Departamento de Lenguajes y Sistemas Informáticos

Unit 8. Layered model

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Escuela Politécnica Superior
Universidad de Alicante

Introduction: Design of Database Access Components

- We start the design deciding how to access and represent the data of our application
- This unit provides a guide in order to help choosing the more appropiate way of exposing, representing and making persistent the data of an application

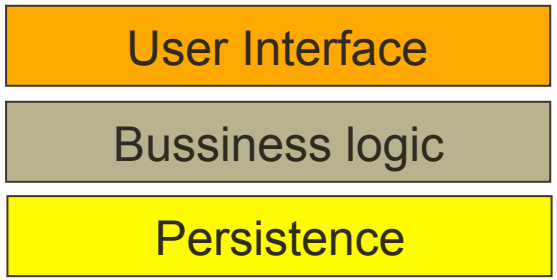
Layered architecture

- Architecture pattern [Buschmann] that establishes the distribution of an application with logic divisons developed and maintaned as independent modules, even in different platforms.

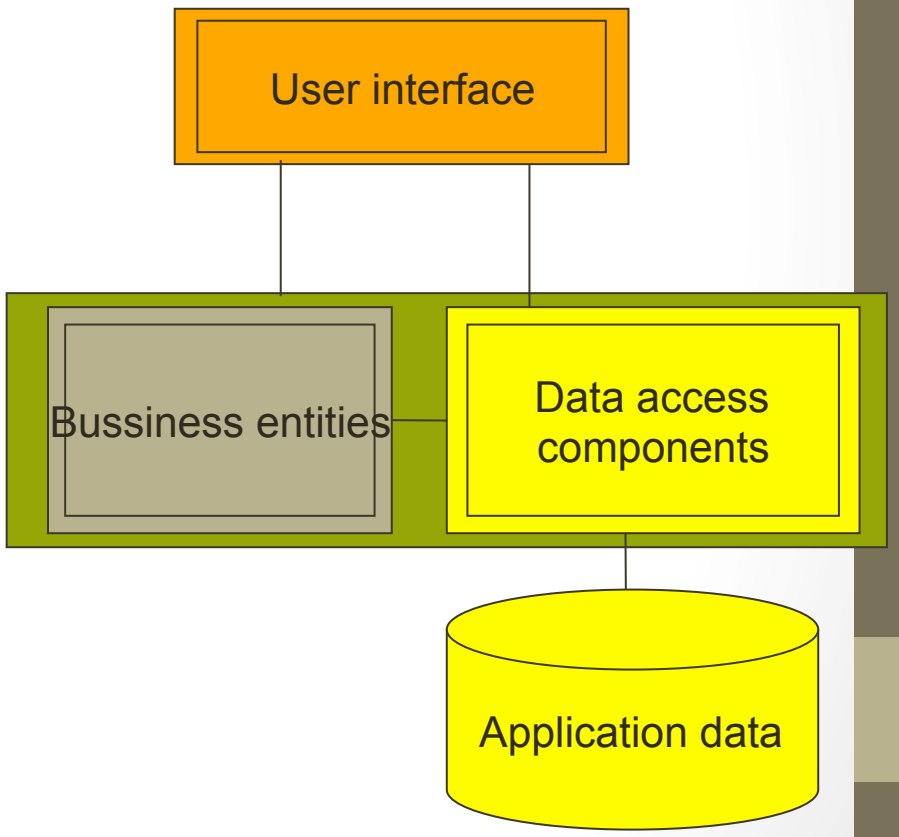
- A 3 layered application is divided into:
  - **User interface**, components which interact with the final user
  - **Bussiness logic**, contains the bussiness rules of our application
  - **Persistence**, contains the access and storing of the data

Architecture of an Application

3 Logical layers



Configuration of components



Bussiness entities(EN)

- Components which represent bussiness entitites of the real world, f.e. a product, an order
- They normally contain the information of a domain class with its attributes, operations and restrictions. Although they could represent a composition of classes
- They have associated a CAD (Data Access Component) which provides them with the access and data mapping
- They can be represented in multiple ways: personalized classes, DataSets, XML, etc.

