Frornt End

Las aplicaciones clientes pueden ser aplicaciones web, win 32 o incluso algún otro servicio que requiera hacer peticiones de algún servicio al Dispatcher.

Desde el punto de vista del cliente las piezas más importantes son:

- ClientServiceBase
- Controller
- Wrapper
- Request y Response

Wrapper

Esta pieza de software está ubicada del lado del cliente y es la encargada de recibir todas las peticiones que los clientes hacen al servidor o distpatcher.

Este componente tiene la inteligencia para establecer la información de contexto del lado del cliente de los servicios y además se encarga de la serializacion automática de los parámetros `Request` o `Responce` para enviarlos al servidor.

Por otro lado detecta la configuración de canal para poder ubicar al dispatcher y abstrae al cliente de toda complejidad de comunicación ante un dispatcher montado en un WebService o un Servicio de windows remoting.

El Wrapper dispone de un método **`ExecuteService`** que es público para los clientes y permite ejecutar un servicio de negocio.

La interfaz genérica de este método es la siguiente:

```
TResponse ExecuteService<TRequest, TResponse>(string providerName,TRequest pRequest)
            where TRequest : IServiceContract
            where TResponse : IServiceContract, new()
```

Es un método que recibe como parámetros el nombre del servicio más un objeto Request.

Como notaran este objeto debe implementar la interfaz IServiceContract y esto lo hace automáticamente heredando de la clase Request explicada anteriormente.

Por otro lado retorna un objeto Responce que implementa también la interfaz IServiceContract, que de la misma manera al heredar de la clase base Response también implementa esta interfaz.

Tipos de Wrapper:

Wrapper local

Utiliza cuna comunicación directa. No hace ninguna llamada a servicios remotos atreves de protocolos como http o TCP

Es muy útil para realizar pruebas unitarias y/o servicios donde no sea necesario el desacople con el BackEnd

Remoting wrapper

Permite la comunicación por medio del protocolo TCP con un servicio de Windows. El servicio de Windows es provisto por el framework y utiliza tecnología remoting para atender todas las peticiones de los clientes.

Esta modalidad usa serialización binaria de los servicios y es propicia para trabajar en una LAN.

Web service Wrapper:

Es el wrapper que permite la comunicación a través del protocolo HTTP con un web service utilizando el protocolo SOAP.

Implementación de los wrappers en la arquitectura

Las aplicaciones Front-End utilizan el wrapper para hacer llamadas a servicios de negocios que se ejecutan en despachadores de servicios. Dentro del framework existen clases que automáticamente realizan la instanciación de los diferentes wrappers configurados.

Estas clases son: FrmBase, UserControlBase y la clase base de ambos `ClientServiceBase`

Nosotros como desarrolladores bien podríamos hacer heredar nuestras clases formulario o user controls de `FrmBase` y `UserControlBase` respectivamente o bien, podríamos crear una clase cualquiera que no sea necesariamente visual y hacerla que herede de `ClientServiceBase.`

Lo que `ClientServiceBase` hace es abstraernos de todo tipo de complejidad de factoia de wrapper y demas cuestiones de servicios. Entre ellas:

- ✓ Cache de servicios
- ✓ Establecimiento inicial de atributos de seguridad
- ✓ Establecimiento inicial de atributos de auditoria
- ✓ Ejecución de servicios
- ✓ Manejo de errores en el cliente
- ✓ Etc.

```
public static TResponse ExecuteService<TRequest, TResponse>(string providerName,TRequest pRequest)
```

Configuración del wrapper

Ver documento: Arquitectura Tecnológica Configuración wrapper.pdf

Interfaces 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.

En el documento Arquitectura Tecnológica Servicios SOA.doc se detalla a profundidad el concepto y la construcción de interfaces de servicios.

De lo que si se hablara en este apartado es la utilización de los mismos en el FrontEnt o aplicación consumidora.

