Smart Factory Communications Using OPC-UA

Santhana Pandiyan Muniraj

ID: 726836896

Department of Mechanical Engineering

The University of Auckland

Supervisor: Prof. Xun Xu

A dissertation submitted for partial fulfillment of the requirements for the MECHENG.789 project in Mechanical Engineering, The University of Auckland, 2020.

Abstract

In recent years, the development of a machine-to-machine communication and Industrial Internet of Things has paved the way for developing intelligent and self-governing autonomous machines in smart industries. These autonomous machines are capable of collecting data from various sensing devices and other interconnected machine systems by utilising OPC UA communication protocols. Currently, these machines have the capabilities to send and receive data which is just sufficient for autonomous decision making. However, current low-level machine-to-machine communication involves only in processing data, and taking corresponding actions are limited to an individual machine system instead of sending commands to other interconnected machines. This paper primarily focusses on enhancing such limitation in the machine-to-machine communication in OPC UA protocols, and on making multiple decisions respective to received data and delivering commanding actions to the smart machines connected in vertical (higher-level machines) and horizontal machines (same level machines) in the process of making advanced machine-to-machine communication in smart factory. Moreover, the presented framework concept was developed and verified using python OPC UA server-client instances.

KEYWORDS: *OPC UA, Machine-to-Machine Communication, Smart Industries.*

Acknowledgements

I would like to express my special thanks to many people who have supported me for this dissertation. First of all, I would like to thank my supervisor, Dr Xun Xu, who guided me and provide valuable suggestions for my dissertation, which allowed me to complete my research as well as gain more knowledge that helped me to learn new concepts.

Secondly, I would like to thank lab technician Eshan Amiri, who gave me immense support throughout my research, such as setting up the machines, providing required logistics and also he has provided a lot of valuable suggestions from a different perspective.

Additionally, I would also like to thank Mr David Tomzik and Mr Zexuan Zhu, for providing me with valuable information about the existing technologies.

Finally, I would like to thank my family and friends who always support my decisions and gives me the strength to complete my master degree.

1 Table of Contents

List of Figures

List of Tables

Abbreviation

M2M	Machine-to-Machine
IIOT	Industrial Internet of Things
OPC UA	Open Platform Communications United Architecture
PC	Personal Computer
HMI	Human-Machine Interface
SCADA	Supervisory Control and Data Acquisition
COM	Component Object Model
DCOM	Distributed Component Object Model
API	Application Programming Interface
OPC DA	OPC Data Access
OPC AE	OPC Alarm & Events
OPC HDA	OPC Historical Data Access
OLE	Object Linking and Embedding
PLC	Programmable logic controller
DCS	Distributed Control System
OPC DX	OPC Data eXchange
OPC XML DA	OPC eXtensible Markup Language Data Access
HTTP	Hyper Text Transfer Protocol
SOAP	Simple Object Access Protocol
MES	Manufacturing Execution System
ERP	Enterprise resource planning
XML	eXtensible Markup Language
TCP	Transmission Control Protocol
FDI	Field Device Integration
EDDC	Electronic Device Description Language
FDT	Field Device Tool
I/O	Input/Output
URL	Uniform Resource Locator
ns	namespace
DT	Digital Twin
PT	Physical Twin
ML	Machine Learning

1 Introduction

The advancement of smart industries and ever-growing needs of the world has brought many favourable changes in the manufacturing industries such as in the form of machine-to-machine (M2M) communication and Industrial Internet of Things (IIoT). This fast-paced growth also led to the development of ubiquitous communication protocol that is very much capable of collecting and processing data. Typical M2M communication protocols work in such a way that it can gather-up all the machine data and respond with a simple task when compared to communication. Considering the current capabilities of the communication protocol that can collect data from all over, but in contrast, it completes a simple task which shows unmatching machine capability. This Ever-present communication is the initial steps of making almighty machines that can perform all actions individual or by combined. Advanced M2M communication is the first steps to achieve such capable machines. This advanced M2M communication protocol can send and receive data, and additionally, it can make the decision and send commanding actions to the interconnected machines. Moreover, this communication can be established by using OPC UA standards. In this paper deals with the making of advancement M2M communication using OPC UA standards.

This dissertation is organised as follows by starting with the main objective of the research project and methodology handled for achieving the research scope in section 2, followed by the comprehensive history of OPC communication protocol from the day of introduction till the update to date, relevant to the research work in section 3. Section 4 contains a brief introduction to the current capabilities of Machine-to-Machine communication and states its problem. Section 5 contains the details of conceptual information relevant to the development of advanced M2M communication and its advantages. Section 6 gives a detailed level process that was undergone for developing the M2M framework using python based OPC UA framework. Section 7 verifies the working of the client-client communication concept utilised for achieving advanced M2M communication. Section 8 contains the details on the future scope and related works that could be achieved by using the developed Advanced M2M communication, and finally, in section 9, this research work finishes with a conclusion.

2 The objective of this Project

The main objective of this project is to develop a framework that would enable an advanced level of M2M communication using OPC UA communication protocol. This project utilises the python-based OPC UA communication protocols to develop advanced M2M communication. The primary purpose of the advanced M2M communication would enable the machines in the smart factories to communicate with each other and allow seamless data transfer. This data transfer would enable the machines to make a decision based on the output from another different machine and deliver a commanding action to other machines that are either connected vertically (higher-level machines) or horizontally (same level machines) for the process of making a smart factory with intelligent and proactive machines. The following are the main objectives of this research.

- To develop a framework that would enable advanced machine-to-machine communication and make proactive decisions based on the output from another machine and also confirm that machine functions usually without impacting its regular task when one or more server machines are disconnected from the framework.
- To develop and verify the OPC UA client-client communication in the process of making an advanced machine-to-machine communication framework.

3 Literature Review

3.1 History of OPC

In the early nineties, most of the automation industries started using an approach of PC and software-based automation system to control industrial equipment. Primarily, this trend was more use of Windows-based computers for controlling and visualising. The major hurdle during that time of development was on how to standardise the communication protocols and create interface connectivity to collect and collaborate data from various device to a single protocol on to the software-based automation system. However, this interface problem is not new to windows operating systems; earlier, it had a similar problem for printers where vendors need to provide different drivers for their product. However, Windows resolved this problem by incorporating the idea of using a single driver for multiple vendor-based printers and in simple term knows as the Plug and Play approach of drivers. Since, the Human Machine Interface (HMI) and Supervisory Control and Data Acquisition (SCADA) software had the

similar problems, a task force initiated in 1995 to produce similar Plug and Play type approach of drivers for accessing data from the Windows-based system for the automation devices [1].

In 1996, the OPC standard specification was released to overcome the earlier issue. This OPC standard was maintained and developed by a non-profit organisation OPC Foundation. The primary feature of this OPC standards API is defined by using the Microsoft Windows technologies models such as Component Object Model (COM) and Distributed Component Object Model (DCOM). This standard focuses on interoperability and multi-vendor specifications [2].

The OPC standard is a series of specifications that were developed by industry vendors, end-user and software developers. This series of the specification defines the interface between the Clients and Servers as well as the Servers and Servers communication which includes the access of data, monitoring alarms and events, and access to historical data and other applications. In 1998, the OPC Data Access (OPC DA) specification was introduced to access the real-time data from the source device and transfer that to the sink device, for instance, the data is accessed from PLC and transferred to Human Machine Interface (HMI) without the knowledge of either device's native protocol. This standard was implemented to a large number of products, and this standard is still the most valuable interface that was developed by the OPC products [3].

During the initial release of the OPC standard, the primary objective of this protocol was to abstract PLC specific protocols such as Modbus, Profibus and few other protocols into a standard interface allowing that to communicate with the HMI and SCADA systems to interpret into an OPC specific protocol to make read/write requests into the device-specific protocols and vice versa. However, this initial developed OPC standard has a drawback in the form of incompatibility to operate on different operating systems other than that of Windows operating systems. Such that this OPC was initially known by the acronym OLE(object linking and embedding) for Process Control. These specifications are now called as the OPC Classic. This Classic OPC standards include OPC Data Access (OPC DA), OPC Alarm and Events (OPC A&E), OPC Historical Data Access (OPC HDA) and other OPC interface standards.

The OPC Data Access (OPC DA) interface allows it to read, write and monitor the data values of the connected devices such as PLCs, DCSs and other devices connected to HMIs. OPC DA is the most widely used OPC standards as the primary functions of these interfaces are the main requirements of the automation industries. The working principle of the OPC DA

is straight forward as OPC DA clients connect with the server and required variables (OPC items) are connected to read, write and monitor the changes in the variable data. This OPC clients connection are established with the OPC server by using an OPCServer object, and this server object is responsible for providing the methods for navigating through the address space hierarchy to browse the desired data and its properties such as the data type and access rights. At the same time, it accesses the data from the client group's data items using identical settings such as update time in the OPCGroup objects.

These added client groups can read and write the data by using the OPC client interface. However, there is a minor issue in accessing data or in other words as repeatability issue. Since the OPC client can access the data in a fraction of seconds, there is always an issue around the repeatability, for instance, a temperature sensor reads the environmental temperature for a two-second period cycle, and the OPC clients read the data for every one second where it would retrieve the unchanged values every alternate second, and this raises the concern over the repeatability. In order to avoid such situations, OPC uses the group policy of update rate where the client only receives the latest and updated values rather than collecting unchanged values.

OPC DA provides access to data in real-time. However, this real-time accessibility is not always permanent, since, there always exists the issue related to accessibility, communication issue or even the device unavailability. For example, there is a possibility that the device gets disconnected from the server or issue with the data type; such type of issues are handled by using the OPC data quality. If the quality of the data is reflected accurate, not available or unknown relates to the following specific qualities as good, bad or uncertain, respectively.

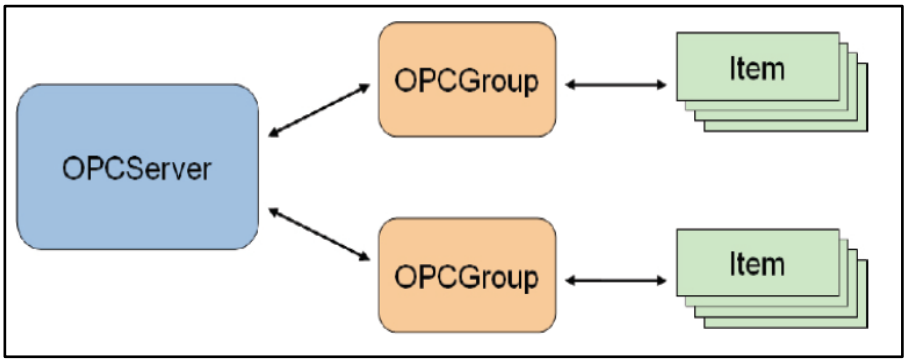

Figure 1. Objects created by OPC client to access data [1]

The OPC A&E interface allows the client services to access the event and alarm notifications. Alarms are the notifications that inform the client about any changes in

condition, for instance, the water level of a tank when it hits the maximum or minimum capacity of the tank, and events are the single notification that informs the client about any occurrences of an event that can be acknowledged. This acknowledgement of the Events and Alarms are made possible by utilising the interface OPC A&E. Hence this OPC A&E provides the flexibility to receive any alarms and events notification generated by the server. To receive any notification, the A&E client connects to the A&E server and subscribes to the respective notification, and any trigger in the server-side will automatically reflect on the client-side. OPC A&E interface has the capabilities to limit the number of notification on the OPC client. Such that capabilities can be configured by specifying filter criteria. This OPC client connects to the A&E server by creating an OPCEventServer object and then signal can be received by using the OPCEventSubscription where the messages can be received. As shown in the figure, the objects of the client-side interact with the server-side.

