Tampere University

Farhan Akram

# DESIGN AND DEVELOPMENT OF DATA COLLECTION FRAMEWORK FOR SHOP FLOOR DATA

Master of Science Thesis
Faculty of Engineering and Natural Sciences
Examiners: Prof. Jose Martinez Lastra and Luis Gonzalez Moctezuma
March 2023

# ABSTRACT

Farhan Akram: Design and Development of Data Collection Framework for Shop Floor Data
Master of Science Thesis
Tampere University
Master's programme in Automation Engineering, Factory Automation and Robotics
March 2023

---

The rapid pace of technological advancements in the industrial world has transformed the traditional factory floor into a smart, digitalized environment. The advent of advanced and innovative manufacturing techniques, coupled with the Internet of Things (IoT), has made shop floor data an essential source of information for industries to monitor and optimize their operations. The ability to collect, process, and analyse shop floor data has become critical to facilitating the digitalization of factories, which can ultimately improve the competitiveness of industries in the global market.

To address this need, the SHOP4CF project aims to develop a comprehensive platform that can facilitate the digitalization of factories. A key module of this project is the Data Collection Framework (DCF), which collects data from various shop floor devices and sensors, analyses, and process data streams, stores the Event Processing in a database and posts the context data to the FIWARE context broker.

In line with this, the proposed master's thesis seeks to design and develop a data collection framework that can effectively collect and process shop floor data and can facilitate the digitalization of factories, which has become essential for industries to monitor and optimize their processes in the era of Industry 4.0. This framework is scalable, flexible, and modular, accommodating various manufacturing processes' requirements and integrating with different devices and sensors used on the shop floor. Additionally, the framework is equipped with real-time data analysis capabilities, allowing manufacturing managers and engineers to monitor event processing information and optimize them in real-time as well as handle large volumes of data generated by shop floor devices and store it in a centralized database for further analysis.

The thesis aims to contribute to the development of Industry 4.0 and improve the competitiveness of industries in the global market. To achieve this, it proposes research questions to identify the suitable architecture for collecting real-time shop floor data, how the data collection framework and event processing can be utilized to create real-time alerts for a safe shop floor environment, and which industrial communication protocols have been tested for accomplishing data acquisition for industrial shop floors.

Structured into six chapters, this thesis provides a comprehensive analysis of the proposed data collection framework's design, implementation, and testing. The literature review and theoretical background in Chapter two provide an in-depth analysis of vital concepts, such as automation pyramid, ISA-95, communication protocols, MQTT, OPC UA, FIWARE, and MongoDB. The thesis's innovative framework is presented in Chapter three, and Chapter four discusses its implementation. Chapter five examines the testing process and the results obtained from the proposed data collection framework, while Chapter six concludes the thesis and proposes future work to improve the framework.

Keywords: Automation Pyramid, IoT, ISA-95, MQTT, OPC UA, FIWARE, Context Data Management, Industry 4.0, Event Processing

# PREFACE

Firstly, I am very thankful to the Almighty for providing me with the strength, perseverance, and motivation to successfully complete my thesis in time.

Secondly, I would like to express my sincere gratitude to Professor Jose Martinez Lastra for providing me the opportunity to work in his research lab, FAST-Lab, and for guiding me with his extensive knowledge and expertise. I am truly grateful for the chance to have been a part of his team and for the invaluable experience. I would like to thank Luis Gonzalez Moctezuma for teaching me the automation courses and concepts in the best possible way and for supervising my thesis. Special thanks to Anne Korhonen and Seyedamir Ahmadi for the guidance during the design phase and providing helpful feedback on my work.

Lastly, I want to express my heartfelt gratitude to my parents and siblings for their unwavering love, prayers, and support throughout my thesis journey. Their constant encouragement and interest in my progress have been instrumental in keeping me motivated and focused. I am truly blessed to have such a wonderful and supportive family, and I will always be grateful for their presence in my life.

Tampere, 19.03.2023

Farhan Akram

# CONTENTS

# LIST OF FIGURES

# LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| DCF | Data Collection Framework |
| ERP | Enterprise Resource Planning |
| FAST-Lab | Future Automation Systems and Technologies laboratory |
| HTTP | Hypertext Transport Protocol |
| IoT | Internet of Things |
| JSON | JavaScript Object Notation |
| M2M | Machine-to-machine |
| OPC UA | Open Platform Communications United Architecture |
| MQTT | Message Queuing Telemetry Transport |
| REST | Representational State Transfer |
| SDK | Software Development Kit |
| SHOP4CF | Smart Human Oriented Platform for Connected Factories |
| SOA | Service Oriented Architecture |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| URL | Uniform Resource Locator |
| SHOP4CF | Smart Human Oriented Platform for Connected Factories |

# 1. INTRODUCTION

## 1.1 Thesis Background

Digitalization has become a catchword in today's industrial world, and it is transforming the way factories operate. With the advent of advanced and innovative manufacturing techniques and the Internet of Things (IoT), shop floor data has become an essential source of information for industries to monitor and optimize their operations. Shop floor data refers to the data generated by different manufacturing processes, such as machine performance, quality metrics, and production output.

To facilitate the digitalization of factories, it is essential to develop a data collection framework that can collect, process, and analyse shop floor data. The SHOP4CF project aims to develop a comprehensive platform that can facilitate the digitalization of factories. One of the critical modules of this project is the Data Collection Framework (DCF), which collects data from different shop floor devices and sensors. Further, the DFC homogenizes and analyses process and data streams using Event Processing.

This thesis proposes the design and development of a data collection framework that can collect and process shop floor data effectively. The framework should be scalable, flexible, and modular, allowing it to accommodate various manufacturing processes' requirements. It should also be able to integrate with different devices and sensors used on the shop floor.

The developed framework should collect real-time data analysis capabilities, allowing manufacturing managers to monitor the production processes and optimize them in real-time. Real-time data analysis can help industries to reduce downtime, improve product quality, and increase productivity. Also, the framework should be able to handle large volumes of data generated by the shop floor devices and store it in a centralized database for further analysis.

## 1.2 Objectives and Research Questions

The objective of this thesis is to develop a data collection framework that can facilitate the digitalization of factories and help industries to monitor and optimize their operations.

By achieving this objective, this thesis can contribute to the development of Industry 4.0 and improve the competitiveness of industries in the global market. The proposed framework can also serve as a foundation for further research in the field of shop floor data collection and analysis. This thesis addresses following research questions:

- What is the suitable architecture for collecting real-time shop floor data?
- How will the data collection framework and event processing be utilized for creating real-time alerts to provide a safe shop floor environment?
- Which industrial communication protocols have been tested for accomplishing data acquisition for industrial shop floor?

## 1.3 Thesis Structure

The thesis is structured into six chapters. Chapter 1 introduces the thesis background, objectives, research questions, and the significance of the study. The purpose and contribution of each chapter are outlined.

Chapter 2 presents a theoretical background by providing a detailed literature review and incorporates important concepts in the field, such as the automation pyramid, ISA-95, communication protocols, MQTT, OPC UA, FIWARE, and MongoDB.

Chapter 3 focuses on the design of the proposed data collection framework, which was developed based on the identified requirements. The chapter discusses the framework architecture, components, event processing, and data modelling.

Chapter 4 focuses on the implementation of the proposed data collection framework. Specifically, the chapter discusses the implementation of MQTT data adapter and OPC UA adapter to transmit data from shop floor devices to the web-based data storage.

Chapter 5 details the testing process and the results obtained from the proposed data collection framework. The chapter shows testing data collection from both MQTT data adapter and OPC UA data adapter.

Chapter 6 concludes the thesis and highlights future work that can be done to improve the framework.

# 2. THEORETICAL BACKGROUND

## 2.1 Automation Pyramid

The Industrial Automation Pyramid is a framework that describes the levels of control in industrial automation systems.

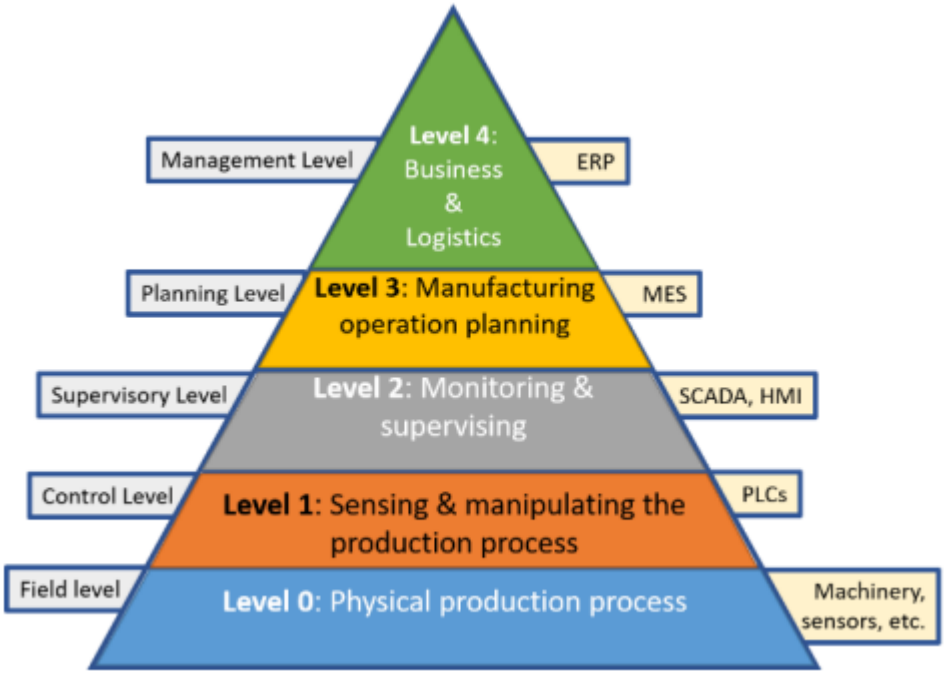


***Figure 1.*** *Automation Pyramid according to ISA 95 model [1].*

At the bottom of the pyramid, Level 0, are the sensors and actuators. These devices are in charge for measuring process variables, such as temperature, pressure, or flow rate, and controlling the process, such as opening or closing valves or turning on or off motors. Sensors and actuators can be analog or digital and are typically connected to a programmable logic controller (PLC) or a distributed control system (DCS) [2].

Level 1 of the pyramid is where control systems are located. These systems receive input from sensors and use that data to control actuators in order to adjust the process. Control systems can be either centralized or distributed, depending on the complexity of the process being controlled. For example, in a simple process, a single control system may be sufficient to control all the actuators. In a more complex process, multiple control systems may be used, each controlling a specific part of the process.

Level 2 of the pyramid is where the supervisory control and data acquisition (SCADA) systems are located. These systems collect data from multiple control systems and provide operators with a graphical interface to monitor and control the process. SCADA systems typically include features such as alarms, data logging, and trending, and can be used for both local and remote monitoring and control.

Level 3 of the pyramid is where manufacturing execution systems (MES) are located. These systems manage the production process and provide data to support decision-making. MES systems typically include features such as quality management, and material management. MES systems can be integrated with other enterprise systems, such as enterprise resource planning (ERP) systems.

Finally, at the top of the pyramid, Level 4, are the enterprise resource planning (ERP) systems. These systems manage business operations, including production planning, inventory management, and accounting. ERP systems can be integrated with other enterprise systems, such as customer relationship management (CRM) systems or supply chain management (SCM) systems.

