University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Edge and cloud computing for automated data processing and visualisation in an IIoT enabled production facility

Andrew Woon Yong Tian

September 23, 2019

Abstract

A steady increase in the usage of digital technologies and automated devices in industrial facilities, in particular production facilities, has led to a substantial increase in the amount of data generated that needs to be processed and analysed. Therefore, this project was undertaken in collaboration with an industry partner to develop a software solution that would be placed on the edge of a network of devices, acting as a gateway for data between the devices and a centralised system or database. The resulting software solution, also known the Edge Connector, utilised the OPC-UA and MQTT communication protocols to receive and transmit data respectively from OPC-UA devices. The system was able to satisfy all functional requirements set by the industry partner and was delivered with no issues.

Acknowledgements

I would like to give my thanks to my academic supervisor, Dr Cao Qi, and my industry supervisor, Michael Entendez Labastida for their support and guidance throughout the duration of this project.

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: Andrew Woon Yong Tian Date: 01 April 2020

Contents

1 Introduction

This chapter includes the primary motivations of the project as well as a brief description of the project itself and it's objective.

1.0.1 Motivation

With the ever-increasing usage and proliferation of digital technology in industrial, commercial and consumer applications, allowing for automation of manufacturing and production, the next industrial revolution, also known as Industry 4.0, heavily involves the introduction of Cyber Physical Systems, which refers to the connecting of virtual space with the physical reality, to develop smart factories. The core concept of Cyber Physical Systems that allows for this is also known as "3C", which refers to "...computation, communication, control, to achieve collaborative and real-time interaction between the real world and the information world..." (Cheng et al. 2016).

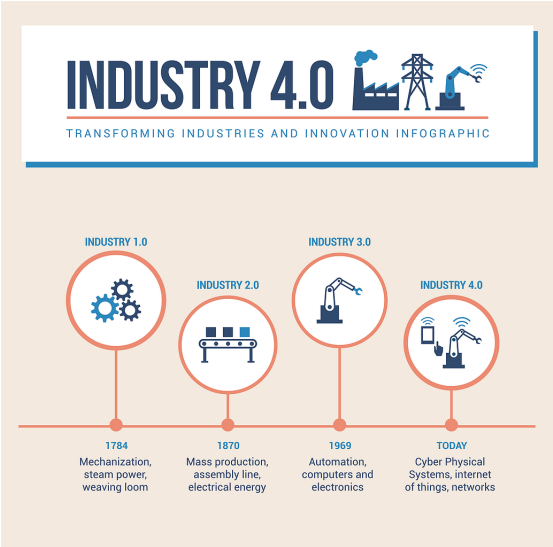


Figure 1.1: *Timeline to Industry 4.0 (*Industry 4.0 Timeline *n.d.).*

With the growing usage of computers and automated technologies, there also comes an exponential growth in the amount of data generated. These data in turn need to be interpreted and analysed in order to generate meaningful results that can be used for the benefit of the organisation. However, before data analysis can even take place, the proper recording, monitoring and storing of data is required in order to generate a data set suitable for analysis.

1.0.2 Aim

In collaboration with the Advanced Remanufacturing and Technology Centre (ARTC), led by the Agency for Science, Technology and Research (A*STAR), the principal aim of the project is to incorporate cloud computing, networking technologies and IoT principles to develop a system that will allow for the connection of devices within an existing production facility to a centralised network. This would subsequently allow for remote access to data generated by the devices, making it easier to perform data analytics or management of devices within the facility without needing physical access.

This project in particular will be focusing on the development of a data processing system to allow for transmission of data from devices within a production facility to a centralised system hosted on Microsoft Azure cloud, allowing for the data to be accessed remotely. As the system will need to handle different protocols, it needs to be able to translate data handled into a common format to facilitate communication between the device and the cloud database. The system also needs to be robust, with the ability to be deployed on devices running on various operating systems, while also allowing for a high level of usability with the ability to configure parameters for data transmission or receipt. As the system will be implemented in industrial devices, it is imperative that it can be run as a background service and have the ability to automatically restart and continue previous operations in the event the device has to restart.

1.0.3 Dissertation Outline

This chapter covered the motivation behind this project as well as it's overall objective and the potential benefits provided by the development of the system. The remainder of the dissertation will cover the development, implementation and evaluation of the system and has been structured into 6 additional chapters as follows:

- Chapter 2 - Background provides additional information regarding concepts, protocols or topics relevant to the project.
- Chapter 3 - Requirements details the functional and non-functional requirements of the data processing system.
- Chapter 4 - Design discusses the design architecture and decision-making process behind certain design choices for the data processing system.
- Chapter 5 - Implementation details the final design of the data processing system.
- Chapter 6 - Evaluation discusses evaluations and testing done to determine the effectiveness or success of the data processing system or justification for certain components of the system.
- Chapter 7 - Conclusion provides a summary of the results of the project, a reflection on the work done as well as consideration for future work that can be done to improve on the project.

1.0.4 Key Concepts

These are some of the key concepts that will be looked at and covered throughout the remainder of this dissertation.

Industry 4.0 With the introduction of the digital and automation technologies for industrial applications during the Digital Revolution in the late 19th Century, there has been a steady shift towards a new industrial revolution, also known as Industry 4.0. Continuing the Digital Revolution's trend of incorporating technology to improve the efficiency of industry, Industry 4.0 refers to the leveraging of digital technology such as cloud computing, data analytics and implementation of Internet of Things (IoT) systems.

Big Data With an increase in the usage of digital technology, there also comes an increase in the amount of data generated and subsequently an increasing need for systems that allow for the access, management and analysis of said data. In particular, the ability to access and analyse data remotely without physical access to the machine. This would allow for better management of supply chains and increase potential for refining industrial processes, increasing productivity while decreasing wastage of time and resources.

Industrial Internet-of-Things (IIOT) A computing concept that refers to the connection of sensors, instruments, industrial machines and other devices to specialised applications and the Internet or a centralised network. This allows for better data collection, management and analysis while also providing the potential for increased remote control over devices within the network and reduced need for physical access.

Cloud Computing Allowing for the provision of computing resources on-demand, cloud computing increases the availability of data storage and computing power while reducing the need for active management, allowing for increased productivity and lowered costs.

Edge Computing An improvement on the concept of Cloud Computing, edge computing involves placing data storage and computational systems closer to where it is required, increasing response times and limiting bandwidth usage from having to transmit data through multiple systems or networks. This in turn also reduces the demand for cloud computing resources, further lowering costs and enhancing productivity of devices connected to edge systems.

2 Background

This chapter includes a brief description of system architectures and protocols that will be relevant to this project as well as provide further details as to what is expected as the end product for this project.

2.1 Machine-to-machine (M2M) Protocols

There are a large number of protocols that can be utilised for M2M communications. However, for industrial applications, there are three major groups of protocols that are commonly used for M2M communications:

1. Service-oriented Architecture
2. Representational State Transfer (REST) Architecture
3. Message-oriented Protocols

2.1.1 Service-oriented Architecture (SOA) Protocols

SOA protocols are protocols that are used "... in industrial automation systems to exchange soft real-time data, for instance between programmable logic controllers and Supervisory, Control and Data Acquisition (SCADA) systems." An example of a SOA protocol commonly used for industrial applications would be the Open Platform Communications Unified Architecture protocol.

Open Platform Communications Unified Architecture (OPU-UA) is the successor to the Open Platform Communications (OPC) series of inter-operability standards and specifications, which were designed to facilitate the secure and reliable exchange of data among industrial devices. However, while the OPC series had separate products with specialised functionalities depending on the type of data access required, the OPC-UA platform integrates all these previously existing functionalities as a singular platform independent service-oriented architecture, allowing for multiple forms of data access with a single set of services. In addition, the OPC-UA specification allows for aggregating of servers, such that multiple OPC-UA servers running on separate devices can be aggregated by one OPC-UA server, allowing for unified access to data from multiple different devices. As such, this makes the OPC-UA platform suitable for industrial applications as industries tend to utilise different devices operating on different systems, which normally would make data collection or analysis tedious or difficult due to the need to develop separate communication protocols for each device, whereas the OPC-UA platform already acts as a standardised communication platform that can work on multiple platforms and systems, reducing the need to manage multiple different platforms and also allowing for easier collection, management and analysis of data from a variety of industrial devices (Lehnhoff et al. 2012).

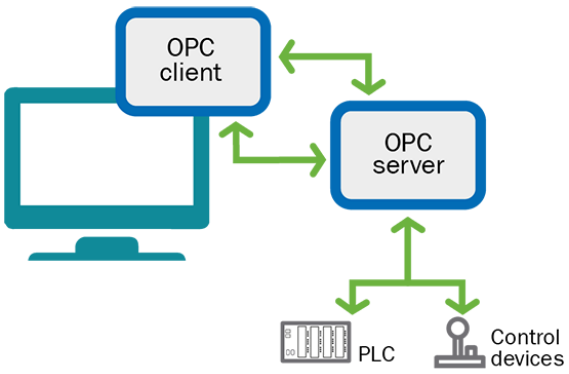


Figure 2.1: *OPC Client-Server design architecture. Upon Client request, Server responds by retrieving relevant data from the relevant devices (*OPC and OPC UA explained *n.d.).*

2.1.2 Representational State Transfer (REST) Architecture Protocols

REST protocols are protocols that follow the REST architecture style, which "...defines constraints to the used components, connectors and data elements." (Durkop et al. 2015) An example of a REST protocol would be the Constrained Application Protocol (CoAP), which was designed specifically for use in constrained networks and sensor networks.

