Chapter 9: Experiment 5 – Robotic Process Automation

This chapter introduces and elaborates the architectural abstractions that were utilized in the experimental system titled Robotic Process Automation System [121-122][102]. While there exist a significant number of architectural elements and design patterns in this system’s architecture, the focus of this discussion is on those that are potentially useful as Plug and Play constructs. The purpose of this platform is to automate inter-system processes by bridging systems using automation. To facilitate this, the platform provides the capability to encode, analyse and automate processes that are expressed using a process programming language. Accordingly, the platform adopts an extensible architecture [121-122][102] in which automation agents are developed independently and then plugged into the platform as defined by the process specification. The reasons for adopting an extensible architecture include the need to extend the functionality required by the automaton to perform the activities, interact with a wide variety of software systems, automate a wide variety of software systems, and the complexity of the software technologies involved in the automation.

In the context of Plug and Play, the objective of this experimental system is to “reduce the complexity of the plugged components while delivering high performance”, wherein, extensible components can be plugged and unplugged based on the process specification and can consistently deliver high performance and longevity (24 x 7 unattended performance). This is achieved by standardizing the plug and play interface, specification driven consumption of extensible component services and lossless resource management. Accordingly, the hypothesis for this experiment is: “If a standardized plug and play mechanism is utilized and adopted to plug components of varying complexity, then consistent high performance is possible”. The core functionality of the platform are: a) support for a wide variety of language constructs in order to encode processes and b) support for a wide variety of automation agents in order to provide operational support for these constructs. The methodology adopted to validate the hypothesis is to treat each of the automation agent as a pluggable component which is utilized as per a user-defined specification. Accordingly, the goal of this experimental system is to “enable Plug and Play of automation agents based on an automation specification”.

This chapter is organized as follows: a) The section on the problem space provides an overview of the robotic process automation problem, b) The section on the solution space provides a possible solution that is Plug and Play compliant, c) The section on the requirements lists down the stakeholder concerns, d) The section on the quality characteristics lists down the functional and non-functional characteristics of the system, e) The section on decomposition brings forth the underlying component framework and
supporting run-time, f) The next 6 sections provide an overview of the various components and their composition, and g) The last section discusses about the Plug and Play architectural abstractions of this experiment. It uses the architecture description template presented in Annex A to express the architecture description of the robotic process automation platform.

9.1 The Problem Space

Globally, the proliferation of information technology (IT) has been substantial, considering the rate of adoption by the masses. The impact of IT is evident in variety of industrial sectors like retail, manufacturing, insurance, telecom, and so on [102][121][130]. In these sectors, every organization has a patchwork of IT systems made up of new and old, compatible and not-so-compatible software. They provide the functionality needed by these organizations. IT provides a platform that enable these organizations to integrate and coordinate their business processes. IT also enables information sharing across all functional and management hierarchies. Each IT system internally supports the transactions, and sometimes the processes to realize this functionality. In all these cases, the presumption is that there exists a smooth flow of information from one system to another system. However, it is not hard to see that there could be duplication of data, and other kinds of overlaps between these IT systems. It is also possible that data in one IT system is needed by transactions in other IT system. This situation is ripe for loss of integrity of these IT systems due to lack of consistency between these systems in relation to the organization. Maintaining integrity across IT systems requires a disciplined usage of the systems, often requiring a role-playing user to login into multiple IT systems and carefully use them to assure integrity.

9.2 The Solution Space

A popular approach among organizations in order to address the IT system integrity issues is the Maker-Checker process. The underlying concept is that for each transaction performed in the IT systems there must be at least two different individuals, who play different roles in the process, who are necessary for completing the transaction. While one individual performs the transaction by making the appropriate changes in the IT system, the other individual is involved in confirmation/authorization of the transaction. Based on the complexity of the transaction, more individuals can be involved. In a typical situation, as illustrated in figure 9.1, 4 different individuals are involved in performing the transaction: The first individual prepares the data that should be transacted in the IT system, the second individual is involved in validating the data that is to transacted, the third individual performs the transaction in the IT systems and the last individual validates the transaction performed by the third individual. Here the segmentation
of the duties and the roles performed by the various individuals plays an important role in the successful execution of the transaction. It is evident that this solution to maintain integrity between these IT systems is manual, tedious and prone to error.