Utilización de las interfaces del lado del Cliente:

La utilización de estas clases (Request y Responses) es muy sencilla, simplemente basta con crear una instancia de la clase Request y rellenarla con la información necesaria de acuerdo las reglas de negocios exigidas y luego llamar al método del ExecuteService (..) . de la clase que implementa Request.

```
req.ExecuteService< CrearFacturaReq, CrearFacturaRes>(req);
```

La mejor forma de entender esto es a través de un ejemplo:

Supongamos un método dentro de nuestro formulario de Windows que rellene el objeto CrearFacturaReq a partir un control DataGridView. Entonces para encapsular la complejidad dejamos que toda esta tarea sea llevada a cabo en un método separado, lo llamaremos BuildRequest.

Nota: Con respecto a al información de contexto del request, no es necesario que el desarrollador se esfuerce en establecer sus valores, se debe dejar este trabajo para que sea realizado por la lógica del wrapper.

```
private CrearFacturaReq BuildRequest()
{
    CrearFacturaReq req = new CrearFacturaReq();
     req.BusinessData.IdCliente = (int) txtNumeroCliente.Text;
    req.BusinessData.FechaFactura = cmbFecha.Value;
    foreach (DataGridViewRow wRow in grdDetalle.Rows)
```

```
            {
            ItemBE wItemBE = new ItemBE();
            wItemBE.Cantidad = Convert.ToInt32(wRow.Cells["Cantidad"].Value);
            wItemBE.IdProducto = Convert.ToInt32(wRow.Cells["IdArticulo"].Value);
             req.BusinessData.ItemBEList.Add(wItemBE);
            }
    return wRequest;
}
```

Luego, una vez que llenamos el Request con la información necesaria, lo que nos resta es llamar al servicio pasándole este y esperar el resultado.

Para esto solo tenemos que enviar esta información al dispatcher y esto lo hacemos a través del ExecuteService que esta embebido en la clase base.

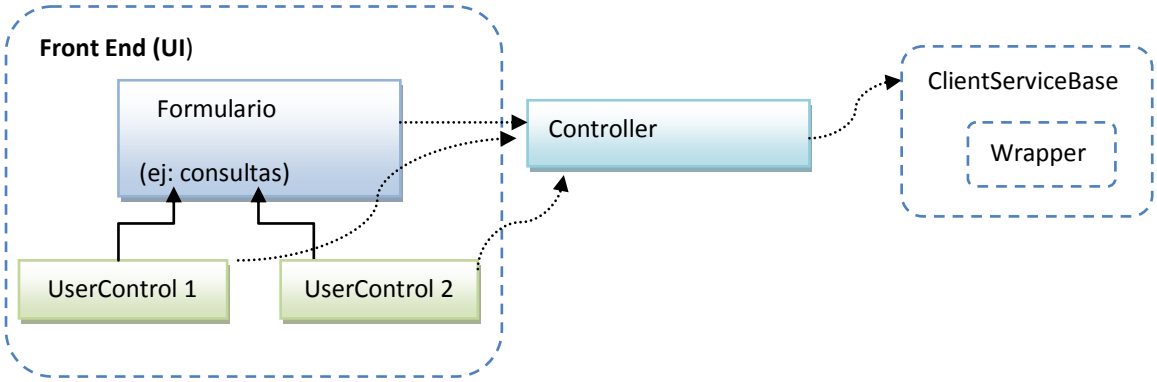
```
private void btnCrear_Click(object sender, EventArgs e)
{
   CrearFacturaReq req = BuildRequest();
   CrearFacturaRes   res = req.ExecuteService<CrearFacturaReq,CrearFacturaRes>( req);
  MessageBox.Show("Se creó la Factura N°: " + wResponse.Result.NroFactura);
  if (res.Error != null)
          throw Fwk.Exceptions.ExceptionHelper.ProcessException(res.Error);
 }
```

Nótese que en este caso no se especifica el nombre del proveedor de wrapper en la ejecución del servicio. Por lo tanto se utilizara el proveedor por defecto.

req.ExecuteService<CrearFacturaReq,CrearFacturaRes>("**WrapperProviderName**" **req**);

Arquitectura de front end

En esta sesión se intentara establecer una modalidad estándar de diseño de los componentes fron end para la comunicación a través de servicios.