The Industrial Automation Pyramid is a useful framework for understanding the different levels of control in industrial automation systems. It helps to identify the types of systems and technologies that are required at each level, as well as the data flows between them. This understanding is helpful in designing, building, and maintaining industrial automation systems.

## 2.2 ISA-95

ANSI/ISA 95 is a set of standards developed by the International Society of Automation (ISA) that provides a framework for integrating enterprise and manufacturing systems. The ISA-95 aims to facilitate the integration of business logistics systems, such as Enterprise Resource Planning (ERP) systems, with manufacturing systems, such as Supervisory Control and Data Acquisition (SCADA) systems, to improve efficiency, reduce costs, and increase productivity [2].

ISA-95 is primarily concerned with the integration of the business logistics systems at levels 3 and 4 of the automation pyramid, which include manufacturing operations management and enterprise management. Other ISA standards support the integration of levels 0, 1, and 2, which include sensors, actuators, and control systems.

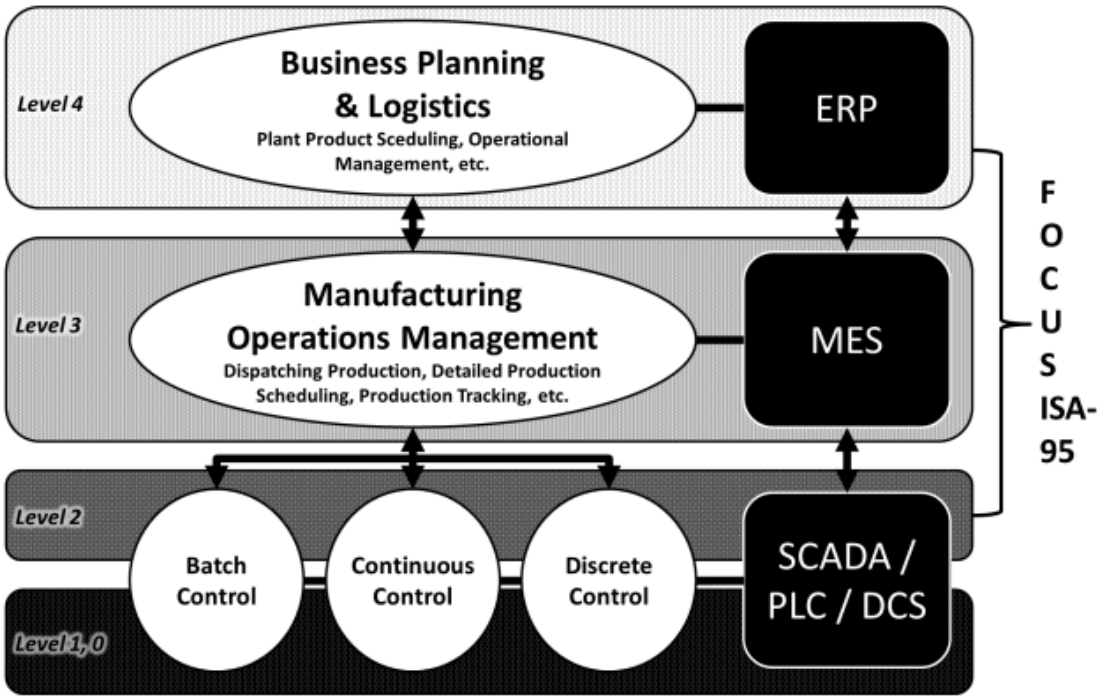


***Figure 2.*** *The levels of control as defined in Part 1 of the ISA-95 standard [2]*

ISA-95 consists of six parts that provide models and terminology for describing the information exchange and activities that occur within and between manufacturing operations systems and enterprise management systems [2]. These parts include:

- *Enterprise control system integration*: This part presents conventional language and object designs that rely on the Purdue Reference Model. It can be employed to identify what data ought to be transmitted between enterprise management systems and manufacturing operational systems.
- *Object model attributes*: This section delineates the characteristics pertaining to every object outlined in the first section. The attributes describe the properties of each object and how they are used in the manufacturing process.
- *Activity models of manufacturing operations management*: This section furnishes models of manufacturing operations management's activity, detailing reference frameworks for delineating production, quality, maintenance, and inventory activities executed on the factory floor. It helps to ensure that all manufacturing operations are coordinated and that the manufacturing process is optimized.
- *Object and attributes for manufacturing operations management integration*: This specification delineates the data transmitted among diverse categories and activities in manufacturing operations management. This ensures that all necessary

information is communicated between systems to optimize manufacturing operations.

- *Business to manufacturing transactions*: This section outlines transactions that determine the procedure for amassing, recovering, transferring, and conserving object data for enterprise-control system fusion. It ensures that all necessary data is collected and shared between the systems involved in manufacturing operations.
- *Messaging service model*: This section delineates the transaction concerning part five, which is a part of a collection of messaging services. It ensures that messages between systems are properly formatted, and that data is transmitted accurately and efficiently.

Together, these parts provide a standardized framework for integrating enterprise and manufacturing systems, enabling greater efficiency and productivity in manufacturing operations.

## 2.3 Communication Protocols

The industrial communication protocols are based on the paradigm of communication such as REST or Pub/Sub. REST is the acronym for Representational State Transfer and is based on the client-server communication model. The servers comprise of distinctively identifiable resources and the client sends the request to the server to retrieve or manipulate the state/values of these resources then the server responds with the response. Each resource is identified using Uniform Resource Identifier.

## 2.4 MQTT

MQTT (Message Queue Telemetry Transport) is a M2M (machine-to-machine) communication protocol used in IoT (Internet of Things) systems. The MQTT protocol works on a publish/subscribe mechanism, and is designed to be lightweight, scalable, reliable, and secure, making it ideal for IoT devices, which typically have resource-constrained networks with low bandwidth [3].

*MQTT Components:*

The main components of MQTT are clients and brokers. Clients in MQTT are responsible for publishing data to brokers, which then distribute the data to other interested clients. The communication between clients and brokers is based on the publish/subscribe pattern, where clients publish data to topics and brokers distribute the data to subscribers

for that topic. This allows for efficient communication, as clients only receive data they have subscribed to, and brokers only need to distribute data to interested clients.

Both the message publisher and the message subscriber are referred to as MQTT clients. An MQTT client can be any device, from a small microcontroller with limited resources to a full-fledged server, that runs an MQTT library and connects to an MQTT broker over a network. The client implementation of the MQTT protocol is simple and streamlined, making it ideal for use in small devices.

The MQTT broker serves as the central hub for all messages. It receives messages from all clients, filters them, determines which clients are subscribed to each message, and sends the messages to the subscribed clients. The broker also stores session data for clients with persistent sessions, handles client authentication and authorization, and is integratable with backend systems.

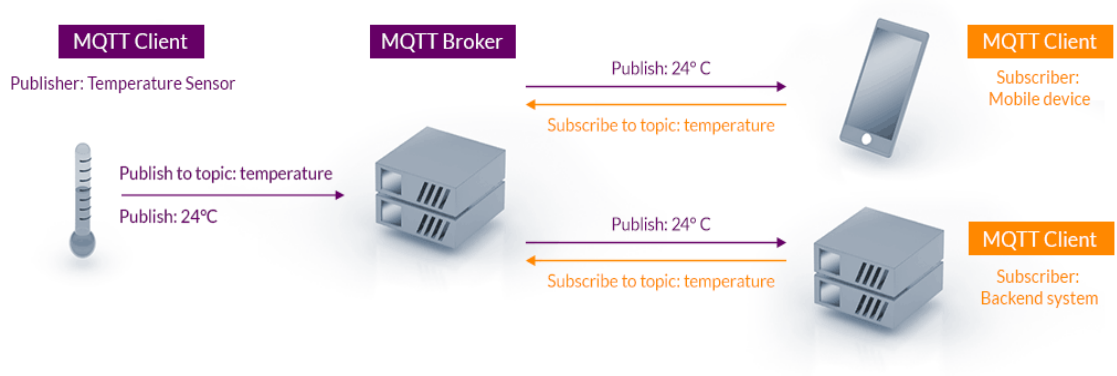


***Figure 3.*** *MQTT Publish / Subscribe Architecture [4]*

The key feature of this pattern is the separation or "decoupling" between the publisher and the subscribers [5]. This decoupling has three main aspects:

- Space decoupling: The publisher and subscriber do not need to know anything about each other. They do not need to exchange information such as IP addresses or port numbers, and they can communicate through a common channel known as a topic.
- Time decoupling: The publisher and subscriber do not need to be running at the same time. The publisher can send messages even when there are no subscribers, and subscribers can receive messages even if the publisher is not running.
- Synchronization decoupling: The publishing and receiving of messages does not affect the normal operation of the publisher and subscriber. They can continue to perform their tasks without being interrupted.

*MQTT Topics*

In MQTT, topics play a vital role in the communication between clients and brokers. A topic is a string in UTF-8 format that the broker uses to filter messages and send them to the appropriate clients [6]. Topics can have one or more levels, separated by a forward slash ("/"). *FastLab/groundfloor/orangeroom/humidity*

It's important to note that topics are case-sensitive, meaning that *"sensor/temperature"* and *"sensor/Temperature"* are treated as two distinct topics.

MQTT clients can subscribe to specific topics or utilize wildcards to receive messages from multiple topics. Wildcards are symbols that allow clients to subscribe to a range of topics. There are two types of wildcards: single-level and multi-level. Single-level wildcards, represented by the "+" symbol, replace a single topic level in the subscription. *FastLab/groundfloor/+/temperature*

Meanwhile, multi-level wildcards, represented by the "#" symbol, cover multiple topic levels. The multi-level wildcard must be placed at the end of the topic, preceded by a forward slash, for the broker to determine which topics match. *FastLab/groundfloor/#*

It's important to note that wildcards can only be used for subscribing to topics, not for publishing messages.

## 2.5 OPC UA

Open Platform Communications Unified Architecture (OPC UA) is a software architecture and communication standard based on Service Oriented Architecture (SOA) for industrial automation. It provides a common platform for communication between various control devices, such as programmable logic controllers (PLCs), human-machine interfaces (HMIs), and other industrial automation devices [7]. Some of the key features and benefits of OPC UA include:

- Platform independence: OPC UA can run on a wide range of operating systems, including Windows, Linux, and macOS, making it easy to integrate into existing automation systems.
- Interoperability: OPC UA provides a common communication protocol that allows different control devices from different vendors to communicate with each other, regardless of the underlying hardware or software.
- Security: OPC UA provides a secure communication mechanism that includes encryption, authentication, and access control, which helps prevent unauthorized access to sensitive data.

- Scalability: OPC UA can be used in small, single-device systems or in large, multi-device systems, making it suitable for a wide range of industrial automation applications.
- Data modelling: OPC UA provides a data modelling capability that allows for the creation of custom data models, which can be used to represent the structure and behaviour of automation devices.
- Publish/Subscribe: OPC UA provides a publish/subscribe mechanism, which allows for real-time communication between control devices, making it well suited for fast-changing industrial automation applications.
- Real-time Communication: OPC UA supports real-time communication for time-sensitive applications, such as process control and monitoring.
- Historical Data Access: OPC UA supplies retrieval of historical data and allows clients to retrieve this data for analysis and reporting.
- International Standards: OPC UA is based on international standards, such as IEC 62541, and has been standardized by the OPC Foundation.

