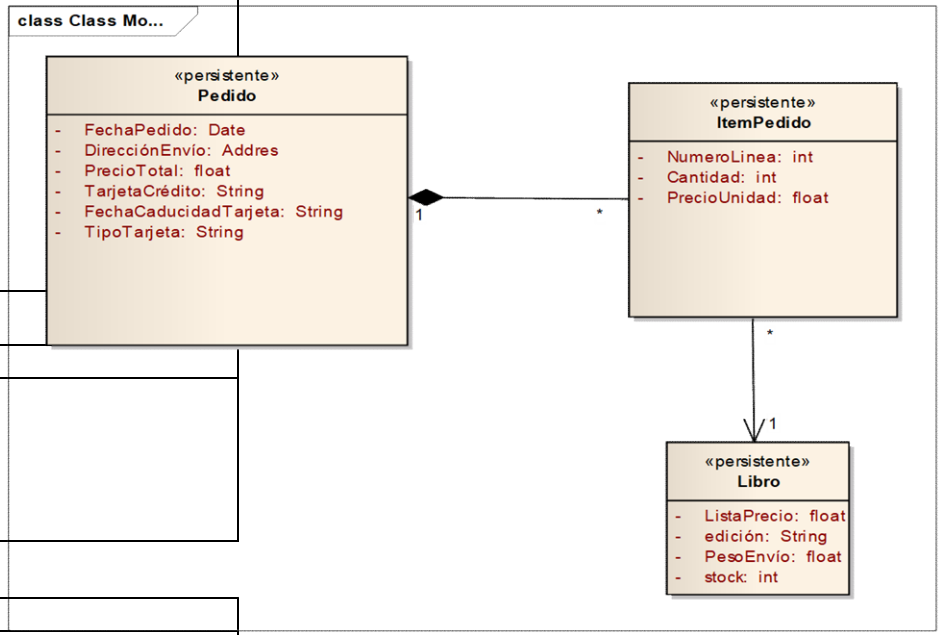


Del modelo estático al modelo de datos relacional.

PEDIDO_	
PEDIDOID	INTEGER
FECHAPEDIDO	DATE
CALLE DIRECCIÓN ENVÍO	VARCHAR(255)
CIUDAD DIRECCIÓN ENVÍO	VARCHAR(255)
PAIS DIRECCIÓN ENVÍO	VARCHAR(255)
PRECIO TOTAL	DECIMAL(10.2)
TARJETA CRÉDITO	VARCHAR(255)
FECHA CADUCIDAD TARJETA	VARCHAR(255)
TIPO TARJETA	VARCHAR(255)

LIBRO	
LIBROID	INTEGER
LISTA DE PRECIOS	DECIMAL(10.2)
EDICIÓN	VARCHAR(255)
PESO ENVÍO	DECIMAL(10.2)
STOCK	INTEGER

ITEMPEDIDO_	
LIBROID	INTEGER
PEDIDOID	INTEGER
LIBROID	INTEGER
NÚMERO DE LÍNEA	INTEGER
CANTIDAD	INTEGER
PRECIO UNIDAD	DECIMAL(10.2)



Del modelo estático al modelo de datos relacional.

- Reglas aplicadas:
- Todas las clases se mapean a una tabla, donde cada atributo es un campo de la tabla, es decir, las reglas serían *clase a tabla* y *atributo a columna.*
- Mapeo de los datos básicos:
  - Un cadena UML será mapeada a SQL VARCHAR(40)
  - Un entero UML será mapeado a INTEGER
  - Una fecha será mapeada a DATE
  - Un float será mapeado a DECIMAL(10.2)
-
- Mapeo de dirección
  - Incluir el tipo de dato en la tabla que contenga los atributos: calle, ciudad, país
-
- Mapear una clave primaria como el nombre de la clase añadiendo “ID”

Del modelo estático al modelo de datos relacional.

