Python Plugin and Low-Coding Software Development Platform

How the next generation RPA platform addresses the issues of high-cost software automation deployment

Shige Sato, CEO
Jerry Chae, Vice President, Engineering

ARGOS LABS Incorporated
www.argos-labs.com
info@argos-labs.com

Executive Summary

Software robots for business processes (bots) or Robotic Process Automation (RPA), are viewed as an essential element in digital transformation efforts by enterprises worldwide, along with other technologies such as artificial intelligence (AI) and machine learning (ML). Bots are expected to relieve humans from tedious and repetitive manual tasks and to boost productivity for companies.

However, the market and technology are still young and in the developing phase. The need for extensive consultation has limited the availability of RPA to only Fortune 500-class enterprises. Further, today's RPA solutions require specially trained engineers to develop, deploy, and maintain the bots. These limitations result in a high cost of RPA deployment and in failures when viewed from an ROI standpoint. While many RPA vendors strive to advance their technology to overcome these issues, ARGOS LABS has taken a step forward and released a new type of RPA platform called ARGOS RPA+.

The ARGOS RPA+ platform is based on a low-to-zero coding philosophy. It is entirely GUI-driven when building bots. It supports both auto-recording and building block approaches. The tools are intuitive to the extent that anyone who can make a PowerPoint presentation can build bots without much difficulty. Furthermore, ARGOS RPA+ includes a set of tools that automatically converts Python programs into building blocks that users can use to develop their bots.

Python is known to be the fastest-growing programming language and is broadly used for AI/ML as well as automation projects. With this new ARGOS RPA+ Python Plugin architecture, organizations worldwide can immediately take advantage of the enormous number of Python resources that are already available. It now becomes possible to easily implement the modular approach provided by ARGOS RPA+ to create bots. This system also offers substantial opportunities to the entire Python community worldwide. A Python repository maintained by ARGOS LABS will form the basis for a marketplace that makes it possible to buy and sell bot components between Python coders and bot builders.

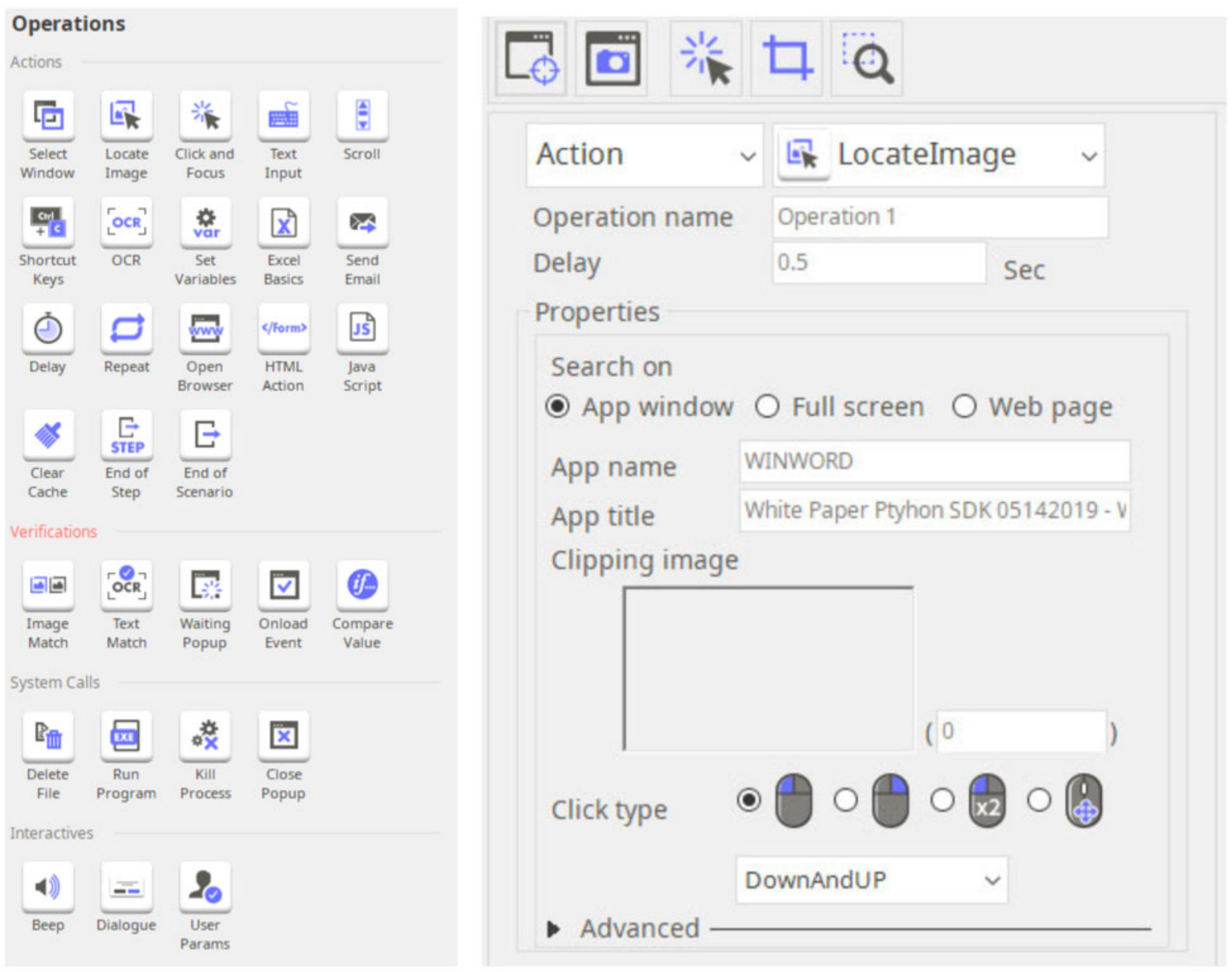
This paper describes how the ARGOS RPA+ Python Plugin architecture has been implemented and how it defines ARGOS RPA+ as not just an RPA platform, but as another evolution in the history of the software programming platform. Essentially, RPA is an alternative to conventional system development methodologies. The concept of RPA emerged because it promised ease of deployment and a return on investment (ROI) substantially better than those of the existing