OPC UA is extensively used in industrial automation applications, such as manufacturing, process control, and building automation. It is also used in other industries, such as energy, transportation, and healthcare.

*OPC UA software Layers*

OPC UA, like Classic OPC, employs a model of client-server communication, where programs can either function as UA clients or as UA servers, exposing their own information to other apps. Three software layers make up an OPC UA application, which may be developed using a variety of programming languages, including C/C++, .NET, or Java.

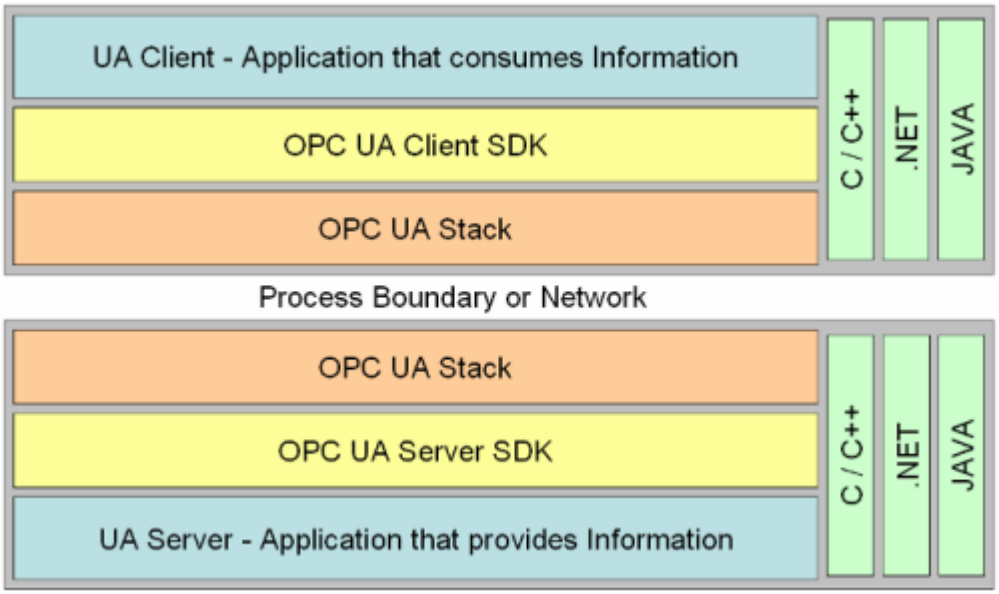


***Figure 4.*** *Software layers of OPC UA [7]*

An OPC UA Application refers to a software package that includes both an OPC UA stack and a software development kit (SDK) designed to map the specific functions of an application to OPC UA. The application's specific functionality is mapped by the SDK while common OPC UA functionality is implemented by the OPC UA client or server SDK. Additionally, various transport mappings of OPC UA are implemented by an OPC UA stack.

The OPC UA Stack consists of three layers: the message encoding layer, the message security layer, and the message transport layer. The message security layer describes how the messages must be protected using the Web Service security standards or a UA binary version of the Web Service standards, while the message encoding layer provides the Service parameters are serialized into binary and XML formats. The message transport layer stipulates the network protocol that is used, such as UA TCP or HTTP and SOAP for Web Services. [7].

For UA client and UA server applications, the UA Stacks offer language-dependent APIs, although all Services provided in OPC UA have parameters that are based on a common abstract Service. This means that the parameters used in these Services are equivalent in structure and functionality. With implementations in different programming languages, the OPC Foundation created and maintains UA Stacks, which provide coverage for the primary development environments. Overall, OPC UA software layers facilitate faster interoperability and reduce development effort for OPC UA applications.

*Nodes and References*

Nodes and the references between them are the fundamental modelling ideas in OPC UA. Depending on their intended use, nodes can have various node classes. Instances, types, and more may all be represented using nodes. Nodes are described through attributes, and the attributes that a node may have are determined by its node class. The node class of a node determines its characteristics and functionality, and thus the attributes that it can possess. The most crucial notion for referencing and transferring information in the Services is the Node Id, which distinctively associates the Node within the server. While exploring or searching the Address Space, the server returns Node Ids, which clients use to address Nodes in Service calls. There can be alternative Node Ids that may be used to address a Node. Even if the node was accessed by a different node id, the canonical node id may be found by reading the node id attribute. Node Ids contain a Namespace that enables them to be specifically defined by various naming authority. These naming authorities may be systems, vendors, or organisations. [8].

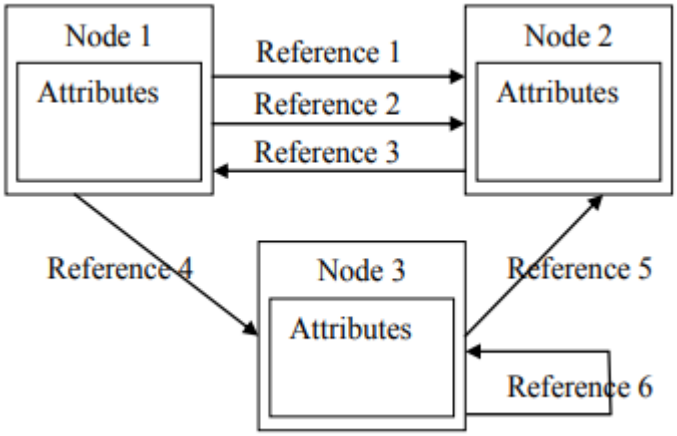


***Figure 5.*** *Nodes and References between Nodes [8]*

*Reference Types*

In OPC UA, a Reference connects two Nodes, but it cannot be accessed directly or have attributes or properties. Instead, the semantic of a Reference is defined by a Reference Type, which is used to expose how Nodes are connected. The OPC UA specification specifies a set of Reference Types, which can be extended by an OPC UA server to reveal specific semantics for References. Reference Types are arranged hierarchically and are characterized as Nodes within the Address Space. The Browse Name and Display Name of a Reference Type define its semantic and must be unique and localized. Information regarding the References used by an OPC UA server can be obtained by OPC UA clients by accessing specific Nodes within the Address Space. Reference Types can be specialized with more specialized types in a Reference Type hierarchy [8].

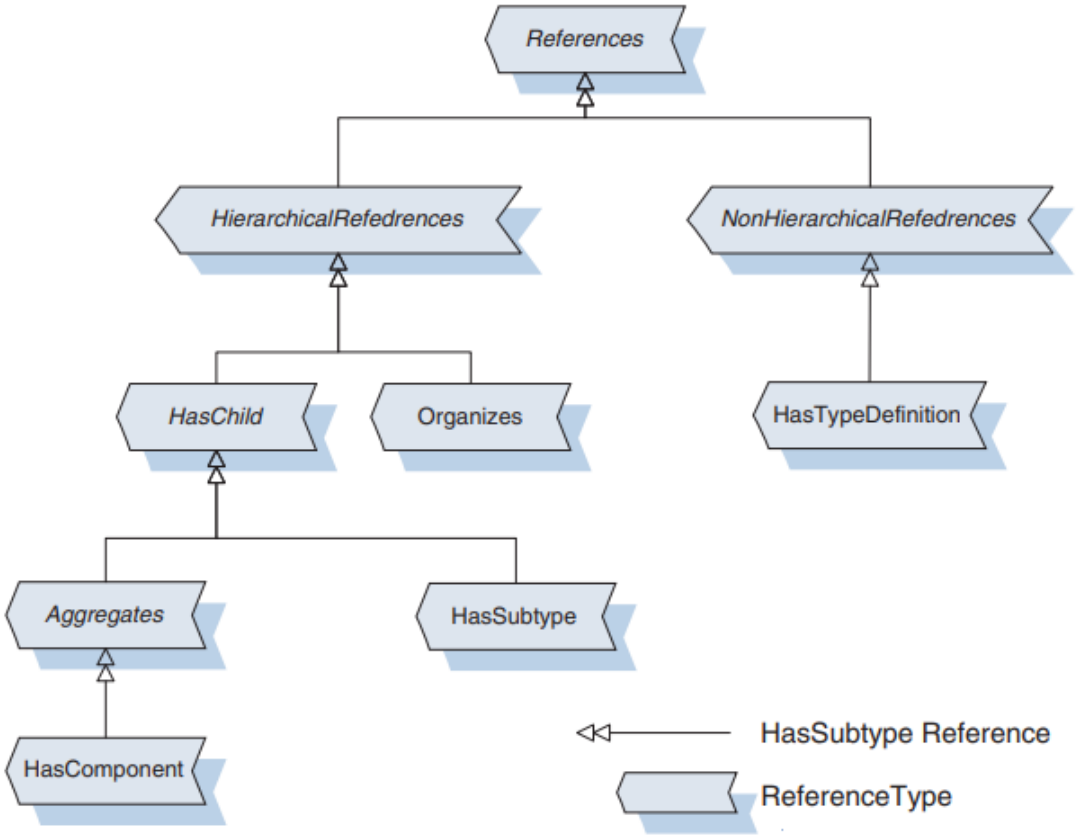


***Figure 6.*** *Base Reference Type hierarchy [8]*

*Built-in and Simple Data Types*

The built-in Data Types of OPC UA have a defined hierarchy, and their handling is specified in the OPC UA specification. The Address Space does not require further information on these Data Types. The behaviour of simple Data Types is governed by their supertypes. The following figure illustrates the data type structure for various basic data types as well as built-in data types.

