Quest Journals
Journal of Software Engineering and Simulation
Volume 9 ~ Issue 9 (September 2023) pp: 36-43
ISSN(Online) :2321-3795 ISSN (Print):2321-3809
www.questjournals.org

Research Paper

Frameworks and Best Practices for Building Enterprise-grade Automation in UiPath

Himadeep Movva
Independent Researcher

Abstract:
Configuring enterprise-grade automation requires following best practices, procedures, and frameworks for scalable, efficient, and robust solutions. UiPath provides frameworks for both attended and unattended automation. At the enterprise level, processes that need to be automated aren't linear or iterative but transactional. Hence, the Robotic Enterprise Framework is essential for efficient solution configuration for the transactional business process. In addition, this research study presents the usage of other frameworks alongside the unattended framework, which was officially rolled out by UiPath. Different frameworks, including a Robotic enterprise Automation framework for un-attended automation, an attended automation framework with form and triggers for attended automation, a framework for background processes that run in the background without UI interaction, and an orchestration process framework for human-in-the-loop automation, are presented. A detailed explanation of frameworks, along with which framework should be used according to the use case, is presented. This paper also mentions best practices that need to be implemented, as per the standards, to configure the best solution for the use case at hand.

Keywords: *Robotic Enterprise Framework, human-int-the-loop, Action Center, Global Exception Handler, try-catch, Retry Scope, Queue, JSON, Orchestration processes, Citrix, Cyber-ark, MVC, triggers, and Config.*

I. Introduction:

If a process needs to run once, it's categorized as linear. It doesn't have multiple transactions and may require just one transaction to be executed. An iterative process might have multiple transactions, but all the transactions will be processed in a loop. The whole process breaks if a single transaction fails, leaving other transactions unexecuted. To overcome this challenge, it is recommended that the Robotic Enterprise Framework be used for transactional business processes [1]. This process represents processing individual transactions as distinct sets of records that may need similar or different business logic to be executed based on incoming data. The frameworks in UiPath are built on top of state machines. They are used to transition from one state to another based on conditions, and finally, the process ends at the final state. Robotic Enterprise Framework has four states, namely Init State, Get Transaction State, Process Transaction State, and End State. This framework can even be customized based on the requirement and use case. This is like a template on which developers can build a use case. There are UiPath official frameworks for the orchestration process, which is used to configure human-in-the-loop automation by integrating with Action Center for validations and approvals; background process, which runs process in the background without needing to interact with User Interface; mobile testing framework, which is used to test the app with automated tests; and attended automation framework, which is used to trigger workflows based on forms and desktop triggers, assisting employees and enabling human-robot collaboration. Each framework is tailored to optimize and streamline efficiency across various use cases.

Through this paper, we will review and revisit the best practices for automation development, including types of workflows, automation best practices, integration with source control, and workflow analyzer. Depending on the use case, an automation project either has sequences or flowcharts or a combination of both. A Global Exception Handler isn't mandatory but is good to have, depending on the complexity of automation. It can log all unhandled exceptions in the workflow. While this research study discusses all the features of UiPath that have been mentioned until now, the major focus will be on the state-of-the-art Robotic Enterprise Framework, as most of the use cases in an enterprise are unattended. This framework is used for processing humongous volumes of transactions with robust try-catch mechanisms that can be built on the framework.

1. Robotic Enterprise Framework:

Robotic Enterprise Framework, also known as RE-Framework, is built using state machines. The next states will be executed based on whether the condition is met. Different states in RE-Framework are Init State, Get Transaction State, Process Transaction State, and End State. This framework is used for transactional business processes, and each transaction is executed either successfully or unsuccessfully regardless of the status of other transactions – In other words, dependency on one transaction

i. Init State:

This state is used to initialize the config sheet and open up the required applications used in the process. The config file is of type .xlsx and has three sheets: settings, constants, and assets. The first step is initializing the system error to nothing. If the config is nothing- this is the case for the first run, the file InitAllSettings.xaml will be invoked. By default, this workflow has various variables and arguments. If the Config variable of dictionary time is nothing, then read the config sheet, and if the config is not nothing, which means it is not the first run, the config sheet will not be read, and the else section will be executed. All the applications will be closed during system exception, and the process starts from the beginning. As the config would not be empty by this time, the block will be executed as mentioned. Settings and constants sheets are stored in one dictionary, with name and value as key-value pairs, and assets are stored in another dictionary with the key-value pair as name and value. The values of the config file are used across the automation. It is recommended not to define any variable by default value but instead get it from config as in case of any changes, there's very little maintenance as it needs to be changed only in the config file. There is a separate try-catch for each of these operations where reading the sheets happens. In case of an exception during initialization or Init, the Bot would reach the end state, and there would be nothing to retry. Proper email logging can configured to identify the issue easily in case of an error so that it'd be easy to debug and apply the fix. Next, InItAllApplications.xaml, which has logic configured to open all the required applications for the process, will be invoked. In this workflow, the assets that are required for login are retrieved using the Get Asset activity; applications are opened up, credentials are typed, clicked login, and validated for certain elements to confirm successful login. Although it depends on case to case, there could be scenarios where data required for the process is pulled using API calls during InItAllApplications.xaml workflow. Hence, as a thumb rule, all the settings or data needed for the process are retrieved in this state, and logic can be mapped accordingly. Also, to ensure every application related to the process is closed before starting the process,KillAllWorkflows.xaml, with process names as in arguments, is invoked If the system exception is nothing, the next state will get transaction data, and if it is anything else, the next state will be the end state. Unlike transactions, which can be retried in case of system exception, InIt is not retried, and a hard stop is made after failure at InIt. Still, the Init state can be retried by configuring the retry scope and the number of times it should be retried. However, this is usually not a recommended approach. An error during Init means that there are issues, such as an incorrect configuration in the config file, and may fail again even if it is retried.

ii. Get Transaction State:

In this state, transaction items are retrieved either from the queue or the data table when queues are not used. When queues aren't used, in the case of a system exception, the TransactionNumber variable will not be incremented as it needs to be retried based on the retry count that is set in the constants sheet. Retries would be made until the maximum retry counter is reached, ensuring that failed transactions are retried without skipping to the next ones. However, if a queue is used, the bot automatically adds a new transaction item to be retried, and the Transaction number will be incremented. Bot treats it as another transaction. The first step of this state is Check Stop Signal activity. If a stop command is passed in the orchestrator while the job is running, this activity's variable ShouldStop will be assigned as True. If so, TransactionItem will be assigned as nothing, and the process will stop. The difference between Kill and Stop is that while kill ends the job abruptly, stop command ends the process gracefully after closing applications. If the stop button is not hit, the bot goes to the else block to retrieve the transaction items using GetTransactionData.xaml workflow.

GetTransactionData.xaml is used to fetch queue items with the status 'new'. Try-catch is used within the Retry scope activity. If getting a queue item fails, the flow will go to the catch block, and the exception is ReThrown so that the Bot logs the message stating that the queue item couldn't be retrieved. Once out_TransactionItem is not nothing 'Then condition of IF statement will be executed to fetch further details about the queue item. Suppose queue items are encrypted; each transaction can be fetched during the Get Transaction Data state and decrypted during the Process state. While the Transaction number and config dictionary are 'in' arguments to this workflow, the transaction-related information is stored as an 'out' argument. If there is an error in the try part when having to get transaction data, the transaction item will be assigned as nothing in the catch block to retrieve the next transaction item from the queue. If the transaction item is nothing, the state machine flow will go to the InIt state, or if the transaction item is not nothing, the pass will go to the Process Transaction state. In short, in case of exception, Init will be executed and process transaction will be executed in case of successful retrieval of queue item.

iii. Process Transaction State:

Once the queue item is retrieved for a particular transaction, the flow will go to the Process state. In this state, all the logic related to the core business logic will be executed. Each transaction can have three statuses in this state: success, business exception, and system exception. As a first step in the process, the business exception variable will be assigned as nothing, and then process.xaml will be invoked. This workflow has two input arguments: config dictionary and transaction item. By configuring robust try-catch mechanisms inside, business logic will be configured. If a business exception occurs while processing the current transaction, it will be marked as failed and will not be retried. However, in case of a system exception, the transaction will be retried. After processing the transaction, InvokeSetTransactionStatus.xaml will be invoked, and the transaction status will be marked accordingly. If a business exception is nothing and the system exception is nothing, the Success path will be executed, marking the transaction as successful. In case of failure in the try block while updating transaction status, the catch block will be executed, and an exception will be rethrown, stating that transaction status cannot be updated. Also, log fields can be used to add certain fields to the logs, ensuring essential information is stored that may be helpful in case of debugging and monitoring. These log fields are available only up to the execution and will be cleared thereafter, helping filter log data when Kibana is configured. If a business exception is not nothing, the flow to mark the transaction as a business exception will be executed, or else if the system is not nothing and business exception is nothing, flow to mark the transaction as system exception will be executed. In the case of the queue, the transaction number will be incremented, and in the case of a non-queue transaction, the retry number will be incremented. If the transaction is either a success or a business exception, Increment the transaction number to get the next transaction to be processed, Reset the counter of retries to allow the next transaction to be retried the correct number of times, and Reset the counter of consecutive system exceptions.