traditional methods for system development. However, most RPA projects so far have fallen short of their promises. Two typical problems leading to failure are a substantial requirement for consultation work, even before starting to build bots, and a burdensome requirement for specially trained professionals to build and maintain bots. ARGOS RPA+ overcomes the limitations of traditional RPA approaches as described in this paper.

1. ARGOS RPA+ and Zero-Coding Platform Architecture

There are two main difficulties associated with RPA methodologies. The first issue, the need for massive consultations, seems to be addressed by AI. Thanks to technological advancement, many start-ups around the world are proposing AI-based "process discovery automation" to circumvent this limitation. The second difficulty is the need for trained professionals to build and maintain bots. This issue is critical as it directly impacts ROI. As specialists are more expensive than generalists, the ARGOS RPA+ platform has been designed to enable generalists, or even non-IT workers, to build and maintain bots. In this way, the ARGOS RPA+ platform addresses this second limitation.

A screenshot of an embodiment of the ARGOS RPA+ platform is shown in Figure 1.



Action: LocateImage

Operation name: Operation 1

Delay: 0.5 Sec

Properties

Search on: [✓ App window] Full screen Web page

App name: WINWORD

App title: White Paper Ptyhon SDK 05142019 - V

Clipping image (0)

Click type

DownAndUP

Advanced

Figure 1. Screenshot of an embodiment of ARGOS RPA+

Figure 1 depicts building blocks and parameter settings associated with an ARGOS RPA+ screen. When developing a bot using ARGOS RPA+, users choose these building blocks and set the parameters. In an embodiment, the system provides 31 building blocks, and some very complex bots can be built simply by combining these building blocks. To implement more complex designs, a Python plugin architecture provides additional building blocks as shown in Figure 2.

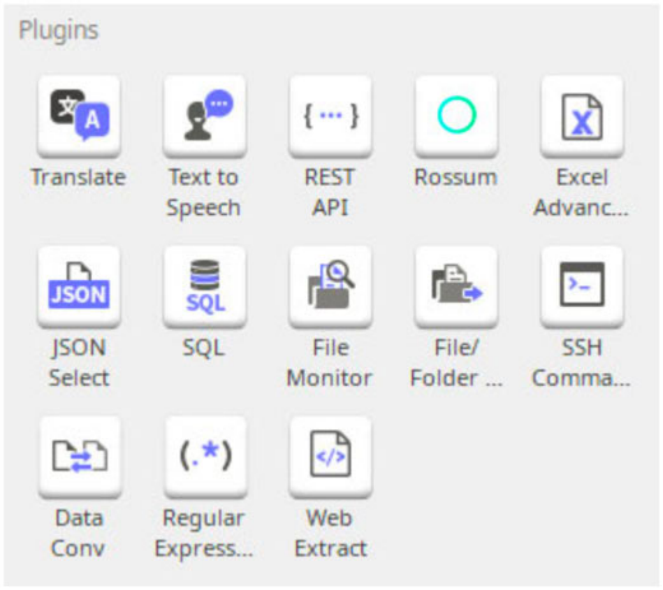


Figure 2. Screenshot showing plugin module of ARGOS RPA+

As shown in Figure 2, the plugin module offers more advanced options, such as JSON and SQL commands, to a developer. ARGOS RPA+ provides 31 basic modules; other, more complex features are implemented as plugins. ARGOS users can choose from a repository (marketplace) maintained by ARGOS LABS, the developer of ARGOS RPA+. The repository includes a library of plugins that allow users to automate their specific business processes. The basic architecture of ARGOS is designed so that plugin modules can be directly converted from Python programs. Python is a popular programming language used to develop automation modules as well as for AI/ML systems. A large number of Python modules are available from public repositories. Consequently, ARGOS bot builders can instantaneously gain access to tens of thousands of functionalities. They can also choose and add only the tools they need. These tools can be very specific to functions and target systems. The flexibility provided to bot builders allows those with unique requirements to create a diverse set of functional plugins. This allows ARGOS RPA+ to function with a core set of building blocks, with additional functionality added using plugins. Based on this architecture, the ARGOS Python Plugin is a more efficient approach by far when compared to solutions with 350+ or 400+ tools.