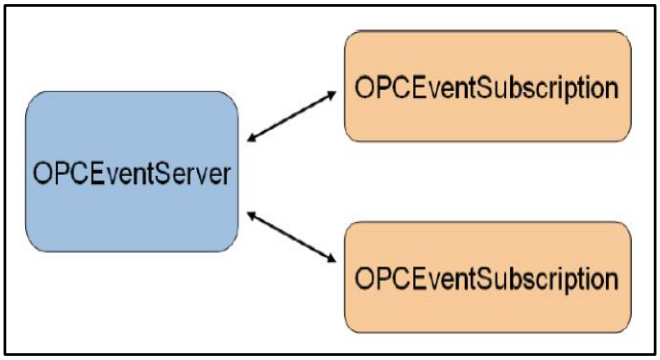

Figure 2. Objects created by OPC client for accessing events [1]

OPC DA gives real-time access to the continually changing data; on the other hand, this OPC HDA provides access to the data that is already stored. OPC HDA interface allows the client to retrieve the data from historical archives. The OPC client establishes a connection with the HDA server by creating an OPCHDAserver object. This object provides the interface for reading and updating the historical data. Moreover, there is another object called OPCHDABrowser object, which is useful in browsing the address space of the HDA server. There are three different ways to read the historical data. The first mechanism involves reading the raw data from the archive, where the data are browsed based upon the time, and variable name and in return the server sends all the data from the archive based on the time. The second mechanism involves fetching the data from the archive of one or more variables of a given timestamp. The Third mechanism involves reading the aggregate values from the specified time from the one or more variables. This returned value might include associated quality and

timestamp. Additionally, this OPC HDA not only reads the value it can also define methods for inserting, replacing and deleting data in the history database.

3.2 Other OPC interface standards

OPC has specified several other standards-based upon the specific needs. However, the OPC specification and the communication interfaces are common for all the COM-based OPC specification.

Additionally, OPC security specifies how to control client access to servers and protect it from unauthorised modifications. There are OPC complex Data, OPC Batch and OPC Data eXchange (DX) extensions are available with OPC DA. The Complex Data provides the specification on transferring complex structured data types [4]. The OPC DX offers the specification for data exchange between Data Access servers by defining a client behaviour and an interface of the client inside a server. OPC Batch extends the DA features that are required for the batch processing.

3.3 Advantages and Disadvantages of Classic OPC

The main advantage of this Classic OPC was a reduction in specification work for different APIs for different logistical needs that do not require a separate configuration and protocols for communication [6]. COM and DCOM provides a transparent mechanism and allows the client device to communicate with COM-object in the server machines in the same or separate network. Since this feature is readily available in all Windows-based operating PCs, this has reduced the development time of the product as well as the marketing time of OPC.

There are two disadvantages of using classic OPC standards; the first disadvantage is the interoperability issue. Since these communication protocols used for developing OPC uses the COM and DCOM technology which is native to the Windows-based operating systems, and it is not compatible with other operating systems is the major drawback of this OPC standards [7]. Another disadvantage is the configurable issue where DCOM communication is difficult to configure, and it has extended and non-configurable timeouts and it cannot be used for internet communications.

3.4 OPC XML-DA

Classic OPC had some issues in terms of interoperability, in order to overcome this disadvantage the OPC XML DA was introduced. This was the first OPC based platform-independent specification which is used to replace the traditional COM/DCOM with HTTP/SOAP and Web services technologies. Along with the platform independency, this has retained the features of the classic OPC DA specification [8].

OPC XML-DA was designed to work in internet access and enterprise integration. It is mainly implemented for platform independence to work on different operating systems such as embedded systems, and on a non-Microsoft platform. However, this specification is not a success due to high resource consumption; hence the performance is highly impacted.

3.5 OPC UA

OPC DA was the most successful interface of the Classic OPC standards; it allows the client system to read, write the current data in automation devices. This is widely used for the HMI and SCADA systems that are produced by different that uses different automation hardware. Additionally, there were other standards developed later, such as OPC A&E and OPC HDA designed for accessing the HMI and SCADA data. Thousands of products overall have successfully adopted OPC standards. Moreover, this has been used in many areas for which it has not been designed for. However, there are still a lot more areas where the manufacturers want to implement this OPC standard but could not accomplish that needs due to the communication constraints prevails due to the COM dependency. For instance, use of an embedded controller to transmit and receive data because of this platform dependency, OPC tried to move away from using the COM/DCOM technologies for communication and started looking for an alternate [9].

OPC XML-DA was the first approach of developed by OPC Foundation for a platform-neutral communication which maintains all the features of classic OPC standards. However, this Web service-based communication protocol did not meet the expectation of inheriting all the features of traditional OPC protocols without compromising in performance and also it had issues in the form of interoperability since this OPC specification uses the XML Web service stacks. Besides these issues, OPC member companies brought forward this specification due to its capabilities use complex data that was a limitation in Classic OPC standards.

Since the OPC XML-DA did not make an impression as per the expectation. The OPC Unified Architecture was developed as an ideal replacement for the existing COM-based specifications without losing any features or its performance. Moreover, this platform-independent system able to describe complex systems. Automation industries require a wide range of application to work from lowest embedded systems till the top-most system such as the SCADA and DCS and up to the MES and ERP systems were significant requirements of the OPC UA.

In order to satisfy the needs of the automation industries, OPC UA is built on different layers, as shown in Fig. 3. The fundamental components of OPC UA are the Transport layer and Data modelling layer. The transport layer defines the various types of mechanisms that were used to optimise the different use case of data transfer. The first version of the transport layer is Binary TCP protocol which is useful in terms of high-performance communication, and it has been mapped to the regularly used internet protocols such as XML, Web services and HTTP communications. Both the transport layer uses the message-based security models for Web services. The second layer of the OPC UA is the data modelling layer which defines the rules and base building blocks that is mandatory for data transmission in OPC UA. Data layer defines the entry points into the address space which the client could use to connect to the server in order to access the data from the server. This address space and base types are built in a hierarchy type as the base can extend the information of the model in a parent-child approach. The main advantage of the OPC UA concept is that the client can access the smallest piece of data with exposing the servers complete complex system.

OPC UA is built by using the following strategy initially, by defining the base information model of the system which can provide the complete detail about the framework that is utilised by the OPC UA. For accessing the server data, the client uses the address space to navigate through the server instances, and the base type is required while accessing the root of the hierarchies. OPC UA uses different layers of an information model that were defined by OPC, by other organisation or vendors, as shown in Fig.4.

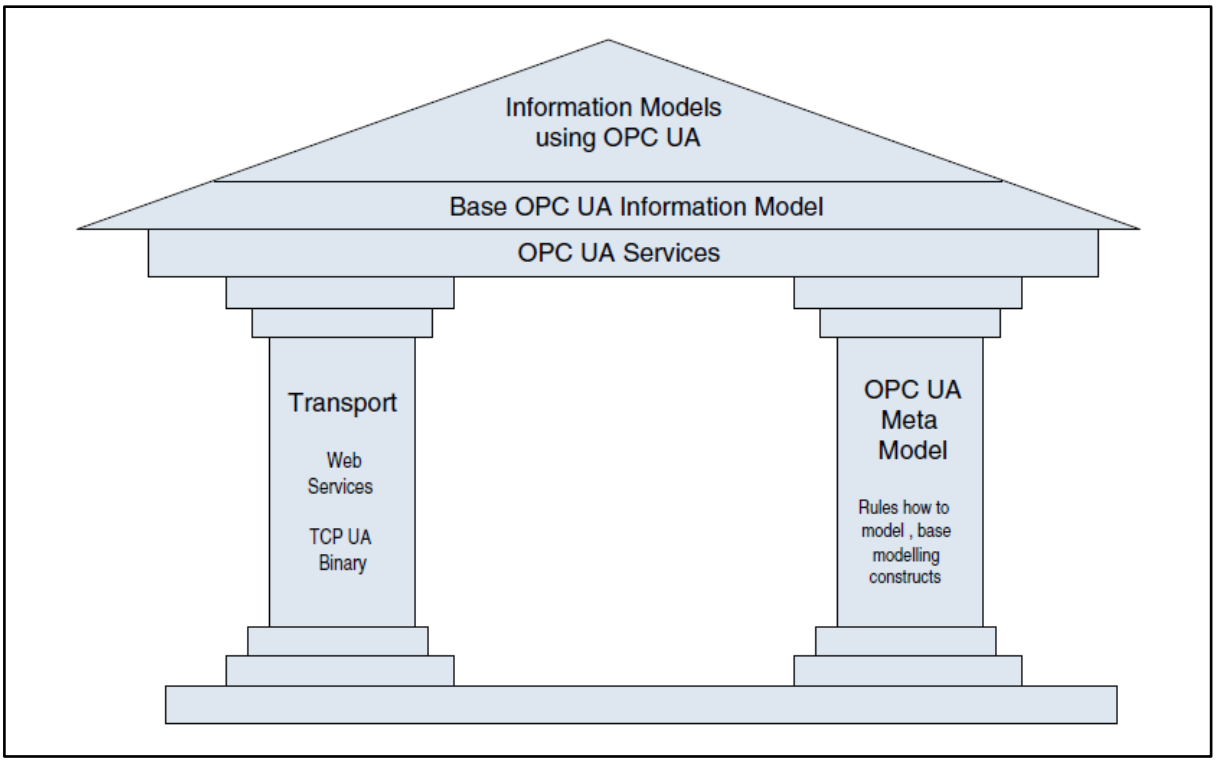

Figure 3. Overview of OPC UA [1]

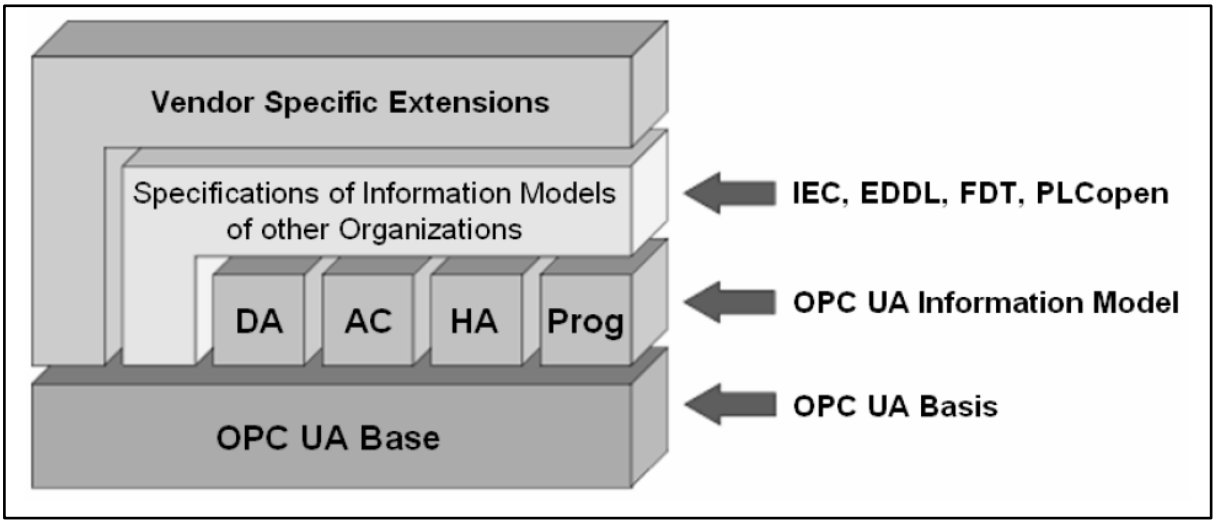

Figure 4. OPC UA layered architecture [1]

OPC UA covers all the successful features of the Classic OPC information model on the top of the OPC UA Base. Such information model includes the OPC DA, OPC HDA, OPC A&E and Programs for starting and manipulating specific program. Additional OPC UA enables other vendor-specific information models on the UA base or even on the top of the information model base such models include Field Device Integration (FDI) combining Electronic Device Description Language (EDDL), and Field Device Tool (FDT) both used to

describe, to configure, and to monitor devices and PLCopen, a standard for PLC programming languages.

