PBtoWS - Procesos: Guía Rápida de implementación del Componente Proxy Backend

Este capitulo contiene información relacionada con el proceso de creación de componentes proxy aplicando la arquitectura propuesta en el desarrollo backend. El objetivo de esta sección es centrar al desarrollador en los aspectos fundamentales que debe tener presente al crear componentes Proxies que serán utilizados en los consumos internos de servicios SOAP Powerbuilder. Tener presente que sólo se explicará el proceso de implementación y se excluirán los demás procesos asociados a la utilización. Para más información favor consultar los pasos del Check List Component

Paso 0: Tipos de Componente Proxy

Antes de iniciar el proceso de creación de la clase de consumo se debe identificar el tipo de proxy el cual puede ser de 2 Tipos:

- **ComponentProcessProxy** - Componente de Proceso Estandard: Este tipo de proxy se debe implementar para el consumo de servicios que representan procesos estandares del ERP.
- **ComponenteCrudProxy** - Componente de Proceso Crud: Este tipo de proxy se debe implementar para el consumo de servicios CRUD (simples).

Paso 1: Creación del Proyecto del Componente Proxy

El primer paso consiste en crear el proyecto del componente Proxy. La información relacionada de esta actividad puede ser consultada en el siguiente link Crear Componente Proxy

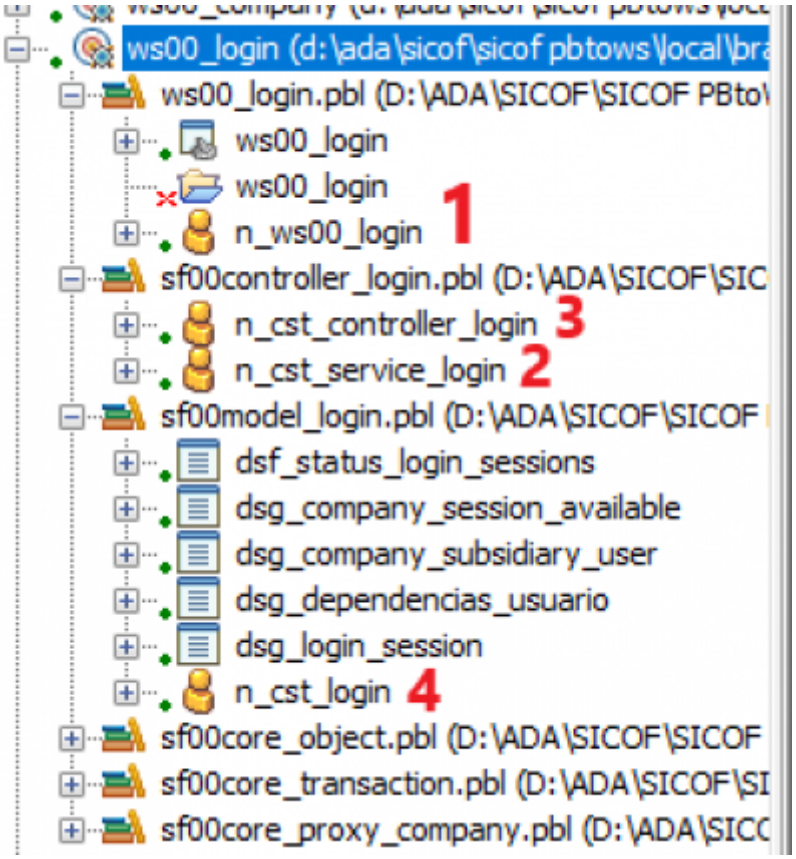
Paso 2: Crear la Clase Base del Componente

El siguiente paso consiste en definir las clases base de operación del componente las cuales deben ser extendidas (heredadas) del paquete de Resolución y Orquestación de Servicios de la sigueinte forma:

- Extender la clase **n_cst_service** para crear la clase de invocación de servicios
- Extender la clase **n_cst_controller_process** para crear los controladores que serán utilizados por las clases invocadoras **n_cst_service**
- Extener la clase **n_cst_model** para implementar la logico del negocio del componente, es decir en estas clases se generará el código fuente.

Nota: Tener presente que las clases invocadoras y controladoras deben agregarse a la libreria sf(XX)controller_(xxxxxxxx).pbl y las clases del modelo ó lógica del negocio debben ser agregadas a la libreria sf(XX)model_(xxxxxxxx).pbl.

A continuación se visualiza una imagen de ejemplo con la implementación de las clases base (tener presente el numero asociado a cada clase para identificar el tipo)



1. **Clase Lanzadora**: Es generada automáticamente al crear el proyecto en ella se registran los metodos que serán expuestos en el servicio.
2. **Clase Invocadora**: Es la clase que va a ser invocada por la Clase Lanzadora, crea los controladores y lanza los métodos que inician los procesos.
3. **Clase Controladora**: Este tipo de clases realizan validaciones, provven metodos de utilidades e invocan las clases de la lógica del negocio.
4. **Clase de Logica del Negocio**: Contiene el código powerbuilder de los procesos del ERP.