Representation of an EN

```
public class ENProduct
{
// Private fields for maintaning the
// status of the Product entity
private int idProduct;
private String name;
private String amountByUnit;
private decimal unitPrice;
private int unitsStock;
private int minStock;
// Properties for exposing the
// product status
public int IdProduct
{
get { return idProduct; }
set { idProduct = value; }
}
```

```
public String Name {
  get { return name; }
  set { name = value; }
}
public String AmountByUnit
{
get { return amountByUnit; }
set { amountByUnit =
value; }
}
public decimal UnitPrice
{
get { return unitPrice; }
set { unitPrice = value; }}
…
```

Representation of an EN (II)

```
// Methods that do some processing
public void IncrementUnitPrice (decimal amount)
{
      unitPrice += amount;
}

public short UnitsOverMinimalLevel
{
      get { return (short)(unitsStock - minStock); }
}
}//End of the class
```

Data Access Components

- The Data Access Components (CADs) encapsute the data access technology and the DB from the rest of the application
- -They provide a simple interface that allows recovering the data from the DB and storing a EN in the DB
- The CADs also contain any bussiness logic we need for reaching the operations related with the data

Operations of a CAD

- A CAD should provide the methods for doing the following tasks over the DB:
  - Create registers in the DB
  - Read registers of the DB and return the EN to the invoking component
  - Update registers of the DB, using bussiness entities provided by the invoking component
  - Delete registers from the DB

- This methods are called CRUD, acronym of “Create, Read, Update and Delete”

Operations of a CAD (II)

- The CAD can also contain methods that do some filter. For instance, a CAD can have a method for finding the most sold product in a catalog during a month
- A CAD accesses a unique DB and encapsulates the operations related with a unique table or group of related tables of the DB
- For instance, you can define a CAD that controls the tables Orders and OrderLine

Example of CAD in .NET

CAD for the Client Class

```
public class ClienteCAD
{
private String conexion;
publica ClienteCAD()
{
// Gets the string connection from a unique location
}
public ENCliente dameCliente (String id)
{
// Code for recovering a DataSet type containing Client data
}
public String Crear (String nombre, String direccion, String ciudad,
String pais, int codPostal){
```

Example of a CAD (II)

```
// Code for creating a client based on the escalar parameters
// It returns the ID of a client in this method.
 }
 public void Actualizar (ENCliente clienteActualizado)
 {
 //Code for updating the DB, based on the data of the client sent as a
//parameter of the type ClienteDataSet
 }
 public void Borrar (String id)
 {
 // Code for deleting the client with the specified ID
 }
 public DataSet dameClientesPorCiudad (string ciudad)
 {
 // Code for recovering clients using a search criteria.
 }}
```

CAD method: BorrarCliente

```
// Method for recovering the Name of the Client
public void BorrarCliente( String clienteID )
{
SqlConnection conn = null;
// Encapsulates all the data access inside the try
String comando = “Delete from Cliente where id = “+ clienteID;
try
{
conn = new SqlConnection(conexion);
conn.Open();
SqlCommand cmd = new SqlCommand(comando, conn );
```

CAD method: BorrarCliente (II)

```
 cmd.ExecuteNonQuery();
}
catch (SqlException sqlex)
{
// Encapsulates the actual exception in a more relevant exception
throw new CADException (“Error borrando el cliente: " + clienteID, sqlex );
}
catch (Exception ex)
{
// It catches the general condition and resends it.
throw ex;
}
finally
{
  if(conn != null) conn.Close(); // We assure that the connection is closed.
}}
```

CAD Method: ObtenerClientesPorCiudad

```
// Method for recovering the clients of a certain city
public DataSet ObtenerClientesPorCiudad( String ciudad )
{
SqlConnection conn = null;
DataSet dsClientes = null;
// Encapsulates all the data access inside the try
string comando = “Select * from Cliente where ciudad = “+ ciudad;
try
{
  conn = new SqlConnection(conexion);
  SqlDataAdapter sqlAdaptador = new SqlDataAdapter (comando, conn);
```

