Advanced Framework Overview

Contents

Overview

This framework has all the capabilities of REFramework which we were already using. There is some more advancement by which I am trying to provide more flexibility to the developers.

Added Features

Other than the REFramework feature, there are some newly added features.

- You can now easily retry Init state if there is any system exception.
- You can now execute your process 24*7. Now, you have two options in this case firstly, you can execute the process without any delay and secondly, you can execute the process after a certain amount of time.
- You can now send the business reports whenever you want to send in between the process execution.

Exception Handling and Recovery

The Advanced Framework offers a robust exception handling scheme and can automatically recover from failures, update statuses of transactions, and gracefully end the execution in case of unrecoverable exceptions. This feature is closely related to the logging capabilities so that all information about exceptions is properly logged and available for analysis and investigation.

Exceptions that happen during the framework's execution are divided into three categories:

- Business Exceptions: This kind of exception is implemented by the class BusinessRuleException and it should be thrown when there are problems related to rules of the business process being automated. For example, if a process expects to receive an email with an attachment, but the attachment does not exist, the process would not be able to continue. In this case, a developer can use the Throw activity to throw a BusinessRuleException, which indicates that there was a problem that prevented the rules of the process to be followed. Note that BusinessRuleExceptions must be explicitly thrown by the developer of the workflow, and they are not automatically thrown by the framework or activities.

- System Exceptions: If an exception is not related to rules of the process itself, it is considered a system exception. Examples of system exceptions include an activity that timed-out due to a slow network connection or a selector not found because of a browser crash. In this case, it will try to recover.
- Invalid Cast Exception: An InvalidCastException exception is thrown when the conversion of an instance of one type to another type is not supported. For example, attempting to convert a Char value to a DateTime or string value to int value throws an InvalidCastException exception.

Depending on the category of exception, business exception, or system exception the Framework decides whether the transaction should be retried. In the case of business exceptions, the transaction is not automatically retried, since issues related to business rules usually require human intervention. On the other hand, in the case of system exceptions, the error might have been caused by a temporary problem, and retrying the same transaction can make it succeed without human intervention.

The third category of exception is Invalid Cast exception occurs only in Init state if InitErrorRetryCount provided by the user is not an integer value, in that case, it will throw an invalid cast exception and process is stopped gracefully. The role of this exception is only available for the Init State, it will not impact any transactions during processing.

Note that both business exceptions and system exceptions are concepts that also exist in Orchestrator under the names Business Exceptions and Application Exceptions. In fact, if the source of transactions is an Orchestrator queue, then the number of retries in the case of system exceptions can be set directly on Orchestrator. If Orchestrator is not used, the configuration for retries is done in the Config.xlsx file, as mentioned in section *Settings*.

Architecture

The Advanced Framework is implemented as a state machine workflow, which is a kind of workflow that defines states that represent a particular circumstance of the execution. Depending on certain conditions, the execution can transition from one state to another to represent the steps of a process.

States

The states of the Advanced Framework can be seen in Figure 1, and they are detailed as follows:

- Initialization: Check the process is already executed before or not? If the process executed before then wait for the next execution cycle time then it will read the config file again and initialize the applications used in the

process and if it's the first time then without any waiting time it will read the configuration file and initialize applications used in the process. If the initialization is successful, the execution moves to the Get Transaction Data state; in case of system failure, it will retry to recover the exception, and if retry attempts failed then it moves to the End Process state. If a system exception occurs during the processing of a transaction, the framework attempts to recover from the error by closing all applications used and returning to the Initialization state so the applications can be initialized again.

- Get Transaction Data: If you want to do the reporting of the transactions processed by the bot in between the process execution then it will check the reporting time and bases on that it will decide that it's a time for reporting or continue to the process without doing reporting. Get the next transaction to be processed. If there are no data to be processed then execution goes to the Init state and wait for the next process execution time and if any errors occur, the execution goes to the End Process state. If a new transaction is successfully retrieved, it is processed in the Process Transaction state.
- Process Transaction: Process a single transaction. The result of the processing can be *Success*, *Business Exception,* or *System Exception*. In the case of *System Exception*, the processing of the current transaction can be automatically retried. If the result is *Business Exception*, the transaction is skipped, and the framework tries to retrieve a new transaction in the Get Transaction Data state. The execution also returns to the Get Transaction Data state to retrieve a new transaction if the processing of the current one is successful.
- End Process: Finalize the process and close all applications used.