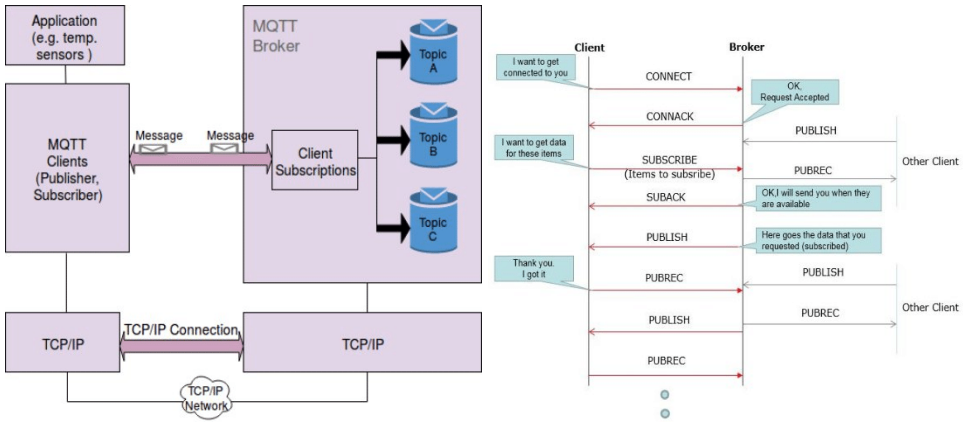
Constrained Application Protocol (CoAP) is a specialised web transfer protocol designed for use with constrained nodes and networks and is a protocol suited for M2M applications, providing support for service and resource discovery as well as a request / response interaction model between application endpoints. Furthermore,as part of its design for use in constrained environments, the CoAP provides multi-cast support, has low overhead and easily interfaces with HyperText Transfer Protocol (HTTP), allowing for simplified integration with the Web (Shelby et al. 2014).

2.1.3 Message-oriented Protocols

Message-oriented protocols are protocols that "...supports the asynchronous data transfer between distributed systems." (Durkop et al. 2015) These protocols typically use message transfer agents which are specialised to transfer and route messages to their destination, essentially acting as middlemen between senders and receivers. An example of a message-oriented protocol suited for industrial applications could be the Message Queuing Telemetry Transport protocol.

Message Queuing Telemetry Transport (MQTT) protocol is a message transport protocol relying on the publish-subscribe pattern, where senders of messages are called publishers while recipients are called subscribers. In the publish-subscribe pattern, devices who act as publishers do not sent message directly to specific recipients, but rather categorise messages into specific classes, which can be subscribed to by other devices within the network, who will then receive the messages from those specific classes. (*MQTT Version 5.0* 2019). In addition, with publish-subscribe pattern allowing for the decoupling of the publisher and subscriber, both parties can continue to operation independently of one another, with messages being able to be retained - by the subscriber - and transmitted - by the publisher - once the other party becomes available again. Finally, in a conventional client-server architecture, a client has to request for a resource before receiving a response from the server. However, with a publish-subscribe pattern, the need for

polling is eliminated as data is published on at regular specified intervals, constantly updating the data received by the subscriber. This potentially allow for potentially faster operations or processing of data as the client or subscriber no longer needs to constantly poll the server or publisher for data (*Publish/subscribe systems* n.d.).



(a) MQTT Architecture *(b) MQTT Communication Process*

Figure 2.2: *MQTT architecture and communication process diagrams. (a) shows a general MQTT Client-Broker architecture (Soni and Makwana 2017). (b) shows the communication process that occurs between a MQTT client and a MQTT broker. Source:*

2.2 OPC-UA to MQTT Edge Connector

As mentioned previously in 1.0.2, the main goal of this project will be to develop a data processing system for an existing production facility that will enable the transmission of data from one or more industrial devices to a cloud database. Therefore, with OPC-UA protocols being used for many of the industrial devices within the facility and the MQTT protocol being used for data transmission, this project will primarily involve developing a OPC-UA to MQTT network interface to automate the transmission of data from these devices to a data visualisation application.

2.3 Similar Projects

There are a number of commercial or industrial systems or applications that achieve similar objectives as this project. This section serves to provide a brief description of these systems and describe how their existence has helped influence the design or development process of this project.

2.3.1 OPC-UA IoT Broker

Developed by Integration Objects, a solutions provider focused on process automation ad IIoT, the OPC-UA IoT Broker is designed as a plug-and-play software solution that allows for data collection from multiple OPC-UA servers and transmission of collected data to a Microsoft Azure IoT hub for storage and processing. The broker itself supports multiple transport protocols such as MQTT, AMQP and HTTP to communicate with the cloud (*OPC UA IoT Broker* n.d.).

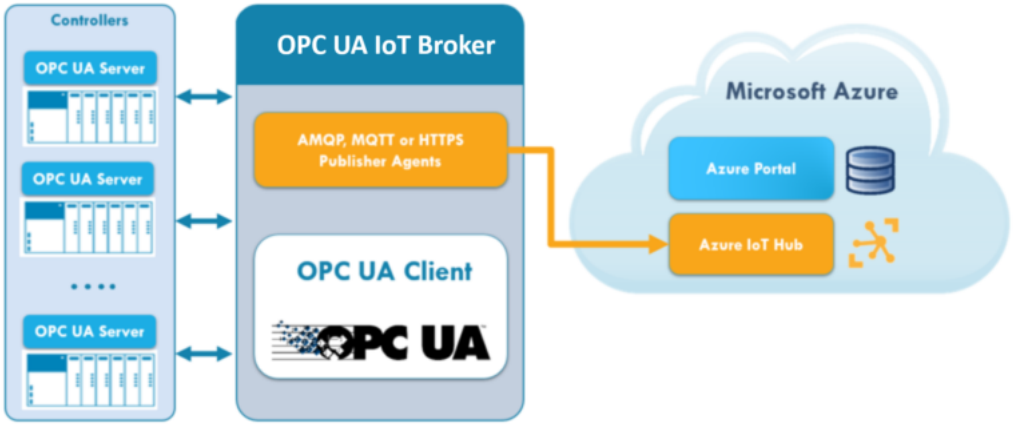


Figure 2.3: *Integration Objects OPC-UA IoT Broker design architecturre (*OPC UA IoT Broker *n.d.).*

With strong similarities between the OPC-UA IoT Broker functionalities and the system planned to be developed for this project, inspiration for the design architecture for the project's system could be drawn from the OPC-UA IoT Broker's design architecture.

2.3.2 DataHub IoT Gateway

An IoT gateway developed by Cogent Real-Time Systems Inc., a solutions provider focused on process control tools, the DataHub IoT Gateway acts as middleware between endpoint devices and cloud services or analytic platforms by connecting OPC-UA clients and servers to MQTT brokers and streaming real-time OPC-UA industrial date from the devices to the brokers (*Cogent DataHub* n.d.).

2.3.3 inray Industriesoftware GmbH OPC Router

The OPC Router is a network gateway middleware designed to facilitate communication between endpoint industrial devices and cloud or network systems. With support for the MQTT protocol as a plug-in, the OPC Router allows for transmission and receipt of MQTT messages. Furthermore, with JSON functionalities, data within the MQTT messages is able to be processed or generated from other forms of data, allowing for increased usability in a industrial environment where there may be multiple devices with different communication protocols or data formats (*OPC Router* n.d.).

2.3.4 Anybus CompactCom

Anybus CompactCom is a suite of hardware-based network gateway solutions developed by HMS Industrial Network's to provide IIoT functionalities and connectivity within industrial facilities. As part of it's functionalities, the Anybus CompactCom interfaces with devices and implements an OPC-UA server and also acts as a MQTT publisher, allowing for transmission of data through both the OPC-UA and the MQTT communication protocols. However, it is unclear whether the Anybus CompactCom is able to allow for commands or data to be sent to the device as well, which makes it only suitable just for data collection from the devices (*Anybus CompactCom* n.d.).

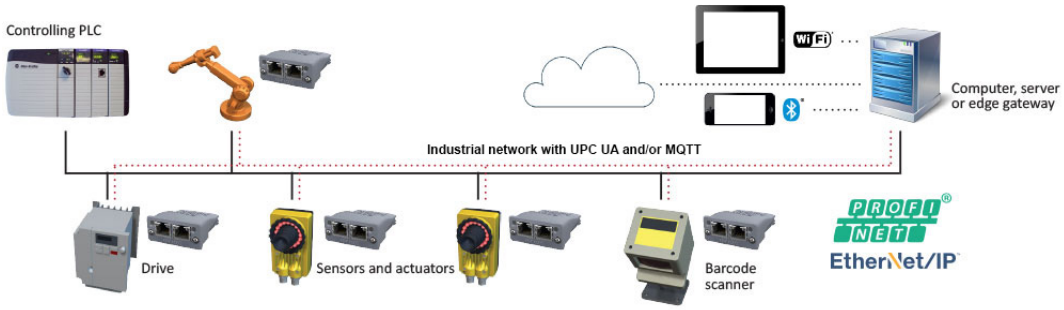


Figure 2.4: *Anybus CompactCom use case (*Anybus CompactCom *n.d.).*

As the Anybus CompactCom is a hardware-based solution, it is interesting to note that the use case for the Anybus CompactCom bears similarity to the design architecture for the Integration Objects OPC-UA IoT Broker in that it allows multiple devices to transmit data to an edge gateway, which can then transmit the data further to the cloud or other devices. Therefore, it would appear that the idea of having a system or gateway on the edge of a network of devices to process data before sending it to a centralised system for storage or analysis is starting to become an industry standard.

2.4 Summary

This chapter included a brief discussion of commonly used communication protocols used for M2M communication. A number of related products with similar functionality to the system developed by this project was also reviewed to determine the viability of this project.

3 Requirements Specification & Analysis

This chapter includes the analysis of the problem specification, details of the requirements gathering process and how the design of the system was determined.

3.1 Problem Specification

With reference to Chapter 1, the problem to be tackled by the project is the collection, analysis and visualisation of data generated by industrial devices such as robots or automated machines, with the proposed solution being the development of a singular system that allows for all of the above without requiring constant physical access to the devices. Therefore, development of the system would require designing a software solution that would be able to automate collection of data from various devices, within an industrial or production facility, which may communicate with different protocols, store this data into a centralised database or system, perform meaningful analysis on the data and finally provide users with a visualisation of the analysed data.

3.2 Requirements Gathering

As this project was conducted in collaboration with A*STAR, they would be the primary stakeholder for this project as they would ultimately be the one to implement, utilise and evaluate the system. Therefore, it was necessary to first determine what were their functional and non-functional requirements for the system and whether any specific technologies were preferred or required for development of the system. Gathering of these requirements were done via a combination of face to face meetings as well as email correspondences and prioritisation of requirements was done following the the MoSCoW prioritisation technique (*MoSCoW Prioritisation* 2019).