› Mapear asociaciones:

If (la asociación A hacia B tiene una clase asociación o la multiplicidad en ambos extremos es mayor que 1) them

crear una tabla que represente a la asociación y crear una clave foránea en ambas tablas que represente a la tabla A yla tabla B

else if (la multiplicidad es de uno a muchos) them

crear clave foránea en la tabla del extremo muchos

else if (la multiplicidad de un extremo es cero a uno) them

Crear una clave foránea en la tabla del extremo cero

else // la multiplicidad es uno a uno

Crear una clave foránea en una de las tablas que referencie a la otra

Del modelo estático al modelo de datos relacional.

- 1.- Por cada clase, se genera una clase destino con el mismo nombre.
- 2.- Cada atributo y cada operación de cada clase, estarán en la clase destino, con los tipos de datos correspondientes.
- Un String de UML se traduce a un String de Java.
- Un Integer de UML se traduce a un int de Java.
- Un Date de UML se traduce a un Date de Java.
- Un Real de UML se traduce a un float de Java.
- Cualquier otro tipo de dato se traduce a una clase con el mismo nombre, los mismos atributos y métodos para acceder a cada uno de esos atributos.

Del modelo estático al modelo de datos relacional.

- 5.- Por cada asociación en el PIM entre estas clases, se genera una asociación en el modelo destino, con los mismos nombres en los finales de asociación.
- 6.- Por cada final de asociación navegable con cardinalidad uno (1), se agrega un método getter y un setter en la clase origen de la asociación, del modelo destino.
- 7.- Por cada final de asociación navegable con cardinalidad muchos (*), se agrega un método getter para recuperar la colección en la clase origen de la asociación, del modelo destino. Además, para poder modificarla, se agrega también un método add y un método remove (concatenado con el nombre del final de la asociación).
- 8.- Si la clase es persistente (estereotipo <<persistent>>), se le agrega a la clase destino un atributo identificador de tipo Integer. Por ejemplo: En el PIM, la clase Category está marcada como persistente, entonces la clase destino contendrá el atributo CategoryID de tipo Integer.
-

3.- Patrones Arquitectónicos.

3.1.- Arquitectura Software en Subsistemas
3.2.- Arquitectura Software Orientada a Objetos
3.3.- **Arquitectura Software Cliente/Servidor**
3.4.- Arquitectura Software Orientada a Servicios
3.5.- Arquitectura Software Orientada a Componentes

3.3.- Arquitectura Software Cliente/Servidor

3.3.1-Introducción

3.3.2-Patrones cliente/servicio

3.3.2.1- P. A. múltiples clientes único servicio

3.3.2.1- P. A. múltiples clientes múltiples servicio

3.3.2.1- P. A. multiniveles clientes/servicio

Diseño de Arquitectura Software Cliente/Servidor

› Un servidor es un proveedor de servicios y un cliente es un solicitante de los servicios.



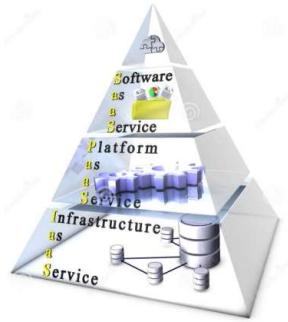

Diseño de Arquitectura Software Cliente/Servidor

- Un servidor es un proveedor de servicios y un cliente es un solicitante de los servicios.
- Servidores típicos son: de archivos, de bases de datos y servidores de impresora de línea.
- La arquitecturas cliente/servidor se basan en patrones de arquitectura cliente/servicio.

Diseño de Arquitectura Software Cliente/Servidor

- Hay que diferenciar entre un servidor y servicio.
  - Un servidor es un sistema software/hardware que proporciona uno o más servicios para múltiples clientes.