If SystemException Is Nothing And BusinessException is Nothing, the bot will transition from Process State to Get Transaction State. Also, if BusinessException IsNot Nothing, the bot will transition to Get Transaction Data state.If SystemException IsNot Nothing, the bot will go to initialization. When queue items are used, after working the transaction, status of the processed queue item must be set. This is done using set transaction status workflow.If the Business Exception is nothing and the system exception is nothing, it means that the transaction is a success, and the transaction number will be incremented after marking it as a success.n If a business exception is nothing, the transaction will be marked as a business exception, and the transaction number will be incremented. If the system exception is not nothing in the case of a queue item, the transaction number will be incremented, and If the Maximum Retry Number is NOT met, the Orchestrator will automatically create a retry item with an incremented retry count.If max rerty number is already reached, bot will mark the trassctiona s failed and will not retry.If System exception is not nothing in case of non queue item, retry number will be incremented and checked aginst maximum retry number and if condition is met, it will be retried as transaction number will not be incremented. Transaction number will be incremented only after maximum retries are reached and will be checked against the counter maxRetryNumber variable.

iv. End State:

If there aren't any transactions to be processed, the bot will transition from the get transaction state to the End State. Also, in case when SystemException isNot Nothing, the bot will go from InIt state to End State. Also, when maximum consecutive exceptions are reached, the bot will break processing during the init state and navigate to the End process state. As discussed in this paper, if a system exception occurs during InIt, the bot will go to the End State. This is the final state of the Robotic Enterprise Framework. In this state, CloseAllApplications.xaml workflow will be invoked, logging out and closing all opened applications. This helps to gracefully close every application for a smooth opening of applications in the next run.

2. Attended Automation Framework:

The attended automation framework in UiPath is used to configure Bots that run alongside human actions. The attended agents can be triggered based on an event, such as the opening of an application or the starting of a process from the UiPath assistant. The attended bot runs on a user's machine and is designed to assist humans. Without human input or trigger, the attended bot couldn't start. This process allows for real-time exception handling by the user when an error occurs, as based on the user's action after the error, the next flow will be executed. Along with human-bot collaboration, this framework can also include action center integration and assigning user tasks for review. The use cases that qualify for this type of automation are front-office automation, scenarios that may require human judgment, and scenarios that require human-robot collaboration. If users need to work when the bot runs without blocking the users of their actions by the bot, the best configuration is to enable the PIP mode (Picture In Picture). Enabling PIP mode opens a new session, and the bot runs the process in that session while users can still perform the required work on their machines. Initialize Config checks and initializes config dictionary. The next state is User Input Form and Process, which displays configured forms and does further process. The Results form state displays results, which can be errors or successes, to the user. The final state would be the End State.

