Ruta de acceso al repositorio de código fuente: https://github.com/mihhdu/UiPath_REFramework

Objetivos del Enhanced REFramework:

- Ofrecer una estructura clara para el marco de trabajo, tanto desde el punto de vista de la estructura de la carpeta de proyecto como del código.
- Separarla en **capas,** de manera que la capa del marco de trabajo ofrezca un motor fácilmente configurable para ejecutar distintos procesos con datos locales en la cola de trabajo.
- La capa de marco de trabajo debe proporcionar una interfaz común para llamar el código lógico de empresa y facilitar una lógica de recuperación de aplicación de nivel superior y reintento de transacción.
- Facilitar el acceso a los componentes lógicos de empresa en el código.
- Mantener los mismos componentes lógicos de empresa ("InitAllApplications.xaml", "GetSetTransactionData.xaml", "ProcessTransaction.xaml", "CloseAllApplications.xaml", "KillallProcesses.xaml"), permitiendo una migración fluida desde la versión previa.

Registro de cambios:

❖ Alcanzar la modularidad y la reusabilidad definiendo una nueva estructura denominada **Workblock**, diseñada para registrar los datos de tiempo de ejecución del código que se ejecuta en el interior. Reutilizar la estructura para estados de marco de trabajo y código de empresa. (Nota 1)

❖ Intentar alcanzar la separación de intereses dividiendo el marco de trabajo en capas. De este modo, la empresa solo tendría que preocuparse por el código lógico de la empresa. (Nota 2)

❖ Las transiciones de máquina de estados es donde se incrementan los iteradores de transacciones gestionadas del marco de trabajo, el TransactionNumber y el RetryNumber, y se activan marcas del sistema. Ahora se guardan en un diccionario denominado SystemReserved.

❖ SetTransactionStatus.xaml ha desaparecido. La activación real del estado de transacción se realiza en la capa de proceso empresarial.

- Process.xaml se ha movido/renombrado a ProcessLayer\ProcessTransaction.xaml.

- Framework\GetTransactionData.xaml se ha movido/renombrado a ProcessLayer\GetSetTransactionData.xaml.

- Framework\InitAllApplications.xaml se ha movido/renombrado a ProcessLayer\InitAllApplications.xaml.

- Framework\CloseAllApplications.xaml se ha movido/renombrado a ProcessLayer\CloseAllApplications.xaml.

- Framework\KillAllProcesses.xaml se ha movido/renombrado a ProcessLayer\KillAllProcesses.xaml.

- ProcessLayer\CloseAllApplications.xaml, llamado en caso de recuperación de un error del sistema, ahora se llama en la InitLayer.

- Ahora, en caso de excepción, el estado Init se reintenta un número de veces configurable desde el archivo Data\Config.xlsx.

- ProcessLayer\KillAllProcesses.xaml ahora se llama en el try catch del archivo ProcessLayer\CloseAllApplications.xaml, lo que permite un mejor control por parte del desarrollador sin necesidad de tocar la capa de marco de trabajo.

- Cuando TransactionItem es un QueueItem, se usa GetQueueMaxRetries.xaml para consultar el servidor Orchestrator y obtener la máxima información de reintento. Esta función usa la autenticación 2018.1 REST API a través del robot, y no es compatible con las versiones previas. Si es totalmente necesario, se puede hacer que esto funcione con versiones más antiguas usando una solicitud HTTP genérica que requeriría autenticación.

- Ahora, la máquina de estados recuerda las excepciones de aplicación consecutivas que se han producido en el estado de transacción de proceso y cancela el proceso al alcanzar el máximo configurable (Data\Config.xlsx). (P. ej.: usar para evitar que un robot consuma elementos)

- Implementar servicios; cada servicio es un motor de máquina de estados por sí mismo (la misma estructura que el motor principal de Enhanced ReFramework). Consta de un archivo Main.xaml y sus propias carpetas ProcessLayer y Data. La carpeta ServicesLayer ahora guarda los servicios o plantillas de servicio predeterminados. (Nota 4)

- WbLogging.xaml es un nuevo flujo de trabajo que lleva a cabo un registro de Workblock. Cada workblock lo llama dos veces, cuando se ejecuta correctamente y cuando falla.

❖ ListOfDictToDt.xaml es un nuevo flujo de trabajo que se usa para convertir el objeto Audit (el estado de los workblocks) del diccionario a dt y lo guarda en disco en formato csv. Componente del marco de trabajo de pruebas.

❖ Reorganizar Data\Config.xlsx y ofrecer algún tipo de orientación dentro del archivo. (Nota 3)

❖ Se ha agregado una carpeta Workblock Snippet que contiene una plantilla de Workblock. Agregarla a la carpeta de la biblioteca y arrastrarla para crear un workblock en un proyecto nuevo de Enhanced ReFramework.