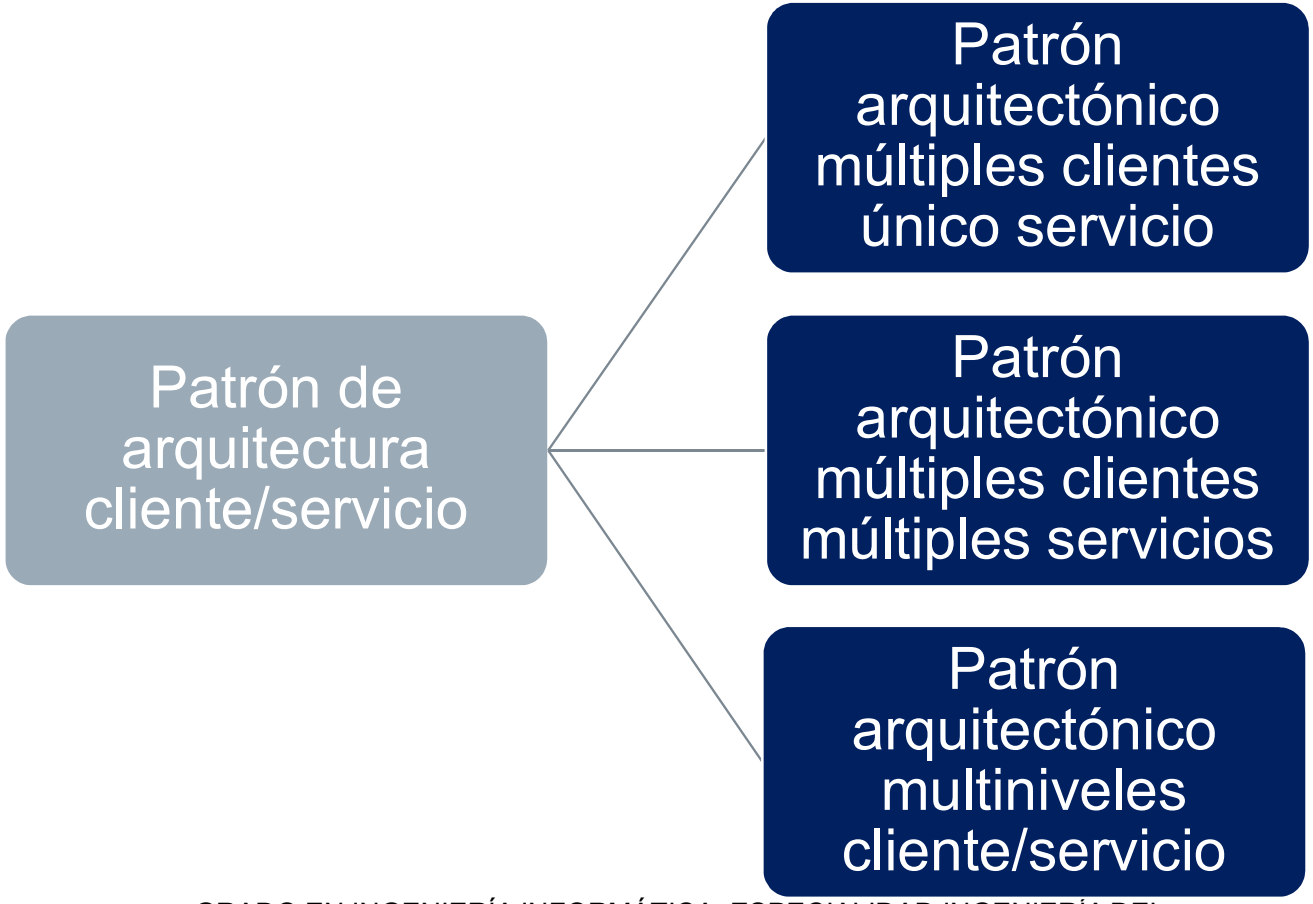
  - Un servicio en un sistema cliente/servidor es un componente software de aplicación que cumple con las necesidades de varios clientes.



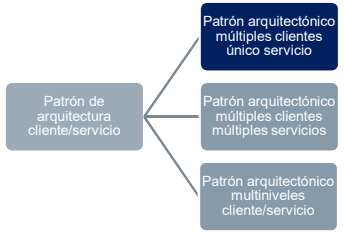
Diseño de Arquitectura Software Cliente/Servidor

- Patrón de arquitectura cliente/servicio:
  - Patrón arquitectónico múltiples clientes único servicio
  - Patrón arquitectónico múltiples clientes múltiples servicios
  - Patrón arquitectónico multiniveles cliente/servicio