3. Orchestration Framework:

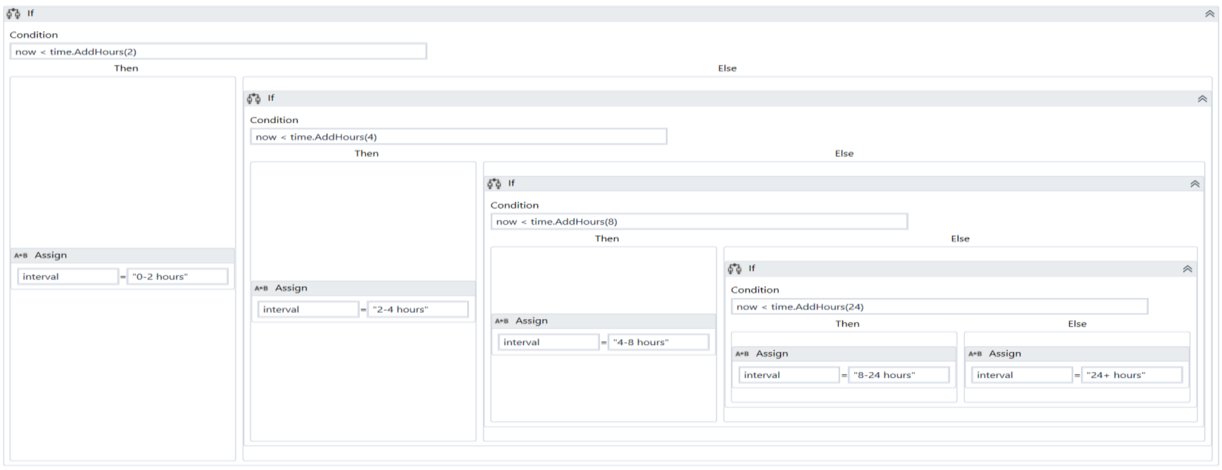
Orchestration processes are long-running workflows that support persistence and run in combination with human interaction. These processes can run in unattended mode also, making them efficient in case when long wait times may be required for a task to start after user action. Human resolution or action may be required before resuming the job again. The Robot, instead of waiting for the user input, gets freed of the job and would be available for the other process to use the robot. Once the task is completed, the available robot is automatically assigned to resume the job. These workflows may orchestrate background processes that may not need UI interaction, running in session 0, a user session process, or human in the loop process, letting the users interact through forms by using create form task and wait form task and resume that need to be completed in the orchestrator. The official template for the orchestration process is available from templates in the studio. Any process can be customized to an orchestration process by navigating to settings and turning on the persistence feature. It is recommended not to change the JSON as it has information related to persistence.

Add Queue Item and Get Reference activity is used to add a queue item and get a reference ID. The input of the Wait for Queue Item and Resume activity would be the output of the Add Queue Item and Get Reference activity. Both activities will be used in conjunction in case when manual review is needed. The Wait for Queue Item and Resume activity would be in hold status until a specific queue item is executed. To determine which queue item should be assigned for human review, business logic, and checks can be configured on the incoming data. If a particular record passes all checks, manual review may not be needed. There could also be a case where a job/process shouldn't run until a certain job/process is completed. In this case, use Start Job and Get Reference activity to invoke another process and get the arguments. Wait for Job and Resume activity will suspend the target job until the reference job is executed. This uses Job Object as input, which would be the output of the previous activity. Below are some of the pointers that would be helpful in building orchestration processes.

- Long-running activities that are part of the persistence activities package can only be used in main.xaml. These activities cannot be used inside invoked workflows because only Main.xaml supports persistence (saving the state of the bot before suspending).
- Reusable components developed within the orchestration process can only be used in another orchestration process. Only in the case of long-running workflows will queue items still be in progress, even after 24 hours, until the transaction status is set through the workflow.
- Delay and retry scope may not work as intended in the main workflow of the orchestration process. If they need to be used, use no persist scope activity.

4. Best Practices for UiPath Development:

Sequences are linear combinations of activities where, in an application, certain clicks or steps need to be performed. Flowcharts, unlike sequences, allow for multiple branching using conditions. The choice between a sequence and a flowchart is dependent on the use case at hand. Choosing the most appropriate branching and conditions has a lot of impact on the visual structure of the project [2]. The IF condition splits a workflow vertically and is ideal for short-branching workflows [3]. However, if multiple conditions are nested under IF, it will be very tough to understand. Hence, for the readability of code and for best practices, multiple branching conditions that do not fit the size of the screen can be avoided. [4] Sometimes, if there is a requirement to use multiple constructs of If and else if, instead of using nested looping, it is recommended to use Switch Case activity, which provides efficient separation of code for each case. [5]



now < time.AddHours(2)
interval = "0-2 hours"
now < time.AddHours(4)
interval = "2-4 hours"
now < time.AddHours(8)
interval = "4-8 hours"
now < time.AddHours(24)
interval = "8-24 hours"
interval = "24+ hours"

Figure 1- The nested conditions of if else are not recommended beyond the screen's visibility

Global Exception Handler can be used per one project. [6] This helps to catch any exception that would be otherwise unhandled elsewhere in the project. There can only be one Global Exception Handler in a project. There will be two arguments, which are errorinfo as in argument and error result as out argument. For the proper functioning of the handler, the arguments shouldn't be changed. The error action inside the exception handler can be any of the 4 types: ErrorAction.Continue re-throws the exception, ErrorAction.Ignore ignores the exception, ErrorAction.Retry retires the activity where the failure occurred and ErrorAction.Abort stops the process after running the handler. When using a handler, if a try-catch is being used, ensure that all the activities under the try loop are captured in a sequence, or else the exception handler won't work. When the handler is executed, it logs the name of the activity that failed, retries the activity 3 times, and then execution will be aborted.