3.2.1 Functional Requirements

Must Have

- 1) Connect to specified endpoint(s) for collection of data from device. This allows users to specify and re-configure data collection endpoints, allowing for re-deployment of the system to different locations or facilities.
- 2) Publish collected data in real-time to a centralised database or system to allow for retention of data in case of system or device failure. This ensures that data analysis will be able to continue regardless of the status of the system or device.
- 3) Allow selection of data points to publish. This prevents transmission of unneeded data to the centralised database or system, helping to save network bandwidth and storage space.

Should Have

- 4) Display monitored data points and status to users. This allows users to verify that data being retrieved is in accordance with their configurations and can also act as a way to monitor the device status. For example, if the value from a particular data point being retrieved from the device is completely different to the value expected, while the values from other data points are as expected, it could indicate a potential issue with the device.

3.2.2 Non-functional Requirements

Must Have

- 5) Connection to specified endpoint(s) should be maintained continuously, with automatic re-connections if the endpoint restarts, ensuring that any data collected is up to date and allowing for data analysis to be as accurate as possible.
- 6) System should install and run as a service and run continuously, to ensure that any changes to data transmitted by monitored devices are captured.
- 7) System should have configurable parameters, allowing user to re-configure which endpoints to connect to and which devices and attributes to monitor. This allows users to reduce the amount of redundant or unnecessary data being collected, reducing usage of network bandwidth and storage space. In addition, configuration options should preferably be displayed in a Graphical User Interface (GUI).

Should Have

- 8) System should be able to recreate previous sessions along with related configurations in the event the system has to restart or the endpoint restarts. This prevents users from having to continuously reconfigure the system every time the system has to restart or fails.

Could Have

- 9) System should be inter-operable on major operating systems. As each facility may have different devices and endpoints running on different operating systems, the system needs to be able to work on major operating systems typically utilised in industrial facilities such as Windows and Linux.

3.2.3 Recommended Technologies

As the developed system would be implemented, utilised and maintained by A*STAR, choice of technologies to be utilised for development was deferred to them, with their recommendations as follows:

- As the system would be required to be inter-operable on any OS platforms, it was recommended to utilise the .NET Core framework along with C as the software development language, as both provide cross-platform capabilities. In the event that the above were to be utilised for development, Microsoft Visual Studio was recommended as the Integrated Development Environment due to it's extensive support for .NET and C.
- Furthermore, due to the extensive presence of industrial robots within ARTC utilising OPC-UA for data transmission, it was recommended that the system should minimally be able to collect and store data from the aforementioned robots. Therefore, this system would be developed primarily to allow for communication with devices utilising OPC-UA, with support for additional protocols being added in future development if required.

- In addition, with a centralised database or system being required for storage of the data collected, a cloud service would be provided by A*STAR to host a web application, which would be responsible for visualising the collected data. As such, data transmission would be done over a network, which could be unstable or be in use by multiple other systems. Therefore, it was recommended that the MQTT protocol be utilised for data transmission by the system to the web application due to the wide usage and support for the MQTT protocol and it's ability to reliably deliver messages in large networks (Naik 2017).

3.3 Summary

This chapter detailed the requirements gathering and analysis process, indicating the finalised functional and non-functional requirements of the system to be developed. In addition, a list of technologies recommended by the industry partner was also discussed.

4 Design

This chapter includes the details of the system architecture and the tools and technologies that were considered for use in development of the system after further analysis of the problem specification and implementation requirements.

4.1 System Architecture

The overall system was broken down into two main components, a data visualisation application which would be hosted on a cloud service and a data processing software solution, which processes data transmitted from a network of devices and then transmits the processed data to the data visualisation application. As indicated in 1.0.2, this project is done in collaboration with ARTC, which had already begun development of the data visualisation application and would focus on continuing development of the application throughout the duration of this project. Therefore, the focus of this project would be on the development of the data processing software solution, hereafter referred to as the Edge Connector system.

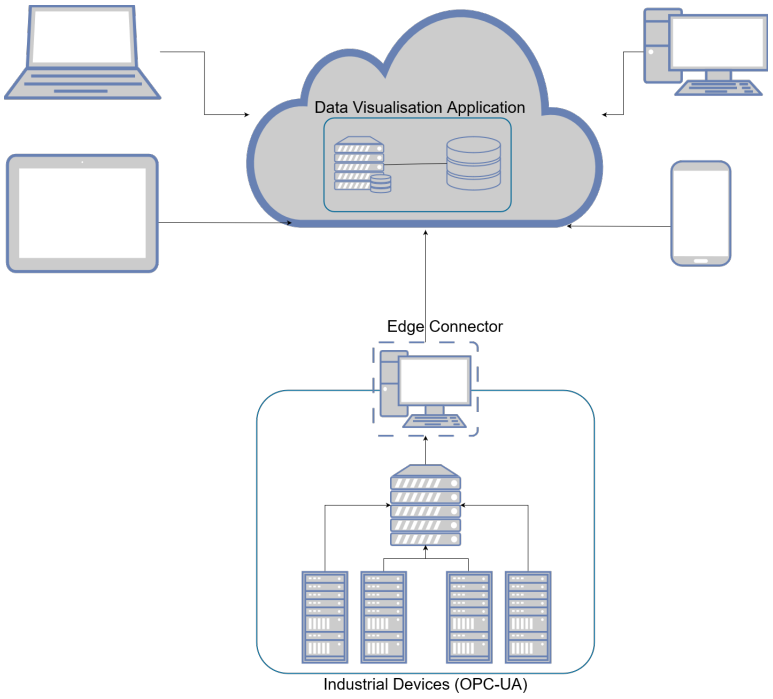


Figure 4.1: *Overall system architecture diagram, showcasing the connections between the different system components*

In accordance with requirements 6 and 7, the system would need to run as both a service, while also being configurable, with the configuration options being displayed in a GUI. However, support for services with GUIs attached is rare, with some support only found in earlier versions of Windows and even then, official Microsoft documentation does not recommend implementing a service with a GUI due to possible security issues such as exposing the configuration of the service to user accounts that should not have access (*Interactive Services - Win32 apps* n.d.). Therefore, it was decided that a separate GUI application would need to be implemented to communicate with the service for configuration. However, as the service and the GUI application could be running in different user or system accounts, it would be necessary to implement a form of inter-process communication (IPC) to allow the service and GUI application to communicate with one another.

4.2 Tools & Technologies

4.2.1 Framework & Language

.NET Core (C#) As the system would be utilised in industrial facilities, which potentially utilises various systems operating on different platforms and operating systems, it was important that the system be developed in a framework that supports cross-platform implementation and flexible deployment. As such, it was determined that .NET Core would be a suitable framework for development due to it's extensive support for cross-platform implementation and continuous support by Microsoft, which makes any system developed suitable for long-term usage (*.NET Core overview* n.d.). Development of the system would also be done in the C programming language, which is supported by the .NET Core framework. Furthermore, both the OPC-UA and MQTT protocols have existing documentation and libraries that support .NET and C implementation, reducing the amount of effort needed to implement the protocols in C from scratch.

4.3 Graphical User Interface Application

In order to comply with requirement 7, which required the system to have configurable parameters with the configuration options displayed in a GUI, a number of frameworks were initially considered for implementation.

4.3.1 Windows Presentation Foundation (WPF)

Windows Presentation Foundation (WPF), which is a User Interface (UI) framework support by the .NET framework and allows for responsive UI. However, it also relies heavily on DirectX and Windows-only libraries or technologies that prevent it from being cross-platform, which would completely prevent requirement 9 from being met even in future development (*Windows Presentation Foundation* n.d.).

4.3.2 Avalonia UI

A cross-platform alternative, Avalonia UI, was considered as a second option. As a relatively new open source XAML framework similar to WPF but with support for Windows, Linux and OSX, it would have allowed for requirement 9 to be met during existing or future development (*Avalonia* n.d.).

4.3.3 Windows Forms (WinForms)

Finally, Windows Forms (WinForms) was also considered due to it's simplicity and ease of learning. In addition, the OPC Foundation, which created and maintained the OPC-UA protocol, has an official .NET stack, which includes sample applications utilising WinForms (*OPC UA .NET* n.d.) Therefore, using WinForms for the GUI development would allow for the usage of cannibalisation of existing controls or code from the sample applications in the OPC-UA .NET stack. Furthermore, WinForms also has some cross-platform support in the form of the Mono framework (*Mono-Winforms* n.d.), which allows the possibility of implementation in OSX and Linux, allowing for requirement 9 to be met in future development (Beribey and Coderes 2019).

As a result, it was decided that WinForms would be utilised for the GUI as it would not only be able to fulfil the most requirements, but also help to reduce time taken for the GUI development due to the possibility of cannibalising or editing and re-using code from the sample applications in the OPC-UA .NET stack.

4.4 Service Application

In order to comply with requirement 6, which requires the system to run continuously on the host machine, the core logic of the system would need to be implemented as a service process on the local system account of the host machine. However, as indicated earlier, support for services with GUIs attached is rare and exposes potential security vulnerabilities within the host machine. Therefore, a mechanism to allow for inter-process communication between the GUI application and the service application. As WinForms was chosen as the framework for the GUI application, it was decided that the IPC mechanism should preferably be officially supported on Windows as well, to ensure compatibility with the GUI application. As such, the following IPC mechanisms were considered:

4.4.1 IPC Mechanisms

Pipes are sections of shared memory used by processes for communication. Pipe servers are processes that create pipes, while pipe clients are processes that connect to pipes. When communicating using pipes, a process first writes information to the pipe, which is then accessed by another process who reads the information written, allowing for inter-process communication. In addition, there are two different types of pipes which provide different functionalities (*Pipes* n.d.):

Anonymous Pipes are unnamed, one-way pipes that allows for data to be transferred between a parent process and a child process. Furthermore, usage of anonymous pipes is restricted to local processes only and cannot be used to communicate over networks (*Anonymous Pipes* n.d.). This adds a level of security as any data transfer occurs only on the host machine. However, as the GUI and service applications needed to communicate with one another, the anonymous pipes mechanism was deemed unsuitable due to it's one-way only pipe.