Paso 4: Modelo de implementación de invocación por capas

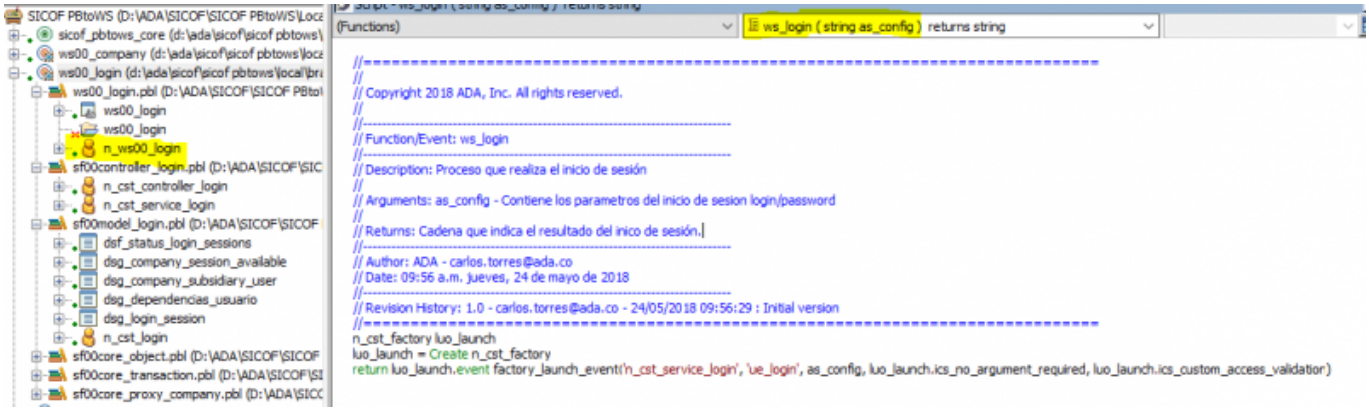
A continuación se explica con ejemplos en imagenes el modelo de implementación de invocación por las capas del framework aplicando la arquitectura propuesta. El componente de referencia es el login.

Modelo de implementación: Clase Lanzadora

El modelo debe implementar invocacion dinamica por eventos, por lo tanto la clase lanzadora debe invocar el evento **factory_launch_event** y recibe:

- Nombre de la clase invocadora
- Nombre del evento de invocación
- Parametros de configuración en formato json **as_config**

- Datos de invocación (solo si lo requiere el servicio) en formato json **as_data**
- Tipo de autenticación de servicio.



Consideraciones

- Los parametros de entrada y salida son string en formato json.
- el parametro as_data será requerido dependendiendo de la implementación de los procesos de la lógica del negocio por lo tanto no siempre es obligatorio.
- la cadena de definición de la clase invocadora y el evento de invocacion deben ingresarse en minusculas.

Modelo de implementación: Clase Invocadora

El modelo debe implementar en el evento invocado la inicialización del controlador principal y debe recibir los parametros inicializados para la ejecución de los procesos soportados. Los argumentos inicializados automaticamente son:

- iuo_json_config: Clase json que contiene los parametros de configuración del servicios EJ contiene el atributo token_session
- iuo_json_data: Clase json (opcional) que contiene los parametros asociados a la ejecución de los procesos soportados.
- iuo_msg: Clase que contiene los mensajes del sistema.
- its_db: Clase que contiene la conexión de la base de datos del cliente.
- SQLCA: conexión genérica a la base de datos de configuración.

De igual forma debe asegurar que el resultado del proceso debe ser devuelto en un objeto tipo **n_cst_return** para luego convertirlo en una cadena en formato json que debe ser asignada a la variable **is_return** para cual es la que siempre se devuelve en la Resolución y Orquestación de Servicios.