Diseño de Arquitectura Software Cliente/Servidor

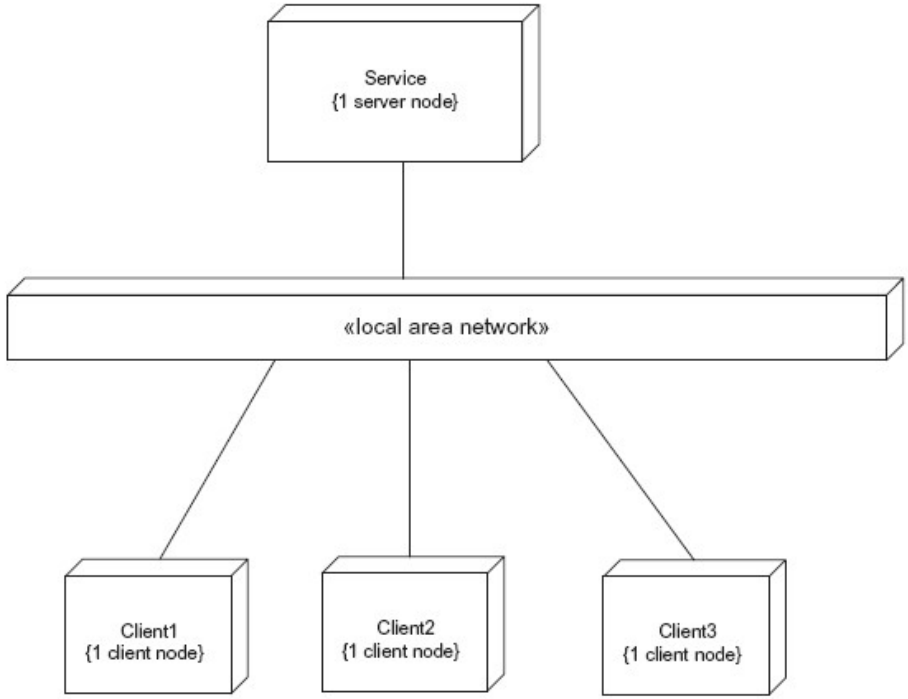


Diseño de Arquitectura Software Cliente/Servidor

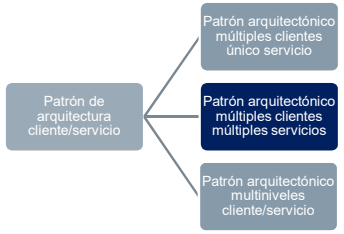


- Patrón arquitectónico múltiples clientes único servicio. Consta de varios clientes que solicitan un servicio y un servicio que cumple con las solicitudes de los clientes.

Ejemplo: Correo Electronico, Spotiffy, ...



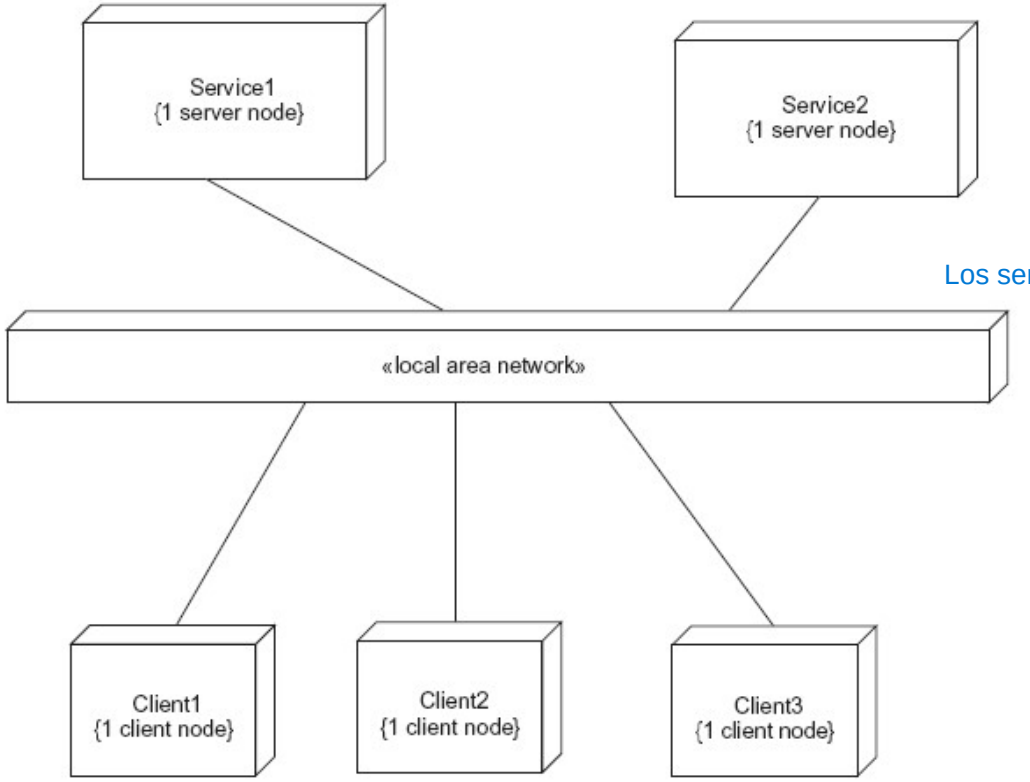
Diseño de Arquitectura Software Cliente/Servidor