ARGOS RPA+ can be adapted to function on different operating systems such as Windows, macOS, different flavors of Linux, etc., on a single bot development platform. Mobile device operating systems such as Android and iOS can also be supported. A Python plugin associated with ARGOS RPA+, described subsequently, provides additional functionality to ARGOS RPA+. This system incorporates sufficient abstraction to use different kinds of application programming interfaces (APIs), thereby allowing the architecture to leverage an ever-growing community of software developers from different markets.

2. ARGOS RPA+ Python Plugin Architectural Overview

The ARGOS RPA+ Python Plugin architecture includes a Python coder. As shown in Figure 3 below, a Python coder first aligns the format of the code according to the ARGOS SDK, which comes with templates, sample codes, utilities, and documentation. Then, by uploading it to ARGOS Python Operation Tools (POT), a backstage-like private repository for prequalified Python modules, the plugin modules are automatically generated after checking for functionalities and security.

The plugin module and the original Python code are stored in the marketplace and the repository respectively. Then, as the bot-builder searches for the building block that serves his/her bot requirements for a specific project, he/she can just visit the ARGOS marketplace, choose the ideal building block, and bring it in as a part of the ARGOS Scenario Studio (STU) which is used to build bots with zero-coding technology.

Finally, when the bot has been built, the bot-builder dispatches it to the ARGOS Process Automation Module (PAM). When PAM sees the bot containing the Python plugin, she goes to the ARGOS repository and retrieves the original Python code to execute it. PAM has the Python interpreter built in.

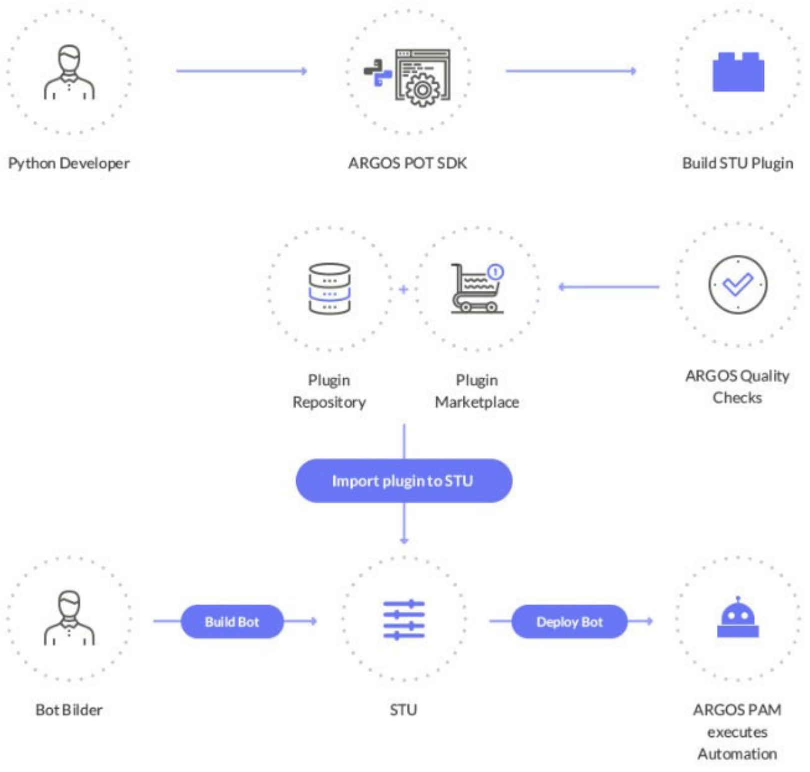


Figure 3. Flow diagram showing creation and implementation of a plugin module

Some examples of plugins that can be included in a bot design are

1) Online tools such as Rossum (https://rossum.ai/) can be integrated into a bot by writing simple Python code to call their APIs. In one instantiation, integration of this design takes approximately three hours to accomplish.
2) Publicly available tools like Google Translate (https://translate.google.com/), which is written in Python, can be integrated into a bot just by processing through the SDK and POT.
3) Existing Python assets inside an organization can all be included as building blocks for bots by the SDK and POT. These building blocks (plugins) can be stored in either private or public repositories and become available to those who have access to them.

The SDK and POT architecture can be implemented using languages other than Python, such as JavaScript (which possesses a similar implementation to Python using node.js and npm), GO (useful for both backend logic as well as command line interface [CLI] utilities such as docker), C#, and so on. In order to implement plugins in languages other than Python, the associated Python code is ported into the desired target language for the SDK and POT utility. The plugin repository is correspondingly extended to support the plugins generated using these programming languages. In this way, the functionality of ARGOS RPA+ can be extended to support a variety of different languages.