Data within the workflows are passed as variables or arguments. Variables are recommended to be defined within a certain scope instead of defined at the highest level, and the recommendation is that variables be in the innermost loop to reduce clutter. Duplicate naming for the variables should be avoided, and even if needed, they should be separated by the scope. In the case of the same variable name, while processing, the variable defined in the innermost loop takes priority. Default values in variables or arguments aren't recommended, and instead, retrieve them by defining them in the config file and retrieving them through the config dictionary. Meaningful naming conventions should be assigned to variables and arguments. Argument names should have a prefix such as in or out to understand the type of argument. Ex: In_fileName , and out_textResult. It is suggested to ensure that, except main, the name of any workflow matches the action that the workflow does. Annotations and/or comments activities are very useful and highly recommended for each activity, describing the activity and providing readability and ease of understanding when another developer looks at the code during maintenance.

There are different types of indicating mechanisms while performing clicks. If activity needs to run in the background, simulated click or send Windows messages can be used. However, the outcome is always dependent on the target application. For some desktop applications, these two options may not work. In those cases, if left empty, the default option, which is hardware events, will be used, and it can work 100%, excluding the case where the application is in the background. Configuring selectors for the Bot to interact with the UI elements is a crucial step for sustainable automation. This configuration includes ensuring attributes don't have volatile values, variable parts are marked with wild cards, and idx attributes are avoided in the selectors. If an attribute has just the wild card, that attribute can be removed. Image-based should only be used as a last resort if nothing else works to identify UI elements on the screen. As image matching requires elements to be fully visible on the screen, ensure that the elements are the same during the development and runtime. Selecting an image that may resemble the configured image match leads to a false positive and may cause unintended consequences, as business logic performed by the bot would be wrong. Ensure that the resolution in the production environment is greater than or equal to the resolution during development. This needs utmost care when the application runs in Citrix.

Splitting the project into smaller workflows is the best practice in project organization. Dedicated workflows provide code readability and testing of individual components. It also allows the team members to collaborate with each other and assign workflows to members of a team. If workflows are large, this wouldn't be possible. It is recommended that workflow files should be less than 5MB, and workflows that are of size greater than 10MB aren't supported [7]. A careful and informed decision needs to be taken when choosing what to use between a sequence and a flowchart. Developing complex logic within a sequence creates a maze kind of container. It would be hard to read and follow, incurring heavy losses in terms of time for development during maintenance. Conversely, using UI interaction-based activities in the flowchart makes it more difficult to build and maintain. As a best practice, integrate a working directory with a centralized repository that stores source code. This can be any tool among GIT, TFS, SVN, or any standard repository. Another important aspect is to avoid hard-coded values or links in the automation. Instead, define them in the config sheet so that the bot gets the values from the config sheet or from assets. Isolated setting must be set to True in the invoke workflow activity, assigning separate space for the invoked workflow. If set to false, the memory assigned to the main will be used to run the invoke workflow. The entire process won't stop if isolated is set to true, and if the invoke workflow fails, the main process will still continue. A repetitive process might have thousands of queue items, and if workflows are not isolated, memory usage builds up, increasing the likelihood of a crash. By enabling the isolated option, each workflow runs independently and frees up memory after use. No credential should be defined in the workflow and should be loaded from Windows Vault, Cyber-Ark, or assets. Also, ensure that passwords are not logged anywhere in the process. During development, we might often require using the same code in different places. In this case, creating and using a library component wherever required in the project would be the best practice.