- Patrón arquitectónico múltiples clientes múltiples servicios. En este patrón los servidores pueden soportar múltiples servicios.
- Los clientes que solicitan algún servicio pueden comunicarse con varios servicios, y los servicios pueden comunicarse entre sí.
- La comunicación del cliente con cada servicio puede ser secuencial o podrían comunicarse con múltiples servicios simultáneamente.

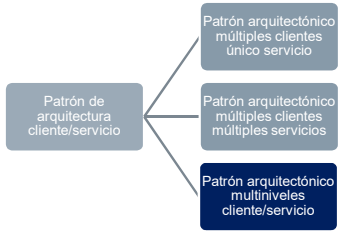
Ejemplo: aplicaciones bancarias, ...

Diseño de Arquitectura Software Cliente/Servidor



Los servicios están en el mismo servidor

Diseño de Arquitectura Software Cliente/Servidor



› Patrón cliente/servicio multicapa o multinivel:

Ejemplo: VPNs, Cajero automático, ...

– Este patrón tiene una capa intermedia que proporciona ambos roles, el de cliente y el de servicio.
– La capa intermedia es un cliente de su capa de servicio, y también ofrece un servicio para sus clientes.
– Es posible tener más de un nivel intermedio.
– Cuando se ve como una arquitectura de capas, el cliente se considera que está en una capa más alta que el servicio porque el cliente depende del servicio y lo utiliza.