The OPC UA uses the Node concept for the sending and receiving data, here in OPC UA Node is an entity that represents the information. Here the Nodes can be a variable, DataType, references, Objects, Method depending on their uses. It is merely a unique identifier that is used to represent an entity based on its use. Below table provides a detailed list of Node attribute that is used for achieving the OPC UA communication [10]. Current research uses the OPC UA Node and its attributes to collect and manipulate data for the desired outcome of the smart industries.

Table 1. OPC UA common attributes [1]

Attribute	DataType	Description
NodeId	NodeId	Unique Identifiers of a Node in an OPC UA and used to address the Node.
NodeClass	NodeClass	An enumeration for identifying the NodeClass of a Node such as Object or Method
BrowseName	QualifiedName	Identifies or name used in the Node for identification when browsing the OPC UA server. It is not localised
DisplayName	LocalizedText	Contains the Name of the Node that should be used to display the name in a user interface. Therefore, it is localised
Description	LocalizedText	This optional Attribute contains a localised textual description of the Node
WriteMask	UInt32	Is optional and specifies which Attributes of the Node are writable, i.e., can be modified by an OPC UA client
UserWriteMask	UInt32	Is optional and specifies which Attributes of the Node can be modified by the user currently connected to the server

The OPC UA generally uses an approach of client-server type of communication. The application that produces and exposes its information to other application interfaces are known as the UA server, and the application that consumes the information from the server is known as the UA client. Here the server and client exchange data by using the Node concepts, and

each Node ID is must for accessing the right information. This OPC UA protocol is configured to exchange data only between the UA server and the UA client.

4 Machine-to-Machine Communication

The primary objective of the OPC UA communication protocol is to provide the interface for machine-to-machine (M2M) communication. In general, OPC UA enables the M2M communication between the automation devices and PCs for HMI and SCADA technologies which is basically, reading and writing of data which accomplishes a simple task of On/Off type of conditions [11]. For instance, the OPC UA server collects data from the low-level automation devices such as PLC which provides the actual state of the I/O peripherals of the devices. This produced information is transferred to the HMI devices for monitoring purposes, and when the value reaches its threshold point, the user will send an input signal to the PLC I/O for actuating respective devices [12, 13].

Currently, available M2M protocols are instrumental in monitoring and switching actuator based on the data. However, this switching is only limited to a simple actuation rather than making any complex decision based on the received data. Hence, this research work considers on rethinking and enhancing the current M2M communication [14].

5 Advanced Machine-to-Machine Communication

Owing to the limitations prevailing on the low-level M2M communication, this research paper proposes a concept of Advanced M2M communication using the OPC UA protocols in smart industries. The Advanced M2M communication not only enables the machine to make its own decisions on the basis of received data but also it can influence the decision of the others machines by sending commanding signals to the machine. Moreover, the advanced M2M communication framework monitors the change in the data that have been received from the other machines when the data reaches its threshold it will make decisions and also can send commanding signals to the other machines for the desired outcome. For instance, a machine A sends the temperature reading of the tool to the machine B, and when it reaches the maximum temperature reading, machine B will send commanding signals to eject the workpiece and actuating signal for coolant on the machine A. This framework not only allows the control of its own action of the machine, and it can also influence the action of other machines made this framework to reach the next level of M2M communication.

This Advanced M2M communication is achieved using the OPC UA communication protocols. However, in general, OPC UA works on the basis on the client-server basis concept, whereas this proposed framework uses the concept of a client-client connection approach. In order to avoid such an exception, it needs some resolution. There is already an existing server-server concept available in OPC UA, which uses an approach of embedding server-client approach to map two servers together, and this framework also needs some approach similar to this. Hence, similar to server-server communication, this client-client framework also uses the concept of embedding two or more client together for transferring data among the client instances.

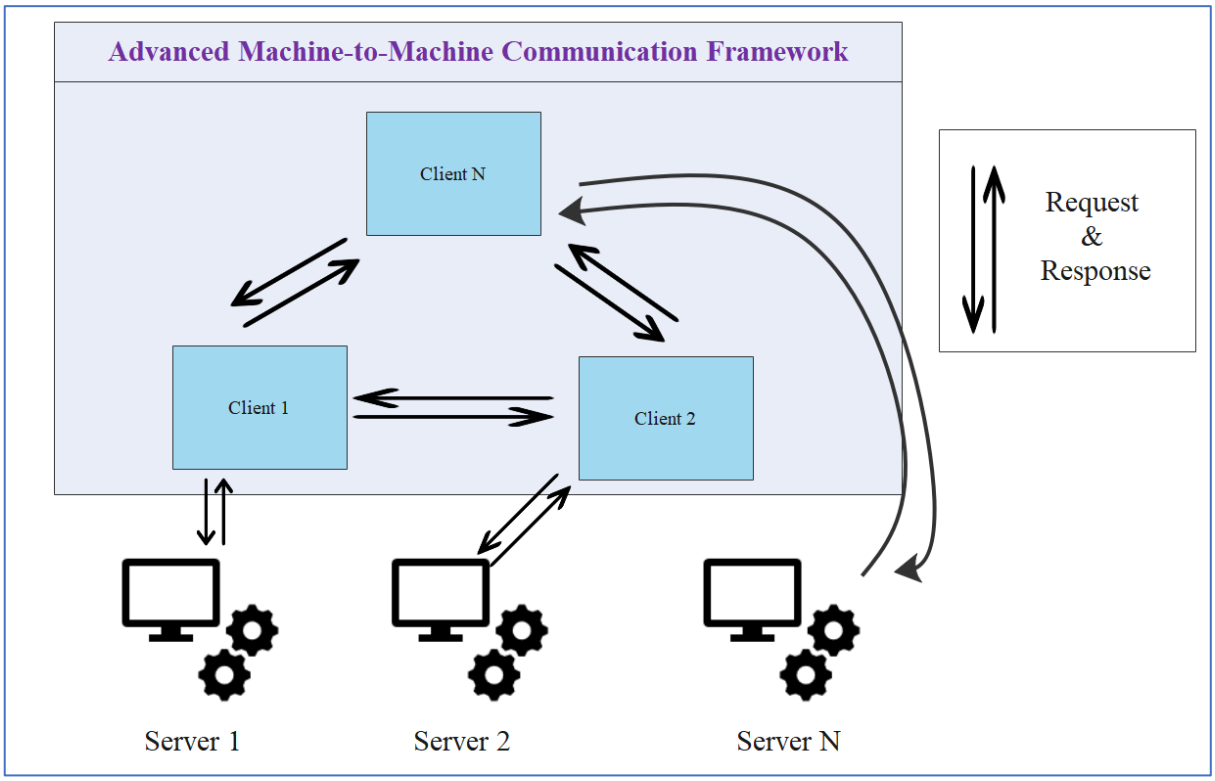

Figure 5. Advanced Machine-to-Machine communication framework

The model Advanced M2M communication framework as shown in Fig. 5, there is multiple UA server available for the machines, and each server has its own client instances within the framework which allows all the client instances to interact with one other without any issues [15, 16]. UA client is compatible to make changes in the server-side; this framework works in such a way that it can allow all the client to share and receive a response and request signals. However, the client-client communication would not be sufficient to produce Advance M2M communication. As it requires decision-making abilities of machine for further processing and these capabilities can be embedded within the framework, and it is made

available for all the client instance which could make it own decision and send signals to other machines. As shown in Fig. 5, each of the client instances is communicating with another client, and there are possibilities to share and receive data. Since the client services have the capabilities to manipulate the data on the server-side, the client-client mapping comes handy in Advance M2M communication. These client services run with its algorithm for decision making, which was created while initialising client instance. This framework allows the client to synchronise data simultaneous with other client instances.

This client-client communication is not limited to data transfer alone as this can also be used to start and stop a machine program embedded in the function method in OPC UA communications. This feature enables the developed framework to control operations of other machines along with comparing the data. This extraction of data and starting machine program is not limited few decisions as it has the capabilities to make multiple decisions based upon the analysing the data. As shown in Fig. 6, the client instances and framework synchronises on distributing data and also involves in starting a program on the server.

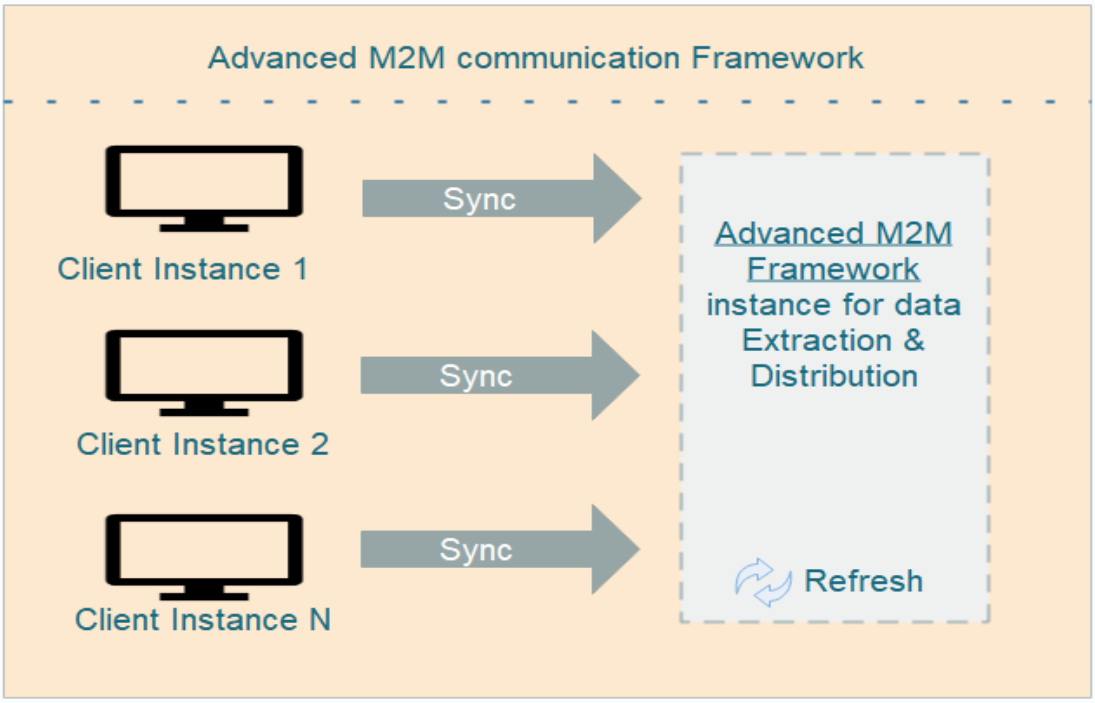

Figure 6. Communication of client and Advanced M2M communication framework

This advanced M2M Communication can be explained by using soccer team instances. Here the team captain, player and opponent player roles are related to the framework, client(Machine), and pre-occupied task. Initially, team captain receives the plan and objective of the game and comes up with a plan, and then he conveys the message to his player to

coordinate and collaborate to achieve the common goal. Similarly, this framework initially uses a set algorithm for data extraction. It synchronises the collected data with other interconnect client machines. Then these machines operate according to the algorithm since these client machines make multiple decisions and these decisions can influence the operation of other client machines, and that too can make decisions to complete to a common objective. As shown in the Fig. 7, initially client machine 1 operated to produce a favourable outcome, however, due to unfavourable obstacles such as rise in temperature, pressure or even completing the initial task by the machine, the client machine 1 need to send data to another machine to start the next task in the industry to achieve a common objective.