3. Workflow to Build an ARGOS Python Plugin

Figure 4 presents a flow diagram that depicts a plugin qualification process. Python code that fits the format requirements of ARGOS SDK will be tested by the coder first. Then, after submission to ARGOS POT, the code will be tested again by ARGOS. This double testing also includes checking the code for compliance with STU specifications.

As shown in the flow diagram in Figure 4, the Python coder can choose the plugin to be listed for either public or private usages. If chosen to be private, additional tests will be omitted, and the plugin becomes instantaneously available to the Python coder from his/her private repository. If the developer chooses to make the plugin publicly available, additional qualification processes are performed for security violations, and then the plugin is eventually listed in the official ARGOS repository.

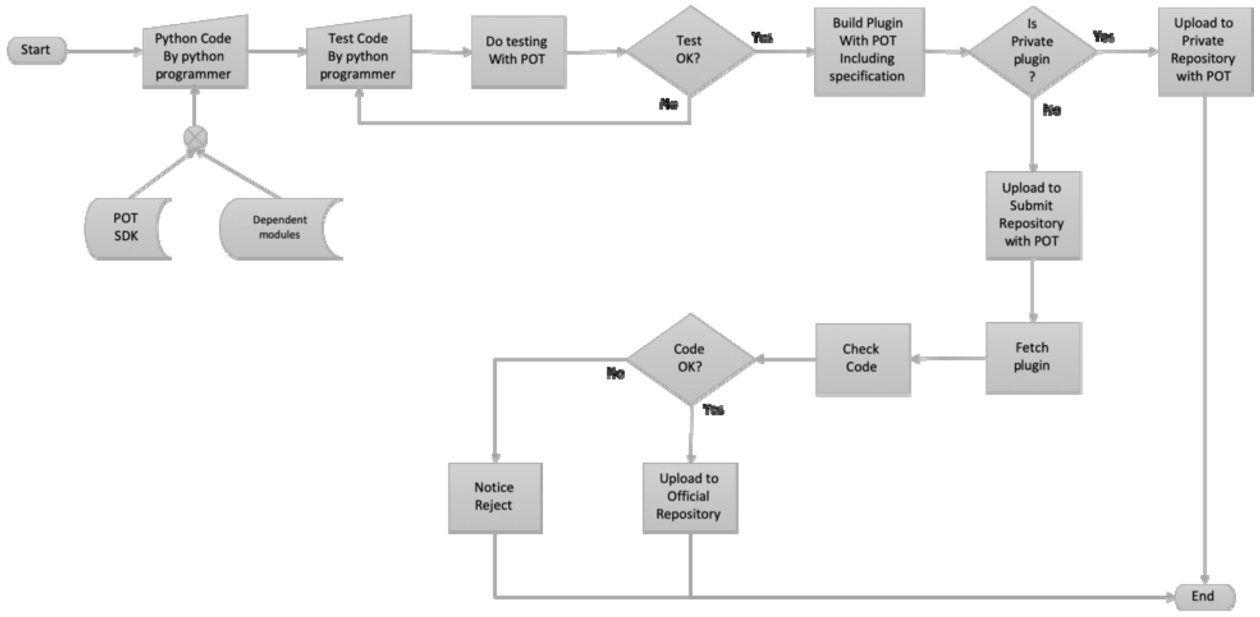

Start
Python Code By python programmer
Test Code By python programmer
Do testing With POT
Test OK?
Yes
No
Build Plugin With POT Including specification
Is Private plugin ?
Yes
No
Upload to Private Repository with POT
POT SDK
Dependent modules
Upload to Submit Repository with POT
Fetch plugin
Check Code
Code OK?
No
Yes
Notice Reject
Upload to Official Repository
End

Figure 4. Flow diagram depicting a plugin qualification process

4. ARGOS RPA+ Python SDK

Python SDK is available to download free of charge to anyone from the ARGOS LABS website. It contains the following components:

- Documentation
- Coding templates
- Sample code
- ICON building utilities
- Packaging tools for submission

The packaging tools for submission will help the users double-check items below that are included in the upload package.

4.1 Operation Tool Attributes

- Tool name and display name
- Description
- Owner
- Group
- Version
- Icon
- Last modified date/time
- Supported platform (Windows, Linux, Mac, iOS, and Android)
- Checksum

4.2 Parameter Attributes

- Parameter name and labels
- Option string in case option

- Action (store, store true, store false, and append)
- Choices (select only from one in the list)
- Default value
- Help
- Input method (password, file/folder read/write, and mouse click)
- Input group
- Show default or not
- Type (string, integer, and float)
- Constraint
    - min_value, max_value, greater, greater_eq, less, less_eq
    - equal, not_equal
    - Regular expression match

5. Extraction of STU Specification by Parsing a Python Program

One of the essential functions of POT is to parse or extract specific information from the Python code, which eventually shows as one of the building blocks in STU. A flow diagram depicting how POT parses a Python program to extract an STU specification is shown in Figure 5. Primarily, the information that becomes the parameter-setting sections is extracted from the argument portion of the Python code and processed as STU specifications. Some implementations of ARGOS RPA+ POT automatically convert Python code into STU building blocks by analyzing Functions and Arguments and generating Specifications for the STU plugin.