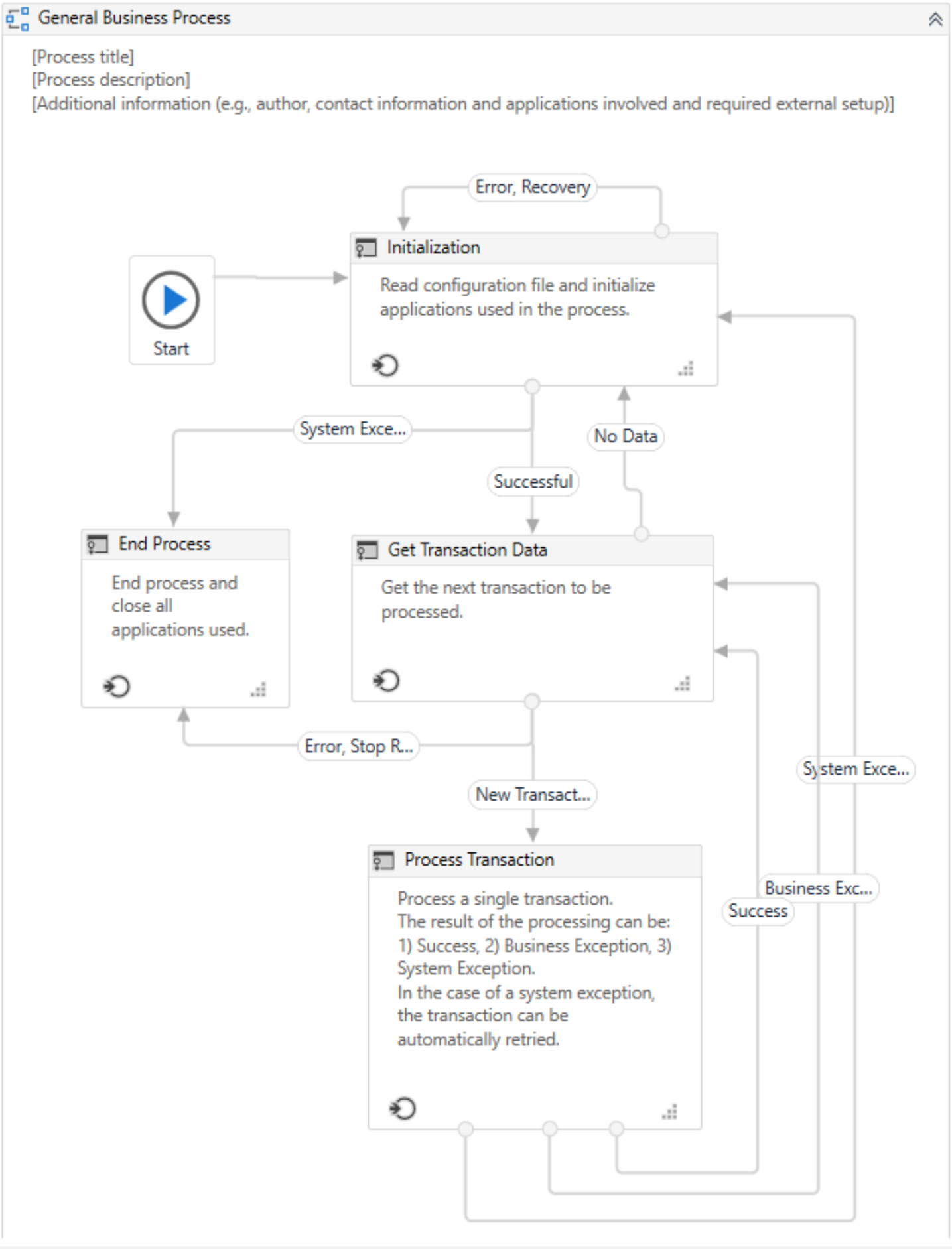

Figure 1 - State Machine with the States of the Advanced Framework.

