Componentes de Negocio (BC)

Los componentes de negocio contienen la lógica de negocio del sistema de información. Dichos componentes son los únicos encargados de actualizar o consultar la base de datos a través de su Componente de Acceso a Datos (DAC).

Los componentes de negocio no contienen datos, solo comportamiento. Los datos los obtienen por parámetro a través de valores escalares (representativos del negocio, por ejemplo ClienteID) o Entidades.

Los componentes de negocio no pueden interactuar con otros componentes de negocio. Por ejemplo, el componente de negocio ClienteBC, no puede interactuar con el componente de negocio PersonaBC a efectos de registrar una persona y luego registrar un Cliente. Para tal propósito se deberá diseñar una componente de negocio de más alto nivel que sea orquestadora de las demás componentes.

A continuación se muestran dos ejemplos:

- El primero permite registrar un Cliente, recibiendo una entidad de negocio como parámetro.
- El segundo permite retornar una colección de Clientes.

Ejemplo 1

```
// Componente de negocio que simula
// crear un Cliente.
namespace Allus.SitemaContable.BackEnd.Cliente.BC
{
    public class ClienteBC
    {
        public bool Create(Cliente pCliente, TelefonosCliente pTelefonosCliente )
        {
            ClienteDAC wClienteDAC = new ClienteDAC();
            int wIdCliente = wClienteDAC.Insert(pCliente);

            TelefoniaDAC.CrearTelefonosCliente(wIdCliente , pTelefonosCliente);

            return wIdCliente ;
        }
    }
}

// Componente que crea un Cliente en
// la base de datos.
namespace Allus.SitemaContable.BackEnd.Cliente.DAC
{
  public class ClienteDAC
  {
    public bool Insert (Cliente pCliente)
    {
    int retVal = SqlHelper.ExecuteNonQuery(
                                                conn,
                                                CommandType.StoredProcedure,
                                                "[Cliente_i]",
                                                new SqlParameter[]
    {new SqlParameter( "@ Name ", pCliente.Name)});
                            return true;
        }
    }
}
```

Ejemplo Nro. 2

```
// Componente que consulta los Clientes
// de un vendedor en la base de datos.
namespace Allus.SitemaContable.BackEnd.Vendedor.BC
{
public class VendedorBC :
{
    public DataSet ConsultarClientes(int pVendedorID)
    {
    VendedorDAC wVendedorDAC = new VendedorDAC();
    ClienteSBE wClienteSBE = null;

    DataSet wDtsCli = VendedorDAC.ObtenerClientes(wVendedorID, wClientes);
    ____________________
     wClienteSBE  = Helpér.MappingCliente(wDtsCli  )
    }
}
}

// Componente que obtiene los Clientes
// de un vendedor desde la base de datos.
namespace Allus.SitemaContable.BackEnd.Vendedor.DAC
{
public class VendedorDAC
{
    public DataSet ObtenerClientes(int vendedorID, ClienteSBE Clientes)
    {
____________________
    Database wDataBase = null;
    DbCommand wCmd = null;

    try
    {
        wDataBase = DatabaseFactory.CreateDatabase();
        wCmd = wDataBase.GetStoredProcCommand("Articulos_s");

        return wDataBase.ExecuteDataset("Articulos_s");
    }
____________________
    catch (Exception ex)
        {
          throw Fwk.Exceptions.ExceptionHelper.ProcessException(ex);
        }
    }
}
```

Conclusiones:

No será posible tener referencias circulares entre componentes de negocio.

Una manera de evitar las referencias cruzadas es crear un BC a un nivel más alto como intermediario. Ejemplo, si en el componente de negocio de Clientes necesito utilizar lógica del componente de negocio de direcciones y viceversa, se utilizará este nuevo BC que ejecuta al componente de negocio de direcciones y luego al de Cliente.

- Los componentes de negocio no acceden a los datos directamente, lo hacen a través de su componente de acceso a datos.
- Los componentes de negocio pueden interactuar con más de una entidad, es decir, no hay una relación uno a uno entre las entidades de negocio y los componentes de negocio.
- Los componentes de negocio se definen de acuerdo a la funcionalidad que realizan. A continuación se definen las reglas para decidir cuándo crear un componente de negocio en Entorno:

1. Si existe la relación [Clientes tiene Perfiles]. Clientes y Perfiles son entidades fuertes, y existe una entidad intermedia que registra la relación. En este caso existe un único componente de negocio llamado Cliente administra sus perfiles.