Diseño de Arquitectura Software Cliente/Servidor

› Ejemplo

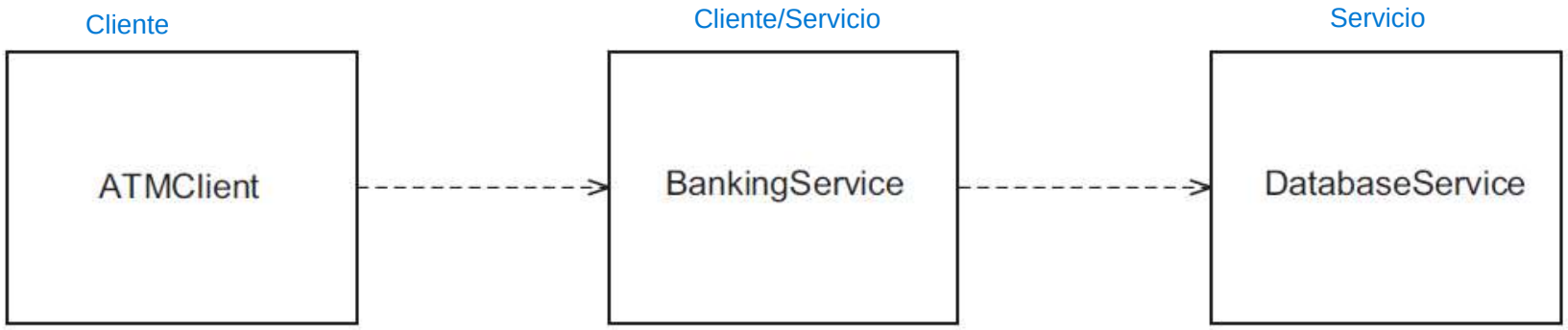


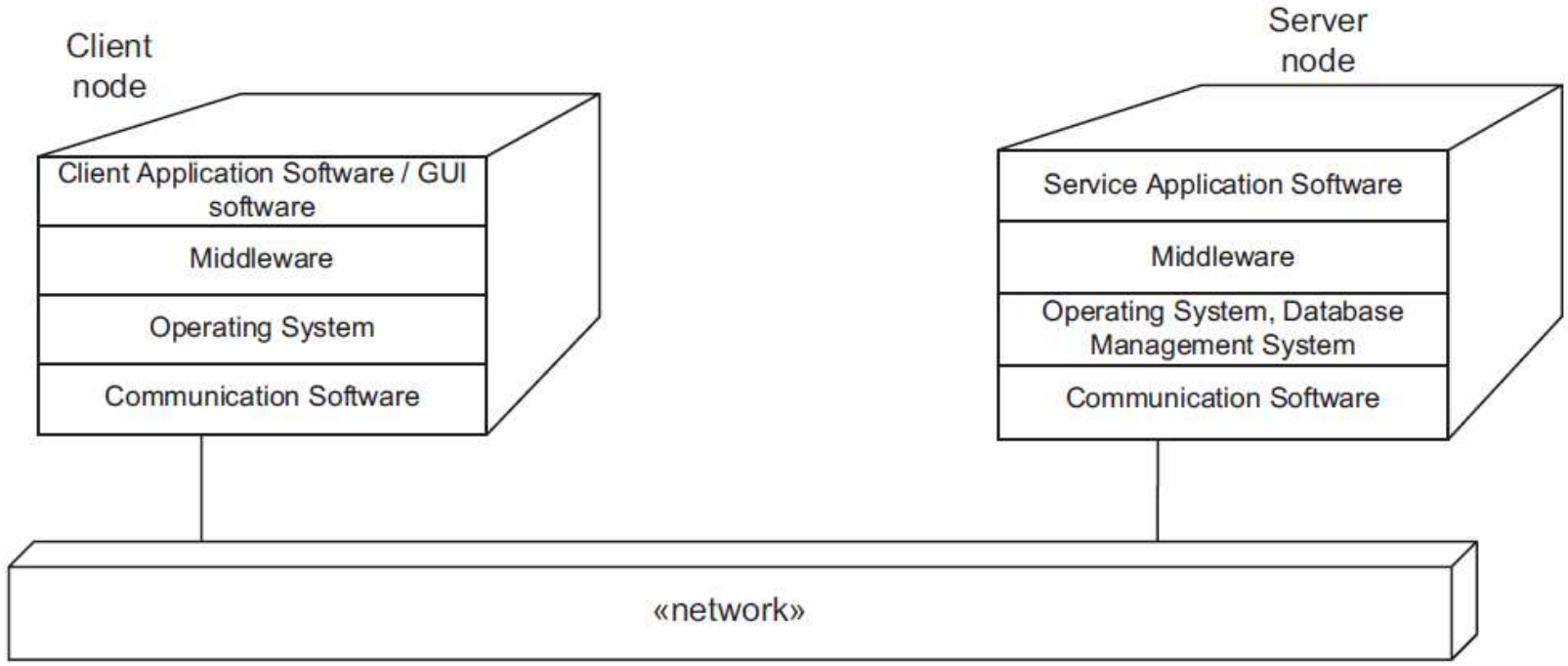
Figure 15.6. Example of the Multi-tier Client/Service architectural pattern: a three-tier Banking System

Diseño de Arquitectura Software Cliente/Servidor

- **Middelware.** Middleware es una capa software que se sitúa por encima de sistemas operativos heterogéneos para proporcionar una plataforma para las distintas aplicaciones, como por ejemplo sistemas cliente/servidor distribuidos.
- Una forma temprana de middleware fue el denominado procedimiento de llamada remoto o remote procedure call (RPC).
- Otros ejemplos de la tecnología de middleware son:
  - Distributed Computing Environment (DCE), que utiliza la tecnología RPC;
  - Java invocación de métodos remotos (RMI),
  - Component Object Model (COM);
  - Jini Java 2 Platform Enterprise Edition (J2EE)
  - Common Object Request Broker Architecture (CORBA).