The proposed solution is to Create an IT robot that would emulate human activities in a computer, run automated IT tasks, perform these IT at a higher rate, high endurance, reliability, precision and speed, organize tasks for integration, repeatability and scalability and manipulate and interact with IT systems to facilitate automation. The proposal is to create a comprehensive collaborative environment for automating human tasks in which many tools work together to provide the desired robotic automation. This environment is the Robotic process automation System that is discussed in this chapter. The various steps involved in this automation are: a) Define people process steps straddling across IT systems for integration, b) Identify manual activities of human actors integrating information in IT systems, c) While enacting, identify activities that can be performed by machine and automate them, d) Gradually increase the level of automation till all the steps are automated and, e) When the entire process is completely automated, systems are bridged by automation. In essence, automation is introduced as a means to reduce/remove human efforts impended in maintaining information integrity. This system aids in identifying those human activities that straddle across multiple IT systems and automates it. Such automation increases accuracy, reduces latency, increases efficiency, improves productivity and reduces hardware burden imposed by manual processes.

Accordingly, the goals of the robotic automation platform are:

- Create an automation platform in which multiple automation agents can be plugged in and can be instantiated in order to automate human activities performed on a computer.
- Support discovery of activities, the software systems/applications that are manipulated when performing these activities and the associated interfaces.
• Support the execution of the process program that is strung together by putting together the different tasks by utilizing appropriate interfaces to computing infrastructure.
• Support the different software systems, associated technologies and also the different graphical user interfaces.
• Support improvement of man-machine work efficiency by allowing machines to do the tedious work while enabling the humans to make qualitative decisions.

9.3 Requirements for Robotic process Automation
i) Reduce time and effort impeded by role players to perform their IT systems integrity maintenance tasks.
ii) Reduce complexity of work done by role players while increasing their productivity and throughput.
iii) Reduce/Remove non-value adding tasks performed by role players and reduce overheads in tasks performed by them.
iv) Increase effectiveness of role players while increasing operational effectiveness of computing resources.
v) Complete tasks on or within schedule and Reduce rework by eradicating bottlenecks and inefficiencies.
vi) Improve efficiency of organizations and the accuracy of information maintained by the organizations thereby reducing business latency.
vii) Improve creativity of human role players by enabling them to do more with less
viii) Establish interfaces to all IT systems in an organization with the ability to read, write and manipulate information.
ix) Accommodate change in IT systems, while manipulating it, which arises due to change in business processes or underlying technology or new layouts.
x) Support sharing tasks, resources and processes with other automation agents as well as facilitate collaboration, cooperation and coordination with other role players
xi) Support customization of the encoded human tasks automation based on the prevalent situation.
xii) Collect as much relevant data as necessary for obtaining the necessary insight for making fact based decisions.

xiii) Support a wide variety of interfaces for wide variety of technologies and IT system architectures that keep evolving all the time.
xiv) Handle micro-level information flows between software applications, supporting processes, and supporting computing resources.

xv) Discover the human tasks that needs to be automated along with the order of their execution.

xvi) Model human tasks to be automated and define the operational strategy to be adopted by the role playing agents.

xvii) Prepare and transform the data necessary for automation of human tasks.

xviii) Automate human tasks expressed as automation processes by interacting with appropriate information systems.

xix) Standardize and scale up human tasks automation for deployment across all customers engagements.

xx) Monitor human tasks that are automated, their execution, outcomes, and performance indicators closely in real-time.