Named Pipes are pipes with defined names that allow for communication between a pipe server, which is a process that creates a named pipe, and one or more pipe clients, which are processes that connect to instances of a named pipe. This is accomplished due to a single named pipe being able to have multiple instances, all sharing the same pipe name, but with their own individual

buffers, handles and communication conduits. Therefore, this allows multiple pipe clients to access the same named pipe via the separate instances. In addition, communication can be one-way or duplex depending on the implementation required, allowing for a wider variety of use cases (*Named Pipes* n.d.). However, named pipes can also be used to allow processes on different machines in a network to communicate, which may pose potential security risks as data can be intercepted while being transmitted in the network.

Memory-Mapped Files are files whose contents have been mapped to virtual memory. This allows multiple processes to modify the file by directly reading and writing to the memory. As a result, this mechanism only allows for inter process communication between local processes. In addition, there are two variants of memory-mapped files, which offer their own advantages and disadvantages (*Memory-Mapped Files* n.d.).

Persisted memory-mapped files are associated with a source file located on a disk. Once all processes have finished interacting with the file, it's contents are saved to a source file on a disk, which is suited for working with files that contain large amounts of data. However, this naturally requires a large amount of disk space on the host machine, which was not required for the Edge Connector system as the system was only responsible for processing and transmitting the data to a centralised system which would be the one to store the data (*Memory-Mapped Files* n.d.).

Non-persisted memory-mapped files are not stored on a disk at all. Instead, the file will be reclaimed by garbage collection and the data will be lost, which is more suitable for smaller files or creation of shared memory for multiple processes (*Memory-Mapped Files* n.d.).

However, as the GUI and service applications would not only have to communicate with one another in real-time, but data changes could occur from either application, it was deemed that using this mechanism for the IPC process would be more difficult to implement due access to the memory space having to be synchronised between both applications. As other IPC mechanisms, such as named pipes would not raise the same difficulty, it was decided that this mechanism would only be utilised if no better options were available.

Ultimately, it was decided that named pipes would be used as the IPC mechanism due to their duplex functionality and ease of implementation. In particular, with the aid of the Windows Communication Framework (WCF), which supports implementation of named pipes and allows for additional configuration options, implementing named pipes would be much easier compared to implementing memory-mapped files (*Choosing a Transport - WCF* n.d.). By implementing the named pipes mechanism as a WCF service and subsequently hosting the WCF service in the service application, the service and GUI applications would effectively be able to communicate with one another via the named pipes mechanism in the WCF service. In addition, similarly to WinForms, Linux has some support for WCF in the form of the Mono framework (*Mono-WCF* n.d.). Therefore, utilising WCF to implement the IPC mechanism would not only save time for implementation, but also still allow for possible cross-platform support.

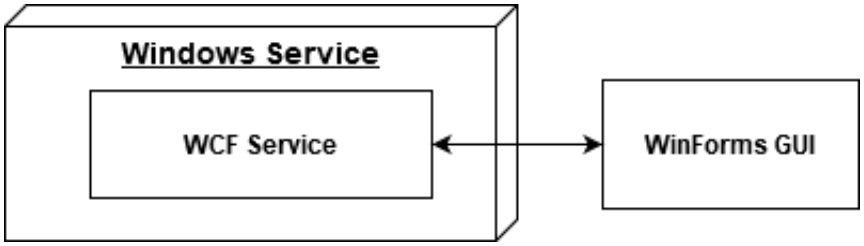


Figure 4.2: *Edge connector system design, showcasing the connections between the different system components*

4.4.2 Communication Protocols

As indicated earlier in Chapter 3, it was recommended by ARTC that the Edge Connector system should utilise the MQTT communication protocol for data transmission to the centralised system and should be able to collect and process data from devices utilising the OPC-UA communication protocol. However, as the framework chosen for development was the .NET framework, it was first necessary to determine whether it would be possible to implement these protocols in .NET. Fortunately, both the MQTT and OPC-UA protocols have supported .NET implementations, with the MQTT protocol having a number of frequently updated open-source .NET implementations, such as the MQTTnet library (*MQTTnet* n.d.), and the OPC-UA protocol having an official .NET stack which also contains sample applications. Therefore, with the presence of the aforementioned resources to support .NET implementations, it was determined that the MQTT and OPC-UA communication protocols would be utilised as per ARTC's recommendation. As the MQTT protocol would be utilised for transmitting data to the data visualisation application, while the OPC-UA protocol would be utilised for receiving data from an OPC-UA server, this meant that the Edge Connector system would need to host an MQTT broker as well as an OPC-UA client.

4.5 Summary

This chapter contained a brief overview of the overall system architecture design of the system which the Edge Connector system would be implemented with, as well as the system architecture design of the Edge Connector system itself. Various components of the Edge Connector system design were also explored in detail, with discussions of available tools or technologies that were considered for the design of the components and why certain tools or technologies were chosen over others.

5 Implementation

This chapter includes details regarding the development and implementation of the Edge Connector system.

5.1 Development Process

Due to the presence of resources such as the OPC-UA .NET stack, which contained sample applications that could effectively be cannibalised to allow development of certain features in the Edge Connector system as well as due to the project being easy to split into different iterations - each focusing on the development of a specific feature - it was determined that following a combination of an iterative and incremental development process would be optimal. The idea behind the process being that the system would be developed through multiple iterations and incrementally, with each iteration being presented to the client - in this case, ARTC - allowing for feedback from the client and potential improvement in future iterations (Farcic 2014).

5.2 Version Control

Due to the iterative nature of the project, extensive usage of version control was necessary to allow for tracking of changes, backing up of code and allow reversion to previous iterations if a serious bug or flaw had been introduced in newer iterations of the project. As such, a private GitHub repository was set-up in order to manage resources related to this project, with the Sourcetree Git GUI being used for easier management of the repository (*Sourcetree* n.d.).

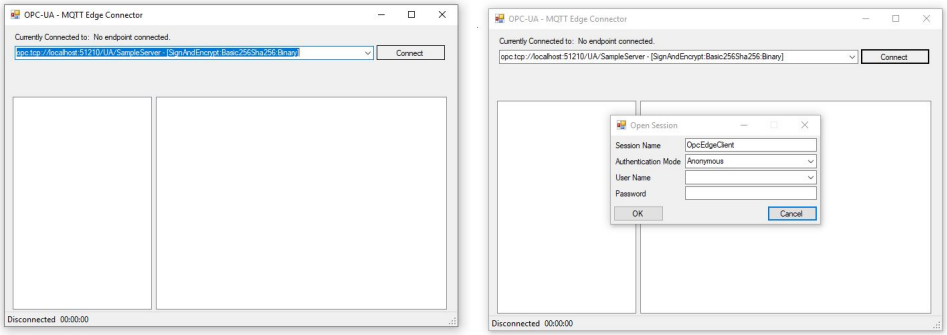
5.3 Feature Implementation

As the core logic of the Edge Connector system would be implemented in the service application, it was initially decided that early iterations would first focus on implementing the core features of the service application, with development of the GUI application being done in later iterations upon completion of the service application. However, as certain features such as display of data points or allowing users to select data points to publish would be difficult to debug and test with only the service application developed, it was decided that it would be better to develop both the service and GUI application in tandem. This would also reduce the risk of integration issues arising later in development if both applications had been separately developed. Therefore, each iteration would focus on a specific feature to be developed for both the service and GUI applications.

5.3.1 Endpoint Connection

The first feature to be implemented was the ability for the system to connect to a specified endpoint to retrieve data. In this case, as the devices we needed to retrieve data from utilise the OPC-UA communication protocol, which utilises Client-Server based communication, the system would need to be able to act as an OPC-UA client and be able to connect to an OPC-UA server.

Therefore, the Edge Connector system would first connect to a specified OPC-UA server endpoint, request for user credentials if required and establish a session. In addition, once connected, the system would need to constantly check if the endpoint was still running and available to fetch data from. This was achieved via use of event handlers to regularly check if the session between the endpoint and the system was still alive. In the event that the session was killed unintentionally, the system would attempt to reconnect until a session could be established or a different endpoint was selected for connection. In addition, to meet requirement 8, which required that the system be able to recreate previous sessions along with related configurations, session information would be stored based on the endpoints connected. This means that whenever the Edge Connector system re-connects to an endpoint it had previously established a session with, the system would attempt to recreate the previous session's configurations.



(a) *Endpoint Connection. Allows users to specify OPC-UA endpoint for connection.*

(b) *Endpoint credentials input. Allows users to input credentials if required by OPC-UA endpoint.*

Figure 5.1: *Edge Connector Endpoint Connection GUI screens (a) the endpoint connection screen that allows users to specify an OPC-UA endpoint for connection. (b) shows the endpoint credentials screen that allows users to specify credentials if required by the OPC-UA endpoint.*

5.3.2 Data Display

Once the Edge Connector system was able to establish and maintain a session with an OPC-UA endpoint, the next feature to be implemented was the ability to browse and view data points from the OPC-UA endpoint. This feature was implemented primarily in the GUI application with the aid of the OPC-UA .NET stack, which included sets of WinForms controls that could be utilised and modified to implement this feature. These controls allowed for the access and display of the the various data points.