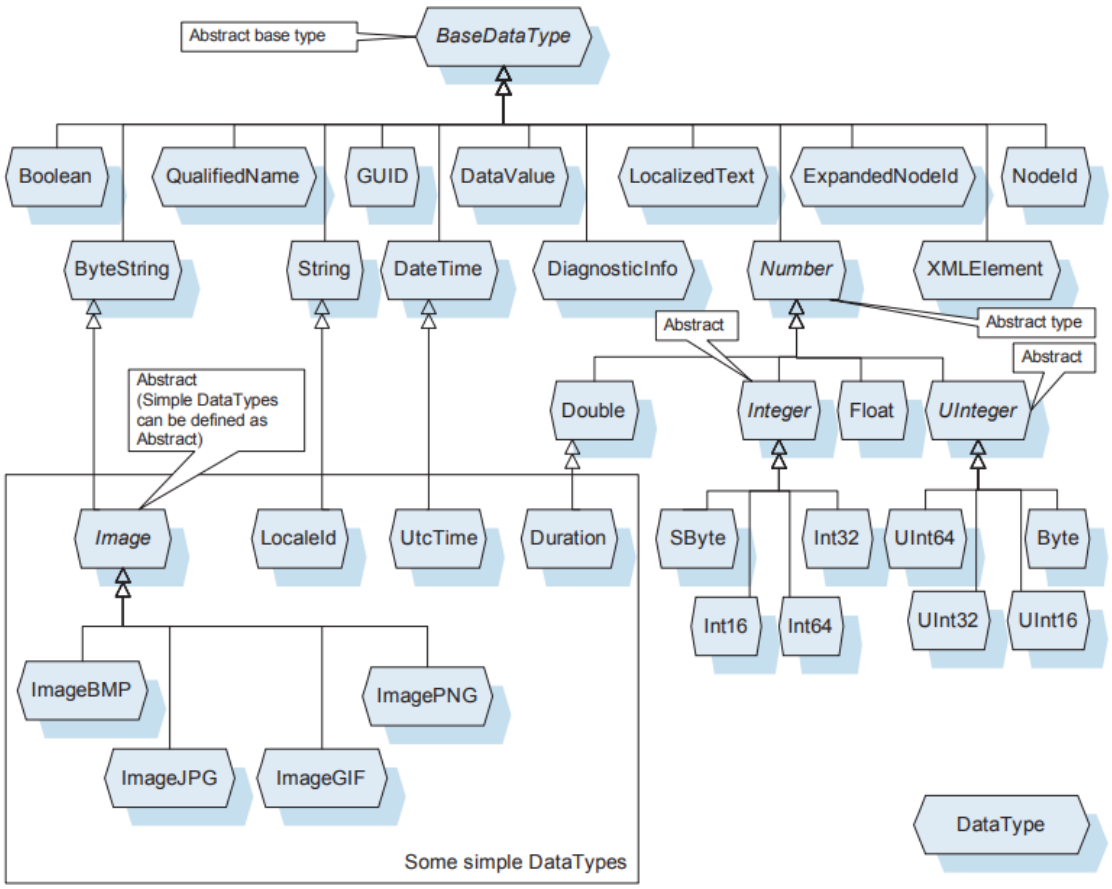


***Figure 7.*** *The Data Types hierarchy [8]*

In object-oriented programming languages, a class is a construct that allows for the creation of named entities that can be accessed and manipulated by their given names, is analogous to a complex Object Type with Instance Declarations in OPC UA. For instance, the Employee class includes two methods, Increase_salary and Salary_after_tax, as well as three public variables: Name, Salary, and Address. Similar to this, the Address class includes two public variables called Street and City. The Employee class is mapped to an Object Type named Employee Type in OPC UA, whereas the Address class is assigned to an Object Type called Address Type. Additionally, The Salary and Name attributes of the Employee Type are accompanied by a distinct Address Type instance that encapsulates the Address information. [8].

One key difference between OPC UA ObjectTypes and typical object-oriented programming languages is that there is no universal method for exposing an OPC UA Method's implementation. However, OPC UA is more versatile since it enables the addition of components to instances without regard to type specification, such as adding a Variable called Award to an Employee instance. It is possible to accomplish this without changing the Object Type or making a subclass of it. A Zip Code Variable might also be added by

the Employee Type underneath the Address Object on InstanceDeclarations. To put information in that location, one would normally need to subtype Address in an object-oriented programming language.

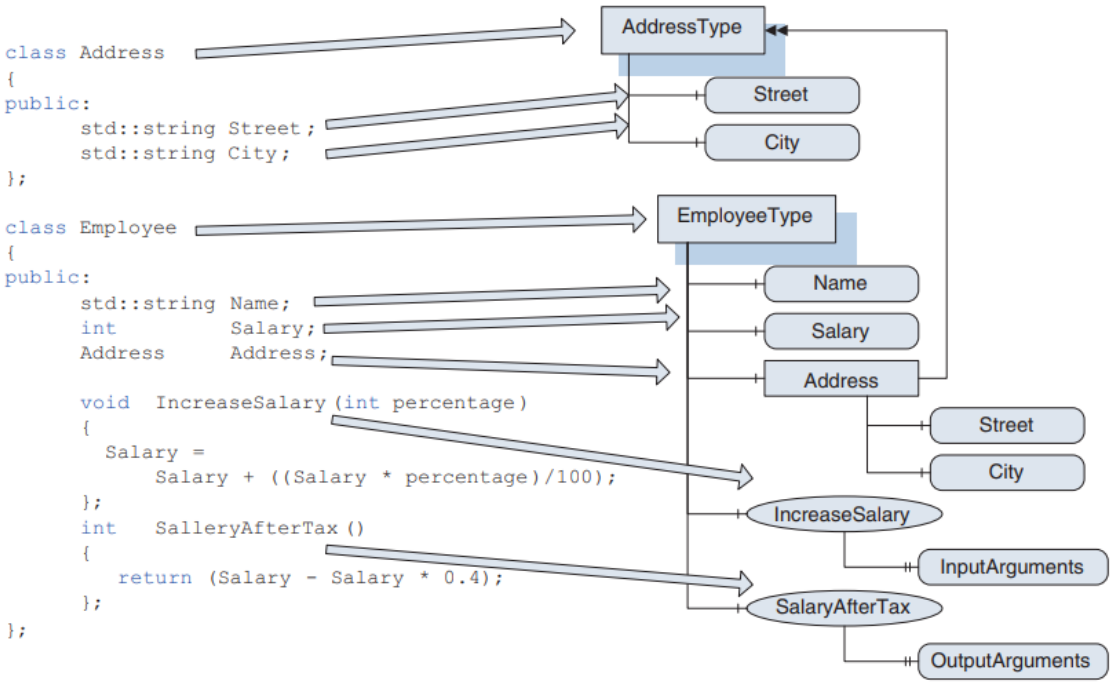


***Figure 8.*** *Transforming a class designed with object-oriented principles into an ObjectType [8]*

## 2.6 FIWARE

FIWARE is an open-source platform that provides a set of standardized APIs and components for building and deploying smart applications and services for the Internet of Things (IoT), big data, and cloud computing. It offers a common framework that enables the integration of different systems and data sources, and the development of interoperable and scalable solutions.

The FIWARE platform consists of various software components and APIs that provide developers with a standardized way to access and integrate various technologies, such as data management, cloud computing, IoT devices, and analytics tools. This allows developers to build scalable and interoperable smart applications and services.

FIWARE has an open architecture i.e., it can be used with a variety of programming languages and platforms. It also supports both cloud and on-premises deployment, providing flexibility to developers and organizations.

The following figure illustrates the core components of FIWARE:

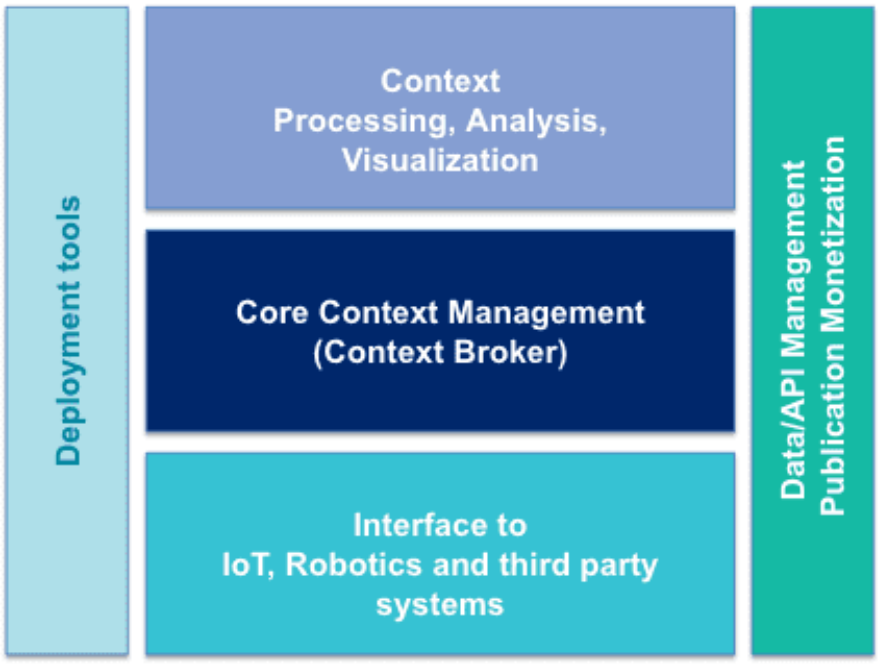


***Figure 9.*** *FIWARE Components [9]*

*Core Context Management (Context Broker)*

The Core Context Management component, also known as the Context Broker, is the core of the FIWARE platform. It is responsible for managing and storing context information, including data from IoT devices, sensors, and other sources. The Context Broker provides a standardized interface for querying and updating context data, and it uses a publish-subscribe model to distribute context data in real-time. Context information comprises of entities (e.g., a robot) and their attributes (e.g. the speed or location of the robot) [10].

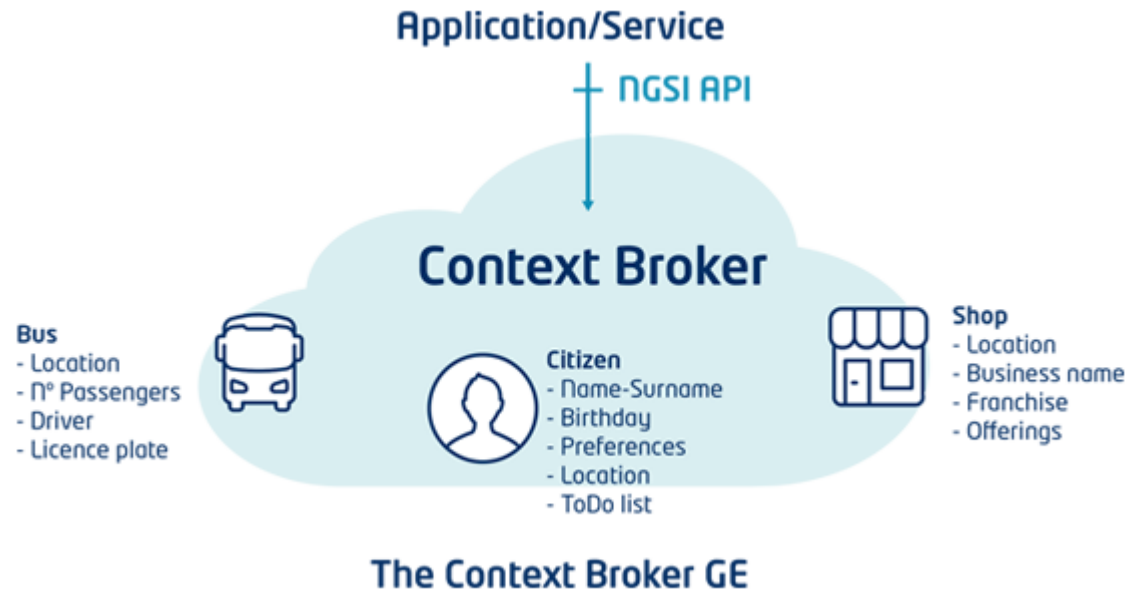


***Figure 10.*** *FIWARE Core Context Management [11]*

*Interface with IoT, Robots and Third-Party Systems*

The Interface with IoT, Robots, and Third-Party Systems component, also known as the IoT Agent, provides a standardized interface (a bridge) for connecting to and communicating with IoT devices, robots, and other third-party systems. The IoT Agent supports various protocols, including MQTT, CoAP, OPC UA, and HTTP, and it provides a framework for developing custom adapters to support other protocols.