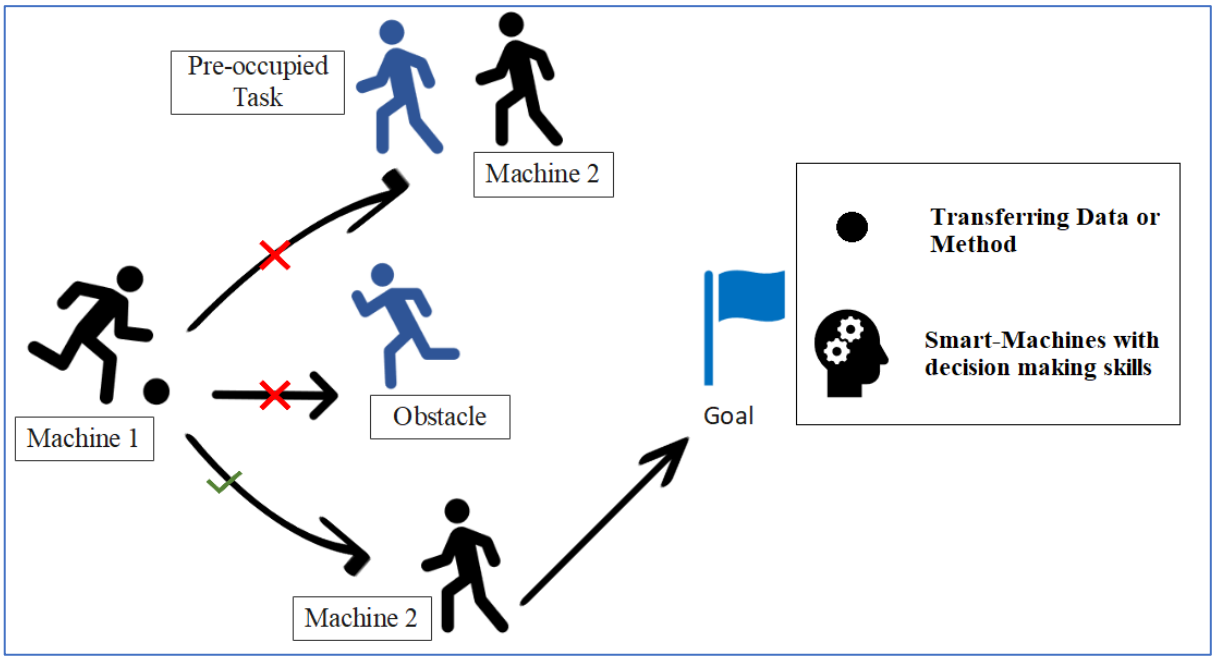

Figure 7. Advanced M2M communication explained using soccer game instance

6 Framework development

Working model of the framework is discussed in details as follows. The first step of the framework is to connect to the server. By using the URL of the server, the framework's client instance can connect. The initial requirement is to provide URL and device name for the localised reference once the information is provided the client instances are created automatically. By using the same method, the client can produce multiple instances of client service in a single framework. However, it has only one limitation that is related to the multiprocessing capabilities of the PCs. As shown in Fig. 8, by the simple use of URL can make the client services to communicate with the server devices remotely.

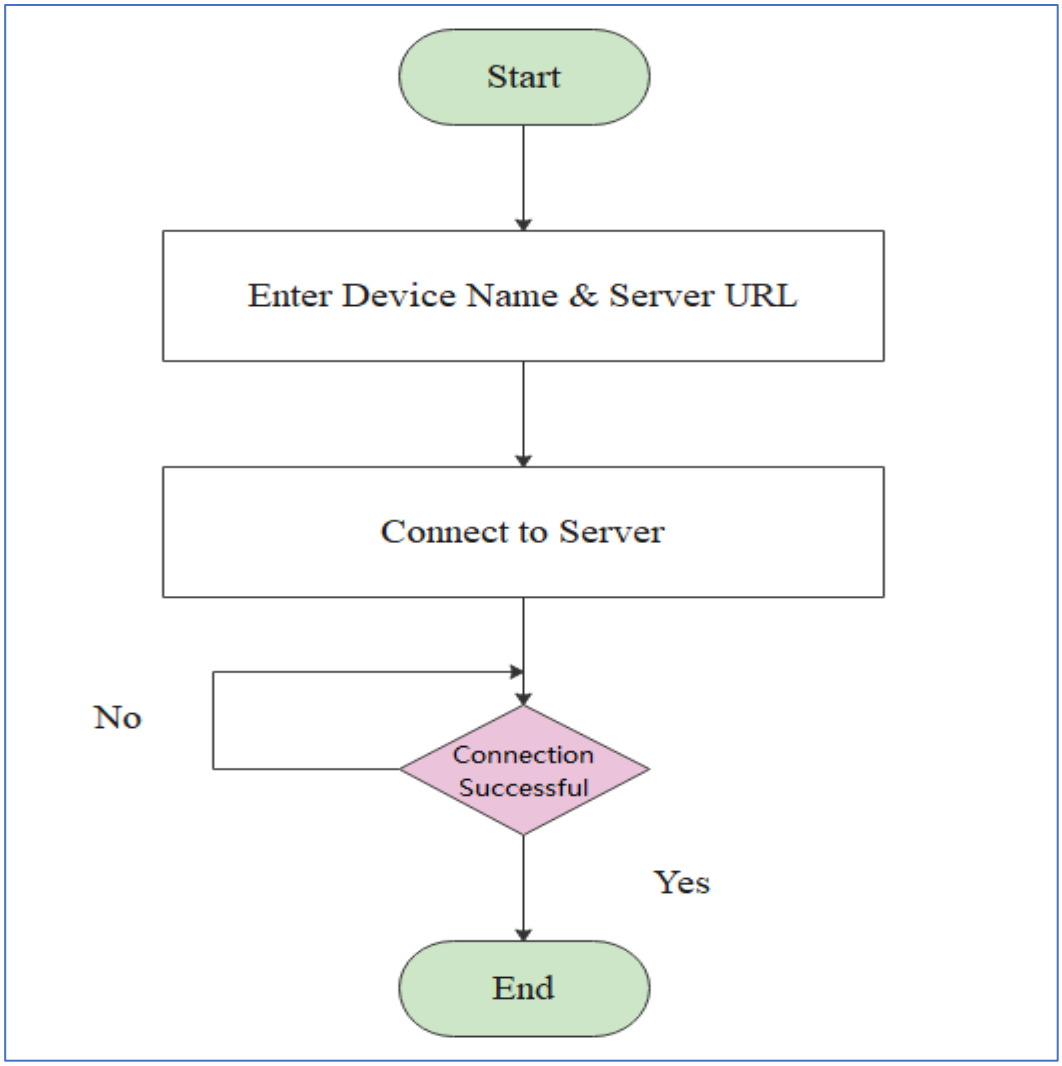

Figure 8. Flowchart for Server and Client communication

After initialising the client-server communication, the next step involves fetching the parent Node of the server machine. Node is an entity that contains complete data that is used for developing the OPC UA server. Once the server-client communication is established, the Node is used to fetch all the data linked to the server as shown in Fig. 9.

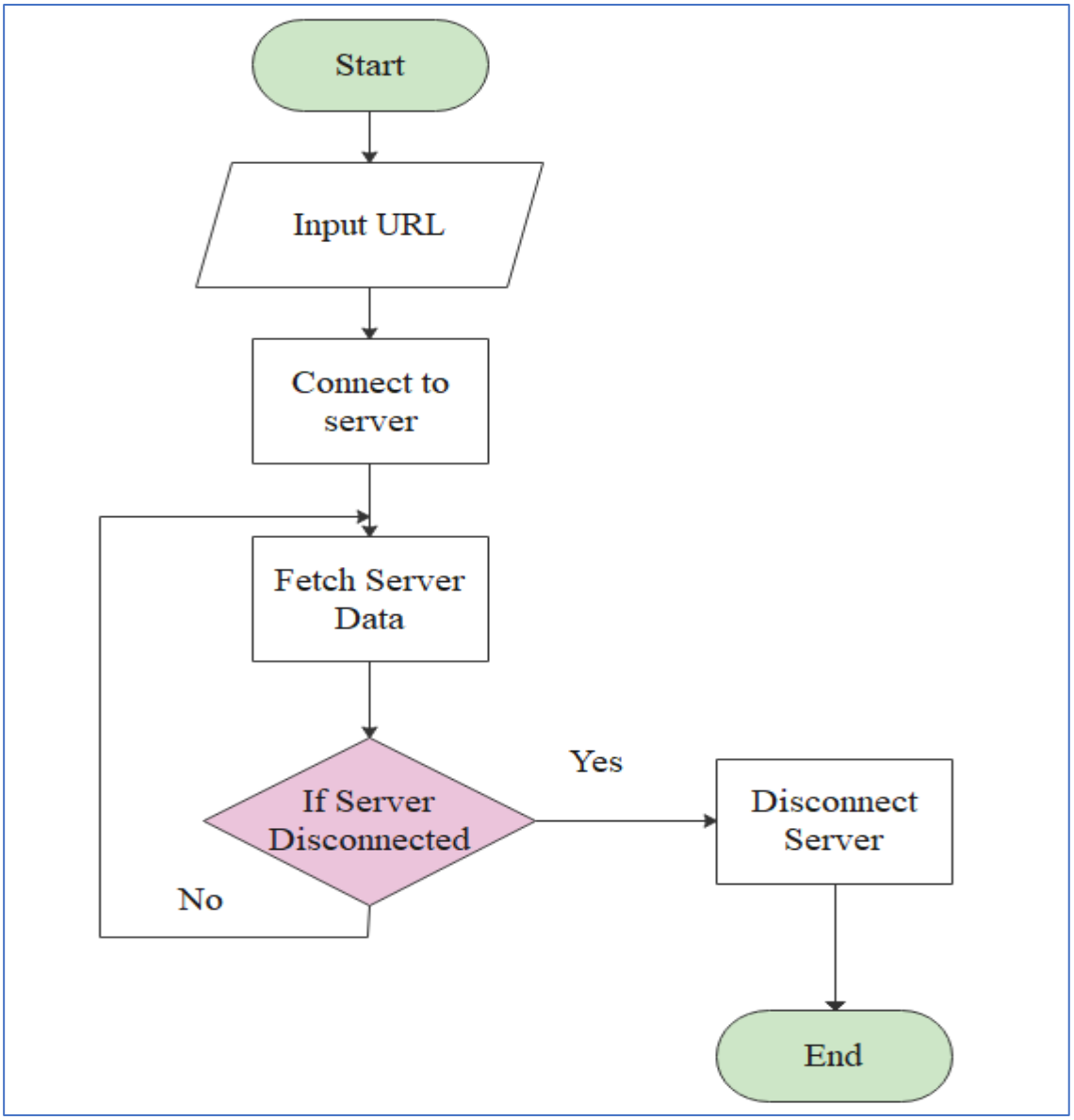

Figure 9. Flowchart for extracting OPC UA server Node data

The next task of this framework involves collecting the entire Node ID and its Class for the hierarchical ordering. This function starts its process by using the parent Node ID by using it, it can collects all the children class of this Node, and it is subjected to condition on verifying namespace of 2 (i.e. ns=2). This condition is used to extract only the variable types in the Node class and allowing it to ignore other Class types such as Methods, reference, and so on. By doing so, it collects all the Node ID and Browse-name of the node class variable, which is the crucial feature for Data access in terms of reading and writing. However, if in case Method Node class is required that can also be collected by changing the browse condition. As shown in Fig. 10, the function first fetches complete Node ID and then tries to evaluate with a condition for variable and Node ID that satisfies the condition and it will be

stored in a list so this Node ID can be used to fetch both the browse-name and variable Node ID.