Deciding between attended and unattended automation is crucial before starting the development of a project, as automation frameworks are different for each type of process. Converting to another type of automation would be difficult once the development is in progress [8]. Not all processes that may seem to require an attended framework actually require one. For example, there could be a scenario in which data produced as output by the bot needs to be reviewed by the user, and then another set of actions needs to be triggered. In this case, two processes can be created instead of the attended framework. After the execution of one process and the completion of the review of data by humans, the other process can be triggered by making the output of one process input of another process. In this case, although there is human intervention, each process can be triggered in an unattended mode. Generally, an unattended process gives high ROI as in the case of attended automation, the user may be blocked, or the bot needs to wait until the human actions are completed. Also, attended automation may run only during normal business hours as they require human actions or validations. Documenting the steps and requirements of the process is essential and is captured using the PDD-Process Definition Document.

UiPath project has two layers: the logic layer that contains all the steps related to the core business process and the GUI layer that encompasses all the UI interaction activities. To avoid issues regarding maintainability and reusability, separate the UI layer by creating a series of libraries with reusable components for GUI interaction. This way , we can create a versatile and robust library that is easy to maintain and highly reusable. Object repository, introduced from version 2020.10, is a way to achieve this reusability. In software development methodology, there are 3 layers: the presentation layer, domain layer, and Persistence layer. In RPA, the presentation layer would be the UI interaction layer that has all the activities related to UI interaction across the applications used in the process. The domain layer would be the core business logic layer in the process. However, the persistence layer may not be applicable to RPA as it is the layer where the logic related to storing the data is stored. While designing RPA workflows, it is recommended to take into account the single responsibility principle. [9] This means that an XAML file should do only one thing, meaning that code must be modularized even if it is layered. Hence, we can have multiple modules for each layer. This principle is associated with low coupling: there should be the least dependency between modules and high cohesion: actions associated with the same application should be kept in the same module. The Object Repository is organized hierarchically, with each node standing in for a screen or element at the application level. The following is the structure as mentioned in Figure 2- UI descriptors in a project: [10]
UI Application: Targeted application, and there can be multiple versions of the same target, with each having different screens
Screen: This groups several elements that are required by the bot to interact with
UI Element: Contains the selectors, full or partial, of the elements that are in the target screen

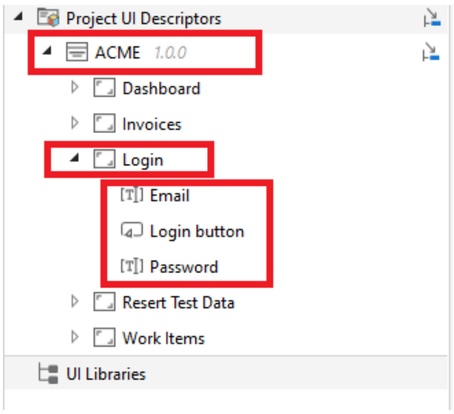


Figure 2- UI descriptors in a project

Object repository is highly efficient in handling UI-related logic. By creating a separate library component for the UI layer for each project, the corresponding library can be used, and the business logic layer, also known as the domain or logic layer, needs to be built for each automation Figure 3- MVC logic in RPA. This allows us to create projects with just the automation logic and no user interface interactions. If there are any changes to the UI, instead of modifying all the projects, just update the library package corresponding to the change and update the package in each project. An older version of the library can be readily retrieved if it needs to be used again because Orchestrator can keep many versions of the same library. In this manner, a single automation project can quickly transition between several versions of the same library and use different versions of the same library in multiple projects. Using this methodology, the main automation process, which is the controller, will have the business process logic, and the UI layer will be the one in which all the interactions with applications happen.