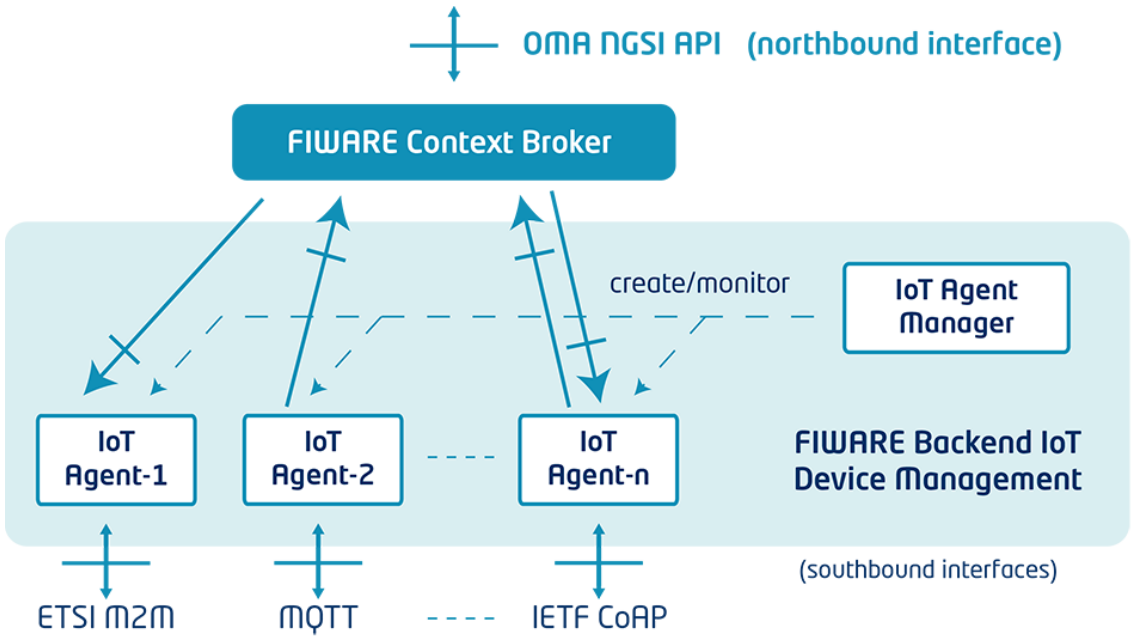


***Figure 11.*** *FIWARE IoT Agents [12]*

*Context Processing, Analysis and Visualization*

The Context Processing, Analysis, and Visualization component provides tools for processing, analysing, and visualizing context data. It includes tools for real-time stream processing, batch processing, and machine learning. The component also provides a variety of visualization tools, including dashboards, charts, and maps.

*Context Data/API Management, Publication and Monetization*

The Context Data/API Management, Publication, and Monetization component provides tools for managing and publishing context data and APIs. It includes a data marketplace for buying and selling context data, and it provides a framework for monetizing context data and APIs.

*Deployment Tools*

The Deployment Tools component provides tools for deploying and managing FIWARE instances. It includes tools for configuring and scaling FIWARE instances, and it provides a framework for managing deployments in a cloud environment. Majority of the FIWARE components are available as Docker Images [13].

*Device To NGSI Mapping*

The Device to NGSI Mapping is a mechanism that maps IoT devices to entities in the NGSI (Next Generation Service Interface) system. This mapping assigns each device as an entity associated with a context provider. The ID of the device is associated with the ID of the entity, and the type of the entity is chosen depending on the protocol utilized. Users can configure the entity name and type either through device pre-provisioning or type configuration [14].

Measurements obtained from the device are mapped to different attributes, with their names and types configurable by the user either globally or on a per-device basis. Three different behaviours can be exhibited by device measures:

Active attributes refer to the measurements that are gathered by a device and transferred to an IoT agent. These measurements are then transmitted to the Context Broker as updateContext requests.

In contrast, lazy attributes refer to sensors that do not actively collect data but instead wait for the IoT agent to request it. In this case, the IoT agent enrols itself as a Context Provider for every passive attribute of the device.

Commands involve modifying an attribute in the entity of the device, for which the IoT Agent is registered as a Context Provider. When a command is received, the IoT agent

promptly replies to the Context Broker and assumes the responsibility of communicating with the device, updating its status and information attributes as it obtains information about the progress of the command.

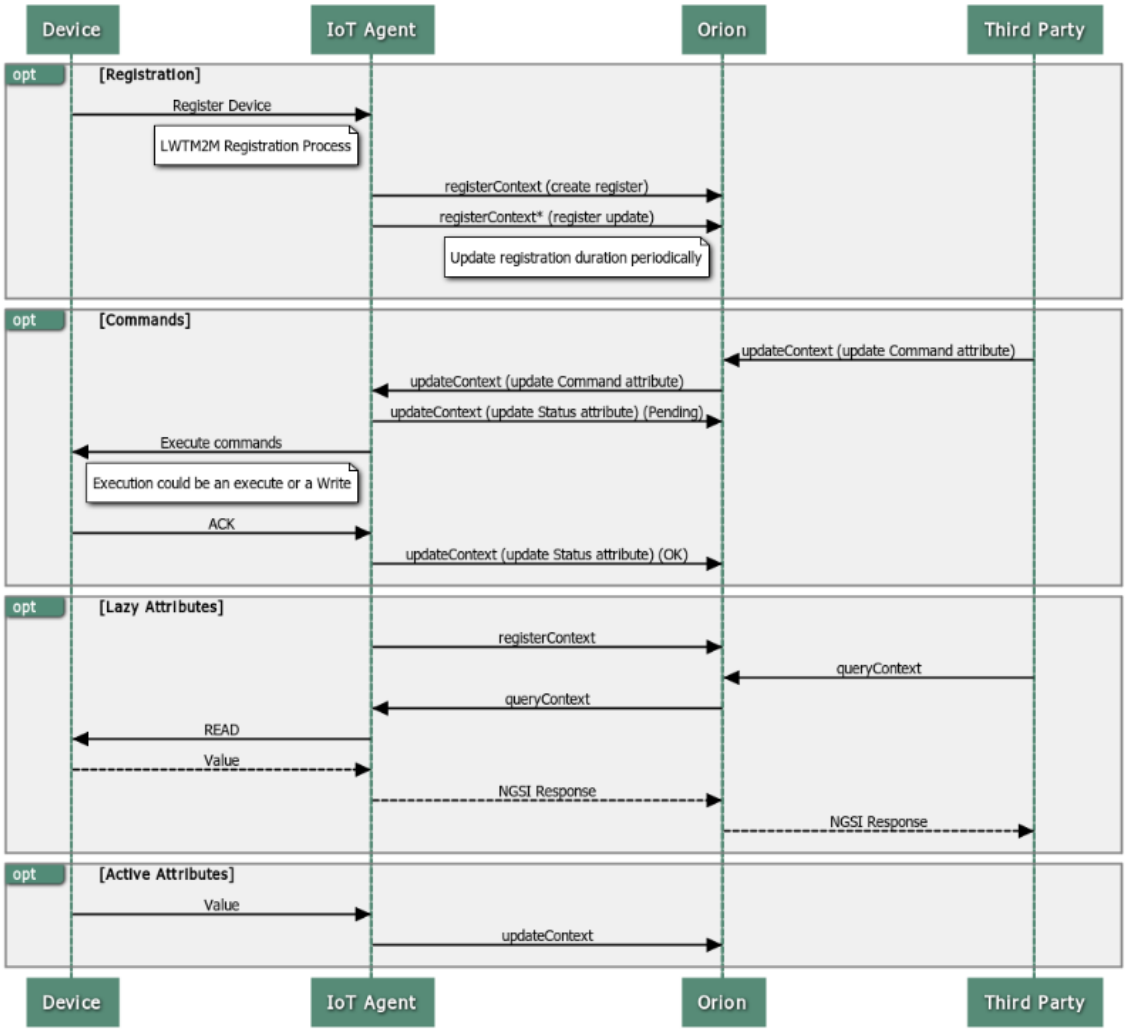


***Figure 12.*** *NGSI interactions by IoT Agent with Context Broker [14]*

*Registration*

When a device is registered to the system, the IoT Agent obtains information about its attributes and registers itself as a Context Provider for any lazy attributes. The registration ID is then kept in the IoT Agent's device registry together with other device information.

Upon the removal of a device from the IoT Agent, its registration becomes deactivated by setting its expiration date to 1 second, as context registrations cannot be removed in NGSI9.

*Commands*

Commands in the IoT Agent are treated as updates over a lazy attribute, similar to the way lazy attributes are handled. When a command is received, it is forwarded by the Context Broker to the IoT Agent, which then interacts with the device to carry out the requested action. The command value includes parameters for the command and metadata, and in the case of an NGSI-LD command, a datasetId may also be provided.

There are two distinct kinds of commands: push commands and poll commands. Push commands are immediately forwarded to the device by the IoT Agent, while poll commands are stored by the IoT Agent for devices that cannot be online all the time. The IoT Agent offers functions to manage command storage and retrieval for devices marked as "polling devices". To enable push commands, devices should be furnished with a suitable protocol endpoint that can be accessed by the IoT agent. [14].

# 3. DESIGN

This chapter explains the framework design of the Data Collection Framework (DCF).

## 3.1 Requirements

The DCF is a critical component of the factory shop floor and enterprise resources, as it plays a central role in enabling the collection and organization of data from these sources. The functional requirement of the DCF is to provide a systematic and efficient method for gathering data from the factory shop floor and enterprise resources and organizing it in a way that is useful for analysis and decision-making [15]. This includes identifying the types of data that need to be collected, defining the methods for collecting and storing data, and developing processes for ensuring the accuracy and integrity of the collected data. Additionally, the DCF is also designed to be flexible and adaptable, so that it can be easily modified to meet changing data collection needs over time.

## 3.2 Framework Architecture

The DCF has been specifically designed to provide a scalable and easy-to-use solution for shop floor data and alert management. The DCF is intended to be a flexible and adaptable tool that can be easily configured to meet the specific needs of different shop floors and processes. One of the key requirements of the project was to ensure that the DCF could effectively handle context data, which is data that provides context or background information about a particular process or event. In order to meet this requirement, the DCF was designed with robust data management and analysis capabilities, including an Event Processing that is capable of identifying patterns and correlations in real-time. Overall, the goal of the DCF is to provide a reliable and easy-to-use solution for shop floor data and alert management, enabling organizations to make more informed decisions and optimize their operations.