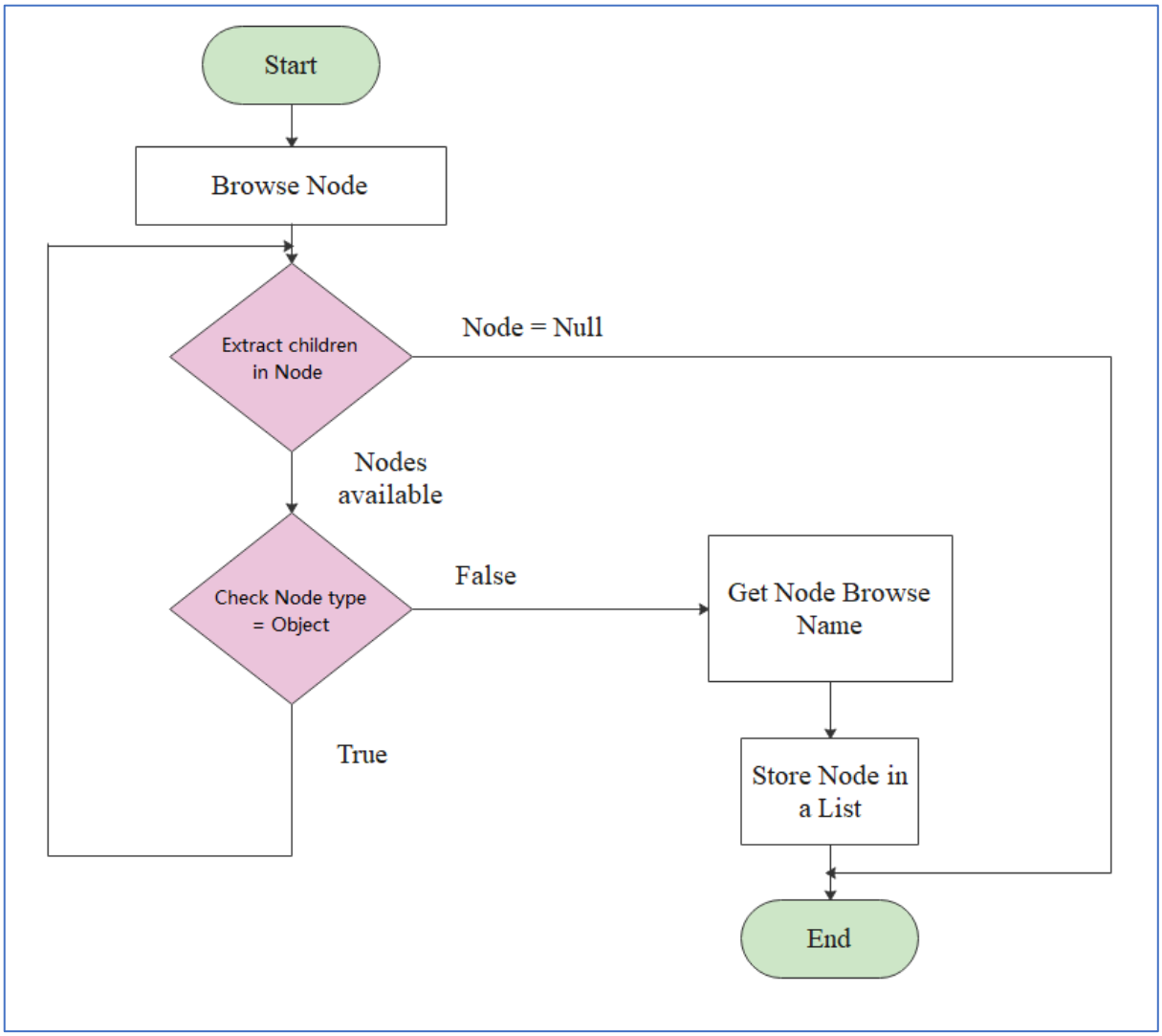

Figure 10. Flowchart for extracting child-node from OPC UA server

OPC UA client services use the address namespace for identifying the Node. This framework collects the variable Node for manipulating client services. Hence the Namespace space for the framework would "2" which is used for entities variable in Node object, and the identifiers would be "i" for the numeric integer, and the data type would be a string in order interact with other interfaces since this is the standard specification for OPC UA. For instance, the current server uses the address namespace as "ns=2" and identifier "i" which increments from "2" based on the number of the variable present. Further details about the namespace configuration are shown in Fig. 11 & 12.

Field	Description
Namespace Index	The index of the OPC UA Server namespace in which the address resides. If the index is 0, the entire *ns==<namespace index>;* clause will be omitted.
Type	The type of address. OPC UA supports the following four address types: i: A numeric address.* s: A string address. g: A GUID address. b: An opaque address (such as a byte string).
Value	The address that is formatted as a string.**

Figure 11. OPC UA address namespacing specification

Address Type	Namespace	Example
Numeric	2	ns=2;i=13
String	3	ns=3;s=Channel1.Device1.Tag1
GUID	0	g=C496578A-0DFE-4B8F-870A-745238C6AEAE
Opaque	2	ns=2;b=M/RbKBsRVkePCePcx24oRA==

Figure 12. OPC UA namespacing specification

The earlier function involves automatic data and address space collection. These collected data need to be processed in order to achieve the advanced M2M communication as well as it should perform regular functions such as transferring data. Prior to this framework development, client-client data is not possible; however, with embedding the multiple client instances in a single framework, made the possibilities to transfer data seamlessly. Initially, the below-illustrated function, as shown in Fig. 13, collects the target Node ID and Input ID of two different server machines (see Appendix B). And then the target node ID is assigned with the data value of the input node ID, which will automatically collect data from the other server by using client-client interaction. Not only, this framework assign output of machine A as input to machine B, it can also set user-defined values to the input of machine B, as shown in Fig. 14.

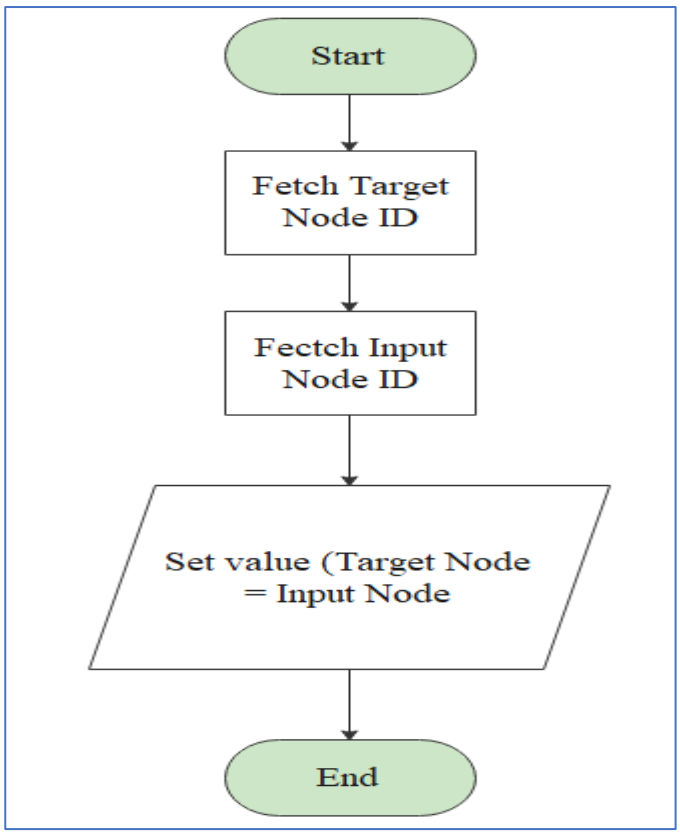

Figure 13. Flowchart for M2M communication to set input

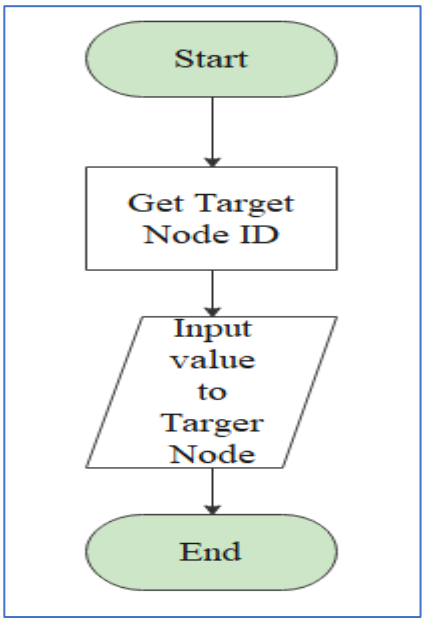

Figure 14. Flowchart for assigning the output of device A as input to device B

The final requirement of the Advanced M2M communication framework is to make decisions based on the input received from the other devices and send commanding signals to

other devices. This framework has capabilities to set four conditions based on the input values such conditions as follows:

- Equal to value
- Not Equal to
- Greater than
- Less than

These four conditions are used to evaluate the received input data from the other devices if the device input satisfies the set condition, it can produce three types of decisions, and it can influence it to command on its own-self as well as on other machines. Such conditions as follows:

- Disconnect server (self or on another server)
- Disconnect Input (self or on another machine)
- Change Input (self or on another machine)

As shown in Fig. 15, initially, the function evaluates to satisfy the condition, once the condition is satisfied, the client will send the commanding signal to self or another server based on the preset decisions made by a user. By using this framework, the user can collect, manipulate both data and methods on the server-side (see Appendix A).

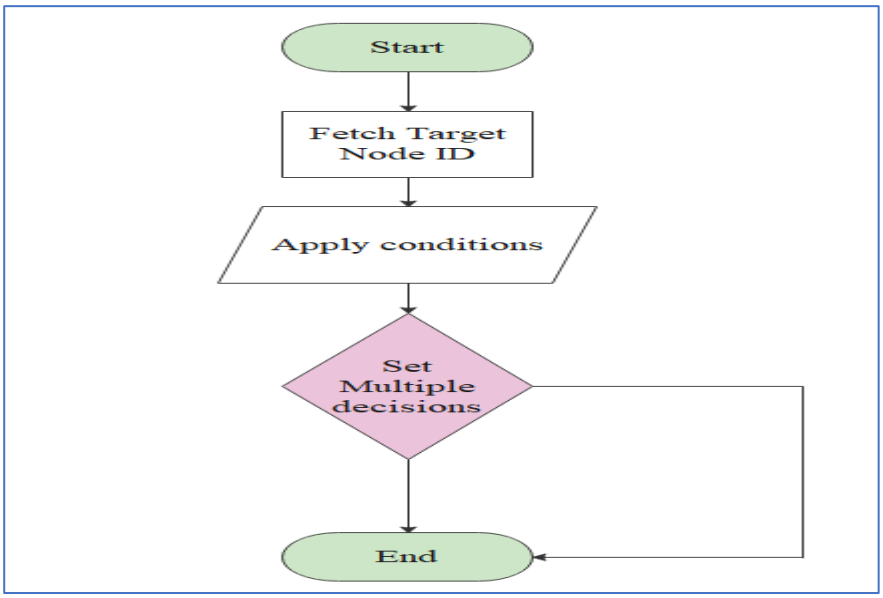

Figure 15. Flowchart for utilising multiple decisions in M2M communication framework

7 Experimental Application

To verify the effectiveness of the proposed framework, we have developed an OPC UA M2M communication framework, as shown in Fig. 16 Initially established two servers using OPC UA standards, and then the established servers are connected to the M2M framework using OPC UA client services. Within the framework, both the clients have the capabilities to send and receive data. On top of the client machine 1 has the capabilities to change the data of the client machine 2, and a client instance can make multiple decisions based on the data received from another client. This received data could be a completion of a process, or it could mean that a machine or its sensor device has reached its saturation set points. Additionally, here we have verified that any connection lost to Server 1 would not have any impact on another interconnected server. This experiment verifies the functioning of OPC UA framework and its possibilities of making client-client to communication and M2M communication in making multiple decisions.

