

PBtoWS - Procesos: Guía Rápida de implementación de la Arquitectura Backend

Este capítulo contiene información relacionada con el proceso de creación de componentes 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 que serán expuestos en servicios SOAP Powerbuilder. Tener presente que sólo se explicara el proceso de implementación de la arquitectura y se excluirán los demás procesos asociados a la creación. Para más información favor consultar los pasos del [Check List Component](#)

Paso 1: Creación del Proyecto

El primer paso consiste en definir la estructura del repositorio del componente. La información relacionada oncesa actividad puede ser consultada en el siguiente link [Crear Componente](#)

Paso 2: Establecer la Clase Transacción Proyecto

Una vez creada la estructura del componente el siguiente paso consiste en redefinir la clase que mapeará las conexiones de la base de datos. Esto es necesario ya que la transacción desempeña un papel fundamental para el procesamiento de la información. La clase encargada de ese proceso es **n_cst_transaction** y está en la librería **sf00core_object.pbl**. Ubíquese en el objeto **application** del proyecto y presione el botón **Additional Properties** para desplegar la ventana de propiedades adicionales del proyecto luego seleccione la pestaña **Variable Types** y en el campo **SQLCA** cambie el valor **transaction** por **n_cst_transaction** aplique los cambios y presione el botón **Ok**. De esta forma ya quedará definida la clase transaction del componente.

Paso 3: Crear las Clases Base del Componente

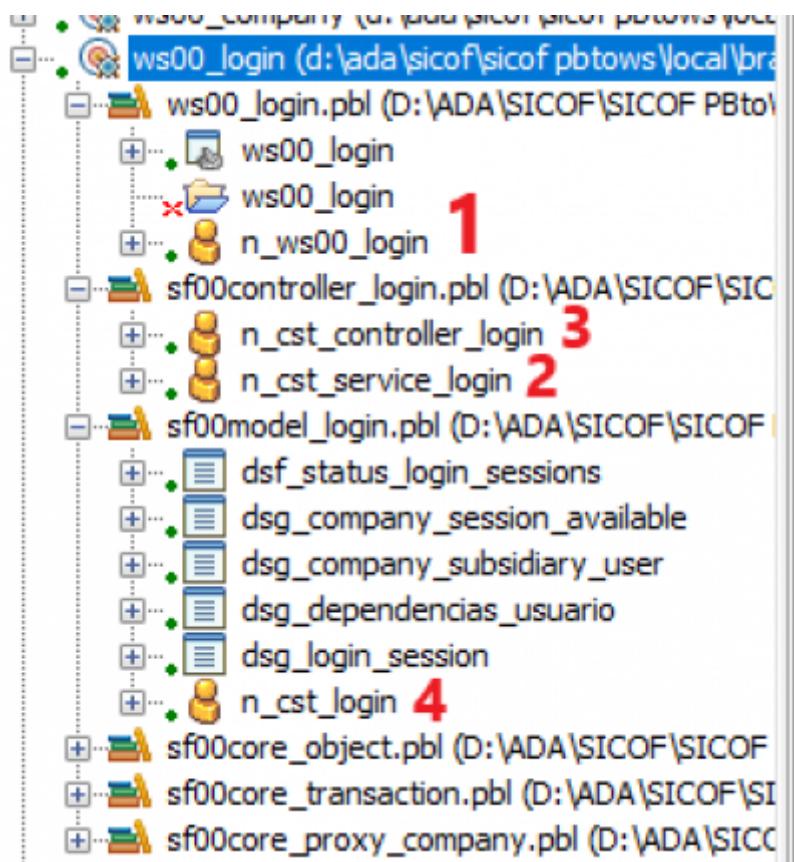
El siguiente paso consiste en definir las clases base de operación del componente las cuales deben ser extendidas (heredadas) del paquete de [Resolución y Orquestación de Servicios](#) de la siguiente forma:

- 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**
- Extender la clase **n_cst_model** para implementar la lógica 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 librería **sf(XX)controller_(xxxxxxxx).pbl** y las clases del modelo ó lógica del negocio deben ser agregadas a la librería **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)



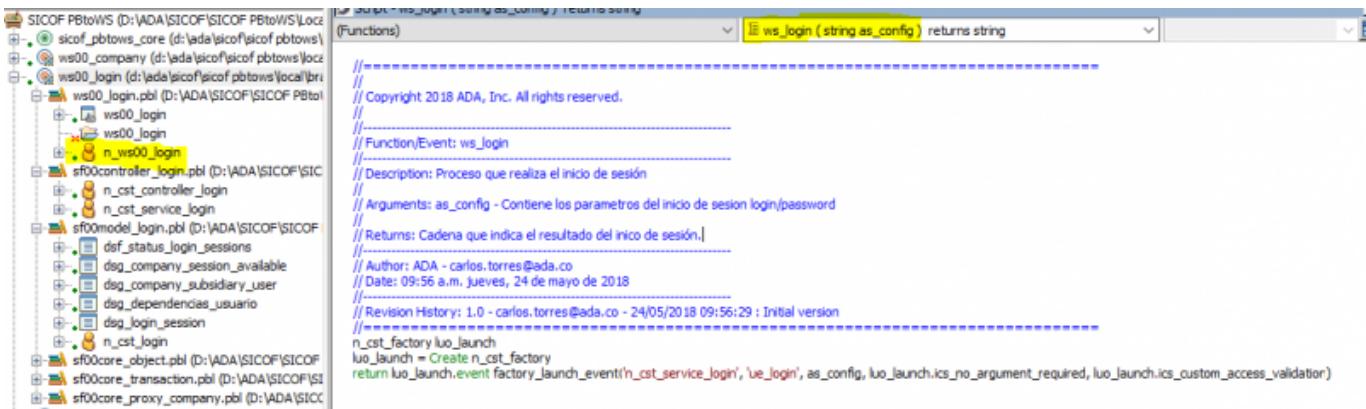
- Clase Lanzadora:** Es generada automáticamente al crear el proyecto en ella se registran los métodos que serán expuestos en el servicio.
- Clase Invocadora:** Es la clase que va a ser invocada por la Clase Lanzadora, crea los controladores y lanza los métodos que inician los procesos.
- Clase Controladora:** Este tipo de clases realizan validaciones, provienen métodos de utilidades e invocan las clases de la lógica del negocio.
- 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 imágenes 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 invocación dinámica por eventos, por lo tanto la clase lanzadora debe invocar el evento `factory_launch_event` y recibe el nombre de la clase invocadora, el evento de invocación, parámetros de configuración formato json, datos de invocación (solo si lo requiere el servicio) y el método de autenticación de servicio.