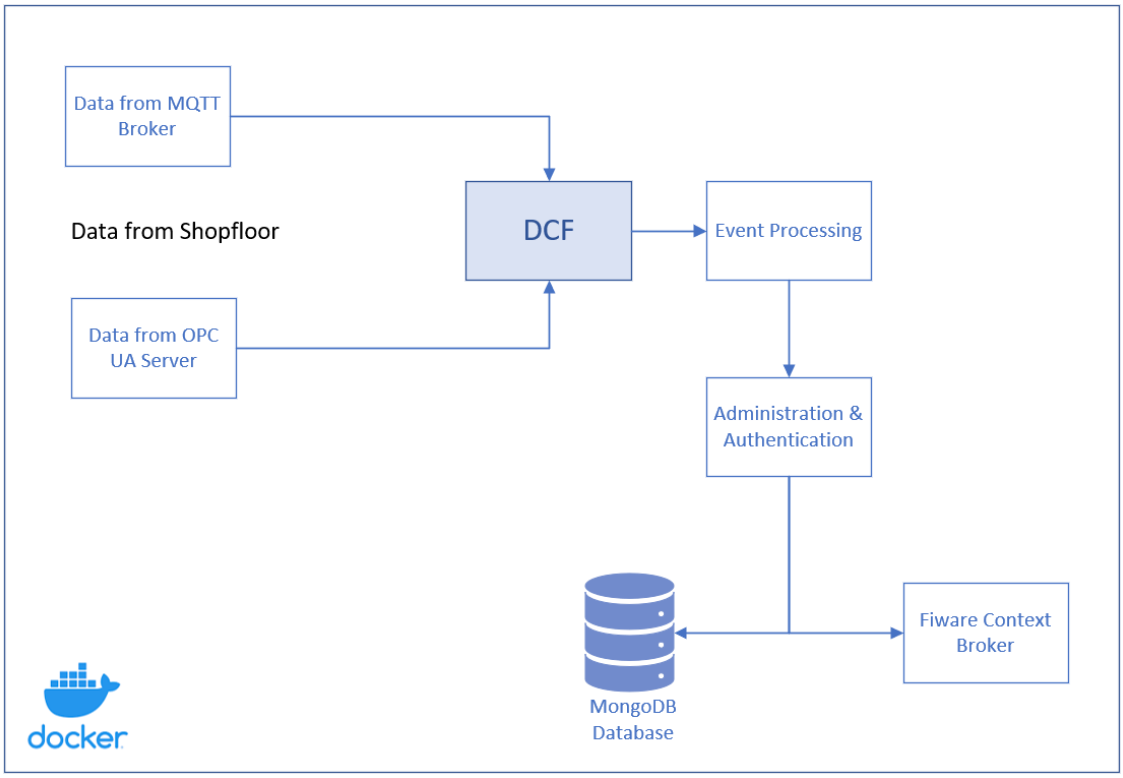


***Figure 13.*** *Data Collection Framework Architecture*

The DCF enables the collection of shop floor data from a variety of sources, including MQTT brokers and OPC UA servers. MQTT is a publish-subscribe messaging protocol that is widely used in the Internet of Things (IoT) and machine-to-machine (M2M) communications. It allows devices to communicate with each other and with centralized servers, enabling the collection of data from a large number of different sources. OPC UA is a standardized communication protocol that is designed for secure and reliable data exchange in industrial automation and other systems. It allows different devices and systems to interoperate and exchange data, enabling the collection of data from a wide range of industrial equipment and systems. By using MQTT and OPC UA, the DCF is able to gather data from a diverse range of shop floor sources, providing a comprehensive view of shop floor operations.

## 3.3 Components

The DCF consists of two main components for the collection of shop floor data and four plugins for the collection of ERP data. The MQTT Data Adapter and OPC UA Data Adapter are responsible for gathering real-time data from MQTT brokers and OPC UA servers. The four plugins for ERP data are responsible for gathering data from enterprise

resource planning (ERP) systems, e.g., task information and restrains, which are used to manage and optimize various processes.

## 3.4 Event Processing

The DCF is designed to support real-time processing of events for shop floor data. This feature allows the DCF to monitor physical characteristics such as time, pressure, and temperature in real-time, and to trigger certain actions or alerts if the values of these characteristics exceed pre-defined thresholds. For example, if the temperature on the shop floor exceeds a certain value, the DCF will trigger an alert and save the timestamp in the database. This real-time processing capability is an important aspect of the DCF, as it enables organizations to respond quickly to changes on the shop floor and to prevent problems from escalating.

## 3.5 Data Modelling

The SHOP4CF data models are used as the foundation for the data modelling. These models provide a standardized way of representing information about products, services, and other entities within the context of the IoT. The SHOP4CF models are based on the FIWARE NGSI-LD specification [16], which is a standard for representing and exchanging data in the IoT using the Linked Data format. The use of these standards allows for interoperability between different systems and enables the integration of data from multiple sources. The data modelling for the DCF has been designed to align with the SHOP4CF models and the FIWARE NGSI-LD specification, ensuring compatibility with other systems that use these standards.

```
{
    "id": "urn:ngsi-ld:Alert:company-xyz:pred-maint-3x29md89",
    "type": "Alert",
    "category": {
        "type": "Property",
        "value": "predictiveMaintenance"
    },
    "subCategory": {
        "type": "Property",
        "value": "incorrectPaintingProcessInKTL"
    },
    "validTo": {
        "type": "Property",
        "value": {
            "@type": "DateTime",
            "@value": "2017-01-02T10:25:55.00Z"
        }
    },
    "description": {
        "type": "Property",
        "value": "Skid no. 12345 needs maintenance"
    },
    "location": {
        "type": "GeoProperty",
        "value": {
            "type": "Point",
            "coordinates": [-3.712247222222222, 40.423852777777775]
        }
    },
    "dateIssued": {
        "type": "Property",
        "value": {
            "@type": "DateTime",
            "@value": "2017-01-02T09:25:55.00Z"
        }
    },
    "alertSource": {
        "type": "Relationship",
        "object": "urn:ngsi-ld:Asset:company-xyz:skid-12345"
    },
    "source": {
```


***Figure 14.*** *SHOP4CF Alert Data model [17]*

# 4. IMPLEMENTATION

Most of the implementation work is done in Python language. Python language has been chosen to implement the DCF since it is widely applicable and is easier to understand and to use. It has a large and active community of users, as well as a collection of libraries and frameworks. Additionally, Python has a simple and readable syntax, which makes it a great language for beginners and experienced programmers alike.

## 4.1 YAML Configuration file

Firstly, the program reads a configuration file. The configuration file is in YAML file and includes the key information for MQTT server broker address, OPC UA server address, MongoDB, FIWARE context broker and others parameter set by user for event processing.

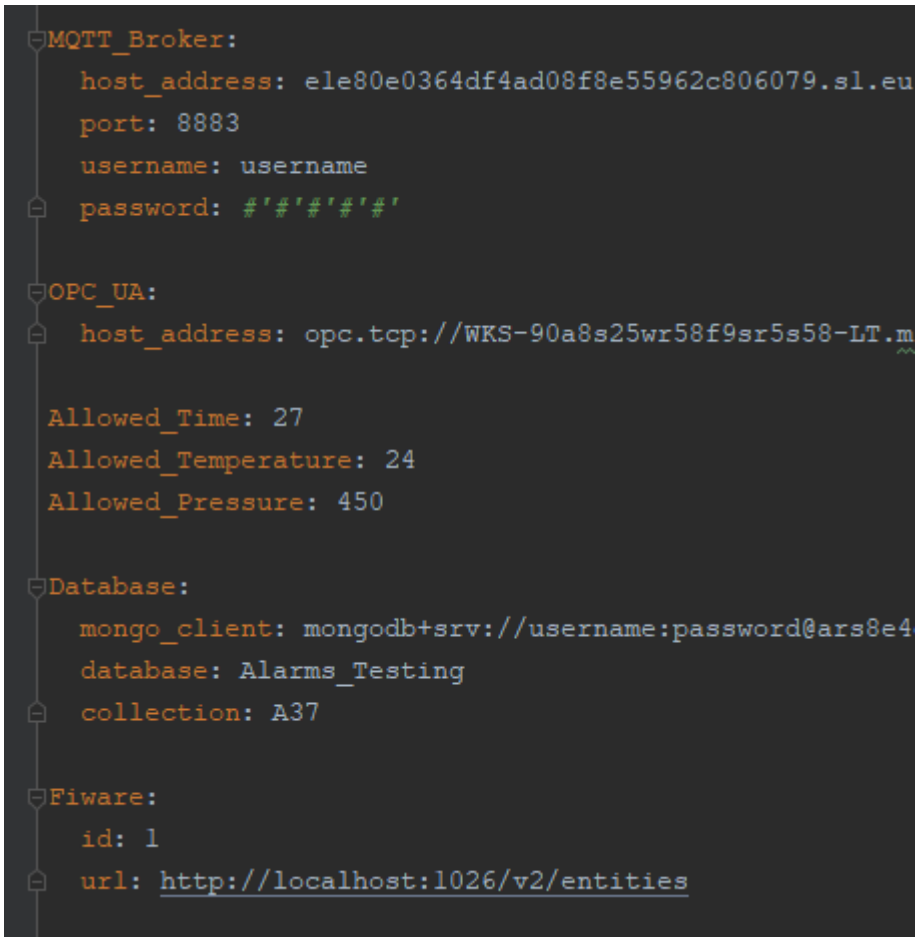


***Figure 15.*** *Some attributes of YAML configuration file for the data adapters*

## 4.2 MQTT Data Adapter

The YAML configuration attributes are imported to Python and are stored in the meaningful parameters. This allows the DCF to access and use the data from the configuration file in a structured and organized way. The process of importing the YAML data into Python involves parsing the file and converting it into a data structure that can be used by the program. In the following figure MongoDB details for the database, FIWARE URL and MQTT broker information is being used to connect respectively.

```
with open('config.yaml') as cf_file:
    config = yaml.safe_load(cf_file.read())

### Mongodb Database details from configuration file
client = MongoClient(config["Database"]["mongo_client"])
db = client.get_database(config["Database"]["database"])
col = db.get_collection(config["Database"]["collection"])

### Allowed time for the Workstation
ALLOWED_TIME = config["Allowed_Time"]

### Fiware Details
id = config["Fiware"]["id"]
url_fiware = config["Fiware"]["url"]

# enable TLS for secure connection
client.tls_set(tls_version=mqtt.client.ssl.PROTOCOL_TLS)

# set username and password
client.username_pw_set(config["MQTT_Broker"]["username"], config["MQTT_Broker"]["password"])

# connect to MQTT Cloud on port 8883 (default for MQTT)
client.connect(config["MQTT_Broker"]["host_address"], config["MQTT_Broker"]["port"])
```

***Figure 16.*** *YAML attributes to Python variables*

After the configuration file is converted to meaningful data, the MQTT data adapter connects with the MQTT broker and the data can be fetched. Certain functions have been implemented that acknowledge after successful connection and subscription to MQTT Broker.

```
def on_connect(client, userdata, flags, rc, properties=None):
    print("CONNACK received with code %s." % rc)

# print which topic was subscribed to
def on_subscribe(client, userdata, mid, granted_qos, properties=None):
    print("Subscribed: " + str(mid) + " " + str(granted_qos))

# print message, useful for checking if it was successful
def on_message(client, userdata, msg):
    global id
    print(f"Topic: {msg.topic}\nMessage: {str(msg.payload)}")
    payload = msg.payload.decode('ASCII')
```

***Figure 17.*** *Some functions of the MQTT data adapter*

## 4.3 OPC UA Adapter

The configuration parameters are also read for OPC UA Adapter in the YAML format and are imported in Python variables.