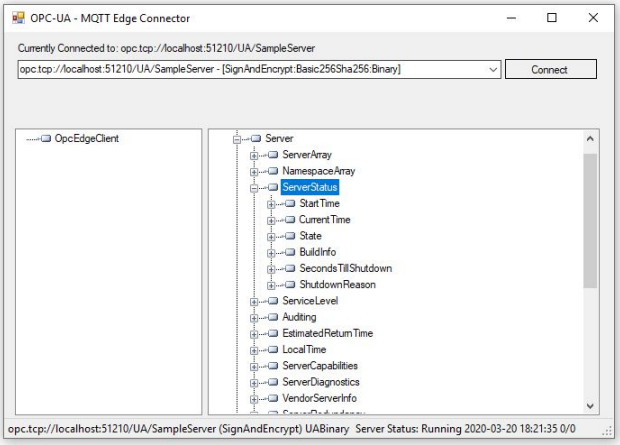


Figure 5.2: *Edge Connector Data Display GUI screen. Allows users to view data points that can be subscribed to or monitored.*

5.3.3 Subscription

With the Edge Connector system being able to maintain a session with the OPC-UA endpoint and display the available data points to the user, the next feature to be implemented was the ability to subscribe to a data point and associate the subscription with the session. Users would be able to select data points via the GUI application and configure certain subscription parameters such as subscription intervals, which determine the intervals at which the data value is transmitted, and keep-alive counts, which are used to transmit a response signalling that the subscription is still active in the event that number of consecutive publishing cycles in which there have been no data changes has reached the specified count. Once the subscription is created, the GUI application then sends the data point information and subscription parameters to the service application to add the subscription to the session. After the subscription has been added to the session, the service application begins retrieving the data from the OPC-UA endpoint, and converting it to JavaScript Object Notation (JSON), which is a data interchange format, to prepare it for transmission via the MQTT protocol.

In addition, during development of this feature, in order to meet requirement 8, the system would also save the subscription parameters and details of any subscribed data points, allowing for the recreation of previous sessions along with previously subscribed data points.

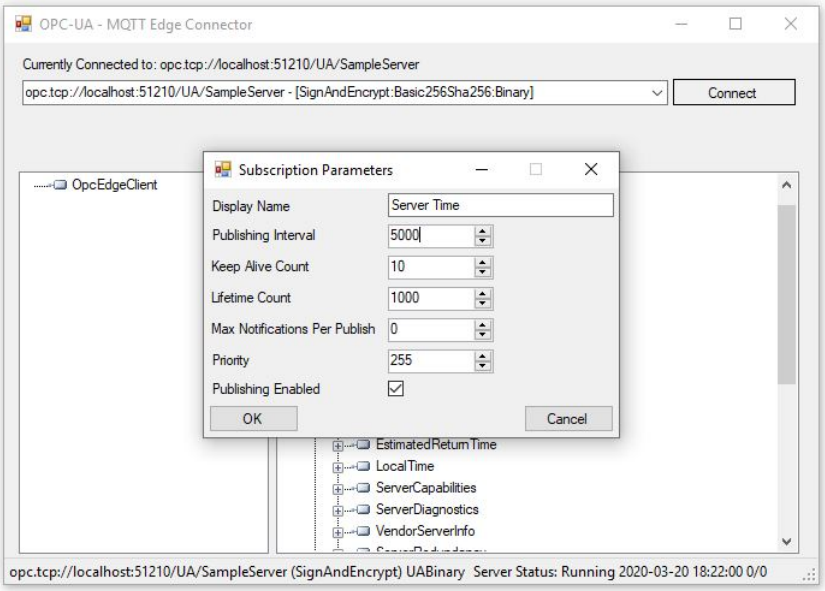


Figure 5.3: *Edge Connector Subscription GUI screen. Allows users to specify subscription parameters when subscribing to a data point.*

5.3.4 Data Publishing

Once a user has selected a data point to subscribe to, the Edge Connector system would publish the latest values from the data point to a predetermined endpoint - in this case, the data visualisation application - depending on the subscription intervals indicated during the subscription process. Data publication was achieved via implementation of a MQTT client, with the aid of the MQTTnet library, within the service application. As indicated earlier, when a subscription is added to the session, the service application converts the data to JSON. This formatted data can then be transmitted by the MQTT client to a specific MQTT broker. In addition, as the session could have multiple subscriptions with varying subscription intervals, this was accomplished via a Timer class which would regularly cross check the last time a data value was published and compare it to the subscription intervals to determine if the data value needed to be published again. On the data visualisation application's side, which hosts an MQTT broker, the messages are received and stored in a database for further analysis and visualisation.

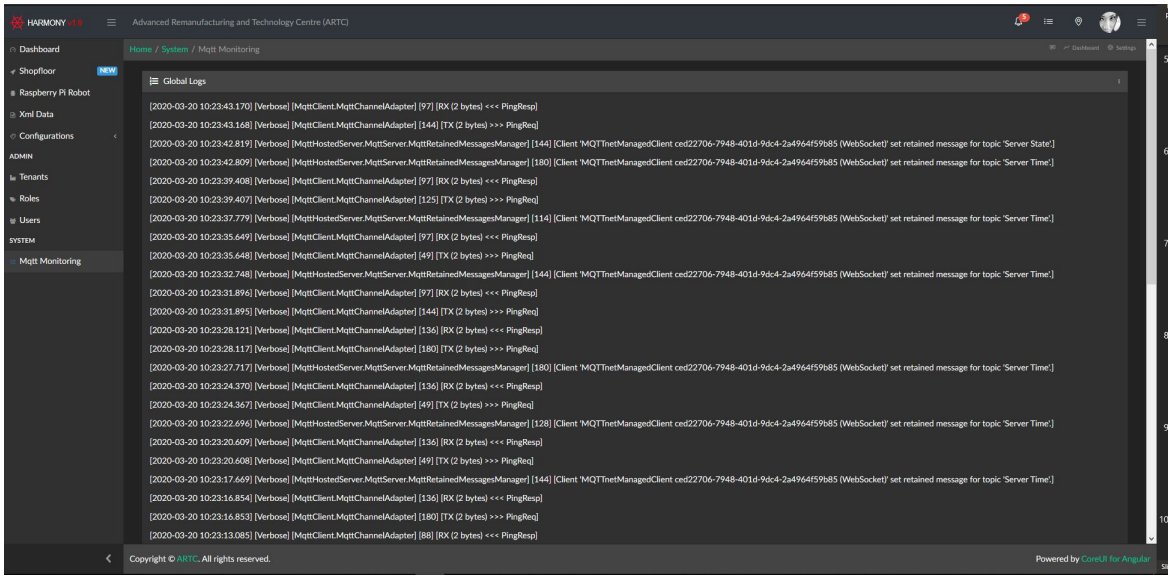


Figure 5.4: *Data visualisation application data monitoring screen. Published messages can be seen with their associated topic.*

5.3.5 Data Monitoring

As an additional 'Should Have' feature, the system would also allow users to monitor real-time changes to selected data values, essentially allowing users to remotely monitor data from OPC-UA devices as long as they have network access to the OPC-UA server the devices are connected to.

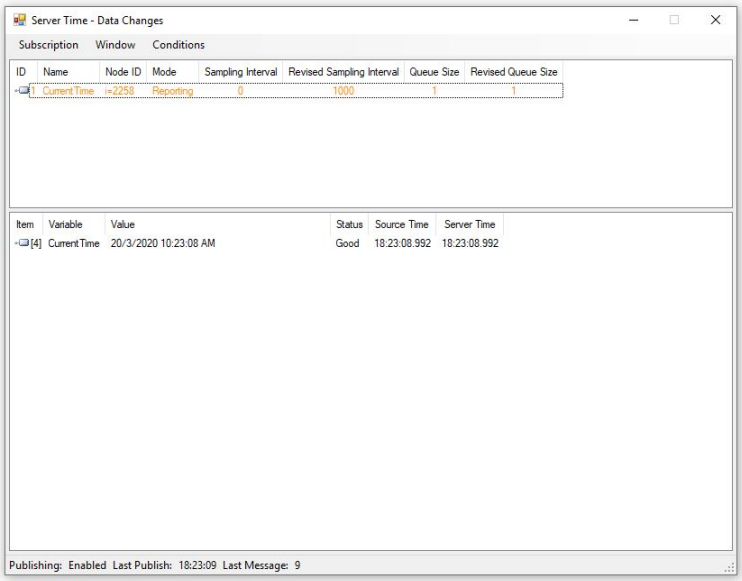


Figure 5.5: *Edge Connector Data Monitoring GUI screen. Allows users to view real-time changes to selected data values.*

5.4 Service and GUI Interaction

As indicated earlier in Chapter 4, a named pipes IPC mechanism implemented as a WCF service would be used to help allow the service and GUI application to communicate with one another. The WCF service would be hosted within the service application to ensure that the lifetime of the IPC mechanism would be tied to the service application's lifetime. As long as the service application continues to run, the IPC mechanism would run alongside it, allowing the GUI and service application to communicate with one another.

5.5 Deployment

As both the service and GUI application were developed using the .NET framework, both applications were also able to be packaged together and installed on a host machine using Windows Installer (*Windows Installer* n.d.). The initial deployment plan was to install the Edge Connector system on a host machine that had network access to the OPC-UA server collecting data from the OPC-UA devices within the production facility in order to demonstrate system functionality. Unfortunately, due to the Coronavirus pandemic in 2020, this deployment plan could not be tested within the facility access due to restricted access for health concerns. However, due to the data visualisation application being hosted on the cloud, it was possible to test a deployment set-up whereby the host machine and OPC-UA server were both accessible via the Internet. During this deployment test, the Edge Connector system was able to successfully subscribe to data points on a public OPC-UA broker - in this case, the OPC UA Rocks public test server (note = Accessed on 20.11.19 n.d.) - and publish the subscribed data to the data visualisation application. Therefore, with the deployment being able to work with both the host machine and OPC-UA server being accessible via the Internet, there should be no issues with a deployment whereby both host machine and OPC-UA server are located in the same local network.

5.6 Summary

This chapter contained details regarding the overall development process and implementation stages for the project. The thought process behind the development process chosen was discussed in detail and the implementation process of the various components of the Edge Connector system was explored.

6 Evaluation

This chapter details the testing and evaluation done to determine the effectiveness of the system developed and the success of the project.

6.1 Feature Testing

As the Edge Connector system was developed following a combination of an iterative and incremental development process, testing of each major feature was done throughout the project at each iteration. In addition, once development had stopped, an additional round of tests were conducted to ensure that there were no major bugs or unexpected behaviour by the system before handing the source code over to ARTC. During this final stage of testing, testing was done by simulating how an average user of the system might utilise it. However, while these tests allowed for the testing of the system features, there were certain components, such as the WCF service and IPC mechanism, were found to be difficult to test without having to write additional code or utilise additional software. Ultimately, it was decided that these components would not require individual tests as it could be argued that certain features - such as the ability to subscribe or publish data points - were able to pass testing, then it would signify that the WCF service and IPC mechanism were working as intended as those features require both the service and GUI application to communicate with one another.