9.4 Quality Characteristics of Robotic Process Automation

Quality characteristics of a system are a set of essential and distinguishing attributes that have a pragmatic interpretation of the system’s inferiority or superiority. By controlling the quality characteristics, it is possible to ensure that the system delivers desired value to its stakeholders [100-102][130] [132-133]. For this discussion purposes, the author utilized ISO 25010 [103] as the reference quality model. The subsequent sections list down the quality characteristics of the robotic process automation system.

1. Functionality:
   a. Suitability: Aids users in performing 80% of their IT tasks
   b. Accurateness: Unattended, 24 x 7 hours task automation
   c. Accurateness: Always performs (100%) the right task for the right role at the right time
   d. Interoperability: Can work in atleast 80% of desktops in the organization
   e. Compliance: Complies to organization policies (100% policies)
   f. Security: Assures confidentiality of transacted data (100% confidentiality)

2. Reliability:
   a. Maturity: Can outsource 80% of tasks to the machine
   b. Fault Tolerance: Handles all system failures (100%) gracefully while performing the tasks
   c. Recoverability: On exceptions, uses rules and conditions to rollback or abort tasks (50%)

3. Usability:
   a. Understandability: Users can comprehend and use entire system (100%)
b. Learnability: Guides thru the steps of using the system (50% guidance)
c. Operability: Uses familiar interfaces for interacting with the system (80% compatibility)

4. Efficiency:
   a. Time Behaviour: Always completes the tasks on or before time and within schedule (100%)
   b. Resource Behaviour: Mimics Users user interaction activities (80%)

5. Maintainability:
   a. Analyzability: Users can keep track of progress always (100%)
   b. Changeability: Users can change their definitions on demand (100%)
   c. Stability: Absorbs 80% of perturbations introduced by Knowledge worker and their environments
   d. Testability: System performance and its use to automate tasks is reproducible (100%)

6. Portability:
   a. Adaptability: Adapts to change in the organization/working environment (100%)
   b. Installability: Organizations can scale up the level of automation across its associates (90%)
   c. Conformance: Conforms to organization infrastructure policies (100% compliance)
   d. Replaceability: Users can replace any of their task definitions (100%)

9.5 Decomposition of Robotic Process Automation Platform

Robotic process automation is a strategy that is adopted to automate tasks in order to contain costs while improving productivity and throughput. It predominantly involves: a) Integrating applications, b) Defining human processes so that the automation agents can understand, c) Streamlining and redefining processes for performance, d) Establishing man-machine interfaces, e) Networking a collection of automation agents, f) Monitoring process progress using soft sensors, g) Speeding up response times, h) Ensuring consistency of quality, and i) Deploying the automation in the organization while monitoring the gains over time. The objective is to reduce the use of human resources by identifying those man-machine interfaces which can be completely automated by using minimal organization resources and minimal interventions. In order to support Robotic process automation, it is necessary to support the life-cycle stages as given below.

The support for the Design life-cycle stage encompasses the identification of those man-machine interfaces that needs to be automated, identification of existing processes that is adopted as part of these interfaces and expression of machine executable processes. These processes are designed while taking into account escalation mechanisms, standard operating procedures, alerts and notifications, and

**Figure 9.2: Decomposition of Robotic Process Automation Platform**

The support for the Programming life-cycle stage encompasses the identification of the appropriate automation agents and process language constructs, formulation of the process program using these constructs, while organizing the process program in a cohesive and modular way for ease of maintenance. The process program is developed while taking into account commonalities and variabilities, extension libraries, programming conventions, extensions, errors and exception handling, language syntax and semantics, machine response times and automation documentation. The intent is to create a process program that can be deployed across the organization and used repeatedly rather than a single instance.
The support for the Execution life-cycle stage encompasses the volume and nature of data that is transacted, the availability and performance of the technology infrastructure, the man-machine interfaces that needs to be automated, the schedules to be followed, the key performance indicators and the audit trials to be generated that will assert successful execution. The process program execution occurs while taking into account operating environment, machine performance parameters, emulated user’s access rights and privileges, automated application stability and consistency, automated application responsiveness and timeliness and the ability of the automation agent to work within its boundaries. The intent is to perform the tasks that the user does with high speed, endurance, repeatability and flexibility.