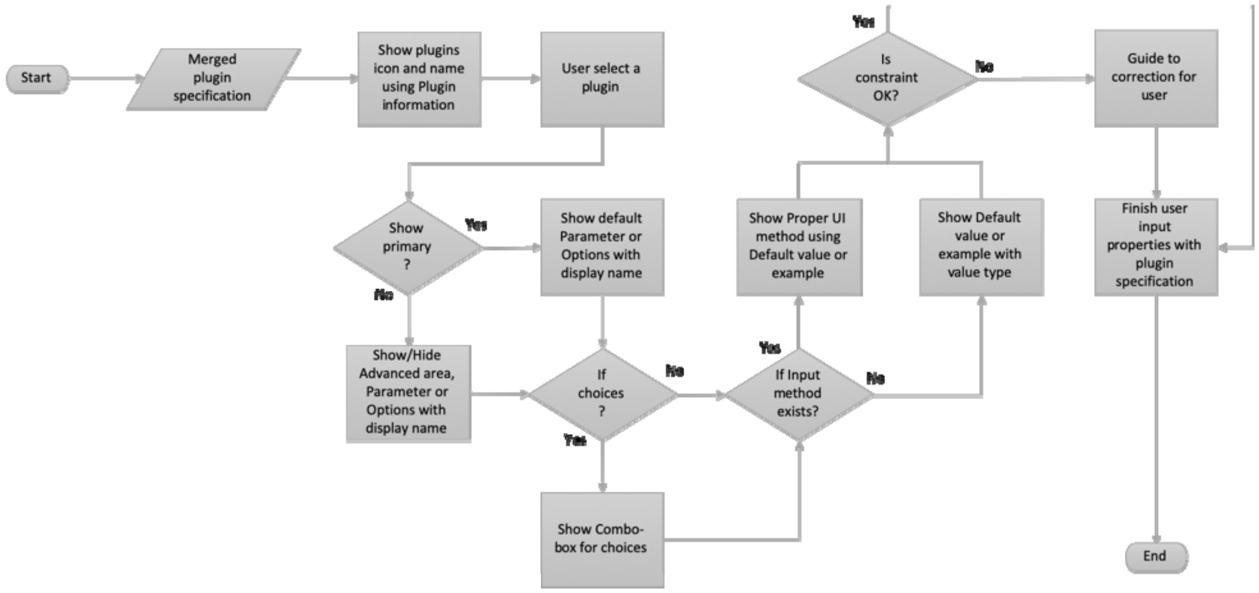

Start
Merged plugin specification
Show plugins icon and name using Plugin information
User select a plugin
Show primary ?
Yes
No
Show default Parameter or Options with display name
Show/Hide Advanced area, Parameter or Options with display name
If choices ?
Yes
No
Show Combo-box for choices
If Input method exists?
Yes
No
Show Proper UI method using Default value or example
Show Default value or example with value type
Is constraint OK?
Yes
No
Guide to correction for user
Finish user input properties with plugin specification
End

Figure 5. Flow diagram illustrating how POT extracts STU specifications from original Python code

6. Management of ARGOS Marketplace and Repository (Public and Private)

ARGOS RPA+ has a cloud-based administrative system called ARGOS Supervisor (nicknamed Chief). Chief also has a set of functionalities to manage the Python plugins from the user viewpoint.

ARGOS users can sign in to their Chief account to see which plugins available to them by browsing through the marketplace and the list of private repositories they are entitled to access. They can make a purchase (it could be a zero-dollar purchase if the plugin is free). Or, the user can access a private repository (or repositories) if he/she has the privilege to use the plugin tools registered in such private repositories. Chief collects the plugin information via ARGOS POT as shown in Figure 6 below. Figure 7 is a flow diagram depicting an associated transaction flow. Some implementations of ARGOS RPA+ prepare both public and private repositories for plugins, enabling Python coders (plugin builders) to select the availability of the plugins.