Notas:

1. Ahora el marco de trabajo está compuesto por **Workblocks**. Cada workblock representa la estructura mínima de ejecución y registro. Está formado por un bloque try-catch. Dentro de la sección Try se encuentra el código que deseamos ejecutar. Tras la ejecución registramos el éxito, reuniendo información de tiempo de ejecución y jerárquica de workblocks **secundarios** y transfiriéndola al Workblock primario. Si se produce una excepción dentro del código, el workblock captura la excepción, registra el fallo, emite la excepción y actúa según las instrucciones de la marca "wbHandleError".

La capa de marco de trabajo en sí tiene 4 Workblocks. Un Workblock maestro al que denominaremos "MainTask". Este workblock tiene tres Workblocks secundarios: "Init", "GetSetData" y "Process".

Los siguientes campos de registro deben registrarse, en un nivel mínimo, en cada workblock:

Se agregan los siguientes campos de registro:

Agregar campos de registro de jerarquía:

+ wbType: nombre del Workblock actual

+ wbParentType: nombre del Workblock primario

+ wbLevel: nivel del Workblock actual (normalmente nivel del Workblock primario + 1)

Agregar campos de registro de ejecución:

+ wbStart: la etiqueta de tiempo del inicio de la ejecución

+ wbDurationSec: duración de la ejecución en segundos

+ wbDurationHrs: duración de la ejecución en horas

+ wbStatus: Si el workblock actual se ejecuta correctamente y todos los workblocks secundarios también, se activa "Successful". Si el workblock actual se ejecuta correctamente y algunos workblocks secundarios no, se activa "Finished with Exceptions". Si el workblock actual falla, se activa "Failed". De lo contrario, "SoftFailed"

+ wbFinalExec: Si el workblock actual se ejecuta correctamente o se encuentra en el último intento y todos los workblocks primarios están también en el último intento, wbFinalExec tiene el valor true; de lo contrario, será false

+ wbParentStart: hora de inicio del wb primario

+ wbHandleError: Si el wb registra en su sección Catch, su wb primario puede ordenarle, mediante la marca wbHandleError, que realice un "Rethrow" de la excepción o bien "Failed" o "SoftFail". Las instrucciones "Failed" y "SoftFail" significan no realizar "Rethrow". La diferencia es que "SoftFail" no provoca que el elemento primario tenga el estado "Finished with Exceptions". Así, el elemento primario puede intentar resolver el problema de elemento secundario de otra manera más adecuada.

+ wbPath: ruta de ejecución del Workblock al workblock actual, reflejando todos los antepasados y el actual

+ wbKey: md5Hash de wbPath+wbStart: clave pseudo única

Reglas que se aplican a los Workblocks:

- wbType debe ser único

- Debe existir únicamente un mensaje de registro que contiene el estado de la ejecución del Workblock al final del Workblock.

- Es recomendable no generar una cantidad excesiva de datos de registros; para ello, evitaremos registrar las ejecuciones correctas de workblocks para cada Workblock individual.

2. Separe los diferentes componentes del marco de trabajo en capas:

1. Capa de marco de trabajo (la representación principal del flujo de datos, una máquina de estados que llama diferentes componentes de código). En esta sección, las partes de código necesaria para la operación del marco de trabajo se encapsulan en una secuencia "System Reserved". Esta capa ofrece iteradores de transacción y mecanismos de recuperación para excepciones de aplicación llamando la estructura definida por el usuario para reinicializar el entorno.

2. Capa de datos (ProcessLayer\GetSetTransactionData.xaml), donde debe realizarse el trabajo con datos transaccionales. Separación clara entre el trabajo con datos y el trabajo con aplicaciones.

3. Capa de proceso empresarial (los flujos de trabajo que deben modificarse para conseguir el comportamiento de proceso deseado: InitAllApplications.xaml, CloseAllApplications.xaml, KillAllProcesses.xaml, ProcessTransaction.xaml)

3.Organice mejor Config.xlsx creando las siguientes hojas adicionales:

a) hoja de introducción (escriba documentación referente a la configuración)

b) hoja de credenciales (escriba aquí las credenciales)

c) hoja de workblock (indique aquí los workblocks que se ejecutan en la capa de marco de trabajo)

d) hoja de tareas (detalles acerca de los servicios en la capa)

4. Implemente una capa de servicios/tareas conectables. Los servicios se encuentran en la carpeta ServiceLayer. Cada tarea tiene su propia carpeta ProcessLayer, con todos los componentes habituales, y su propio archivo Data\Config.xlsx.