Accordingly, the robotic process automation platform can be decomposed into the top level components: a) Process programming framework using which process programs can be developed, b) Process programming framework run-time that supports the process programming framework and also enables manipulation of the different processes, c) Process execution framework using which the process programs are automated, d) Process execution framework run-time that supports process execution framework and also provides appropriate bridges to different man-machine interfaces and software applications, e) Automation managers which assigns tasks to the automation agents and manages their performance, f) Data managers which manages the data to be transacted, the process programs that are developed, the key performance indicators and the state of the executed process. This decomposition is illustrated in Fig 9.2.

9.6 Process Programming Framework

In order to automate a process, it is necessary to understand the process and encode it using a notation (supporting both actions and data) that can be understood and executed by the machine. Additional information like the task parameters, sequence of task execution, inputs and outputs to the tasks, placeholders for data types needs to be expressed using this notation in order to have complete information. In this endeavour, the language that is used for representing the process program and the process of developing the program is of essence. Process programming is often the way in which the process is expressed so as to automate a specific task or solve a given problem. The difficulty with process programming is the fact that the man-machine interactions that is expressed using the process programming constructs and simple data types is complex. Typically, a programming technique would comprise of: a) a development life-cycle, b) a set of resources, c) a set of information structures and information flows, d) programming conventions and e) a schema for the process programs. The Process programming framework, as shown in Figure 9.3, is designed to support a wide variety of programming
techniques. It supports man-machine interface identification by means of an interrogator. It supports parameters configuration by means of a configurator. It supports syntax and lexical analysis by means of language tools. It supports decomposition and modularization of programs by means of program blocks. It integrates all these components and provides the ability to manipulate the process program by means of an integrated development environment.

**Figure 9.3: Decomposition of Process Programming Framework**

### 9.7 Process Programming Framework Run-Time

There are many automation agents that are used to automate various tasks in the process. All these agents are nearly independent of each other and continuously evolve. A programming framework that supports all these agents becomes too complex to design as a single monolithic system. Modularity and extensibility are design principles that facilitate extension of the programming framework so that new interrogators and configurators that utilize these automation agents can be added to it as and when demanded. These interrogators and configurators that bridges the automation agent to the programming framework are the pluggable components. A good design practice to support modularity in such situations is to embed the necessary extensibility services in the run-time system. The objective of the run-time system is to provide necessary interfaces that support interaction with the automation agents.

A design pattern used in embedded systems to support sharing of information is the bus. The bus connects the major components of the embedded systems and allows communication of information between the connected systems using address resolution. This design pattern is a candidate pattern in the programming framework run-time [PPFR] for supporting information sharing between the different...
Another design pattern that is used in USB devices for integrating parts is the host controller. The host controller facilitates interaction between the USB based devices and the host system. The host controller is a candidate pattern in the PPFR for supporting interaction between the parts. Another design pattern used in system integration to support information exchange is the interoperability bus. This bus facilitates conversion of information in one form into another. The bus is a candidate pattern in the PPFR for supporting information exchange between the parts.