2. Existe la relación Empleado tiene Hijos. Hijo es una entidad débil mientras que Empleado es una entidad fuerte. En este caso existe un único componente de negocio llamado Empleado que administra la entidad débil.

- Los componentes de negocio no deben requerir identidad de usuario para su ejecución, la identidad del usuario (autorización) será requerida por el Dispatcher, antes de ejecutar cualquier tipo de servicio.

- La interfaz de entrada para los métodos de actualización de los BCs son id's y/o entidades (simples o colecciones).

- La interfaz de salida de los BCs pueden ser:

1. DataSets no tipificados u objetos que implementen IDataReader.

2. Un único escalar, por ejemplo un saldo dado un id de cuenta.

3. Una o más entidades simple o colección, por ejemplo:

```
void LeerClienteConDirecciones(int pIdCliente, out Cliente pCliente, out[1] DireccionSBE pDirecciones)
```

- El nombre de los métodos de los BCs, se definen utilizando la siguiente nomenclatura/patrón:

[Operación][Entidad][Aspecto]

Ejemplos:

- CrearClienteConDirección, donde Crear es la operación, Cliente es la entidad y ConDirección es el aspecto.
- ActualizarClienteDatosBásicos, donde Actualizar es la operación, Cliente es la entidad y DatosBásicos el aspecto.

Componentes de Acceso a Datos (DAC)

Los componentes de acceso a datos son el único punto de entrada a la base de datos. Por otro lado, son los que conocen la lógica de datos y como optimizar el acceso y la actualización de los mismos.

Los DAC permiten desacoplar las problemáticas de acceso a datos de la lógica de negocio.

Ejemplo (crear la cabecera de la factura por un lado y los detalles por el otro)

```
using System;
using System.Data;
using System.Data.Common;
using Microsoft.Practices.EnterpriseLibrary.Data;
```

[1] ***Nota**: el indicador out en C# para entidades desarrolladas por los programadores o el generador de código no es necesario debido siempre la entidad pasa por referencia. Solo se pone este indicador a fin de dejar mas claro el código e indicar que el método modificara totalmente o en parte la clase pasada por parámetro. Es decir luego de llamar al método anterior es altamente probable de que la clase ya no sea la misma que antes de llamada.*

```
using Fwk.Bases;
using Allus.SitemaContable.BackEnd.Facturacion.BE;

namespace Allus.SitemaContable.BackEnd.Facturacion.DAC
{
  /// <summary>
  /// TableDataGateway para Factura.
  /// </summary>
  /// <Date>2006-04-19T14:11:34</Date>
  /// <Author>moviedo</Author>
  internal class FacturaDAC
        {
                /// <summary>
                /// Crear cabecera Factura Cliente especial
                /// </summary>
                /// <param name="pFacturaBE">FacturaBE</param>
                /// <returns>void</returns>
                /// <Date>2006-04-19T14:11:34</Date>
                /// <Author> moviedo </Author>
                public void Crear(FacturaBE pFacturaBE)
                {
                        Database wDataBase = null;
                        DbCommand wCmd = null;

                        try
                        {
                                wDataBase = DatabaseFactory.CreateDatabase();
                                wCmd = wDataBase.GetStoredProcCommand("Factura i");

                                /// Fecha
wDataBase.AddInParameter(wCmd, "Fecha", System.Data.DbType.DateTime,     pFacturaBE.Fecha);
                                /// Numero
                                wDataBase.AddOutParameter(wCmd, "Numero",
System.Data.DbType.Int32, 4);
                                /// Monto
wDataBase.AddInParameter(wCmd, "Monto", System.Data.DbType.Double, pFacturaBE.Monto);
                                /// Cliente
wDataBase.AddInParameter(wCmd, "Cliente", System.Data.DbType.String, pFacturaBE. Cliente);

                                wDataBase.ExecuteNonQuery(wCmd);
                                pFacturaBE.Numero = (System.Int32)
wDataBase.GetParameterValue(wCmd, "Numero");
                        }
                        catch (Exception ex)
                 {
                   throw Fwk.Exceptions.ExceptionHelper.ProcessException(ex);
                   }                  }

   }
                                                }
```

```
using System;
using System.Data;
using System.Data.Common;
using Microsoft.Practices.EnterpriseLibrary.Data;
using Fwk.Bases;
using Allus.SitemaContable.BackEnd.Facturacion.BE;

namespace Allus.SitemaContable.BackEnd.Facturacion.DAC
{
        /// <summary>
        /// TableDataGateway para DetalleFactura.
        /// </summary>
        /// <Date>2006-04-19T14:11:34</Date>
        /// <Author>moviedo</Author>
        internal class DetalleFacturaDAC
        {
                /// <summary>
/// Crear
/// </summary>
/// <param name="pDetalleFacturaBE">DetalleFacturaBE</param>
/// <returns>void</returns>
/// <Date>2006-04-19T14:11:34</Date>
/// <Author>moviedo</Author>
public void Insert(DetalleFacturaBE pDetalleFacturaBE)
```

```
        {
                Database wDataBase = null;
                DbCommand wCmd = null;

                try
                {
                wDataBase = DatabaseFactory.CreateDatabase();
                wCmd = wDataBase.GetStoredProcCommand("DetalleFactura_i");


                wDataBase.AddInParameter(wCmd, "Numero", DbType.Int32,
pDetalleFacturaBE.Numero);

               wDataBase.AddInParameter(wCmd, "IdArticulo", DbType.Int32,
pDetalleFacturaBE.IdArticulo);

        wDataBase.AddInParameter(wCmd, "Cantidad",DbType.Int32,
pDetalleFacturaBE.Cantidad);

                wDataBase.ExecuteNonQuery(wCmd);

                }
                catch (Exception ex)
                  {
                    throw Fwk.Exceptions.ExceptionHelper.ProcessException(ex);
                    }
                }

        }
    }
```