Una tarea es un marco de trabajo estándar que actúa como un esclavo que se ejecuta dentro del flujo de trabajo principal. Ahora el marco de trabajo implementa dos servicios del sistema, "FirstRunTask" y "GetDataTask". Ambos están desactivados por defecto. Si están activados:

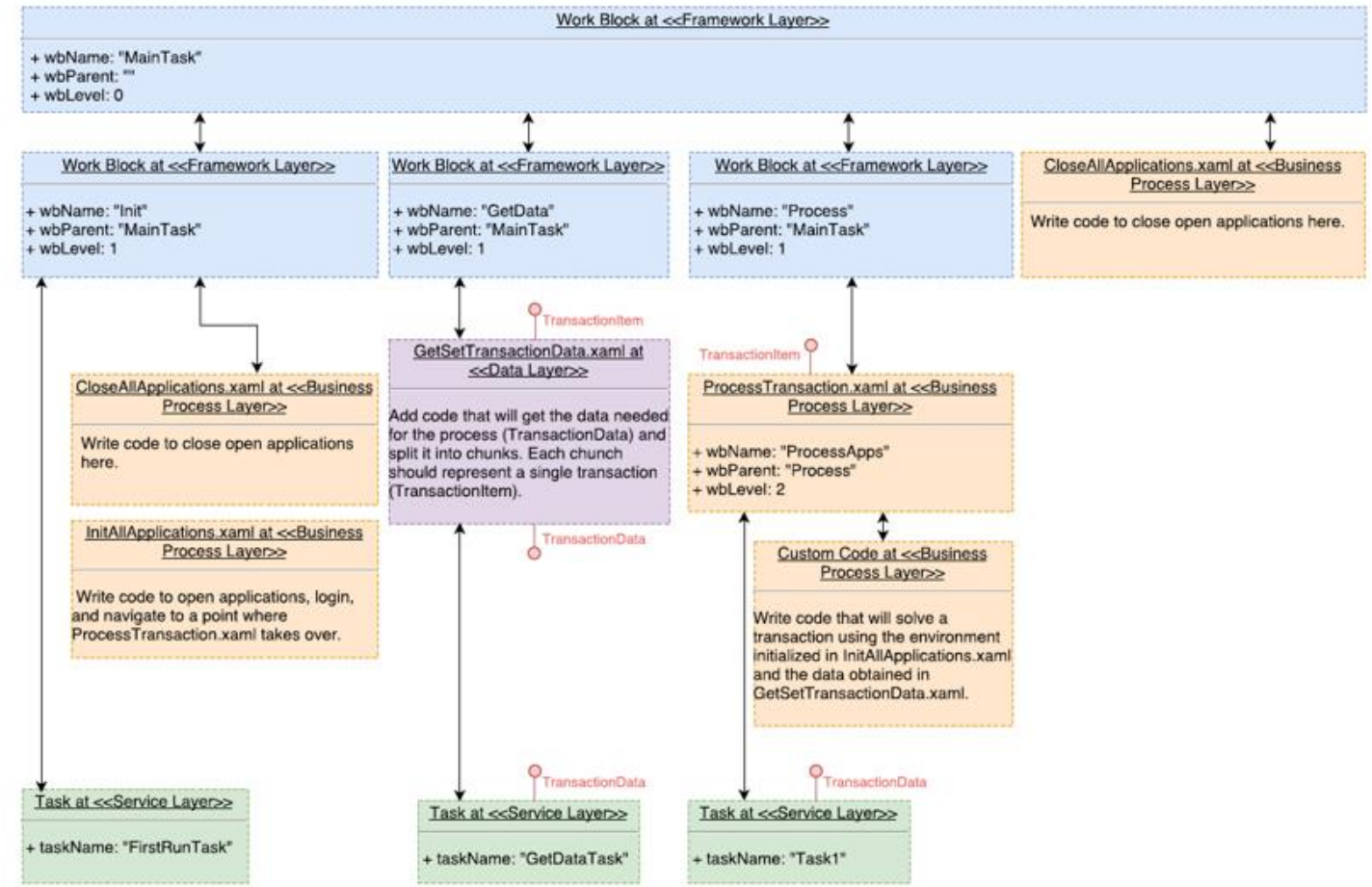
a) FirstRunTask se ejecuta una sola vez, al inicio del proceso. Posible caso de utilización: dispatcher de cola

b) GetDataTask se ejecuta una sola vez, en la primera ejecución del proceso, o en todas las ejecuciones del proceso. Posible caso de utilización: Los datos necesarios para el procesamiento se obtienen de un recurso que puede fallar (por ejemplo, un sitio web). En ese caso, basta con configurar GetDataTask para que obtenga los datos de modo independiente y los emita en la tarea principal.

Incluimos dos ejemplos de servicios más, "Task1" y "Task2", que podrían llamarse en el archivo ProcessLayer\ProcessTransaction.xaml. Esto es útil cuando, dentro de un proceso, tenemos que trabajar con varias aplicaciones independientes. En lugar de implementarlo todo dentro de la sección de proceso, nos limitamos a lanzar un servicio que resuelve la tarea mencionada.

Hay que tener en cuenta que, aunque teóricamente puedan existir servicios dentro de implementaciones de servicios, recomendamos no hacerlo en ningún caso y pensamos que un desarrollo horizontal suele ofrecer mejores resultados.

Default framework workblocks, shown by layers; includes tasks, which are framework copies invoked to solve specific tasks



Framework WorkBlocks UML