6.2 Requirements Validation

As this project was undertaken in collaboration with ARTC, who had specified requirements as indicated in Chapter 3, part of the evaluation process involved determining how many of the specified requirements were met.

Overall, out the 9 requirements set for the Edge Connector system, 7 requirements were completely met, with all of the 'Must Have' and 1/2 'Should have' requirements being met. The remaining unmet 'Should Have' requirement, Requirement 8 was deemed to be only partially met as while the system could recreate sessions with their related configurations, the recreation was limited to recreation of the previous session and would only trigger if the system was configured to connect to the same endpoint as the previous session. The other unmet requirement, Requirement 9, which was a 'Could Have' requirement, was determined to have been unmet due to the usage of Windows specific components such as WinForms and WCF in the development of the system, it was determined that the system would be unlikely to work in other operating systems. However, it should be noted this was not able to be properly tested due to time constraints and as indicated in Chapter 4, Linux does have some support for WinForms and WCF in the form of the Mono framework. Therefore, it is very possible with some minor changes to the Edge Connector system that it would at least be able to work in Linux systems.

Table 6.1: *Justification for requirements validation*

Requirement	Met / Unmet	Validation
1	Met	Met by Endpoint Connection feature
2	Met	Met by Data Publishing feature
3	Met	Met by Data Display, Subscription and Data Publishing features
4	Met	Met by Data Display feature
5	Met	Met by Service Application implementation
6	Met	Met by Service Application implementation
7	Met	Met by GUI Application implementation and Data Display feature
8	Partially Met	Partially met by Service Application implementation
9	Unmet	Potentially unmet due to usages of Windows specific components

6.3 Performance Evaluation

Initial plans for evaluation involved measuring the performance of the Edge Connector system when deployed in an actual production facility and comparing differences in performance if a different communication protocol was utilised for data transmission instead of the MQTT protocol. Unfortunately, due to the Coronavirus pandemic, the evaluation could not be carried out. However, with additional research, a number of papers published in previous years were found that provided analysis comparing the performance of the MQTT protocol to other communication protocols.

One such paper directly compared the performance of the OPC-UA protocol and the MQTT protocol for transmission of data (Silveira Rocha et al. 2018), which heavily relates to this project. Without the Edge Connector system in place, the OPC-UA protocol would have been used for data transmission instead, as the OPC-UA server would need to transmit the data directly to the data visualisation application, which would most likely have had a OPC-UA client implemented to receive the data.

In the paper, an evaluation was done to compare the time taken for transmission of data with variable sizes to a server (Silveira Rocha et al. 2018). In addition, the evaluation was also done with MQTT protocol being configured with various levels of Quality of Service (QOS), which defines how the delivery of the message is guaranteed, with QoS 0 defining that the message should be received at most once, QoS 1 defining that the message should be received at least once and QoS 2 defining that the message should be received exactly once. The MQTT protocol utilised in the Edge Connector system utilises QoS 2, which means that the recipient should only receive the message once (*QoS levels* n.d.). This means that the recipient has to transmit a response to the sender to acknowledge receipt of the message, otherwise the sender may keep transmitting the message repeatedly until an acknowledgement is received. However, while this guarantees message receipt and prevents transmission of duplicate messages, this also results in a slower time taken to transmit data.

Payload Size [Bytes]	Protocol	Mean [ms]	Median [ms]	St. Dev. [ms]	Max. [ms]	Min. [ms]
100	mqtt_0	6.08	6.08	0.57	28.63	5.42
	mqtt_1	54.50	54.07	9.02	636.33	51.97
	mqtt_2	65.91	65.50	5.76	512.41	62.78
	opcua	67.22	67.07	1.06	553.90	60.31
10k	mqtt_0	8.62	8.34	1.46	47.10	7.70
	mqtt_1	14.14	14.09	0.89	52.11	12.87
	mqtt_2	25.63	25.47	3.19	331.98	23.63
	opcua	42.23	40.11	2.72	60.65	17.27
100k	mqtt_0	35.60	35.21	5.00	385.32	33.56
	mqtt_1	34.79	34.30	2.55	89.07	31.10
	mqtt_2	46.57	46.15	2.44	88.47	43.72
	opcua	58.01	55.40	5.86	85.04	37.97

Figure 6.1: *Comparison between time taken for data transmission with variable file size using OPC-UA and MQTT (Silveira Rocha et al. 2018).*

As can be seen in the figure above, the time taken for transmitting a payload of size 100 bytes using the MQTT protocol configured with QoS level 2 is fairly close to the time taken for transmitting the same data using the OPC-UA protocol. However, it can also be seen that as the payload size increase, the time taken for data transmission using the OPC-UA protocol increases much faster than the time taken for the MQTT protocols. Therefore, the MQTT protocol still appears to be the better option for data transmission as compared to the OPC-UA protocol regardless of the QoS level.

In addition, in the same paper, another evaluation was also carried out to compare the loop-back time to various server locations when transmitting a payload of a set size using both protocols (Silveira Rocha et al. 2018). The results are also similar to the earlier evaluation, with the OPC-UA protocol having a greater increase in time taken as the distance between the sender and recipient increases.

Location	Protocol	Mean [ms]	Median [ms]	St. Dev. [ms]	Max. [ms]	Min. [ms]
Local Network	mqtt_0	5.33	5.31	0.46	8.19	4.87
	mqtt_1	15.82	15.77	0.50	22.05	13.47
	mqtt_2	25.44	25.51	0.53	27.51	22.91
	OPC UA	29.12	29.27	0.61	26.83	26.83
São Paulo	mqtt_0	33.80	31.69	1.82	204.50	25.89
	mqtt_1	58.93	54.12	2.57	382.85	50.52
	mqtt_2	62.80	57.78	2.82	357.82	59.50
	OPC UA	62.11	60.01	2.64	515.79	59.21
Oregon	mqtt_0	175.39	173.79	3.44	610.21	152.44
	mqtt_1	210.02	211.52	3.17	759.37	189.17
	mqtt_2	238.90	232.16	4.85	749.84	215.75
	OPC UA	251.61	241.11	4.11	812.40	222.65
Frankfurt	mqtt_0	243.27	243.12	2.88	437.77	218.94
	mqtt_1	311.94	307.06	3.30	811.85	288.09
	mqtt_2	372.61	365.61	5.53	756.41	352.76
	OPC UA	389.17	388.49	5.26	899.63	333.71
Tokyo	mqtt_0	327.79	317.27	7.79	760.25	295.48
	mqtt_1	408.04	395.88	8.29	1025.10	385.04
	mqtt_2	416.40	401.98	9.76	1135.43	398.56
	OPC UA	433.13	419.78	12.16	1235.61	401.58

Figure 6.2: *Comparison between time taken for data transmission with variable locations using OPC-UA and MQTT (Silveira Rocha et al. 2018).*

Therefore, it appears that regardless of the payload size and location of the senders or recipients, the MQTT protocol is better suited for data transmission as compared to the OPC-UA protocol.

6.4 Summary

This chapter discussed the feature testing process and provided validation for the list of requirements which were deemed to have been met by the final iteration of the Edge Connector system. As the initial evaluation was unable to be conducted due to the Coronavirus pandemic, evaluation results from a similar study was used to showcase the performance comparison between the MQTT and OPC-UA protocols, with the MQTT protocol being shown to have better performance than the OPC-UA protocol in terms of speed and transmission of large payloads.

7 Conclusion

This chapter includes the concluding statement for the project, a brief summary of the project as well a personal reflection and potential future work that could be accomplished if given the opportunity.

7.1 Summary

The Edge Connector system is a data processing system that was designed to support the processing, monitoring and transmission of data from a network of devices in a production facility to a centralised system, with the system making use of the OPC-UA and MQTT communication protocols to read and transmit data respectively.Development of the system was done following a combination of an iterative and incremental development process, allowing for regular feedback to be obtained from the industry partner. In addition, evaluation of the system was done via feature testing during each iteration of the project, with a final set of tests conducted upon completion of the Edge Connector system and prior to handover of the source code to the industry partner. Unfortunately, due to the Coronavirus pandemic, further evaluation was unable to be carried out. However, based on existing evaluations of the OPC-UA and MQTT protocols as well as from the limited amount of testing and evaluation done on the system, the Edge Connector system has the potential to be an IIoT software solution that can benefit the productivity and improve operations of industrial facilities where it is deployed.

7.2 Reflection

Undertaking this project has taught me more about the process behind designing, developing and implementing a software project. In particular, the importance of time management and scope definition was greatly stressed throughout the project, especially with the presence of other work or assignments that had to be completed throughout the project duration, which emphasised the need to properly allocate time for this project as well as determine the scope of work that could be completed during the time spent. Another important lesson learnt was the importance of proper documentation. Again, due to the presence of external work or assignments, work was carried out on the project at irregular intervals, which meant that documenting the work done and the work remaining for each iteration was essential to stay on track. Lastly, this project has taught me about the importance of being able to properly read and understand code written by others. Due to the presence of resources such as the OPC-UA .NET stack and sample applications, being able to read and understand the code written in those applications allowed for the cannibalisation of the code for the implementation of the Edge Connector system, which saved valuable time for testing and debugging.

7.3 Future work

Given the chance to continue work on the Edge Connector system, there are definitely a number of possible improvements or alterations that could be made. Currently, while the system does have the ability to recreate sessions and their configurations, it is only able to do so for the previous session and only if the system is re-connecting to the same endpoint as the previous session. A possible improvement that would be fairly easy to implement would be to allow the system to allow the user to decide if they would like to save and load session data from a file, which would allow for easy set-up and configuration of the system across multiple host machines on multiple networks. Another possible improvement would be to allow the system to send alert messages to the centralised system depending on the data received from the devices. For example, if a device were to malfunction or send erroneous data, it would be good if the Edge Connector system had the ability to detect these errors and send the appropriate alert messages to the centralised system. This could reduce the amount of active monitoring required, allowing users to remotely determine and perhaps even troubleshoot issues based on the alerts received without needing physical access to the devices.