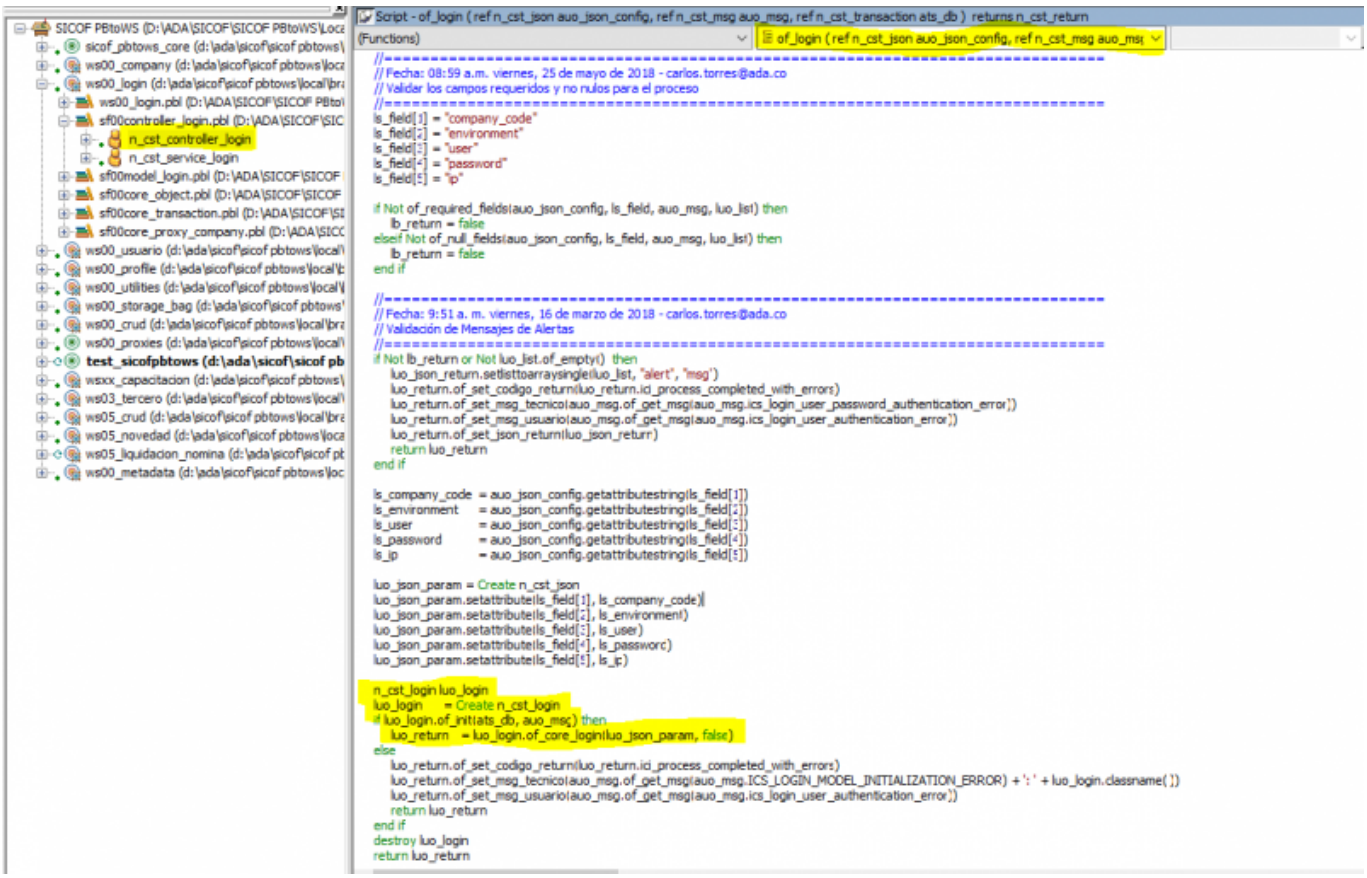
A continuación se visualiza la imagen de la implementación del Componente Login.

Consideraciones

- En lo posible evite el uso de variables de instancia en los controladores.
- No utilizar la variable SQLCA para realizar transacciones.
- No implementar lógica de negocio en las clases n_cst_service

Modelo de implementación: Clase Controladora

El modelo debe implementar un método en el controlador que permita la inicialización de la clase modelo que contiene la lógica del negocio del proceso. Ademas debe validar los argumentos de entrada y las reglas que puedan aplicar a la ejecución del proceso.



Consideraciones

- El método del controlador debe recibir todos los argumentos que requiera el proceso.
- Los argumentos del proceso que se envien a la clase modelo deben ser encapsulados en una clase json.
- Los controladores no deben procesar el objeto de retorno.
- Los controladores pueden modificar los parametros de ejecución de los procesos.
- En la mayoria de los casos se deben inicializar las variables de transacción y mensaje.

Modelo de implementación: Clases Modelo (Lógica del Negocio)

Estas clases contienen el código Powerbuilder de los procesos del ERP.

Consideraciones

- Evite en lo posible el uso de clases del framework relacionadas la arquitectura ejemplo: **n_cst_controller**, **n_cst_service**, **n_cst_factory**, **n_cst_core**
- Extienda las clases modelo deacuerdo al módulo de componente Ejemplo **n_cst_model_nomina** para los componentes del módulo de Nómina. No extienda de la clase **n_cst_model** directamente.
- Asegurese que el retorno de los procesos **n_cst_return** tiene toda la información necesaria para la respuesta, no realice implementaciones parciales entre capas.