![Diagram of Process Programming Framework Runtime](image)

**Figure 9.4: Software Decomposition of Process Programming Framework Runtime**

In the electrical industry, Plugs and sockets are design patterns that allow electrically operated devices to be connected to the power supply. While the plug is a movable connector attached to the device, the socket is fixed connector that is attached to the power supply. The plug and sockets are candidate design patterns that can be used in the PPFR to attach automation agents to the programming framework. While the interrogator socket can be used to query and obtain the interface specification from the automation agents, the configurator socket can be used to configure the parameters and attribute values for the specific action to be performed by the automation agent. Accordingly, the PPFR provides the underlying structure to plug-in interrogators and configurators using interrogator and configurator sockets, exchange and share information using the information bus. As shown in figure 9.4, it is a composition of: a) Interrogator Socket which serves as the query interface for an automation agent facilitated through its run-time, b) Configurator Socket which serves as the configuration interface for an automation agent facilitated through its run-time, c) Host Controller which provides resources, capabilities and services to the automation agents, d) Information Bus which provides the resources, capabilities and services for manipulating and transforming the data that is placed in the Bus.
9.8  **Process Execution Framework**  

Process execution is the process by which the machine performs the tasks that are defined as the process program. These tasks trigger sequence of actions in the automation agents on the pre-configured software systems. The effect of these actions depends on the configuration of the tasks as defined in the process program. It also depends on the semantics of the language in which the process program is expressed. During process execution, the tasks are executed in a prescribed order that is encoded in the process program. Process execution typically starts at the task that is earmarked as the first tasks of the process. A task can either invoke some action on an automation agent or it can dictate what the next task should be. By default, the task that is sequenced to be the next task in the process program is taken up. When each of these tasks are taken up, a set of actions is scheduled for the automation agents which then go about and perform these actions on the underlying software system infrastructure. The transaction and process data to be utilized for the execution is initialized and populated with the data that is relevant for this particular execution instance.

![Diagram of Process Execution Framework](image)

**FIGURE 9.5: DECOMPOSITION OF PROCESS EXECUTION FRAMEWORK**

The Process execution framework, as shown in Figure 9.5, is designed to support a wide variety of process programs that are developed based on the underlying process language. During execution, the state of the execution and the performance indicators are saved in persistent storage as part of the process data of that specific execution instance. This information is useful in cases where the automation is interrupted for human-input and is restarted at a later point in time.
9.9 Process Execution Framework Run-Time

There are many automation agents that are used during process execution. All these agents are nearly independent of each other and continuously evolve. While the automation agents provide the necessary functional capabilities, the run-time provides the interfaces for plugging in these functional capabilities into the execution framework. In order to invoke the desired functionalities/actions in an automation agent, the execution framework run-time facilitates the dynamic late binding of automation agents into the framework. It provides a standard mechanism for this purpose which comprises of: a) a locator service to uniquely identify a plugged in automation agent, b) a creation service to facilitate the instantiation of the agent, c) a bus service to exchange data, d) an invocation service to facilitate function invocation and e) an eventing service to facilitate event handling. In essence, the run-time controls the various automation agents in such a way that during execution it is difficult to separate them from native components.

A design pattern used in embedded systems to support sharing of information is the bus. The bus connects the major components of the embedded systems and allows communication of information between the connected systems using address resolution. This design pattern is a candidate pattern in the execution framework run-time [PEFR] for supporting information sharing between the different parts. Another design pattern that is used in USB devices for integrating parts is the host controller. The host controller facilitates interaction between the USB based devices and the host system. The host controller is a candidate pattern in the PPFR for supporting interaction between the parts. In the electrical industry, plugs and sockets are design patterns that allow electrically operated devices to be connected to the power supply. While the plug is a movable connector attached to the device, the socket is fixed connector that is attached to the power supply. The plug and sockets are candidate design patterns that can be used in the PEFR to attach automation agents to execution framework. The automation socket provides the mechanism to instruct the automation agent to perform the necessary action on the underlying software system. In the Windows operating system, applications are event-driven. The various applications do not call a function to obtain the events (keyboard, mouse, display etc), but rather wait for events to be posted onto their respective message queues. This eventing capability is a candidate design pattern for the PEFR in order to handle the various events that occur in the parts that are plugged into the execution framework.
As shown in figure 9.6, it is a composition of: a) Automation Socket which serves as the interface for invoking the actions in different automation agents, b) Host Controller which provides resources, capabilities and services to the automation agents, c) Information Bus which provides the resources, capabilities and services for manipulating and transforming the data that is placed in the Bus, e) Event Manager that serves as the placeholder for receiving and processing notifications from the different automation agents, and f) Event Handler which serves as the placeholder for invoking services/functionalities from various automation agents and processing their results in response to the various actions triggered by the execution framework.