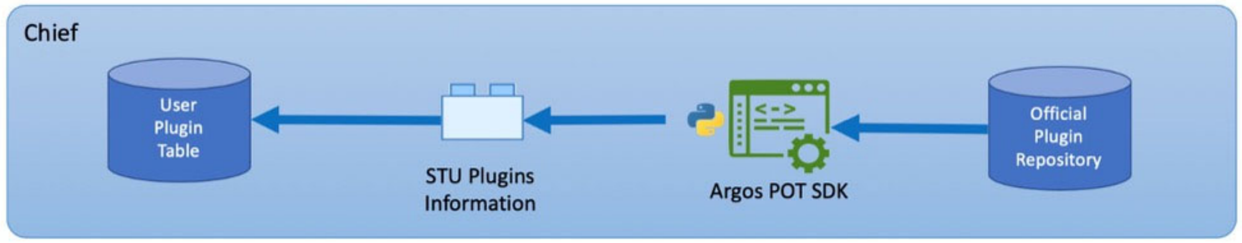


Figure 6. Process of making plugin information available to a user via Chief

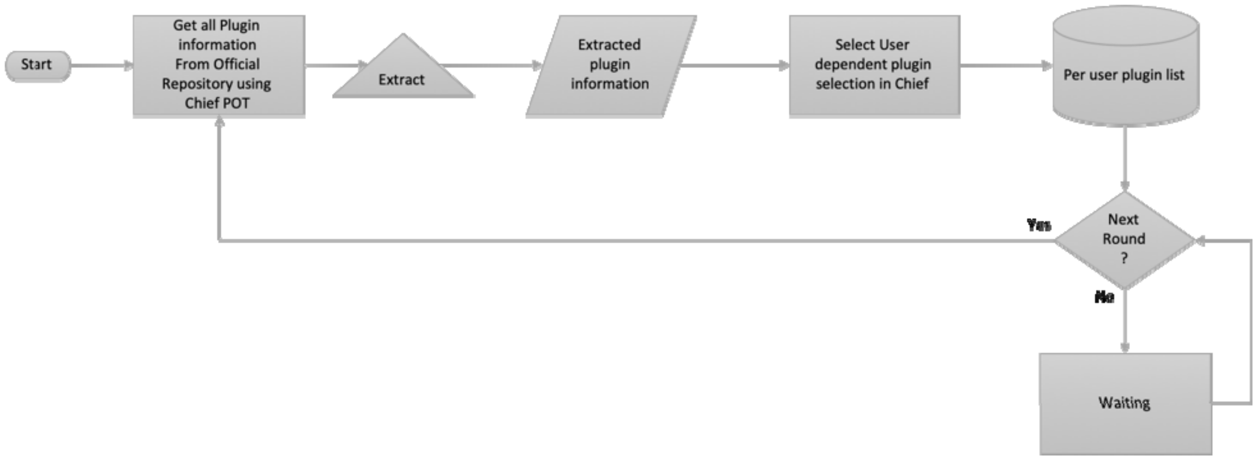

Start
Get all Plugin information From Official Repository using Chief POT
Extract
Extracted plugin information
Select User dependent plugin selection in Chief
Per user plugin list
Next Round ?
Yes
No
Waiting

Figure 7. Flowchart depicting how Chief collects plugin information and presents it to a user

7. Acquisition of Plugin Modules by STU

STU integrates a software called Plugin Package Manager (PPM), which communicates with Chief through REST API and the official or private plugin repository. Every time STU starts, PPM will contact Chief to get a user-dependent plugin list and repository to get the specifications of the plugin. It covers both public and private repositories. This process is illustrated in Figure 8, while an associated flow diagram is shown in Figure 9.

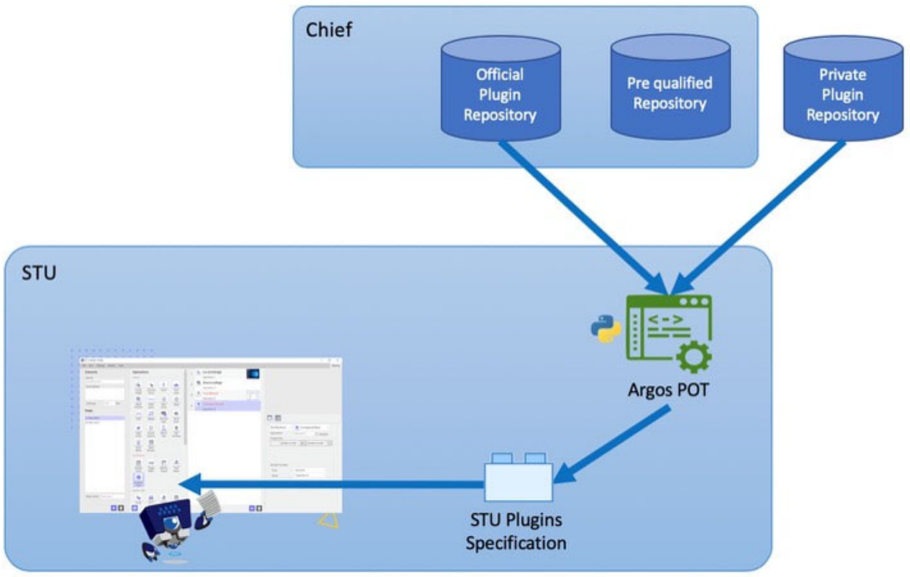

Chief
Official Plugin Repository
Pre qualified Repository
Private Plugin Repository
STU
Argos POT
STU Plugins Specification