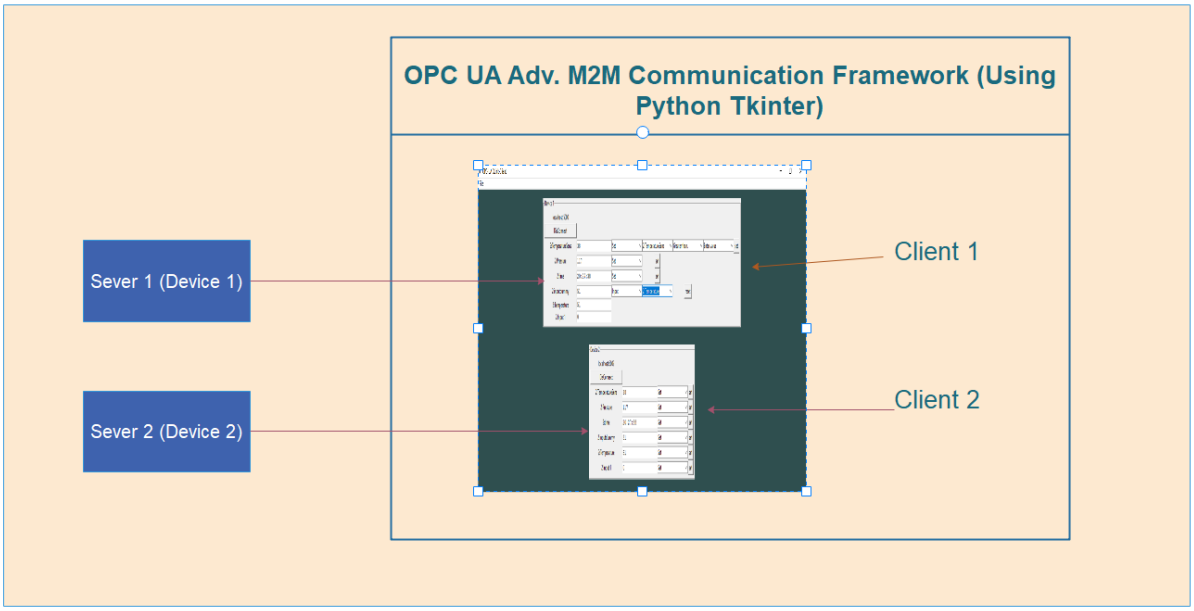

Figure 16. M2M communication experimental application

8 Future Scope of Advanced M2M

Advanced machine-to-machine communication can be further enhanced to communicate and control each machine available in the factory by using a concept of nested advanced m2m communication. This nested advanced m2m communication slightly an enhanced version of advanced m2m communication where it has an embedded server with its m2m communication framework which allows two or more m2m communication framework to share with each

other and that allows every machine in the factory is connected in a hierarchical order. As shown in Fig. 1. all the low-level machine servers are connected to the factory shop floor level framework, and this shop floor level server embedded frameworks are connected to factory level framework, and finally, all the factory level framework is connected to a centralised network where every machine are connected and controlled from a single network.

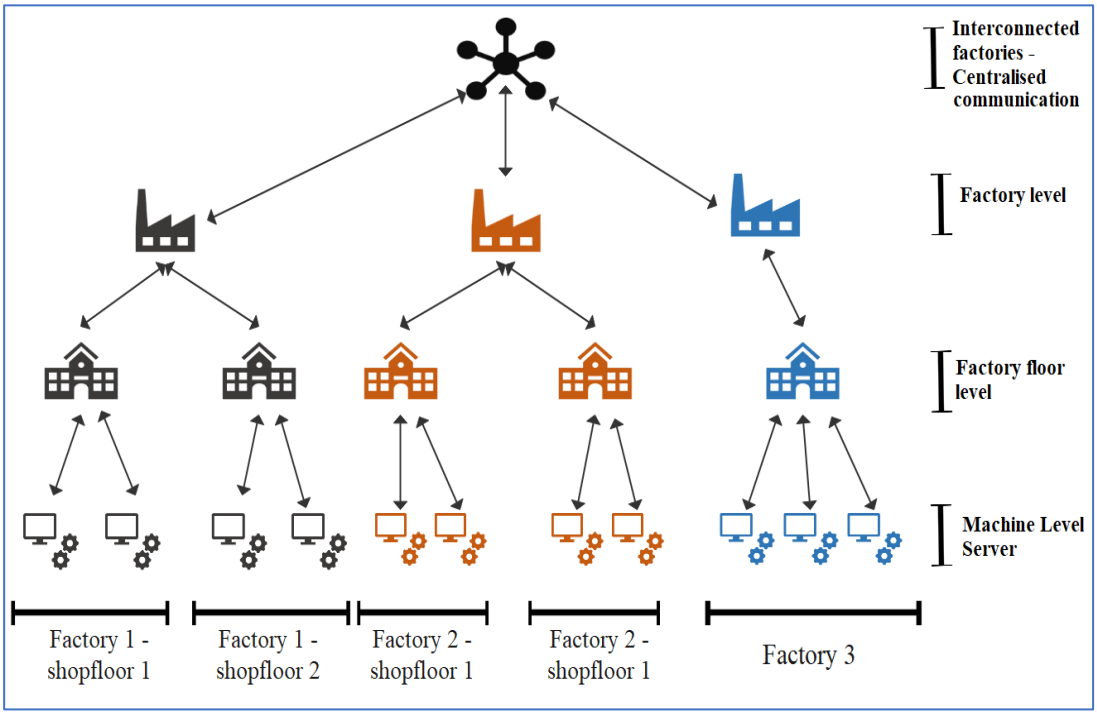

Figure 17. Advancement of M2M communication

Advanced machine-to-machine communication paves the way for advancement in two forms. The first advancement will be in the form of the Digital Twin (DT) which consists of a digital representation of physical systems, knowns as Physical Twin (PT), able to run simulations of the system lifecycle and actuate reflecting synchronously with PT, and vice-versa [17,18]. Here in Advanced M2M, the physical system represents the server that is connected to the physical machine, and the PT represents the client instances that were created within the framework, this created client services using the M2M framework contains the details of I/O of the machines, status, and it functions which is already a type of DT technology that could represent and control the physical entity. Still, this DT has much more capabilities to enhance in terms of delivering virtual entity for simulation and real-time monitoring [19, 20].

The second advancement of M2M communication is to improve its decision-making capabilities exponentially using the concept of machine learning [21]. Current analysis has

proved this framework capability to make multiple decisions based on the available information [22]. However, these decision-making capabilities scope much smaller than machine learning capabilities, along with the decision-making qualities of the framework this machine learning concept can enhance the training capabilities of machine such that it can track the activity of each machine and it can develop its strategy on completing any tasks [23].

9 Conclusion

This paper presents the current status and advancements of Machine-to-Machine communication in smart factories. Moreover, it emphasises on M2M communication using OPC UA standards. Here, the core concept of OPC UA, M2M communication, proposed advanced M2M communication concept, and its advancements was discussed in detail. This paper also describes the methodology for developing the Advanced M2M communication using OPC UA standards. Furthermore, this M2M communication uses a new concept called the client-client communication framework, and this framework had been introduced to achieve advanced M2M communication. Finally, an application was developed to prove the working of this two concepts, first being the advanced machine-to-machine communication to delivery commanding actions based on the received data and the other being the establishment of communication among numerous client-client instances using the M2M communication framework. However, still, these research activities are going to be active since this paper introduces another concept called nested M2M communication as a further enhancement for the current research.

10 References

[1] Mahnke, Wolfgang, Stefan-Helmut Leitner, and Matthias Damm. *OPC unified architecture*. Springer Science & Business Media, 2009, ISBN-10:9783642088421.

[2] OPC Foundation, *OPC UA Part 1 - Overview and Concepts RC 1.04.07 Specification,* http://www.opcfoundation.org/

[3] Imtiaz, Jahanzaib, and Jürgen Jasperneite. "Scalability of OPC-UA down to the chip level enables "Internet of Things"." In *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, pp. 500-505. IEEE, 2013.

[4] Schleipen, Miriam. "OPC UA supporting the automated engineering of production monitoring and control systems." In *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 640-647. IEEE, 2008.

[5] Schleipen, Miriam, Syed-Shiraz Gilani, Tino Bischoff, and Julius Pfrommer. "OPC UA & Industrie 4.0-enabling technology with high diversity and variability." *Procedia Cirp* 57 (2016): 315-320.

[6] Weyrich, Michael, Jan-Philipp Schmidt, and Christof Ebert. "Machine-to-machine communication." *IEEE Software* 31, no. 4 (2014): 19-23.

[7] Kritzinger, Werner, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihn. "Digital Twin in manufacturing: A categorical literature review and classification." *IFAC-PapersOnLine* 51, no. 11 (2018): 1016-1022, https://doi.org/10.1016/j.ifacol.2018.08.47.

[8] Zhou, Guanghui, Chao Zhang, Zhi Li, Kai Ding, and Chuang Wang. "Knowledge-driven digital twin manufacturing cell towards intelligent manufacturing." *International Journal of Production Research* 58, no. 4 (2020): 1034-1051.

[9] Drahoš, Peter, Erik Kučera, Oto Haffner, and Ivan Klimo. "Trends in industrial communication and OPC UA." In *2018 Cybernetics & Informatics (K&I)*, pp. 1-5. IEEE, 2018.

[10] Cupek, Rafal, Łukasz Gólczyński, and Adam Ziebinski. "An OPC UA Machine Learning Server for Automated Guided Vehicle." In *International Conference on Computational Collective Intelligence*, pp. 218-228. Springer, Cham, 2019.

[11] Tähkävuori, Ville. "Machine learning framework for OPC UA data (Industry 4.0)." (2019).

[12] Dietrich, Bastian, Jessica Walther, Matthias Weigold, and Eberhard Abele. "Machine learning based very short term load forecasting of machine tools." *Applied Energy* 276 (2020): 115440.

[13] Mühlbauer, Nikolas, Erkin Kirdan, Marc-Oliver Pahl, and Georg Carle. "Open-Source OPC UA Security and Scalability." In *2020 25th IEEE International Conference*

on Emerging Technologies and Factory Automation (ETFA), vol. 1, pp. 262-269. IEEE, 2020.
[14] Kožár, Slavomír, and Petr Kadera. "Integration of IEC 61499 with OPC UA." In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1-7. IEEE, 2016.
[15] Henssen, Robert, and Miriam Schleipen. "Interoperability between OPC UA and AutomationML." *Procedia Cirp* 25 (2014): 297-304.
[16] Cavalieri, Salvatore, Damiano Di Stefano, Marco Giuseppe Salafia, and Marco Stefano Scroppo. "A Web-based Platform for OPC UA integration in IIoT environment." In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1-6. IEEE, 2017.
[17] Amodu, Oluwatosin Ahmed, and Mohamed Othman. "Machine-to-machine communication: An overview of opportunities." *Computer Networks* 145 (2018): 255-276.
[18] Anpat, Sourabh, and Nirav Salot. "Machine to machine communication in a communication network." U.S. Patent Application 13/078,619, filed October 4, 2012.
[19] Williams, David, and Jeffrey Hill. "Machine learning." U.S. Patent Application 10/939,288, filed May 19, 2005.
[20] Tao, Fei, Jiangfeng Cheng, Qinglin Qi, Meng Zhang, He Zhang, and Fangyuan Sui. "Digital twin-driven product design, manufacturing and service with big data." *The International Journal of Advanced Manufacturing Technology* 94, no. 9-12 (2018): 3563-3576.
[21] Kritzinger, Werner, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihn. "Digital Twin in manufacturing: A categorical literature review and classification." *IFAC-PapersOnLine* 51, no. 11 (2018): 1016-1022.
[22] Leng, Jiewu, Hao Zhang, Douxi Yan, Qiang Liu, Xin Chen, and Ding Zhang. "Digital twin-driven manufacturing cyber-physical system for parallel controlling of smart workshop." *Journal of ambient intelligence and humanised computing* 10, no. 3 (2019): 1155-1166.
[23] NI, OPC. n.d. "NI OPC." NI OPC Manual. Accessed 2020. https://www.ni.com/pdf/manuals/375754a.html.