### 9.10 Automation Agent

An automation agent operates on a specific computing resource (GUI element, Virtual Memory and so on) of a software system and includes a set of events that can occur, optional conditions that are checked, actions that can be performed on this computing resource, modalities of data transfer from/to the computing resource, the data schema for exchanging information with the computing resource, and error events that are triggered when there is a failure. The agent is made up of a set of data and functions that manipulate the underlying software system. Access to both the data and the functions of the agent
is achieved through the use of interfaces. Using these interfaces, the agent can either be instructed to perform an action or it can wait for desired conditions (which results in a firing of an event) before performing the desired action. An agent also responds to a set of events that occur during the life cycle of its existence. The choice as to which action should be executed depends on the configuration parameters of the agent. The effect of the triggered action depends on the operational semantics that has been defined for that particular action. When an action is triggered, the state of the computing resource, before and after that action is performed, is stored as event audit data of that specific execution instance.

### 9.11 Automation Agent Run-Time

There are many automation agents that are used during process execution. All these agents are nearly independent of each other and continuously evolve. While the automation agent provides the desired functionality necessary for interfacing/interacting with a wide variety of software systems and technology infrastructure, the automation agent run-time provides the necessary run-time library support, operating system interface support and the ability to plug the automation agent into the automation socket of the execution framework. Once the automation agent is plugged to the execution framework, the automation agent run-time serves as an abstraction layer that translates the function invocation on the execution framework into an invocation of the relevant function in the automation agent. When the automation agent is no longer necessary, the run-time provides the facility to unplug the automation agent from the execution framework.

The run-time utilizes the interface services offered by the Interface manager to interact with the underlying software system and technology infrastructure. It utilizes the plug and play services offered by the interface manager to dynamically attach and detach from the automation socket. The interface manager treats the host system as another software system to interact with and treats it at par with the systems that it manipulates. The run-time utilizes the data services provided by the data manager to obtain data from the various software systems that it interacts with (including the host system). The run-time utilizes the action invocation services provided by the service manager to trigger specific sequence of actions on the target software system. The run-time utilizes the services provided by the event manager to monitor and handle the events that are triggered by the host system as well as the target system. Its decomposition is presented in figure 9.7.
The experimental system addresses its primary goal of enabling Plug and Play of automation agents based on an automation specification by:

a) Encapsulating automation functionality by encoding them into the automation agent component.

b) Interfacing with a wide variety of software technologies using corresponding automation agents.

c) Encoding the plug and play interface mechanism as sockets and compatible plugs.

d) Encapsulating the plug capability as part of the automation agent run-time component.

e) Encapsulating the socket capability as part of the component framework run-time component.

f) Listening for events and dispatching them to appropriate handlers using the plug & socket interface.

g) Providing automation services to the component framework by using the plug & socket interface.

h) Providing interrogation services to the component framework by using the plug & socket interface.

The critical elements events, services, automation specification and transaction data that are required for proper functioning of the robotic process automation platform have been made accessible through the plug and socket interface. Depending on the complexity of the underlying software system and desired actions to be performed on them, the automatons are designed at varying levels of complexity while conforming to the same interface. Individually, these agents are designed for high performance,
low resource consumption and low processor and memory footprint and collectively they perform
together to realize high performance, throughput and efficiency. It is observed that "plug and play
architectures are high performant and scalable". Accordingly, the hypothesis "If a standardized plug and
play mechanism is utilized and adopted to plug components of varying complexity, then consistent high
performance is possible" was verified. Further, the underlying principles and constraints that are
discussed in the subsequent sections aid in reducing the complexity of the plugged components while
delivering high performance.