The OPC UA adapter is connected to the OPC UA Server using the python library “opcua”, and the sever URL. The node information is then retrieved and processed.

```
try:
    client = Client(url)
    client.connect()
    print("Client Connected")

except Exception as err:
    print("Error while creating the connection ", err)
    sys.exit(1)
```

```
pressure = client.get_node("ns=3;i=1005")
Pres = pressure.get_value()
print(f"Pressure: {Pres}")

temperature = client.get_node("ns=3;i=1002")
Temp = temperature.get_value()
print(f"Temperature: {Temp}")
print()
```

```
    Time = client.get_node("ns=3;i=1001")
    elapsed_time = Time.get_value()
    print(f"Time:  {elapsed_time}")
```

***Figure 18.*** *OPC UA adapter connecting to the OPC UA server and retrieving the node information.*

## 4.4 Event Processing

The event processing has also been implemented for both data adapters. Live values of e.g. time, temperature and pressure will be processed from OPC UA servers and MQTT brokers and if the values are exceeding than defined, then the event will be stored in the database.

```
Time = client.get_node("ns=3;i=1001")
elapsed_time = Time.get_value()
print(f"Time:  {elapsed_time}")

if elapsed_time >ALLOWED_TIME:
    Description = "****************\nSome Thing Wrong\n****************"
    print(Description)
    Time_Stamp = datetime.datetime.now().strftime('%Y-%m-%dT%H:%M:%S.%f')[:-3]+"Z"
    print(f"Time_Stamp: {Time_Stamp}")

    data = {
        "Time_Stamp": Time_Stamp,
        "Current_Value": elapsed_time,
        "Description": Description
    }
    col.insert_one(data)
```

***Figure 19.*** *A snapshop of Event Processing*

## 4.5 Context data to FIWARE Context Broker

The data is also transformed in the context data and posted to FIWARE context broker for other FIWARE entities.

```
context_data = {
    "type": "Alert",
    "id": "urn:ngsi-ld:Alert:ARC:" + str(id),
    "category": {
        "type": "Property",
        "value": "Weather"
    },
    "dateIssued": {
        "type": "Property",
        "value": {
            "@type": "DateTime",
            "@value": str(Time_Stamp)
        }
    }
}
json_context_data = json.dumps(context_data)
headers = {'Content-Type': 'application/json'}

try:
    r = requests.post(url_fiware, data=json_context_data, headers=headers)
    print("posted")
    print(r.text)
except Exception as err:
    print("Error while posting to fiware ", err)
id += 1
```

***Figure 20.*** *An Example of posting the Context data to FIWARE Context Broker*

## 4.6 Containerization

Containerization is a process of packaging and deploying software applications in a self-contained environment, known as a container. Containers provide a way to package and distribute an application and its dependencies, such as libraries and runtime, in a single unit, which can be easily deployed and run on any host with the necessary containerization software. Docker was used as a containerization platform for the data adapters.

```
FROM python:3

ADD mqtt-client.py .

COPY config.yaml .

RUN pip install pymongo pymongo[srv] cryptography paho.mqtt pyyaml requests
RUN apt-get update

CMD ["python", "-u", "./mqtt-client.py"]
```

***Figure 21.*** *Docker file for the MQTT Data Adapter*

```
FROM python:3.8.10

ADD opc_ua_client.py .

RUN pip install opcua pymongo pymongo[srv] cryptography paho.mqtt

CMD ["python", "-u","./mqtt_client.py"]
```

***Figure 22.*** *Docker file for the OPC UA Adapter*

# 5. TESTING & RESULTS

For the shop floor data, the testing of the OPC UA and the MQTT data adapters have been tested with various OPC UA servers and MQTT brokers.

## 5.1 OPC UA Adapter

Firstly, a local OPC UA server was developed or the testing purpose.

The OPC UA server is developed using the “opcua” python library.

```
server = Server()

url = "opc.tcp://127.0.0.1:12345"
server.set_endpoint(url)

name = "OPCUA_SERVER"
address_space = server.register_namespace(name)

objects = server.get_objects_node()

param = objects.add_object(address_space, "Parameters")

temp = param.add_variable(address_space, "Temperature", 0)
pressure = param.add_variable(address_space, "Pressure", 1)
time = param.add_variable(address_space, "Time", 0)

temp.set_writable()
pressure.set_writable()
time.set_writable()

server.start()
print()
print(f"Server started at {url}")
print()
```

***Figure 23.*** *OPC UA server*

The OPC UA server was used to publish the dummy temperature and pressure values.

```
Temperature = random.uniform(22, 25)
Pressure = random.uniform(200, 500)
Time = datetime.datetime.now()

temp.set_value(Temperature)
print(f"Temperature:  {temp.get_value():.2f}")

pressure.set_value(Pressure)
print(f"Pressure: {pressure.get_value():.2f}")

time.set_value(Time)
print(f"Time: {time.get_value()}")
print()

sleep(2)
```

***Figure 24.*** *Dummy Temperature and Pressure values on the OPC UA server*

```
Server started at opc.tcp://127.0.0.1:12345

Temperature:  24.56
Pressure: 325.96
Time: 2022-11-26 15:30:00.962298

Temperature:  24.77
Pressure: 317.89
Time: 2022-11-26 15:30:02.967138

Temperature:  23.07
Pressure: 460.48
Time: 2022-11-26 15:30:04.977158

Temperature:  24.76
Pressure: 364.34
Time: 2022-11-26 15:30:06.987946
```

***Figure 25.*** *Output of the OPC UA server*

The OPC UA data adapter is connected to the OPC UA server.

```
Client Connected

Temperature:  24.54
****************
Temperature Rise Recorded
****************
Pressure: 421.27
Time: 2022-11-26 16:17:20.740660

Temperature:  24.55
****************
Temperature Rise Recorded
****************
Pressure: 417.47
Time: 2022-11-26 16:17:22.756573

Temperature:  23.72
Pressure: 374.23
Time: 2022-11-26 16:17:24.758495

Temperature:  23.35
Pressure: 218.44
Time: 2022-11-26 16:17:26.773077

Temperature:  22.02
```

***Figure 26.*** *Output of the OPC UA data adapter*

After the local OPC UA server testing, the testing of OPC UA data adapter was also done for Prosys OPC UA Server.

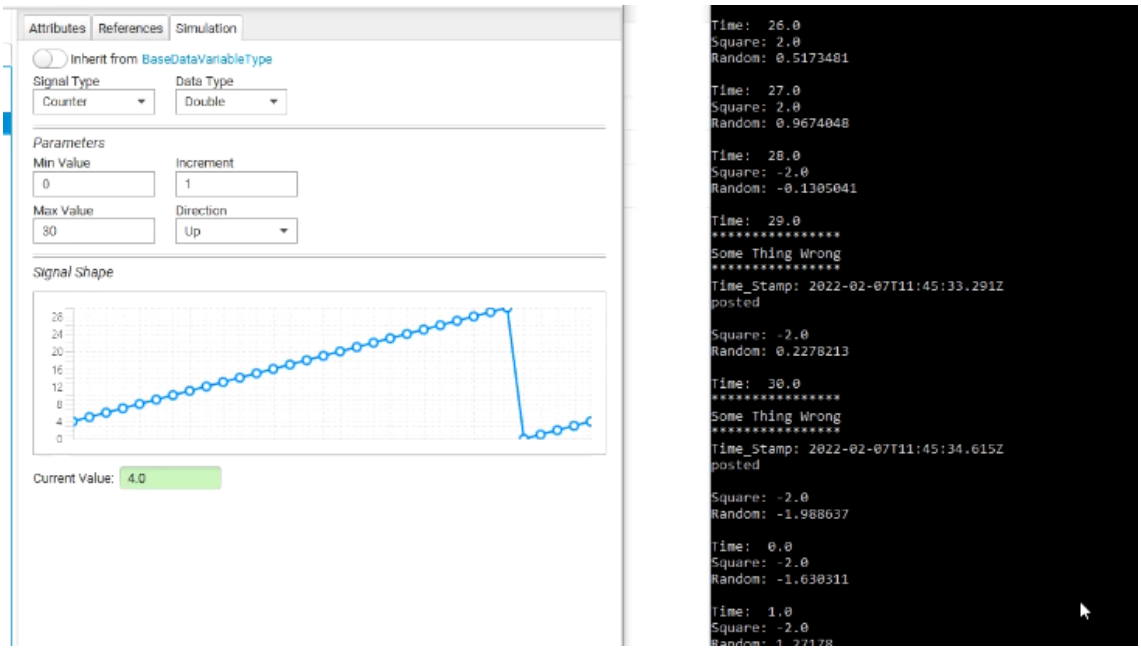


***Figure 27.*** *Testing with Prosys OPC UA server: Prosys OPC UA Server (Left side), OPC UA Data Adapter (Right side)*

If the Temperature or Pressure is more or less than the predefined range, it is recorded as an event is and is stored in the database.


***Figure 28.*** *MongoDB Database for the recorded events*

Context data being published to FIWARE context broker.

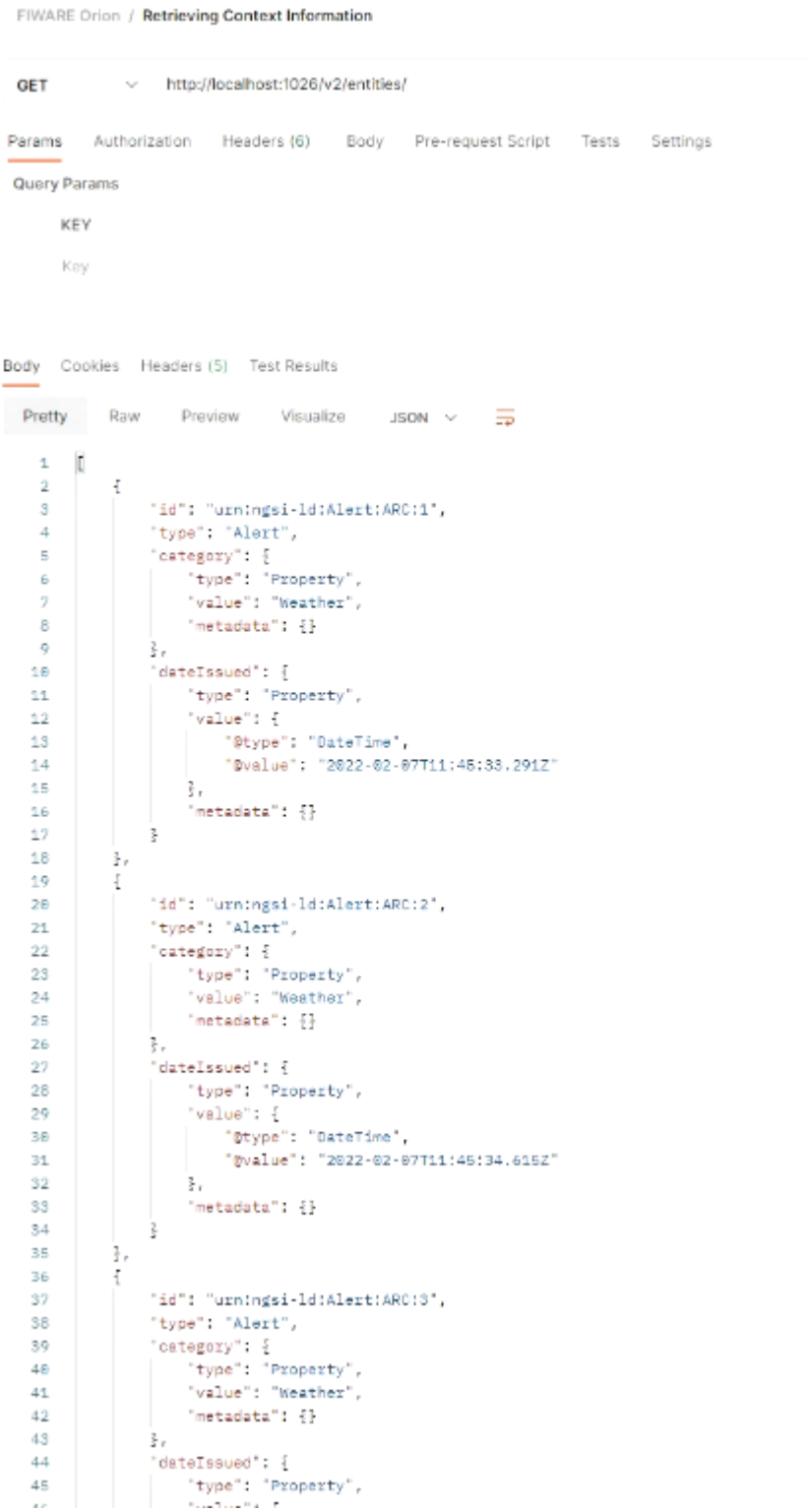


***Figure 29.*** *Context Data in FIWARE Context Broker*

## 5.2 MQTT Adapter

The MQTT adapter output first displays the success message on connecting successfully with the MQTT broker.

```
C:\Users\     k\OneDrive - TUNI.fi\Desktop\DCF\dcf-mqtt>python mqtt-client.py
CONNACK received with code Success.
Subscribed: 1 [<paho.mqtt.reasoncodes.ReasonCodes object at 0x000002029009FBE0>]
```

***Figure 30.*** *MQTT Adapter Output*

The MQTT adapter displays the real-time messages and data of the subscribed topic. Since several different topics can be subscribed simultaneously, the adapter displays the topic name and the message. As an example, the dummy temperature values were posted to the broker and the adapter was fetching, processing, and displaying the temperature values.