11 Appendix A: Complete code of advanced M2M framework

```
# UI development python framework -- Tkinter

from tkinter import *
from tkinter import ttk

# Python inbuilt framework -- Time

from time import sleep

# Python based OPCUA framework for Machine to Machine communication protocol

from opcua import Client

# Partial python framework for managing UI user clicks

from functools import partial

#Global variable declarations

global varName, varNode #OPC UA Node Name and Node ID stored globally within framework for client-client communicaiton
varNode=[] # Initialising list with "Null" value
varName=[] # Initialising list with "Null" value
btnids=[]  # Initialising list with "Null" value
btnids1=[] # Initialising list with "Null" value
btnids2=[] # Initialising list with "Null" value
listCondition=["Greater than.","Less than.","Equal to","not Equal to"] # List of preset conditions
listinput=["Enter value","Use input"] # List of preset Input conditions
listaction=["Disconnect input","Disconnect server","Set input"] # List of preset Decisions
serverlist=[] # Initialising list with "Null" value for Server localised name
serverAddress=[] # Initialising list with "Null" value for server address manipulation
abortInput=[] # Initialising list with "Null" value



#Framework for connecting client services

def testclient(url,port):

    """ Internal Function -- OPC UA client function for making a communication with the server.
    Multiple instance of client services can be made using this framework.
    Function takes 2 input argument URL & Port and returns the server Node ID"""

    go="opc.tcp://"+url+":"+port #Binding URL in a OPC UA Webservice format
    global client # Global defining a client for accessing it on other functions
    client=Client(go) #Calling Client function

    client.connect()
```

```
    print("connected")


    #Queue
    root = client.get_root_node() # Assigning the Client Node ID to a variable
    return root

# Python Tkinter framwork for UI function starts here...

mainwindow=Tk()

# mainwindow.geometry("1600x1000+0+0")
mainwindow.title("OPC-UA Quick-Client") # Defining Title
mainwindow.configure(bg='darkslategray') # Background Colour configuration
lbl_title = Label(mainwindow,text="Welcome to the OPC-UA Quick-
Client", fg="white", bg = 'darkslategray').pack # Assigning Title name and UI colour


# Tkinter Framework Functions

def calci(i,dropdown,nodeList,frame,nrow):
    """ Internal Function...
    Function for selection Condition or entering Input
    """
    print(i)
    global varName, varNode,btnids1,btnids,client # Globals
    ids=dropdown.get() #Fetching the row and column of Tkinter UI


    print(ids)
    btnids[i].destroy() #Destroys Tkinter widgets

    if ids=="Set":
        #Condition 1 apply conditions to the CLient for decision making

        global listCondition, listinput #Globals

        try:


            dropdown_con=ttk.Combobox(frame,value=nodeList) # Displays NodeList in a dropdown
            dropdown_con.current(0) #preselects the top one
            dropdown_con.grid(row=nrow,column=7) #Location on UI tkinter framework
            dropdown_con1=ttk.Combobox(frame,value=listCondition) #List of conditions
            dropdown_con1.current(0) #preselects the top one
            dropdown_con1.grid(row=nrow,column=8) #Location on UI tkinter framework
            dropdown_in=ttk.Combobox(frame,value=listinput) #List of Output Nodes
            dropdown_in.current(0) #preselects the top one
```

```
            dropdown_in.grid(row=nrow,column=9) #Location on UI tkinter framework

            valueset1=Button(frame,text='set',command=partial(setCondition,i,dropdown_con,dropdown_con
1,dropdown_in,frame)) #Widget tkinter button and call a function
            valueset1.grid(row=nrow,column=10) #Location on UI tkinter framework
            btnids1.append(valueset1) #Binds the button configuration in a list


        except Exception as e:
            print(e)
            #Exceptions to disconnect server.
            client.disconnect()



    else:

        try:
            # Input data is selected for Node

            dummy=[] #Dummy list
            dropdown_input=ttk.Combobox(frame,value=nodeList) # List of Nodes
            dropdown_input.current(0) #preselects the top one
            dropdown_input.grid(row=nrow,column=7) #Location on UI tkinter framework

            valueset1=Button(frame,text='set',command=partial(setInput,i,dropdown_input,frame,dummy))
#Widget tkinter button and call a function
            valueset1.grid(row=nrow,column=8) #Location on UI tkinter framework
            dummy.append(valueset1) #Binds the button configuration in a list



        except Exception as e:
            print(e)
             #Exceptions to disconnect server.
            client.disconnect()



def setInput(i,dropdown_input,frame,dummy,newrow=None):
    """ Internal Function
        Widget button press calls the SetInput function
    """

    global btnids2,varNode,varName, abortInput #Globals
    dummy2=[]
    if dummy:
        #Conditions for autodisconnecting function loop when the applied condition on UI is statisfied
        c_row=dummy[0].grid_info() #Extracts UI tkinter configuration of the widget
        newrow=c_row.get("row") #Extracts UI tkinter configuration of the widget
```

```
        dummy[0].destroy() #Destroy button configuration
        dummy.pop(0) # Removes from List
    ids=dropdown_input.get() # Extracts the select combobox value


    var=(varNode[varName.index(ids)].get_value()) #Variable to store the NodeID
    print(var)
    varNode[i].set_value(var) #Assigns the input of Node A to Node B

    if not ids in abortInput:
        #COndition to abort refresh loop

        valueset1=Button(frame,text='reset',command=partial(resetButton,i,dropdown_input,frame,dummy2)
) #Button function
        valueset1.grid(row=newrow,column=8) #UI location
        dummy2.append(valueset1) #Adds to a list of button configuration



    if not ids in abortInput:
        #COndition to abort refresh loop
        frame.after(1000,lambda:setInput(i,dropdown_input,frame,dummy,newrow)) #UI function refresh ev
ery 1 sec

    else:
        #After Abort destorys button and variables
        abortInput.remove(ids)
        dropdown_input.destroy()


def resetButton(i,dropdown_input,frame,dummy2):
    """ Internal Function...
    For ResetButton to reapply for condition
    """
    global varName,varNode,abortInput #Globals
    inputType=["Set","Input"] #Initial input condition either Set a condition or apply a output to a n
ode


    c_row=dummy2[0].grid_info() #Extracting UI widget configuration to a list
    nrow=c_row.get("row") #Extracting row ID

    ids=dropdown_input.get() #Extract the selected combobox value
    abortInput.append(ids) #Adds to list
    print(dummy2)
    dummy2[0].destroy() #Destroy widget button

    dummy2.pop(0) #Removes from list
    print(dummy2)
    print("done")
    dropdown=ttk.Combobox(frame,value=inputType) #Dropdown with a input type manual or device iput
```

```
    dropdown.current(0) #Initial input selected
    dropdown.grid(row=nrow,column=5) #UI Location
    valueset=Button(frame,text='set',command=partial(calci,i,dropdown,varName,frame,nrow)) #Button to
apply condition and calls function
    valueset.grid(row=nrow,column=7) #UI location
    btnids.append(valueset) #Widget configuration stored in a list



def setCondition(i,dropdown_con,dropdown_con1,dropdown_in,frame):
    """Internal FUnction...
    Applying conditions..
    """
    global btnids1, listaction, serverlist, varNode,varName #Globals
    btnids1[i].destroy() #Destroys the button pressed
    selection=dropdown_in.get() #Gets the value of selected combobox
    tempcondition=dropdown_con1.get() #Gets the value of selected combobox of condition
    c_row=dropdown_in.grid_info() #Fetches the configuration list
    newrow=c_row.get("row") #Fetches the rows configuration
    print(newrow)

    if selection=="Enter value": #FIrst condition
        dropdown_in.destroy() #Destroy the button
        enterInput=Entry(frame) #Input entry widget created on frame for inserting input value
        enterInput.grid(row=newrow,column=9) #UI Location
        dropdown_action=ttk.Combobox(frame,value=listaction) #Combobox for list of actions



        dropdown_action.current(0) #Selects initial Combobox option
        dropdown_action.grid(row=newrow,column=10) #UI Location
        dummy=[] #Dummy List
        dropdown_address=ttk.Combobox(frame,value=serverlist) #Combobox for list of servers
        dropdown_address.current(0) #Initial server is selected
        dropdown_address.grid(row=newrow,column=11) #UI Location
        #Button widget and calls a function
        valueset1=Button(frame,text='apply',command=partial(afteraction,i,dropdown_action,frame,dummy,
dropdown_con,enterInput,tempcondition,newrow,dropdown_address))
        valueset1.grid(row=newrow,column=12) #UI location
        dummy.append(valueset1) #Adds button configuration in a dummy list

        # valueset1=Button(frame,text='set',command=partial(applyentered_value,i,dropdown_con,enterInp
ut,tempcondition,frame))
        # valueset1.grid(row=newrow,column=11)
    else:
        dropdown_in.destroy() #Destroy button
        dropdownInputlist=ttk.Combobox(frame,value=varName)
        dropdownInputlist.current(0)
        dropdownInputlist.grid(row=newrow,column=9)
        dropdown_action=ttk.Combobox(frame,value=listaction)
```

```
        dropdown_action.current(0)
        dropdown_action.grid(row=newrow,column=10)
        dummy=[]
        dropdown_address=ttk.Combobox(frame,value=serverlist)
        dropdown_address.current(0)
        dropdown_address.grid(row=newrow,column=11)
        #Buttin to apply condition and calls a function
        valueset1=Button(frame,text='apply',command=partial(afteractionInput,i,dropdown_action,frame,d
ummy,dropdown_con,dropdownInputlist,tempcondition,newrow,dropdown_address))
        valueset1.grid(row=newrow,column=12)
        dummy.append(valueset1)

def afteractionInput(i,dropdown_action,frame,dummy,dropdown_con,dropdownInputlist,tempcondition,newrow
,dropdown_address):
    """ Internal Function...
    action after function applied
    """
    global listaction, varName #Globals
    selection=dropdown_action.get() #Extracts the value of combobox
    serverid=dropdown_address.get() #Extracts the value of combobox
    dropdown_address.destroy()
    dummy[0].destroy()
    dummy.pop(0)

    print("next")


    # ["Disconnect input","Disconnect server","Set input"]
    if selection=="Disconnect input":
        print("Disconnect input")
        #Calls input disconnect function
        valueset1=Button(frame,text='set',command=partial(applyenteredinput_value,i,dropdown_con,dropd
ownInputlist,tempcondition,frame,selection,dummy))
        valueset1.grid(row=newrow,column=13)
        dummy.append(valueset1)
        print("others")

    elif selection=="Disconnect server":
        print("Disconnect server123")
        print("others")
        #Calls function to disconnect server
        valueset1=Button(frame,text='set',command=partial(applyenteredinput_value,i,dropdown_con,dropd
ownInputlist,tempcondition,frame,selection,dummy))
        valueset1.grid(row=newrow,column=13)
        dummy.append(valueset1)

    else:
        print("input")
        #Function called to swap the Node ID of input to change function
```

```
        dropdown_toInput=ttk.Combobox(frame,value=varName)
        dropdown_toInput.current(0)
        dropdown_toInput.grid(row=newrow,column=13)
        dropdown_toAssign=ttk.Combobox(frame,value=varName)
        dropdown_toAssign.current(0)
        dropdown_toAssign.grid(row=newrow,column=14)
        valueset1=Button(frame,text='set',command=partial(applyinput,i,dropdown_toInput,dropdown_toAss
ign,frame,dummy))
        valueset1.grid(row=newrow,column=15)
        dummy.append(valueset1)
        print("set input")

def applyinput(i,dropdown_toInput,dropdown_toAssign,frame,dummy):
    """ Internal Function...
    Internally, calls function
    """
    global varName,varNode
    x1=dropdown_toInput.get()
    x2=dropdown_toAssign.get()
    val1=varNode[varName.index(x1)]
    val2=varNode[varName.index(x2)].get_value()
    print(val2)
    val1.set_value(val2)
    frame.after(1000,lambda:applyinput(i,dropdown_toInput,dropdown_toAssign,frame,dummy)) #Refesh the
UI loop


def applyenteredinput_value(i,dropdown_con,dropdownInputlist,tempcondition,frame,selection,dummy):
    """ Internal Function...
    Sets the input value
    """
    #["Greater than.","Less than.","Equal to","not Equal to"]
    global btnids1,varNode,varName,abortInput #Globals
    state=True #Flag for existing loop
    dummy[0].destroy()
    tempinputselect=dropdownInputlist.get()
    tempinputvalue=float(varNode[varName.index(tempinputselect)].get_value())
    ids=dropdown_con.get()
    var=float(varNode[varName.index(ids)].get_value()) #stores values in a variable
    if tempcondition=="Greater than.":
        #Condition 1 greater is selected
        if tempinputvalue<var:
            print("value is greater")
            state=False
            if selection=="Disconnect input":
                print("Disconnect input")
                abortInput.append(ids)

            elif selection=="Disconnect server":
                print("Disconnect server")
```

```
                stop_server(frame,client)
            else:
                print("input")

    elif tempcondition=="Less than.":
        #Condition 2 Less than is selected
        if tempinputvalue>var:
            print("value is less than")
            state=False
            if selection=="Disconnect input":
                print("Disconnect input")
                abortInput.append(ids)

            elif selection=="Disconnect server":
                print("Disconnect server")
                stop_server(frame,client)
            else:
                print("input")
    elif tempcondition=="Equal to":
        #Condition 3 Equal to is selected
        if tempinputvalue==var:
            print("value is equal")
            state=False
            stop_server(frame,client)
    elif tempcondition=="not Equal to":
        #Condition 4 Not equal to is selected
        if tempinputvalue != var:
            print("value is not equal")
            state=False
            stop_server(frame,client)
    else:
        print("nothing selected")
    if state:
        #Refesh until flag changes
        frame.after(1000,lambda:applyenteredinput_value(i,dropdown_con,dropdownInputlist,tempcondition
,frame,selection,dummy))



def afteraction(i,dropdown_action,frame,dummy,dropdown_con,enterInput,tempcondition,newrow,dropdown_ad
dress):
    """ Internal function...

    """
    global listaction #globals
    selection=dropdown_action.get()
    serverid=dropdown_address.get()
    dropdown_address.destroy()
    dummy[0].destroy()
    dummy.pop(0)
```

```
    # ["Disconnect input","Disconnect server","Set input"]
    if selection=="Disconnect input":
        print("Disconnect input")

    elif selection=="Disconnect server":
        print("Disconnect server")
    else:
        print("input")


    # ["Disconnect input","Disconnect server","Set input"]
    if selection=="Disconnect input":
        print("Disconnect input")
        #enables button and call function
        valueset1=Button(frame,text='set',command=partial(applyentered_value,i,dropdown_con,enterInput
,tempcondition,frame,selection,dummy))
        valueset1.grid(row=newrow,column=11)
        dummy.append(valueset1)

    elif selection=="Disconnect server":
        print("Disconnect server123")
        #enables button and call function
        valueset1=Button(frame,text='set',command=partial(applyentered_value,i,dropdown_con,enterInput
,tempcondition,frame,selection,dummy))
        valueset1.grid(row=newrow,column=11)
        dummy.append(valueset1)

    else:
        print("input")
        #enables button and call function
        dropdown_toInput=ttk.Combobox(frame,value=varName)
        dropdown_toInput.current(0)
        dropdown_toInput.grid(row=newrow,column=13)
        dropdown_toAssign=ttk.Combobox(frame,value=varName)
        dropdown_toAssign.current(0)
        dropdown_toAssign.grid(row=newrow,column=14)
        valueset1=Button(frame,text='set',command=partial(applyinputE,i,dropdown_toInput,dropdown_toAs
sign,frame,dummy,enterInput,tempcondition,dropdown_con))
        valueset1.grid(row=newrow,column=15)
        dummy.append(valueset1)
        print("set input")

def applyinputE(i,dropdown_toInput,dropdown_toAssign,frame,dummy,enterInput,tempcondition,dropdown_con
):
    """ Internal FUnction...
    """
    global varName,varNode
    tempinputvalue=float(enterInput.get()) #type cast entered value
```

```
    ids=dropdown_con.get()
    var=float(varNode[varName.index(ids)].get_value()) #extracts and stores data in a variable
    state=True #Flag for existing loop

    if tempcondition=="Greater than.":
        if tempinputvalue<var:
            state=False
            #Calls a function
            assignvalues(dropdown_toInput,dropdown_toAssign,frame)



    if state:
        #Refreshes loop
        frame.after(1000,lambda:applyinputE(i,dropdown_toInput,dropdown_toAssign,frame,dummy,enterInpu
t,tempcondition,dropdown_con))


def assignvalues(dropdown_toInput,dropdown_toAssign,frame):
    """ Internal Function"""
    global varName,varNode
    x1=dropdown_toInput.get()
    x2=dropdown_toAssign.get()
    val1=varNode[varName.index(x1)]
    val2=varNode[varName.index(x2)].get_value()
    print(val2)
    val1.set_value(val2)
    frame.after(1000,lambda:assignvalues(dropdown_toInput,dropdown_toAssign,frame))



def applyentered_value(i,dropdown_con,enterInput,tempcondition,frame,selection,dummy):
    """ Internal Function"""
    #["Greater than.","Less than.","Equal to","not Equal to"]
    global btnids1,varNode,varName,abortInput
    state=True
    dummy[0].destroy()
    tempinputvalue=float(enterInput.get())
    ids=dropdown_con.get()
    var=float(varNode[varName.index(ids)].get_value())
    if tempcondition=="Greater than.":
        if tempinputvalue<var:
            print("value is greater")
            state=False
            if selection=="Disconnect input":
                print("Disconnect input")
                abortInput.append(ids)

            elif selection=="Disconnect server":
                print("Disconnect server")
                stop_server(frame,client)
```

```
            else:
                print("input")

    elif tempcondition=="Less than.":
        if tempinputvalue>var:
            print("value is less than")
            state=False
            stop_server(frame,client)
    elif tempcondition=="Equal to":
        if tempinputvalue==var:
            print("value is equal")
            state=False
            stop_server(frame,client)
    elif tempcondition=="not Equal to":
        if tempinputvalue != var:
            print("value is not equal")
            state=False
            stop_server(frame,client)
    else:
        print("nothing selected")
    if state:
        frame.after(1000,lambda:applyentered_value(i,dropdown_con,enterInput,tempcondition,frame,selec
tion,dummy))


def receive(nodeList,nodeID,frame,flag):
    """ Internal function...

    """


    inputType=["Set","Input"]

    try:

        global client
        global btnids1,btnids
        for i in nodeList: #Creates a list of Node in a parent node

            #Check every node progressively
            indexnumber=nodeList.index(i)
            Mylabel=Label(frame,text=i)
            nrow=10+nodeList.index(i)
            Mylabel.grid(row=nrow,column=0)



            if flag:
                dropdown=ttk.Combobox(frame,value=inputType)
                dropdown.current(0)
```

```
                dropdown.grid(row=nrow,column=5)
                valueset=Button(frame,text='set',command=partial(calci,nodeList.index(i),dropdown,node
List,frame,nrow))
                valueset.grid(row=nrow,column=7)
                btnids.append(valueset)



            i=Text(frame,height=1,width=20)
            i.grid(row=nrow,column=1,columnspan=3)
            i.delete('1.0',END)
            i.insert(END,nodeID[indexnumber].get_value())
        flag=FALSE


        Mylabel.after(1000,lambda:receive(nodeList,nodeID,frame,flag))

    except Exception as e:
        print(e)
        client.disconnect()

def browse_name(node):
    """ Internal function...
        THis function extracts the list of node available in a server.
        By applying a proper if condition this can used to extract selective node type from server"""
    global varName, varNode
    for childId in node.get_children(): #iterates through every single node of parent class
        ch = client.get_node(childId) #gets the child node ID


        if str(ch.get_node_class()) == 'NodeClass.Object': #IF object node recursive as a parent node
            browse_name(ch)
        elif str(ch.get_node_class()) == 'NodeClass.Variable' and 'ns=' in str(ch): #Looks for variabl
e node with namespace
            try:
                app=str(ch.get_browse_name()).replace("QualifiedName(","")



                papp=app.replace(')','')


                #print(ch.get_value(),"and",papp)



                varName.append(papp) #stores node name in a list
                varNode.append(ch) #stores node ID in a list
                #print(varNode[0].get_value())
```

```
            except ua.uaerrors._auto.BadWaitingForInitialData:
                pass

    return varName,varNode

def stop_server(frame,client):
    """Internal function..."""
    #server disconnection
    client.disconnect()
    print("Server disconnected")
    frame.destroy()


def start_server(top,xyz,e1,e2):
    """Internal function"""
    #initiating server
    global serverAddress
    y=e1.get()
    z=e2.get()
    top.destroy()
    x=y+":"+z #OPC UA URL
    serverAddress.append(x)
    frame=LabelFrame(mainwindow,text=xyz,padx=5,pady=5)
    frame.pack(padx=15,pady=15)
    New3=Label(frame, text=x).grid(row=2,column=0)
    Dis_one=Button(frame,text="DisConnect",width=20,command=lambda:stop_server(frame,client))
    Dis_one.grid(row=6,column=0)

    # Displays.insert('1.0',x)

    root=testclient(y,z)
    nodeList,nodeID=browse_name(root)
    flag=True
    receive(nodeList,nodeID,frame,flag)



def stop():
    """Internal function"""
    global client
    client.disconnect()
    print("Disconencted from server")


def start(top,e2):
    """Internal function"""
    global serverlist
    xyz=e2.get()
```

```
    serverlist.append(xyz)
    top.destroy()
    top=Tk()
    Displays=Text(top,height=1,width=20)
    Displays.grid(row=0,column=0,columnspan=3)
    Displays.insert('1.0',xyz)

    e1=Entry(top)
    e1.grid(row=4,column=2)
    New1=Label(top, text="server").grid(row=4,column=0)
    e2=Entry(top)
    e2.grid(row=5,column=2)
    New2=Label(top, text="PortNumber").grid(row=5,column=0)

    one=Button(top,text="Connect",width=20,command=lambda:start_server(top,xyz,e1,e2))
    one.grid(row=6,column=4)




def create_new():
    """Internal function...
    Initial framework without client instance"""

    top=Tk()
    e2 =Entry(top)
    e2.grid(row=4,column=2)



    #Button press
    New1=Label(top, text="Device Name",width=20).grid(row=4,column=0)

    one=Button(top,text="Create",width=20,command=lambda:start(top,e2))
    one.grid(row=4,column=4)


#Tkinter framework menus
my_menu =Menu(mainwindow)
mainwindow.config(menu=my_menu)
file_menu=Menu(my_menu)
my_menu.add_cascade(label="File",menu=file_menu)
file_menu.add_command(label="New Tool",command=create_new)
file_menu.add_separator()
file_menu.add_cascade(label="Exit",command=mainwindow.quit)


mainwindow.mainloop()

#Tkinter framework ends
```