Conclusiones:

- Los componentes de acceso a datos exponen métodos para actualizar y consultar datos en el sistema de información. Proveen funciones que permiten ejecutar consultas dinámicas. Brindan soporte al sistema de paginación (en el caso de consultas que recuperen gran volumen de registros).

- Los componentes de acceso a datos pueden interactuar con una o más tablas del modelo de datos. Por ejemplo, el componente que graba una factura, graba tanto sus datos fijos como secciones simples y secciones múltiples.

- Es muy importante, que en los SPs ejecutados por los componentes de acceso a datos, se utilice el adecuado nivel de aislamiento (ISOLATION LEVEL). Dado que los bloqueos de datos de una tabla son una de las mayores causas de contención en la DB, es necesario bloquear en forma exclusiva sólo los datos que se requiera por problemáticas de negocio.

- Analizar la posibilidad de ejecutar consultas utilizando el nivel de aislamiento READ UNCOMMITTED, siempre que la problemática de negocio lo permita.

- Cada vez que se necesite bloquear exclusivamente un registro (XLOCK), se deberá analizar cuidadosamente el impacto, inclusive se deberá analizar si no existe otro mecanismo de implementación. En un entorno con mucha concurrencia sobre los mismos registros, el XLOCK genera mucha contención y provoca tiempos de respuestas altos.

- Desde un componente de acceso a datos no se puede invocar a otro componente de acceso a datos.

- Los componentes de acceso a datos son los encargados de encriptar / desencriptar los datos en los casos que fuera necesario.

- Los componentes de acceso a datos deben ser desarrollados de forma homogénea en todo el sistema de información. Es decir, más allá de que interactúen con conjuntos de datos diferentes, el comportamiento de un componente de acceso a datos debe ser similar en todos los casos:

  - Ejecutan Stored Procedures en la DB. La única excepción es la ejecución de selects construidos dinámicamente para resolver servicios de búsqueda.

    Ej.: Una consulta avanzada que basada en templates de consultas según la particularidad de la búsqueda.

    Ej.: Hay situaciones donde se obtienen listado de ciertas tablas pero de acuerdo a lo que el usuario selecciona desde distintos ComboBox con un patrón de filtro en cascada. Tal combinación puede que requiera incluir además condiciones distintas en la cláusula Where, distintas Inner Join con otras tablas. Para tal problemática es posible que se decida armar template de script de SP para las distintas situaciones.

  - Los Store procedures a ejecutar deben ser lo más atómicos posibles, Disminuyendo al máximo reglas de negocio en c/SP.