A Edge Connector system

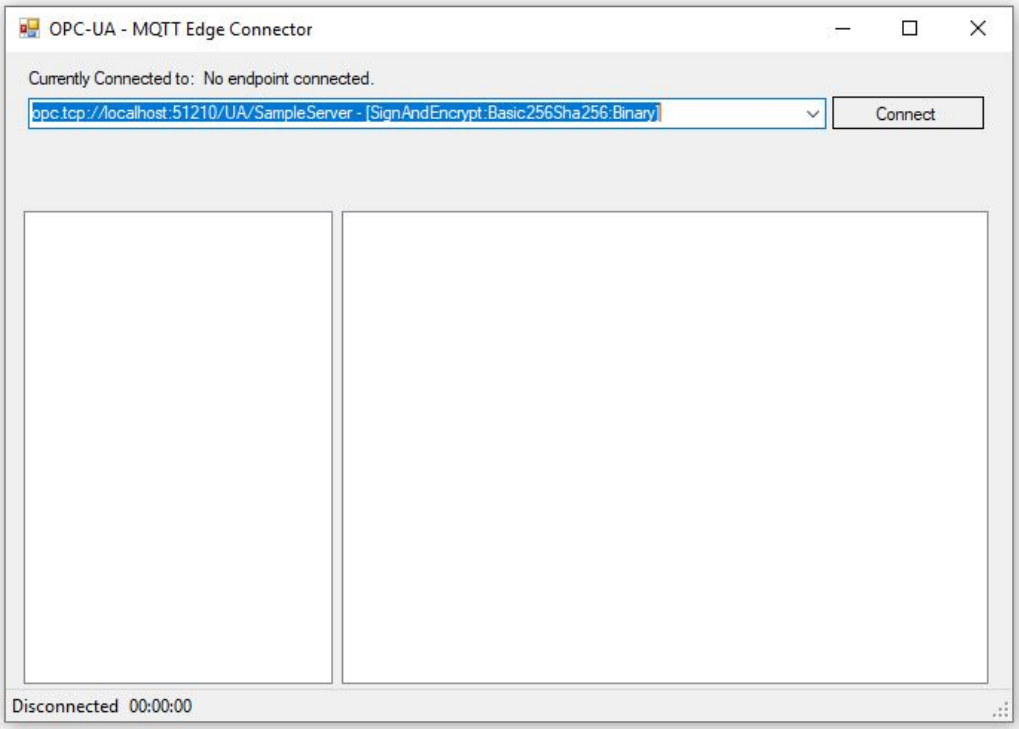


Figure A.1: *Data Monitoring GUI screen. Allows users to view real-time changes to selected data values.*

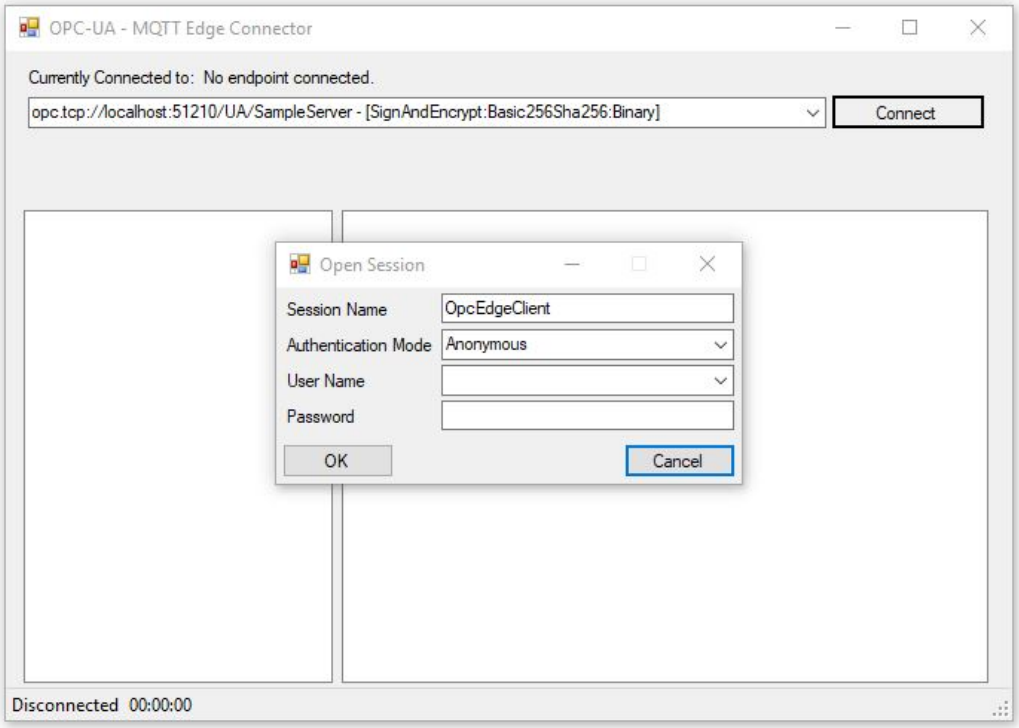


Figure A.2: *Endpoint credentials input. Allows users to input credentials if required by OPC-UA endpoint.*

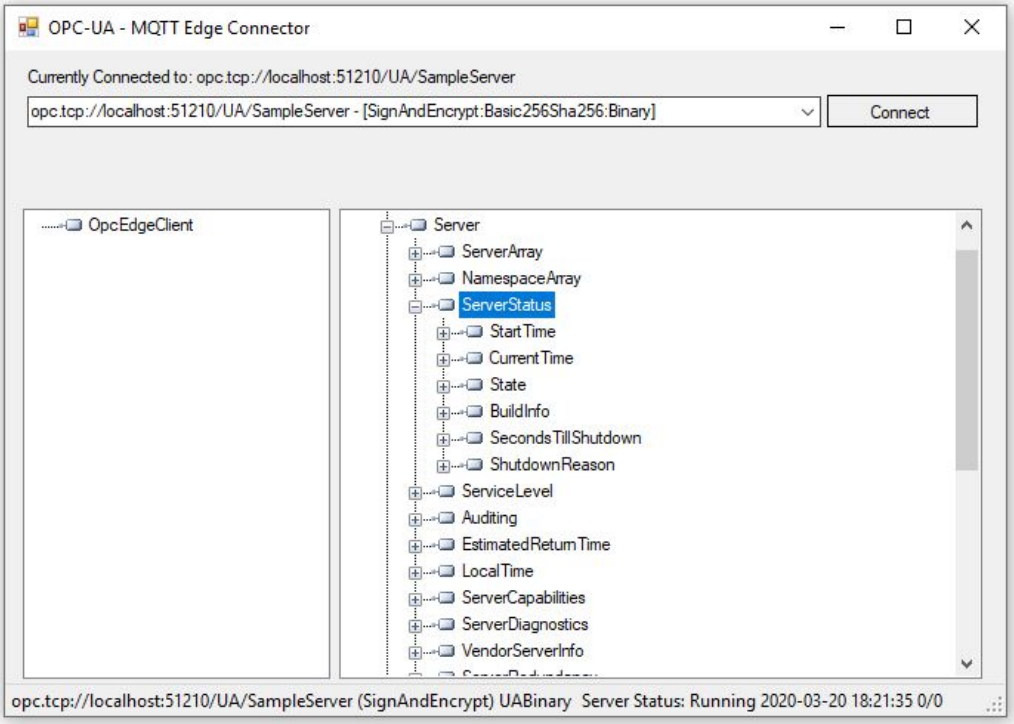


Figure A.3: *Edge Connector Data Display GUI screen. Allows users to view data points that can be subscribed to or monitored.*

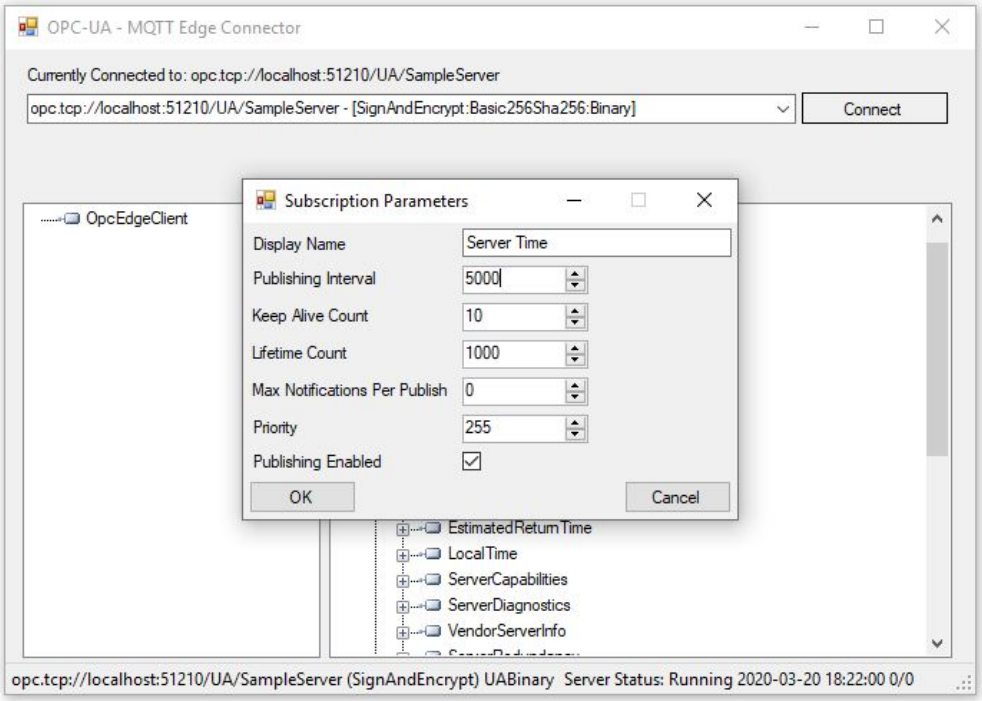


Figure A.4: *Edge Connector Subscription GUI screen. Allows users to specify subscription parameters when subscribing to a data point.*