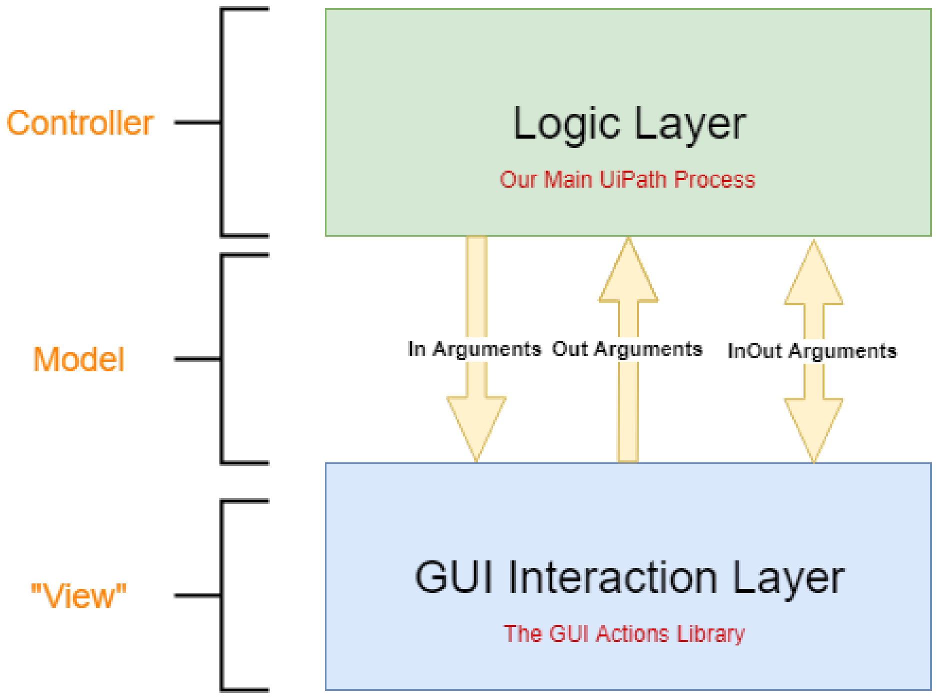


Figure 3- MVC logic in RPA

II. Conclusion:

For enterprise-scale projects of large scale and high volume, Robotic Enterprise Framework provides a head start and the best template to start with. Later on, as the development progresses, additional features and tweaks can be made to this framework.UiPath also provides official frameworks to develop attended processes, orchestration processes, mobile testing processes [11] and many other processes. Using template frameworks gives us a head start in development as the frameworks would have already arealdy tested and used by the subject matter experts. It’s like using thoroughly analyzed, well-tested, and highly recommended templates. This paper also discussed some of the best practices that must be taken care of in development. Camel casing is recommended for variable naming convention and ensures that the naming exactly depends on the role of the variable used. Also, ensure that automation is developed in several small components that do a specific function for each workflow. The naming is recommended to follow action verb notation, conveying the action that the workflow does. The best approach is to create and/or use as many reusable components as possible during the development. Also, the best practice of defining UI descriptors as libraries, importing them as libraries into the projects, and using them in the projects is discussed. The best practices can also be enforced across the projects through the usage of Analyze Project and not letting the project publish until errors and/or warnings are cleared. This research study also discusses the saving of time and increase in efficiency during the maintenance phase by using modern design experience, reusable components, and UI Descriptors library components. All in all, following these guidelines and incorporating best practices, including the ones discussed in this research paper, can make automation robust, least susceptible to errors, and highly efficient with a minimal error rate.

References

[1] A. Tripathi, Learning Robotic Process Automation: Create Software robots and automate business processes with the leading RPA tool–UiPath, Birmingham: Packt, 2018.

[2] [Online]. Available: https://docs.uipath.com/studio/standalone/2022.10/user-guide/modern-design-experience. [Accessed August 2023].

[3] [Online]. Available: https://docs.uipath.com/studio/standalone/2022.10/user-guide/workflow-design. [Accessed August 2023].

[4] [Online]. Available: https://docs.uipath.com/studio/standalone/2023.10/user-guide/workflow-design. [Accessed August 2023].

[5] [Online]. Available: https://docs.uipath.com/studio/standalone/2022.10/user-guide/st-mrd-009. [Accessed August 2023].

[6] [Online]. Available: https://docs.uipath.com/studio/standalone/2022.10/user-guide/global-exception-handler. [Accessed August 2023].

[7] [Online]. Available: https://docs.uipath.com/studio/standalone/2022.10/user-guide/project-organization. [Accessed May 2023].

[8] [Online]. Available: https://docs.uipath.com/studio/standalone/2022.10/user-guide/automation-lifecycle. [Accessed May 2023].

[9] [Online]. Available: https://docs.uipath.com/studio/standalone/2023.10/user-guide/reusable-components. [Accessed May 2023].

[10] [Online]. Available: https://docs.uipath.com/studio/standalone/2022.10/user-guide/reusable-components. [Accessed May 2023].

[11] D.Andrade, "Challenges of automated software testing with robotic process automation RPA-A comparative analysis of UiPath and automation anywhere," *International Journal of Intelligent Computing Research (IJICR),* vol. 11, no. 1, 2020.