Figure 8. How STU obtains new or updated plugins from POT

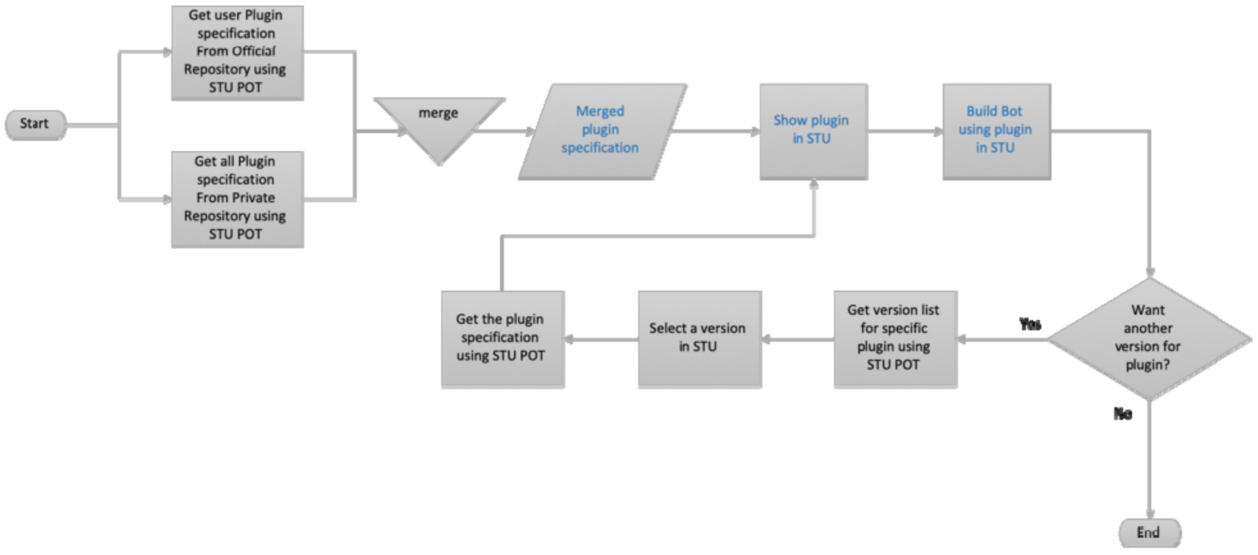

Start
Get user Plugin specification From Official Repository using STU POT
Get all Plugin specification From Private Repository using STU POT
merge
Merged plugin specification
Show plugin in STU
Build Bot using plugin in STU
Want another version for plugin?
Yes
Get version list for specific plugin using STU POT
Select a version in STU
Get the plugin specification using STU POT
No
End

Figure 9. Flow diagram depicting how STU obtains and updates plugins

8. Bot Execution Scheme of the Plugin Architecture

Once the bot has been built with plugins, STU packages the bot into a file and hands it to PAM. Then, PAM, using her own Plugin Package Manager (PPM), identifies the plugins being used in a specific bot and asks for the plugin package that contains the original Python script. Then, the plugin gets transmitted to PAM. This plugin package can come from either the public or the private repository. Figure 10 shows this mechanism, while Figure 11 presents the associated flow diagram.

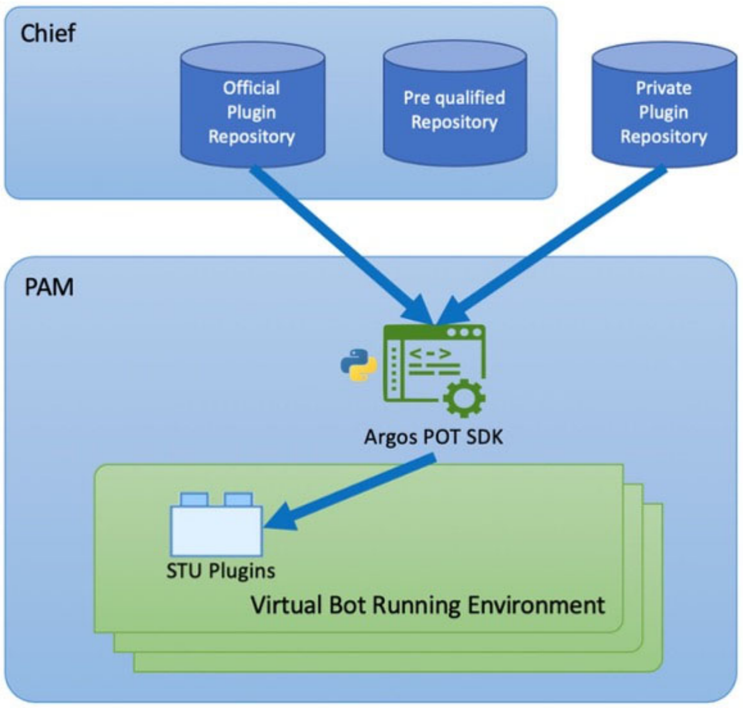

Chief
Official Plugin Repository
Pre qualified Repository
Private Plugin Repository
PAM
Argos POT SDK
STU Plugins
Virtual Bot Running Environment