The testing of MQTT data adapter was done through HiveMQ MQTT broker.


***Figure 31.*** *Testing with HiveMQ MQTT Broker*

Temperature values were being displayed on MQTT data Adapter.

```
Topic: testtopic/machine1_temp
Message: b'23'
23

Topic: testtopic/machine1_temp
Message: b'25'
25

Topic: testtopic/machine1_temp
Message: b'29'
29
```

***Figure 32.*** *Temperature Values being displayed through MQTT data adapter*

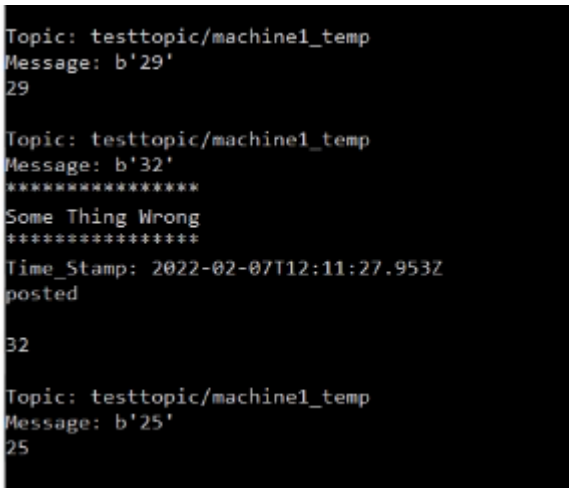


***Figure 33.*** *A Snapshot of event Processing*

Event being stored in the MongoDB database.


***Figure 34.*** *Event Alert stored in MongoDB.*

Context data being published to FIWARE context broker.

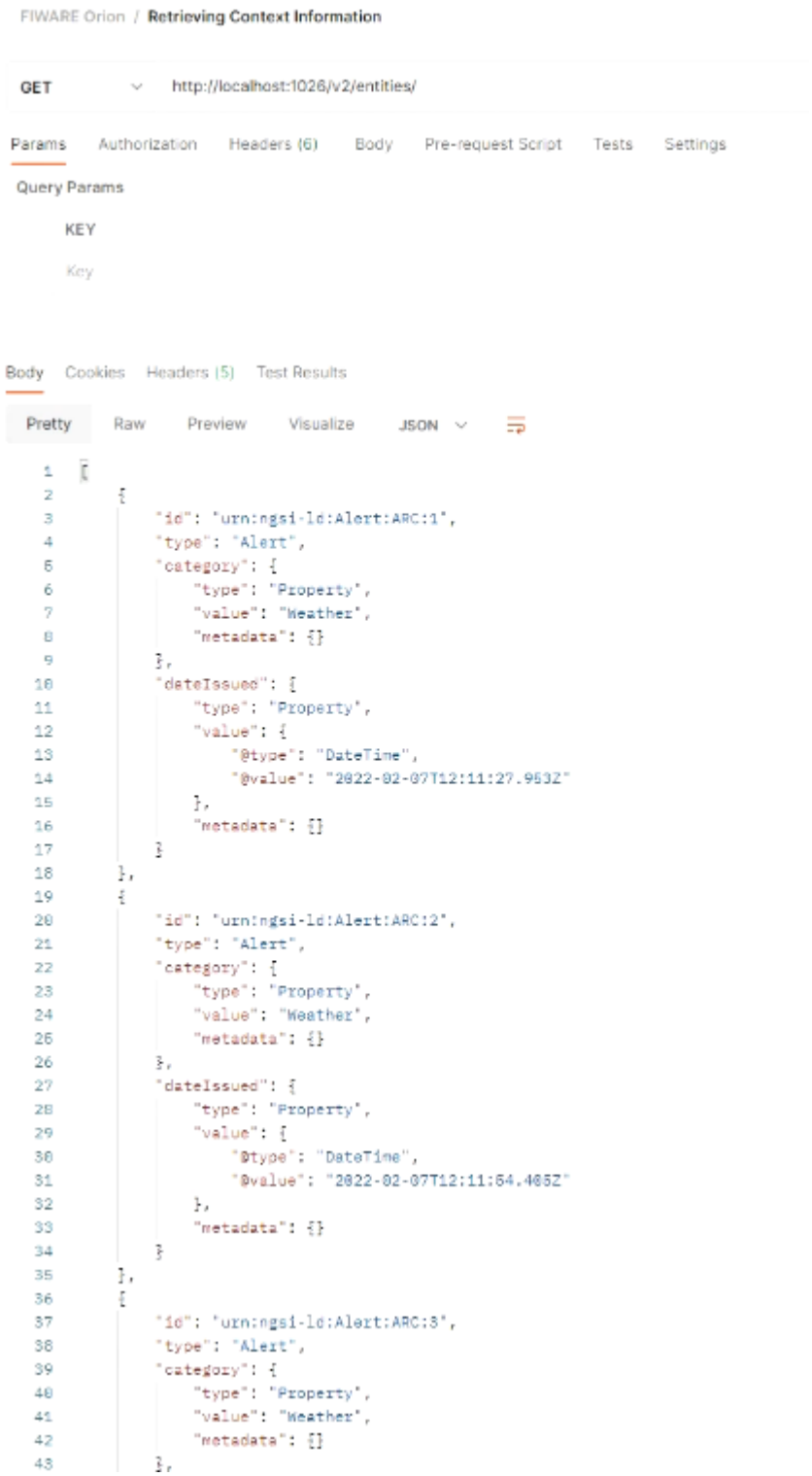


***Figure 35.*** *Retrieving Context Data in FIWARE Context Broker*

# 6. CONCLUSIONS

This thesis has successfully addressed the research questions and presented a design and development of a data collection framework for shop floor data, which facilitates the digitalization of factories. The proposed framework's architecture has been carefully designed to ensure real-time data collection and transmission, while also being scalable and efficient.

The implementation of the proposed framework involved the use of MQTT data adapter and OPC UA adapter to transmit data from shop floor devices to the web-based data storage. The framework's containerization ensured its portability and scalability.

The framework's event processing capability has been utilized to create real-time alerts, ensuring a safe shop floor environment for workers. The alerts generated by the framework are saved and stored and supervisors can view these alerts and prepare for the potential hazards.

The results have shown that the proposed framework is effective in acquiring data from the shop floor and can be integrated with various communication protocols.

Overall, this thesis has contributed to the field of digitalization of factories by providing a comprehensive and practical solution for data collection on the shop floor. The proposed framework's implementation, testing, and results have shown its potential to facilitate the digital transformation of factories, enhance their competitiveness, and improve their performance and efficiency, while also ensuring a safe shop floor environment for workers.

## 6.1 Future Work

Although this thesis has presented a comprehensive and practical solution for data collection on the shop floor, there are still opportunities for future work to further improve the proposed framework's performance and functionality. Some potential areas of future work include:

- Integration with advanced analytics and machine learning techniques: The proposed framework can be integrated with advanced analytics and machine learning techniques to provide real-time insights and predictions. This integration can enable factory managers and engineers to make informed decisions based on real-time data, resulting in improved factory performance and efficiency.

- Improvement of the user interface (UI): While the proposed framework's UI provides essential functionalities, there is still room for improvement. The UI can be designed to be more intuitive and user-friendly, making it easier for supervisors and managers to access and analyse data. The framework can be enhanced with charts and graphs. This enhancement can help identify trends and patterns, highlighting areas for improvement and refinement.
- Deployment of the framework in real-world scenarios: Although the proposed framework has been tested in a simulated environment, its deployment in real-world scenarios can provide valuable insights into its performance and functionality. The deployment of the framework can also highlight potential areas for improvement and refinement.

# REFERENCES


[1] E.M. Martinez, P. Ponce, I. Macias, A. Molina, "Automation Pyramid as Constructor for a Complete Digital Twin, Case Study: A Didactic Manufacturing System." Sensors 2021, 21, 4656. https://doi.org/10.3390/s21144656

[2] C. Verdouw, R. Robbemond, J.W. Kruize, "Integration of Production Control and Enterprise Management Systems in Horticulture.", HAICTA 2015 7th International Conference on Information and Communication Technologies in Agriculture, Food and Environment, Greece, 2015

[3] MQTT Version 3.1.1 Plus Errata 01. Edited by Andrew Banks and Rahul Gupta. 10 December 2015. OASIS Standard Incorporating Approved Errata 01, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[4] MQTT: The Standard for IoT Messaging, https://mqtt.org/

[5] Publish & Subscribe - MQTT Essentials, https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/

[6] MQTT Topics, https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/

[7] W. Mahnke, S. H. Leitner, M. Damm, “Introduction”, in OPC Unified Architecture, 1st ed, Germany: Springer Berlin, Heidelberg, 2009.

[8] W. Mahnke, S. H. Leitner, M. Damm, “Information Modelling: Concepts”, in OPC Unified Architecture, 1st ed, Germany: Springer Berlin, Heidelberg, 2009.

[9] FIWARE Catalogue, https://www.fiware.org/catalogue/

[10] Orion Context Broker, https://github.com/telefonicaid/fiware-orion/

[11] Core Context Management, https://fiwaretourguide.readthedocs.io/en/latest/core/introduction/

[12] FIWARE IoT Agents, https://fiwaretourguide.readthedocs.io/en/latest/iot-agents/introduction/

[13] FIWARE Catalogue, https://github.com/FIWARE/Catalogue/

[14] FIWARE IoT Agent, https://iotagent-node-lib.readthedocs.io/en/latest/architecture.html

[15] SHOP4CF - SHOP4CF, https://shop4cf.eu/

[16] FIWARE Data Models, https://fiware-datamodels.readthedocs.io/en/stable/ngsi-ld_howto/index.html

[17] SHOP4CF Data Models, Documentation of FIWARE data models used in SHOP4CF, https://shop4cf.github.io/data-models/alert.html

AI-based application, ChatGPT, has been used to produce structurally fluent text for the thesis.