Aquí tenemos un formulario cualquiera que utiliza sus user controls todos estos componentes interactúan con un componente común el controller.

Por lo tanto, los componentes de UI utilizaran una instancia singlenton del controller o un controller estático. (El diseño del controller estático o instanciado bajo patrón singlenton se deja a criterio del grupo de desarrollo)

La idea de crear un controller es solo para definir un único lugar donde se alojen todas las llamadas a los servicios. Trabajando de esta manera logramos un estándar y/o convención de trabajo, de modo que cuando ocurran cambios en llamadas a servicios o siertos mudulos del sistema dejen de utilizar una arquitectura SOA. Simplemente se actualizaran una capa/componente de la arquitectura → El controller.

Responsabilidades del controller:

- El controller es el único componente que tendrá la responsabilidad de hacer llamadas a los servicios.

Dentro de los Controllers las clases Request hacen llamadas a ***ExecuteServise***. Internamente la clase Request tiene toda la inteligencia para resolver las siguientes cuestiones:

- Identificar el o los wrapper/s de comunicación con el despachador de servicios.
- Aplica lógica de caching de acuerdo los parámetros establecidos en los componentes de chache en el Request.
- Seteo inicial de atributos de seguridad
- Seteo inicial de atributos de auditoria
- Ejecución de servicios
- Manejo de errores en el cliente
- Etc.

Patrón de ejecución de servicios en el controller:

A continuación se muestran unos ejemplos de métodos que realizan llamadas a servicios del lado del cliente por medio de un controller.

Dado el siguiente servicio:

```
Servisio:
   SearchParamsByTypeService:BussinesService<SearchParamsByTypeRequest,SearchParamsByT
   ypeResponse>

Request:  SearchParamsByTypeReq: Request<Param>

Response: SearchParamsByTypeRes:Response<ParamList>
```

Se trata de un servicio de busqueda de tipos de parámetros por tipo como por ejemplo: Tipos de Colores, tipos de empleado, tipos de salas, etc.

```
/// <summary>
/// Busca params por
/// </summary>
/// <param name=" pType">tipo de la enumeracion TypesEnum </param>
/// <returns>lista de valores tipo params</returns>
public ParamList SearchParams(ParamsEnum.TypesEnum pType)
   {
    /Creo Request
            SearchParamsByTypeReq req = new SearchParamsByTypeReq();
    //Seteo parametros
    req.BusinessData.ParamTypeId = pType;
    req.CacheSettings.CacheOnClientSide = true;

    ///Llamo al servicio y obtengo el response
    SearchParamsByTypeRes res =  req.ExecuteService< SearchParamsByTypeRequest,
                    SearchParamsByTypeResponse>
                    ("NOMBRE_PROV_WRAPPER",req);

     //Si veinen errores proceso la excepcion
      if (res.Error != null)
              throw Fwk.Exceptions.ExceptionHelper.ProcessException(res.Error);

           //retorno el resultado (colección de parametros)
         return res.BusinessData;
}
```

Note que en el controller no hay nada relacionado a coneccions a web services o servicios remoting, tampoco hay reglas de negocio ni accesos a bases de datos. Toda esa complejidad es absorvida por las

capas correspondientes como caching, blocking, logging del framework y Dispatcher,SVC, BC, DAC etc diseñados bajo los patrones de la arquitectura.

Entonces desde cualquier componente como un formulario o user control podríamos llenar una grilla haciendo una llamada al controller:

```
private void btnfillGrid_Click(object sender, EventArgs e)
{
   grgParamsGrid.DataSource =
        _Controller.SearchParams(ParamsEnum.TypesEnum.TipoEmpleado);
}
```