Interfaces de servicios Request y Response

Las interfaces Request y Response son los contratos que los aplicativos clientes deberían entender y respetar con la finalidad de consumir los servicios de negocios.

Ejemplo de Implementación

Mostraremos a continuación un ejemplo práctico de como se ven implementados los objetos `Request` y `Response` y como son llamados desde una aplicación cliente.

Construcción de las interfaces del servicio (service contract)

Interfaz de petición (Request)

Lo primero que debemos crear es un esquema que represente funcionalmente los datos que serán enviados al servicio. Supongamos un servicio que cree una factura en un sistema, lo llamaremos `CrearFacturaService`. *Más adelante detallaremos como se implementa un servicio y el patrón de nombrado.*
Diseñamos el esquema correspondiente al Request:

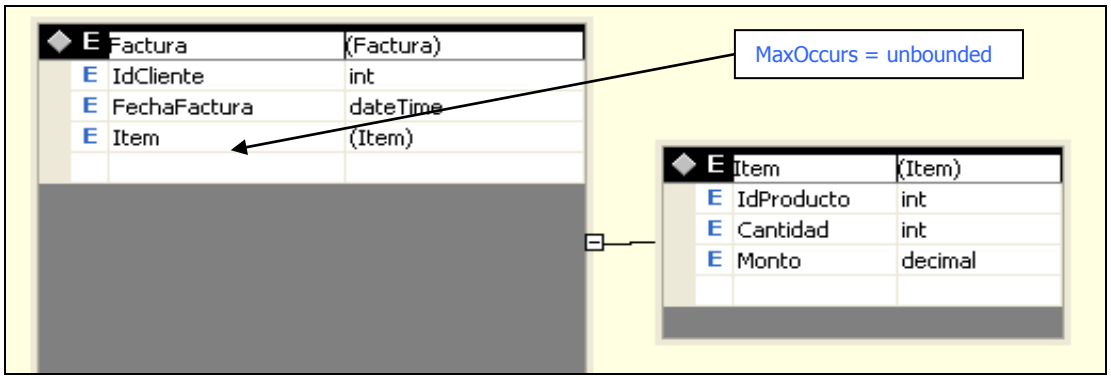

Figura 4. CrearFacturaREQ.xsd

En base a este esquema creamos la clase FacturaBE, ItemBE y ItemBECollection que heredan de Entity y corresponden a los datos funcionales de entrada al servicio. Notemos aquí que la primera clase FacturaBE es la principal y la que encapsula todos los datos de negocio y es la que tiene la posibilidad de hacer llamadas a las demás subclases que aparecen en el esquema (en este caso solo la colección de Items).

La clase FacturaBE como es la cabecera de todas las demás es la que va a ser llamada desde el objeto CrearFacturaReq principal atravez del atributo heredado "BusinessData"
Ej: CrearFacturaReq.BusinessData

Por lo tanto, una vez que tenemos diseñadas las clases de negocio, solo nos falta construir una clase que sea la que va a heredar de la clase Request, es decir CrearFacturaReq:

```
public class CrearFacturaReq : Request< FacturaBE >
{
}
```

A continuación mostramos como nos queda el código del Request completo:

```
using System;
```

```
using System.Collections.Generic;
using Fwk.Bases;
using System.Xml.Serialization;

namespace Pelsoft.SistemaContable.Facturas.ICVC
{
    public class CrearFacturaReq : Request<FacturaBE >
    {

    }

    [XmlInclude(typeof(FacturaBE)), Serializable]
    public class FacturaBE : Entity
    {
        public int? IdCliente{ get ; set; }
        public DateTime? FechaFactura   { get ; set; }
        public ItemBECollection ItemBECollection { get ; set; }
    }

    public class ItemBECollection : Entities<ItemBE>{}

    [XmlInclude(typeof(ItemBE)), Serializable]
    public class ItemBE : Entity
    {
        public int? IdProducto{ get ; set; }
        public int? Cantidad{ get ; set; }
    }
}
```

Esta clase CrearFacturaReq es la clase que realmente recibe el Dispatcher y contiene tanto información técnica o de contexto como información funcional o de negocio.
Gráficamente a un objeto Request lo podemos ver de la siguiente manera:

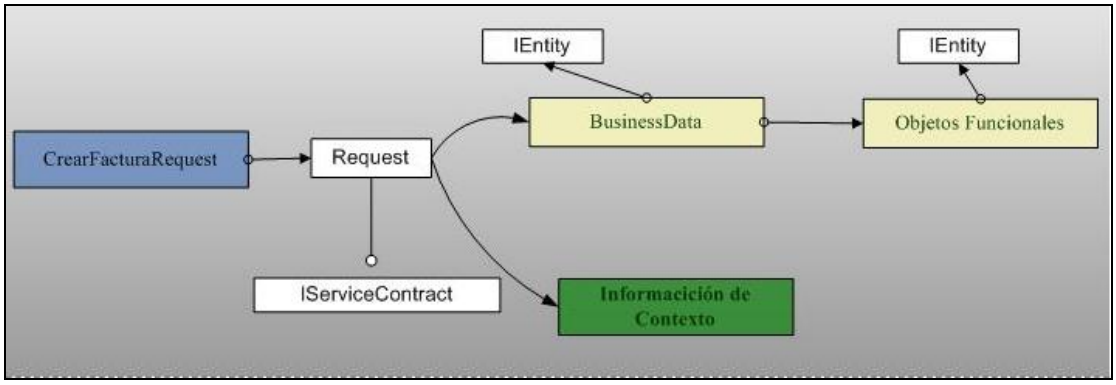

Figura 5. Modelo Request de un Servicio

Interfaz de respuesta
De manera similar necesitamos construir una interfaz que represente el resultado del servicio, es decir, el Response.
Entonces construimos el esquema XSD que lo represente, como se muestra a continuación:

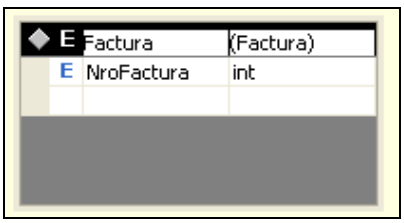

Figura 6: CrearFacturaRES.xsd

Luego desarrollamos el objeto de negocio, `FacturaBE`, que representa al esquema de arriba viéndose como se muestra en sniped debajo.

```
using System;
using System.Collections.Generic;
using Fwk.Bases;
using System.Xml.Serialization;

namespace Pelsoft.SitemaContable.Facturas.ICVC
{
   /// <summary>
   /// Request CrearFacturaRes .-
   /// </summary>
   public class CrearFacturaRes : Response< FacturaBE >
   {
   }
   public class FacturaBE : Entity
   {
      public int? NroFactura { get ; set; }
  }
```

Por último solo nos resta construir la clase CrearFacturaRes que la agregamos en el mismo namespace donde agrego la clase FacturaBE. Cuando creamos esta clase lo que estamos haciendo vemos que le pasamos como parámetro genérico, a la clase base Response, un tipo de dato definido por nosotros que es FacturaBE y esta misma clase va a ser la será el contenido del **BusinessData** del Response.

Entonces cuando se quiera acceder al valor de negocio retornado por el servicio se lo haga de la siguiente manera:

```
CrearFacturaRes res = null;
lblFactura.Text = "Factura N°: " + res.BusinessData.NroFactura;
```

Tipo genérico funcional recibido por un Request o Response (IEntity)

Como habrán notado las dos clases tanto Request como Response recientemente construidas son definidas como una clase que hereda de una clase base que recibe como parámetro genérico cualquier clase que implemente IEntity (Entity o Entities). Estas clases bases están definidas en el framework.

Por lo tanto si en nuestra clase del ejemplo:

CrearFacturaReq : Request<FacturaBE>

Es un request que recibe un objeto de negocio de tipo factura. Si quisiéramos que, en un futuro, este request pueda recibir una colección de facturas (lista), simplemente bastaría con definir una clase de tipo Entities<Entity> llamémosla FacturaList que reciba una clase FacturaBE. Es decir crearíamos una lista contenedora de objetos FacturaBE:

```
public class FacturaList : Entities<FacturaBE>{}
```

Y luego adaptar nuestra clase CrearFacturaReq para que reciba tal colección de modo que se vea de la siguiente manera:

```
[Serializable]
public class CrearFacturaReq : Request<FacturaList>
{
}
```

Esto es posible dado que la firma de las clases Requests o Responses base es:

```
:Request<T> where T : IEntity
:Response<T> where T : IEntity
```

```
Y

Entity : IEntity
Entities : IEntity
```