Figure 10. Diagram showing how PAM retrieves the original Python code from repositories

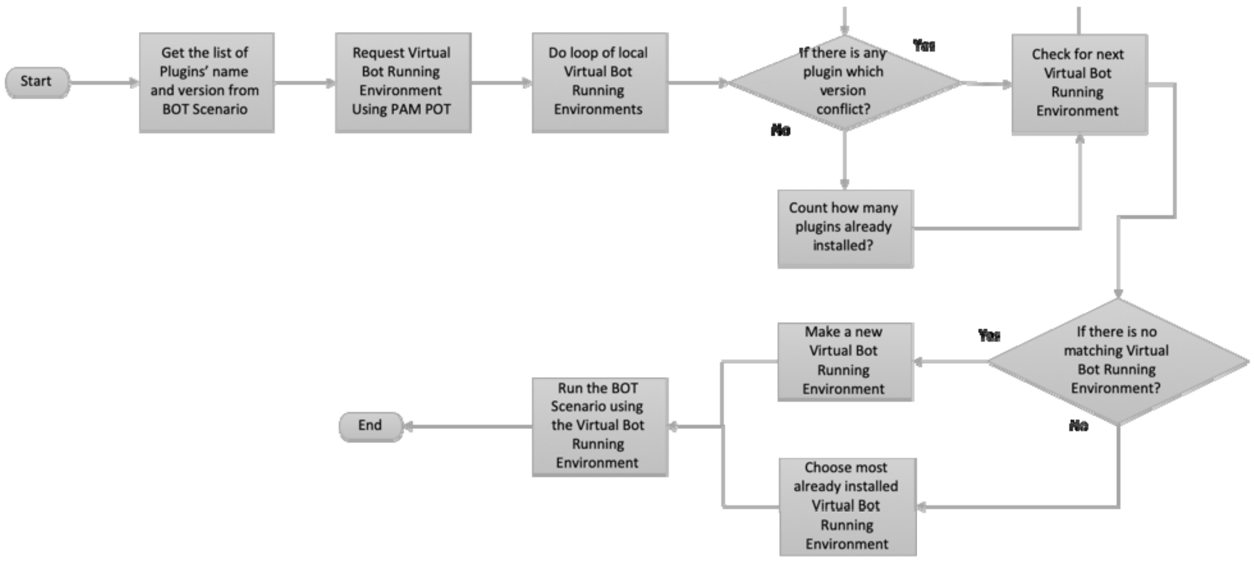

Start
Get the list of Plugins' name and version from BOT Scenario
Request Virtual Bot Running Environment Using PAM POT
Do loop of local Virtual Bot Running Environments
If there is any plugin which version conflict?
Yes
No
Check for next Virtual Bot Running Environment
Count how many plugins already installed?
If there is no matching Virtual Bot Running Environment?
Yes
No
Make a new Virtual Bot Running Environment
Choose most already installed Virtual Bot Running Environment
Run the BOT Scenario using the Virtual Bot Running Environment
End

Figure 11. Flow diagram depicting how PAM obtains the original Python code for execution

9. Governance of Bots with Plugins for Version Control and Security Management

Python-based plugin architecture makes it possible for ARGOS RPA+ users to access tens of thousands of functionalities that are already available in the existing Python community. This is one huge benefit of "openness." On the other hand, the risky aspects of openness are managing security and controlling versions; in other words, how to ensure the governance of bots. ARGOS RPA+ has an architecture that provides this governance. PAM, the Process Automation Module that executes the bot, has a mechanism to prepare a “sandbox”-like environment specific to every single bot. This is like a virtual device for a bot and makes it extremely efficient to execute the automation because the components that are required by the bot have been prepared in advance. When running the bot, PAM checks for the bot’s versions and components as well as its authenticity for the builder and the executor. These mechanisms can provide safety measures and management capabilities against accidentally or intentionally executing unauthorized bots in the field.

Furthermore, these mechanisms can solve the version-mangling problem. For example, two bots, Bot A and Bot B, both have a dependency on the same plugin, Plugin C. The problem becomes apparent when we start requiring different versions of Plugin C. Maybe Bot A needs v1.0.0, while Bot B requires the newer v2.0.0, for example. Using a sandbox architecture helps mitigate this problem.

10. Summary

ARGOS RPA+ is not just an RPA; rather, it can be considered another form of evolution in the history of the software development platform. It brings in one layer of abstraction from Python, just like the C languages that evolved from the preceding “high-level” languages. Connecting RPA with the Python community is a natural development because Python is essential to future technologies such as AI/ML and automation. As the term *RPA* is quickly shifting toward new terms such as *intelligent process automation*, ARGOS RPA+ has already taken a few steps forward in the same direction. ARGOS RPA+ and its Python connection will bring scalability to five dimensions.

- Functionality
- Deployment environment
- Deployment platform
- Deployment size
- Future proof

All of these scalabilities are equally important in enterprises’ software development platform selection today.