12 Appendix B: Important code Modules

Below Function is used to extract the node list from the parent Node

```
def browse_name(node):
    """ Internal function...
        THis function extracts the list of node available in a server.
        By applying a proper if condition this can used to extract selective node type from server"""
    global varName, varNode
    for childId in node.get_children(): #iterates through every single node of parent class
        ch = client.get_node(childId) #gets the child node ID


        if str(ch.get_node_class()) == 'NodeClass.Object': #IF object node recursive as a parent node
            browse_name(ch)
        elif str(ch.get_node_class()) == 'NodeClass.Variable' and 'ns=' in str(ch): #Looks for variabl
e node with namespace
            try:
                app=str(ch.get_browse_name()).replace("QualifiedName(","")



                papp=app.replace(')','')


                #print(ch.get_value(),"and",papp)



                varName.append(papp) #stores node name in a list
                varNode.append(ch) #stores node ID in a list
                #print(varNode[0].get_value())



            except ua.uaerrors._auto.BadWaitingForInitialData:
                pass

    return varName,varNode
```

13 Appendix C: NI OPC Server-Client communication flow

Below flow chart shows the communication between the NI OPC server and OPC client to the different devices and different OPC protocols. The middle green block represents the NI OPC application which includes Client and Server. NI OPC server has capabilities to three different tasks such as

1) Create an OPC DA server for Automation devices

2) Bridging/Tunnelling different OPC specific protocols

3) Create simulation servers

Furthermore, this OPC Quick client has capabilities to connect to the NI OPC server created using the above three features and also it can connect directly to the OPC DA servers.