9.12.1 Principle of simplicity

According to Maier [51], performance and complexity do not go together and it is essential to simplify.
In the case of the robotic process automation platform, this principle is supported by selecting only simple
things for automation with high repeatability, endurance and consistency. Accordingly, each of the
automation agents are designed specifically for the purpose of automating specific human actions on a
specific software system. They are attached to the underlying software system when necessary and
detached after the intended purpose is achieved.

9.12.2 Principle of compatible support elements

According to Maier [51], the architecture of the supporting element should fit into the system that it
supports. In the case of the robotic process automation platform, the plug interface of the automation
agent and the socket interface of the programming framework are designed such that only compatible
automatons can utilize this interface. This principle is further implemented by allowing only the
compatible automatons of a software system to be plugged into it.

9.12.3 Principle of grouping and separation

According to Maier [51], while partitioning a system, it is necessary to group those elements that are
strongly related to each other and separate elements that are unrelated. In the case of the robotic process
automation platform, all those actions that can be performed on a software system are grouped together
and encoded into a single automation agent. This facilitates a logical separation of those automation
agents that are not relevant to that specific software system.
9.12.4 Principle of minimal communication

According to Maier [51], while choosing the sub-systems, it is necessary to define them in such a way that they are independent to each other and there is minimal communication between them. Accordingly, in the robotic process automation platform, the automation agent obtains the tasks that it should perform from the component framework. It exchanges the information necessary for performing the task from the component framework as well as passes on the information that it gathered back to the component framework.

9.12.5 Principle of independent partitioning and low complexity

According to Maier [51], while partitioning a system, it is necessary to choose the elements so that they are as independent as possible; that is elements of low external complexity and high internal complexity. In the case of the robotic process automation platform, the automation agents are independent of each other and serves as the extensible components. In order to complete a transaction, more than one automation agent is involved and each of them works within the boundaries of the tasks that they are instructed to perform. The tasks that they perform are of low complexity but they are performed with high repeatability, endurance, consistency and speed thereby delivering the desired performance, efficiency and throughput.

9.12.6 Principle of least privilege

According to Whitman [123], every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job. In the case of the robotic process automation platform, the different automation agents that perform the tasks on behalf of the user, utilizes only those privileges that are necessary for performing the task allocated to it. In case sufficient privileges are not available then the automation agents fails in performing the tasks.

9.12.7 Principle of economy

According to Whitman [123], every subsystem’s design should be as simple and as small as possible. In the case of the robotic process automation platform, the different automation agents are designed to be as small in size and as optimal in speed. They are designed to not do any other action other than those that are tasked to them by the component framework in accordance with the automation specification.

9.13 The Plug and Play Architectural Abstractions

This section introduces and elaborates the plug and play architectural abstractions of the robotic process automation platform. While many of these abstractions were discussed in earlier experiments,
their capabilities have improved at the end of this experiment, and are expressed as such. These abstractions are considered, based on the premise that “Software Plug and Play is a collection of standards, encoded in the run-time and embedded in the component schema, in order to support functional and structural extensibility of the robotic process automation platform”. These abstractions are:

**9.13.1 Plug**

A plug comprises of a collection of software resources that enables the plugged component to be executed on an underlying computing system. It serves as the gateway for the events, data and services that the plugged component manipulate as part of its computation and allows two way communication between the component framework and the plugged components. The plug encodes the interface mechanism for connecting to a compatible socket thereby enabling the plugged component to function as an external component of a component framework.

**9.13.2 Socket**

A socket is a handle that allows the plugged components to utilize the data, events and services that are exposed by the component framework. It facilitates two way communication of events, data and service invocation between the plug and the component framework. As an interface, the socket performs a type checking for compatible plugs to validate the connection and disconnects those that fail the validation.