Diseño de Arquitectura Software Cliente/Servidor

› Ejemplo



Ejemplo de la arquitectura multicapa es el modelo en tres capas

› La arquitectura en tres capas, consta como su propio nombre indica de 3 capas: capa cliente, capa lógica y capa servidor.

› El objetivo es hacer cada capa independiente del resto y que cada una se comporte de forma transparente.

Ejemplo de la arquitectura multicapa es el modelo en tres capas

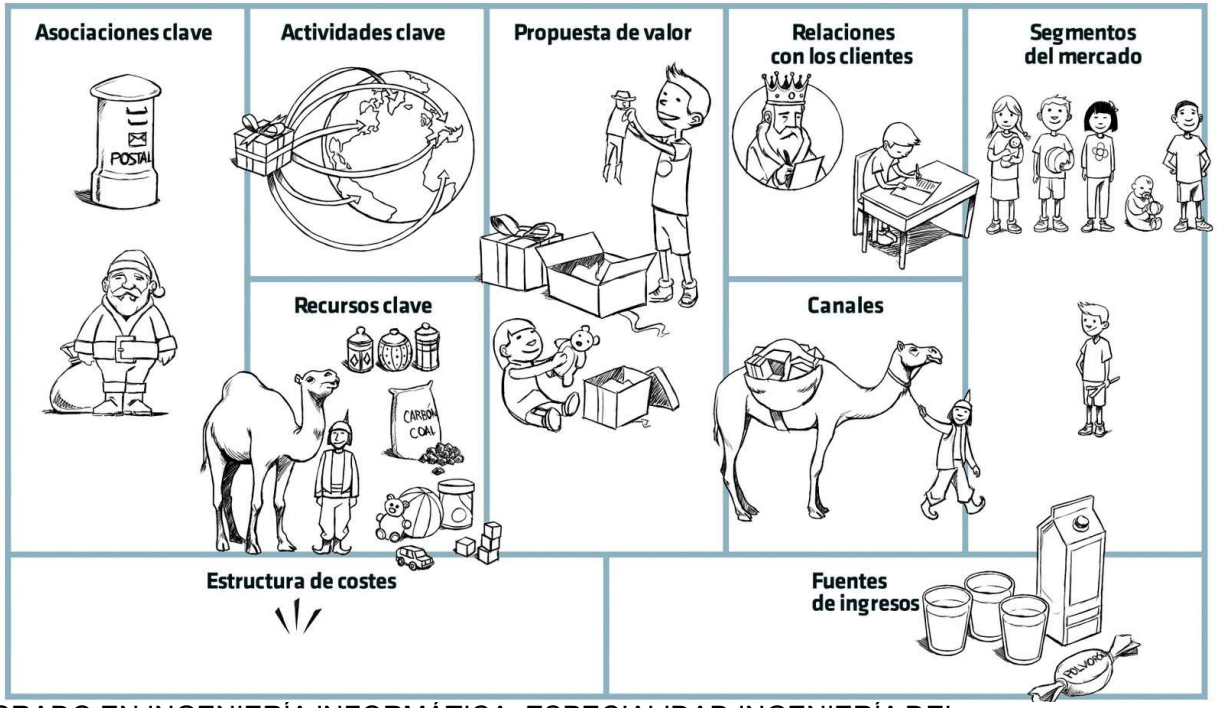
› La parte cliente queda limitada a una simple interfaz gráfica, en la segunda capa quedará la lógica de negocio y en la tercera capa queda el SGBD.