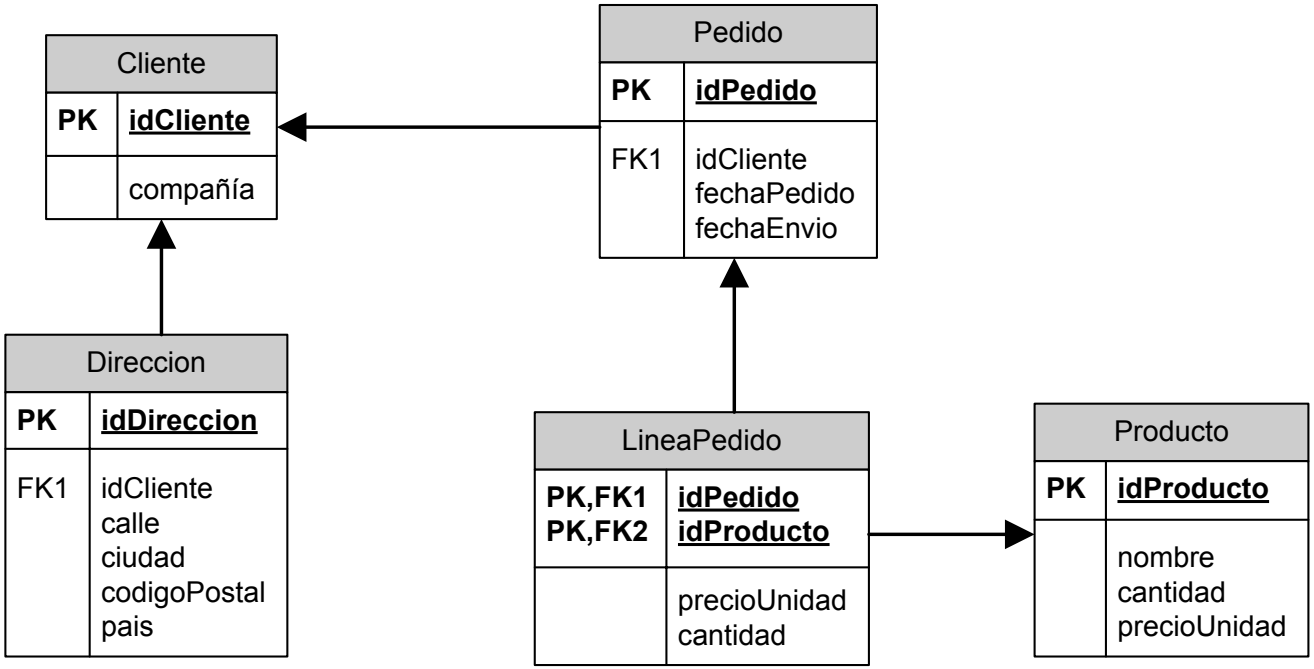
Método CAD: ObtenerClientesPorCiudad

```
 dsClientes = new DataSet();
 sqlAdaptador.Fill (dsClientes);
 return dsClientes;
}
catch (SqlException sqlex)
{
  throw new CADException (“Error en la consulta de clientes por ciudad: " +
clienteID, sqlex );
}
catch (Exception ex)
{
// Catches the general condition and resends it.
throw ex;
}
finally
{
if(conn != null) conn.Close(); // We assure that the connection is closed.
```

From Relational to Bussiness Entity

- A DB contains multiple tables with relations and we have to decide how to map the tables into different EN
- When we define the EN we have to consider “how” we are using the information in the application
- It is better identifying the core of EN which encapsulate the functionality of the application, before defining an EN for each table

From Relational to Bussiness Entity



Reduced Data Base of a small Shop

From Relational to Bussiness Entity

- The minimal functional requirements of a shop are:
  - Obtaining information about the Client, including his addresses
  - Obtaining the list of orders for a client
  - Obtaining the list of items for a particular order
  - Sending a new order
  - Obtaining or updating the information of a product or set of products

From Relational to Bussiness Entity

- For fulfilling these requirements, we can do it defining three logic EN that control the application:
  - A Client that will contain his addresses
  - An Order that will contain its lines of order
  - And a Product

From Relational to Bussiness Entity

- For each EN, we define one CAD that will be defined as follows:
  - ClienteCAD: This class provides the services for recovering and modifying the data in the Cliente and Dirección tables
  - PedidoCAD: This class provides the services for recovering and updating the data of the Pedido and LineaPedido tables
  - ProductoCAD: This class provides the services needed for recovering and updating the data of the Producto table

From Relational to Bussiness Entity: Recommendations

- Take your time for analyzing and modeling the EN of your application, instead of defining one EN for each table
- Base yourself on UML compositions and inheritance for composing complex objects
- Do not define separated EN for representing many-to-many tables. These relationships can be implemented by means of collections in the implied EN.

From Relational to Bussiness Entity: Recommendations

- Define all the methods that return a specific tyoe of a EN in a unique CAD
  - For instance, if we are getting all the orders of a certain client, implement a function in PedidoCAD called ***ObtenerPedidosPorCliente*** which returns all the orders filtered by an idCliente
  - In an opposite way, if we are getting all the clients that have done an order of a specific product, implement a function in the ClienteCAD ***ObtenerClientesPorProducto***

Another possible tasks in a CAD

- CADs can also do some other tasks in their implementation:
  - Controlling the security and authorization
  - Paginating the data
  - Doing transactions of complex entities
  - Invoking to stored procedures