The screenshot shows the SICOF PBtoWS tool interface. On the left is a tree view of the project structure. On the right is a code editor with the following content:

```

ws_login (string as_config) returns string
-----
// Copyright 2018 ADA, Inc. All rights reserved.
//
// Function/Event: ws_login
//
// Description: Proceso que realiza el inicio de sesión
//
// Arguments: as_config - Contiene los parametros del inicio de sesión login/password
//
// Returns: Cadena que indica el resultado del inicio de sesión.
//
// Author: ADA - carlos.torres@ada.co
// Date: 09:56 a.m. jueves, 24 de mayo de 2018
//
// Revision History: 1.0 - carlos.torres@ada.co - 24/05/2018 09:56:29 : Initial version
-----
n_cst_factory luo_launch
luo_launch = Create n_cst_factory
return luo_launch.event factory_launch_event('n_cst_service_login', 'ue_login', as_config, luo_launch.ics_no_argument_required, luo_launch.ics_custom_access_validation)

```

Consideraciones

- Los parametros de entrada y salida son string en formato json.
- el parametro as_data será requerido dependiendo 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.

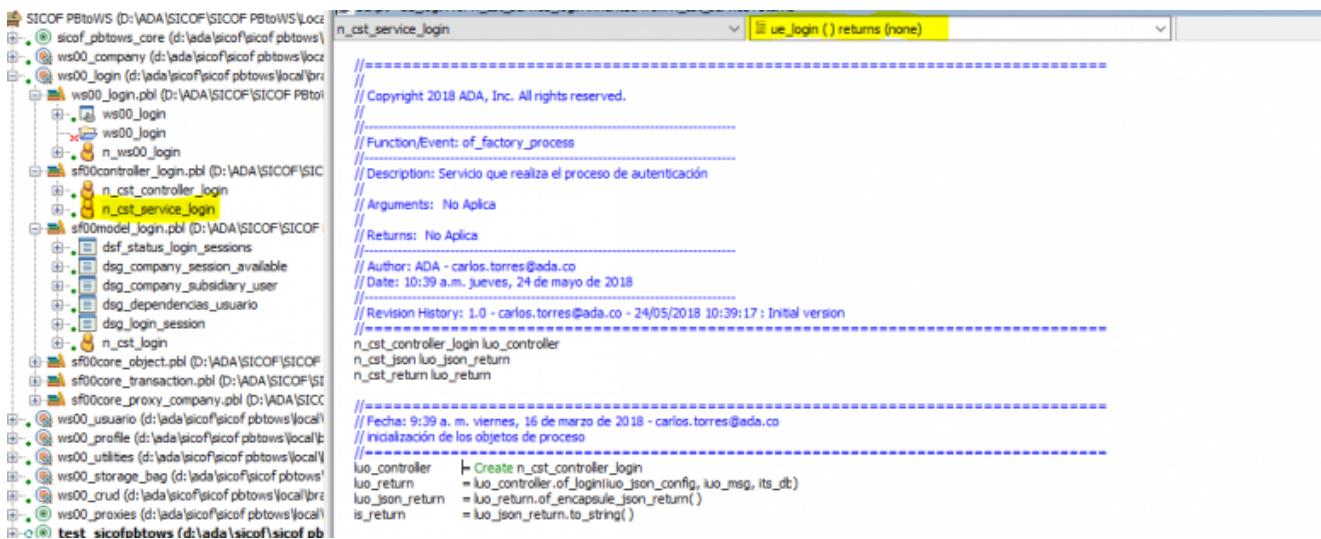
Modelo de implementación: Clase Invocadora

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.



```
=====
// Copyright 2018 ADA, Inc. All rights reserved.
//
// Function/Event: of_factory_process
//
// Description: Servicio que realiza el proceso de autenticación
//
// Arguments: No Aplica
//
// Returns: No Aplica
//
// Author: ADA - carlos.torres@ada.co
// Date: 10:39 a.m. jueves, 24 de mayo de 2018
//
// Revision History: 1.0 - carlos.torres@ada.co - 24/05/2018 10:39:17 : Initial version
//
n_cst_controller_login luo_controller
n_cst_return luo_return

=====
// Fecha: 9:39 a. m. viernes, 16 de marzo de 2018 - carlos.torres@ada.co
// inicialización de los objetos de proceso
//
luo_controller  -> Create n_cst_controller_login
luo_return      = luo_controller.of_login luo_json_config, luo_msg, its_db
luo_json_return = luo_return.of_encapsule_json_return()
is_return       = luo_json_return.to_string()
```

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

From:
<http://wiki.adacsc.co/> - Wiki

Permanent link:
<http://wiki.adacsc.co/doku.php?id=ada:tips:sicoferp:general:pbtows:procesos:guiarapidacomponente&rev=1565636389>

Last update: 2019/08/12 18:59