**9.13.3 Component Framework**

A component framework defines specific interaction and composition standards that should be adopted by the constituent components and provides the set of elements required for supporting the execution of these components. It implements a component schema which describes the characteristics of the constituent components in terms of the services, service operations, events, and other metadata that it should support. It handles events raised by all of its constituent elements (including the plugged components) as well as dispatches events for processing by these constituent elements. It consumes services provided by all its constituent elements as well as provides services required by them. It allocates tasks for the plugged components and monitors its completion. It uses the success information received from the plugged components to decide upon the future course of action.
9.13.4 **Plugged Component**

A plugged component extends and adds functionality to the component framework thereby enabling the evolution of the underlying system. It utilizes the run-time services provided by the plug to attach itself to the component framework. It processes various events that it receives as well as dispatches some of these events to the component framework. It performs the tasks allocated by the component framework and returns the success or failure of the action back to the component framework.

9.13.5 **Event**

An event is an encapsulated set of software actions that are triggered during component execution, along with a set of data that capture the computational state under which the event was triggered. Such events are triggered when a set of pre-defined execution conditions are met. They can be triggered inside the component framework and dispatched to the plugged component for processing or triggered inside the plugged component and dispatched to the framework for processing.

9.13.6 **Service**

A service is an encapsulated set of software functionalities offered by a component, along with a set of policies of usage that declare the information necessary for the successful execution of the service, that can be invoked by other components for different purposes.

9.13.7 **Task**

A task corresponds to a set of actions that are performed by the plugged component in order to achieve a purpose. The task is the minimal unit of work that can be assigned to a plugged component. While performing the task, additional meta-data information is made available so as to complete the transaction. The success or failure of performing the task is utilized to make decisions on future course of action.

9.13.8 **The Plug and Play Component Model**

The Plug and play component model is put together by considering the plug and play architectural abstractions that were discussed earlier. Accordingly, the constituent elements of The Plug and play component model are: a) The component framework which provides the composition semantics, the logical structure for facilitating components composition, underlying structure in terms of interfaces, resources and protocols for the composition of components playing specific roles, b) The component
framework run-time which serves as a socket and provides the necessary functionality for supporting composition of components, communication between components, manipulation of components, management of components performance, invocation of functions and services supported by the components and transformation of data from one form to another, c) The information repository which hosts the information that can be manipulated, processed and transformed by the plugged components, and d) The components manager which schedules the tasks to be performed by the plugged components along with their respective run-times. Figure 9.8 provides a decomposition of this model.

![Figure 9.8: Decomposition of Plug and Play Component Model](image)

While the plugged component hosts the services and functionalities necessary for the composed component, its run-time provides functionality and resources needed by the component, and also the ability to serve as a plug. It provides basic services necessary for the various components to work together, generation of events which are then passed onto the component framework for further action, a connection receptacle that handles the dynamic connection between the component framework and the plugged component and so on. The interface services provided by the plug enables the socket to communicate and interact with the plugged component using the accepted interfaces. While the component framework hosts the functionalities and processes that are utilized by the end-users, its run-time provides functionality and resources needed by it, along with the ability to serve as a socket. It
provides basic services necessary for various plugged components to work together, for raising and handling events, for invoking various functions that are supported by the plugged component and for managing data flow between the plug and socket.

9.14 Summary

This chapter introduces and elaborates the architectural abstractions that were utilized in the experimental system titled robotic process automation platform [121-122] [102][130]. The approach to reduce the complexity of the plugged components while delivering high performance was illustrated by utilizing automation agents that perform tasks on underlying software systems and are attached and detached to the platform on demand. Accordingly, the hypothesis “If a standardized plug and play mechanism is utilized and adopted to plug components of varying complexity, then consistent high performance is possible” is verified. Understanding the key architectural principles of this experiment helped identify the underlying plug and play architectural abstractions. This was culled out and expressed in the form of an abstract Plug and Play component model.