    Ej: Un Componente DAC que inserta en una tabla registros dependiendo de la cantidad (count) de registros de un select de otra/s tabla/s . Este count deberá ser obtenido en otro componente DAC y tanto el que inserta como el que obtiene Count deberían ser llamados desde una capa superior BC o SVC

  - Deben ejecutar sentencias SQL en forma Batch a efectos de disminuir la cantidad de Roundtrips al servidor de base de datos. Los componentes de la arquitectura deben estar orientados a disminuir los roundtrips a la DB, a través del uso de esta técnica. Ahora, esto no significa que solo se pueda realizar un único roundtrip por SC o ST. El objetivo es intentar no superar los 5 o 6 roundtrips a la DB por SC o ST. Ejemplo, en el caso de tener que grabar los ítems de una factura, es recomendable que se envíe un lote de sentencias armadas con la llamada al SP correspondiente pero con diferentes parámetros.

En el siguiente ejemplo se puede ilustrar el uso de un batch para crear todos los medios de contacto de un vendedor determinado:

```
/// <summary>
/// Crea en Batch los medios de contacto de un vendedor
/// </summary>
/// <param name="pFacturaBE">FacturaBE</param>
/// <returns>void</returns>
/// <Date>2006-04-19T14:11:34</Date>
/// <Author>moviedo</Author>
internal void CrearMediosDeContacto(MedioContactoSBE pMedioContactoSBE,int
pIdVendedor)
{
    using (SqlConnection wCnn = new SqlConnection(mCnnString))
    using (SqlCommand wCmd = new SqlCommand() )
     {
     try
      {
        wCnn.Open();
       wCmd.CommandType = CommandType.Text;
       wCmd.Connection = wCnn;
       StringBuilder BatchCommandText = new StringBuilder();

       BatchCommandText.Append("DECLARE @RetVal INT; ");
       foreach(MedioContactoBE wMedioContactoBE in pMedioContactoSBE)
       {
```

```
            BatchCommandText.Append("EXEC MedioContacto i ");
            BatchCommandText.Append("@pIdVendedor = " );
           if (wMedioContactoBE.IsIdTipoMedioContacto == null)
                        {
                        BatchCommandText.Append("NULL");
                        }
                          else
{
BatchCommandText.Append(wMedioContactoBE.IdTipoMedioContacto);
                        }
         BatchCommandText.Append( ", ");

         /// IdRelacionCategoria.
        BatchCommandText.Append("@IdRelacionCategoria = " );
        BatchCommandText.Append(pIdRelacionCategoria);
         BatchCommandText.Append( ", ");

         /// CodigoPais.
          BatchCommandText.Append("@CodigoPais = " );
         if (wMedioContactoBE.IsCodigoPais == null)
          {
             BatchCommandText.Append("NULL");
          }
             else
          {
             BatchCommandText.Append("'");
              BatchCommandText.Append(wMedioContactoBE.CodigoPais);
              BatchCommandText.Append("'");
           }
         BatchCommandText.Append( ", ");

         /// IdMedioContacto (OUTPUT).
         BatchCommandText.Append("@IdMedioContacto = @RetVal; ");
                                      }
         if (BatchCommandText.Length > 0)
         {
           wCmd.CommandText = BatchCommandText.ToString();
           wCmd.ExecuteNonQuery();
           wCnn.Close();
         }
     }
         catch (Exception ex)
         {
                throw ExceptionHelper.ProcessException(ex, this);
         }
    }
}
```

- A efectos de lograr que esto sea factible todos los componentes de la arquitectura deben tener en cuenta el manejo de datos empaquetados. Por ejemplo, si es necesario grabar cinco documentos y por cada documento se debe validar el saldo del Cliente. Es conveniente que el Servicio Transaccional realice, a través de componentes de negocio (svc), primero las cinco validaciones en forma batch y luego las cinco creaciones del documento. Grabando los datos del tipo de documento (un roundtrip), los datos de las secciones simples (en forma batch) y los datos de las carteras (en forma batch).

- Acceder a la DB con un mismo usuario (string de conexión) a efectos de poder utilizar las ventajas de Connection Pooling.

- La interfaz de entrada para los métodos de actualización de los DACs son id's y/o BEs simples o colecciones. En el caso de usar id's estos pueden ser escalares o colecciones de id's de un mismo tipo de entidad.