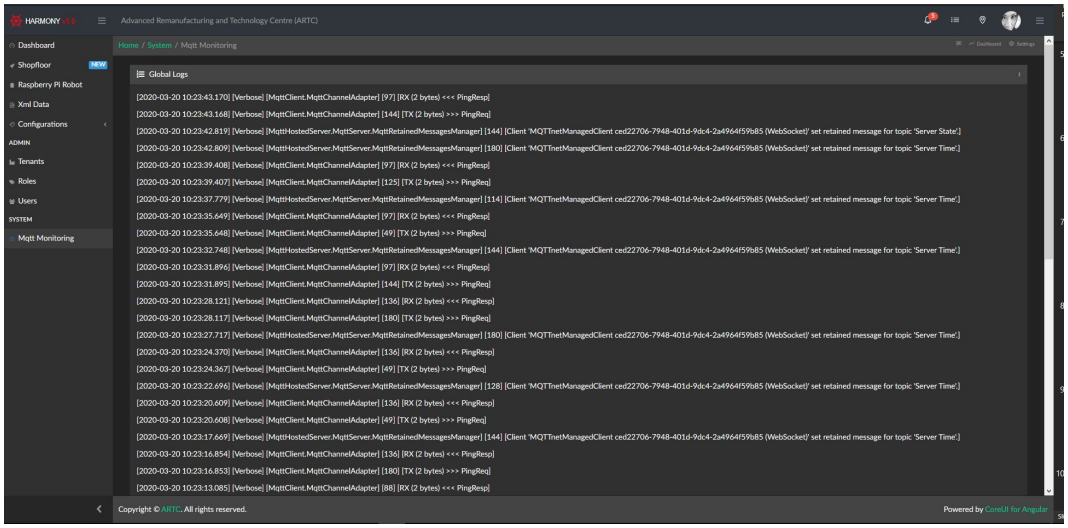


Figure A.5: *Data visualisation application data monitoring screen. Published messages can be seen with their associated topic.*

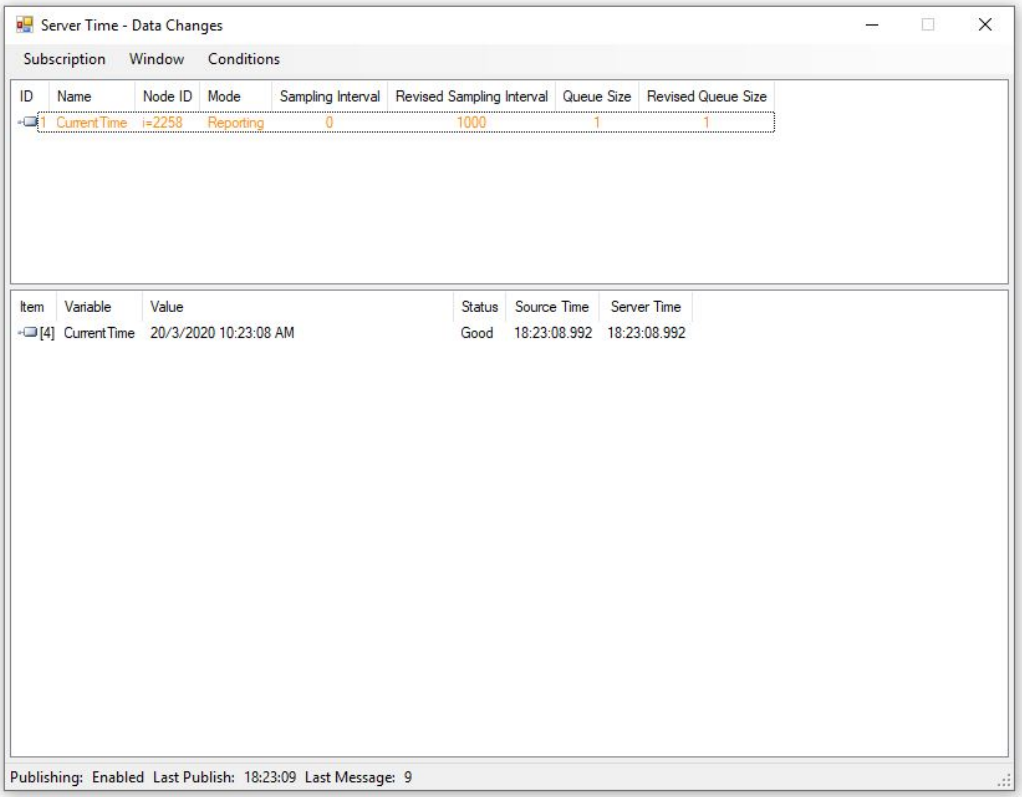


Figure A.6: *Edge Connector Data Monitoring GUI screen. Allows users to view real-time changes to selected data values.*

Bibliography

Anonymous Pipes (n.d.). Accessed on 10.11.19.
URL: *https://docs.microsoft.com/en-us/windows/win32/ipc/anonymous-pipes*

Anybus CompactCom (n.d.). Accessed on 20.12.19.
URL: *https://www.anybus.com/products/embedded-index/embedded-features/opc-ua-mqtt*

Avalonia (n.d.). Accessed on 10.12.19.
URL: *https://avaloniaui.net/*

Beribey and Coderes (2019), 'Why is winform still not dead?âĂŁ-âĂŁshould we learn winform?'. Accessed on 10.12.19.
URL: *https://medium.com/coderes/why-is-winform-still-not-dead-should-we-learn-winform-5d7764635790*

Cheng, G., Liu, L., Qiang, X. and Liu, Y. (2016), Industry 4.0 development and application of intelligent manufacturing, *in* '2016 International Conference on Information System and Artificial Intelligence (ISAI)', pp. 407–410.

Choosing a Transport - WCF (n.d.). Accessed on 10.11.19.
URL: *https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/choosing-a-transport*

Cogent DataHub (n.d.). Accessed on 20.12.19.
URL: *https://cogentdatahub.com/products/datahub-iot-gateway/*

Durkop, L., Czybik, B. and Jasperneite, J. (2015), Performance evaluation of m2m protocols over cellular networks in a lab environment, *in* '2015 18th International Conference on Intelligence in Next Generation Networks', pp. 70–75.

Farcic, V. (2014), 'Software development models: Iterative and incremental development'. Accessed on 20.11.19.
URL: *https://technologyconversations.com/2014/01/21/software-development-models-iterative-and-incremental-development/*

Industry 4.0 Timeline (n.d.). Accessed on 20.11.19.
URL: *http://www.imm.dtu.dk/ jbjo/cps.html*

Interactive Services - Win32 apps (n.d.). Accessed on 10.12.19.
URL: *https://docs.microsoft.com/en-us/windows/win32/services/interactive-services*

Lehnhoff, S., Rohjans, S., Uslar, M. and Mahnke, W. (2012), Opc unified architecture: A service-oriented architecture for smart grids, *in* '2012 First International Workshop on Software Engineering Challenges for the Smart Grid (SE-SmartGrids)', pp. 1–7.

Memory-Mapped Files (n.d.). Accessed on 10.11.19.
URL: *https://docs.microsoft.com/en-us/dotnet/standard/io/memory-mapped-files*

Mono-WCF (n.d.). Accessed on 10.11.19.
URL: *https://www.mono-project.com/docs/web/wcf/*

Mono-Winforms (n.d.). Accessed on 10.11.19.
URL: *https://www.mono-project.com/docs/gui/winforms/*

MoSCoW Prioritisation (2019). Accessed on 10.12.19.
URL: *https://www.agilebusiness.org/page/ProjectFramework*$_{10}$*MoSCoWPrioritisation*

MQTTnet (n.d.). Accessed on 10.11.19.
URL: *https://github.com/chkr1011/MQTTnet*

MQTT Version 5.0 (2019). Accessed on 20.12.19.
URL: *https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html*

Naik, N. (2017), Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http, *in* '2017 IEEE International Systems Engineering Symposium (ISSE)', pp. 1–7.

Named Pipes (n.d.). Accessed on 10.11.19.
URL: *https://docs.microsoft.com/en-us/windows/win32/ipc/named-pipes*

.NET Core overview (n.d.). Accessed on 10.12.19.
URL: *https://docs.microsoft.com/en-us/dotnet/core/about*

note = Accessed on 20.11.19, S. P. (n.d.), 'Opc ua rocks public test server'.
URL: *https://opcua.rocks/open62541-online-test-server/*

OPC and OPC UA explained (n.d.). Accessed on 20.11.19.
URL: *https://www.novotek.com/uk/solutions/kepware-communication-platform/opc-and-opc-ua-explained/*

OPC Router (n.d.). Accessed on 20.12.19.
URL: *https://www.opc-router.com/opc-router-details/*

OPC UA IoT Broker (n.d.). Accessed on 20.12.19.
URL: *https://integrationobjects.com/opc-products/opc-unified-architecture/opc-ua-iot-broker/*

OPC UA .NET (n.d.). Accessed on 10.12.19.
URL: *http://opcfoundation.github.io/UA-.NETStandard/*

Pipes (n.d.). Accessed on 10.11.19.
URL: *https://docs.microsoft.com/en-us/windows/win32/ipc/pipes*

Publish/subscribe systems (n.d.).
URL: *https://aws.amazon.com/pub-sub-messaging/benefits/*

QoS levels (n.d.).
URL: *https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/*

Shelby, Z., Bormann, C. and Hartke, K. (2014), 'The constrained application protocol (coap)'. Accessed on 20.12.19.
URL: *https://tools.ietf.org/html/rfc7252*

Silveira Rocha, M., Serpa Sestito, G., Luis Dias, A., Celso Turcato, A. and BrandÃčo, D. (2018), Performance comparison between opc ua and mqtt for data exchange, *in* '2018 Workshop on Metrology for Industry 4.0 and IoT', pp. 175–179.

Soni, D. and Makwana, A. (2017), A survey on mqtt: A protocol of internet of things(iot).

Sourcetree (n.d.). Accessed on 20.11.19.
URL: *https://www.sourcetreeapp.com/*

Windows Installer (n.d.).
URL: *https://docs.microsoft.com/en-us/windows/win32/msi/using-windows-installer*

Windows Presentation Foundation (n.d.). Accessed on 10.12.19.
URL: *https://docs.microsoft.com/en-us/dotnet/framework/wpf/*