Ejemplo de la arquitectura multicapa es el modelo en tres capas

› La capa cliente puede ser una aplicación, una página web, etc. Su única función es interactuar con el usuario

Ejemplo de la arquitectura multicapa es el modelo en tres capas

› La capa lógica introduce el modelo, es la capa intermedia y proporciona una serie de servicios a los clientes.



Ejemplo de la arquitectura multicapa es el modelo en tres capas

› Capa servidor, esta capa consiste en un gestor de BD, aunque existen muchos tipos de SGBD, haciendo que la capa lógica no sea totalmente independiente esto se puede subsanar mediante SQL.

Ejemplo de la arquitectura multicapa es el modelo en tres capas

- Interconexión entre capas:
  - Enlace cliente agente: Java RMI (Remote Method Invocation), DCE (Distributed Computing Enviroment), DCOM(Distributed Component Objetc Model) y CORBA (Common Object Request Broker Architecture) (las más adecuadas pueden ser estas últimas)
  - Enlace agente servidor: Además de las rutinas proporcionadas por el fabricante del SGBD que tienen el problema de hacer que la capa lógica sea menos independiente, se puede usar ODBC(Open DataBase Connectivity) o JDBC(Java DataBase Connectivity) , lo cual consiste en hacer todas las consultas en SQL, esto permite cambiar el SGBD con facilidad.

Ejemplo de la arquitectura multicapa es el modelo en tres capas

- Ventajas de la arquitectura en tres capas:
  - Reutilización.
  - Escalabilidad.
  - Distribución.
  - Heterogeneidad.
  - Se adapta muy bien a aplicaciones en internet.
  - Mantenimiento
  - Eficiencia.

Ejemplo de la arquitectura multicapa es el modelo en tres capas

- Desventajas:
  - Seguridad.
  - Uso intensivo de la red.

3.- Patrones Arquitectónicos.

3.1.- Arquitectura Software en Subsistemas
3.2.- Arquitectura Software Orientada a Objetos
3.3.- Arquitectura Software Cliente/Servidor
3.4.- **Arquitectura Software Orientada a Servicios**
3.5.- Arquitectura Software Orientada a Componentes

Arquitectura Software Orientado a Servicios

- La Arquitectura Software Orientada a Servicios → múltiples servicios autónomos.
- Los servicios son distribuidos → diferentes nodos → diferentes proveedores de servicio.
- El objetivo de la Arquitectura software Orientada a Servicios es el desarrollo de aplicaciones software que se componen de servicios distribuidos, tal que los servicios individuales pueden ejecutarse sobre diferentes plataformas y ser implementados en diferentes lenguajes.

Arquitectura Software Orientado a Servicios

- Un proveedor de servicios soporta servicios usados por múltiples clientes.
- La diferencia con la arquitectura cliente/servidor, es que en ésta el cliente se comunica con un servicio específico proporcionado por una configuración fija de servicios, la Arquitectura Software Orientada a Servicios se basa en un concepto que es el de servicios débilmente acoplados que se comunican con los clientes con la ayuda de agentes de servicio.

Arquitectura Software Orientado a Servicios

- **Conceptos, Arquitecturas y patrones para arquitecturas orientadas a servicios**
  - **E**l objetivo de AOS es diseñar servicios como componentes autónomos reusables.
  - Los servicios están destinados a ser autónomo y débilmente acoplado, lo que significa que las dependencias entre los servicios se mantienen a un mínimo.

Arquitectura Software Orientado a Servicios

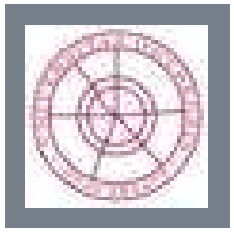
- **Los servicios deben ser diseñados de acuerdo a ciertos principios:**
  - **Débil acoplamiento.**
  - **Contrato de servicios.**
  - **Autonomía.**
  - **Abstracción.**
  - **Reutilización.**
  - **Poca información acerca de los clientes.**
  - **Detectabilidad.**





Arquitectura Software Orientado a Servicios

- Tecnologías que soportan la arquitectura orientada a servicios:
- SOAP
- .NET, J2EE, Web-Sphere, y WebLogic.



Arquitectura Software Dirigida por Modelos.