Table 2 - Workflows Invoked in States.

State	Invoked Workflows
Initialization	InitAllSettings.xaml KillAllProcesses.xaml GettingReportingTime.xaml InitAllApplications.xaml
Get Transaction Data	GetTransactionData.xaml • ValidateReportingTime.xaml ○ SendReport.xaml
Process	Process.xaml SetTransactionStatus.xaml • RetryCurrentTransaction.xaml • TakeScreenshot.xaml • CloseAllApplications.xaml • KillAllProcesses.xaml
End Process	CloseAllApplications.xaml KillAllProcesses.xaml

Each state invokes one or more workflows, which are listed in Table 2 and detailed in section *Workflows*.

Global Variables

Table 3 shows the variables declared in the Main.xaml file and which are passed as arguments to workflows invoked in different states.

One important variable that is passed to almost all workflows invoked in Main.xaml is the Config dictionary. This variable is initialized by the InitAllSettings.xaml workflow in the Initialization state, and it contains all the configuration declared in the Config.xlsx file. Since it is a dictionary, the values in Config can be accessed by its keys, like *Config("Department")* or *Config("System1_URL")*. Note that, although it is present in the Config.xlsx file, the Description of each value is not included in the dictionary.

Table 3 - Shared Variables.

Name	Default Type	Description
TransactionItem	QueueItem	Transaction item to be processed. The type of this variable can be changed to match the transaction type in the process. For example, when processing data from a spreadsheet that is read into a DataTable, this type can be changed to DataRow (refer to section *Practical Example 2: Using Tabular Data* for a sample). In another scenario, if transactions are paths to image files to be processed, this variable's type can be changed to String.
SystemException	Exception	Used during transitions between states to represent exceptions other than BusinessRuleException.
BusinessException	BusinessRuleException	Used during transitions between states and represents a situation that does not conform to the rules of the process being automated.
TransactionNumber	Int32	Sequential counter of transaction items.
Config	Dictionary(Of String, Object)	Dictionary structure to store configuration data of the process (settings, constants and assets).

RetryNumber	Int32	Used to control the number of attempts of retrying the transaction processes in case of system exceptions.
TransactionField1	String	Optionally used to include additional information about the transaction item.
TransactionField2	String	Optionally used to include additional information about the transaction item.
TransactionID	String	Used for information and logging purposes. Ideally, the ID should be unique for each transaction.
TransactionData	DataTable	Used in case transactions are stored in a DataTable, for example, after being retrieved from a spreadsheet.
IsExecutionCompleted	Boolean	Used to check is process execute first time or it was executed before as well?
ReportingTimeCollection	String[]	Used in case Reporting Time Collection is stored in an array of string, for example, after being retrieved from orchestrator or config file.
ReportingCounter	Int32	Counter for iterating all reporting timings.
NextReportingTime	DateTime	Used to get the next reporting time if available.

Main Workflow Arguments

Table 4 shows the arguments declared in the Main.xaml file and which are passed as parameters from the orchestrator.

Name	Default Type	Direction	Description
in_OrchestratorQueueName	String	In	Used to get the queue name from the orchestrator, if required. You can also provide the queue name from the config file under the settings sheet.
in_OrchestratorInitErrorRetry	Int32	In	The number of times a robot should retry Init state during system exceptions from the orchestrator. You can also provide the same from the config file under the constants sheet.
in_OrchestratorProcessPostpone Time	String	In	Used to provide a delay for the next execution cycle of the process. You can also provide the same from the config file under the constants sheet. If it's nothing then the process will continue executing without any delay.
in_OrchestratorReportingTime	String	In	Used to get the reporting times from the orchestrator, if you want to share the reports in between the process execution. You can also provide the same from the config file under the constants sheet.

Table 4 – Main Workflow Arguments.